→ 5., aktualisierte Auflage

Klaus Schmeh

Kryptografie

Verfahren • Protokolle • Infrastrukturen

iX EDITION

dpunkt.verlag

Klaus Schmeh ist seit 1997 als Unternehmensberater mit Schwerpunkt Kryptografie aktiv. Seit 2004 arbeitet er für die Gelsenkirchener Firma cryptovision. Nebenbei ist Klaus Schmeh ein erfolgreicher Journalist, der 20 Bücher und 150 Zeitschriftenartikel verfasst hat. Etwa die Hälfte seiner Werke beschäftigt sich mit kryptografischen Themen. Klaus Schmeh hat damit mehr zum Thema Kryptografie veröffentlicht als jede andere Person in Deutschland. Seine Stärke ist die anschauliche Vermittlung komplexer Zusammenhänge, die auch in seinen anderen Veröffentlichungen (meist zu populärwissenschaftlichen Themen) zum Tragen kommt.

***iX*-Edition**
In der *iX*-Edition erscheinen Titel, die vom dpunkt.verlag gemeinsam mit der Redaktion der Computerzeitschrift *iX* ausgewählt und konzipiert werden. Inhaltlicher Schwerpunkt dieser Reihe sind Standardthemen aus IT, Administration und Webprogrammierung.

Klaus Schmeh

Kryptografie

Verfahren, Protokolle, Infrastrukturen

5., aktualisierte Auflage

dpunkt.verlag

Klaus Schmeh

klaus.schmeh@dpunkt.de

Lektorat: Dr. Michael Barabas
Copy-Editing: Annette Schwarz, Ditzingen
Satz: Klaus Schmeh
Herstellung: Birgit Bäuerlein
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Buch 978-3-86490-015-0
PDF 978-3-86491-267-2
ePub 978-3-86491-268-9

5., aktualisierte Auflage 2013

Ringstraße 19 B
69115 Heidelberg

5 4 3 2 1 0

Vorwort von Prof. Bernhard Esslinger

»Transparenz. Das ist das Höchste, was man sich in einer technologisch hoch entwickelten Gesellschaft erhoffen kann. … sonst wird man einfach nur manipuliert ...«

Daniel Suarez in »Darknet«

»The best that can be expected is that the degree of security be great enough to delay solutions by the enemy for such a length of time that when the solution is finally reached, the information thus obtained has lost all its value.«

William Friedman in *Military Cryptanalysis*

»Microsofts Ziel sei es nicht, jede potenzielle Hintertür zu schließen, sondern das Geschäftsmodell organisierter Fälscher zu stören, die illegale Vista-Kopien verkaufen.«

Heise Newsticker, 11.04.07

»Immer wenn man etwas konkret formuliert, wird man angreifbar, aber wenn man nicht konkret wird, ist es nicht nachvollziehbar.«

Unbekannt

Als Herr Schmeh mich fragte, ob ich das Vorwort zu seinem Kryptografie-Buch schreibe, war meine erste Reaktion: »Warum ich und warum ein weiteres Buch über Kryptologie?«

Auf beide Fragen hatte Herr Schmeh eine einleuchtende Antwort:

- Ich sollte das Vorwort schreiben, da er jemand suchte, der intensive theoretische, praktische und berufliche Erfahrung auf diesem Gebiet habe und diese Erfahrungen pointiert in das Vorwort einfließen ließe (ich war Entwicklungsleiter der Sicherheitskomponenten des Systems R/3 und CISO bei der SAP AG, Leiter IT-Sicherheit bei der Deutschen Bank AG und bin als Chef des »Crypto-

graphy Competence Center« dort für die angemessene Nutzung von Kryptografie zuständig. Außerdem habe ich einen Lehrauftrag zu IT-Sicherheit und leite seit über zehn Jahren ein Open-Source-Projekt, das das bisher erfolgreichste Lernprogramm zu Kryptologie erstellt).

- Sein Buch habe aufgrund von mehreren Eigenschaften ein Alleinstellungsmerkmal: Aktualität, Umfang/Vollständigkeit, Betonung der Anwendungssicht (nicht der Mathematik), Behandlung auch der umliegenden Felder (Geschichte, Gesellschaft, Politik, Wirtschaftsspionage, …) und – aufgrund seiner journalistischen Erfahrung – die gewohnt leicht verständliche Beschreibung auch komplexer Zusammenhänge.

Ich schreibe dieses Vorwort als Privatperson.

Kryptografie ist eine in mehrfacher Hinsicht spannende Angelegenheit:

- für Historiker, weil sie schon immer Teil des strategischen und taktischen Arsenals der Mächtigen war,
- für Mathematiker und Informatiker, weil sich in der Zahlentheorie und der mathematischen Kryptologie ständig neue Forschungsergebnisse ergeben (z.B. die Möglichkeiten für die Cloud durch homomorphe Verschlüsselung, die 2008 von Dinur/Shamir veröffentlichte neue Kryptoanalysemethode Cube Attack, generische Analysemethoden wie SAT-Solver, das Knacken des Standardverschlüsselungsverfahrens in WLAN-Netzen innerhalb von weniger als einer Minute durch die Uni Darmstadt 2007, das Zerlegen eines gegebenen 232-stelligen Produktes in seine beiden Primzahl-Faktoren durch Kleinjung etc. im Jahre 2009 usw.). Und das zukünftige Quanten-Computing sorgt dafür, dass weiter intensiv an neuen Verfahren geforscht wird;
- für Praktiker und Sicherheitsverantwortliche, weil es stets neue Entwicklungen gibt: Auf der Angreiferseite werden etablierte Protokolle, die man für sicher hielt, weil sie starke kryptografische Primitive verwenden, kreativ missbraucht oder mit Man-in-the-Middle-Attacken umgangen (SSL allein im Browser wird den stärker werdenden Angriffen nicht gerecht). Auf der Seite »der Guten« kommen neue Techniken zum Einsatz: Smartcard-Chips auf USB-Tokens, Nutzen von TPMs nicht nur in Laptops, sondern auch auf Servern, Nutzen von virtualisierbarer Hardware, Open-Source-Lösungen wie OpenXPKI, die nicht nur die Grundfunktionalität einer PKI bieten, sondern praxisnahe Schnittstellen enthalten, z.B. um Smartcards zu personalisieren, Kartenausgabesysteme anzubinden, an Zertifikats-Renewal-Software (CertNanny) anzudocken und Tracking-Systeme zu integrieren, sowie
- für IT-Manager, weil sich hier ganz praktisch die Fragen nach dem richtigen Umgang mit dem Risikomanagement stellen, nach den angemessenen Maßnahmen im Umgang mit Risiken, nach der Balance zwischen technischen und organisatorischen Maßnahmen (Anweisungen, Schulungen), nach der erlangten Sicherheit, die sich aus der Wahl der richtigen Algorithmen/Protokolle, korrekter Implementierung und der Benutzerfreundlichkeit ergibt.

Unternehmen investieren nicht einfach in IT-Sicherheit, sondern es werden Risikobetrachtungen angestellt, und es wird versucht, das optimale Maßnahmenbündel zur Verringerung/Vermeidung (Mitigation) des Risikos zu finden. Dabei kann Kryptografie die richtige Maßnahme sein, sie ist es aber nicht immer. Sie ist es vor allem dann, wenn sie mit Sachverstand eingesetzt wird. Manchmal sind organisatorische Maßnahmen billiger, manchmal wirken Mitarbeiterschulungen nachhaltiger. Immer kommt es auf den richtigen Mix an. Unter den technischen Maßnahmen wirkt Kryptografie proaktiv – im Gegensatz zu reaktiven Maßnahmen wie Monitoring.

Investitionen ergeben sich nicht nur aus langfristig geplanten Überlegungen, sondern vermehrt auch wenn Aufsichtsbehörden, Kreditgeber oder Börsen Auflagen erteilten (z.B. »Two-Factor Authentication« der FFIEC, Schlüsselaufbewahrung in HSMs als Forderung der MAS, Basel-2, Compliance-Forderungen, SOx).

Im Gegensatz zur Lehre an den Hochschulen und zur Arbeit der mathematisch orientierten Forscher stellen sich den Anwendern primär die Fragen zu den Kosten der Umsetzung (einmalige Kosten für Entwicklung und Roll-out sowie laufende Kosten), zur Vermeidung von Outages und zur Akzeptanz bei den Benutzern.

Dabei ergeben sich im Umfeld der Kryptografie die sonst auch in der IT üblichen Erscheinungen:

- Gartner-Hype-Kurven, die z.B. von PKI zuerst die Lösung aller Sicherheitsprobleme erwarteten, dann PKI »verdammten«, und nun ist PKI doch fast überall im Einsatz (Online-Banking, Webauthentisierung, SOA, Flaschenpfandsystem)
- »Angesagte« Produkte bieten für ein bestimmtes Problem eine Lösung an, aber gleichzeitig schafft ihr Einsatz andere, neue Probleme (z. B. neue, mathematisch sehr spannende Verfahren mit schönen Namen, die von Firmen mit Venture Capital vermarktet werden. Dabei ist dann die Anzahl der Mitarbeiter in den Vertriebs-, Marketing- und Rechtsabteilungen um ein Vielfaches höher als die Anzahl der kryptografischen Kompetenzträger oder der eigentlichen Softwareentwickler). Ebenso zu hinterfragen sind angesagte Begriffe wie BYOD, bei denen noch ein ganzes Bündel an Fragen ungeklärt ist: Hierbei sollten Firmen ihren Mitarbeitern eher erstklassige Smartphones (auch zur Privatbenutzung in einem abgetrennten Bereich) ausgeben, als jeden Handytyp der Mitarbeiter zuzulassen. Interessen von Herstellern und Netzwerk-Providern zielen aber eher auf den privaten Besitz ab, da dort im Gegensatz zu den Firmen keine besonderen Firmenkonditionen zu gewähren sind.
- kleine Gruppen im Management, die ihre speziellen Gadgets wollen und die sie sich auch genehmigen können, obwohl die Sicherheitsarchitektur und die Interoperabilität dafür nicht ausreichend gegeben sind (was z.B. dazu führt, dass gerade wichtige E-Mails im Klartext versandt werden)

- Eine Konzentration der Anbieterfirmen und ein Marktverhalten einzelner großer IT-Security-Anbieter, das darauf abzielt, die Kunden abhängig zu machen. Nicht offengelegte Schnittstellen werden als Sicherheitsmerkmal verkauft (Security by Obscurity oder verborgene Hintertüren?). Die nächste Hardwaregeneration gibt es umsonst, dafür sind die Updates umso teurer. Bisherige BUS-/SOAP-/RPC-Systeme werden zu SOA-Lösungen erklärt. Manche Hersteller integrieren die WS-Security-Funktionen nicht nur nachträglich, sondern leisten sich dabei auch typische Security-Anfängerfehler. Die seit 2004 gebrochene MD5-Hashfunktion wird von »professionellen Trustcentern« noch immer zum Signieren genommen, selbst wenn der Kunde nicht darauf besteht und obwohl es erfahrungsgemäß nur eine Frage der Zeit ist, bis eine Schwäche ausgenutzt wird (siehe die die Kollision nutzende »Rogue CA«, die im Dezember 2008 publiziert wurde). Manche Produkte hatten den symmetrischen Login-Key im Client-Executable abgelegt ...
- Arbeitsgruppen über Layout und Businessmodelle mit endlosen Diskussionen, weil da jeder mitreden zu können meint – im Gegensatz zu sehr erfolgreichen technischen Arbeitsgruppen
- Technisch überlegene Standards »vergessen« den Benutzer, der nicht weiß, wie er ein Zertifikat in den Mail-Client oder den Browser auf seinem Privatrechner bringen soll.
- Diskussionen um rechtliche Erfordernisse, die von sehr wenigen Dogmatikern beherrscht werden, die Einfluss auf die Politik und den Gesetzgeber nehmen (z.B. im deutschen SigG/SigV) und die selbst dann an ihren teuren, theoretisierenden Empfehlungen festhalten, wenn fast keiner diese nutzt und wenn sie unserer internationalen Wettbewerbsfähigkeit im Wege stehen. Man braucht sich nicht zu wundern, wenn die Standards in den verbreiteten Produkten dann von einzelnen, schnellen Herstellern erstellt und in internationale Normungsgremien (IETF, IEEE, PKCS) eingebracht werden, die kein Verständnis für inkompatible nationale Sonderwege haben. Ebensowenig braucht man sich dann zu wundern, dass innerhalb von pragmatisch agierenden Zusammenschlüssen (wie der EB-CA oder den virtuellen Behörden-Poststellen) die Sicherheit real deutlich erhöht wird mithilfe von fortgeschrittenen Zertifikaten (die zigmillionenfach im Einsatz sind), während die akkreditiert qualifizierten »Sonderlocken« noch nicht einmal die 100.000 erreichten. Mit solchen über die EU-Direktive hinausgehenden akkreditiert qualifizierten Signaturen (die zudem bei der Validierung das inkompatible »Kettenmodell« verlangen) erschwert man die Verbreitung der digitalen Signatur beträchtlich. Hier zeigt sich, dass man sehr genau spezifizieren sollte, für welche Fälle man Anforderungen aufstellt: So wenig, wie man für die allermeisten der im Alltag geschlossenen Verträge einen Notar braucht, so selten muss man bei elektronischen Verträgen vom Spezialfall des Anscheinsbeweises im Prozessfall ausgehen.

- Leider gibt es immer wieder Partnerfirmen, die sichere E-Mail-Kommunikation nicht oder nur unvollständig unterstützen – besonders auffällig ist das bei manchen Anwaltskanzleien, die hochsensible Daten ihrer Kunden unverschlüsselt versenden oder verschlüsselt erhaltene Mail entschlüsseln und dann unverschlüsselt an die Blackberrys ihrer Mitarbeiter weiterleiten.
- Und natürlich hat alles mehrere Seiten, so dass sich sicher auch Kritiker (und berechtigte Einwände) an dieser Kritik finden …

In der Realität hat Kryptografie inzwischen fast überall Einzug gehalten: von Pay-TV über Auto-Wegfahrsperre, Handy bis in jeden Webbrowser. Fast alle großen Unternehmen betreiben eigene PKIs, mit denen ihre Mitarbeiter sichere E-Mails versenden, sich sicher in WLANs anmelden oder sicher Dateien auf outgesourcten Servern verschlüsseln können. Softwarehersteller wie SAP statteten ihre Software mit generischen Schnittstellen wie der GSS-API aus, damit die Kunden wahlweise die Sicherheitsfunktionen von PKI- oder von Kerberos-basierten Systemen nutzen können.

Kryptografie erwies sich dann als sicher und erfolgreich im Einsatz in Firmen und im Internet, wenn sie

- hohe Interoperabilität gewährleistete (keine Insellösungen),
- für die Benutzer (nahezu) transparent war (kein oder kaum Mehraufwand) und
- ausgereift war.

Wurde Expertenwissen genutzt, konnte viel Geld gespart oder konnten leicht Fehler vermieden werden, die im Nachhinein aufwendig zu beheben sind: Das Management von Schlüsseln für Maschinen, Dienste, Personen und Infrastrukturen muss verstanden und geplant werden. [Beispielsweise haben CAs mit Modullängen von 512 Bit keinen Sinn. Hier hat Microsoft im August 2012 mit seinem Security Advisory 2661254 gute Dienste geleistet (das entsprechende Windows-Update verhindert die Verwendung von Zertifikaten mit RSA-Schlüsseln von weniger als 1024 Bit Länge durch die MS-Krypto-API). Ein anderes Beispiel: Der Lebenszyklus von Schlüsseln muss Zertifikats-Renewal und Schlüsselverlust von vornherein berücksichtigen.] Insbesondere kleineren Firmen, die kostenlose PKI-Software von Microsoft oder aus dem Open-Source-Bereich oder Managed PKIs von Trustcentern wie SignTrust etc. nutzen (und damit die rein technische Seite abdecken), ist hier zu raten, Experten-Know-how kurzfristig in der Architekturphase einzukaufen.

Das erforderliche Expertenwissen ist inzwischen breiter vorhanden, da viele Lehrstühle IT-Sicherheitsexperten ausgebildet haben. Sowohl für diese neuen Kollegen als auch für alle, die Fragen zu diesem Thema haben, vermittelt das Buch von Klaus Schmeh einen hervorragenden Überblick. Es geht so weit in die Tiefe, dass man die Verfahren verstehen und einordnen kann und dass man Produkt- und Protokollentscheidungen treffen kann. Im ausführlichen Literaturver-

zeichnis lassen sich dann alle Originalpapiere, die auch die genaue Mathematik enthalten, problemlos finden. Zusätzlich können Sie spielerisch einzelne Verfahren mit der im Buch erwähnten freien Lernsoftware CrypTool ausprobieren.

Und zu Recht wird in diesem Buch unter den »wichtigsten weiterführenden Büchern« Ross Anderson mit *Security Engineering* aufgeführt, denn die praktischen Probleme sind oft verschieden von denen, über die theoretisch orientierte Experten gerne diskutieren (z.B. das Ausnutzen von Padding-Fehlern statt Angriffe mit differenzieller Kryptoanalyse, das Eindringen über Passwortraten und auf Anwendungsebene, das Nutzen der menschlichen Psychologie und immer stärker auch die Ökonomie der IT-Sicherheit und der Malware-Industrie). Beide Sichtweisen sind wichtig.

Gefährlich für die kritischen Infrastrukturen sind nicht einzelne Hacker, sondern rational handelnde und oft mafiamäßig/militärisch organisierte Angreifer, die auf Gewinn aus sind und zuerst meist den leichtesten/kostengünstigsten Weg für ihre modernen Raubzüge wählen. Das »Spiel« zwischen Cyber-Kriminellen und »guten« Cyber-Nutzern und ihren Verbündeten wird nicht enden. Dabei wird die Benutzerinteraktion (ermöglicht vom Softwareentwickler, aber getragen vom Verständnis und Mitwirken des Nutzers) immer zentral und schwierig bleiben.

Auch diese Themen im Umfeld der Kryptografie werden in diesem Buch aktuell behandelt. Für die Zielgruppe der Nichtmathematiker ist es daher weiterhin *das* Kryptografie-Standardwerk im deutschsprachigen Raum.

Ich wünsche auch der 5. Auflage alles Gute.

Bernhard Esslinger
November 2012

Inhaltsübersicht

Inhaltsverzeichnis

Teil 1
Wozu Kryptografie?

Teil 4
Public-Key-Infrastrukturen

Anhang

Teil 1

Wozu Kryptografie?

1 Einleitung

Verschlüsselung kann über Leben und Tod entscheiden. Dies zeigte sich nie dramatischer als im Zweiten Weltkrieg, als britische Spezialisten die deutsche Verschlüsselungsmaschine Enigma knackten. Dieser Erfolg war der Regierung in London so wichtig, dass sie vor den Toren der Hauptstadt eigens eine Dechiffrier-Fabrik aus dem Boden stampfte, in der Tausende von Menschen mit speziellen Maschinen an der Entzifferung von Enigma-Funksprüchen arbeiteten. Die Folgen der industriell organisierten Dechiffrierung bekamen nicht zuletzt deutsche U-Boot-Besatzungen im Nordatlantik zu spüren. Deren verschlüsselte Positionsmeldungen konnten die Briten oftmals entziffern, um sie anschließend für tödliche Angriffe zu nutzen.

Während sich die britischen Codeknacker an der Enigma versuchten, dechiffrierten die anderen kaum weniger erfolgreich ein weiteres deutsches Verschlüsselungsgerät: die Lorenz-Maschine. Gleichzeitig gelang es in Schweden einem genialen Mathematiker, auch die dritte bedeutende deutsche Chiffriermaschine –

den Geheimschreiber T-52 – zu knacken. Doch die Deutschen waren derweil nicht untätig. Sie konnten nicht nur routinemäßig die US-Verschlüsselungsmaschine M-209 dechiffrieren, sondern knackten auch zahlreiche andere Geheimcodes ihrer Kriegsgegner. Wie viele Soldaten im Zweiten Weltkrieg aufgrund dechiffrierter Funksprüche ihr Leben verloren und welchen Ausgang der Zweite Weltkrieg ohne die Arbeit der Codeknacker genommen hätte, wird sich wohl nie klären lassen. Eines ist jedoch sicher: Die Behauptung, Verschlüsselung könne über Leben und Tod entscheiden, ist nicht übertrieben.

1.1 Kryptografie heute

Knapp 70 Jahre nach der Blütezeit der Enigma ist das Thema Verschlüsselung (Kryptografie) aktueller denn je. Der wichtigste Grund dafür ist das Internet, das Mitte der neunziger Jahre den Sprung zum Massenmedium vollzog, nachdem es zuvor kaum mehr als ein Spielzeug für Wissenschaftler gewesen war. Im Fahrwasser des Internets, dessen Sicherheitsmängel schnell offenkundig wurden, setzte die Kryptografie zur größten Blüte ihrer langen Geschichte an. Hersteller von Verschlüsselungslösungen schossen wie Pilze aus dem Boden. Kryptografie wurde zum Hype-Thema, auch an der Börse.

Heute, etwa 15 Jahre nach dem Durchbruch des Internets, ist der Kryptografie-Hype zwar zu Ende, doch die Bedeutung des Themas ist deshalb nicht gesunken. Inzwischen unterstützen alltägliche Anwendungen wie Smartphones, Webbrowser oder E-Mail-Programme kryptografische Funktionen. Dazu gibt es zahlreiche Speziallösungen für unterschiedlichste Zwecke. Protokolle wie HTTP oder IP wurden mit Krypto-Funktionen nachgerüstet, deren Einsatz weit verbreitet ist. In neuere Technologien wie WLAN, Bluetooth oder UMTS wurde die Verschlüsselung von Anfang an fest eingebaut.

Nicht zu übersehen ist ein Trend zum Pragmatismus in der Kryptografie. Die Hersteller von Verschlüsselungslösungen mussten feststellen, dass ihre Kunden die Sicherheit nicht über alles stellen, sondern stets einen Kompromiss zwischen Sicherheit, Kosten und Praxistauglichkeit suchen. Diese Entwicklung hat dazu geführt, dass sich komplexe Kryptografie-Anwendungen wie S-HTTP, SET oder Ecash trotz eines hohen Ansehens in der Krypto-Szene nicht am Markt durchsetzen konnten. Dafür feierten einige Kryptografie-Anwendungen Erfolge, die man in den Neunzigern noch als minderwertig bezeichnet hätte. Bridge-CAs, E-Mail-Verschlüsselungs-Gateways und Roaming Keys sind Beispiele dafür. Die praktischen Vorteile der genannten Lösungen haben dafür gesorgt, dass viele Anwender Abstriche bei der Sicherheit in Kauf nahmen.

Durch diese Entwicklungen haben sich auch die Anforderungen an ein Kryptografie-Buch in den letzten 15 Jahren erheblich geändert. Interessierten sich die Leser anfangs noch vor allem für die kryptografischen Verfahren an sich, so spielt heute deren Umsetzung in die Praxis eine mindestens ebenso wichtige Rolle. Die-

ses Buch trägt diesem Umstand Rechnung. Es gibt noch einen weiteren Grund, warum ein Kryptografie-Buch heute anders aufgebaut sein muss als noch in den neunziger Jahren: Die Kryptografie ist inzwischen in vielerlei Hinsicht zum Allgemeingut geworden. Verfahren und Protokolle, die man früher in Büchern nachschlagen musste, werden heute in Wikipedia-Artikeln und auf anderen Webseiten zu Dutzenden beschrieben. Ein Kryptologie-Buch muss diesen Quellen gegenüber einen Mehrwert bieten. Es muss ein Leitfaden für den Einsteiger sein, es muss Zusammenhänge aufzeigen, es muss Bewertungen abgeben, und es muss neue Entwicklungen berücksichtigen. Und dennoch muss ein Kryptografie-Buch auch als Nachschlagewerk für denjenigen funktionieren, der nur schnell etwas wissen will. Ich habe mein Bestes gegeben, um alle diese Anforderungen in diesem Buch zu erfüllen.

1.2 Die fünfte Ausgabe

Das Buch, das Sie in den Händen halten, ist bereits die fünfte Ausgabe. Dies sind die Vorgänger:

- *Safer Net – Kryptografie im Internet und Intranet* erschien 1998 [Schm98]. Der Internet- und Kryptografie-Boom, der damals in vollem Gange war, spiegelt sich unübersehbar darin wider. Obwohl ich es nicht unbedingt so geplant hatte, kam »Safer Net« vor allem bei Einsteigern sehr gut an. Der anschauliche Schreibstil und mein Bemühen, die mathematische Seite der Kryptografie auf ein notwendiges Minimum zu reduzieren, trugen dazu bei.
- *Kryptografie und Public-Key-Infrastrukturen im Internet* aus dem Jahr 2001 ist eine Neubearbeitung des Erstlings [Schm01]. Auch dieses Buch war ein Kind seiner Zeit: Die ersten Abnutzungserscheinungen des Internet- und Kryptografie-Booms waren 2001 bereits erkennbar; erste Enttäuschungen waren Realität geworden. Der Pragmatismus hatte die Kryptografie erfasst. Dies änderte allerdings wenig an der Wichtigkeit des Themas, weshalb sich das Buch erneut bestens verkaufte. Da ich nach wie vor einen nicht allzu mathematischen Zugang vermittelte, fanden sich unter den Lesern wieder viele Einsteiger.
- *Kryptografie – Verfahren, Protokolle, Infrastrukturen* stammt aus dem Jahr 2007 und war die dritte Ausgabe dieses Buchs [Schm07/2]. Die Kryptografie hatte inzwischen ihre Entwicklung in Richtung Pragmatismus fortgesetzt und war – genauso wie das Internet – zu einem alltäglichen Werkzeug geworden. Vor allem aber war die Anzahl der kryptografischen Verfahren, Protokolle und Standards weiterhin deutlich angestiegen. Um möglichst viele der neuen Entwicklungen in meinem Buch unterzubringen, war es mir wichtig, dessen Umfang im Vergleich zum Vorgänger deutlich zu erweitern. Damit wollte ich zudem eine Marktlücke schließen, die sich ergeben hatte, weil es keinen adäquaten Nachfolger für das seit 1995 nicht mehr aktualisierte Meisterwerk

Angewandte Kryptographie von Bruce Schneier gab [Schn96]. Aus heutiger Sicht denke ich, dass ich das hochgesteckte Ziel erreicht habe. Mit einem Umfang von über 800 Seiten bot mein Buch in der dritten Ausgabe mehr Inhalt als das von Schneier und wurde außerdem zum meines Wissens umfangreichsten Werk seiner Art weltweit.

- *Kryptografie – Verfahren, Protokolle, Infrastrukturen* erschien 2009 und bildete die vierte Ausgabe [Schm09/1]. Erstmals hatte sich der Titel im Vergleich zur vorherigen Version nicht geändert. Auch die Philosophie des Buchs blieb die gleiche. Dies lag an der kryptografischen Großwetterlage, die längst nicht mehr so stürmisch war wie in den Anfangsjahren des Internets. Mit einem Plus von 50 Seiten gegenüber der Vorversion wurde dieses Buch endgültig die umfangreichste Sammlung von Wissen zum Thema Kryptografie, die es als Buch zu kaufen gibt. Die von Bruce Schneier hinterlassene Lücke gehörte damit der Vergangenheit an.

Im Frühjahr 2012 nahm ich die fünfte Ausgabe in Angriff, die Sie nun in den Händen halten. Erneut war es nicht notwendig, grundsätzliche Dinge am Buch zu ändern. Dafür gab es viel Neues aufzunehmen und Bestehendes zu aktualisieren. Aus Kostengründen wollte der Verlag den Umfang nicht weiter ausdehnen, obwohl es genug Stoff gegeben hätte. Ich strich daher einige Inhalte – vor allem Dinge aus den neunziger Jahren, die inzwischen ihre Bedeutung verloren haben – und kürzte an einigen Stellen. Nach wie vor ist dieses Buch die größte zusammenhängende Sammlung von kryptografischem Fachwissen zwischen zwei Buchdeckeln.

1.3 Mein Bedauern, meine Bitten und mein Dank

Es ist unnötig zu erwähnen, dass sich trotz aller Mühe Fehler, Ungenauigkeiten und sonstige Mängel in mein Buch eingeschlichen haben. Dies sollte für Sie ein Ansporn sein: Teilen Sie mir Fehler, Anregungen und Kritik mit. Da es sicherlich irgendwann eine sechste Ausgabe geben wird, werden derartige Mitteilungen nicht im Papierkorb landen. Senden Sie Ihre Anregungen bitte an klaus.schmeh@dpunkt.de. Auch eine Errata-Liste können Sie auf diese Weise anfordern. Es ist vermutlich noch viel unnötiger zu erwähnen, dass zum Gelingen eines solchen Buchs neben dem Autor noch viele andere beigetragen haben. Bei all denjenigen möchte ich mich an dieser Stelle recht herzlich bedanken, auch wenn ich nicht alle namentlich nennen kann. Insbesondere gilt mein Dank folgenden Personen:

- Dr. Michael Barabas vom dpunkt.verlag für seine Unterstützung bei der Realisierung dieses Buchs
- Prof. Bernhard Esslinger für seine ausführlichen Anmerkungen und das Vorwort

Außerdem sei folgenden Personen für ihre Unterstützung gedankt:

Dennis Andrick, Sacha Bán, Jacques Basmaji, Dr. Rainer Baumgart, Birgit Bäuerlein, Guido Bertoni, Christian Blumberg, Walter Borenich, Marco Breitenstein, Dr. Jean-Christophe Curtillet, Dr. Frank Damm, Joan Daemen, Ralf Defort, Bernd Degel, Martin Deißler, Karsten Dewitz, Hans Peter Dittler, Benjamin Drisch, Peter Ehrmann, Carl Ellison, Lutz Feldhege, Oliver Ferreau, Helge Fischer, Daniela Freitag, Elmar Frey, Dr. Silvan Frik, Carsten Gäbler, Thomas Garnatz, Thomas Gawlick, Susanne Gehlen, Stefan Haferbecker, Klaus Hesse, Dirk Heuzeroth, Andreas Hoffmeister, Markus Hoffmeister, Frank Hoherz, Dr. Detlef Hühnlein, Katrin Idemudia, Robert Joop, Markus Jünemann, Robert Jung, David Kahn, Dr. Wolfgang Kahnert, Ralf Kirchhoff, Paul Knab-Rieger, Andreas Knöpfle, Kai-Uwe Konrad, Stephanie Kopf, Andreas Krügel, Steffen Leudolph, Thomas Mai, Dr. Martin Manninger, Stefan Milner, Michael Naujoks, Mathias Neuhaus, Matthias Niesing, Peter Pahl, Wolfgang Patzewitz, Dr. Sachar Paulus, Michaël Peeters, Gunnar Porada, Alberto Pricoli, Lars Prins, Holger Reif, Prof. Dr. Helmut Reimer, Stefan Reuter, Guido Ringel, Thomas Rolf, Prof. Dr. Haio Röckle, Prof. Dr. Christoph Ruland, Matthias Sakowski, Tahar Schaa, Patrick Schäfer, Christoph Schlenker, Volker Schmeh, Bernd Schröder, Annette Schwarz, Christian Schwaiger, Mark Sebastian, Uwe Skrzypczak, Dr. Michael Sobirey, Jochen Stein, Moritz Stocker, Malte Sussdorf, Dr. Uwe Tafelmeier, Dr. Hubert Uebelacker, Boris Ulrich, Gilles Van Assche, Dr. Carsten Vogt, Gerald Volkmann, Prof. Dr. Arno Wacker, Rüdiger Weis, Dominik Witte, Vanessa Wittmer, Reinhard Wobst, Oliver Wolf, Zeljko Zelic, Dr. Volker Zeuner, Ursula Zimpfer.

2 Was ist Kryptografie und warum ist sie so wichtig?

2.1 The Name of the Game

Die Frage, was Kryptografie ist, hat zwei Antworten: eine kurze und eine lange.

2.1.1 Die kurze Antwort

Kryptografie ist die Lehre der Verschlüsselung von Daten.

2.1.2 Die lange Antwort

Die Kryptografie ist eine Wissenschaft, die sich mit Methoden beschäftigt, die durch Verschlüsselung und verwandte Verfahren Daten vor unbefugtem Zugriff schützen sollen. Das eine wichtige Hilfsmittel der Kryptografie ist die Mathematik, denn nur durch eine mathematische Denkweise und mithilfe von mathema-

tischen Kenntnissen ist es möglich, Verfahren zur sicheren Verschlüsselung zu entwickeln. Das andere wichtige Hilfsmittel ist der Computer. Dieser führt die Verschlüsselungsverfahren aus und leistet wichtige Dienste bei der Untersuchung von kryptografischen Methoden auf Schwachstellen.

Das Alice-Bob-Mallory-Modell

In der Kryptografie allgemein und speziell in diesem Buch geht man meist von folgendem Modell aus: Zwei Personen (nennen wir sie **Alice** und **Bob**) tauschen über einen abhörbaren Kanal Daten aus. In diesem Buch ist dieser Kanal in der Regel das Internet, manchmal kann es jedoch auch eine Telefonleitung, eine drahtlose Funkverbindung oder der Transport einer CD-ROM in der Hosentasche sein. Entscheidend ist nur, dass es technisch möglich ist, die übertragenen Daten abzuhören, sei es nun durch Anzapfen der Telefonleitung oder Klauen der Diskette. »Abhören« ist in diesem Buch immer im allgemeinen Sinne zu verstehen und kann somit manchmal auch »mitlesen« oder »analysieren« bedeuten.

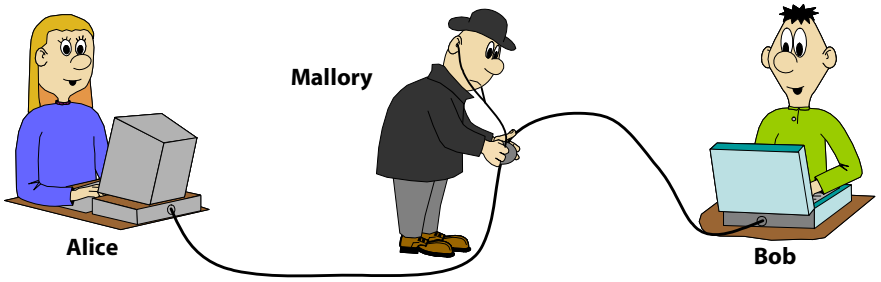


Abb. 2–1 *Alice und Bob tauschen Nachrichten aus, die Mallory mitlesen und manipulieren kann. Dieses einfache Modell liegt dem Buch zugrunde.*

Um das Modell weiterhin einfach zu halten, gehen wir von der Worst-Case-Vermutung aus, dass ein Bösewicht (nennen wir ihn **Mallory**) den Übertragungskanal nach Belieben abhören und die Übertragung beeinflussen kann. In unserem Fall kann Mallory also alle Daten, die Alice und Bob über das Internet austauschen, in beliebiger Weise abfangen, manipulieren und anschließend weiterleiten. Nimmt man dieses Modell als Basis, dann kann die Kryptografie durch Verschlüsselung und ähnliche Maßnahmen verhindern, dass

- Mallory mit den abgefangenen Daten etwas anfangen kann,
- Mallory übertragene Daten unbemerkt verändert,
- Mallory sich unbemerkt Alice gegenüber als Bob ausgibt (bzw. umgekehrt),
- Alice unerkannt behaupten kann, dass eine von ihr gesendete Nachricht in Wirklichkeit eine Fälschung von Mallory sei.

Allerdings kann Mallory auch Dinge tun, die man mit den Mitteln der Kryptografie nicht verhindern kann. Dazu gehört:

- Mallory kann Nachrichten verändern (er kann es nur nicht unbemerkt tun).
- Mallory kann Daten abfangen (er hat nur nichts davon, wenn die Nachrichten sachgemäß verschlüsselt sind).
- Mallory kann die Leitung kappen, einen Router zum Absturz bringen oder ein Rechenzentrum in die Luft sprengen.

Wie Sie sich denken können, sind die Voraussetzungen für dieses Modell nicht immer realistisch. Dass jemand die gleichen Möglichkeiten hat wie Mallory, ist jedoch gerade in Computernetzen weitaus häufiger der Fall, als Sie vielleicht vermuten. Vor allem ist es meist sehr schwer, die Abhörgefahr richtig einzuschätzen. Daher hat es sich bewährt, Mallory von vornherein als sehr gefährlichen Gegner einzuschätzen. Falls Ihnen diese Annahme trotzdem übertrieben erscheint, sollten Sie Kapitel 3 abwarten. Dort werde ich die Abhör- und Manipulationsgefahr im Internet näher unter die Lupe nehmen und zeigen, dass sie tatsächlich ein gewaltiges Problem ist. Deshalb halten wir uns von Anfang an an Murphys erstes Gesetz der Kryptografie: Mallory ist immer gefährlicher, als man denkt (siehe auch Abschnitt 40.6).

Einige Fachbegriffe

Das Wort »Kryptografie« kommt aus dem Griechischen, wo *kryptein* »verstecken« und *gráphein* »schreiben« bedeutet. Neben der Kryptografie gibt es noch die **Kryptoanalyse**. Diese beschäftigt sich nicht mit der Verschlüsselung, sondern mit der unbefugten Entschlüsselung von bereits verschlüsselten Daten. Kryptografie und Kryptoanalyse fasst man in dem Begriff **Kryptologie** zusammen, der also der allgemeinste dieser Fachbegriffe ist. Da die Kryptografie ohne Kryptoanalyse wertlos ist, macht man zwischen Kryptologie und Kryptografie meist keinen Unterschied. Auch in diesem Buch schließt die Kryptografie die Kryptoanalyse mit ein, und ich spreche somit nicht von Kryptologie, sondern von Kryptografie.

2.2 Die Kryptografie – ein wichtiges Teilgebiet

Die Kryptografie gehört in ein Teilgebiet der Informatik, das **Computersicherheit** oder **IT-Sicherheit** genannt wird (in diesem Zusammenhang oft nur als **Sicherheit** bezeichnet). Die Kryptografie ist damit reich an Verwandtschaft, denn die Computersicherheit ist ein sehr großer Bereich, der viele unterschiedliche Teilgebiete umfasst. Etwas unübersichtlich wird das Ganze nicht zuletzt deshalb, weil es im Deutschen nur ein Wort für Sicherheit gibt, im Englischen dagegen zwei, die aber nicht dasselbe bedeuten: Safety und Security. Deshalb ist es auch sinnvoll, den Bereich der Computersicherheit in die Bereiche (**Computer-**)**Security** und (**Computer-**)**Safety** aufzuteilen, und das mit folgenden Bedeutungen:

- In der Computer-Safety geht es um den Schutz vor normalerweise unbeabsichtigten Schäden. Dazu gehören defekte Geräte, unbeabsichtigtes Löschen, Übertragungsfehler, Festplatten-Crashes, Blitzeinschläge, Überschwemmungen, falsche Bedienung, kaputte Disketten und Ähnliches.
- Die Computer-Security dagegen bezeichnet die Sicherheit vor absichtlichen Störungen. Dazu gehören Viren, die Sabotage von Hardware, Hackereinbrüche, das Schnüffeln in geheimen Daten und dergleichen.

Da in diesem Buch kaum von Safety, sondern fast immer nur von Security die Rede ist, werde ich ab jetzt Security mit dem deutschen Wort Sicherheit gleichsetzen. Die Security kann man weiter aufteilen: Es gibt zunächst den Bereich Sicherheit für Computer, die als unabhängiges System betrachtet werden, und außerdem Sicherheit für vernetzte Computer (dieser Bereich heißt auch **Netzwerksicherheit**). Erstere sind naturgemäß längst nicht so gefährdet wie Letztere. Die Kryptografie ist in erster Linie ein Hilfsmittel für die Netzwerksicherheit. Man kann also auch von einem Teilgebiet der Netzwerksicherheit reden.

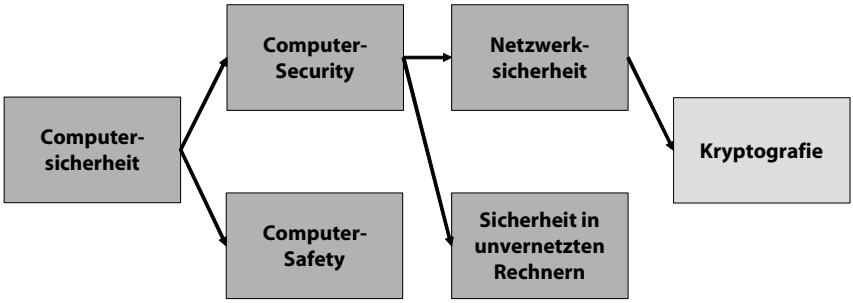


Abb. 2–2 *Die Kryptografie ist ein Teilgebiet eines Teilgebiets eines Teilgebiets der Computersicherheit. Dennoch ist die Kryptografie eine wichtige Wissenschaft.*

2.3 Warum ist die Kryptografie so wichtig?

Jetzt denken Sie vielleicht: Warum muss ich eigentlich ein ganzes Buch lesen, nur um ein Teilgebiet eines Teilgebiets eines Teilgebiets der Computersicherheit zu verstehen, die ja selbst nur ein Teilgebiet der Informatik ist? Die Antwort ist einfach: Es gibt Teilgebiete der Computersicherheit, die nicht wichtig genug sind, um ein Buch darüber zu schreiben (oder zu lesen). Die Kryptografie gehört jedoch nicht dazu. Wie Sie im nächsten Kapitel sehen werden, ist das beschriebene Alice-Bob-Mallory-Modell einfach zu realistisch, um es nur unter »ferner liefen« zu betrachten. Jede E-Mail, jedes Telefongespräch und jede WWW-Nachricht kann in die falschen Hände gelangen und so ungeahnte Folgen nach sich ziehen. Wer alles ein Interesse hat, als Mallory aufzutreten, erfahren Sie in den folgenden Abschnitten.

2.3.1 Wirtschaftsspionage

Der wichtigste Grund, warum die Kryptografie seit Jahren einen Boom erlebt, ist nicht etwa die Sorge um private E-Mails, die Alice an ihren Freund Bob schickt und die Mallory lesen könnte. Verantwortlich sind vielmehr handfeste wirtschaftliche Interessen. Als große Gefahr gelten beispielsweise staatliche Geheimdienste, die sich nach Ende des Kalten Kriegs vermehrt auf die Wirtschaftsspionage verlegen. Nach [Karkow] sind derartige Aktivitäten in Ost und West gleichermaßen zu verzeichnen. Selbst befreundete Staaten wie Frankreich und die USA haben die deutsche Wirtschaft im Visier, und teilweise wird nicht einmal ein Hehl daraus gemacht. Öffentliche Datennetze sind dabei ein immer wichtiger werdendes Betätigungsfeld, da sie oft den einfachen Zugang zu brisanten Informationen ermöglichen.

Als Weltmeister im Lauschen gilt unangefochten die amerikanische Geheimbehörde NSA (National Security Agency). Den Namen dieser Organisation sollten Sie sich merken, denn sie wird noch einige Male in diesem Buch erwähnt werden. Die NSA ist der weltweit größte Arbeitgeber von Mathematikern und ebenfalls weltweit der größte Abnehmer von Hardware. Über die Fähigkeiten dieser Behörde beim Knacken von Codes und im maschinellen Auswerten von abgehörten Informationen werden wahre Wunderdinge erzählt. Genaues weiß man allerdings nicht, da die gesamte Arbeit der NSA unter strengster Geheimhaltung abläuft.

Natürlich kann Wirtschaftsspionage auch ohne Hilfe von Geheimdiensten betrieben werden. Dass Firmen Hacker zum Ausspionieren der Konkurrenz anheuern, ist sicherlich kein unrealistisches Szenario. Es gibt Schätzungen, nach denen der jährlich durch Wirtschaftsspionage entstandene Schaden allein in Deutschland die Milliardengrenze weit überschreitet. Auch wenn nur ein Teil davon über Computernetze wie das Internet erfolgen dürfte, ist der in diesem Bereich entstehende Schaden beträchtlich.

2.3.2 Kommerz im Netz

Die Abwehr von Wirtschaftsspionage ist nicht die einzige Existenzberechtigung der Kryptografie. Genauso wichtig ist der Schutz von kommerziellen Transaktionen. Ein Beispiel dafür ist das Online-Bezahlen beim Einkaufen im Netz. Auch das Online-Banking muss geschützt werden. Mallory ist in diesem Fall also nicht in Geheimdienstkreisen, sondern im Hacker- oder Kriminellenmilieu zu suchen. Das hat auch Auswirkungen auf die zu verwendenden Krypto-Methoden. Während für den Schutz vor Geheimdiensten nur das Beste gut genug ist, müssen Krypto-Verfahren im Finanzbereich nicht unbedingt unknackbar sein. Niemand wird schließlich einen Millionenetat aufwenden, um sich im Internet vor der Bezahlung eines Paars Socken zu drücken oder um den Kontostand seines Nach-

barn zu erfahren. Dies zeigt, dass Kryptografie viel mit Verhältnismäßigkeit zu tun hat, und das wird in diesem Buch berücksichtigt.

2.3.3 Die Privatsphäre

Möchten Sie, dass ein per E-Mail verschickter Liebesbrief oder sonst eine private Mitteilung von einem Unbekannten (oder Bekannten) gelesen wird? Mithilfe der Kryptografie lässt sich so etwas verhindern. Als Privatperson sagen Sie jetzt vielleicht, Sie hätten generell nichts zu verbergen, und wer wollte schon die E-Mails, die Sie verschicken, lesen oder gar manipulieren. Wozu also Kryptografie? Dann müssen Sie sich jedoch auch fragen, warum Sie Ihre Briefe in einen Umschlag stecken, anstatt sie als Postkarte zu versenden. Trauen Sie Ihrem Postboten nicht? Sind es zu viele unbekannte Hände, durch die ein Brief geht? Oder ist es nur eine routinemäßige Vorsichtsmaßnahme, die ja nicht viele Umstände macht? Egal wie der Grund lautet, man kann ihn auch als Begründung für den Einsatz der Kryptografie verwenden.

Zudem ist nicht zu bestreiten, dass Geheimdienste und Ermittlungsbehörden auch Privatpersonen über das Netz überwachen. Wer weiß, was andere wohl so alles über Sie wissen und ob diese Erkenntnisse alle auf legale Weise gesammelt wurden. Hier hat die Kryptografie eine weitere gesellschaftliche Dimension: Sie ermöglicht es einem Individuum, den Schutz seiner Privatsphäre selbst sicherzustellen. Sie können mit Kryptografie-Unterstützung bis zu einem gewissen Grad selbst bestimmen, ob Ihre Kommunikation überwacht wird oder nicht. Ein solches Privileg ist vor allem in totalitären Staaten nicht zu unterschätzen. Phil Zimmermann, Entwickler der populären Verschlüsselungssoftware PGP (Pretty Good Privacy), betont nicht umsonst, dass er längst seine Wände mit Dankesschreiben von Widerstandsgruppen aus Unrechtsstaaten tapezieren kann. Gerade totalitäre Regime sind es auch, die den Einsatz von Kryptografie am härtesten bekämpfen.

2.4 Anwendungen der Kryptografie

Wozu kann die Kryptografie konkret eingesetzt werden? Die folgende Liste nennt einige Möglichkeiten, erhebt jedoch keinen Anspruch auf Vollständigkeit:

- *E-Mail*: Verschlüsselte E-Mails sind ein klassischer Anwendungsbereich der Kryptografie.
- *World Wide Web*: Da das WWW für kommerzielle Portale und für Geschäftsprozesse genutzt wird, ist ein Kryptografie-Einsatz oft dringend geboten.
- *Virtuelle Private Netze*: Unternehmen mit mehreren Niederlassungen koppeln ihre lokalen Netze oft über das Internet. Es lohnt sich, alle Daten beim Verlassen eines Firmennetzes zu verschlüsseln, um sie bei Wiedereintritt in das Fir-

menterrain zu entschlüsseln. Diese Technik wird als VPN (Virtuelles Privates Netz) bezeichnet.

- *Online-Bezahlsysteme*: Das Online-Bezahlen ist ohne Zweifel eine Sache, die kryptografisch abgesichert werden sollte.
- *Internet-Banking*: Wozu zur Bank gehen, wenn man einen Internetanschluss hat? Aber bitte nur mit Kryptografie.
- *Dateien*: Nicht nur übertragene Informationen, sondern auch Daten, die auf einer Festplatte oder einem Dateiserver liegen, müssen geschützt werden.

Sie sehen, es gibt viel zu tun für die Kryptografie. Allerdings hat jede der genannten Anwendungen ihre ganz speziellen Anforderungen an die Kryptografie, und das macht die Sache zwar kompliziert, dafür aber auch äußerst interessant.

2.5 Und wer zum Teufel ist Alice?

Noch einmal ein Wort zu unseren Freunden Alice und Bob: Wie bereits erwähnt, sind die beiden in einer misslichen Situation, denn sie wollen miteinander Daten austauschen, haben dafür aber nur ein von Mallory abhörbares Netz zur Verfügung. Immerhin hat diese Konstellation für uns einen Vorteil: Die Prinzipien der Kryptografie können wir uns anhand von Alice, Bob und Mallory veranschaulichen. Das Ganze ist übrigens keine Erfindung von mir – Alice, Bob und einige andere Charaktere findet man seit Jahrzehnten in großen Teilen der Kryptografie-Literatur.

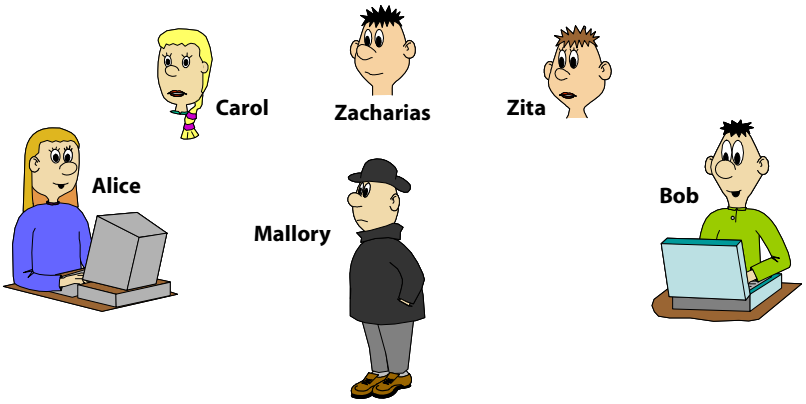
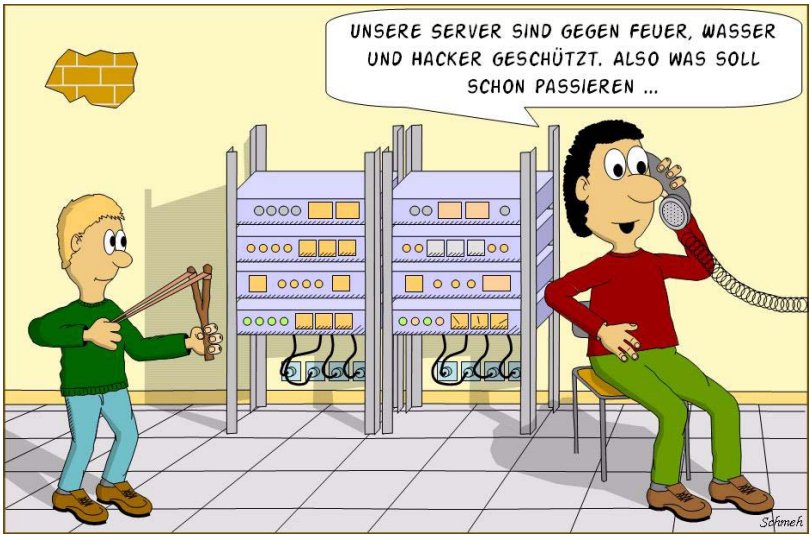


Abb. 2–3 *Mit Alice, Bob und dem Bösewicht Mallory werden in diesem Buch kryptografische Verfahren erklärt. Weitere Charaktere sind Carol, Zacharias und Zita.*

Bei der Konzeption dieses Buchs habe ich beschlossen, Alice, Bob und Mallory etwas mehr Leben einzuhauchen. Zunächst einmal gab ich den dreien eine Heimat: Sie leben in dem wenig bekannten Staat Kryptoland (Internetdomäne »kl«),

wo es eine Firma namens Krypt & Co. gibt. Neben Alice, Bob und Mallory treten noch andere Personen in diesem Buch auf: Alices Freundin Carol, Online-Shop-Besitzer Otto, Administrator Paul und die Alice und Bob vollkommen unbekannten Zacharias und Zita. Sie alle tragen dazu bei, dass ich kryptografische Sachverhalte möglichst anschaulich darstellen kann.

3 Wie Daten abgehört werden können



In diesem Buch kann Mallory jede zwischen Alice und Bob verschickte Nachricht mitlesen. Bevor wir so etwas aber als gegeben betrachten, sollten wir uns fragen, ob diese Annahme realistisch ist. Müssen wir wirklich bei jeder E-Mail, bei jeder Webseite, bei jedem Fernzugriff auf einen Rechner und bei jedem Telefonat mit einem unbekannten Dritten rechnen? Diese Frage ist ganz eindeutig mit »jein« zu beantworten. Ganz so krass wie bei Alice, Bob und Mallory geht es in der Realität zum Glück dann doch nicht zu. Andererseits ist die Gefahr dennoch groß genug, um unser Alice-Bob-Mallory-Modell als Grundlage für unsere Betrachtungen zu wählen. In den folgenden Unterkapiteln sehen wir uns die Sache genauer an.

3.1 Mallory am Übertragungsmedium

Eine naheliegende Möglichkeit, Daten in einem Computernetz abzuhören, besteht darin, die Übertragungsleitung direkt anzuzapfen. Dies bedeutet beispielsweise, dass sich Mallory an einem Kabel zu schaffen macht und die darin fließenden Informationen mitprotokolliert. Wie ein solcher Angriff funktioniert, hängt stark vom Übertragungsmedium ab. In jedem Fall benötigt Mallory ein Analysegerät, das die abgefangenen Signale in Daten umwandelt. Solche Protokoll-Analyser sind in der Regel legal erhältlich und haben für Wartungszwecke durchaus ihre Berechtigung. Der Preis hängt von der jeweiligen Übertragungstechnologie ab. Außerdem muss Mallory Zugang zum jeweiligen Medium haben, was bei einem Kabel sicherlich schwieriger ist als bei einer drahtlosen Übertragungstechnik.

3.1.1 Kupferkabel

Kupferkabel sind zwar nicht mehr die modernste Technik, werden aber nach wie vor häufig auf der Verbindung zwischen Telefonanschluss und Ortsvermittlungsstelle eingesetzt. In diesem Anwendungssegment ist es oft günstiger, vorhandene Kupferleitungen mit den speziell für diesen Zweck entwickelten Technologien ISDN oder DSL zu nutzen, als moderne Glasfaserleitungen zu verlegen. Kupferkabel ist für Mallory vergleichsweise einfach abzuhören, sofern er Zugang dazu hat. Bei Niederfrequenzsignalen (insbesondere bei einem analogen Telefongespräch) kann Mallory schon mit einer Metallklemme und einem einfachen Gerät die durchfließenden Daten abfangen. Mit einer etwas aufwendigeren Technik, die sich **induktives Ankoppeln** nennt, kommt Mallory auch bei Hochfrequenzsignalen, wie sie von ISDN und DSL verwendet werden, zum Erfolg.

3.1.2 Glasfaser

Glasfaserkabel sind gegenwärtig das leistungsfähigste Übertragungsmedium. Sie gelten als relativ abhörsicher, denn Mallory muss einen vergleichsweise großen Aufwand treiben, um an die darin fließenden Daten heranzukommen. Allerdings hat sich die zu diesem Zweck verfügbare Technik in den letzten Jahren deutlich verbessert. Mallory stehen drei Wege zur Verfügung:

- *Abhören am Spleiß*: Mallory kann eine Glasfaserleitung durchtrennen und durch Erhitzen wieder miteinander verbinden (eine solche Verbindung nennt man Spleiß). Am Spleiß tritt Strahlung aus, die man messen und in die ursprünglichen Datensignale zurückverwandeln kann. Allerdings sorgt ein Spleiß für einen nachweisbaren Lichtverlust, wodurch Mallorys Aktivitäten auffallen könnten.

- *Abhören an einer Biegung*: Wird eine Glasfaserleitung gebogen, dann tritt ebenfalls Strahlung aus, die für Mallory nutzbar ist. Allerdings geht auch hier ein Teil des Lichts verloren, was dem Betreiber der Leitung auffallen könnte.
- *Berührungsloses Abhören*: Inzwischen ist es mit empfindlichen Fotodetektoren sogar möglich, Licht aufzufangen, das ohne besonderes Zutun seitlich aus dem Kabel strahlt (Rayleigh-Streuung). Diese Art des Abhörens erzeugt keinen Lichtverlust und ist daher kaum nachweisbar.

Das Abhören von Glasfaserkabeln soll vor allem in Geheimdienstkreisen beliebt sein. Für Normalbürger ist diese Möglichkeit dagegen meist zu aufwendig.

3.1.3 Drahtlose Datenübertragung

Am einfachsten ist die Arbeit für Mallory, wenn Alice und Bob eine drahtlose Form der Datenübertragung nutzen. Es gibt verschiedene Technikvarianten:

- *Elektromagnetische Wellen*: Diese werden beispielsweise im Mobilfunk und für drahtlose lokale Netze (WLAN, WiMAX) eingesetzt. Da solche Wellen praktisch alles durchdringen, machen sie das Abhören einfach. Mallory braucht lediglich eine geeignete Antenne, um alle Daten empfangen zu können.
- *Laser*: Auch die drahtlose Datenübertragung per Laser ist mit geeignetem Gerät abhörbar.
- *Richtfunk*: Mallory kann Richtfunkstrecken abhören, wenn er sich in die Strecke zwischen den Empfangsantennen oder deren Verlängerung stellt [Faber].

3.1.4 Satellit

Satelliten kommen in der Telekommunikation oft dann ins Spiel, wenn ländliche oder abgelegene Gegenden versorgt werden sollen. Auch das Internet lässt sich über Satellit nutzen. Wie andere drahtlose Computernetz-Technologien ist auch eine Satellitenverbindung relativ einfach abzuhören. Die Gefahr ist sogar noch größer als bei einem Handy oder WLAN, da der Empfangsbereich eines Satelliten eine deutlich größere Fläche einnimmt. Mallory benötigt neben einer Satellitenschüssel lediglich einen PC mit Receiver-Karte und eine geeignete Software (gibt es kostenlos im Internet), um alle Daten empfangen zu können, die vom Satelliten kommen. Eine Untersuchung der Ruhr-Universität Bochum im Jahr 2004 zeigte, wie es geht. Zwei Forschern gelang es, innerhalb von nur 24 Stunden Name, Adresse, Geburtsdatum, Einkommen und EC-Kartennummer eines Opfers herauszufinden, indem sie eine Satellitenverbindung abhörten [RUB]. Immerhin ist die Unsicherheit des Internetzugangs über Satellit inzwischen so bekannt, dass viele Internet-Provider Verschlüsselungsmechanismen nachgerüstet haben.

3.2 Mallory am Gerät

Anstatt ein Übertragungsmedium zu überwachen, kann Mallory sein Glück auch an einem Gerät versuchen, das Daten speichert oder weiterleitet.

3.2.1 Netzkomponenten

Während Mallory an der Übertragungsleitung stets das verwendete Protokoll nachvollziehen muss, ist die Sache meist deutlich einfacher, wenn er sich an einer Netzwerkkomponente – etwa einem Router, einem Switch, einem Gateway oder einem Multiplexer – zu schaffen machen kann. Derartige Geräte bieten oft sogar Funktionen an, die ein einfaches Abhören ermöglichen. Die entsprechenden Techniken heißen beispielsweise Monitor-Ports, Mirror-Ports oder Trunk Access Points [Faber]. Auch eine Broadcast-Funktion, die von vielen Komponenten angeboten wird, lässt sich entsprechend missbrauchen. Allerdings ist es für Mallory naturgemäß schwieriger, an eine Netzkomponente heranzukommen, als an das Übertragungsmedium.

3.2.2 Mitlesen und Verändern von Dateien

Auch wenn Daten nicht verschickt werden, sondern auf einem PC oder Server gespeichert sind, sind sie gefährdet. Wenn Mallory beispielsweise mit dem Systemadministrator kooperiert, kommt er recht einfach an Daten auf dem Server heran. In der Regel gibt es in einem Unternehmen mehrere Systemadministratoren, und die Zahl der Personen mit privilegiertem Zugriff auf das Dateisystem steigt mit der Größe der Firma.

Das größte Problem, wenn es um gespeicherte Daten geht, sind zweifellos Laptops. Laptop-Diebstahl ist eines der häufigsten Delikte in den Industrieländern [zTrace]. Nach Angaben der US-Firma zTrace werden etwa 10 Prozent aller Laptops pro Jahr gestohlen. Von knapp 800 IT-Managern gaben 67 Prozent an, schon einmal einen Laptop-Diebstahl erlebt zu haben. Es ist klar, dass Mallory auf diese Weise recht einfach an vertrauliche Daten herankommt. Dabei nützt es auch nur bedingt, dass die gängigen Betriebssysteme einen Passwortschutz bieten, denn die Daten auf der Festplatte sind mit etwas Know-how auch ohne Passwort auslesbar.

3.3 Mallory in Computernetzen

Oft kann Mallory spezielle Eigenschaften eines Computernetzes nutzen, um lauschen zu können.

3.3.1 Telefon

In Deutschland und vielen anderen Ländern sind Telefongesellschaften dazu verpflichtet, Ermittlungsbehörden und Geheimdienste mithören zu lassen. Sie müssen sogar die für diesen Zweck benötigten technischen Möglichkeiten zur Verfügung stellen. Diese werden rege genutzt: Offizielle Zahlen des Bundestags belegen, dass die Anzahl der staatlich abgehörten Kommunikationsverbindungen inzwischen im zweistelligen Millionenbereich pro Jahr liegt – bei steigender Tendenz [CoWo]. Zum Glück können die Lauscher vom Dienst nur einen kleinen Teil davon genauer unter die Lupe nehmen, wodurch Normalbürger wie Alice und Bob normalerweise nicht die große Überwachung fürchten müssen.

Schwieriger wird es für Mallory, wenn dieser nicht die Unterstützung der Telefongesellschaft in Anspruch nehmen kann. Dann kommt er vielleicht zum Erfolg, wenn er sich bei Bob im Keller umschaut (Bob wohnt in einem großen Mietshaus), wo mit großer Wahrscheinlichkeit ein Schaltschrank hängt, in dem er sich in die zu Bob führende Leitung einklinken kann. Mallory kann theoretisch auch auf einen Masten klettern oder den Boden aufgraben, um an eine Leitung zu gelangen. Eine etwas einfachere Methode besteht darin, sich an den Verteilerkästen (meist ein grauer Kasten am Straßenrand) oder an der Ortsvermittlungsstelle (meist ein garagengroßes Gebäude) zu schaffen zu machen.

3.3.2 Abhören im LAN

Ein Local Area Network (LAN) ist ein Computernetz, das sich typischerweise über ein Firmen- oder Behördengelände erstreckt. Die Technologie, die in den meisten drahtgebundenen LANs zum Einsatz kommt, ist unter dem Namen IEEE 802.3 bekannt. Auch der Name **Ethernet** wird dafür verwendet.

Ungeswitchtes Ethernet

In der einfachsten Form besteht ein Ethernet-LAN aus mehreren Endgeräten (Computer, Drucker usw.), die alle mit demselben **Hub** verbunden sind. Alle am Hub hängenden Endgeräte haben eine eindeutige Nummer, die als **MAC-Adresse** bezeichnet wird. Will beispielsweise Alices PC über das LAN einen Drucker ansprechen, dann sendet er ein Datenpaket an den Hub. Das Datenpaket enthält neben dem Druckauftrag die MAC-Adresse von Alices PC (Absender) sowie die MAC-Adresse des Druckers (Empfänger). Der Hub nimmt das Datenpaket entgegen und leitet es an alle anderen angeschlossenen Endgeräte weiter. Wenn der

Drucker (oder ein anderes Endgerät) ein Datenpaket empfängt, betrachtet er zunächst die Empfängeradresse. Ist diese die eigene, dann nimmt er das Paket entgegen. Ist es nicht die eigene, wird das Datenpaket nicht weiter beachtet.

Wie man sich leicht vorstellen kann, ist ein Ethernet mit Hub (man nennt dies ein ungeswitchtes Ethernet) äußerst abhöranfällig. Wenn Mallory Zugang zu einem PC im LAN hat, landen sämtliche darin verschickten Nachrichten bei ihm. Mallory muss lediglich dafür sorgen, dass seine Netzwerkkarte nicht nur Datenpakete mit der eigenen MAC-Adresse als Empfänger, sondern auch alle anderen Datenpakete entgegennimmt. Die meisten Netzwerkkarten haben eine entsprechende Konfigurationsmöglichkeit eingebaut (**Promiskuitätsmodus**). Mit einer geeigneten Software (**Sniffer**) kann Mallory im Promiskuitätsmodus den gesamten Netzverkehr mitlesen.

Geswitchtes Ethernet

Deutlich mehr Sicherheit im LAN ergibt sich, wenn statt eines Hubs ein Switch genutzt wird (man nennt dies auch ein geswitchtes Ethernet). Ein Switch funktioniert ähnlich wie ein Hub, er merkt sich jedoch die MAC-Adressen der angeschlossenen Endgeräte. Erhält der Switch ein Datenpaket, dessen Empfänger-MAC-Adresse er kennt, dann leitet er dieses nur an das entsprechende Endgerät weiter. Die anderen Endgeräte im LAN bekommen nichts davon mit. Bei einem geswitchten LAN hat Mallory zunächst keine Chance, Datenpakete abzufangen, die nicht an ihn adressiert sind. Er kann sich jedoch der Technik des ARP-Spoofings (siehe Abschnitt 3.4.1) bedienen, um seine eigene MAC-Adresse als die eines anderen Endgeräts auszugeben.

3.3.3 ISDN-Sicherheitsprobleme

ISDN (Integrated Services Digital Network) ist eine Technik, die über bereits vorhandene Kupferleitungen einen digitalen Telefonanschluss realisiert. ISDN bietet verschiedene Features (Leistungsmerkmale), wie Konferenzschaltungen, Anrufweiterleitungen oder Freisprechen. Darüber hinaus lässt sich eine ISDN-Anlage auf vielfältige Weise programmieren, wobei über eine ISDN-Verbindung auch eine Fernwartung möglich ist. In der Anfangszeit wurden über die Sicherheit von ISDN wahre Horrorgeschichten erzählt. Mallory kann sich beispielsweise per Fernwartung Zugang zu einer ISDN-Anlage verschaffen und diese umprogrammieren, um die Leistungsmerkmale zu seinen Gunsten zu nutzen. Mit dem Leistungsmerkmal Freisprechen kann er beispielsweise das im Telefon vorhandene Mikrofon aktivieren, um einen Raum abzuhören. Das Leistungsmerkmal Dreierkonferenz kann Mallory missbrauchen, um von einem beliebigen Anschluss aus unbemerkt einem beliebigen Telefongespräch beizuwohnen. Angeblich gibt es Geräte, mit denen ein solches Abhören von jedem Anschluss aus durchführbar ist.

Es gibt auch ISDN-Leistungsmerkmale mit den Namen Zeugenschaltung und Abhören. Diese Funktionen, die in einigen Ländern vorgeschrieben sind, sind zwar in Deutschland verboten. Allerdings machen sich manche Hersteller nicht die Mühe, unterschiedliche Versionen ihrer Geräte anzubieten. Stattdessen verschweigen sie lediglich die verbotenen Leistungsmerkmale in der Dokumentation – zur Freude von Mallory.

Trotz dieser alarmierenden Meldungen über ISDN-Schwachstellen gibt es kaum Berichte über tatsächliche Vorfälle, in denen ein Hacker eine ISDN-Anlage entsprechend manipuliert hätte. Abhörer Mallory benötigt also in jedem Fall eine Menge Know-how und Insiderwissen, um auf diese Weise zum Erfolg zu kommen. Dennoch muss man davon ausgehen, dass ISDN für Abhörexperten ein äußerst interessantes Betätigungsfeld ist.

3.3.4 DSL

DSL (Digital Subscriber Line) kann man als Nachfolger von ISDN betrachten. Es bietet ähnlich wie ISDN einen digitalen Anschluss an das Telefonnetz über vorhandene Kupferleitungen, wobei die Datenraten allerdings um ein Vielfaches höher sind. Im Vergleich zum Vorgänger scheint DSL deutlich besser geschützt zu sein, denn bisher ist nichts über Sicherheitslücken bekannt geworden, die mit denen von ISDN vergleichbar sind.

3.3.5 Mobilfunk

Als Mobilfunk noch eine analoge Angelegenheit war (zu Zeiten des C-Netzes), hatte Mallory leichtes Spiel. Schon mit einem Scanner und einem Invertierungsdecoder (gab es für etwa 500 Euro) konnte er damals Telefongespräche belauschen [BaHaJK]. In den neunziger Jahren kam jedoch der digitale Mobilfunk auf (zunächst gemäß dem GSM-Standard) und machte die Sache deutlich schwerer. GSM sowie dessen Nachfolger UMTS und LTE sehen bei der Datenübertragung einen ständigen Frequenzwechsel vor (**Frequency Hopping**), der ein Abhören nur mit aufwendigen Geräten erlaubt. Außerdem werden alle Daten verschlüsselt übertragen (allerdings nur zwischen Handy und Basisstation).

Beim GSM-Standard kann Mallory einen sogenannten IMSI-Catcher einsetzen. Ein solcher täuscht vor, eine Basisstation zu sein, und verleitet dadurch ein Handy, sich bei ihm einzubuchen. Über den IMSI-Catcher kann Mallory dann die gesamte Kommunikation des Handys abhören. IMSI-Catcher werden insbesonderen von Ermittlungsbehörden und Geheimdiensten eingesetzt. Bei den neueren Techniken (UMTS und LFE) funktioniert diese Technik jedoch nicht mehr. Hobby-Hacker haben daher heute im Mobilfunk kaum noch eine Chance. Polizei und Geheimdienst können ein Mobilgespräch jedoch – im Rahmen der gesetzlichen Möglichkeiten – genauso abhören wie jedes andere.

3.3.6 WLANs

Wireless LANs (WLANs) sind heute in vielen Büros, Restaurants und Privatwohnungen das Mittel der Wahl zur Einwahl in das Internet. Wie alle drahtlosen Netztechnologien ist auch das WLAN prinzipiell äußerst abhöranfällig. Zum Glück waren sich die Schöpfer des WLAN-Standards dieser Gefahr bewusst. Sie sahen deshalb eine Verschlüsselungsfunktion vor, die Abhörern wie Mallory die Arbeit deutlich erschwert. Allerdings kann man ein WLAN auch ohne Verschlüsselung betreiben, was manche Anwender leichtsinnigerweise tatsächlich tun. Darüber hinaus hat sich die erste Generation der WLAN-Verschlüsselung als nicht besonders sicher erwiesen (siehe Abschnitt 33.2.1). Auch drahtlose Netztechnologien wie Bluetooth, Zigbee oder WiMAX sind durch eine eingebaute Verschlüsselung geschützt – für Mallory wird es daher schwierig.

3.4 Mallory im Internet

Der Aufbau und die Funktionsweise des Internets bieten Mallory einige weitere interessante Angriffsmöglichkeiten. Viele davon gehören zur Familie der Spoofing-Angriffe. Als **Spoofing** bezeichnet man das Vortäuschen falscher Tatsachen, in diesem Fall vor allem das Fälschen von Datenpaketen.

3.4.1 ARP-Spoofing

Jedes Endgerät in einem LAN hat eine MAC-Adresse. Im Internet ist jedoch nicht die MAC-Adresse, sondern die IP-Adresse von Bedeutung. Will ein Gerät im LAN andere Komponenten auf IP-Ebene ansprechen, dann muss es eine Tabelle führen, in der zur jeweiligen IP-Adresse die passende MAC-Adresse eingetragen ist. Da IP-Adressen in der Praxis oft dynamisch vergeben werden (MAC-Adressen bleiben dagegen meist gleich), ändert sich diese Tabelle häufig. Weiß ein Endgerät nicht, welche MAC-Adresse sich hinter einer IP-Adresse verbirgt, dann kann es mithilfe des *Address Resolution Protocol* (ARP) eine Anfrage an alle Rechner im LAN schicken. Das Endgerät mit der passenden IP-Adresse (oder ein anderes, das dieses Endgerät kennt) meldet sich dann.

Wenn Mallorys PC an ein LAN angeschlossen ist, dann kann er ARP zu einem interessanten Angriff (**ARP-Spoofing**) missbrauchen. Dazu muss er seinen PC so konfigurieren, dass dieser auf eine ARP-Anfrage eine falsche Anwort liefert. Fragt jemand, welche MAC-Adresse sich hinter einer bestimmten IP-Adresse verbirgt, dann antwortet Mallorys PC mit der eigenen MAC-Adresse. Auf diese Weise werden die Tabellen der Endgeräte im LAN manipuliert, und Mallory gehen Datenpakete zu, die nicht für ihn bestimmt sind. Er kann sogar Datenpakete manipulieren, bevor er sie an den rechtmäßigen Empfänger weiterleitet. ARP-Spoofing funktioniert auch und gerade in geswitchten LANs.

3.4.2 Abhörangriffe auf Router

Ein Router leitet Datenpakete durch das Internet. Wie viele andere Netzkomponenten ist auch ein Router ein interessanter Angriffspunkt für Mallory. Da dies bekannt ist, ist ein Router normalerweise nicht für jedermann zugänglich, schon gar nicht für einen gänzlich Unbeteiligten. Mit krimineller Energie oder mit Unterstützung eines Geheimdiensts lässt sich das aber möglicherweise umgehen. Hat Mallory den entsprechenden Zugriff, dann kann er einen Router so konfigurieren, dass dieser Pakete mit bestimmten Eigenschaften kopiert oder umleitet (etwa auf einen Router, der sich unter Mallorys Kontrolle befindet). Zwar lässt sich bei einer Nachricht im Internet nie im Voraus sagen, über welche Router sie ihren Weg zum Ziel findet. Wenn Mallory sich jedoch den Router eines Providers vorknöpft, dann kann er zumindest alle Nachrichten lesen, die von dessen Kunden verschickt werden oder diese erreichen. Eine interessante Möglichkeit bietet sich für Mallory außerdem durch eine Funktion des IP-Protokolls, die sich Source Routing nennt. Mit dieser kann ein Absender festlegen, über welche Router ein IP-Paket seinen Weg nimmt. Hat Mallory einen Komplizen bei der Firma Krypt & Co., dann kann dieser dafür sorgen, dass alle IP-Pakete, die von dieser Firma über das Internet geschickt werden, über einen Router geleitet werden, zu dem er Zugang hat.

3.4.3 IP-Spoofing

Auch ohne Zugang zu einem Router kann Mallory **IP-Spoofing** betreiben. Dabei generiert er IP-Pakete, in denen als Absender eine falsche IP-Adresse angegeben wird. Mit einer geeigneten Software ist dies nicht allzu schwierig. IP-Spoofing ist eine interessante Methode zum Fälschen von Daten. Für Abhöraktivitäten kann Mallory diese Technik allerdings nicht nutzen, da alle Antwortpakete des Empfängers an die gefälschte Adresse gehen. IP-Spoofing ist jedoch eine Voraussetzung für einen weiteren Angriff: DNS-Spoofing.

3.4.4 DNS-Spoofing

DNS-Spoofing ist eine weitere Gemeinheit aus Mallorys Trickkiste. Der DNS (Domain Name System) ist ein Internetdienst, der zu einer Textadresse (z.B. www.kryptoboerse.kl) die zugehörige IP-Adresse (z.B. 153.125.34.43) liefert. Wenn Alice also »www.kryptoboerse.kl« in ihren Webbrowser eintippt, um Aktienkurse abzurufen, dann sendet der Browser zunächst eine Anfrage an einen DNS, um die zugehörige IP-Adresse zu erhalten (sofern der Browser die IP-Adresse nicht bereits kennt). Mit dieser IP-Adresse holt sich der Webbrowser dann die gewünschte Seite und zeigt sie Alice an.

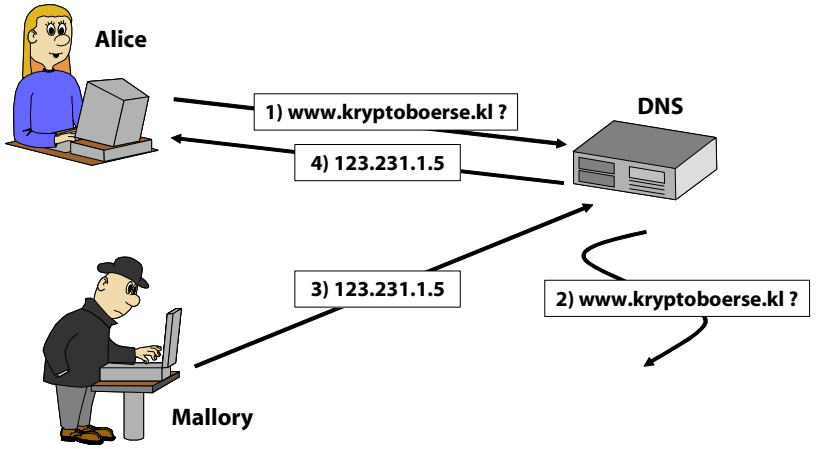


Abb. 3–1 *Beim DNS-Spoofing sorgt Mallory dafür, dass Alice eine falsche IP-Adresse vom DNS erhält.*

DNS-Spoofing bedeutet, dass Mallory einem DNS-Nutzer (in unserem Fall ist dies Alices Browser) eine falsche IP-Adresse unterjubelt. Diese falsche IP-Adresse kann dann zu einer von Mallory gefälschten Seite führen, auf der falsche Aktienkurse zu finden sind. Wenn die falsche Seite gut gemacht ist (wenn sie also wie die echte aussieht), dann hat Alice kaum eine Chance, den Schwindel zu bemerken. Ihr bescheidenes Vermögen, das sie in Aktien angelegt hat, könnte so schnell schrumpfen.

DNS-Spoofing gibt es in verschiedenen Varianten. Eine gängige Methode sieht etwa so aus: Mallory sendet eine DNS-Anfrage an einen DNS (nennen wir ihn DNS 1), in der er nach der IP-Adresse von www.kryptoboerse.kl fragt. Wenn Mallory Glück hat, kennt DNS 1 die gefragte Adresse nicht und stellt nun selbst eine Anfrage über das Internet an DNS 2. Mallory schickt jedoch seiner Anfrage eine falsche Antwort hinterher und kommt so der Antwort von DNS 2 zuvor. DNS 1 hält nun die falsche, von Mallory erhaltene Adresse für die echte. Er sendet daher die falsche Adresse als Antwort an Mallory zurück (was Mallory nicht weiter interessiert) und merkt sich diese (technisch gesprochen: Sie wird in einem Cache gespeichert). Wenn nun in der Folgezeit wieder jemand nach der IP-Adresse von www.kryptoboerse.kl fragt, erhält er die von Mallory angegebene Adresse.

Eine Hürde hat das DNS-Protokoll allerdings eingebaut: Jede Anfrage erhält eine 16-Bit-Nummer (*Query-ID*), die in der Antwort angegeben sein muss, damit sie akzeptiert wird. Um eine Antwort fälschen zu können, die von DNS 2 an DNS 1 geliefert wird, muss Mallory diese Query-ID erraten. Dies ist jedoch nicht gerade schwierig, denn viele DNS-Implementierungen erhöhen die Query-ID mit jeder neuen Anfrage um eins. Mallory muss also lediglich eine unverfängliche Anfrage nach seiner eigenen Adresse an DNS 1 stellen, um an die aktuelle Query-ID heranzukommen. Die nächste Query-ID kann er dann erraten. Sicherheitshal-

ber kann Mallory auch mehrere Antworten mit unterschiedlichen Query-IDs fälschen und abschicken, in der Hoffnung, dass eine stimmt (mehrere widersprüchliche Antworten stören einen Nameserver meist nicht). Es ist sogar denkbar, dass Mallory 65.535 Antworten mit allen infrage kommenden Query-IDs abschickt.

Beim DNS-Spoofing sorgt Mallory dafür, dass Alice eine falsche IP-Adresse vom DNS erhält.

3.4.5 Mitlesen von E-Mails

Nahezu jeder Internetnutzer hat schon einmal eine Mail erhalten, die nicht für seine Augen bestimmt war. Meist liegt dies daran, dass der Absender sich bei der E-Mail-Adresse vertippt hat. Manchmal ist auch ein Fehler im Mailsystem schuld. Wesentlich gefährlicher als solche versehentlichen Fehlzustellungen sind jedoch gezielte Abhöraktionen durch Insider. Besitzt Mallory Administratorenrechte auf einem Mailserver, dann kann er sämtliche Mails eines ganzen Unternehmens oder aller Kunden eines Internet-Providers lesen und verändern. Da eine E-Mail immer als Ganzes auf einem Mailserver liegt und dort auch archiviert wird, gibt es kaum eine interessantere Angriffsstelle.

Auch bereits beschriebene Angriffe wie das Abhören von LANs oder WLANs wirken besonders gut gegen E-Mails. Dies liegt daran, dass E-Mails in einem menschenlesbaren Format übertragen werden und daher beispielsweise schon bei einem Blick auf den Sniffer auffallen. Wenn Mallory eine E-Mail nicht abhören, sondern fälschen will, wird seine Arbeit noch einfacher. Er muss einfach eine E-Mail mit falschem Absender verschicken (**Mail-Spoofing**). Mail-Spoofing wird von vielen SPAM-Versendern angewandt und ist nicht besonders schwierig. Wer selbst einen Mailserver aufsetzen kann, hat damit keine große Mühe. Es gibt auch Webseiten, die das Verschicken einer Mail mit beliebigem Absender anbieten.

3.4.6 URL-Spoofing

Beim **URL-Spoofing** macht sich Mallory eine Funktion namens URL-Rewriting zunutze, die mit den gängigen Webserver-Produkten realisiert werden kann. URL-Rewriting ermöglicht folgendes Vorgehen: Wenn Bob die Adresse »http://www.mallory.kl/http://www.kryptoboerse.kl« mit seinem Browser aufruft, dann landet die Anfrage erst einmal bei Mallorys Webserver (dieser hat die Adresse www.mallory.kl). Mittels URL-Rewriting ruft Mallorys Webserver nun die Webseite www.kryptoboerse.kl auf und leitet diese an Bob weiter. Wenn Bob sich die aufgerufene Adresse nicht genau ansieht, dann merkt er nicht, dass Mallory sich dazwischengeschaltet hat. Richtig gefährlich wird es spätestens dann, wenn Mallory den Inhalt der ursprünglichen Seite ändert. Damit URL-Spoofing funktioniert, muss Mallory Bob erst einmal die falsche Adresse unterjubeln. Dies kann er etwa dadurch tun, dass er auf einer von ihm erstellten Webseite auf die falsche Adresse verweist.

3.4.7 Abhören von Internettelefonie

Das Telefonieren über das Internet (Voice over IP) hat sich in den letzten Jahren zu einer populären Alternative zum herkömmlichen Telefon entwickelt. Allerdings verzichten viele VoIP-Systeme auf eine Verschlüsselung und ermöglichen daher das Mithören. Mallory muss für eine Abhöraktion jedoch erst einmal an die Datenübertragung herankommen. Wenn sich Alices Rechner in einem LAN oder WLAN befindet, kann er dazu die in den Abschnitten 3.3.2 und 3.3.6 beschriebenen Tricks nutzen. Ansonsten braucht er Zugang zu einem Gateway, den meist nur Administratoren haben. Sind diese Voraussetzungen geschaffen, dann ist der Rest ein Kinderspiel: Mit einer VoIP-Software kann Mallory mithören.

3.5 Ein paar Fälle aus der Praxis

Nachdem Sie nun wissen, wie Mallory unseren beiden Helden Alice und Bob gefährlich werden kann, will ich durch einige Beispiele belegen, dass das bisher Gesagte auch in der Praxis funktioniert. Die hier geschilderten Fälle bilden nur die Spitze eines riesigen Eisbergs, denn man kann davon ausgehen, dass nur ein Bruchteil der Aktivitäten von Mallory und Co. jemals entdeckt werden (manche Schätzungen gehen von unter fünf Prozent aus). Hinzu kommt, dass von den entdeckten Fällen nur die wenigsten an die Öffentlichkeit gelangen.

3.5.1 Abgehörte E-Mails

Dass manche Administratoren fleißig die E-Mails mitlesen, die über einen Mailserver laufen, ist nichts Neues. Bekannt wird dies meist nur dann, wenn ein Lauscher auf diese Weise brisante Informationen erhält und diese der Öffentlichkeit zuspielt. Hier ein paar Beispiele:

- *Blodget-Fall*: Im Jahr 2002 gelangten einige interne E-Mails der Investment-Bank Merrill Lynch an die Öffentlichkeit. Peinlich war dies vor allem für den Merrill-Lynch-Mitarbeiter Henry Blodget, der in einer Mail eine Aktie als POS (gemeint war »Piece Of Shit«) bezeichnete, während er diese in der Öffentlichkeit anpries. Blodget wurde von der US-Börsenaufsicht angeklagt und erhielt ein lebenslängliches Tätigkeitsverbot.
- *Grubman-Fall*: Jack Grubman, ein weiterer US-Analyst, prahlte in einer E-Mail, dass die Kinder von Citigroup-Chef Sandy Weill einen Platz in einem exklusiven Kindergarten erhalten hätten, nachdem dieser den Kurs einer bestimmten Aktie nach oben befördert hatte. Diese Mail wurde öffentlich und trug dazu bei, dass Weill 2003 seinen Hut nehmen musste.
- *Quattrone-Fall*: Der Wall-Street-Banker Frank Quattrone kam 2006 nur knapp an einer Gefängnisstrafe vorbei, nachdem einige seiner E-Mails in die falschen

Hände gelangt waren. Die abgefangenen Nachrichten belegten, dass Quattrone Beweisstücke hatte verschwinden lassen [Donway].

- *Stonecipher-Fall*: Dem ehemaligen Boeing-Chef Harry Stonecipher wurde seine E-Mail-Korrespondenz mit einer Kollegin, mit der er ein Verhältnis hatte, zum Verhängnis. Die (vermutlich von einem Mitarbeiter der IT-Abteilung) abgefangenen Nachrichten belegten, dass Stonecipher gegen einen Kodex verstoßen hatte, an dessen Entstehung er selbst mitgearbeitet hatte. Stoneciphers Ehe war nach dieser Affäre zu Ende. Seine Karriere bei Boeing ebenfalls.
- *Hotmail-Fall*: 1999 wurde bekannt, dass die Postfächer des von Microsoft betriebenen E-Mail-Dienstes Hotmail ohne Passwort zugänglich waren. 40 Millionen Anwender waren betroffen. Die Panne kam durch eine Java-Sicherheitslücke zustande, die von schwedischen Hackern entdeckt wurde. Wie viele vertrauliche E-Mails bei dieser Gelegenheit in die falschen Hände gerieten und welcher Schaden dabei entstand, ist nicht bekannt.
- *Texas-Police-Fall*: Im September 2011 gab eine Hackergruppe bekannt, über Monate hinweg den E-Mail-Verkehr der texanischen Polizei mitgelesen zu haben. Die erbeuteten Informationen wurden im Internet veröffentlicht. Mit dieser Aktion wollten die Hacker gegen das ihrer Meinung nach rassistische Verhalten der texanischen Polizei protestieren [Short].
- *Assad-Fall*: Ebenfalls 2012 gelangten Oppositionelle an etwa 3.000 E-Mails des syrischen Diktators Baschar al-Assad. Die Inhalte waren wenig schmeichelhaft [Freibu].

3.5.2 Abgehörte Telefonate

Auch abgehörte Telefonate sind keine Seltenheit:

- *Stallone und der Detektiv*: 2008 wurde der Detektiv Anthony Pellicano in Los Angeles zu einer langjährigen Haftstrafe verurteilt, weil er die Telefonleitungen von Sylvester Stallone und anderen Hollywood-Stars angezapft hatte [BurCon].
- *Prinz Charles und Lady Diana*: Auch die sensationshungrige britische Boulevard-Presse schlüpft bisweilen in die Rolle eines Mallory. In den achtziger Jahren, als in Großbritannien noch ein analoges Funktelefonsystem (vergleichbar mit dem deutschen C-Netz) im Einsatz war, gehörten sowohl Prinz Charles als auch Lady Diana zu den Leidtragenden – allerdings unabhängig voneinander. Diana wurde beim Telefonat mit einem Liebhaber belauscht, ihren früheren Ehemann traf es während eines Ferngesprächs mit Camilla Parker-Bowles. Dabei soll Prinz Charles einen Satz gesprochen haben, der fast zum geflügelten Wort wurde: »Lass mich dein Tampon sein.«

- *Angezapfte Handys in Griechenland*: Im März 2005 fiel Mitarbeitern der Telefongesellschaft Vodafone in Griechenland eine Häufung von Kundenbeschwerden auf. Die Suche nach dem Fehler brachte Interessantes zutage: Auf den Servern des Mobilfunkanbieters war ein Schnüffelprogramm installiert, das die Telefonate bestimmter Vodafone-Kunden aufzeichnete und weiterleitete [Aswest]. Etwa 100 Mobiltelefone sollen über ein Jahr lang auf diese Weise abgehört worden sein. Ziel der Observation waren offensichtlich Mitglieder der Regierung, hochrangige Militärs und Polizeibeamte.
- *Der BND im Irak*: Anfang 2006 kam eine politische Affäre ins Rollen, in deren Mittelpunkt der Bundesnachrichtendienst (BND) stand [Welt]. Stein des Anstoßes war der Verdacht, dass der BND den USA während des Irak-Kriegs Informationen über Bombardierungsziele in Bagdad geliefert haben soll. Als Quelle wurden zwei BND-Mitarbeiter genannt, die während des Kriegs in Bagdad verblieben waren. An dieser Stelle ist für uns vor allem interessant, was die Leipziger Volkszeitung im Februar 2006 berichtete: Da bei Kriegsbeginn die sichere Kommunikationsverbindung zwischen dem Irak und der deutschen Zentrale ausgefallen war, bedienten sich die BND-Mitarbeiter eines Satellitentelefons. Dieses wurde angeblich von einem US-Geheimdienst abgehört. So soll es in einem vertraulichen Bericht der Bundesregierung gestanden haben.
- *Abhörskandal in Italien*: 2006 nahm die Staatsanwaltschaft in Mailand Ermittlungen wegen eines Abhörskandals auf [Spie06]. Mitarbeitern eines italienischen Mobilfunkanbieters wurde vorgeworfen, widerrechtlich und im großen Stil Telefongespräche abgehört zu haben. Die dabei gewonnenen Informationen sollen einerseits dem italienischen Geheimdienst SISMI zugespielt worden sein. Andererseits fanden Ermittler eine DVD mit zahlreichen Abhörprotokollen bei einer Detektei. Es handelte sich laut Presseberichten um sensibles Material, das dazu geeignet war, die betroffenen Personen – darunter auch Prominente, Politiker und Fußballgrößen – zu erpressen.
- *Belusconi*: Auch der ehemalige italienische Ministerpräsident Silvio Berlusconi hatte schon unter abgehörten Telefonaten zu leiden [Südd11]. Als die Polizei gegen einen Geschäftspartner des Politikers ermittelte, hörte sie dessen Telefon ab und zeichnete dabei auch einige Telefonate mit Berlusconi auf. Dumm nur, dass diese Aufzeichnungen im September 2011 in die Hände der italienischen Boulevard-Presse gelangten. Unter anderem wurde nun öffentlich: In einem Gespräch hatte Berlusconi damit geprahlt, dass elf Frauen vor seinem Zimmer Schlange gestanden hätten, um mit ihm die Nacht zu verbringen. Er habe aber nur mit acht von ihnen geschlafen, weil »du es nicht mit allen treiben kannst«.

3.5.3 Abgehörte Faxe

Neben E-Mails werden auch Faxe gerne mitgelesen:

- *Die Serben und die NATO*: 1999 warf die NATO Bomben auf Serbien. Später wurde bekannt: Verschiedene Einsatzbefehle wurden unverschlüsselt per Fax vom NATO-Hauptquartier in die Hauptstädte der Mitgliedsländer geschickt. Die Serben hörten mit und wussten Bescheid.
- *Onyx und das abgefangene Fax*: Das weiter unten erwähnte Überwachungssystem Echelon, das unter anderem von der NSA genutzt wird, ist nicht das einzige seiner Art. Seit dem Jahr 2005 ist bekannt, dass auch die Schweiz eine solche Einrichtung betreibt: das Abhörsystem *Onyx* [Engele]. Dieses hat offenbar die Aufgabe, über geeignete Parabolantennen Satellitenkommunikation abzuhören und auszuwerten. Damit soll der Terrorismus bekämpft werden. Die Existenz von Onyx wurde bekannt, nachdem eine undichte Stelle Informationen darüber der Presse zuspielte. Im Januar 2006 veröffentlichte eine Schweizer Boulevard-Zeitung Inhalte eines Faxes, das aus dem ägyptischen Außenministerium stammte. Darin wurden frühere Berichte anderer Quellen über CIA-Geheimgefängnisse in Osteuropa für Terrorismusverdächtige erneut bestätigt und erstmalig auch Nationalitäten der Gefangenen genannt.
- *Siemens und der ICE*: Der bisher spektakulärste bekannt gewordene Fall von Spionage durch das Abhören von Nachrichten traf 1993 die Firma Siemens [Ulfkot]. Der Münchner Weltkonzern machte sich damals Hoffnungen auf einen Auftrag der Regierung Südkoreas für den Bau eines Hochgeschwindigkeitszugs nach Vorbild des ICE. Offensichtlich gelang es dem französischen Geheimdienst, ein Fax mit dem Angebot von Siemens abzufangen und der französischen Konkurrenz zuzuspielen. In Frankreich war man nach dem Abhörangriff in der Lage, das Angebot zu unterbieten, und so entging Siemens ein Milliardengeschäft.

Abb. 3–2 *Radarkuppeln der (inzwischen geschlossenen) Echelon-Station in Bad Aibling*

3.5.4 Weitere Fälle

Hier sind noch einige weitere Fälle, in denen das Abhören einer Datenübertragung eine Rolle spielte:

- *Echelon*: Mallorys Kollegen sind zweifellos auch in Geheimdienstkreisen zu suchen, auch wenn über derartige Aktivitäten naturgemäß wenig bekannt ist. Bekannt sind jedoch einige Details über das Überwachungssystem **Echelon**, ein Gemeinschaftswerk der Staaten USA, Großbritannien, Kanada, Neuseeland und Australien. Echelon hat den Zweck, Satellitenkommunikation abzuhören. Echelon-Abhörstationen stehen bzw. standen beispielsweise in Darmstadt (Deutschland), Menwith Hill (Großbritannien) und Sugar Grove (USA). Wenn man weiß, was heutzutage alles über Satelliten gesendet wird (Telefonate, Internetdaten, ...), dann kann man sich leicht vorstellen, welche Möglichkeiten Echelon den Betreiberstaaten bietet. Früher galt Echelon für viele als Hirngespinst von Verschwörungstheoretikern. Im Januar 1998 erschien jedoch ein Bericht der EU, in dem die Existenz von Echelon erstmals von offizieller Stelle bestätigt wurde (der sogenannte STOA-Report [STOA]). Gemäß dem STOA-Report existiert Echelon bereits seit den siebziger Jahren. Interessant sind auch andere Zahlen aus dem STOA-Report: 15 bis 20 Milliarden Euro sollen weltweit jährlich für das Abhören von Kommunikation ausgegeben werden. Über den heutigen Zustand von Echelon ist wenig bekannt.
- *Enercon und die NSA*: Die ostfriesische Firma Enercon, ein Hersteller von Windkraftanlagen, wurde laut Medienberichten Ende der neunziger Jahre durch die NSA abgehört [Schm02]. Dabei gelangte die NSA in den Besitz von geheimen Bauplänen, die der US-Konkurrenz des Auricher Unternehmens zugespielt wurde. Vermutlicher Schaden: 100 Millionen Mark und 300 Arbeitsplätze.

3.6 Ist Kryptografie gefährlich?

Obwohl die Kryptografie in zahlreichen Anwendungsgebieten unverzichtbar ist, können von ihr auch Gefahren ausgehen. Dies ist insbesondere dann der Fall, wenn Kriminelle Daten verschlüsseln und sie dadurch für die Polizei unlesbar machen. Die öffentliche Diskussion zu diesem Thema schlug besonders Ende der neunziger Jahre hohe Wogen und ist bis heute nicht ganz verebbt. Im Mittelpunkt steht dabei stets folgende Frage: Soll wirkungsvolle Verschlüsselung für jedermann zugänglich sein oder soll der Staat gegen die Verbreitung der Kryptografie vorgehen?

Gesetze, die einen unkontrollierbaren Einsatz von Verschlüsselung verhindern sollen, gibt oder gab es in verschiedenen Ländern. In den USA galt beispielsweise bis 1999 ein Exportverbot für starke Verschlüsselungstechnik. Selbst Standardprodukte wie Betriebssysteme oder Webbrowser durften nur mit abgespeck-

ten Verschlüsselungsfunktionen das Land verlassen. Davon profitierten nicht zuletzt einige deutsche Hersteller, die für ihre Krypto-Produkte dankbare Abnehmer fanden. In Frankreich machte man sich lange Zeit strafbar, wenn man eine Verschlüsselungssoftware verwendete. Unternehmen konnten sich zwar eine Lizenz zum Verschlüsseln einholen, an Privatpersonen wurde eine solche jedoch nicht vergeben. Anfang 2000 wurden diese Regeln aufgehoben.

Eine Kompromisslösung zwischen Krypto-Verbot und Krypto-Freiheit besteht darin, dass Daten zwar verschlüsselt werden dürfen, die verwendeten Schlüssel jedoch bei einer staatlichen Stelle hinterlegt werden müssen (**Schlüsselhinterlegung**). Diese staatliche Stelle darf die Schlüssel nur in Ausnahmefällen herausgeben, etwa wenn eine richterliche Genehmigung vorliegt. Der Schlüssel kann hierbei aus Sicherheitsgründen auf mehrere Hinterlegungsstellen aufgeteilt werden. Das Thema Schlüsselhinterlegung wurde Ende der neunziger Jahre in den USA diskutiert, als die dortige Regierung die Einführung von Verschlüsselungs-Chips mit hinterlegtem Schlüssel plante (die Chips hießen *Clipper* und *Capstone*). Der Plan scheiterte jedoch am Widerstand der Öffentlichkeit und an technischen Problemen.

Abb. 3–3 *Unter dem Motto »Wird Datenschutz im Netz zur Gefahr für den Staat?« schaltete die Gesellschaft für Informatik 1997 ganzseitige Werbeanzeigen, um sich gegen eine Krypto-Regulierung einzusetzen.*

In Deutschland, Österreich und der Schweiz gibt es derzeit keine Gesetze, die die Verwendung von Kryptografie einschränken. Über den Sinn oder Unsinn einer entsprechenden gesetzlichen Beschränkung wird jedoch seit Jahren diskutiert. Ihren Höhepunkt erlebte diese Diskussion in Deutschland 1997, als das Thema von den Medien im Sommerloch dankbar aufgegriffen wurde. Als Anhänger der Pro-Regulierungs-Fraktion outete sich damals etwa Peter Frisch, Präsident des Bundesamts

für Verfassungsschutz (»Wir müssen verschlüsselte Botschaften lesen können«) [Spie97]. Auch der damalige Bundesinnenminister Manfred Kanther zog eine Schlüsselhinterlegung in Erwägung. Zahlreiche Organisationen (z. B. der Chaos Computer Club und der Branchenverband Teletrust) leisteten jedoch erfolgreich Widerstand, und so verschwand das Thema schnell wieder in der Versenkung.

3.6.1 Nachteile einer Krypto-Beschränkung

Zu den Gegnern einer Krypto-Beschränkung gehören Datenschützer, Bürgerrechtler, Verbraucherschützer, Liberale und so gut wie die gesamte Computer- und Telekommunikationsbranche. Hier sind die Argumente, die sie ins Feld führen:

- Die Kryptografie bietet Schutz gegen staatliche Eingriffe. Wenn es nicht erlaubt ist, wirksam zu verschlüsseln, dann sind staatlichen Schnüffeleien Tür und Tor geöffnet.
- Die Kryptografie erschwert Wirtschaftsspionage. Werden keine wirksamen Verschlüsselungsmaßnahmen verwendet, dann ist Wirtschaftsspionage an der Tagesordnung – ein gewaltiger Nachteil für den jeweiligen Standort.
- Es gibt in Deutschland und anderen Staaten verfassungsrechtliche Bedenken gegen Krypto-Beschränkungen. Vorschriften, die dem Staatsbürger die Verwendung von Verschlüsselung verbieten, sind möglicherweise verfassungswidrig. Sie könnten gegen die informationelle Selbstbestimmung, Vertraulichkeit der Kommunikation und die Entfaltungsfreiheit verstoßen.
- Krypto-Gesetze sind leicht zu umgehen. Während sich ein Krimineller kaum durch ein Verbot beeindrucken lässt, müssen sich vor allem kommerzielle Anwender zähneknirschend an die Vorschriften halten, womit die Ziele einer Krypto-Beschränkung ad absurdum geführt würden.
- Krypto-Gesetze schwächen das Vertrauen in Produkte, die auf dem Markt erhältlich sind. Amerikanische Firmen erlitten empfindliche Umsatzeinbußen, weil sie ihre starke Krypto-Software lange Zeit nicht exportieren durften. Eine generelle Krypto-Beschränkung hätte ähnliche Konsequenzen. Schon Vorschriften zur Schlüsselwiederherstellung würden das Vertrauen von Anwendern im In- und Ausland enorm schwächen.
- Was man selbst tut, muss man auch anderen zubilligen. Wenn ein demokratisches Land das Verschlüsseln reglementiert, dann kann man sich nicht beklagen, wenn dies andere Staaten auch tun. Auch solche mit totalitären Regierungen.

3.6.2 Vorteile einer Krypto-Beschränkung

Die Gegenposition vertreten vor allem staatliche Behörden, die für innere Sicherheit zuständig sind. Ihr Plädoyer für eine Krypto-Beschränkung würde etwa so aussehen:

- Die Verbrechensbekämpfung verliert ohne Krypto-Beschränkung ein wirksames Werkzeug. Das Abhören von Telefongesprächen und sonstiger Kommunikation ist eine wirksame Waffe gegen Kriminelle. Es wäre leichtsinnig, diese Waffe kampflos aus der Hand zu geben.
- Das Post- und Fernmeldegeheimnis ist nicht unantastbar. Durch den Einsatz von Kryptografie gehen jedoch auch legale Formen des Abhörens verloren.
- Krypto-Gesetze können zwar umgangen werden, dies geschieht aber meist nicht. Anwender sind schließlich faul, und Kriminelle sind auch faul. In den meisten Fällen sind sie zu faul, um sich eine illegale Software zu besorgen, die ihre Daten verschlüsselt, auch wenn dies noch so einfach ist. Steganografie (das Verstecken von Informationen in unauffälligen Daten) wird aus den gleichen Gründen nicht eingesetzt. Die Polizei lebt von den Fehlern, die Kriminelle machen, und sie hat damit meist Erfolg. Wenn Verschlüsselung daher zum festen Bestandteil von E-Mail- und Textverarbeitungsprogrammen wird, dann verschlüsselt jeder automatisch, ohne überhaupt daran zu denken. Bei einer Krypto-Beschränkung bieten kommerziell erhältliche Programme dagegen keine sichere Verschlüsselung. Es gibt keinen Zweifel, dass die meisten Kriminellen den Fehler machen, in diesem Fall auf eine spezielle Verschlüsselungssoftware zu verzichten.
- Durch Maßnahmen zur Schlüsselwiederherstellung kann trotz sicherer Verfahren staatlicher Missbrauch weitgehend vermieden werden. Dies stellt in den Augen vieler Verfechter einer Krypto-Beschränkung einen gangbaren Kompromiss dar. Wir müssen unserem Staat in vielen Dingen vertrauen, warum trauen wir ihm nicht zu, auf unsere Schlüssel aufzupassen?

Nicht zu bestreiten ist, dass das erste Argument auf dieser Liste etwas für sich hat. Es gibt in der Tat zahlreiche Fälle, in denen eine unverschlüsselte Kommunikationsverbindung einem Kriminellen zum Verhängnis wurde. Hier eine Auswahl:

- *Reemtsma-Entführung*: Das bekannteste Beispiel für eine erfolgreiche polizeiliche Abhöraktion ist die Entführung des Industriellen Jan-Philipp Reemtsma. Der Drahtzieher dieses Verbrechens, Thomas Drach, setzte sich nach der Tat nach Südamerika ab und telefonierte von dort aus regelmäßig mit einem Freund in Deutschland. Die Polizei hörte die Gespräche ab, konnte aber nie den genauen Aufenthaltsort Drachs ermitteln. Dies ging so lange, bis Drach sich irgendwann am Telefon verplapperte und dadurch verriet, dass er sich gerade in Buenos Aires aufhielt. Einige Stunden später saß er hinter Schloss und Riegel.

- *Kunstraub*: In ihrem Buch *Aktenzeichen Kunst* schildern die Autoren Nora und Stefan Koldehoff etwa 20 besonders spektakuläre Diebstähle von Kunstgegenständen [KolKol]. Die Schäden lagen häufig im zweistelligen Millionenbereich. Interessanterweise wurden sechs der beschriebenen Fälle mithilfe abgehörter Telefongespräche aufgeklärt. In einem Fall betätigte sich die Polizei zusätzlich als Mitleser von E-Mails. Dabei gab es ein Problem: Viele der abgefangenen Nachrichten waren verschlüsselt.
- *Mafia*: Abgehörte Handygespräche haben auch schon zahlreiche Mafiosi hinter Gitter gebracht. Der prominenteste von ihnen ist der Mafiaboss Bernardo Provenzano, der den italienischen Fahndern nach 43-jähriger Flucht im April 2006 ins Netz ging. Provenzano hatte aus Angst vor polizeilichen Abhöraktivitäten grundsätzlich kein Telefon benutzt, sondern lediglich über schriftliche Nachrichten kommuniziert. Einer seiner Vertrauten war jedoch weniger vorsichtig und verabredete mit einem Komplizen telefonisch die Versorgung Provenzanos mit frischer Wäsche. Die italienische Polizei hörte mit und konnte den berüchtigten Mafioso umgehend verhaften.
- *Wettskandal*: Ebenfalls ein abgehörtes Telefonat brachte 2006 einen Wettskandal im österreichischen Fußball ins Rollen. Das Landeskriminalamt Hessen belauschte einige Telefongespräche zwischen einem Wettpaten und zwei Verdächtigen und stieß dabei auf Hinweise auf Spielmanipulationen.
- *Fußballskandal*: Und noch einmal Fußball. Auch in Italien gab es 2006 einen handfesten Skandal um Spielmanipulationen, der vor allem Rekordmeister Juventus Turin ins Zwielicht (und in die zweite Liga) brachte. Der Korruptionssumpf kam durch ein von der Staatsanwaltschaft abgehörtes Telefonat ans Licht.
- *Doping-Skandal im Radsport*: 2006 kam einer der größten Skandale der Sportgeschichte ins Rollen. In dessen Mittelpunkt stand der spanische Arzt Eufemiano Fuentes, der Dutzende von Radsportlern mit Doping-Mitteln versorgt haben soll. Eine kurz vor der Tour de France öffentlich gewordene Liste nannte 58 Fahrer, darunter große Teile der Weltelite inklusive dem Deutschen Jan Ullrich. Praktisch alle Tour-Favoriten wurden von der Teilnahme ausgeschlossen. Bei der Aufdeckung des Skandals durch die spanische Polizei spielten abgehörte Telefongespräche und SMS-Nachrichten eine wichtige Rolle.
- *Terroranschlag 2006*: Im August 2006 planten islamistische Terroristen Selbstmordanschläge auf mehrere Flugzeuge mithilfe von Flüssigsprengstoff. Die britische Polizei konnte rechtzeitig einschreiten und verhinderte damit möglicherweise eine Katastrophe. Laut Presseberichten hatten Geheimdienste die Telefon-, Internet- und E-Mail-Aktivitäten der Verdächtigen überwacht und dadurch der Polizei die entscheidenden Tipps gegeben [Focus].

- *Einbrecher*: 2007 verhängte das Frankfurter Landgericht gegen zwei Rumänen mehrjährige Haftstrafen [FR]. Die beiden hatten eine ganze Serie von Einbrüchen im gesamten Bundesgebiet verübt. Dabei hatte das Gaunerpaar kaum Spuren hinterlassen. Die Polizei konnte die beiden Täter am Ende dennoch überführen: In Telefonaten, die abgehört wurden, hatten sie mehrfach Beutestücke erwähnt.
- *Mafia-Mord*: 2007 erschütterte ein sechsfacher Mafiamord in Duisburg die ganze Republik. Den dritten und letzten der mutmaßlichen Täter fasste die Polizei maßgeblich aufgrund eines abgehörten Telefonats [Witte].
- *Callgirl-Affäre*: 2008 musste Eliot Spitzer, der Gouverneur des US-Bundesstaats New York, zurücktreten [Südd10]. Von der Polizei abgehörte Telefonate belegten, dass der als Saubermann bekannte Politiker Kunde von Prostituierten gewesen war (Prostitution ist in New York verboten). Die Mitschnitte ließen sogar den Verdacht aufkommen, dass Spitzer enge Kontakte zu einem Callgirl-Ring unterhielt. Ironischerweise wurde Spitzer damit zum Opfer von Fahndungsmaßnahmen, an deren Zustandekommen er selbst maßgeblich beteiligt gewesen war.
- *Costa Concordia*: Als am 13. Januar 2012 das Kreuzfahrtschiff Costa Concordia in der Nähe von Rom verunglückte (es gab 17 Todesopfer), geriet der Kapitän in die Kritik. Ein aufgezeichnetes Telefonat belegte, dass er das Schiff verbotenerweise verlassen hatte, während die Evakuierung noch im Gange war [FTD]. Zunächst hatte er das bestritten.
- *Strauss-Kahn*: Im Frühjahr 2012 erhob die französische Staatsanwaltschaft Anklage wegen schwerer Zuhälterei gegen den früheren IWF-Chef Dominique Strauss-Kahn. Zu den belastenden Indizien gehören mehrere Hundert abgehörte SMS [Veiel].

Diese keineswegs vollständige Aufstellung zeigt, dass das Überwachen von Telefongesprächen und Internetnachrichten für die Polizei eine wirksame Waffe ist. Auch die immer wieder vorgebrachte Behauptung, es träfe nur die kleinen Fische, ist falsch.

3.6.3 Fazit

Obwohl ich gerne zugebe, dass ich ein Gegner der gesetzlichen Krypto-Beschränkung bin, halte ich die Argumente der Befürworter nicht für völlig absurd. Vor allem das Argument, Krypto-Verbote seien sinnlos, weil sie umgangen werden können, lassen die Befürworter zu Recht nicht gelten. Schließlich heißt die Tatsache, dass etwas leicht umgangen werden kann, noch lange nicht, dass dies von allen gemacht wird. Eine Krypto-Regulierung sorgt nun einmal dafür, dass alle Standardprogramme keine wirksame Kryptografie einsetzen, und solange dies nicht der Fall ist, wird sich nur eine Minderheit die Mühe machen, sie zu umgehen.

Wie sehr sich die Regierungen mancher Länder gegen den Einsatz von Kryptografie sträuben und welche Tricks sie anwenden, ist ein beeindruckendes Indiz dafür, dass von staatlicher Seite in großem Stil gelauscht wird. Der Große Bruder von heute wirft seine Blicke nicht direkt auf uns; er beobachtet lediglich, was wir über das Netz von uns geben und was uns zugeschickt wird. Der gläserne Mensch ist out, der vernetzte Mensch ist in. Wenn Sie sich dagegen wehren wollen, dann bleibt Ihnen nur der Einsatz von Kryptografie. Bleibt zu hoffen, dass es keine Gesetze geben wird, die dies verhindern sollen.

4 Symmetrische Verschlüsselung

Als Nächstes wollen wir einige einfache **Verschlüsselungsverfahren** betrachten, mit denen Alice und Bob ihre Nachrichten vor Bösewicht Mallory verbergen können. Alle in diesem Kapitel beschriebenen Methoden sind schon recht alt und für heutige Ansprüche meist nicht mehr geeignet. Sie bilden jedoch ein wichtiges Fundament für moderne Verschlüsselungsverfahren, von denen in diesem Buch noch ausführlich die Rede sein wird.

4.1 Symmetrische Verschlüsselung

Wie man sich leicht klarmacht, ist es sinnvoll, wenn Alice und Bob ein Verschlüsselungsverfahren (man spricht auch von einem **Verschlüsselungsalgorithmus** oder einer **Chiffre**) einsetzen, in das eine Geheiminformation (der sogenannte **Schlüssel**) mit einfließt. Je nach Verfahren ist der Schlüssel ein Passwort, eine Geheimnummer oder einfach nur eine Folge von Bits. Bei einem guten Verschlüsselungs-

verfahren ist es für Mallory unmöglich, eine Nachricht ohne Kenntnis des Schlüssels zu entschlüsseln – selbst wenn er das Verfahren genau kennt. Für Alice und Bob muss es bei Kenntnis des Schlüssels dagegen einfach sein, einen Verschlüsselungsvorgang rückgängig zu machen.

Da bei einem Verschlüsselungsverfahren die Sicherheit nur vom Schlüssel abhängen sollte, wird die Funktionsweise eines solchen in der Regel nicht geheim gehalten. Es gilt sogar als gute Strategie, gerade den entgegengesetzten Weg zu gehen und ein Verschlüsselungsverfahren so bekannt wie möglich zu machen. Erst wenn genügend Experten sich damit beschäftigt haben und wenn es erfolglos auf alle denkbaren Schwächen abgeklopft wurde, kann man davon ausgehen, dass auch Bösewicht Mallory keine Chance hat.

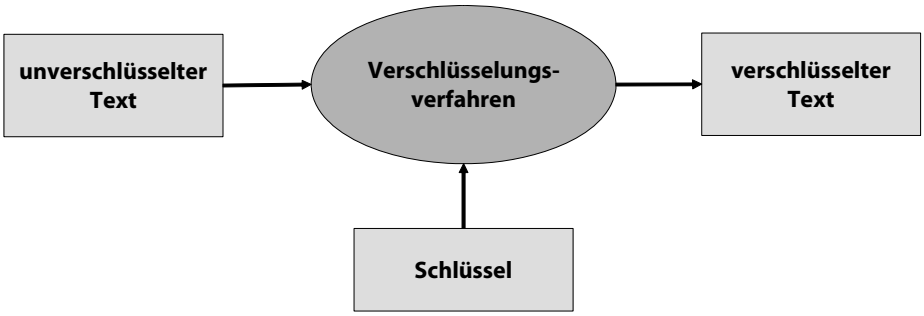


Abb. 4–1 *Bei einem guten Verschlüsselungsverfahren geht ein Schlüssel (eine Art Passwort) in die Verschlüsselung ein.*

In der Praxis ist es ohnehin oft schwierig bis unmöglich, ein Verschlüsselungsverfahren geheim zu halten. Vor allem dann, wenn ein Verfahren in einer verbreiteten Software implementiert ist, wird sich früher oder später jemand die Mühe machen, den Quellcode zu analysieren. Sie werden in diesem Buch mehrere Verfahren kennenlernen (z. B. A5, Crypto1, RC2 und RC4), die ursprünglich geheim waren, irgendwann jedoch durch ein Informationsleck öffentlich bekannt wurden. Wenn die Sicherheit eines Verfahrens von der Geheimhaltung abhängt, dann bezeichnet man das als **Security by Obscurity**, und das gilt unter Kryptografen nahezu als Schimpfwort. Das heißt jedoch nicht zwangsläufig, dass es immer sinnvoll ist, die Funktionsweise eines Verschlüsselungsverfahrens bekannt zu machen. Militär und Geheimdienste verzichten meist darauf. Es gilt aber: Falls die Funktionsweise irgendwann doch bekannt wird, muss die Sicherheit dennoch gewährleistet bleiben.

Die Idee, dass allein die Geheimhaltung des Schlüssels die Sicherheit eines Verfahrens garantieren muss, wurde Ende des 19. Jahrhunderts erstmals von dem Niederländer Auguste Kerckhoffs formuliert. Man spricht deshalb auch vom **Kerckhoffs'schen Prinzip**. Der Computer-Pionier Claude Shannon griff dieses später auf und formulierte es wie folgt: »The enemy knows the system.« Es ver-

steht sich von selbst, dass das Kerckhoffs'sche Prinzip auch in diesem Buch gilt. Wir gehen also davon aus, dass Mallory die von Alice und Bob verwendeten Verfahren genau kennt.

4.1.1 Kryptografische Fachbegriffe

Verschlüsselungsverfahren, die gemäß der beschriebenen Weise mit geheimen Schlüsseln arbeiten, nennt man auch **symmetrische Verfahren** oder **Secret-Key-Verfahren.** Später werden Sie Verschlüsselungsverfahren kennenlernen, bei denen ein Teil des Schlüsselmaterials nicht geheim ist und die als asymmetrische Verfahren oder Public-Key-Verfahren bezeichnet werden. Man spricht in diesem Zusammenhang auch von **symmetrischer Kryptografie** und **asymmetrischer Kryptografie** oder von **Public-Key-Kryptografie** und **Secret-Key-Kryptografie**. Verbreitet sind auch Ausdrücke wie symmetrische Verschlüsselung, asymmetrische Verschlüsselung, Secret-Key-Verschlüsselung oder Public-Key-Verschlüsselung.

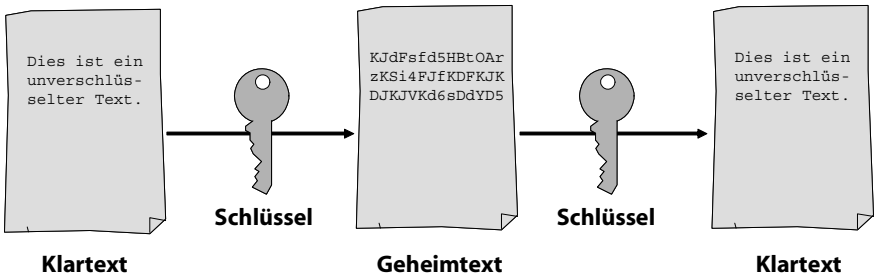


Abb. 4–2 *Eine Nachricht, die verschlüsselt wird, heißt Klartext. Das Resultat der Verschlüsselung wird als Geheimtext bezeichnet.*

Eine Nachricht, die verschlüsselt wird, heißt **Klartext**, der verschlüsselte Text **Geheimtext** (auch wenn es sich nicht um einen Text im eigentlichen Sinne handeln muss). In der Sprache der Mathematik sind Verschlüsselung und Entschlüsselung Funktionen, für die wir die Buchstaben *e* (encrypt) bzw. *d* (decrypt) verwenden. Ist *k* der Schlüssel, *m* der Klartext und *c* der Geheimtext, dann gelten folgende zwei Formeln:

$$e(m,k)=c$$
$$d(c,k)=m$$

4.1.2 Angriffe auf Verschlüsselungsverfahren

Versucht Mallory, einen Geheimtext zu entschlüsseln, den Alice an Bob schickt, oder den Schlüssel herauszufinden, so nennt man dies einen **Angriff** oder eine **Attacke**. Die Kryptoanalyse ist somit die Wissenschaft der Angriffe auf Verschlüs-

selungsverfahren. Man sagt auch, Mallory versucht das Verfahren zu **brechen** oder zu **knacken**. Angriffe auf Verschlüsselungsverfahren kann man in verschiedene Gruppen einteilen:

- Kennt Mallory den Klartext nicht, dann spricht man von einer **Ciphertext-Only-Attacke**.
- Kennt Mallory den Klartext und versucht, für das Entschlüsseln weiterer Nachrichten den Schlüssel zu erfahren, so nennt man dies eine **Known-Plaintext-Attacke**.
- Will Mallory den Schlüssel herausfinden und hat dabei die Möglichkeit, den Klartext selber zu wählen, so spricht man von einer **Chosen-Plaintext-Attacke**.

Alle drei Angriffsformen setzen voraus, dass Mallory das Verfahren kennt (es gilt ja das Kerckhoffs'sche Prinzip). Angriffe auf unbekannte Verfahren spielen in der Kryptografie-Literatur so gut wie keine Rolle. Von den drei genannten Angriffsvarianten ist die Ciphertext-Only-Attacke bei weitem die schwierigste. Gleichzeitig ist sie jedoch auch diejenige, die in der Praxis am häufigsten vorkommt, denn im Normalfall weiß Mallory ja nicht, was in der abgefangenen Nachricht steht. Trotzdem spielen auch die beiden anderen Angriffe eine wichtige Rolle. Eine Known-Plaintext-Attacke ist häufig dann möglich, wenn sich Nachrichten oder Teile davon wiederholen. Wenn Alice für ihre E-Mails beispielsweise stets den gleichen Briefkopf verwendet oder alle ihre Mails mit den Worten »Hallo Bob« beginnt, dann kennt Mallory, sofern er dies weiß, zumindest Teile des Klartexts. Mit einer Known-Plaintext-Attacke kann er nun versuchen, den Schlüssel herauszufinden, um so auch den Rest der E-Mail zu entschlüsseln.

Die Chosen-Plaintext-Attacke ist der einfachste Angriff. Bei vielen traditionellen Verschlüsselungsverfahren ist er ein Kinderspiel – bei modernen zum Glück nicht. Eine Chosen-Plaintext-Attacke kann Mallory beispielsweise am Entschlüsselungschip eines Pay-TV-Decoders starten. Der Schlüssel ist in diesem Fall in einer Hardware (Smartcard) gespeichert und damit nicht ohne weiteres auslesbar. Da Mallory jedoch zum eigenen Decoder unbeschränkten Zugang hat, kann er diesen mit selbst gewähltem Klartext füttern. Größere Hoffnungen sollte sich Mallory jedoch nicht machen: Die im Bezahlfernsehen verwendeten Verfahren sind gegenüber einer Chosen-Plaintext-Attacke nur bedingt anfällig.

4.2 Monoalphabetische Substitutionschiffren

Nach all den theoretischen Vorüberlegungen wollen wir uns nun einige Verschlüsselungsverfahren anschauen. Wir beginnen mit sehr einfachen Algorithmen und arbeiten uns dann zu komplexeren Methoden vor.

4.2.1 Cäsar-Chiffre

Da Alice und Bob sich hauptsächlich Texte zuschicken, bietet sich zunächst ein Verschlüsselungsverfahren an, das jeden Buchstaben durch einen anderen ersetzt. Beispielsweise kann jeder Buchstabe um eine Zahl *n* verschoben werden:

Ist *n*=1, dann gilt: aus `A` wird `B`, aus `B` wird `C`, aus `C` wird `D`, ...

Ist *n*=2, dann gilt: aus `A` wird `C`, aus `B` wird `D`, aus `C` wird `E`, ...

Der Klartext

```
DER MENSCH MACHT FEHLER, FUER KATASTROPHEN IST DER COMPUTER ZUSTAENDIG.
```

wird bei *n=5* zu folgendem Geheimtext:

```
IJW RJSXHM RFHMY KJMQJW, KZJW PFYFXYWTUMJS NXY IJW HTRUZYJW EZXYFJSINL.
```

Sie haben es sicher erraten, *n* ist der Schlüssel bei diesem Verfahren, das man als **Cäsar-Chiffre** bezeichnet (weil es schon Julius Cäsar bekannt war). Die Cäsar-Chiffre kennt 26 verschiedene Schlüssel (den Schlüssel 26 sollten Alice und Bob aus naheliegenden Gründen jedoch nicht verwenden). Die Cäsar-Chiffre gehört zu den zahlreichen Verfahren, die Buchstaben oder Buchstabengruppen durch andere ersetzen. Ein Verfahren mit dieser Eigenschaft wird **Substitutionschiffre** genannt. Um die Cäsar-Chiffre einzusetzen, kann man eine **Chiffrierscheibe** oder einen **Chiffrierschieber** verwenden. Beide Verschlüsselungswerkzeuge haben eine lange Geschichte und wurden in vielen Varianten gebaut.

Ein Spezialfall der Cäsar-Chiffre ist **ROT-13**. Bei diesem Verfahren wird um 13 Buchstaben verschoben. Aus KRYPTO wird also XELCGB. Wie Sie sich leicht klarmachen, funktioniert das Entschlüsseln bei ROT-13 genauso wie das Verschlüsseln (diese Eigenschaft nennt man **involutorisch**). Einen wichtigen Einsatzbereich hat ROT-13 momentan im Geocaching (dies ist eine moderne Variante der Schnitzeljagd, bei der GPS-Empfänger verwendet werden). Auf Geocaching-Webseiten werden Hinweise, die das Auffinden eines Verstecks erleichtern, mit ROT-13 verschlüsselt. Dabei geht es weniger um Geheimhaltung als darum, ein unbeabsichtigtes Lesen der Hinweise zu verhindern.

Die Cäsar-Chiffre ist alles andere als sicher. Selbst eine Ciphertext-Only-Attacke ist möglich. Von Hand oder mit Computerunterstützung muss Mallory dazu die Zahlen 1 bis 25 durchprobieren und kann so den Schlüssel schnell ermitteln. Einen solchen Angriff, bei dem alle möglichen Schlüssel durchprobiert werden, nennt man **vollständige Schlüsselsuche** oder auch **Brute-Force-Attacke**. Eine andere wirksame Attacke auf die Cäsar-Chiffre basiert auf der sogenannten Häufigkeitsanalyse, die vor allem bei längeren Texten hervorragend funktioniert. Die Häufigkeitsanalyse beruht auf der einfachen Tatsache, dass in einem Text normalerweise nicht alle Buchstaben gleich häufig vorkommen. In der deutschen Sprache ist das `E` mit über 17 Prozent der häufigste Buchstabe (siehe Abbildung 4–3), gefolgt vom `N` (10 Prozent) und dem `R` (7 Prozent). Dies kann Mallory ausnutzen,

indem er beim obigen Beispiel-Geheimtext das Vorkommen der einzelnen Buchstaben zählt. Es ergeben sich folgende Häufigkeiten:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3 1 0 0 1 4 0 3 3 9 2 4 2 0 1 1 3 3 2 2 0 6 4 5 0 0
```

Sie sehen, das `J` ist mit 9 Vorkommnissen der häufigste Buchstabe des Geheimtexts. Tatsächlich ist es das `J`, auf das das `E` abgebildet wurde. Dass der zweithäufigste Buchstabe das `V` ist, obwohl dieses nicht für das `N` steht, gehört zu den Widrigkeiten im Leben eines Kryptoanalytikers. Bei längeren Texten kommt so etwas in der Regel nicht vor.

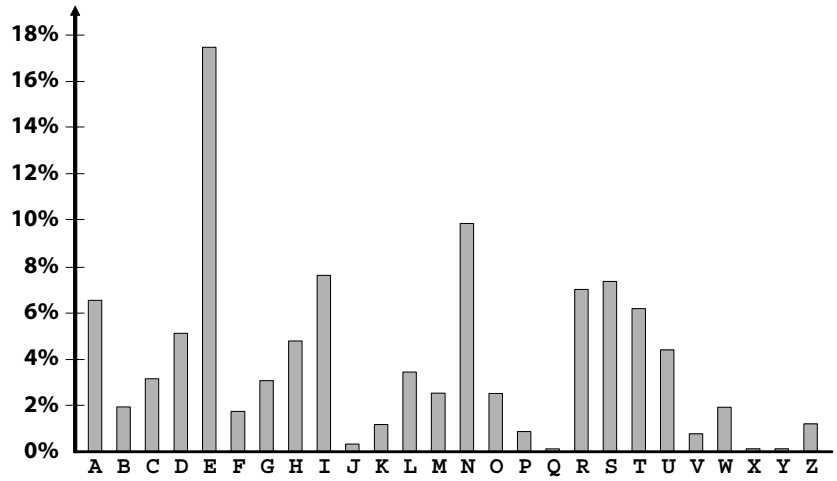


Abb. 4–3 *Die Buchstaben in der deutschen Sprache sind ungleich verteilt. Das E ist mit Abstand der häufigste Buchstabe. Dies kann sich Mallory beim Knacken einer Verschlüsselung zunutze machen.*

4.2.2 Freie Buchstabensubstitution

Anstatt jeden Buchstaben im Alphabet zu verschieben, können Alice und Bob auch eine Tabelle aufstellen, in der jeder Buchstabe in der oberen Zeile auf den darunter stehenden abgebildet wird. Das Folgende ist ein Beispiel:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
N E U Z Y O V D K T M F J R L B G H A C P W S Q I X
```

Der Satz GEBEN IST SELIGER DENN NEHMEN verschlüsselt sich hierbei in VYEYR KHC AYFKVYH ZYRR RYDJYR. Im Gegensatz zur Cäsar-Chiffre hat dieses Verfahren keinen gängigen Namen. Man kann es als **freie Buchstabensub-**

stitution bezeichnen. Die freie Buchstabensubstitution ist eine Verallgemeinerung der Cäsar-Chiffre. Eine vollständige Schlüsselsuche ist in diesem Fall nicht praktikabel, da es 26! und damit etwa $4 \cdot 10^{26}$ mögliche Schlüssel gibt. Besonders sicher ist das Verfahren dennoch nicht, denn auch hier kommt Mallory mit einer Häufigkeitsanalyse zum Ziel. Allerdings genügt es dieses Mal nicht, nur den häufigsten Buchstaben im Geheimtext zu bestimmen. Stattdessen muss Mallory für mehrere (am besten alle) die relative Häufigkeit berechnen. Hat er dies erst einmal gemacht und ist der Geheimtext lang genug, dann hat er mithilfe einer Statistik wie in Abbildung 4–3 keine Probleme mehr, den Klartext zu bestimmen. Die freie Buchstabensubstitution zeigt, dass eine große Anzahl möglicher Schlüssel noch lange keine Sicherheit garantiert.

Alice und Bob können Mallory die Arbeit noch etwas erschweren, indem sie nicht von 26 Buchstaben ausgehen, sondern beispielsweise von 256 ASCII-Zeichen. In diesem Fall sind Wortzwischenräume und Satzzeichen für Mallory nicht mehr als solche zu erkennen. Zudem ist es aufwendiger, eine Tabelle wie in Abbildung 4–3 für 256 Zeichen aufzustellen. Einen großen Sicherheitsgewinn können Alice und Bob so allerdings kaum erzielen. Darüber hinaus ist zu berücksichtigen, dass wir bisher nur über eine Ciphertext-Only-Attacke geredet haben. Eine Known-Plaintext- und erst recht eine Chosen-Plaintext-Attacke ist bei einer freien Buchstabensubstitution trivial.

4.2.3 Homophone Chiffre

Da die unterschiedlichen Buchstabenhäufigkeiten bei den bisher betrachteten Verfahren den entscheidenden Schwachpunkt bilden, liegt es nahe, mehrere Geheimtextzeichen für dasselbe Klartextzeichen bereitzustellen. Beispielsweise können Alice und Bob für das E abwechselnd die Zeichen D, z, 0 und $ verwenden. Wenn Mallory anschließend eine Häufigkeitsanalyse vornimmt, wird er das E kaum finden. Geheimtextzeichen, die für dasselbe Klartextzeichen stehen, nennt man **Homophone**. Ein Verfahren, bei dem Homophone zum Einsatz kommen, heißt **homophone Chiffre**.

Im Folgenden nehmen wir an, dass für den Geheimtext die Zahlen zwischen 00 und 99 zur Verfügung stehen. Am wirkungsvollsten ist eine homophone Chiffre, wenn Alice und Bob jedem Buchstaben so viele Zahlen zuweisen, wie es seiner prozentualen Häufigkeit im Klartext entspricht. Im Deutschen erhält demnach das E (17 Prozent Häufigkeit) 17 Zahlen, während dem N 10 und dem I 8 Zahlen zugeordnet sind. Die folgende Tabelle beschreibt ein Beispiel (unter den Buchstaben steht jeweils die Anzahl der zugeordneten Zahlen, wobei ich teilweise gerundet habe, damit es aufgeht).

A	B	C	D	E	F	G	H	I	J	K	L	M
6	2	3	5	17	1	3	4	8	1	1	3	2
34,45, 54,72, 79,86	14,71	04,28, 61	07,39, 47,87, 99	11,20, 24,32, 42,48, 49,53, 60,65, 70,78, 81,84, 91,96, 98	15	27,38, 63	21,43, 59,88	00,08, 22,31, 35,64, 76,89	66	01	44,52, 75	12,80

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
10	2	1	1	7	7	6	4	1	2	1	1	1
10,29, 40,41, 62,74, 83,85, 90,92	30,58	03	37	19,33, 51,67, 82,93, 97	05,16, 25,36, 46,55, 68	26,50, 56,69, 77,94	02,23, 57,73	17	06,18	95	09	13

.

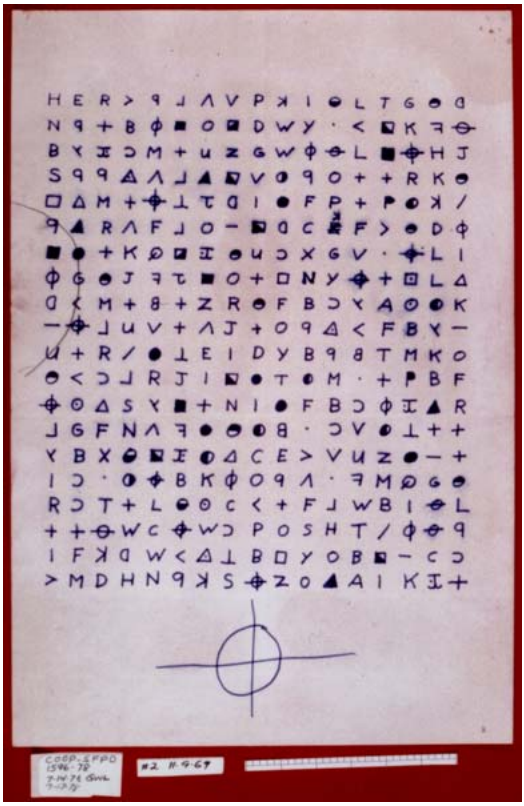


Abb. 4–4 *Der Zodiac-Killer schickte vier verschlüsselte Nachrichten an verschiedene Zeitungen, von denen nur eine gelöst wurde. Die Abbildung zeigt eine der ungelösten Nachrichten.*

Der Klartext EBENE lässt sich so in den Geheimtext 20 71 48 29 98 verschlüsseln. Eine homophone Chiffre ist nicht leicht zu knacken. Völlig sicher ist sie dennoch nicht. Wenn Mallory ein paar Wörter oder Buchstaben des Texts erraten kann, dann reicht dies möglicherweise aus, um weitere Buchstaben zu ergänzen und so schließlich den gesamten Text zu rekonstruieren. Wie so etwas funktionieren kann, zeigt das Beispiel des sogenannten Zodiac-Killers [Schm12/1]. Dabei handelt es sich um einen Serienmörder, der Ende der sechziger Jahre in Kalifornien sein Unwesen trieb. Der bis heute nicht identifizierte Täter schrieb insgesamt vier verschlüsselte Nachrichten an verschiedene Zeitungen, von denen bisher nur eine geknackt wurde. Die erfolgreiche Kryptoanalyse gelang einem rätselbegeisterten Ehepaar, das die Nachricht aus der Presse kannte. Die beiden vermuteten, dass der Verfasser eine homophone Chiffre verwendet hatte, und sie nahmen an, dass das erste Wort im Text »I« (ich) lautete. Dieser Ansatz erwies sich als Volltreffer und reichte, um den gesamten Klartext zu ermitteln. Leider enthielt der entschlüsselte Text keine Informationen, die zum Täter führten. Die drei weiteren verschlüsselten Nachrichten des Zodiac-Killers sind nach wie vor ungelöst. Es ist nicht bekannt, ob der Täter auch hier eine homophone Chiffre verwendete.

Kann Mallory eine Known-Plaintext-Attacke anwenden, dann ist das Knacken einer homophonen Verschlüsselung trivial. Wenn Alice und Bob befürchten, dass Mallory in den Besitz eines Klartexts kommt, dann sollten sie daher für jede Datenübertragung einen neuen Schlüssel (also eine neue Tabelle) vereinbaren. Dies macht das ohnehin unhandliche Verfahren noch unhandlicher. Die homophone Chiffre wurde daher bereits in der Vor-Computer-Ära selten verwendet und ist seit Aufkommen des Computers nur noch historisch interessant.

4.2.4 Bigramm-Substitution

Anstatt immer nur einen Buchstaben auf einmal zu ersetzen, können Alice und Bob auch mit Buchstabenpaaren (Bigrammen) arbeiten. Diese Vorgehensweise bezeichnet man als **Bigramm-Substitution**. Ein Beispiel ist durch folgende Tabelle gegeben (es handelt sich dabei nur um einen Auszug, eigentlich ist die Tabelle viel größer):

AA	...	AL	...	DE	...	LI	...	ND	...	RB	...	RU	...	ST	...	ZZ
JQ		CV		UD		PC		MY		FH		CC		SL		XC

Nehmen wir nun den Satz DER BALL IST RUND. In Buchstabenpaare aufgeteilt lautet dieser: DE RB AL LI ST RU ND. Ersetzen wir jedes Paar gemäß der Tabelle, dann erhalten wir folgenden Geheimtext: UD FH CV PC SL CC MY. Eine Ciphertext-Only-Attacke auf eine Bigramm-Substitution ist für Mallory nicht ganz einfach. Möglich ist sie dennoch, denn auch bei Buchstabenpaaren funktioniert eine Häufigkeitsanalyse. Im Deutschen ist beispielsweise EN vor ER

und CH das häufigste Bigramm. Allerdings braucht Mallory hier schon einen Text von mehreren Hundert Buchstaben, um zum Erfolg zu kommen.

Ein Nachteil der Bigramm-Substitution ist offensichtlich: Als Schlüssel benötigen Alice und Bob eine Tabelle mit $26 \times 26 = 676$ Einträgen. Dies ist äußerst unhandlich, zumal die beiden aus Sicherheitsgründen für jede Nachricht eine neue Tabelle verwenden sollten. Aus diesem Grund hatte die Bigramm-Substitution nie eine praktische Bedeutung. Es gibt jedoch eine Alternative: Anstatt eine Tabelle zu nutzen, können Alice und Bob auch ein handlicheres Verfahren für das Ersetzen von Bigrammen verwenden. Das mit Abstand bekannteste davon ist die **Playfair-Chiffre**. Diese ist nach Lyon Playfair (1818–1898) benannt.

Wie die Playfair-Chiffre funktioniert, will ich an folgendem Klartext zeigen: BALM OF HURT MINDS GREAT NATURE'S SECOND COURSE CHIEF NOURISHER IN LIFE'S FEAST. Wie Sie in Abschnitt 4.5.2 sehen werden, spielt dieser Text im Zusammenhang mit einem ungelösten Krypto-Rätsel eine wichtige Rolle. Da bei Playfair kein Bigramm aus zwei gleichen Buchstaben bestehen darf, muss Absenderin Alice zwischen NATURE'S und SECOND den Buchstaben X einfügen. Außerdem benötigt sie ein Schlüsselwort. Dieses laute SURPRISE. Damit erstellt Alice folgende Tabelle (sie lässt das J weg, damit eine 5×5-Matrix entsteht), in der das Schlüsselwort (ohne sich wiederholende Buchstaben) gefolgt von den restlichen Buchstaben des Alphabets steht:

```
S U R P I
E A B C D
F G H K L
M N O Q T
V W X Y Z
```

Nun gilt für die Buchstabenpaare BA, LM, OF, HU usw. jeweils folgende Regel: Die Buchstaben eines Paares werden durch die beiden entgegengesetzten Buchstaben im Rechteck, das sie bilden, ersetzt. Aus LM wird so FT. Stehen die beiden Buchstaben in einer Reihe, dann werden sie durch die jeweils rechts davon stehenden ersetzt. Aus BA wird dadurch CB. So entsteht folgender Geheimtext: CBFTM HGRIO TSTAU FSBDN WGNIS BRVEF BQTAB QRPEF BKSDG MNRPS RFBSU TTDMF EMA BIM. Eine Variante des Playfair-Verfahrens, der sogenannte Doppelkasten, wurde noch im Zweiten Weltkrieg von den Deutschen eingesetzt.

4.2.5 Wörter-Codes und Nomenklatoren

Anstatt Buchstaben oder Buchstabenpaare zu ersetzen, können es Alice und Bob auch mit ganzen Wörtern versuchen. Man bezeichnet dies als **Wörter-Code**. Um einen solchen zu nutzen, benötigen Alice und Bob ein **Codebuch**, das den Schlüssel des Verfahrens bildet. Ein Codebuch listet zu jedem Wort einer Sprache das zugehörige Codewort auf. Dies kann etwa so aussehen:

```
...
Sommer 387831
Sonate 918108
Sonne 173634
Sonntag 928320
...
```

Ein Codebuch ähnelt somit einem Wörterbuch. In der Praxis wurden meist Verfahren eingesetzt, die einen Wörter-Code mit einer freien Buchstaben-Substitution kombinieren. Will Alice hierbei ein Wort verschlüsseln, dann sucht sie es im Codebuch. Steht es nicht drin (dies kommt beispielsweise bei Eigennamen und Ortsnamen häufig vor), dann verschlüsselt sie es buchstabenweise. Eine solche Kombination aus Wörter-Code und Buchstaben-Substitution heißt **Nomenklator**.

Wörter-Codes und Nomenklatoren haben in der Kryptologie-Geschichte eine ausgesprochen wichtige Rolle gespielt. Die ersten Codebücher gab es bereits im 14. Jahrhundert [Schm12/1]. Bis in den Zweiten Weltkrieg hinein zählten sie zu den gängigsten Verfahren. Der Umgang mit Codebüchern war zwar recht umständlich (Sender und Empfänger mussten stets das gleiche besitzen), doch dafür war das Verschlüsseln damit einfach und bot eine vergleichsweise hohe Sicherheit. Auf hoher militärischer Ebene und in der Diplomatie – also da, wo die Sicherheitsansprüche besonders hoch sind – waren Codebücher sogar die wichtigste Verschlüsselungstechnik überhaupt.

4.3 Polyalphabetische Substitutionschiffren

Die bisher betrachteten Verfahren haben alle die Eigenschaft, dass beim Entschlüsseln ein bestimmter Geheimtextbuchstabe (bzw. eine Geheimtextwort) immer auf den gleichen Klartextbuchstaben (bzw. auf das gleiche Klartextwort) abgebildet wird. Wenn Mallory im Geheimtext mehrfach den Buchstaben G findet, dann weiß er, dass sich jedes Mal der gleiche Buchstabe dahinter verbirgt. Eine Chiffre mit dieser Eigenschaft nennt man **monoalphabetisch**. In diesem Unterkapitel geht es um Verfahren, die nicht monoalphabetisch und daher **polyalphabetisch** sind.

4.3.1 Vigenère-Chiffre

Das bekannteste polyalphabetische Verschlüsselungsverfahren ist die **Vigenère-Chiffre**. Diese ist nach dem französischen Kryptografen Blaise de Vigenère (1523–1596) benannt. Um deren Funktionsweise zu verstehen, nehmen wir an, dass Alice folgenden Klartext verschlüsseln will:

```
ALLES IST EINE FOLGE VON BITS.
```

Der Schlüssel ist in diesem Fall ein Wort, zum Beispiel `ALICE`. Zur Verschlüsselung schreibt Alice Klartext und Schlüssel folgendermaßen untereinander:

```
ALLES IST EINE FOLGE VON BITS.
ALICE ALI CEAL ICEAL ICE ALIC.
```

Anschließend addiert Alice die Buchstaben spaltenweise (A=0, B=1, C=2, ...; nach Z fängt sie wieder bei A an):

```
ALLES IST EINE FOLGE VON BITS.
ALICE ALI CEAL ICEAL ICE ALIC.
------------------------------
AWTGW IDB GMNP NQPGP DQR BTBU.
```

Der Geheimtext lautet also AWTGW... Wie Sie sehen, kann der gleiche Geheimtext-Buchstabe hier für verschiedene Klartext-Buchstaben stehen, wie zum Beispiel das erste W für ein L und das zweite W für ein S. Die Vigenère-Chiffre ist daher polyalphabetisch. Dank dieser Eigenschaft ist die Vigenère-Chiffre wesentlich sicherer als die Cäsar-Chiffre, zumal eine vollständige Schlüsselsuche hier schon sehr aufwendig ist. Trotzdem kann auch die Vigenère-Chiffre schon mit einer Ciphertext-Only-Attacke leicht gebrochen werden. Der entscheidende Punkt bei diesem Angriff ist die Länge des Schlüssels. Ist Mallory diese bekannt, dann besteht sein Problem nur noch darin, mehrere Cäsar-Chiffren zu brechen, und das ist nicht besonders schwierig. Um die Schlüssellänge zu ermitteln, genügt es oft schon, den Geheimtext auf Buchstabenfolgen abzusuchen, die sich wiederholen. Kommt beispielsweise im Geheimtext die Buchstabenkombination BJHG zweimal vor und beträgt der Abstand 56, so ist dies ein Indiz dafür, dass 56 durch die Schlüssellänge teilbar ist. Findet Mallory ein weiteres Muster, das doppelt vorkommt, mit dem Abstand 105, so ist der Fall schon klar: Die Teiler von 56 sind 2, 4, 7, 8, 14 und 28, die Teiler von 105 sind 3, 5 und 7. Da 7 als einzige Zahl sowohl 105 als auch 56 teilt, ist dies mit großer Wahrscheinlichkeit die Schlüssellänge.

Es gibt noch andere Methoden, mit denen Mallory die Schlüssellänge ermitteln kann, doch diese wollen wir hier überspringen. Klar ist jedenfalls Folgendes: Eine Ciphertext-Only-Attacke mit Computerunterstützung ist auch in diesem Fall kein Problem, und Mallory hat bei bekanntem oder gar frei wählbarem Klartext erst recht keine Mühe.

4.3.2 Vernam-Chiffre

Obwohl die Vigenère-Chiffre genauso alt wie unsicher ist, ist sie keineswegs nutzlos. Sie kann sogar zu einer sehr sicheren Chiffre ausgebaut werden, wenn Alice und Bob den richtigen Schlüssel verwenden. Damit es Mallory so schwer wie möglich gemacht wird, sollten Alice und Bob den Schlüssel möglichst lang wählen. Je länger der Schlüssel nämlich ist, desto mehr Cäsar-Chiffren muss Mallory brechen und desto weniger Text steht diesem für jede Cäsar-Chiffre zur Verfügung. Idealerweise wählen Alice und Bob den Schlüssel einer Vigenère-Chiffre

sogar so lang wie den Klartext. Diesen Spezialfall nennt man dann **Vernam-Chiffre** (benannt nach ihrem Erfinder Gilbert Vernam). Die Vernam-Chiffre ist mit einer einfachen Häufigkeitsanalyse oder mit einer vollständigen Schlüsselsuche nicht zu brechen. Wirklich sicher ist sie aber trotzdem nicht. Falls der Geheimtext – und damit auch der Schlüssel – lang genug sind und beide aus einer natürlichen Sprache (zum Beispiel Deutsch) stammen, kann sich Mallory zunutze machen, dass Buchstaben im Klartext und Schlüssel nicht mit der gleichen Häufigkeit auftreten und dass damit auch im Geheimtext eine ungleiche Verteilung vorliegt. Mallory kann also wiederum eine Häufigkeitsanalyse einsetzen. Natürlich ist diese wesentlich komplizierter als bei einer einfachen Substitutionschiffre, für geübte Kryptoanalytiker mit Computerunterstützung vom Schlage eines Mallory ist so etwas jedoch nur eine Fleißaufgabe.

4.3.3 One-Time-Pad

Wenn Mallory die Vernam-Chiffre auf die beschriebene Weise brechen will, dann müssen Schlüssel und Klartext jeweils ungleichmäßige Buchstabenverteilungen besitzen. Wählen Alice und Bob dagegen als Schlüssel eine rein zufällige Buchstabenfolge, dann hat eine Häufigkeitsanalyse keinen Erfolg mehr. Eine Vernam-Chiffre, bei der der Schlüssel eine solche Zufallsfolge ist, nennt man **One-Time-Pad**. Wie der Name sagt, wird beim One-Time-Pad der Schlüssel (der wiederum die gleiche Länge hat wie der Klartext) nur einmal verwendet, ansonsten wäre die Buchstabenfolge ja nicht mehr zufällig.

Gibt es nun überhaupt eine Methode der Kryptoanalyse, die gegen den One-Time-Pad Erfolg verspricht? Interessanterweise nicht, zumindest dann, wenn der Schlüssel wirklich zufällig ist (was unter zufällig zu verstehen ist, lesen Sie in Kapitel 15). Dann kann sogar bewiesen werden, dass der Geheimtext ebenfalls vollkommen zufällig ist und damit nicht gebrochen werden kann. Oder mit anderen Worten: Jeder mögliche Klartext kann in jeden möglichen Geheimtext verschlüsselt werden, und das mit jeweils gleicher Wahrscheinlichkeit – da hat Mallory ausgespielt. Der One-Time-Pad in seinen verschiedenen Formen ist übrigens das einzige Verschlüsselungsverfahren, für das diese Eigenschaft gilt. Es ist absolut sicher.

Der One-Time-Pad funktioniert nicht nur mit den 26 Buchstaben des Alphabets. Genauso gut können auch die 256 ASCII-Zeichen oder eine andere Menge von Zeichen verwendet werden. In der modernen Kryptografie werden nur die zwei Zahlen 0 und 1 verwendet. Der Klartext ist in diesem Fall eine Bit-Folge, der Schlüssel ebenfalls. Die Addition von einem Klartext-Bit mit einem Schlüssel-Bit entspricht dabei einer sogenannten **Exklusiv-oder-Verknüpfung**. Eine Exklusiv-oder-Verknüpfung liefert genau dann den Wert 1, wenn ein Eingabewert 0 und der andere 1 ist. Ansonsten ist das Ergebnis 0. Notiert wird diese Operation mit dem Zeichen »⊕«. Es ergeben sich daher folgende Gleichungen:

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

Die Exklusiv-oder-Verknüpfung spielt in der Kryptografie eine sehr wichtige Rolle und wird Ihnen in diesem Buch noch des Öfteren begegnen.

Nun, da wir schon am Anfang dieses Buches ein absolut sicheres und obendrein einfaches Verschlüsselungsverfahren entdeckt haben, wozu müssen wir uns überhaupt weiterhin mit diesem Thema beschäftigen? Ganz einfach, weil der One-Time-Pad auch seine Nachteile hat:

- Der Umgang mit einem Schlüssel, der genauso lang ist wie die Nachricht, ist nicht besonders handlich. Wollen Alice und Bob dieses Verfahren im Internet anwenden, dann muss zu jedem verschickten Bit ein Schlüssel-Bit existieren, das nur Alice und Bob bekannt ist. Verwendeten beispielsweise alle Internetanwender den One-Time-Pad, dann würde sich das Datenaufkommen im Internet verdoppeln.
- Es ist wesentlich schwieriger, als man denkt, große Mengen an Zufallszahlen herzustellen, die wirklich zufällig sind. Mehr dazu im Kapitel über Zufallszahlen (Kapitel 15).

Diese Nachteile sind so gravierend, dass der One-Time-Pad in der Praxis kaum eingesetzt wird. Es gibt jedoch sehr viele Verfahren, die die dahinterstehende Idee ausnutzen, indem sie aus einem kurzen Schlüssel einen langen generieren, der dann wie im One-Time-Pad eingesetzt wird. Eine große Bedeutung hatte der One-Time-Pad früher im Geheimdienst- und im Militärbereich. Spione trugen damals häufig Zettel mit zufälligen Buchstabenfolgen mit sich, die sie als One-Time-Pad-Schlüssel verwendeten. Ein solches Verschlüsselungsverfahren war sehr sicher und vergleichsweise einfach zu handhaben. Der Spion brauchte dazu nur Stift und Papier.

4.4 Permutationschiffren

Neben den Substitutionschiffren gibt es eine weitere Klasse von Verschlüsselungsverfahren: die **Permutationschiffren**. Bei einer Permutationschiffre werden die Buchstaben des Klartexts nicht durch andere ersetzt, sondern in ihrer Reihenfolge vertauscht. Betrachtet man beispielsweise jeweils fünf Klartextbuchstaben auf einmal, dann ist durch folgende Vorschrift eine Permutationschiffre gegeben:

> Buchstabe 4 kommt auf Position 1, 1 auf 2, 2 auf 3, 5 auf 4 und 3 auf 5. Der Schlüssel ist hierbei (in Kurzform): (4, 1, 2, 5, 3).

Der Klartext

```
ES GIBT ZWEI ARTEN VON LEUTEN: SOLCHE, DIE ZU ENDE BRINGEN, WAS SIE ANFANGEN.
```

wird durch diese Chiffrierung zum Geheimtext:

```
IE SBGE TZIW EARNT LVO ENNUTS: EHOLEC, ZDI UE EENB DGRIENS, NWS ANI EFAEANNG.
```

Eine solche Verschlüsselung können Alice und Bob auch mit einem Passwort als Schlüssel durchführen, das leichter zu merken ist als eine Folge von Zahlen. Die Vertauschungsvorschrift erhalten sie, indem sie die Buchstaben dieses Worts alphabetisch sortieren. Beispielsweise ergibt sich aus dem Passwort ALICE die alphabetisch sortierte Folge ACEIL, was (1, 4, 5, 3, 2) entspricht. Am einfachsten können Alice und Bob dieses Verfahren nutzen, wenn Absenderin Alice den Klartext unter dem Schlüsselwort zeilenweise aufschreibt und dann die Spalten umsortiert:

```
ALICE          ACEIL
_____          _____
ESGIB          EIBGS
TZWEI          TEIWZ
ARTEN          AENTR
VONLE          VLENO
UTENS          UNSET
OLCHE    =>    OHECL
DIEZU          DZUEI
ENDEB          EEBDN
RINGE          RGENI
NWASS          NSSAW
IEANF          INFAE
ANGEN          AENGN
```

Der Geheimtext lautet also EIBGSTEIWZAENTR... Dieses Verfahren wird auch als **Zeilentransposition** bezeichnet. Sicherer ist es jedoch, wenn Alice den Geheimtext aus der Tabelle spaltenweise generiert. In diesem Fall entsteht dadurch: ETAVUODERNIAIE... Eine in dieser Form verwendete Permutationschiffre wird auch als **Spaltentransposition** oder als **Würfel** bezeichnet. Führt Alice Verfahren zweimal hintereinander mit unterschiedlichen Passwörtern aus, dann spricht man von einem **Doppelwürfel**.

Die Kryptoanalyse einer Permutationschiffre funktioniert naturgemäß anders als bei einer Substitutionschiffre. Das Buchstabenzählen dient hier allenfalls dazu, die Sprache des Texts zu ermitteln. Für die eigentliche Kryptoanalyse sind dagegen stets Buchstabenkombinationen von Interesse. Besonders häufige Kombinationen, wie (im Deutschen) EN, ER, CH, EIN oder UND, liefern Mallory Anhaltspunkte dafür, wie die ursprüngliche Reihenfolge ausgesehen haben könnte. Das Q ist interessant, weil ihm fast immer ein U folgt. Seltene Kombina-

tionen (etwa JH oder VV) sind für Mallory ebenfalls von Nutzen, da sie anzeigen, was nicht zusammengehört. Der US-Kryptologe Herbert Yardley (1889–1958) beschreibt in seinem Buch *The American Black Chamber*, wie er im Jahre 1918 die Permutationschiffre des deutschen Spions Lothar Witzke knackte [Yardle]. Sein wichtigster Ansatzpunkt war der Buchstabe C, der im Deutschen fast immer vor einem H oder einem K steht [Schm12/1].

Bei einfachen Permutationschiffren kann Mallory mit den genannten Kryptoanalyse-Methoden durchaus zum Erfolg kommen. Mit zunehmender Schlüssellänge wird die Aufgabe für Mallory jedoch immer schwieriger. Besonders schwer zu knacken ist ein Doppelwürfel. Wenn Alice und Bob dieses Verfahren richtig verwenden (mit zwei langen Wörtern, deren Länge keine gemeinsamen Teiler haben), dann ist die Kryptoanalyse selbst mit Computerunterstützung schwierig bis unlösbar. Der Doppelwürfel zählt daher zu den besten Verfahren, die sich ohne Computerunterstützung praktikabel nutzen lassen. Im Kalten Krieg war der Doppelwürfel deshalb – wie der One-Time-Pad – ein beliebtes Verfahren für Spione, die verschlüsselte Nachrichten an ihre Agentenführer verschickten. 1999 stellte der ehemalige BSI-Präsident Otto Leiberich in einem Zeitschriftenartikel die Frage, ob und wie der Doppelwürfel zu lösen sei [Leiber]. Er schlug vor, zu diesem Zweck einen mit dem Doppelwürfel verschlüsselten Text zu erstellen und Interessierte zum Knacken dieser Verschlüsselung aufzurufen. Dieser Aufforderung bin ich im Jahr 2007 in meinem Buch *Codeknacker gegen Codemacher* nachgekommen [Schm07/1]. Im Folgenden ist der Text noch einmal abgedruckt:

VESINTNVONMWSFEWNOEALWRNRNCFITEEICRHCODEEA
HEACAEOHMYTONTDFIFMDANGTDRVAONRRTORMTDHE
OUALTHNFHHWHLESLIIAOETOUTOSCDNRITYEELSOANGP
VSHLRMUGTNUITASETNENASNNANRTTRHGUODAAARAO
EGHEESAODWIDEHUNNTFMUSISCDLEDTRNARTMOOIREEY
EIMINFELORWETDANEUTHEEEENENTHEOOEAUEAEAHUHI
CNCGDTUROUTNAEYLOEINRDHEENMEIAHREEDOLNNIRAR
PNVEAHEOAATGEFITWMYSOTHTHAANIUPTADLRSRSDNOT
GEOSRLAAAURPEETARMFEHIREAQEEOILSEHERAHAOTNT
RDEDRSDOOEGAEFPUOBENADRNLEIAFRHSASHSNAMRLT
UNNTPHIOERNESRHAMHIGTAETOHSENGFTRUANIPARTAOR
SIHOOAEUTRMERETIDALSDIRUAIEFHRHADRESEDNDOION
ITDRSTIEIRHARARRSETOIHOKETHRSRUAODTSCTTAFSTHCA
HTSYAOLONDNDWORIWHLENTHHMHTLCVROSTXVDRESDR

Der Klartext entstammt der englischen Sprache. Gleiches gilt für die beiden verwendeten Schlüsselwörter (sie sind relativ lang und haben eine unterschiedliche Anzahl an Buchstaben). Der verschlüsselte Text ist auch in der Dokumentation der kostenlosen Krypto-Lernsoftware CrypTool (ab Version 1.4.20) enthalten, und er ist online verfügbar [Schm08/1]. Bis Ende 2012 hat mir niemand eine Lösung zugeschickt. Das Rätsel ist also noch offen.

Der Doppelwürfel und andere Permutationschiffren wurden in der Geschichte der Kryptografie meist unterschätzt. Nur wenige Verschlüsselungsexperten erkannten, dass das Umstellen der Buchstaben oft mehr Sicherheit bietet als das Ersetzen. Doch trotz einer potenziell hohen Sicherheit haben Permutationschiffren auch einige Nachteile. So ist eine Known-Plaintext-Attacke für Mallory möglich, sofern Alice und Bob nicht für jede Nachricht einen neuen Schlüssel verwenden. Außerdem lässt sich am Geheimtext auch ohne Entschlüsselung einiges erkennen. Insbesondere kann Mallory über die Buchstabenhäufigkeit auf die verwendete Sprache schließen. Bei kürzeren Nachrichten (etwa solchen, die nur aus einem Wort bestehen) ist die Sicherheit einer Permutationschiffre nicht besonders hoch.

Abb. 4–5 *Das Voynich-Manuskript ist das bekannteste Rätsel der Kryptografie-Geschichte. Bis heute ist nicht bekannt, was sich hinter den seltsamen Buchstaben verbirgt.*

4.5 Ungelöste Verschlüsselungen

Die meisten Verschlüsselungsverfahren, die ich auf den vorhergehenden Seiten beschrieben habe, sind schon recht alt und lassen sich mit dem Computer einfach lösen. Geheimtexte, die älter als etwa 60 Jahre sind, stellen Kryptografen generell selten vor größere Probleme. Es gibt jedoch Ausnahmen. Drei ungelöste Verschlüsselungen, die historisch besonders interessant sind, schauen wir uns im Folgenden an. Ausführlichere Betrachtungen dazu können Sie in [Schm12/1] nachlesen.

4.5.1 Das Voynich-Manuskript

Das Voynich-Manuskript ist das wohl bedeutendste ungelöste Rätsel der Kryptografie-Geschichte [Schm12/1]. Es handelt sich dabei um ein handgeschriebenes Buch aus dem 15. Jahrhundert, in dem zahlreiche Abbildungen enthalten sind. Der Text ist in unbekannten Buchstaben verfasst. Insgesamt enthält das Voynich-Manuskript etwa 170.000 Schriftzeichen, was eigentlich eine gute Voraussetzung ist, um eine erfolgreiche Kryptoanalyse durchzuführen (auch wenn das verwendete Verfahren nicht bekannt ist). Bisher sind jedoch alle Versuche, das Voynich-Manuskript zu entschlüsseln, erfolglos geblieben.

Einigermaßen sicher ist inzwischen immerhin, dass der Text nicht in einer natürlichen Sprache in unverändertem Zustand verfasst ist. Diesen Schluss lassen zahlreiche statistische Untersuchungen zu, deren Ergebnisse kaum mit einer bekannten Sprache in Einklang zu bringen sind. Möglich ist dagegen eine natürliche Sprache in veränderter Form, eine Kunstsprache oder eine Verschlüsselung. Ebenfalls denkbar (und meiner Meinung nach nicht gerade unwahrscheinlich) ist die Hypothese, dass der gesamte Voynich-Text nur bedeutungslosen Unfug enthält.

Allerdings tappt die Voynich-Forschung bezüglich dieser Fragen noch reichlich im Dunkeln. Die Befürworter der Verschlüsselungstheorie konnten bisher noch nicht erklären, welches Verfahren der Verfasser verwendete. Auch nach einer passenden Kunstsprache oder Sprachveränderung suchen Voynichologen bisher vergebens. Selbst die Unfug-Theorie leidet darunter, dass bisher niemand herausgefunden hat, mit welcher Methode der Urheber einen sinnlosen Buchstabensalat mit den entsprechenden statistischen Eigenschaften produziert hat.

4.5.2 Das Thouless-Kryptogramm

Der britische Psychologe Robert Thouless (1894–1984) startete 1948 ein kurioses Experiment [Schm12/1]. Er veröffentlichte einen verschlüsselten Text und kündigte an, den Schlüssel – falls möglich – nach seinem Tod aus dem Jenseits zu übermitteln. Sollte dies gelingen, dann wäre bewiesen, dass Tote mit Lebenden kommunizieren können. Der Geheimtext, den er zu diesem Zweck erstellte, lautete:

CBFTM HGRIO TSTAU FSBDN WGNIS BRVEF BQTAB
QRPEF BKSDG MNRPS RFBSU TTDMF EMA BIM

Es wäre sicherlich sinnvoll gewesen, das verwendete Verfahren zu veröffentlichen und nur den Schlüssel geheim zu halten. Der in Verschlüsselungsfragen nicht besonders beschlagene Thouless gab jedoch weder das eine noch das andere bekannt. So stand lediglich der Geheimtext öffentlich zur Verfügung. Dennoch meldete sich bereits nach einigen Wochen ein Kryptografieexperte, der die Lösung gefunden hatte. Das Verfahren entpuppte sich als Playfair-Chiffre, der

Schlüssel lautete SURPRISE. Der Klartext war ein Shakespeare-Zitat: BALM OF HURT MINDS GREAT NATURE'S SECOND COURSE CHIEF NOURISHER IN LIFE'S FEAST. Allerdings hatte Thouless vorgebaut, falls sein Geheimtext nicht bis zu seinem Tod geheim bleiben würde, und einen zweiten Text mit dem gleichen Ziel veröffentlicht:

INXPH CJKGM JIRPR FBCVY WYWES NOECN SCVHE GYRJQ
TEBJM TGXAT TWPNH CNYBC FNXPF LFXRV QWQL

Dieses Mal beschrieb Thouless genau, wie er beim Verschlüsseln vorgegangen war. Das Verfahren ist eine Abwandlung der Vernam-Chiffre, bei der jedes Wort im Schlüsseltext der Verschlüsselung eines Buchstabens dient. Wo der Schlüsseltext nachzulesen ist (vermutlich handelt es sich um einen Auszug aus einem bekannten Buch), wollte Thouless aus dem Jenseits mitteilen. Das ist in den drei Jahrzehnten seit Thouless' Ableben jedoch noch nicht passiert, eine erfolgreiche Kryptoanalyse ist ebenfalls nicht gelungen. Thouless' zweiter Geheimtext ist also ein ungelöstes Rätsel. Es gibt noch einen dritten Text, den Thouless zwecks posthumer Übermittlung des Schlüssels hinterlassen hat. Der Kryptograf James Gillogly fand 1995 (also elf Jahre nach Thouless' Tod) die Lösung. Jenseitige Hilfe nahm er dabei nicht in Anspruch [GilHar].

4.5.3 Dorabella-Chiffre

Die Dorabella-Chiffre ist ein 87 Buchstaben langer Geheimtext, den der britische Komponist Edward Elgar (1857–1934) im Jahr 1897 an eine Bekannte schickte [Schm07/1]. Der Text ist in Fantasiebuchstaben verfasst, die jeweils aus einem oder mehreren Bögen bestehen. Die Dorabella-Chiffre blieb der Nachwelt erhalten, bisher hat jedoch niemand die Lösung gefunden.

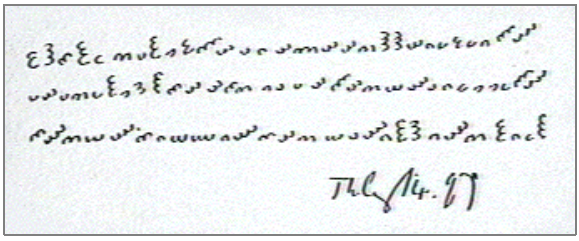


Abb. 4–6 *Die Dorabella-Chiffre aus dem Jahr 1897 ist bisher ungelöst.*

5 Die Enigma und andere Verschlüsselungsmaschinen

Um das Jahr 1920 begann in der Kryptografie eine neue Ära und gleichzeitig eines der spannendsten Kapitel der Technikgeschichte. Noch im Ersten Weltkrieg hatten die beteiligten Armeen ihre Funksprüche mit Verfahren verschlüsselt, die sich mit Papier und Bleistift durchführen ließen [Schm07/1]. Die Folgen waren oft verheerend, denn praktisch alle damals eingesetzten Chiffren wurden früher oder später geknackt. Der Bedarf an verbesserten Verschlüsselungsverfahren war daher offensichtlich und rief nach Kriegsende verschiedene Tüftler auf den Plan, die spezielle Maschinen für das Verschlüsseln entwickelten. In den zwanziger Jahren kamen mehrere Geräte dieser Art auf den Markt. Bis zum Aufkommen des Computers gehörten mechanische und elektromechanische Verschlüsselungsmaschinen zur Standardausstattung von Behörden und Militär.

Gleichzeitig rüsteten jedoch auch die Codeknacker kräftig auf. Sie betrieben ihr Handwerk teilweise in industriellen Dimensionen und schafften es immer wieder, scheinbar sichere Verschlüsselungsmaschinen zu knacken. Vor allem der

Zweite Weltkrieg war von einem unglaublichen Wettlauf zwischen Kryptografen und Dechiffrierern geprägt, dessen Einzelheiten erst Jahrzehnte später öffentlich bekannt wurden. Dabei spielte nicht zuletzt die legendäre deutsche Verschlüsselungsmaschine Enigma eine wichtige Rolle. Auch sie galt als sicher und wurde doch geknackt. Die ausgesprochen faszinierende Geschichte der mechanischen Verschlüsselungsmaschinen, die bis etwa 1970 dauerte, wird ausführlich in meinem Buch *Codeknacker gegen Codemacher* beschrieben [Schm07/1]. An dieser Stelle will ich mich mit einer kurzen Zusammenfassung begnügen.

5.1 Rotorchiffren

Zu den wichtigsten Pionieren der maschinellen Verschlüsselungstechnik gehörte der US-Amerikaner Edward Hebern. Dieser kam 1918 auf die Idee, ein elektromechanisches Gerät zu bauen, das Texte verschlüsselte, die über eine Schreibmaschinentastatur eingegeben wurden.

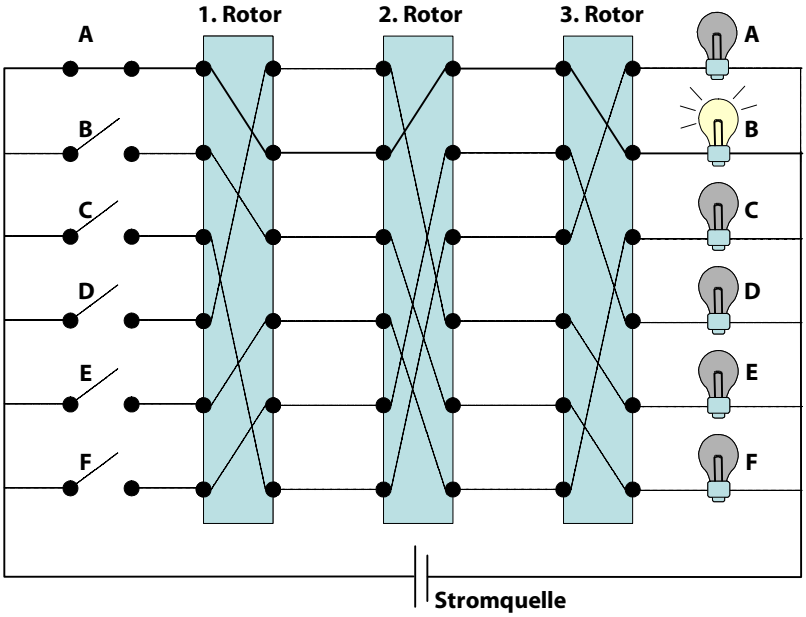


Abb. 5–1 *Eine Rotorchiffre mit drei Rotoren (statt 26 sind nur sechs Buchstaben vorhanden): Wird die »A«-Taste gedrückt, dann leuchtet durch die Verdrahtung die »B«-Lampe auf. Da sich die Rotoren drehen, ändert sich der Weg des Stroms ständig.*

5.1.1 Heberns Rotormaschine

Das Design von Heberns Maschine sah eine oder mehrere kreisrunde Scheiben (Rotoren) vor, die auf beiden Seiten mit jeweils 26 Metallkontakten versehen waren. Im Folgenden gehen wir von drei derartigen Rotoren aus. Jeder Kontakt auf der einen Seite eines Rotors war mit einem Kontakt auf der anderen Seite verdrahtet. Die drei Rotoren waren wie bei einem Tachometerzähler aneinandergekoppelt. Wurde ein Kontakt auf der linken Seite des linken Rotors durch Drücken einer von 26 Tasten mit einer Stromquelle verbunden, dann floss Strom durch alle drei Rotoren hindurch und brachte nach dem dritten Rotor eine von 26 angeschlossenen Lampen zum Aufleuchten. Jede Taste war mit einem Buchstaben beschriftet, jede Lampe ebenfalls.

Durch diese Anordnung ließ sich jeder Buchstabe auf einen durch die Rotorverdrahtung festgelegten anderen abbilden – dadurch entstand ein Verschlüsselungsverfahren (siehe Abbildung 5–1). Dieses wurde dadurch noch komplizierter, dass nach jedem Tastendruck der linke Rotor um eine Einheit gedreht wurde. Nach einer vollen Umdrehung drehte sich auch der mittlere Rotor um eine Einheit, entsprechend nach einer vollen Umdrehung des mittleren der rechte (wie beim Tachometerzähler). Eine Chiffre dieses Typs wird **Rotorchiffre** genannt. Fast zeitgleich mit Edward Hebern kamen drei weitere Erfinder auf die Idee, Rotor-Chiffriermaschinen zu bauen. Der bekannteste davon war Arthur Scherbius, von dem noch die Rede sein wird (seine Maschine war die Enigma). Aus Kryptografen-Sicht ist eine Rotorchiffre eine polyalphabetische Substitutionschiffre. Die Tabelle, nach der aus dem Klartext der Geheimtext gebildet wird, ändert sich nach jedem Buchstaben, da sich nach jeder Eingabe mindestens ein Rotor dreht. Erst wenn sich der rechte Rotor einmal komplett gedreht hat, wiederholt sich die Substitutionsvorschrift. Dies ist bei drei Rotoren nach 26^3 (also 17.576) Buchstaben der Fall.

Wie jedes gute Verschlüsselungsverfahren, so arbeiten auch Rotorchiffren mit einem Schlüssel. Geht man davon aus, dass die Verdrahtung der Rotoren über längere Zeit konstant bleibt und Mallory bekannt ist, dann ist die Anfangsstellung der Rotoren der Schlüssel. Die Anzahl der Schlüssel ist dann so groß wie die der Rotorstellungen und beträgt bei drei Rotoren 17.576. Diese Zahl lässt sich noch deutlich erhöhen, wenn man die Rotoren austauschen kann. Hat man beispielsweise fünf unterschiedlich verdrahtete Rotoren zur Verfügung und kann davon jeweils drei in beliebiger Reihenfolge einsetzen, dann versechzigfacht sich die Anzahl der Schlüssel auf etwa eine Million. Die meisten Betreiber von Rotor-Chiffriermaschinen gingen zudem davon aus, dass der Angreifer die Verdrahtung der Rotoren nicht kannte.

5.1.2 Die Enigma

Ebenfalls um das Jahr 1918 baute der Deutsche Arthur Scherbius unabhängig von Hebern eine Verschlüsselungsmaschine, die eine Rotorchiffre realisierte. Seiner Maschine gab er den bezeichnenden Namen **Enigma** (griechisch für »Rätsel«) – sie wurde später zur bekanntesten Verschlüsselungsmaschine überhaupt. Die Enigma hatte in ihrer gängigsten Variante drei Rotoren. Es gab jedoch ein zusätzliches Bauteil: Hinter dem dritten Rotor war ein weiterer, unbeweglicher Rotor (ein sogenannter Reflektor) angebracht, der nur auf einer Seite Kontakte hatte. Diese waren miteinander paarweise verdrahtet (siehe Abbildung 5–3). Wurde eine der 26 Tasten gedrückt, dann floss der Strom durch die drei Rotoren in den Reflektor und von dort wieder zurück, wiederum durch die drei Rotoren hindurch, um danach eine Lampe zum Leuchten zu bringen. Der Reflektor, so dachte man damals, würde die Maschine deutlich sicherer machen – ein fataler Irrtum, wie wir heute wissen.

Arthur Scherbius konnte sich nicht lange über seine Erfindung freuen. 1926 starb er nach einem Unfall, und 1934 ging seine Firma in Konkurs. Bereits Ende der zwanziger Jahre setzte jedoch die Deutsche Wehrmacht die Enigma für militärische Zwecke ein und bescherte der Nachfolgegesellschaft von Scherbius' Unternehmen Ende der 30er Jahre dann doch noch ein einträgliches Geschäft.

Abb. 5–2 *Die Enigma wurde im Zweiten Weltkrieg von den Deutschen eingesetzt. Sie gilt als die berühmteste Verschlüsselungsmaschine der Welt.*

Kryptoanalyse der Enigma

Wie die Enigma von polnischen und britischen Kryptografen zwischen 1928 und 1945 trotz ständiger Verbesserungen immer wieder geknackt wurde, ist eine der spannendsten Geschichten, welche die Kryptografie überhaupt zu bieten hat. Den Anfang dieser Geschichte markierte das Jahr 1927, als sich der polnische Geheimdienst eine der damals noch käuflich erwerbbaren Enigma-Maschinen besorgte und dadurch die Funktionsweise der Maschine kannte (dies bestätigt, dass man immer davon ausgehen sollte, dass der Abhörer das Verfahren kennt). Die drei polnischen Mathematiker Marian Rejewski, Henryk Zygalski und Jerzy Rozycki machten sich in der Folgezeit an die Arbeit und konnten schon bald erste Erfolge verzeichnen. Nicht zuletzt auch dank der Unterstützung durch einen Spion gelang es ihnen 1932, die Verdrahtung von Wehrmacht-Enigmas zu rekonstruieren. Bereits im folgenden Jahr waren sie so weit, dass sie die jeweilige Rotor-Anfangsstellung (also den Schlüssel) bestimmen konnten, wodurch sie die Enigma geknackt hatten. 1938 schafften es Rejewski und seine Kollegen sogar, eine Maschine zu konstruieren, die das Entschlüsseln von Enigma-Nachrichten deutlich erleichterte und als Vorläufer heutiger Computer betrachtet werden kann. Die Erfinder nannten die Maschine »Bomba« nach dem polnischen Wort für Eisbombe.

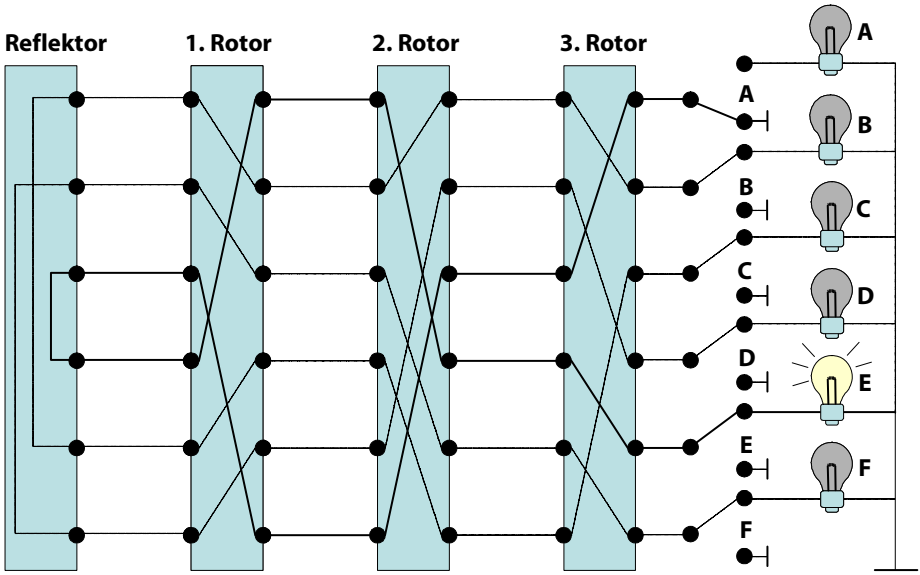


Abb. 5–3 *Die Enigma realisiert eine Rotorchiffre, bei der jeder Rotor zweimal durchlaufen wird. Diese Eigenschaft wird durch den Reflektor gewährleistet, der nur auf einer Seite Kontakte hat.*

Probleme machte den Polen allerdings die Tatsache, dass die Deutschen unterschiedliche Enigma-Versionen verwendeten (insgesamt mindestens 50 bis Kriegsende) und für besonders sicherheitskritische Bereiche einen vierten Rotor und

weitere Verbesserungen einführten. Diese Übermacht veranlasste die polnischen Mathematiker 1938 dazu, den britischen Geheimdienst einzuweihen. Die Briten wussten die Informationen zu nutzen. Unter Mitwirkung des bedeutenden Mathematikers Alan Turing entwickelten sie die Bomba zu einer leistungsfähigen Maschine weiter, die sie »Bombe« nannten (auch dies bedeutet »Eisbombe« und ist nicht mit dem Wort »bomb« zu verwechseln). Im Deutschen ist der Begriff **Bombe** bzw. **Turing-Bombe** verbreitet. In ihrem Dechiffrier-Zentrum in Bletchley Park bei London betrieben die Briten bis zum Ende des Zweiten Weltkriegs die Enigma-Kryptoanalyse mithilfe von Turing-Bomben in industriellen Ausmaßen. Bis zu 7.000 Mitarbeiter – darunter viele Frauen – waren dort unter strengster Geheimhaltung mit dem Dechiffrieren beschäftigt. Die meisten Angestellten kannten nur ihr unmittelbares Arbeitsumfeld und wussten daher nicht, was für eine entscheidende Bedeutung ihre Arbeit hatte.

Mithilfe der Dechiffrier-Fabrik in Bletchley Park gelang es den Briten zwar, einen Großteil der abgefangenen deutschen Funksprüche zu entschlüsseln. Die zahlreichen Enigma-Varianten und die diversen Verbesserungen machten jedoch auch ihnen zu schaffen, und so gab es durchaus auch Nachrichten, welche die Briten nicht lösen konnten. Dennoch ist heute klar: Die erfolgreiche Kryptoanalyse der Enigma hatte einen enormen Einfluss auf den Verlauf des Zweiten Weltkriegs. Vor allem der U-Boot-Krieg im Nordatlantik wäre ohne die geknackte Enigma anders verlaufen, denn durch abgefangene und entschlüsselte Funksprüche waren die Alliierten oftmals bestens über die Position der deutschen U-Boote informiert. Möglicherweise hätte Deutschland im Zweiten Weltkrieg sogar eine Atombombe abbekommen, wenn die Enigma nicht geknackt worden wäre. Denn dies hätte den Krieg zweifellos verlängert, und bekanntlich warfen die Amerikaner bereits drei Monate nach der Kapitulation Deutschlands die erste Atombombe auf Hiroshima.

Abb. 5–4 *Mit der Turing-Bombe knackten die Briten die Enigma.*

Von der dramatischen Geschichte der Enigma erfuhr die Öffentlichkeit nach dem Zweiten Weltkrieg erst einmal nichts. Der britische Premierminister Winston Churchill ließ die Maschinen in Bletchley Park vernichten, die Resultate der Codeknacker blieben Staatsgeheimnis. Erst 1974 wurde die Sache öffentlich. Die ganze Wahrheit über die Enigma ist damit sicherlich noch nicht auf dem Tisch. Man kann beispielsweise nur darüber spekulieren, was die Sowjetunion über die Enigma wusste. Es ist kaum anzunehmen, dass sich von den zahlreichen hervorragenden sowjetischen Mathematikern keiner damit beschäftigte.

Wie die Enigma-Kryptoanalyse funktionierte

Die Kryptoanalyse der Enigma ist deutlich komplexer als die der Vigenère- oder gar der Cäsar-Chiffre. Ich kann auf dieses Thema daher nur überblicksweise eingehen. Zunächst einmal ist es wichtig zu wissen, dass eine Ciphertext-Only-Attacke auf die Enigma bei unbekannter Verdrahtung sehr schwierig ist. Es gibt jedoch wirksame Known-Plaintext-Attacken. Mit diesen gelang es den Polen und den Briten, die Verdrahtung einiger Maschinen zu bestimmen.

Zur Bestimmung des Schlüssels bei bekannter Verdrahtung (also der Rotor-Anfangsstellung) gibt es eine Ciphertext-Only-Attacke, bei der eine Eigenschaft der Enigma hilft, die auf den Reflektor zurückzuführen ist – obwohl gerade dieser die Maschine sicherer machen sollte. Wie Sie sich leicht überzeugen können, sorgt der Reflektor dafür, dass kein Buchstabe bei der Verschlüsselung auf sich selbst abgebildet werden kann. Kennt man ein längeres Wort, das irgendwo im Klartext vorkommen könnte (der Wortschatz im Krieg war ja begrenzt), dann schiebt man dieses so lange über den Geheimtext, bis kein Buchstabe des Worts mit dem Geheimtext übereinstimmt. Nun hat man mit einer gewissen Wahrscheinlichkeit ein Klartext-Geheimtext-Paar. Auf dieser Basis konnte der Schlüssel oft bestimmt werden.

Trotz all dieser Kryptoanalyse-Ansätze hätten die Polen und die Briten nicht allzu viel erreicht, wenn ihnen nicht einige günstige Umstände geholfen hätten. Dazu gehörten etwa der bereits erwähnte Spion und die Tatsache, dass den Briten 1941 ein deutsches U-Boot samt Schlüsselbuch in die Hände fiel. Vor allem gehörte dazu aber auch die sträfliche Sorglosigkeit der Deutschen. Diese begingen so ziemlich jeden Fehler, den man bei der Nutzung eines Verschlüsselungssystems machen kann. Immer wieder verwendeten sie einfach zu erratende Anfangsstellungen (etwa `AAA` oder `ABC`). Ein täglicher Schlüsselwechsel bereitete den Dechiffrierern häufig keine großen Probleme, da Routinemeldungen oftmals mit gleichem Wortlaut und täglich zur gleichen Uhrzeit abgeschickt wurden – dies ermöglichte eine Known-Plaintext-Attacke. Solche und ähnliche Fehler erleichterten den Polen und Briten ihre Arbeit ungemein.

Bleibt noch die Frage, ob die Enigma ohne den Reflektor (also bei gleicher Bauweise wie die Hebern-Maschine) wesentlich sicherer gewesen wäre. Vermutlich nicht, denn auch die Hebern-Maschine ist zu knacken, wenn auch auf andere

Weise. Der US-Kryptologe William Friedman (siehe Abschnitt 40.1.1) demonstrierte dies im Jahr 1926. Erst spätere Generationen der Rotor-Chiffriermaschinen boten eine ausreichend hohe Sicherheit.

5.1.3 Weitere Rotor-Chiffriermaschinen

Schon in den dreißiger Jahren war einigen Experten bewusst, dass weder die Enigma noch die Hebern-Maschine ausreichende Sicherheit bot. Die Hinzunahme weiterer Rotoren konnte dieses Problem alleine nicht lösen, da sich nur bei sehr langen Funksprüchen mehr als drei Rotoren bewegen. Sowohl die Deutschen als auch die Briten und Amerikaner erkannten, dass stattdessen eine Änderung der Fortschaltung der Rotoren Abhilfe schaffen konnte. Diese sollten sich nicht mehr tachometerartig, sondern nach einem möglichst unregelmäßigen Mechanismus bewegen.

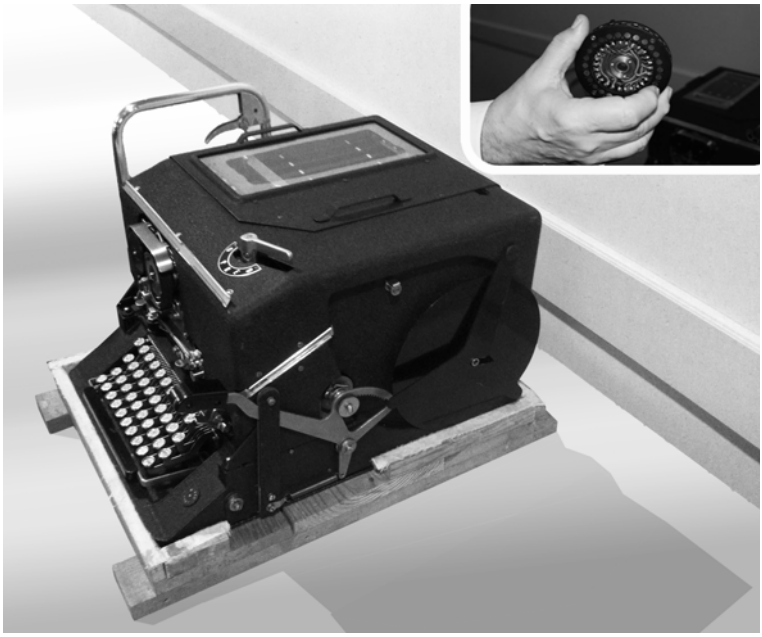


Abb. 5–5 *Die SIGABA wurde von den US-Amerikanern im Zweiten Weltkrieg eingesetzt. Sie bot eine größere Sicherheit als die Enigma.*

Auch die deutschen Krypto-Experten beschäftigten sich mit einer Änderung der Fortschaltung bei der Enigma. Sie kamen jedoch zu dem Schluss, dass ein solcher Schritt die Periode verkürzen würde, und verzichteten darauf. Die US-Amerikaner konstruierten dagegen eine neue Maschine, die sie **SIGABA** nannten. Es handelte sich dabei um eine Rotor-Chiffriermaschine mit 15 Rotoren, von denen jedoch nur fünf verdrahtet waren. Die restlichen zehn Rotoren steuerten die Fort-

schaltung der fünf verdrahteten. Die SIGABA-Rotoren drehten sich also nicht nach dem Tachometer-Prinzip, sondern nach einer komplexen Mechanik, die alle 15 Rotoren einbezog. Einen Reflektor gab es bei der SIGABA nicht. Für einen Außenstehenden war praktisch nicht nachvollziehbar, wie die Bewegung ablief.

Die komplexe Fortschaltung und die große Anzahl an Rotoren machten die SIGABA so sicher, dass sie nach heutigem Kenntnisstand nie geknackt wurde. Von Nachteil war lediglich, dass die SIGABA groß und unhandlich war. Sie war deshalb für den Einsatz im Gefecht nur bedingt geeignet. An der Front mussten die US-Soldaten deshalb mit der deutlich weniger sicheren M-209 vorliebnehmen, von der noch die Rede sein wird.

Während die US-Amerikaner mit der SIGABA arbeiteten, hatten die Briten ein ähnliches Gerät namens **Typex**. Dieses arbeitete mit fünf unregelmäßig fortgeschalteten Rotoren und wurde ebenfalls – nach heutigem Kenntnisstand – nie geknackt. Allerdings war auch die Typex kein Gerät für den Einsatz an der Front, sondern wurde nur von hochrangigen Militärs verwendet. Auch nach dem Zweiten Weltkrieg kamen noch zahlreiche Rotor-Chiffriermaschinen zum Einsatz. So entstand in der Schweiz die **NEMA** und in der Sowjetunion die **Fialka**. Die Schweizer Crypto AG brachte eine Maschine namens **HX-63** auf den Markt. Innerhalb der NATO wurde ein Gerät namens **KL-7** eingesetzt. Alle diese Maschinen wurden allem Anschein nach nie geknackt, da sie ausreichend viele Rotoren und eine komplexe Fortschaltmechanik hatten.

5.2 Andere Verschlüsselungsmaschinen

Nicht alle Verschlüsselungsmaschinen arbeiteten nach dem Rotorprinzip. Vor allem in den dreißiger Jahren, als die Nachfrage nach sicheren Verfahren deutlich zunahm, ließen sich die Konstrukteure einige Alternativen einfallen, um den Codeknackern ihr Handwerk zu erschweren. Im Zweiten Weltkrieg kamen mehrere unterschiedliche Typen von Verschlüsselungsmaschinen zur Anwendung, von denen noch die meisten geknackt wurden. Erst in den fünfziger Jahren gewannen die Chiffren-Designer gegenüber den Kryptoanalytikern die Oberhand, was einen entscheidenden Einschnitt in der langen Geschichte der Kryptografie darstellt.

5.2.1 Die Kryha-Maschine

Zu den Kuriositäten in der Kryptografie-Geschichte zählt zweifellos die Verschlüsselungsmaschine des Ingenieurs und Geschäftsmanns Alexander von Kryha [Schm10]. Diese wird als **Kryha-Maschine** bezeichnet. Von Kryha kam Anfang der zwanziger Jahre aus der Ukraine nach Deutschland, wo er sich als Unternehmer betätigte. Nachdem er zuvor beim ukrainischen Militär mit Verschlüsselung zu tun gehabt hatte, entwickelte er ein Verschlüsselungsgerät, das er um 1925 auf

den Markt brachte. Es gab drei (kryptografisch gleichwertige) Varianten der Kryha-Maschine: die *Kryha Standard*, die *Kryha Liliput* und die *Kryha Elektrik*. Uns soll nur die Kryha Standard interessieren.

Die Kryha Standard enthielt einen Federantrieb und musste daher wie eine Uhr aufgezogen werden. Die Verschlüsselung wurde durch zwei konzentrische Buchstabenscheiben angezeigt, von denen sich die innere drehte. Im Innern der Maschine steckte ein unregelmäßig gezahntes Rad, das in 17 Einheiten aufgeteilt war. Pro Knopfdruck drehte es sich – bewegt vom Federantrieb – mit der inneren Buchstabenscheibe um eine Einheit. Eine Einheit entsprach zwischen einem und sechs Buchstaben auf der Scheibe. Geht man davon aus, dass sich die Zahnung des Rads und die Position der Buchstaben auf der Scheibe nicht änderten, dann konnte die Maschine 442 Zustände annehmen (dies ergibt sich aus 17 Zahnradstellungen und 26 Möglichkeiten, die Buchstabenscheibe dagegen zu verdrehen). Dies entsprach der Zahl der möglichen Schlüssel. Es versteht sich von selbst, dass eine Verschlüsselungsmethode mit einer so geringen Schlüsselzahl nicht besonders sicher ist.

Der wenig anspruchsvolle Aufbau der Kryha-Maschine zeigt: Im Gegensatz zu anderen Erfindern von Verschlüsselungsmaschinen war Alexander von Kryha kein überragender Techniker. Dafür war er ein begnadeter Selbstdarsteller und Vermarktungsexperte. Er verpasste seiner Maschine ein modernes Aussehen und verkaufte sie in einer samtgefütterten Ledertasche. Die Werbematerialien, die von Kryha drucken ließ, waren anspruchsvoll gestaltet und wirken heute noch erstaunlich aktuell. Zusätzlich startete der umtriebige Unternehmer mehrere PR-Aktionen und erreichte eine ausführliche Berichterstattung in den Medien.

Abb. 5–6 *Die in Deutschland entwickelte Kryha-Maschine erwies sich als unsicher.*

Angesichts der aggressiven Vermarktung versäumte es von Kryha jedoch, sich um die kryptografischen Eigenschaften seiner Maschine zu kümmern. Dabei wäre dies

dringend notwendig gewesen, denn die Kryha-Maschine zählte zu den unsichersten Verschlüsselungsmaschinen, die je gebaut wurden. Der bereits erwähnte US-Kryptologe William Friedman knackte das Gerät 1933 in weniger als drei Stunden. Trotz ihrer Schwächen fand die Kryha-Maschine eine gewisse Verbreitung in Europa und den USA. Im Zweiten Weltkrieg, als es einen großen Bedarf an Verschlüsselungstechnik gab, spielte Kryhas Gerät jedoch kaum eine Rolle – gegenüber der Enigma und anderen Chiffriermaschinen war es in allen Belangen unterlegen. Trotzdem versuchte Alexander von Kryha ab 1950 noch einmal, seine Maschine zu vermarkten. Da die Technik inzwischen hoffnungslos veraltet war, ging dies gründlich schief. 1955 nahm sich von Kryha in Baden-Baden das Leben.

5.2.2 Hagelin-Maschinen

Der Schwede Boris Hagelin gilt als der Einzige, der mit dem Verkauf von Verschlüsselungsgeräten reich wurde. Hagelin, der sowohl als Geschäftsmann als auch als Ingenieur hohes Ansehen erwarb, gründete in den zwanziger Jahren zusammen mit dem Tüftler Arvid Damm ein Unternehmen für Verschlüsselungstechnik. Damm hatte um 1920 etwa zeitgleich mit Arthur Scherbius und Edward Hebern eine Rotor-Verschlüsselungsmaschine entwickelt und versuchte nun, diese zu vermarkten. Als Damm 1927 starb, musste Hagelin alleine weitermachen. Nach einer längeren Durststrecke, die Hagelin nur durch die Unterstützung seines wohlhabenden Vaters überstand, kam für sein Unternehmen 1934 der Durchbruch. Als ersten großen Auftraggeber konnte Hagelin die französische Regierung gewinnen, die seine leicht abgewandelte Rotor-Verschlüsselungsmaschine **B-21** übernahm. Das nach leichten Modifikationen als **B-211** bezeichnete Gerät war kompakt und einfach zu bedienen, erwies sich jedoch später als unsicher. Doch immerhin, ein Anfang war gemacht.

Abb. 5–7 *Die B-211 von Hagelin war etwa zur gleichen Zeit wie die Enigma im Einsatz. Sie erwies sich als unsicher, was sich jedoch bei den Nachfolgemodellen ändern sollte.*

Zum Markenzeichen Hagelins wurde ein Typ von Verschlüsselungsmaschinen, den er in den Jahren darauf im Auftrag der Franzosen erfand. 1935 kam die erste davon unter dem Namen **C-35** auf den Markt. Das Design dieser als **C-Maschinen** bezeichneten Geräte basierte auf einer Geldwechselmaschine, die Hagelin einige Jahre zuvor entwickelt, aber nie gebaut hatte. Das Herzstück der C-Maschinen bildete ein Rad (Stangenrad), das sich bei der Eingabe eines Buchstabens um 360 Grad drehte und dabei eine Buchstabenscheibe antrieb. Dieser Antrieb war zusätzlich von mehreren unregelmäßig gezahnten Rädern abhängig, deren Zahnung sich ändern ließ.

Zu Hagelins größtem Verkaufserfolg wurde eine C-Maschine, die er an die US-Armee verkaufen konnte. Unter dem Namen **M-209** produzierten die USA in Lizenz insgesamt 140.000 Geräte dieses Typs. Diese kamen im Zweiten Weltkrieg in der US-Armee zum Einsatz. Im Gegensatz zur schweren und sperrigen SIGABA, die nur auf höherer militärischer Ebene verwendet wurde, ließ sich die M-209 bequem im Marschgepäck eines Soldaten unterbringen. Sie wurde dementsprechend vor allem für taktische Funksprüche eingesetzt.

Inzwischen weiß man, dass die M-209 unsicher war. Den Deutschen gelang es, die Maschine zu knacken, wie beispielsweise der ehemalige BSI-Präsident Dr. Otto Leiberich berichtet [Leiber]. 2004 hatte ich zudem die Gelegenheit, mich mit dem damals 84-jährigen Frankfurter Reinold Weber zu unterhalten, der seinerzeit am Knacken der M-209 beteiligt gewesen war und der nach über 60 Jahren erstmals öffentlich über seine Arbeit sprach. Die ausgesprochen spannende Geschichte Webers ist kostenlos im Internet nachzulesen [Schm04].

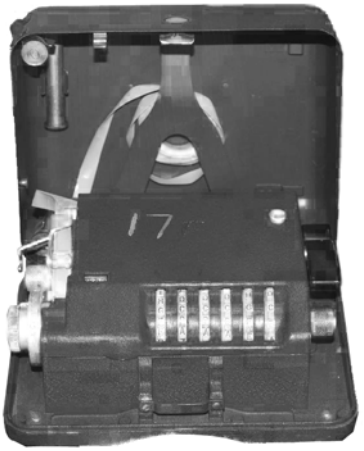


Abb. 5–8 *Die M-209 gilt als die am häufigsten gebaute Verschlüsselungsmaschine.*

5.2.3 Die Purple

Im Jahr 1937 entwickelte die japanische Marine eine Verschlüsselungsmaschine, die »97-shiki O-bun In-ji-ki« hieß. Dieses Gerät, das in der Literatur meist als **Purple** bezeichnet wird, spielte im Pazifikkrieg zwischen Japan und den USA eine wichtige Rolle. Es hatte einen ungewöhnlichen Aufbau, für den es anderswo keine Parallele gibt. Die Konstrukteure der Purple kannten die Enigma und ließen sich von deren Funktionsweise inspirieren. Statt Rotoren wählten sie jedoch Telefon-Vermittlungsschalter als wichtigstes Bauelement. Wie bei der Enigma drückte der Bediener eine Buchstabentaste auf einer Schreibmaschinentastatur, wodurch sich ein Stromkreis schloss, der den zugehörigen Geheimtextbuchstaben anzeigte.

Eine vereinfachte Purple-Variante ist in Abbildung 5–9 zu sehen. Der dort verwendete Telefonschalter hat drei Eingänge und pro Eingang drei Ausgänge. Über ein Verbindungsstück (*Zeiger*) ist jeder Eingang mit einem der drei zugehörigen Ausgänge verbunden. Mit der Eingabe eines Buchstabens bewegen sich alle drei Zeiger um eine Einheit nach unten (ist der unterste Ausgang erreicht, dann geht es mit dem obersten weiter). Die Verdrahtung zwischen den Ausgängen und den Lampen muss so gestaltet sein, dass nie zwei Zeiger bei derselben Lampe landen.

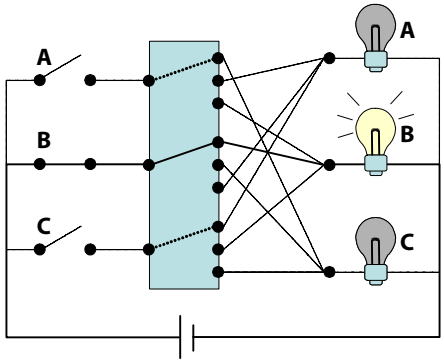


Abb. 5–9 *Die Purple (hier eine vereinfachte Version mit drei Buchstaben) arbeitete mit Telefon-Vermittlungsschaltern, in denen sich mehrere Zeiger synchron bewegten.*

Die echte Purple arbeitete mit 13 Vermittlungsschaltern. Jeder davon hatte sechs Ein- und 150 Ausgänge. Die Verdrahtung ist in Abbildung 5–10 zu sehen. Die mit x gekennzeichneten Elemente führen eine festverdrahtete Permutation durch, die uns an dieser Stelle nicht interessieren soll. Die jeweils 150 Ausgänge an einem Vermittlungsschalter sind nicht einzeln abgebildet. Wie man sieht, bleibt bei den meisten Vermittlungsschaltern ein Eingang unbelegt. Die Anfangsstellung der Zeiger diente als Schlüssel. Die Ausgabe erfolgte nicht über Lampen, sondern wie bei einer elektrischen Schreibmaschine. Die Purple war zwar komplexer aufgebaut als die Enigma, doch dafür war es einfacher, bei bekanntem Aufbau den Schlüssel zu ermitteln.

Ab 1939 versuchten die US-Amerikaner, die Purple zu knacken. Leiter des Projekts war William Friedman, der vielleicht beste Codeknacker der Krypto-Geschichte. Die Aufgabe war schwierig, denn Friedmans Truppe hatte keinerlei Informationen über den Aufbau der Maschine. Sie kannten lediglich einige ältere japanische Verschlüsselungsmaschinen, wodurch sie ahnten, dass Telefon-Vermittlungsschalter zu den Bauelementen gehörten. In langwieriger Arbeit kamen Friedman und seine Leute dem Design der Purple auf die Spur. 1940 gelang es ihnen, das Gerät nachzubauen, auch eine Methode zur Berechnung des Schlüssels wurde gefunden. Ab 1941 konnten die Amerikaner den Funkverkehr der Japaner routinemäßig innerhalb weniger Stunden entschlüsseln. Eine der größten Leistungen in der Geschichte der Kryptoanalyse war perfekt.

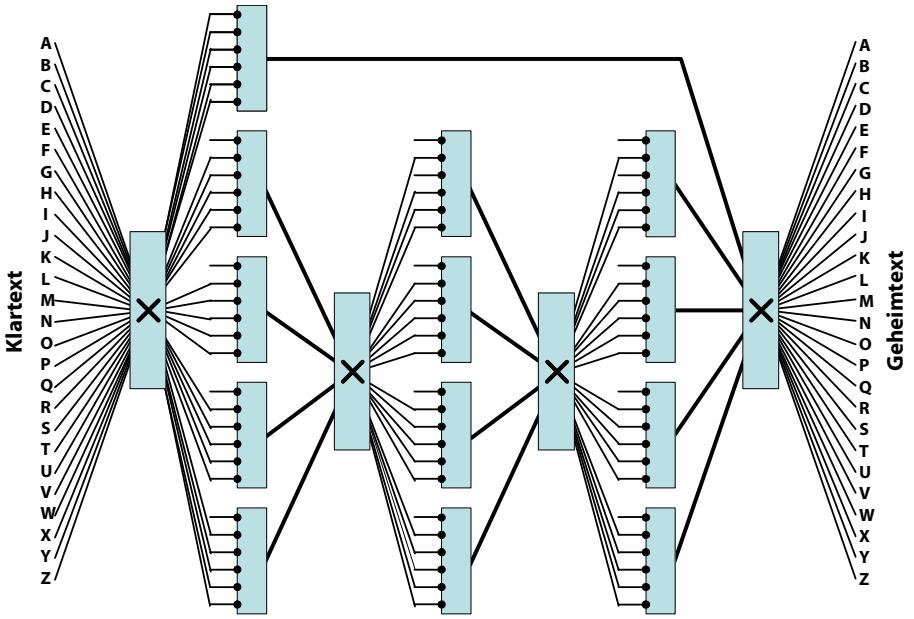


Abb. 5–10 *Die Purple arbeitete mit 13 Telefon-Vermittlungsschaltern, die je sechs Eingänge hatten (teilweise blieb einer davon ungenutzt). In jedem Schalter verbanden sechs Zeiger die Eingänge mit 150 Ausgängen. Zeiger und Ausgänge sind hier nicht abgebildet.*

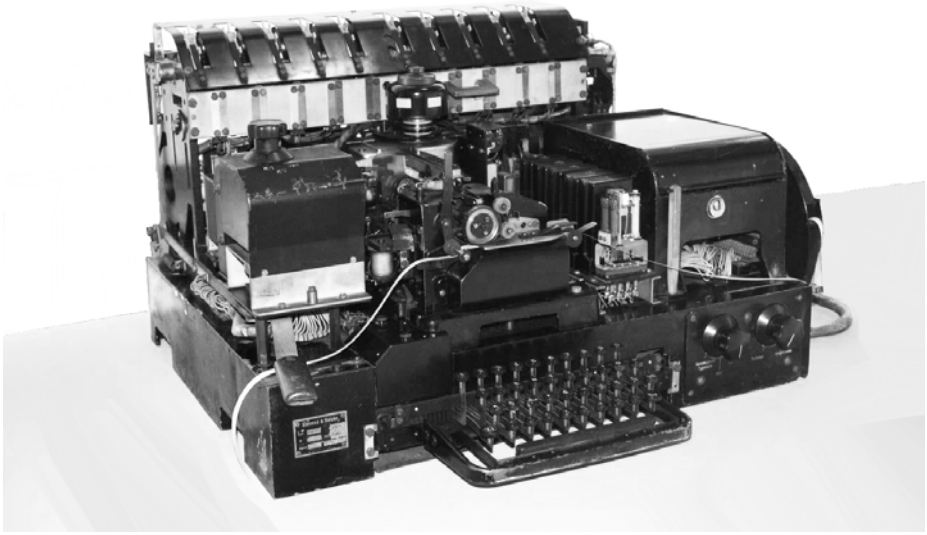


Abb. 5–11 *Der Geheimschreiber war nach der Enigma das bedeutendste deutsche Verschlüsselungsgerät des Zweiten Weltkriegs.*

5.2.4 Der Geheimschreiber

Die Enigma war nicht die einzige Verschlüsselungsmaschine, die die Deutschen im Zweiten Weltkrieg einsetzten. Das bekannteste unter den mindestens zehn anderen Geräten trug den Namen **T-52** und wird heute meist als **Geheimschreiber** bezeichnet. Es wurde von der Firma Siemens & Halske hergestellt. Im Vergleich zur Enigma war der Geheimschreiber ein größeres Gerät, das nur stationär einsetzbar war. Es wurde von der Luftwaffe eingesetzt und insgesamt in einigen hundert Exemplaren gebaut. Das Funktionsprinzip des Geheimschreibers unterschied sich deutlich von dem der Enigma. Zweck des Geräts war die Ver- und Entschlüsselung von Fernschreiben, die damals in einem fünfstelligen Binärcode (*Baudot-Code*) verschickt wurden. Aufgabe des Geheimschreibers war es also, einen Fünf-Bit-Wert auf einen anderen abzubilden. Dies geschah online: Tippte der Bediener einen Buchstaben ein, dann wurde dieser nach der Verschlüsselung direkt per Fernschreiber verschickt. Umgekehrt druckte das Gerät eingehende Buchstaben nach der Entschlüsselung automatisch aus.

Eine vereinfachte Darstellung der Funktionsweise des Geheimschreibers ist in Abbildung 5–12 zu sehen. Das wichtigste Bauelement waren gezahnte Räder, deren Zähne teilweise gekürzt (nicht aktiv) waren. In der Abbildung sind fünf solcher Räder zu sehen. Die Anzahl der Zähne variiert von Rad zu Rad. Jeweils ein Zahn pro Rad nimmt eine Position ein, in der er »gelesen« wird (Leseposition). Befindet sich ein aktiver Zahn in der Leseposition, dann entspricht dies einer Null, ansonsten einer Eins. Zur Verschlüsselung werden die fünf Bit des

Klartextbuchstabens mit den fünf Bit der Lesepositionen exklusiv-oder-verknüpft. Im Beispiel entsteht so aus der Zahl 11000 die Zahl 11110. Nach der Eingabe eines Buchstabens drehen sich alle fünf Zahnräder um jeweils einen Zahn weiter.

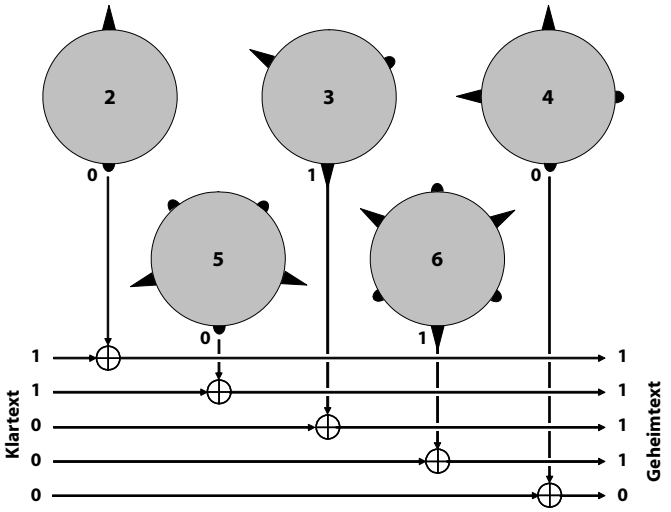


Abb. 5–12 *Vereinfachte Variante eines Geheimschreibers: Fünf gezahnte Räder bilden ein Muster von Nullen und Einsen, die mit dem Klartext exklusiv-oder-verknüpft werden.*

Die vollständige Funktionsweise des Geheimschreibers ist in Abbildung 5–13 ersichtlich. Dabei kommen zehn Räder zum Einsatz. Fünf davon liefern die zum Verschlüsseln verwendeten Bits. Die anderen fünf sorgen für deren Transposition. Die Anzahl der Zähne auf den Rädern beträgt zwischen 47 und 73 und ist so gewählt, dass der kleinste gemeinsame Teiler 1 ist. Dadurch wird eine möglichst lange Periode erreicht. Die Anfangsstellung der Räder entspricht dem Schlüssel.

Wer nun denkt, der Geheimschreiber hätte mehr Sicherheit geboten als die Enigma, der täuscht sich. Der schwedische Mathematiker Arne Beurling, der im neutralen Schweden verschlüsselte Nachrichten aus dem besetzten Norwegen vorliegen hatte, schaffte es, eine frühe Version der Maschine in nur zwei Wochen zu knacken. Er brauchte dazu keine fremde Hilfe und keine Maschinen. Dies gilt neben dem Knacken der Purple als eine der größten Leistungen in der Geschichte der Kryptoanalyse.

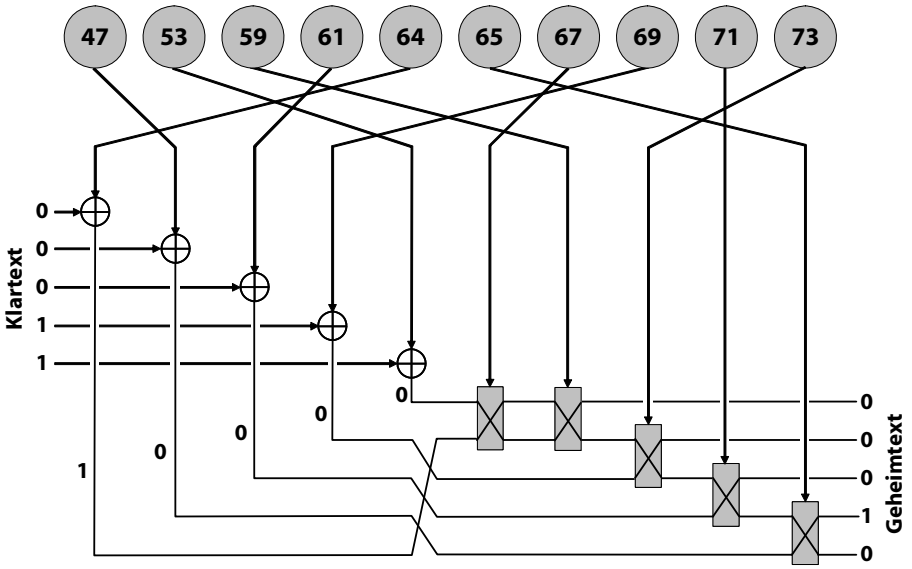


Abb. 5–13 *Der Geheimschreiber arbeitete mit zehn gezahnten Rädern. Fünf davon beeinflussten den Klartext direkt, die anderen fünf sorgten für eine zusätzliche Durchmischung.*

5.2.5 Lorenz-Maschine

Die **Lorenz-Maschine** diente wie der Geheimschreiber der Verschlüsselung von Fernschreiben und verwendete den bereits erwähnten Baudot-Code. Sie kam auf höchster militärischer Ebene zum Einsatz. Im Gegensatz zum Geheimschreiber hatte die Lorenz-Maschine keine Tastatur, sondern wurde an einen Fernschreiber angeschlossen. Ansonsten ähnelte die Funktionsweise der Lorenz-Maschine der des Geheimschreibers. Der Aufbau sah gezahnte Räder mit aktiven und passiven Zähnen vor. Insgesamt gab es 12 Räder, wobei die Zahl der Zähne zwischen 23 und 61 lag (die Räder werden im Folgenden mit der Anzahl ihrer Zähne bezeichnet). Nur die Räder mit 41, 31, 29, 26 und 23 Zähnen drehten sich mit jedem Buchstaben. Rad 37 drehte sich nur dann, wenn die Leseposition von Rad 61 auf 1 stand. Stand die Leseposition von Rad 37 auf 1, dann drehten sich auch die Räder 59, 53, 51, 47 und 43, die den Klartext direkt beeinflussten.

Wie die Enigma und der Geheimschreiber wurde auch die Lorenz-Maschine geknackt. Die Geschichte dieser Kryptoanalyse ist mindestens so spannend wie die der Enigma und wurde ebenfalls von britischen Spezialisten in der Dechiffrier-Fabrik in Bletchley Park durchgeführt. Um das Dechiffrieren zu bewerkstelligen, entwickelten die dortigen Kryptoanalytiker eine spezielle Maschine, die bereits einige Merkmale eines Computers hatte (sie arbeitete mit binären Daten, war jedoch nicht frei programmierbar). Der Name dieser kleiderschrankgroßen

Maschine war **Colossus**. Ein Nachbau dieses technischen Wunderwerks ist heute im Museum von Bletchley Park zu bewundern.

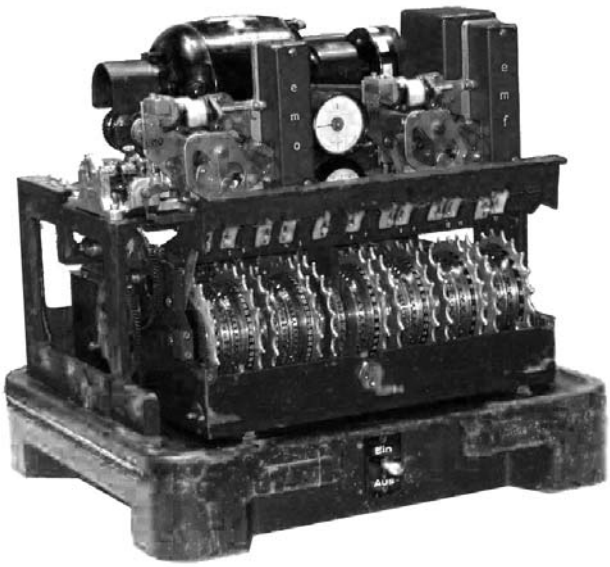


Abb. 5–14 *Die Lorenz-Maschine kam auf höchster militärischer Ebene zum Einsatz. Die Briten konnten sie knacken.*

5.2.6 Schlüsselgerät 41 (Hitler-Mühle)

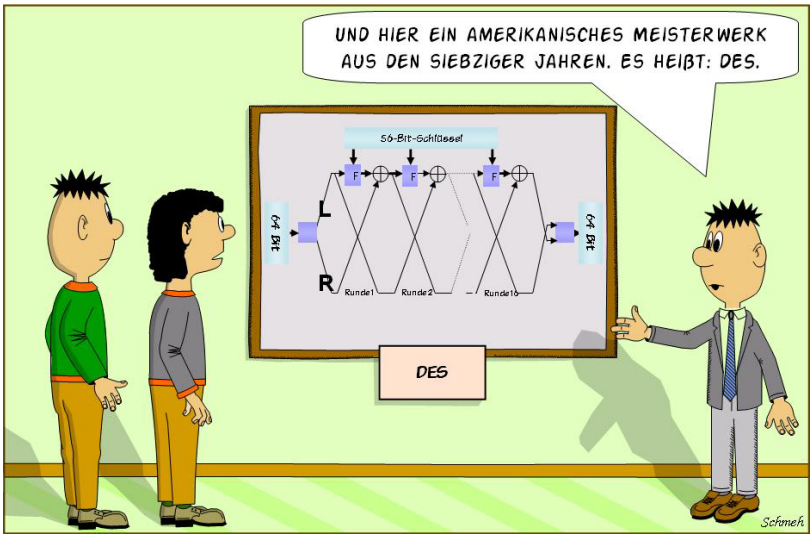
Zu den weniger bekannten deutschen Verschlüsselungsmaschinen aus dem Zweiten Weltkrieg gehört das **Schlüsselgerät 41**, das auch als **Hitler-Mühle** bezeichnet wird. Es entstand 1941, also während des Zweiten Weltkriegs. Ziel der Konstrukteure war es, mit dieser Maschine die damals in der deutschen Armee zu Zehntausenden eingesetzte Enigma zu ersetzen. Zwar wussten die Deutschen nicht, dass die Briten die Enigma zu diesem Zeitpunkt längst geknackt hatten, doch die mangelnde Sicherheit ihrer wichtigsten Verschlüsselungsmaschine war einigen deutschen Kryptologen durchaus bekannt. Außerdem war das Schlüsselgerät 41 im Vergleich zur Enigma handlicher und leichter zu bedienen. Gegen Kriegsende lagen den Wanderer-Werken in Chemnitz, wo das Gerät gebaut wurde, 13.000 Bestellungen vor, allerdings wurden nur 500 davon ausgeliefert. Technisch gesehen war das Schlüsselgerät 41 eng mit den C-Maschinen von Boris Hagelin verwandt. Man kann sogar von einem Ideenklau sprechen. Allerdings bauten die deutschen Entwickler einen zusätzlichen Mechanismus ein, der die Verschlüsselung vermutlich noch sicherer machte. Zweifellos wäre der Zweite Weltkrieg anders verlaufen, wenn die Deutschen früher von der unsicheren Enigma auf diese Maschine umgestellt hätten.

Abb. 5–15 *Das Schlüsselgerät 41 hätte den Zweiten Weltkrieg verlängern können – wenn es früher zum Einsatz gekommen wäre.*

Teil 2

Moderne Kryptografie

6 Der Data Encryption Standard



Nachdem wir bisher nur historisch interessante Verschlüsselungsverfahren betrachtet haben, kommen wir nun zu den Methoden, die heute verwendet werden. Natürlich werden diese vom Computer ausgeführt. Klartext, Geheimtext und Schlüssel sollten Sie sich dabei nicht als Buchstaben oder Zahlen vorstellen, sondern als Bit-Folgen. Zum Auftakt schauen wir uns ein besonders wichtiges Computer-Verschlüsselungsverfahren an, auch wenn dieses inzwischen in die Jahre gekommen ist: den Data Encryption Standard (DES).

6.1 DES-Grundlagen

Der **Data Encryption Standard** (**DES**) wurde Anfang der siebziger Jahre bei IBM in den USA entwickelt. Die US-Standardisierungsbehörde NIST (National Institute of Standards and Technology) machte ihn 1977 zum offiziellen Standard [FIPS-46]. Etwa zwei Jahrzehnte lang blieb der DES das Maß aller Dinge für die

symmetrische Verschlüsselung. Für die Kryptografie bedeutete der DES einen großen Sprung nach vorne, denn mit ihm stand erstmals ein sicheres, computertaugliches Verschlüsselungsverfahren zur Verfügung, das in allen Einzelheiten bekannt, frei von Patentrechten und allgemein akzeptiert war.

Leider waren die Umstände, unter denen der DES entwickelt wurde, reichlich undurchsichtig. Die NSA war an diesem Vorgang beteiligt und sorgte dafür, dass die Designkriterien des DES erst einmal unter Verschluss gehalten wurden – was das Vertrauen in dieses Verfahren nicht unbedingt stärkte. Dennoch zeigte sich im Lauf der Jahre, dass die Entwickler des DES erstklassige Arbeit geleistet hatten. Kryptoanalytiker klopften den DES auf alle erdenklichen Schwachstellen ab und bissen sich dabei ein ums andere Mal die Zähne aus. Nur die geringe Schlüssellänge, von der noch die Rede sein wird, stieß vielen Experten sauer auf.

Im Gegensatz zu den bisher betrachteten Verfahren können Alice und Bob den DES nur mit einem Computer sinnvoll einsetzen – er ist etwas komplizierter als eine Vigenère- oder Rotor-Chiffre. Dennoch ist der DES, dafür dass er praktisch nicht zu knacken ist, bemerkenswert simpel. Deshalb werde ich den DES als Mutter aller modernen Chiffren detailliert erklären. Dabei sollten Sie beachten, dass ich mit diesen Erklärungen weder ein Geheimnis verrate, noch Hackern in die Hände spiele. Die Funktionsweise des DES ist allgemein bekannt, und das mit voller Absicht, denn nur ein weitläufig bekanntes Verfahren kann auch ausgiebig auf Schwachstellen untersucht werden.

Die genauen Gedankengänge der DES-Entwickler sind leider nicht überliefert. In jedem Fall mussten diese sich fragen, welche Art von Verschlüsselung am besten geeignet war. Eine monoalphabetische Substitutionschiffre erschien unsinnig, da eine solche leicht durch eine Häufigkeitsanalyse zu knacken ist. Ein Code-Buch mit vielen Tausend oder gar Hunderttausend Einträgen verbot sich bei den damaligen Kosten für Speicherplatz von selbst. Eine Permutationschiffre hätte einen sehr häufigen Schlüsselwechsel erfordert, weil sonst eine Known-Plaintext-Attacke möglich geworden wäre. Außerdem verändert das Permutieren von Bits die Häufigkeit der Nullen und Einsen nicht, was beispielsweise Rückschlüsse auf die Sprache zulässt. Scheinbar blieb also nur eine polyalphabetische Substitution übrig.

Am Ende entschieden sich die Väter des DES tatsächlich für eine Substitutionschiffre – interessanterweise jedoch nicht für eine polyalphabetische, sondern für eine monoalphabetische. Diese Entscheidung ergab nur Sinn, wenn es gelang, eine Häufigkeitsanalyse zu unterbinden. Tatsächlich fanden die DES-Entwickler einen Weg dazu: Sie ließen ihr Verfahren nicht etwa einzelne Buchstaben, sondern jeweils 64 Bit auf einmal verschlüsseln. Wie man leicht nachrechnet, gibt es etwa 2^{64} (also etwa 10^{19}) verschiedene 64-Bit-Blöcke, und das sind für Bösewicht Mallory schlichtweg zu viele, um praktisch verwertbare Häufigkeitsstatistiken anzufertigen. Die Wahrscheinlichkeit, dass ein 64-Bit-Block in einem Geheimtext doppelt vorkommt, ist selbst bei größeren Datenmengen gering – von größeren

Häufigkeiten, aus denen Mallory auf den Klartext schließen kann, ganz zu schweigen.

Der DES ist für alle Daten geeignet, die in digitaler Form vorliegen. Will Alice beispielsweise einen Text verschlüsseln, der im ASCII-Format gespeichert ist (jedes Zeichen entspricht dabei 8 Bit), dann teilt sie diesen in Blöcke von jeweils acht Buchstaben ein. Sollte der letzte Block weniger als acht Buchstaben aufweisen, dann füllt sie den Rest mit beliebigen anderen Zeichen auf. Auf ähnliche Weise kann sie auch Grafiken, komprimierte Dateien oder Audiodateien verschlüsseln. Jeder 64-Bit-Block Klartext wird vom DES in einen 64-Bit-Block Geheimtext verschlüsselt. Klar- und Geheimtext haben also die gleiche Länge, was eine wichtige Eigenschaft des DES ist. Es ist zwar einfacher, ein sicheres Verschlüsselungsverfahren zu entwickeln, wenn der Geheimtext länger sein darf als der Klartext. Beim DES und fast allen anderen modernen symmetrischen Verfahren wird auf diesen Vorteil jedoch verzichtet, weil man weder Übertragungskapazität noch Speicherplatz verschwenden will. Der Schlüssel des DES-Verfahrens ist wie Klar- und Geheimtext ein 64-Bit-Block. Davon werden jedoch acht Bit als Prüfsumme verwendet, wodurch die wirkliche Schlüssellänge des DES nur 56 Bit beträgt. Deshalb werde ich für den Rest des Buchs im Zusammenhang mit dem DES immer von einem 56-Bit-Schlüssel sprechen.

Intern ist der DES, wenn man so will, eine Kombination aus One-Time-Pad, Permutations- und Substitutionschiffre, die jeweils auf Bitfolgen angewendet werden. Dabei kommen nur die folgenden, ausgesprochen einfachen Funktionen zum Einsatz:

- die Exklusiv-oder-Verknüpfung (siehe Abschnitt 7.1)
- die Permutation (Reihenfolge einer Bit-Folge wird verändert)
- die Substitution (eine Bit-Folge wird durch eine andere ersetzt)

Die Erfahrung der letzten 20 Jahre hat gezeigt, dass für eine wirkungsvolle symmetrische Verschlüsselung keine aufwendigeren Funktionen als diese notwendig sind. Dies hat enorme Vorteile, wenn es beispielsweise um die Realisierung in einer maschinennahen Programmiersprache oder in Hardware geht. Die dem DES zugrunde liegenden Funktionen wurden sogar so gewählt, weil sie effektiv in Hardware implementierbar sind. Mit einer Hardware-DES-Verschlüsselung können Alice und Bob daher hohe Geschwindigkeiten erzielen.

6.2 Funktionsweise des DES

Die Funktionsweise des DES ist in Abbildung 6–1 dargestellt. Das Verfahren sieht vor, dass der zu verschlüsselnde 64-Bit-Block zunächst einmal permutiert wird (sogenannte **Initialpermutation**, abgekürzt IP). Der resultierende 64-Bit-Strom wird dann in zwei Teile L (für links) und R (für rechts) aufgeteilt, die aus je 32 Bit bestehen. Anschließend wird 16-mal derselbe Vorgang wiederholt (man spricht von den 16 Runden des DES):

1. Auf *L* wird die Funktion *F* angewendet. Deren Ergebnis wird mit *R* exklusiv-oder-verknüpft, das Ergebnis dieser Verknüpfung wird zum neuen *L*.
2. *L* wird zum neuen *R*.

Nach Beendigung der 16 Runden werden die 64 resultierenden Bit noch einmal permutiert (**Endpermutation**), und zwar genau umgekehrt wie bei der Anfangspermutation (daher auch die Abkürzung IP^{-1}). Die dadurch entstehenden 64 Bit sind das Ergebnis der Verschlüsselung, also der entstehende Geheimtextblock. Das Entscheidende am beschriebenen Ablauf ist die Funktion *F* (man nennt diese auch **Rundenfunktion**). Diese erzeugt einen Ausgabewert, den wir als zufällig betrachten können. Dieser Ausgabewert wird mit einem Zwischenergebnis der Verschlüsselung exklusiv-oder-verknüpft. Diese Vorgehensweise kennen Sie schon vom One-Time-Pad. In die Funktion *F* geht ein Teil des Schlüssels mit ein, und zwar in jeder Runde ein anderer Teil. Wie diese Teilschlüssel zustande kommen (die sogenannte Schlüsselaufbereitung), werde ich gleich noch erklären.

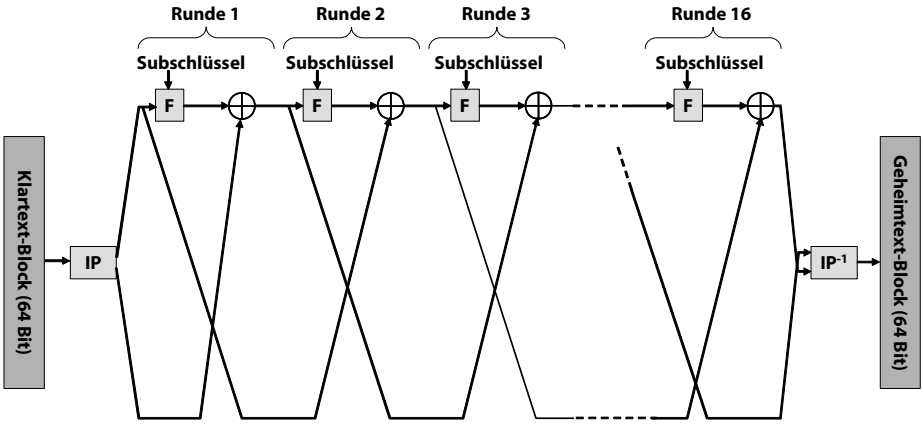


Abb. 6–1 *Der DES sieht 16 Runden vor, in die jeweils ein Subschlüssel eingeht. Der Aufbau der Funktion F wird im nächsten Bild erklärt.*

6.2.1 Die Rundenfunktion *F*

Damit die Funktion *F* möglichst zufällige Bit-Folgen als Ergebnis ausgibt, müssen die Eingaben (32-Bit-Block und Teilschlüssel) auf ausgeklügelte Weise miteinander verknüpft werden. Überraschenderweise besteht *F* jedoch nicht aus vielen verwirrenden Operationen, sondern ist relativ übersichtlich aufgebaut (Abbildung 6–2). Die Funktion *F* arbeitet, wie beschrieben, auf einem Bit-Block der Länge 32. Dieser Block wird zunächst einmal permutiert und durch die doppelte Verwendung einiger Bits auf 48 Bit ausgedehnt. Diese Erweiterung ist vom Schlüssel unabhängig und für sich genommen kryptografisch bedeutungslos. Der Schlüssel kommt nun jedoch ins Spiel, indem der resultierende 48-Bit-Block mit

48 Teil-Bits des Schlüssels exklusiv-oder-verknüpft wird. Diese Verknüpfung ist gleichzeitig die einzige Stelle, an der der Schlüssel in den DES eingeht. Da der DES 16 Runden hat, wird sie 16-mal durchlaufen. Die 48 Bit, die wir nach der Verknüpfung mit dem Teilschlüssel haben, werden anschließend in acht Blöcke zu je 6 Bit aufgeteilt. Jeder 6-Bit-Block bildet eine Eingabe zu einer der Funktionen s_1 bis s_8 (sogenannte **Substitutionsboxen** oder **S-Boxen**). Jede S-Box realisiert eine unterschiedliche Substitution. Zu jedem der 64 möglichen Eingabewerte gibt es einen festgelegten 4-Bit-Ausgabewert, der in einer Tabelle gespeichert ist. Acht Ausgabewerte zu je vier Bit, das macht zusammen 32 Bit, und genau diese bilden den Ausgabewert der Funktion *F*.

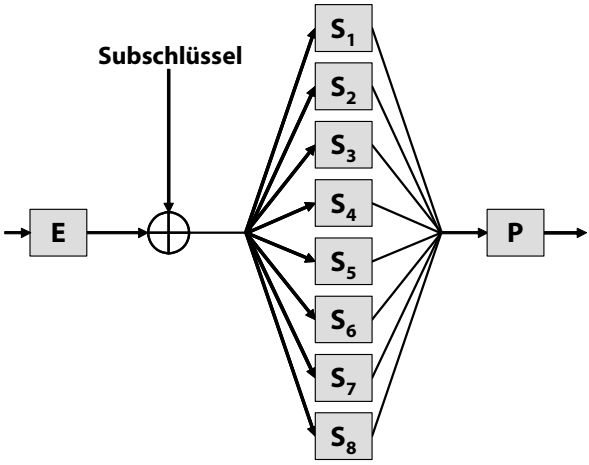


Abb. 6–2 *Die Funktion F ist der wichtigste Teil des DES. Die eingehenden 32 Bit werden auf 48 erweitert und mit einem Subschlüssel exklusiv-oder-verknüpft. Anschließend werden je sechs Bit durch vier neue Bits substituiert und nach einer Permutation ausgegeben.*

6.2.2 Die Schlüsselaufbereitung des DES

Zur vollständigen Beschreibung fehlt jetzt noch die Methode, mit der die 16 48-Bit-Teilschlüssel (Subschlüssel) aus dem 56-Bit-Schlüssel des DES generiert werden (also die **Schlüsselaufbereitung**). Die Schlüsselaufbereitung ist beim DES vergleichsweise einfach, da nur Bit-Verschiebungen und Permutationen verwendet werden. S-Boxen gibt es in der DES-Schlüsselaufbereitung dagegen nicht (im Gegensatz zu vielen anderen Verfahren). Wie bereits beschrieben, besteht ein DES-Schlüssel formal aus 64 Bit, von denen 8 aber nur als Prüfsumme verwendet werden. Der erste Schritt der Schlüsselaufbereitung besteht darin, die Richtigkeit der Prüf-Bits zu kontrollieren und diese anschließend zu löschen oder eine Fehlermeldung auszugeben. Die verbleibenden 56 Bit sind der eigentliche Schlüssel, und daraus müssen 16 Subschlüssel der Länge 48 generiert werden. Es ist also nicht zu vermeiden, dass die Teilschlüssel in hohem Maße voneinander abhängen, was

die Sicherheit zumindest nicht erhöht. Dies hat sich jedoch bisher nicht als Schwäche des DES erwiesen.

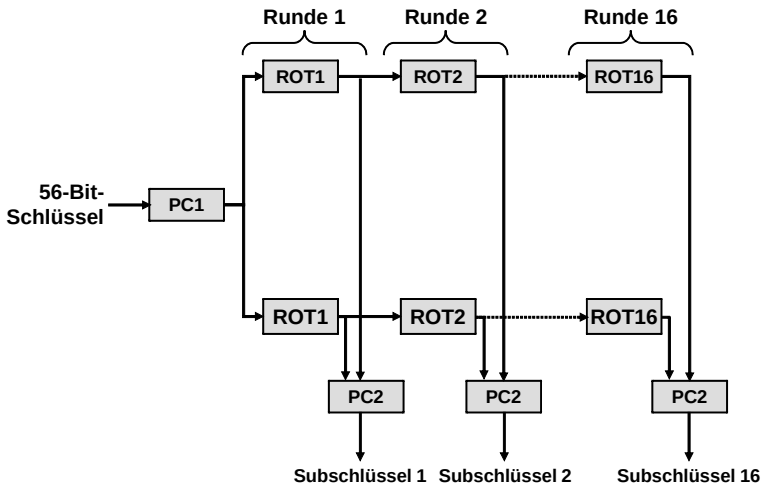


Abb. 6–3 *Die Schlüsselaufbereitung des DES generiert aus einem 56-Bit-Schlüssel 16 Subschlüssel der Länge 48 Bit. PC steht für eine Permutation (Permuted Choice), ROT für eine Linksrotation.*

Wie in Abbildung 6–3 ersichtlich, werden die 56 Schlüssel-Bits des DES nach der Permutation PC1 in zwei Hälften geteilt. Beide Hälften bilden die Eingabe zu einer weiteren Permutation (PC2), deren Ergebnis der Teilschlüssel der ersten Runde ist. In den darauffolgenden Runden entstehen die neuen Hälften dadurch, dass die beiden alten jeweils bitweise rotiert werden, wobei wiederum der Teilschlüssel durch die Permutation PC2 entsteht. Ob um eins oder um zwei rotiert wird, ist in einer Tabelle festgehalten, die Teil der DES-Spezifikation ist.

6.2.3 Entschlüsseln mit dem DES

Von einem guten Verschlüsselungsverfahren erwarten wir nicht nur, dass Mallory ohne Schlüssel keine Chance hat, den Klartext zu ermitteln. Vielmehr ist es ebenso wichtig, dass Alice und Bob bei Kenntnis des Schlüssels schnell und einfach entschlüsseln können. Diese Eigenschaft ist beim DES sehr gut erfüllt, denn die Entschlüsselung funktioniert nahezu gleich wie die Verschlüsselung. Der einzige Unterschied liegt in der Schlüsselaufbereitung: Die 16 Teilschlüssel werden in der umgekehrten Reihenfolge wie beim Verschlüsseln verwendet. Wenn Sie sich den Ablauf des DES in Abbildung 6–1 noch einmal genau anschauen, dann wird Ihnen auffallen, dass diese Umkehrbarkeit des DES unabhängig vom inneren Aufbau der Funktion *F* ist. Es ist also egal, welche 16 Funktionen in der jeweiligen Verschlüsselungsrunde ausgeführt werden – wenn der Geheimtext die 16 Runden ein zweites Mal durchläuft und dabei dieselben 16 Funktionen in umgekehrter Reihenfolge zum Einsatz kommen, dann kommt der Klartext zum Vorschein.

6.3 Sicherheit des DES

Der DES ist ein Musterbeispiel dafür, dass die Offenlegung der Funktionsweise eines Verschlüsselungsverfahrens seine Sicherheit erhöht. In den über 20 Jahren seines Bestehens wurde von Kryptografen alles Erdenkliche versucht, um irgendwelche Schwachstellen im DES zu finden. Die Ergebnisse waren jedoch so bescheiden, dass heute kaum noch Zweifel an der Sicherheit des DES bestehen (abgesehen von der Schlüssellänge). So ist es auch kein Wunder, dass der DES in vielen Bereichen eingesetzt wurde und noch heute wird. Geldautomaten, E-Mails, Online-Banking und Pay-TV-Boxen sind nur einige der Beispiele dafür. Trotz allem gibt es interessante Kryptoanalyse-Resultate zum DES. Darauf werde ich nun näher eingehen.

6.3.1 Vollständige Schlüsselsuche

Die mit 56 Bit relativ kurze Schlüssellänge des DES wurde von Anfang an heftig kritisiert. Schließlich wäre es eine Kleinigkeit gewesen, einen längeren Schlüssel vorzuschreiben. Die Entwickler bei IBM wollten dies auch (sie dachten an 128 Bit), doch sie wurden von der NSA zurückgepfiffen. Diese legte die Schlüssellänge schließlich auf 56 Bit und damit auf einen Wert fest, der eine Brute-Force-Attacke gerade noch realistisch erscheinen ließ – jedenfalls für jemanden, der milliardenteure Hardware zur Verfügung hatte. Schon bald wurde deshalb über die Möglichkeit eines Supercomputers spekuliert, der den DES durch vollständige Schlüsselsuche knacken konnte.

Solche Spekulationen waren Ende der 70er Jahre noch Gedankenspiele. Spätestens mit Beginn der 90er konnte man jedoch davon ausgehen, dass zumindest im Geheimdienst- und Militärumfeld derartige DES-Knackmaschinen im Einsatz waren. Schätzungen besagten, dass sich für einige hundert Millionen Dollar ein Computer bauen ließe, der den DES sogar in Sekundenschnelle knackt. Solche Maschinen sollen sogar in Serie produziert und an Geheimdienste verkauft worden sein.

Währenddessen sorgte der Fortschritt in der Computertechnik dafür, dass auch ein Brute-Force-Angriff mit handelsüblichen Rechnern immer realistischer wurde. Die Firma RSA Data Security setzte eine Prämie von 10.000 US-Dollar für denjenigen aus, der den DES in einer Known-Plaintext-Attacke knacken konnte (der Wettbewerb trug den Namen DES-Challenge, Details dazu gibt es in Abschnitt 40.4.1). Rocke Verser, ein Softwareexperte aus Colorado, entwickelte daraufhin ein Programm, das DES-Verschlüsselungen durchführte und das er über das Internet verteilte. Über 14.000 Internetanwender stellten Rechenzeit zur Verfügung, um jeweils einen Teil des DES-Schlüsselraums zu durchsuchen. Am 18. Juni 1997 war es so weit: Ein Teilnehmer aus Salt Lake City stieß auf den richtigen Schlüssel und durfte sich mit Verser die Prämie teilen. Dessen verteilte Aktion lief bis zu diesem Zeitpunkt seit vier Monaten und hatte erst ein Viertel aller DES-Schlüssel auf

deren Richtigkeit getestet. Mit etwas weniger Glück hätte dieser Kryptoanalyse-Versuch also bis zu 16 Monaten dauern können.

In der zweiten Runde der DES-Challenge sank der Rekord für eine DES-Known-Plaintext-Attacke Anfang 1998 zunächst auf 39 Tage, dann auf 56 Stunden. Im Januar 1999 wurde in der dritten Runde der DES-Challenge erstmals knapp die 24-Stunden-Grenze unterschritten. Dieser Rekord steht bis heute (2012). Wissenschaftler von der Ruhr-Universität Bochum haben in den letzten Jahre zwei Geräte gebaut (Copacobana und dessen Nachfolger Rivyera), mit denen sie in ähnlich kurzer Zeit eine DES-Nachricht per vollständiger Schlüsselsuche knacken können – allerdings mit deutlich geringerem Aufwand [KPPPRS, SciEng]. Ihre beiden DES-Knack-Maschinen bauten sie aus handelsüblicher Hardware zusammen, für die sie jeweils etwa 10.000 Euro bezahlten.

6.3.2 Differenzielle und lineare Kryptoanalyse

Eine von der vollständigen Schlüsselsuche unabhängige Methode, die Mallory auf den DES anwenden kann, ist die **differenzielle Kryptoanalyse**. Sie gehört zu den Chosen-Plaintext-Attacken (Details finden sich in Abschnitt 7.3.1). Es gibt Verschlüsselungsverfahren, bei denen die differenzielle Kryptoanalyse schon mit einigen hundert oder noch weniger gewählten Klartextblöcken funktioniert. Der DES gehört jedoch nicht dazu. Stattdessen muss Mallory beim DES so viele Blöcke verschlüsseln, dass die differenzielle Kryptoanalyse gegenüber der vollständigen Schlüsselsuche keinen Vorteil mehr bringt. Das ist kein Zufall, denn die DES-Entwickler kannten diese Angriffstechnik bereits (obwohl sie damals noch nicht öffentlich bekannt war) und wählten die S-Boxen so, dass diese möglichst wenig Angriffsfläche bieten. Dies ist ein weiterer Beleg dafür, dass die DES-Entwickler erstklassige Arbeit geleistet haben.

1992 erfand der Japaner Mitsuru Matsui die **lineare Kryptoanalyse** (siehe Abschnitt 7.3.2). Mit dieser Methode gelang Matsui 1994 eine erfolgreiche Chosen-Plaintext-Attacke gegen den DES, für die er zwölf Workstations 50 Tage lang beschäftigte. Er benötigte dazu jedoch 2^{43} (also über 1.000.000.000.000) gewählte Klartextblöcke, was etwa der Anzahl der Buchstaben in 150 Millionen Exemplaren dieses Buches entspricht. Das ist zwar deutlich besser als eine vollständige Schlüsselsuche, für Bösewicht Mallory aber dennoch viel zu aufwendig.

6.3.3 Schwache Schlüssel

Von den 2^{56} möglichen DES-Schlüsseln haben vier eine ungewöhnliche Eigenschaft: Verschlüsselt Alice eine Nachricht damit doppelt, dann erhält sie die ursprüngliche Nachricht. Oder anders ausgedrückt: Verschlüsselung und Entschlüsselung verlaufen bei diesen vier Schlüsseln jeweils identisch. Natürlich ist diese Eigenschaft nicht besonders vorteilhaft. Deshalb werden die vier genannten Schlüssel auch **schwache DES-Schlüssel** genannt. Die vier schwachen DES-Schlüs-

sel lauten (in Hexadezimalschreibweise) 00 00 00 00 00 00 00, 00 00 00 0F FF FF FF, FF FF FF F0 00 00 00 und FF FF FF FF FF FF FF. Als echte Schwäche kann man die schwachen DES-Schlüssel allerdings nicht auslegen, denn deren Anzahl ist viel zu gering, um Alice und Bob gefährlich zu werden. Wählen die beiden ihre Schlüssel per Zufall, dann ist die Wahrscheinlichkeit, dass dabei ein schwacher DES-Schlüssel entsteht, kleiner als zwei Sechser im Lotto in Folge. Anstatt zu prüfen, ob ein schwacher DES-Schlüssel vorliegt, ist es für Mallory einfacher, die vier fraglichen Werte per Brute Force durchzuprobieren.

Neben schwachen gibt es auch **semischwache DES-Schlüssel**. Diese haben die Eigenschaft, dass die Verschlüsselung mit einem davon durch die Verschlüsselung mit einem anderen davon rückgängig gemacht werden kann. Zwölf semischwache DES-Schlüssel gibt es (also sechs Paare). Auch hier gilt wieder: Die Wahrscheinlichkeit für einen per Zufall generierten semischwachen Schlüssel ist sehr gering, und für Mallory ist das Durchprobieren der jeweiligen Schlüssel allemal einfacher als das Ausnutzen der Schwäche. Trotzdem gibt es DES-Implementierungen, die schwache und semischwache Schlüssel aussortieren.

6.4 Triple-DES

Da die relativ kurze Schlüssellänge des DES die wichtigste Schwäche des Verfahrens ist, stellt sich folgende Frage: Können Alice und Bob dem Bösewicht Mallory dadurch ein Schnippchen schlagen, dass sie den DES mehrfach hintereinander einsetzen? Die Antwort: Sie können, wenn sie es richtig machen. Die folgenden Betrachtungen beziehen sich alle auf den DES, sie lassen sich aber auf nahezu jedes andere symmetrische Verfahren übertragen.

6.4.1 Doppel-DES

Getreu dem Motto »Doppelt hält besser« besteht die naheliegendste Alternative zum DES darin, den DES doppelt zu verwenden. Dies bedeutet, dass Alice ihre Nachricht an Bob nacheinander zweimal mit dem DES verschlüsselt und dabei zwei verschiedene Schlüssel verwendet. Ist m der Klartext, e die Verschlüsselungsfunktion, c der Geheimtext, und sind $k1$ und $k2$ zwei Schlüssel, dann gilt:

$$e=e_{k1}(e_{k2}(m))$$

Eine solche doppelte DES-Verschlüsselung wäre allerdings sinnlos, wenn es zu zwei beliebigen Schlüsseln *k1* und *k2* je einen Schlüssel *k3* gäbe, für den gilt:

$$e_{k3}(m)=e_{k1}(e_{k2}(m))$$

In diesem Fall könnte man nämlich jede doppelte Verschlüsselung durch eine einfache ersetzen. Mathematisch würde dies bedeuten, dass der DES bezüglich der Hintereinanderausführung von Verschlüsselungen eine Gruppe ist. Einen solchen

Schlüssel *k1* gibt es im Allgemeinen jedoch nicht, und der DES ist deshalb keine Gruppe – dies ist mathematisch bewiesen [CamWie]. Wir können also zunächst einmal davon ausgehen, dass wir mit einer Verdoppelung die Sicherheit des DES tatsächlich erhöhen. Dummerweise ist diese Erhöhung der Sicherheit jedoch längst nicht so stark, wie es zunächst aussieht. Zwar wird durch eine Doppelverschlüsselung die Schlüssellänge auf 112 Bit erhöht, wodurch eine vollständige Schlüsselsuche für Mallory aussichtslos ist. Es gibt jedoch einen Angriff, der die Möglichkeit bietet, die Suche zu verkürzen. Dieser Angriff nennt sich **Meet-in-the-Middle-Attacke** und gehört zur Familie der Known-Plaintext-Attacken.

Die Funktionsweise der Meet-in-the-Middle-Attacke ist recht einfach. Mallory nimmt dazu den (bekannten) Klartext, verschlüsselt ihn mit allen 2^{56} möglichen DES-Schlüsseln und speichert alle Ergebnisse. Anschließend nimmt er den Geheimtext, entschlüsselt ihn mit allen 2^{56} möglichen DES-Schlüsseln und speichert alle Ergebnisse. Nun muss er noch die Ergebnisse der Verschlüsselungsaktion mit denen der Entschlüsselungsvorgänge vergleichen. Wenn alles seine Richtigkeit hat, dann gibt es eine Übereinstimmung (ein »Treffen in der Mitte«), und die beiden DES-Schlüssel, die dazu geführt haben, sind die gesuchten. Sollte es mehrere Übereinstimmungen geben, dann findet Mallory durch Probieren die richtige.

Eine solche Meet-in-the-Middle-Attacke ist recht aufwendig. Mallory benötigt dazu viel Zeit und einen gigantischen Speicher. Geht man jedoch davon aus, dass Mallory genügend Speicherplatz besitzt und er auf diesen beliebig schnell zugreifen kann, dann dauert dieser Angriff auf den Doppel-DES nur etwa doppelt so lange wie die vollständige Schlüsselsuche beim einfachen DES, die bekanntlich nicht ganz unrealistisch ist. Diese Quote ist nicht zu verachten, schließlich dauert eine vollständige Schlüsselsuche beim Doppel-DES immerhin 2^{56} (also fast 10^{17}) Mal länger als beim einfachen DES. Fazit: Wenn Mallory den DES durch vollständige Schlüsselsuche knacken kann, dann liegt auch eine Meet-in-the-Middle-Attacke auf den Doppel-DES im Bereich des Möglichen.

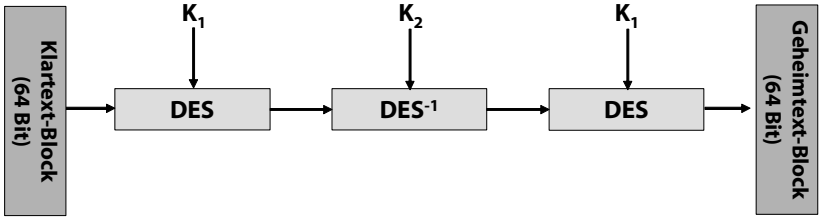


Abb. 6–4 *Beim Triple-DES wird der DES dreimal hintereinander angewendet. Die mittlere Anwendung ist eine Entschlüsselung.*

6.4.2 Triple-DES

Nachdem eine doppelte DES-Verschlüsselung Alice und Bob nicht so recht weiterhilft, können es die beiden mit einer dreifachen versuchen. Und in der Tat, diese Methode erweist sich als deutlich besser. Zwar ist auch der dreifache DES (mit drei verschiedenen Schlüsseln) nicht gegen eine Meet-in-the-Middle-Attacke immun, dafür ist er aber mindestens so sicher, wie man es vom Doppel-DES erwarten würde – und das reicht mehr als aus. Diese dreifache DES-Variante (der sogenannte **Triple-DES** oder 3DES) ist äußerst populär, denn sie verbindet die Sicherheit des herkömmlichen DES mit einer größeren Schlüssellänge.

Da sich das Verschlüsseln beim DES nicht wesentlich vom Entschlüsseln unterscheidet, ändert es nur wenig, wenn man die mittlere der drei Verschlüsselungen des Triple-DES durch eine Entschlüsselung ersetzt. Diese Variante wird in der Praxis am häufigsten verwendet, weil sie einen minimalen Vorteil hat: Benötigt Alice nur eine Einfachverschlüsselung, dann verwendet sie drei gleiche Schlüssel. Eine Ver- und eine Entschlüsselung heben sich dann gegenseitig auf, und übrig bleibt ein einfacher DES. Hier das Ganze noch einmal in mathematischer Schreibweise: Ist e die Verschlüsselungsfunktion, d die Entschlüsselungsfunktion, m die Nachricht, c der Geheimtext, und sind $k1$, $k2$, $k3$ drei DES-Schlüssel, dann gilt:

$$c=e_{k1}(d_{k2}(e_{k3}(m)))$$

Es ist nicht unbedingt notwendig, dass Alice und Bob beim Triple-DES drei verschiedene Schlüssel verwenden. Wenn $k1$ und $k3$ gleich sind, dann ist die Schlüssellänge mit 112 zwar kürzer, reicht aber dennoch aus. Deshalb wird in der Praxis meist der Triple-DES mit zwei Schlüsseln eingesetzt. Die wichtigste Schwäche des DES – die kurze Schlüssellänge – ist damit beseitigt. Dafür gibt es einen naheliegenden Nachteil: Der Triple-DES ist dreimal langsamer als der DES.

6.5 DES-Fazit

Keine Frage, mit dem DES ist den Experten von IBM ein echter Geniestreich gelungen. Obwohl die gesamte Informatik und erst recht die Kryptografie damals noch in den Kinderschuhen steckten, machten die DES-Entwickler kaum Fehler. Sie schufen ein schnelles und sicheres Verfahren, das gegen Angriffe abgesichert war, die erst Jahrzehnte später von anderen entdeckt wurden. Schade nur, dass dieser brillante Algorithmus nur 56 Schlüssel-Bits hat – ein Geburtsfehler, der allem Anschein nach durch eine Intervention der NSA zustande kam und damit nicht den eigentlichen DES-Erfindern anzulasten ist. Dennoch gilt: Aufgrund des kurzen Schlüssels sollte man den DES heute nicht mehr verwenden (außer als Triple-DES).

Neben der geringen Schlüssellänge gibt es noch einige andere Nachteile des DES, die zwar seine Sicherheit nicht beeinträchtigen, aber dennoch ins Gewicht fallen:

- Der DES wurde für Hardwareimplementierungen geschaffen. In Software ist das Verfahren vergleichsweise langsam. Dies gilt beispielsweise für die zahlreichen Permutationen, die der DES verwendet. Permutationen sind in Hardware durch eine geeignete Verdrahtung ohne Performanzverlust zu realisieren, in Software sind sie dagegen vergleichsweise aufwendig. Die beiden Permutationen am Anfang und am Ende sind sogar gänzlich überflüssig, da sie sich ohne Kenntnis des Schlüssels herausrechnen lassen.
- Der DES ist zwar für die Hardware der 70er Jahre optimiert, jedoch nicht für die Hardware des 21. Jahrhunderts. Dadurch könnte das Verfahren auch in Hardware schneller sein.
- Der DES hat eine Blocklänge von 64 Bit. Inzwischen gelten 128 Bit oder eine variable Blocklänge als zeitgemäß (siehe Abschnitt 7.1).

Diese Punkte sind zusammen mit der zu kurzen Schlüssellänge der Grund dafür, dass der DES inzwischen als veraltet gilt und heute nur noch aus Kompatibilitätsgründen eingesetzt wird. In den letzten 20 Jahren sind zahlreiche DES-Alternativen entwickelt worden. Um diese geht es in den folgenden Kapiteln.

7 Chiffren-Design

Dank des DES und zahlreicher weiterer Verfahren, die nach diesem entstanden sind, ist inzwischen recht gut bekannt, wie ein symmetrisches Verschlüsselungsverfahren aufgebaut sein muss, um sicher und praktikabel zu sein. In diesem Kapitel werden wir uns dieses Thema etwas näher anschauen (als weiterführende Lektüre empfehle ich [SKWHFW]). Anschließend stelle ich Ihnen einige Verschlüsselungsalgorithmen aus den achtziger und neunziger Jahren vor, bevor es in späteren Kapiteln um neuere Verfahren geht.

7.1 Chiffren-Design

Ein gutes Verschlüsselungsverfahren zu entwickeln, ist heute zwar um ein Vielfaches leichter als noch vor 25 Jahren, doch trivial ist die Sache noch längst nicht. Erfahrungsgemäß sollte man dafür einige Personenjahre an Arbeit einplanen. Der große Aufwand entsteht nicht zuletzt durch die hohen Anforderungen, die heute an einen symmetrischen Verschlüsselungsalgorithmus gestellt werden.

7.1.1 Anforderungen an die Sicherheit

Es gibt zwei Möglichkeiten, ein Verschlüsselungsverfahren zu entwickeln:

1. Man macht es möglichst kompliziert und hofft, dass dadurch keine Schwachstellen entstehen – oder dass Bösewicht Mallory diese vor lauter Kompliziertheit nicht findet (**Security by Intricacy**).
2. Man macht es möglichst durchdacht und versucht, Schwachstellen gezielt nicht entstehen zu lassen.

Alle in der Fachwelt anerkannten symmetrischen Verschlüsselungsverfahren folgen der letztgenannten Strategie. Security by Intricacy ist dagegen verpönt. Zu viel Unübersichtlichkeit kann schließlich schnell dazu führen, dass der Chiffren-Designer selbst die Übersicht verliert und etwaige Schwachstellen übersieht. Wer also ein Verschlüsselungsverfahren entwickelt, sollte auf einen übersichtlichen Aufbau achten und sich genau darüber im Klaren sein, welcher Bestandteil des Verfahrens welche Aufgabe hat.

Wenig beliebt in Kryptografen-Kreisen ist zudem Geheimniskrämerei. Wer erreichen will, dass sein Verfahren von Fachleuten anerkannt wird und vielleicht sogar eine praktische Verbreitung findet, sollte (dem Kerckhoffs'schen Prinzip folgend) alle Designdetails veröffentlichen. Wie bereits mehrfach erwähnt, gilt ein Verfahren nur dann als sicher, wenn seine Veröffentlichung der Sicherheit nicht schadet. Zwar halten sich Militärorganisationen und Geheimdienste sowie einige Unternehmen nicht an diesen Grundsatz, doch diese haben meist genug Know-how, um ihre Entwicklungen intern auf Schwachstellen prüfen zu können.

Mögliche Schwachstellen

Doch was genau ist bei einem Verschlüsselungsverfahren überhaupt eine Schwachstelle? In der Fachwelt gilt (jedenfalls für symmetrische Verfahren) heute meist der Grundsatz: Es darf keinen besseren Angriff geben als die vollständige Schlüsselsuche. Dies bedeutet insbesondere, dass selbst eine Chosen-Plaintext-Attacke, für die Mallory Terabytes von gewählten Klartextblöcken und Jahrmillionen an Rechenaufwand benötigt, als Schwachstelle gilt. Oft sind die Anforderungen sogar noch höher: Sogar Angriffe, die nur bei einer leicht reduzierten Rundenzahl funktionieren, gelten als Makel.

Darüber hinaus werden auch statistische Auffälligkeiten aller Art bei einem Verschlüsselungsverfahren schon als Schwachstelle betrachtet. Schließlich könnte jedes statistische Ungleichgewicht einen Ansatz für einen Angriff bilden. Ein gutes symmetrisches Verfahren sollte daher ein **Zufallsorakel** (Random Oracle) bilden. Als Zufallsorakel bezeichnet man eine Funktion, bei der kein erkennbarer Zusammenhang zwischen der Eingabe (in diesem Fall Klartext und Schlüssel) und der Ausgabe (in diesem Fall Geheimtext) existiert – auch nicht in wenigen Einzelfällen. Die Zufallsorakel-Eigenschaft ist beispielsweise verletzt, wenn einer der folgenden Fälle gegeben ist:

- Ein Verschlüsselungsverfahren liefert bei Verwendung des Schlüssels 000...0 stets einen Geheimtext, dessen letztes Bit 1 lautet.
- Ein Verschlüsselungsverfahren liefert bei Verwendung des Schlüssels 000...0 zu 60 Prozent Geheimtexte, deren letztes Bit 1 lautet.
- Wenn der Klartext und der Schlüssel invertiert werden, wird auch der Geheimtext invertiert. Dies ist beim DES der Fall. Unter anderem deshalb ist der DES kein perfektes Zufallsorakel.
- Bei bestimmten Schlüsseln ist die Verschlüsselung mit der Entschlüsselung identisch (bei einer doppelten Verschlüsselung entsteht also wieder der Klartext). Auch das ist beim DES der Fall – wenn auch nur bei sehr wenigen Schlüsseln. Auch deshalb ist der DES kein perfektes Zufallsorakel.

Am besten ist die Zufallsorakel-Eigenschaft erfüllt, wenn Alice und Bob den Geheimtext mithilfe einer per Zufallsgenerator erstellten Tabelle aus Klartext und Schlüssel bestimmen. Dieser Goldstandard ist jedoch in der Kryptografie nicht praktikabel, da die Tabelle viel zu groß wäre (bei 64 Bit Block- und Schlüssellänge bräuchten Alice und Bob 2^{128} Tabelleneinträge). Ein Zufallsorakel impliziert, dass die Änderung eines Klartext-Bits im Schnitt die Hälfte aller Geheimtext-Bits verändert (dies nennt man **Avalanche-Effekt**). Darüber hinaus ist der Geheimtext eines Zufallsorakels nicht von einer Zufallsfolge zu unterscheiden.

Nun werden Sie vielleicht einwenden, dass derart hohe Ansprüche an ein Verschlüsselungsverfahren reichlich überzogen wirken. Ein Kryptograf wird Ihnen dann jedoch entgegnen, dass man die Latte immer höher legen sollte, als es in der Praxis erforderlich scheint. Dadurch schafft man sich ein Sicherheitspolster gegenüber Angriffen, die möglicherweise zukünftig entdeckt werden. Es gibt jedoch noch einen weiteren Grund: Die Anforderungen an Verschlüsselungsverfahren sind so hoch, weil es möglich ist, sie zu erfüllen. Es gibt inzwischen zahlreiche Verfahren, bei denen die vollständige Schlüsselsuche die beste bekannte Angriffsmöglichkeit bietet und die auch sonst keine der genannten Schwachstellen aufweisen. Für Alice und Bob ist es daher schlichtweg unnötig, sich mit weniger zufriedenzugeben.

Dass beim Chiffren-Design keine Langeweile aufkommt, liegt nicht zuletzt an der bereits erwähnten Tatsache, dass es außer dem unhandlichen One-Time-Pad bis heute kein Verfahren gibt, das beweisbar sicher ist. Als sicher gilt ein Verschlüsselungsverfahren daher erst dann, wenn intensive Untersuchungen keine Schwachstellen zutage gebracht haben. Selbst die besten Kryptografen können jedoch am Ende keinerlei Sicherheitsgarantie abgeben. Es ist daher immer denkbar, dass ein eben noch als sicher eingestuftes Verschlüsselungsverfahren jeglichen Wert verliert, nachdem ein schlauer Kopf die entscheidende Schwäche entdeckt hat.

Sicherheit gegenüber speziellen Angriffen

Ein gutes symmetrisches Verschlüsselungsverfahren sollte ausreichend lange Schlüssel vorsehen. Auf diese naheliegende Anforderung werde ich in Abschnitt 7.1.2 näher eingehen. Weniger selbstverständlich ist dagegen die Forderung nach einer geeigneten Blocklänge. Eine Blocklänge von 64 Bit kann beispielsweise in den Augen eines paranoiden Kryptografen ein Sicherheitsproblem sein, wie folgende Überlegung verdeutlicht: 64 Bit entsprechen acht ASCII-Zeichen, und es kann durchaus vorkommen, dass sich acht ASCII-Zeichen in einem Klartext wiederholen. Wenn Mallory einen Teil des Klartexts kennt, kann er mit diesem Wissen möglicherweise auf einen ihm unbekannten Teil des Texts schließen (dabei spielt die verwendete Betriebsart eine wesentliche Rolle, siehe Abschnitt 19.1). Die beschriebene Problematik hat dazu geführt, dass die meisten aktuellen Verschlüsselungsverfahren eine Blocklänge von 128 Bit vorsehen.

Eine weitere wichtige Frage beim Chiffren-Design ist die nach schwachen Schlüsseln. Als solche bezeichnet man Schlüssel, die bestimmte Angriffe auf das Verfahren ermöglichen, die normalerweise nicht funktionieren. Möglichkeiten für schwache Schlüssel gibt es viele. Beim DES sind beispielsweise vier Schlüssel bekannt, bei denen eine doppelte Verschlüsselung den Klartext ergibt (siehe Abschnitt 6.3). Darüber hinaus existieren einige weitere DES-Schlüssel mit anderen Schwächen. Es gibt auch Verfahren, bei denen man Schlüssel entdeckt hat, die eine differenzielle Kryptoanalyse zulassen.

In der Praxis sind schwache Schlüssel kein Problem, solange ihre Zahl überschaubar ist. So bringt es Mallory beispielsweise wenig, wenn ein Verfahren 1.000 schwache Schlüssel besitzt. Denn anstatt die damit verbundene Schwäche zu nutzen, kann Mallory genauso die 1.000 schwachen Schlüssel per Brute Force durchprobieren. Die Wahrscheinlichkeit, dass ein solcher Schlüssel per Zufall eingesetzt wird, ist ohnehin minimal. Gefährlich wird es jedoch dann, wenn beispielsweise jeder tausendste Schlüssel eines Verfahrens schwach ist. Dann hat Mallory eine realistische Chance, diese Schwäche nutzen zu können.

Zwei weitere Fragen, die sich jeder Chiffren-Designer stellen muss, sind folgende: Wenn Mallory einen Schlüssel kennt, kann er daraus einen anderen ableiten? Und wenn Mallory einen Teil eines Schlüssels kennt, kann er daraus den fehlenden Teil ermitteln? Man spricht hierbei von einer **Related-Key-Attacke**. Dass eine solche durchaus funktionieren kann, zeigt das WLAN-Protokoll WEP (Abschnitt 33.2.1). Es versteht sich von selbst, dass ein gutes symmetrisches Verschlüsselungsverfahren gegenüber einer Related-Key-Attacke nicht anfällig sein darf.

7.1.2 Die ideale Schlüssellänge

Nehmen wir nun an, wir haben ein symmetrisches Verschlüsselungsverfahren, bei dem die vollständige Schlüsselsuche der beste Angriff ist. Die Sicherheit des Verfahrens hängt nun vor allem von der Schlüssellänge ab. Mit jedem zusätzlichen Schlüssel-Bit verdoppelt sich der Aufwand, den Mallory zum Knacken benötigt. Welchen Schutz eine Schlüssellänge von 56 Bit bietet, haben wir im Kapitel über den DES gesehen: Der aktuelle Rekord für eine vollständige DES-Schlüsselsuche liegt derzeit bei etwa 22 Stunden. Wie sieht es nun bei einer Schlüssellänge von 128 Bit aus? In diesem Fall gelten folgende Überlegungen:

- Es gibt $2^{128} = 3{,}4 \cdot 10^{38}$ Schlüssel.
- Verwendet Mallory einen speziellen Computer, der 100 Milliarden Schlüssel pro Sekunde durchprobieren kann (die in Abschnitt 6.3.1 erwähnten Spezialrechner Copacobana und Rivyera erreichen eine solche Größenordnung), dann dauert das Durchprobieren aller Schlüssel $3{,}4 \cdot 10^{27}$ Sekunden.
- Besitzt Mallory nicht nur einen, sondern 100 dieser Spezialrechner, dann kann er denselben Vorgang in $3{,}4 \cdot 10^{25}$ Sekunden schaffen.
- Hat Mallory großes Glück, dann stößt er schon auf den richtigen Schlüssel, nachdem er nur 1 Prozent aller Schlüssel durchprobiert hat. In diesem Fall benötigt er nur $3{,}4 \cdot 10^{23}$ Sekunden.
- $3{,}4 \cdot 10^{23}$ Sekunden entsprechen etwa 10^{16} Jahren. Zum Vergleich: Das Alter des Universums liegt bei etwa 10^{10} Jahren. Mallory müsste also die Zeit seit dem Urknall eine Million Mal verstreichen lassen, um zum Erfolg zu kommen.

Nun werden Sie vielleicht einwenden, dass Computer immer stärker werden. Vielleicht wird irgendjemand eines Tages einen Computer bauen, der einen 128-Bit-Schlüssel sehr viel schneller knackt. Nehmen wir daher einmal an, Mallory habe einen Superrechner zur Verfügung, der dies könnte. Nehmen wir außerdem an, dieser Superrechner arbeite extrem energiesparend und brauche nur ein Billionstel Joule an Energieaufwand, um einen Schlüssel durchzuprobieren (dies ist mit aktuellen Computern völlig utopisch). Dann ergibt sich folgende Rechnung:

- Ein Billionstel Joule entspricht 10^{-12} Joule.
- Da wir $3{,}4 \cdot 10^{38}$ Schlüssel haben, können wir diese Zahl mit 10^{-12} Joule multiplizieren. Das Ergebnis lautet $3{,}4 \cdot 10^{26}$ Joule. Dies ist der Energieaufwand, den der besagte Rechner für das Durchprobieren aller Schlüssel benötigt.
- $3{,}4 \cdot 10^{26}$ Joule entsprechen $3{,}4 \cdot 10^{26}$ Watt-Sekunden oder $3{,}4 \cdot 10^{23}$ Kilowatt-Sekunden oder $9{,}4 \cdot 10^{19}$ Kilowatt-Stunden.
- Wenn wir davon ausgehen, dass Mallory enormes Glück hat und schon nach einem Hundertstel aller durchprobierten Schlüssel auf den richtigen stößt, dann beträgt sein Energieaufwand $9{,}4 \cdot 10^{17}$ Kilowatt-Stunden.

Zum Vergleich: Die derzeitige Gesamtmenge der jährlich von allen Kraftwerken der Welt produzierten Energie beträgt etwa $8{,}2 \cdot 10^{12}$ Kilowatt-Stunden. Oder anders ausgedrückt: Sämtliche Kraftwerke der Welt müssten über 100 Jahre lang ausschließlich für diesen einen Superrechner arbeiten. Bei einem (sehr günstigen) Preis von einem Cent pro Kilowatt-Stunde würde die Stromrechnung für das gesamte Unterfangen etwa $9{,}4 \cdot 10^{15}$ Euro betragen. Dies ist deutlich mehr als das weltweite Bruttosozialprodukt (etwa $3 \cdot 10^{13}$ Euro). Nebenbei würden die Siliziumatome im Universum knapp werden, wenn tatsächlich eines Tages ein entsprechender Superrechner gebaut werden könnte.

Wir können also festhalten: 128 Bit Schlüssellänge sind per Brute Force nicht zu knacken. Die folgende Tabelle bestätigt diese Erkenntnis. Sie basiert auf der Annahme, dass eine vollständige DES-Schlüsselsuche 24 Stunden dauert (dies entspricht etwa dem aktuellen Weltrekord).

Schlüssellänge	Anzahl der Schlüssel	Dauer einer vollständigen Schlüsselsuche
40 Bit	$1{,}1 \cdot 10^{12}$	1,3 Sekunden
56 Bit	$7{,}1 \cdot 10^{16}$	24 Stunden
64 Bit	$1{,}8 \cdot 10^{19}$	256 Tage
80 Bit	$1{,}2 \cdot 10^{24}$	45.965 Jahre
128 Bit	$3{,}4 \cdot 10^{38}$	$1{,}3 \cdot 10^{19}$ Jahre
192 Bit	$6{,}3 \cdot 10^{57}$	$2{,}4 \cdot 10^{38}$ Jahre
256 Bit	$1{,}2 \cdot 10^{77}$	$4{,}4 \cdot 10^{57}$ Jahre

Obwohl es auf den ersten Blick unnötig scheint, sehen viele neuere Verfahren eine Schlüssellänge von 192 oder gar 256 Bit vor. In solchen extralangen Schlüsseln ist zweifellos ein großzügiger Paranoia-Zuschlag enthalten. Außerdem werden in langfristige Überlegungen häufig Quantencomputer einbezogen (siehe Abschnitt 7.3.3). 256 Schlüssel-Bits sind insofern gerechtfertigt, als sie gegenüber Quantencomputern nur die Sicherheit von 128 Bit bieten.

Wichtiger als die bisher nur theoretisch interessanten Quantencomputer sind allerdings andere Überlegungen. So können besonders lange Schlüssel sinnvoll sein, weil die vollständige Schlüsselsuche in der Praxis eben oft nicht der beste Angriff ist. In einem solchen Fall bedeutet mehr Schlüssel zwar nicht zwangsläufig mehr Sicherheit, viele Angriffe funktionieren aber dennoch bei kürzeren Schlüsseln besser. Außerdem kann es immer Situationen geben, in denen Mallory einen Teil des Schlüssels kennt – je länger der Schlüssel, desto weniger ist eine solche Information wert. Und schließlich gibt es in vielen Bereichen der Kryptografie Geburtstagsangriffe (Abschnitt 14.1.3) und Meet-in-the-Middle-Attacken (Abschnitt 7.2.1), die zu einer Halbierung der effektiven Schlüssellänge führen. Da man bei der Entwicklung eines Krypto-Moduls nie so recht weiß, für welche Anwendungs-

fälle dieses einmal eingesetzt wird, ist es oft sinnvoll, die beiden genannten Angriffe ins Kalkül zu ziehen und die Schlüssellänge doppelt so hoch wie scheinbar notwendig zu wählen.

Übertreiben sollten es Alice und Bob mit der Schlüssellänge allerdings auch nicht. Sie kommen ansonsten schnell mit anderen Anforderungen des Chiffren-Designs in Konflikt (z.B. geringer Speicherplatzbedarf, schnelle Schlüsselaufbereitung). So ist es nicht ratsam, Schlüssellängen von 512 oder gar 1.024 Bit einzusetzen, geschweige denn einige Tausend Bit.

7.1.3 Hintertüren

Vermutlich hat sich jeder Chiffren-Designer schon einmal folgende Frage gestellt: Kann ich in mein Verfahren eine Schwachstelle (**Hintertür**) einbauen, die nur ich kenne und über die ich unerkannt alle mit diesem Verfahren verschlüsselten Nachrichten entschlüsseln kann? Dieser Gedanke wurde erstmals in den siebziger Jahren öffentlich diskutiert, als die Standardisierungsbehörde NIST den DES veröffentlichte. Da sich damals noch kaum jemand einen Reim auf das komplizierte und willkürlich erscheinende Design des DES machen konnte, vermuteten einige bewusst eingebaute Schwachstellen darin. Dieser Verdacht hat sich aber nicht bestätigt.

Eine naheliegende Frage lautet nun: Ist es überhaupt möglich, ein Verfahren mit einer Schwachstelle zu versehen, die andere prinzipiell nicht entdecken können? Nach heutigem Wissensstand ist dies nicht der Fall. Ein Chiffren-Designer, der eine Hintertür einbaut, läuft daher stets Gefahr, entdeckt zu werden – ein Risiko, dem sich wohl kaum jemand aussetzen wird, der sein Verfahren veröffentlicht. Eine Hintertür ergibt also nur dann einen Sinn, wenn die Funktionsweise des Verfahrens geheim gehalten wird oder wenn die Schwachstelle nicht im Verfahren selbst liegt (beispielsweise kann eine Software Schlüssel verwenden, die für Angreifer Mallory leicht zu erraten sind).

Obwohl sich bewusst eingebaute Hintertüren für einen Chiffren-Designer kaum lohnen, sind sie immer wieder ein Thema. Die Entwickler des AES (siehe Kapitel 8) schrieben beispielsweise: »Wir glauben, dass der Aufbau des Verschlüsselungsverfahrens [gemeint ist der AES] nicht genügend Freiheiten lässt, um eine Hintertür einzubauen« [DaeRij97]. Das beste Argument für die Abwesenheit einer Hintertür ist jedoch nicht etwa eine solche Erklärung, sondern ein transparentes und gut dokumentiertes Design – wenn jeder Bestandteil eines Verfahrens eine klar erkennbare und plausibel erklärte Funktion hat, dann bleibt wenig Raum für absichtlich eingebaute Schwachstellen.

Eine besondere Rolle spielen in diesem Zusammenhang Konstanten, die in ein Krypto-Verfahren eingehen. Einige Verschlüsselungsalgorithmen sehen beispielsweise Variablen vor, die mit Konstanten initialisiert werden. Andere Verfahren leiten ihre S-Boxen von Konstanten ab. Das Dilemma hierbei: Kommt als

Konstante ein Wert wie 00000000 oder FFFFFFFF zum Einsatz, dann bringt dies eine Regelmäßigkeit ins Verfahren, die zu einer Schwachstelle führen könnte; verwendet der Chiffren-Designer dagegen unregelmäßige Werte wie D4F87DA5 oder 9021BB3C, dann entsteht der Verdacht, diese Zahlen seien bewusst gewählt. Noch wahrscheinlicher als eine absichtlich eingebaute Hintertür ist dabei eine versteckte Botschaft. Man stelle sich einmal vor, in einem weltweit genutzten Verfahren kommen die Konstanten 4F87DA5 und 9021BB3C vor – und irgendwann stellt sich heraus, dass diese Werte eine rassistische oder obszöne Botschaft enthalten.

Um derartige Fälle von vornherein auszuschließen, verwenden Chiffren-Designer als Konstanten meist sogenannte **Leerer-Ärmel-Zahlen** (Nothing up my sleeve numbers). Als Leerer-Ärmel-Zahl bezeichnet man eine unregelmäßig aufgebaute Zahl, die nicht willkürlich ausgewählt ist. Ein Beispiel hierfür ist die Zahl 314159265 – sie ist einerseits unregelmäßig, andererseits aber klar erkennbar von der Zahl Pi abgeleitet und daher unverdächtig. Es gibt viele weitere Möglichkeiten. Das Verschlüsselungsverfahren Khafre verwendet beispielsweise Zahlen aus dem Buch *A Million Random Digits with 100,000 Normal Deviates* [RAND]. Der Name »Leerer-Ärmel-Zahl« bzw. »Nothing up my sleeve number« ist an einen Zauberkünstler angelehnt, der dem Publikum seine leeren Ärmel zeigt, um zu verdeutlichen, dass er dort nichts versteckt hat.

7.2 Aufbau symmetrischer Verschlüsselungsverfahren

Wäre Sicherheit das einzige Kriterium bei der Entwicklung eines Verschlüsselungsverfahrens, dann wäre die Sache einfach. In diesem Fall würde beispielsweise ein DES-ähnliches Verfahren, das mit mehreren Hundert Runden statt mit den üblichen 16 arbeitet, ausreichen, um Mallory das Handwerk zu legen. Sicherheit ist jedoch nicht das einzige Kriterium, und je mehr sichere Verschlüsselungsverfahren verfügbar sind, desto mehr spielen andere Aspekte des Chiffren-Designs eine Rolle.

Das zweitwichtigste Designkriterium für ein Verschlüsselungsverfahren ist meist die Verschlüsselungsgeschwindigkeit. Wenn Alice einen Algorithmus einsetzt, um eine ganze Festplatte zu verschlüsseln oder um ein Telefongespräch in Echtzeit abzusichern, ist die Wichtigkeit dieses Merkmals offensichtlich. Hinzu kommt, dass viele Verschlüsselungsverfahren nicht auf einem PC, sondern in eingebetteter Hardware oder auf einer Smartcard eingesetzt werden. In solchen Umgebungen gilt: Je performanter das Verschlüsselungsverfahren, desto geringer die Hardwarekosten. Ein Verschlüsselungsverfahren ist im Idealfall auf allen gängigen Plattformen performant zu implementieren und erzielt sowohl in Hardware als auch in Software hohe Geschwindigkeiten. Viele neuere Verschlüsselungsverfahren sind spezialisiert – beispielsweise auf ressourcenschwache Hardwareumgebungen, in denen meist nur kurze Nachrichten verschlüsselt werden.

Nach Sicherheit und Schnelligkeit ist meist der Ressourcenverbrauch das nächste Thema für den Chiffren-Designer. Soll ein Verfahren in vergleichsweise leistungsschwacher Hardware zur Anwendung kommen, dann sollte der Programmcode nicht allzu viel Platz wegnehmen, und der Bedarf an Arbeitsspeicher sollte so gering wie möglich sein. Kommt die Stromversorgung von einer Batterie, dann ist außerdem ein niedriger Energieverbrauch von Bedeutung.

7.2.1 Einfache Operationen

Kryptografie-Einsteiger neigen oft zur Vermutung, ein Verschlüsselungsverfahren müsse aus hochkomplexen mathematischen Funktionen zusammengesetzt sein. In Wirklichkeit ist jedoch das genaue Gegenteil der Fall. Die Entwickler des DES haben es vorgemacht: Der DES setzt sich aus drei der einfachsten Bit-Operationen zusammen, die denkbar sind – aus der Exklusiv-oder-Verknüpfung, der Substitution und der Permutation. Bei den weiteren symmetrischen Verschlüsselungsverfahren, die Sie in diesem Buch kennenlernen werden, verhält es sich kaum anders. Auch diese nutzen die drei genannten Operationen ausgiebig, wobei teilweise noch einige weitere – ebenfalls wenig komplexe – Bit-Funktionen dazukommen. Die folgende Tabelle nennt die wichtigsten Operationen dieser Art:

Zeichen	Name	Beispiel
$\oplus$	exklusives Oder	$1110 \oplus 1011 = 0101$
+	Addition	$1110 + 1011 = 1001$
-	Subtraktion	$1110 - 1011 = 0011$
<<	Linksverschiebung	$1110 << 2 = 1000$
<<<	Linksrotation	$1110 <<< 2 = 1011$
>>	Rechtsverschiebung	$1110 >> 2 = 0011$
>>>	Rechtsrotation	$1110 >>> 2 = 1011$
$\vee$	Oder	$1110 \vee 1011 = 1111$
$\wedge$	Und	$1110 \wedge 1011 = 1010$
||	Konkatenation	$1110 \,\|\| \, 1011 = 11101011$

Die Vorliebe für einfache Operationen beim Chiffren-Design ist einerseits ein Zugeständnis an die geforderte Schnelligkeit. Durch das geschickte Aneinanderreihen einfacher Bit-Operationen lässt sich erfahrungsgemäß mehr Sicherheit pro Taktzyklus erzeugen als mit Logarithmen, trigonometrischen Funktionen oder anderen mathematischen Raffinessen. Der Verzicht auf komplexe Bit-Manipulationen dient andererseits aber auch der Übersichtlichkeit. Substitutionen, Permutationen und die Exklusiv-oder-Verknüpfung sind gut untersucht und lassen Mallory bei geeignetem Einsatz kaum Schlupflöcher.

7.2.2 Linearität

Eine wichtige Rolle in der Kryptografie spielt der Begriff **Linearität**. Anschaulich gesprochen ist eine Funktion linear, wenn ihr Schaubild eine Gerade darstellt (wir betrachten zunächst nur Funktionen mit einer Variablen). Wenn wir mit reellen Zahlen rechnen, dann sind beispielsweise $f(x)=3$, $f(x)=4x$ oder $f(x)=2x+7$ lineare Funktionen. Jede lineare Funktion lässt sich in der Form $f(x)=ax+b$ schreiben. Nicht linear sind zum Beispiel $f(x)=x^2$ und $f(x)=1/x$. In der Kryptografie haben wir es allerdings nicht mit reellen Zahlen, sondern mit Binärzahlen zu tun. Hier ist eine Funktion linear, wenn man sie in der Form $f(x)=Ax+b$ schreiben kann. A ist dabei eine Matrix, die aus Binärzahlen besteht, x und b sind binäre Vektoren (also mehrstellige Binärzahlen).

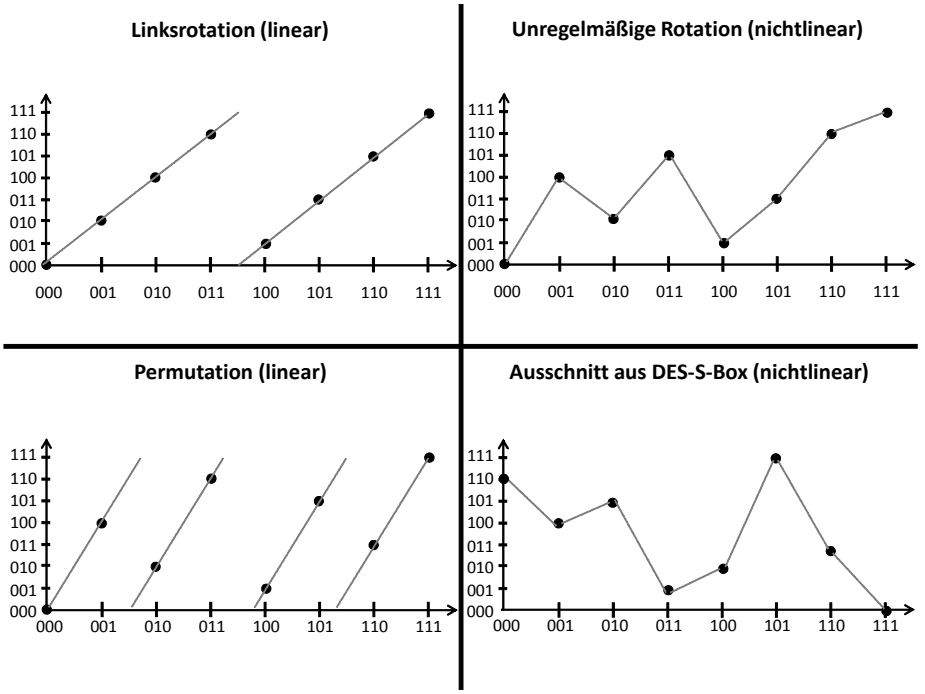


Abb. 7–1 *Bei einer linearen Funktion liegen die Punkte des Schaubilds auf einer Geraden bzw. auf mehreren parallelen Strecken.*

In Abbildung 7–1 sind einige Funktionen dargestellt, deren Ein- und Ausgaben jeweils dreistellige Binärzahlen sind. Die beiden Funktionen auf der linken Seite sind linear. Wie Sie sehen, liegen die Punkte der Schaubilder entweder auf einer Geraden oder auf mehreren parallelen Strecken. Typische lineare Funktionen sind die Permutation (die Linksrotation ist ein Spezialfall davon) und die Exklusiv-

oder-Verknüpfung mit einer Konstanten. Lineare Funktionen spielen in der Mathematik eine wichtige Rolle, da sie regelmäßig und gut zu untersuchen sind. Insbesondere sind lineare Funktionen einfach umkehrbar. Im Übrigen gibt es auch Möglichkeiten, Linearität zu quantifizieren – man kann also beispielsweise von einer »nahezu linearen« Funktion oder von einer »hochgradig nichtlinearen« Funktion sprechen.

Beim Chiffren-Design ist Linearität zunächst einmal nicht erwünscht. Schließlich sollte ein Verschlüsselungsverfahren weder regelmäßig noch einfach umkehrbar sein. In Abbildung 7–1 sind auf der rechten Seite zwei Beispiele für nichtlineare Funktionen ersichtlich: eine unregelmäßige Rotation (alle Ausgangswerte entstehen durch Rotation des Eingangswerts, manchmal wird um ein Bit, manchmal um zwei Bits und manchmal um null Bits rotiert) sowie ein Ausschnitt aus einer DES-S-Box. Auch eine Modulo-Addition ist nicht linear (man beachte, dass die herkömmliche Addition in den reellen Zahlen linear ist, die Modulo-Addition aufgrund des Übertrags aber nicht). Solche nichtlinearen Funktionen lassen sich dank ihrer Unregelmäßigkeit oftmals gut für das Verschlüsseln nutzen. Trotzdem ist Linearität in der Kryptologie alles andere als nutzlos – wie Sie gleich sehen werden, macht gerade das Zusammenspiel aus linearen und nichtlinearen Bestandteilen ein gutes Verschlüsselungsverfahren aus.

Bleibt noch die Frage, wie Linearität definiert ist, wenn nicht nur eine, sondern zwei Variablen in eine Funktion eingehen (der Fall, dass mehr als zwei Variablen im Spiel sind, soll uns nicht interessieren). Für uns genügt es, folgende Fälle zu betrachten (x und y sind die beiden Variablen, die in die Funktion eingehen):

- $x<<<y$: Wenn die Variable x um y Einheiten nach links oder rechts rotiert wird, dann ergibt das Schaubild keine Gerade. Eine solche **datenabhängige Rotation** ist also nicht linear – im Gegensatz zu einer Rotation um einen konstanten Wert.
- $x+y \bmod n$: Da die Modulo-Addition einer Konstanten nicht linear ist, ist es die Modulo-Addition zweier Variablen erst recht nicht.
- $x \oplus y$: Die Exklusiv-oder-Verknüpfung zweier Binärzahlen ergibt stets eine Gerade und ist daher immer linear, auch wenn es sich bei beiden Werten um Variablen handelt.

7.2.3 Konfusion und Diffusion

Praktisch alle gängigen symmetrischen Verschlüsselungsverfahren folgen einem zweiteiligen Grundgedanken. Der erste Teil sieht vor, dass zu verschlüsselnde Daten unkenntlich gemacht werden (**Konfusion**). Beim DES sind die S-Boxen für die Konfusion zuständig. Diese sind so konstruiert, dass Mallory von der Ausgabe möglichst wenig auf die Eingabe schließen kann – insbesondere sind sie hochgradig nichtlinear. Auch alle anderen gängigen Verfahren verwenden S-Boxen oder

ähnliche nichtlineare Konstrukte. In den meisten Fällen sind diese als Ersetzungstabellen gegeben. Teilweise kommen auch mathematische Formeln in S-Boxen zum Einsatz, die jedoch oft vorberechnet und wiederum mithilfe von Ersetzungstabellen umgesetzt werden. Es gibt auch Verfahren, die datenabhängige Rotationen anstelle von S-Boxen nutzen (beachten Sie, dass datenabhängige Rotationen nicht linear sind). Der Vorteil hierbei ist, dass Rotationen ohne Tabellen auskommen, was Speicherplatz spart.

Es ist durchaus möglich, nahezu alleine mithilfe von S-Boxen ein sicheres Verschlüsselungsverfahren zu entwickeln. Allerdings benötigt man dazu sehr große S-Boxen, was nicht praktikabel ist. Daher beschränken sich alle gängigen Verfahren auf kleinere S-Boxen, die mehrfach angewendet werden, und sorgen dazwischen für eine gute Durchmischung. Der zweite Teil des genannten Grundgedankens lautet daher: Zu verschlüsselnde Daten müssen zwischen den S-Boxen gut durcheinandergewirbelt werden (**Diffusion**). Beim DES geschieht dies vor allem durch die beiden Permutationen, die in der Rundenfunktion zum Einsatz kommen (die Permutationen am Anfang und am Ende einer DES-Verschlüsselung sind dagegen kryptografisch unerheblich). Permutationen sind linear. Auch alle anderen gängigen symmetrischen Verschlüsselungsverfahren verwenden die Permutation oder andere lineare Funktionen, um die jeweiligen Daten zu vermischen. Linearität ist das Mittel der Wahl, um eine hohe Diffusion zu erreichen.

Man kann die Sache daher auf einen einfachen Nenner bringen: Ein symmetrischer Verschlüsselungsalgorithmus realisiert Konfusion durch nichtlineare und Diffusion durch lineare Bestandteile. Die Kunst des Chiffren-Designers besteht darin, diese Grundprinzipien auf möglichst wirkungsvolle Weise miteinander zu kombinieren. Darüber hinaus muss dann noch der Schlüssel in das Konstrukt eingebracht werden – in den meisten Fällen erfolgt dies durch eine Exklusiv-oder-Verknüpfung.

7.2.4 Rundenprinzip

Da ein Verschlüsselungsverfahren möglichst wenig Speicherplatz benötigen sollte, arbeiten alle bekannten symmetrischen Algorithmen nach dem Rundenprinzip. Dieses kennen Sie bereits vom DES. Es sieht vor, dass eine Verschlüsselung in Teilschritte (Runden) aufgeteilt wird, die im Wesentlichen identisch ablaufen. Es versteht sich von selbst, dass ein solcher Aufbau bei geeigneter Implementierung weniger Speicherplatz benötigt als eine Vorgehensweise, die zahlreiche unterschiedliche Verschlüsselungsschritte vorsieht. Die Zahl der Runden kann variieren. Beim DES sind es 16, andere Verfahren arbeiten meist mit Rundenzahlen zwischen 8 und 32.

Klar ist, dass in einer Runde mindestens drei Elemente vorhanden sein müssen: ein nichtlineares zur Konfusion (S-Box), ein lineares zur Diffusion sowie eines zur Einbringung eines Rundenschlüssels. Dabei kann es jeweils auch mehrere Elemente geben, die den gleichen Zweck erfüllen (wie die beiden Permuta-

tionen in der Rundenfunktion des DES), oder ein Element mit mehreren Funktionen. Darüber müssen die verschiedenen Bestandteile der Runden so aufgebaut sein, dass Alice und Bob bei Kenntnis des Schlüssels schnell und speicherplatzarm entschlüsseln können.

Die bekannteste Architektur eines symmetrischen Verschlüsselungsverfahrens bilden die sogenannten **Feistel-Chiffren** (auch Feistel-Netzwerke genannt). Dieser Chiffrentyp ist nach dem Kryptografen Horst Feistel benannt, der als IBM-Mitarbeiter an der Entwicklung des DES beteiligt war. Der DES ist die bekannteste Feistel-Chiffre. Wie eine Feistel-Chiffre arbeitet, zeigt ein Blick auf die Funktionsweise des DES (die beiden Permutationen am Anfang und am Ende vernachlässigen wir an dieser Stelle). In jeder Runde wird die eine Hälfte des Blocks einer Rundenfunktion zugeführt, deren Ergebnis mit der anderen Blockhälfte exklusiv-oder-verknüpft wird. Die eigentliche kryptografische Arbeit (Konfusion durch Substitution, Diffusion durch Permutation, Einbringung des Schlüssels) wird in der Rundenfunktion erledigt. Das Interessante an einer Feistel-Chiffre ist, dass die Verschlüsselung gleich funktioniert wie die Entschlüsselung (die Subschlüssel müssen jedoch in umgekehrter Reihenfolge verwendet werden). Diese Eigenschaft ist völlig unabhängig vom inneren Aufbau der Rundenfunktion.

Neben dem DES gibt es noch zahlreiche weitere Feistel-Chiffren. Kein anderer Chiffrentyp ist so weit verbreitet und so gut untersucht. Die wichtigste Alternative ist die Familie der **SP-Chiffren** (auch SP-Netzwerke genannt). SP steht hierbei für Substitution-Permutation. Eine SP-Chiffre führt in jeder Runde – neben der Addition eines Subschlüssels – eine (nichtlineare) Substitution und eine (lineare) Permutation durch, ohne zuvor die bei Feistel-Chiffren übliche Teilung eines Blocks vorzunehmen. Eine SP-Chiffre verfolgt somit im Vergleich zu einer Feistel-Chiffre einen direkteren Ansatz. Der Chiffren-Designer hat hierbei weniger Freiheiten, da die Entschlüsselung bei einer SP-Chiffre (anders als bei einer Feistel-Chiffre) kein Selbstgänger ist. Alle Schritte einer SP-Runde müssen daher so gewählt werden, dass eine Umkehrung bei Kenntnis des Schlüssels einfach möglich ist. Meisterhaft realisiert ist dies etwa beim AES (siehe Kapitel 8).

Neben Feistel- und SP-Chiffren gibt es noch andere Typen von symmetrischen Verschlüsselungsverfahren. Einige davon sind eng mit den beiden genannten verwandt, andere nicht. Ein Designelement, das sich nahezu unabhängig vom sonstigen Aufbau eines Verschlüsselungsverfahrens nutzen lässt, ist **Whitening**. Historisch ist Whitening entstanden, um die geringe Schlüssellänge des DES (56 Bit) zu erhöhen [KilRog] – der DES mit Whitening wird auch DESX genannt. Für das DES-Whitening benötigen Alice und Bob einen 184 Bit langen Schlüssel (dies entspricht der doppelten Block- zuzüglich der Schlüssellänge), mit dem sie wie folgt vorgehen: 64 Bit des Schlüssels werden mit dem Klartextblock exklusiv-oder-verknüpft. Weitere 56 Bit dienen als Schlüssel zur DES-Verschlüsselung des resultierenden Blocks. Die verbleibenden 64 Bit werden mit dem Ergebnis der DES-Verschlüsselung exklusiv-oder-verknüpft (siehe Abbildung 7–2).

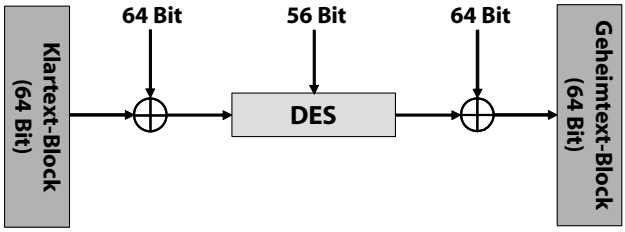


Abb. 7–2 *Whitening bedeutet, dass am Anfang und am Ende des Verfahrens ein Teil des Schlüssels mit dem Klar- bzw. Geheimtext verknüpft wird.*

Whitening hat sich als simple, aber sehr wirksame Technik erwiesen. Viele Chiffren-Designer hat dies so sehr überzeugt, dass sie Whitening als festen Bestandteil in ihre Verfahren eingebaut haben (statt einer Exklusiv-oder-Verknüpfung kommt dabei manchmal auch eine Modulo-Addition zum Einsatz). Dabei wird das Schlüsselmaterial für das Whitening jedoch meist nicht direkt dem Schlüssel entnommen, sondern im Rahmen der Schlüsselaufbereitung aus dem Schlüssel generiert. Zu den Verschlüsselungsverfahren, die Whitening als festen Bestandteil vorsehen, gehören der AES, Blowfish, Twofish, Serpent, PRESENT und viele mehr.

Schlüsselaufbereitung

Die Menge des benötigten Schlüsselmaterials ist bei allen gängigen symmetrischen Verschlüsselungsverfahren deutlich größer als die jeweilige Schlüssellänge (beim DES sind es beispielsweise 768 Bit gegenüber 56 Bit). Deshalb ist es notwendig, den Schlüssel aufzubereiten und Subschlüssel zu bilden. Schon bei der Schlüsselaufbereitung sollte man auf eine gute Diffusion achten, ansonsten drohen Angriffe wie die auf SAFER+ (Abschnitt 9.5) oder MAGENTA (Abschnitt 9.7). Eine andere Frage lautet, ob eine Schlüsselaufbereitung auch Konfusionselemente (also nichtlineare Bestandteile) enthalten soll. Interessanterweise sind sich die Chiffren-Designer in diesem Punkt nicht einig. Die DES-Entwickler verzichteten darauf. Stattdessen pickt sich der DES in jeder Runde 48 aus den 56 Bit des Schlüssels heraus. Viele andere Verfahren verwenden dagegen S-Boxen oder andere komplexere Abläufe zur Gewinnung der Subschlüssel. Dennoch konnte bisher noch niemand nachweisen, dass die minimalistische Schlüsselaufbereitung des DES ein Nachteil ist. Einzige Ausnahme: Je aufwendiger die Schlüsselaufbereitung, desto aufwendiger die vollständige Schlüsselsuche.

Davon abgesehen gilt auch für die Schlüsselaufbereitung: Sicherheit ist nicht das einzige Kriterium. Wenn ein Schlüssel häufig geändert wird, dann ist auch die Geschwindigkeit ein wichtiger Punkt. Eine einfache und schnelle Schlüsselaufbereitung, wie sie der DES bietet, ist also wünschenswert. Davon abgesehen sollte ein Chiffren-Designer anstreben, dass die Schlüsselaufbereitung nicht allzu viel Speicherplatz in Anspruch nimmt. Dies wird meist dadurch erreicht, dass bei der

Aufbereitung des Schlüssels die gleichen Designelemente (z.B. die gleichen S-Boxen) wie bei der Verschlüsselung verwendet werden.

7.3 Kryptoanalyse-Methoden

Zahlreiche Kryptografen haben in den letzten Jahrzehnten nicht nur für Alice und Bob gearbeitet, sondern auch für Mallory. Mit anderen Worten: Neben vielen Verschlüsselungsverfahren sind auch einige gute Methoden zur Kryptoanalyse entstanden. Um Letztere geht es im Folgenden. Fast immer handelt es sich dabei um Verfahren für Chosen-Plaintext- oder zumindest für Known-Plaintext-Attacken, während Ciphertext-Only bei modernen kryptografischen Verfahren kaum noch funktioniert. Und nicht vergessen: Alles, was besser funktioniert als die vollständige Schlüsselsuche, gilt für die betroffenen Verfahren schon als Schwäche.

7.3.1 Differenzielle Kryptoanalyse

Die **differenzielle Kryptoanalyse** ist die bekannteste und in vielen Fällen wirksamste Methode, die Mallory gegen moderne symmetrische Verschlüsselungsverfahren anwenden kann. Sie wurde 1990 von den israelischen Kryptografen Eli Biham und Adi Shamir entwickelt [BihSha] und gehört zu den Chosen-Plaintext-Attacken. Um zu erklären, wie die differenzielle Kryptoanalyse funktioniert, betrachten wir das (für die Praxis natürlich völlig ungeeignete) Mini-Verschlüsselungsverfahren in Abbildung 7–3. Das Verfahren hat eine Block- und Schlüssellänge von jeweils 3 Bit. Es arbeitet mit einer S-Box, die 3-Bit-Werte auf 3-Bit-Werte abbildet. Es gibt nur eine Runde.

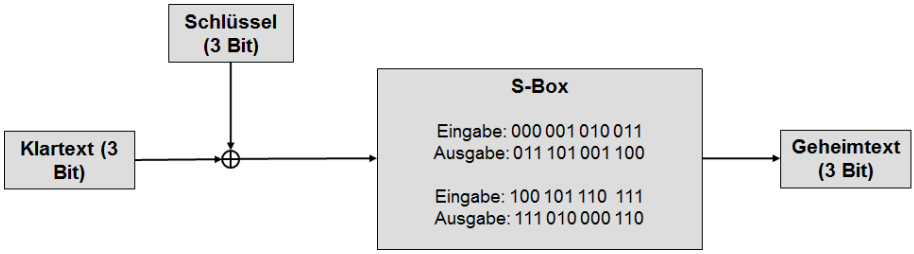


Abb. 7–3 *Dieses Ein-Runden-Verschlüsselungsverfahren lässt sich mit differenzieller und linearer Kryptoanalyse brechen.*

Schauen wir uns zunächst die S-Box an. Für diese gibt es acht mögliche Eingaben (die Binärzahlen von 000 bis 111). In der differenziellen Kryptoanalyse spielen sogenannte Differenzen eine wichtige Rolle – als Differenz zweier Bitfolgen bezeichnet man hierbei das Ergebnis deren Exklusiv-oder-Verknüpfung (dies ist etwas verwirrend, da die Exklusiv-oder-Verknüpfung oft auch als Addition bezeichnet wird). Die folgende Tabelle zeigt die acht Eingabewerte der S-Box, den jeweils

zugehörigen Wert mit der Differenz 110 (dies ist die sogenannte Eingangsdifferenz) sowie die Differenzen nach der Anwendung der S-Box (Ausgangsdifferenzen):

1. Eingabewert	000	001	010	011	100	101	110	111
2. Eingabewert	110	111	100	101	010	011	000	001
Eingangsdifferenz	110	110	110	110	110	110	110	110
1. Ausgabewert	011	001	000	100	010	101	111	110
2. Ausgabewert	111	110	010	101	000	100	011	001
Ausgangsdifferenz	100	111	010	001	010	001	100	111

Wie Sie sehen, sind die Ausgangsdifferenzen ungleich verteilt. So kommen 001, 010, 100 und 111 jeweils doppelt vor, die Werte 000, 011, 101 und 110 dagegen gar nicht. Interessant ist außerdem folgende Beobachtung: Wenn wir statt der S-Box das gesamte Verfahren betrachten (also inklusive der Exklusiv-oder-Verknüpfung mit dem Schlüssel), dann kommen bei einer Eingangsdifferenz von 110 genau die gleichen Ausgangsdifferenzen (nämlich 001, 010, 100 und 111) in gleicher Häufigkeit (jeweils doppelt) vor. Dies liegt daran, dass sich eine Differenz nicht ändert, wenn die beiden beteiligten Werte mit dem gleichen Wert exklusiv-oder-verknüpft werden. Wenn wir statt 110 eine andere Eingangsdifferenz betrachten, sieht das Ergebnis ähnlich aus: Die Ausgangsdifferenzen kommen unterschiedlich häufig vor (manche gar nicht), und an den Häufigkeiten ändert sich nichts, wenn der Schlüssel wechselt.

Mit diesem Wissen kann Mallory einen ersten Angriff (eine Chosen-Plaintext-Attacke) auf unser Ein-Runden-Verfahren starten. Dazu verschlüsselt er zwei Klartexte mit Eingangsdifferenz 110 (da es sich um eine Chosen-Plaintext-Attacke handelt, kann Mallory eine solche Verschlüsselung veranlassen). Nehmen wir an, er erhält als Ergebnis (und damit als Ausgangsdifferenz) 100. Dann weiß er, dass entweder das Paar 000/100 oder das Paar 100/000 in die S-Box eingegangen ist, da nur diese beiden die Ausgangsdifferenz 100 generieren. Zu jedem der beiden Paare kann Mallory einen Schlüsselkandidaten ermitteln. Durch Ausprobieren findet er heraus, welcher der richtige ist. Dieser Angriff erfordert drei bis vier Arbeitsschritte. Er ist nicht unbedingt effektiver als eine vollständige Schlüsselsuche (es gibt acht Schlüssel), doch das wird sich bei den folgenden, komplexeren Beispielen ändern.

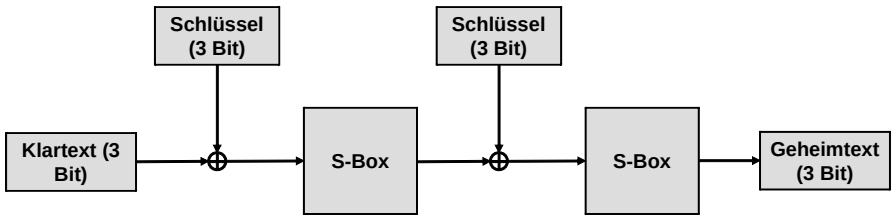


Abb. 7–4 *Auch bei einem Zwei-Runden-Verfahren funktioniert die differenzielle Kryptoanalyse.*

Als Nächstes betrachten wir das Verschlüsselungsverfahren in Abbildung 7–4. Dieses verwendet die gleiche S-Box wie das zuvor behandelte, hat aber zwei Runden. Pro Runde wird ein Subschlüssel der Länge 3 Bit eingebracht, es gibt also $2^6=64$ Schlüssel. Im Rahmen einer differenziellen Kryptoanalyse verschlüsselt Mallory zwei Klartexte mit der Differenz 110. Nehmen wir an, er erhält als Ergebnis die Differenz 111. Mallory weiß, dass die Ausgangsdifferenz der ersten S-Box 001, 010, 100 oder 111 betragen muss, da die Eingangsdifferenz 110 nur diese vier Werte erzeugt. Durch Ausprobieren oder Nachschauen in einer vorberechneten Tabelle findet Mallory außerdem heraus, dass nur die Werte 110, 100, 011 und 001 als Eingangsdifferenzen der zweiten S-Box infrage kommen, da nur diese die Ausgangsdifferenz 111 erzeugen. Durch einen Abgleich der Ausgangsdifferenzen der ersten und der Eingangsdifferenzen der zweiten S-Box kann Mallory erkennen, dass die Differenz auf dem Weg von der ersten zur zweiten S-Box 001 oder 100 beträgt (die Exklusiv-oder-Verknüpfungen mit dem Schlüssel ändern wiederum nichts an den Differenzen). Zu jeder der beiden Differenzen gibt es zwei Paare, die sie erzeugen. Außerdem gibt es jeweils zwei Eingangsdifferenzen der ersten S-Box, die sie erzeugt haben können, und zu diesen wiederum zwei Paare. Insgesamt haben wir damit acht Kombinationen, die Mallory durchprobieren muss, um den richtigen Schlüssel zu finden. Gegenüber einer vollständigen Schlüsselsuche bei 64 Schlüsseln ist das schon ein Vorteil.

Die differenzielle Kryptoanalyse lässt sich in ähnlicher Form auch auf ein Drei-Runden-Verschlüsselungsverfahren übertragen. Die Schlüssellänge beträgt hier 9 Bit, wodurch es 512 Schlüssel gibt. Durch das Suchen von Differenzkombinationen kann Mallory den Schlüssel in 10–20 Arbeitsschritten ermitteln, was bei 512 Schlüsseln ein deutlicher Vorteil gegenüber Brute Force ist.

In der Praxis hat es Mallory allerdings mit deutlich komplexeren Verfahren zu tun. Der DES hat beispielsweise 16 Runden, acht verschiedene 6×4-S-Boxen und pro Runde einen 48-Bit-Subschlüssel. Die differenzielle Kryptoanalyse ist dennoch auch hier anwendbar (eine gute Erklärung dazu findet sich in [Heys]). Dazu muss Mallory zunächst die S-Boxen untersuchen und die Verteilung von Eingangs- sowie Ausgangsdifferenzen betrachten. Anschließend kann er mithilfe gewählter Klartexte nach Differenzen suchen, die sich über die Runden hinweg fortpflanzen. Die Einbringung des Schlüssels per Exklusiv-oder-Verknüpfung ändert wiederum die Differenzen nicht. Die Permutationen des DES kann Mallory bei der Verfolgung der Differenzen leicht herausrechnen, da diese nur die Reihenfolge der Bits verändern. aufgrund der vielen Runden kann Mallory normalerweise keine eindeutigen Differenzenpfade ermitteln, die sich vom Klartext bis zum Geheimtext erstrecken, doch er kann Varianten ermitteln, aus denen er durch Probieren die richtige findet. In vielen Fällen findet Mallory den Schlüssel auf diese Weise deutlich schneller als durch eine vollständige Suche.

Die differenzielle Kryptoanalyse wurde in den neunziger Jahren so manchem neu entwickelten Verschlüsselungsverfahren zum Verhängnis. Als besonders anfällig erwies sich das DES-ähnliche Verfahren FEAL, dessen ursprüngliche Ver-

sion mit nur acht gewählten Klartext-Geheimtext-Paaren zu knacken ist [Schn06]. Auch das etwa zur gleichen Zeit entstandene Verfahren Khafre kann Mallory mit differenzieller Kryptoanalyse angreifen, und zwar mit etwa 1.500 Paaren (in der 16-Runden-Version) [Schn06]. Der Rechenaufwand ist in beiden Fällen überschaubar. Der DES erwies sich dagegen überraschenderweise als weitgehend immun gegenüber der differenziellen Kryptoanalyse. Zwar ist ein solcher Angriff möglich, doch der Aufwand ist etwa so groß wie bei der vollständigen Schlüsselsuche. Dies liegt daran, dass die Entwickler des DES die differenzielle Kryptoanalyse kannten – 20 Jahre bevor sie von Shamir und Biham wiedererfunden wurde.

Nach Bekanntwerden der differenziellen Kryptoanalyse reagierten die Chiffren-Designer recht schnell auf die neue Bedrohung. Praktisch alle Verfahren, die ab Mitte der neunziger Jahre entwickelt wurden, sind so gestaltet, dass sie diesem Angriff widerstehen. Die DES-Entwickler haben gezeigt, wie es geht: Man gestaltet die S-Boxen so, dass die Ausgangsdifferenzen möglichst gleich häufig vorkommen. Wie man dies erreicht, ist inzwischen in zahlreichen Forschungsarbeiten beschrieben.

7.3.2 Lineare Kryptoanalyse

1992 erfand der Japaner Mitsuru Matsui die **lineare Kryptoanalyse** [Mats93]. Diese Technik aus der Familie der Known-Plaintext-Attacken ist nach der differenziellen Kryptoanalyse die zweitwichtigste Angriffsmethode auf symmetrische Verschlüsselungsverfahren. In der linearen Kryptoanalyse geht es darum, in den nichtlinearen Teilen (also meist in den S-Boxen) eines Verfahrens lineare Bestandteile ausfindig zu machen. Durch geschickte Nutzung dieser linearen Bestandteile kann Mallory in vielen Fällen einzelne Schlüsselbits ermitteln und gegebenenfalls den Rest per vollständiger Schlüsselsuche herausfinden. Eine etwas ausführlichere Betrachtung als in diesem Buch findet sich in [Heys].

Wenn wir wieder das Verfahren in Abbildung 7–3 als erstes Beispiel verwenden, dann benötigt Mallory für eine lineare Kryptoanalyse Gleichungen, bei denen die Exklusiv-oder-Verknüpfung eines (oder mehrerer) Klartext-Bits sowie eines (oder mehrerer) Schlüsseltext-Bits mit einem (oder der Exklusiv-oder-Verknüpfung mehrerer) Schlüsselbits gleichgesetzt werden. Die folgende Gleichung ist ein Beispiel dafür (m ist der Klartext, c der Geheimtext und k der Schlüssel, wobei die Suffixe das jeweilige Bit anzeigen):

$$m_1 \oplus m_3 \oplus c_1 = k_1 \oplus k_2$$

Weitere Möglichkeiten wären $m_1 \oplus c_1 \oplus c_3 = k_1$ oder $m_1 \oplus m_2 \oplus m_3 \oplus c_1 = k_1 \oplus k_3$. Wie man leicht nachrechnet, gibt es 343 Gleichungen dieser Art. Da insgesamt neun Variablen im Spiel sind ($m_1, m_2, m_3, c_1, c_2, c_3, k_1, k_2, k_3$), sind $2^9 = 512$ Belegungen möglich. Mit einem Computerprogramm kann Mallory nun prüfen, wel-

che der 343 Gleichungen besonders viele Belegungen hat, bei denen die Gleichung stimmt. Wenn beispielsweise herauskommt, dass die Gleichung $m_1 \oplus m_3 \oplus c_1 = k_1$ in 448 von 512 Belegungen (also in 87,5 Prozent der Fälle) korrekt ist, dann bedeutet das für Mallory: Mit dem ersten und dem dritten Bit des Klartexts (der Klartext ist ihm bekannt, da wir von einer Known-Plaintext-Attacke ausgehen) sowie mit dem ersten Geheimtext-Bit kann er mit 87,5 Prozent Erfolgswahrscheinlichkeit das erste Schlüsselbit berechnen. Der Aufwand dafür ist minimal, da Mallory nur eine Berechnung durchführen muss.

Wenn Mallory geeignete Gleichungen findet (man spricht auch von einer linearen Approximierung), dann kann er auch die beiden anderen Schlüsselbits ermitteln. Eventuell lohnt es sich für ihn, für ein Schlüsselbit mehrere Gleichungen anzuwenden, um die Wahrscheinlichkeit für ein korrektes Ergebnis zu erhöhen. Wenn auf der rechten Seite einer Gleichung kein einzelnes Bit steht, sondern beispielsweise $k_1 \oplus k_2$, dann muss Mallory möglicherweise ein kleines Gleichungssystem lösen. Fehlende Schlüsselbits kann er bei Bedarf auch per Brute Force ermitteln.

Mit diesem Wissen kann Mallory auch komplexere Verschlüsselungsverfahren lösen. Hierzu muss er für jede Runde möglichst gute lineare Approximierungen finden und diese anschließend zu möglichst guten linearen Approximierungen des gesamten Algorithmus zusammensetzen. Dabei helfen ihm auch Gleichungen, die nicht besonders häufig, sondern besonders selten korrekt sind. In der Praxis muss sich Mallory am Ende oft mit linearen Approximierungen zufriedengeben, die nur in knapp über (bzw. knapp unter) 50 Prozent der Belegungen korrekt sind. Dementsprechend kann eine lineare Kryptoanalyse recht aufwendig sein.

Mallory kann die lineare Kryptoanalyse auch auf den DES anwenden – sogar deutlich effektiver als die differenzielle Kryptoanalyse. Dem Erfinder der Methode, Mitsuru Matsui, gelang 1994 ein erfolgreicher Angriff gegen den DES, für den er zwölf Workstations 50 Tage lang beschäftigte. Er benötigte dazu jedoch 2^{43} (also über 1.000.000.000.000) Klartextblöcke – das entspricht etwa der Anzahl der Buchstaben in 150 Millionen Exemplaren dieses Buches. Das ist zwar deutlich besser als eine vollständige Schlüsselsuche, in der Praxis jedoch nicht anwendbar.

Gute Chiffren-Designer haben sich bereits seit den neunziger Jahren auf die lineare Kryptoanalyse eingestellt, weshalb moderne Verfahren stets mit entsprechenden Gegenmaßnahmen ausgestattet sind. Dazu muss der Chiffren-Designer die S-Boxen so gestalten, dass sie möglichst wenig lineare Bestandteile enthalten. Wie das geht, ist in der Literatur gut dokumentiert. Es ist durchaus möglich, S-Boxen zu konstruieren, die sowohl gegen die differenzielle als auch gegen die lineare Kryptoanalyse sicher sind.

7.3.3 Kryptoanalyse mit Quantencomputern

Wer sich schon einmal mit moderner Physik (insbesondere der Quantenmechanik) beschäftigt hat, weiß, dass Atome und Elementarteilchen teilweise seltsame Eigenschaften haben. So können bestimmte Teilchen mehrere gegensätzliche Zustände gleichzeitig einnehmen, und manche Zustände sind erst in dem Moment festgelegt, in dem man sie misst. Es gibt zwei für die Kryptografie interessante Entwicklungen, welche die Quantenmechanik nutzen. Die eine ist die Quantenkryptografie, um die es an dieser Stelle nicht gehen soll. Die zweite Entwicklung sind die sogenannten **Quantencomputer**. Die Architektur eines Quantencomputers nutzt quantenmechanische Gesetze, durch die Algorithmen möglich sind, die auf einem herkömmlichen Computer nicht funktionieren. Bisher gibt es zwar noch keine praxistauglichen Quantencomputer, doch theoretische Überlegungen dazu wurden schon viele angestellt.

Zu den Algorithmen, die ein Quantencomputer ausführen kann, sind für uns vor allem zwei von Bedeutung, da diese dem Knacken von Verschlüsselungen dienen. Der wirksamere der beiden ist der Shor-Algorithmus, um den es in Abschnitt 11.3.3 geht (für symmetrische Verschlüsselungsverfahren spielt dieser keine Rolle). Der zweite ist der **Grover-Algorithmus** [Grover]. Diesen kann Mallory für die vollständige Schlüsselsuche bei symmetrischen Verschlüsselungsverfahren nutzen. Mit dem Grover-Algorithmus muss Mallory nur $2^{n/2}$ Schritte durchführen, um 2^n Schlüssel zu prüfen – eine Ersparnis, die nur durch die Nutzung der Quantenmechanik möglich ist. Mit anderen Worten heißt dies: Wenn sich Alice und Bob gegen eine vollständige Schlüsselsuche mit dem Quantencomputer schützen wollen, dann müssen sie die Schlüssellänge doppelt so lang wählen, wie sie es ohne Quantencomputer tun würden.

Organisationen, die beim Verschlüsseln eine hohe und langfristige Sicherheit benötigen, verwenden oft besonders lange Schlüssel, weil sie die Möglichkeiten von Quantencomputern einkalkulieren. Dabei ist jedoch völlig offen, ob es jemals brauchbare Quantencomputer geben wird. Die bisher realisierten Exemplare können nur mit einzelnen Bits umgehen und sind noch weit davon entfernt, für nützliche Berechnungen geeignet zu sein. Alice und Bob müssen sich also vorerst keine Sorgen machen.

7.3.4 Weitere Kryptoanalyse-Methoden

1994 zeigten die Kryptologen Langford und Hellman, dass man die differenzielle und die lineare Kryptoanalyse miteinander verbinden kann – die **differenzielle-lineare Kryptoanalyse** war geboren [LanHel]. Daneben gibt es auch die Möglichkeit, statt häufiger Differenzen nichtvorkommende Differenzen zu Hilfe zu nehmen. Diese Methode heißt auf Englisch »impossible differential cryptanalysis« [BiBiSh], auf Deutsch kann man sie als **Kryptoanalyse mit unmöglichen Differen-**

zen bezeichnen (die naheliegende Übersetzung »unmögliche differenzielle Kryptoanalyse« ist dagegen unsinnig).

Eine völlig andere Variante der Kryptoanalyse kam im Jahr 2002 auf und verursachte einiges an Wirbel: die **quadratische Kryptoanalyse** (auch als XSL-Angriff bekannt) [CouPie]. Diese sieht vor, dass Mallory ein Verschlüsselungsverfahren mit einem System quadratischer Gleichungen nachbildet. Gelingt ihm dies, dann kann er versuchen, dieses Gleichungssystem zu lösen. Interessanterweise benötigt Mallory nur ein paar einzelne Klartext-Geheimtext-Paare, um so vorzugehen, während andere Angriffe oft astronomische Mengen an Analysematerial voraussetzen. Nun gibt es zwar Methoden zum Lösen quadratischer Gleichungssysteme – zum Beispiel die XSL-Methode (Extended Sparse Linearisation). Doch ganz so einfach ist die Sache trotzdem nicht. Die Zahl der Gleichungen und Variablen geht bei der quadratischen Kryptoanalyse nämlich schnell in die Tausende, was in der Praxis nicht mehr handhabbar ist. Bisher hat es daher noch niemand geschafft, auf diese Weise auch nur in die Nähe eines erfolgreichen Angriffs zu kommen. In den letzten Jahren ist es um die quadratische Kryptoanalyse ziemlich ruhig geworden.

Eine weitere Form der Kryptoanalyse stellten Adi Shamir und Itai Dinur im Jahr 2008 vor [ShaDin]. Sie wird als **Cube-Attacke** bezeichnet. Eine Cube-Attacke sieht vor, dass Angreifer Mallory ein Verfahren mit Polynomen niederen Grades nachbildet und auf dieser Grundlage die einzelnen Bits des Schlüssels berechnet. Erste Untersuchungen deuten zwar darauf hin, dass sich die gängigen Verschlüsselungsverfahren nicht auf diese Weise knacken lassen. Der Ansatz ist jedoch allemal interessant und lässt sich vielleicht zu einem wirklungsvolleren Werkzeug ausbauen.

Zu den neuesten Errungenschaften der Kryptoanalyse-Forschung gehört die **Biclique-Kryptoanalyse** [BoKhRe]. Der Begriff »Biclique« stammt aus der Graphentheorie und setzt sich aus »bi« (»zwei«) und »Clique« (»Gruppe«) zusammen. Es bezeichnet einen vollständigen, bipartiten Graphen. Bei der Biclique-Kryptoanalyse werden Bicliquen aus Schlüsseln gebildet. Dabei kommen Bestandteile aus der differenziellen Kryptoanalyse und der Meet-in-the-Middle-Attacke zum Einsatz. Die Biclique-Kryptoanalyse gehört zu den Chosen-Plaintext-Attacken und benötigt nur einige wenige Klartext-Geheimtext-Paare.

Seit einigen Jahren spielt außerdem die Sicherheit gegenüber Seitenkanalangriffen eine zunehmend wichtige Rolle (siehe Abschnitt 17.1). Seitenkanalangriffe nutzen neben Klar- und Geheimtext weitere Informationen wie Stromverbrauch oder Verschlüsselungsdauer.

Sie sehen also: Es gibt für den Chiffren-Designer eine Menge zu beachten. Wer denkt, er könne als Hobby-Kryptologe ohne größeres Know-how ein gutes Verschlüsselungsverfahren entwickeln (solche Fälle gibt es immer wieder), liegt in aller Regel falsch.

7.4 Beispiele für symmetrische Verschlüsselungsverfahren

Die hohe Qualität des DES zeigt sich nicht zuletzt daran, dass erst Mitte der Achtziger nennenswerte Alternativen zu diesem Verfahren entwickelt wurden. Die ersten DES-Konkurrenten erreichten das Vorbild jedoch nicht und wiesen meist die eine oder andere Sicherheitslücke auf. Erst in den Neunzigern, also etwa 20 Jahre nach der Veröffentlichung des DES, konnten sich schließlich andere Algorithmen etablieren. Danach wandelte sich das Bild jedoch schnell, und inzwischen hat die akademische Kryptografie mehrere Dutzend symmetrische Verschlüsselungsverfahren hervorgebracht, die Alice und Bob guten Gewissens anstelle des DES einsetzen können. Viele davon werde ich in diesem Buch vorstellen. Ich beginne mit einigen frühen DES-Alternativen.

7.4.1 RC2 und RC5

Der wohl bedeutendste lebende Kryptograf ist der US-Amerikaner Ron Rivest (siehe Abschnitt 40.1.5). Zu den zahlreichen Verfahren, die Rivest alleine oder im Team entwickelt hat, zählt eine Reihe symmetrischer Verschlüsselungsalgorithmen, die er schlicht als RC (Rivest-Cipher) bezeichnete und von RC1 bis RC6 durchnummerierte. Die Ähnlichkeit der Namen sollte nicht darüber hinwegtäuschen, dass sich die Verfahren teilweise erheblich voneinander unterscheiden. Um RC1 und RC3 brauchen wir uns nicht zu kümmern, da es diese nicht zur Praxisreife brachten. Um RC4, das zur Familie der Stromchiffren gehört, geht es in Abschnitt 16.2. RC6 ist ein vergleichsweise neues Verfahren, es wird in Abschnitt 9.3 behandelt. In den nächsten Abschnitten werden wir die beiden verbleibenden Verfahren RC2 und RC5 betrachten. Diese zeichnen sich durch ein einfaches Design aus, bieten eine variable Schlüssellänge und spielen in der Praxis eine wichtige Rolle.

7.4.2 RC2

RC2 ist ein symmetrisches Verschlüsselungsverfahren, das Ron Rivest 1987 fertigstellte [RFC2268]. Es ist nach dem DES der zweitälteste in diesem Buch beschriebene symmetrische Algorithmus der Computer-Ära. Die Blocklänge von RC2 beträgt (wie beim DES) 64 Bit, während die Schlüssellänge zwischen 8 und 1.024 Bit frei wählbar ist. Rivests wichtigstes Designziel ist damit klar erkennbar: Er wollte einen Ersatz für den DES schaffen, der längere Schlüssel zulässt. Außerdem sollte das Verfahren besser in Software implementierbar sein als der auf Hardware ausgerichtete DES. Die Tatsache, dass RC2 auch 20 Jahre nach seiner Fertigstellung noch im unveränderten Design eingesetzt wird, zeigt, dass Ron Rivest seinerzeit gute Arbeit geleistet hat.

An der Entwicklung von RC2 war die Firma Lotus beteiligt, die für ihre Software Notes ein geeignetes Verschlüsselungsverfahren suchte. Um die damals gül-

tigen Exportbeschränkungen einzuhalten, wurden 24 Bit des von Notes verwendeten 64-Bit-Schlüssels bei der NSA hinterlegt. Die NSA beteiligte sich zudem aktiv an der endgültigen Festlegung des Designs von RC2. Die Funktionsweise von RC2 wurde zunächst geheim gehalten, um ein Reverse Engineering der Notes-RC2-Implementierung zu verhindern. Die Geheimniskrämerei dauerte jedoch nur bis 1996, als ein Unbekannter die Spezifikation von RC2 in einer Usenet-Gruppe veröffentlichte (Gleiches war schon zuvor mit RC4 passiert). Die Firma RSA Data Security, in deren Dienst Rivest das Verfahren entwickelt hatte, machte RC2 kurz darauf auch offiziell öffentlich, um den E-Mail-Verschlüsselungsstandard S/MIME (er sieht RC2 als eines von mehreren Verfahren vor) zu etablieren.

Funktionsweise von RC2

Das Design von RC2 ähnelt in einigen Punkten einer Feistel-Chiffre, sieht jedoch vier statt zwei Datenströme vor. Ein 64-Bit-Klartextblock wird in vier 16-Bit-Teilblöcke aufgeteilt, die jeweils den Eingabewert für einen Datenstrom bilden. Im Rahmen der Schlüsselaufbereitung werden 64 16-Bit-Werte generiert, die als K_0, K_1, ..., K_{63} bezeichnet werden. Für den Verschlüsselungsablauf spielen zwei Operationen eine Rolle:

- *Mix*: Wie die Mix-Operation aussieht, ist in Abbildung 7–5 ersichtlich. Das Zeichen »<<<« steht für eine Linksrotation. Die Zahl *s* kann vier Werte annehmen: 1, 2, 3 und 5. Der Subschlüssel ist beim ersten Aufruf K_0, beim zweiten K_1, beim dritten K_2 usw.
- *Mash*: Die Mash-Operation sieht vor, dass zu einem der vier Datenströme ein Subschlüssel addiert wird (modulo 2^{16}). Welcher Subschlüssel dies ist, hängt vom benachbarten Datenstrom ab (gemeint ist jeweils der rechte Nachbar, wobei der vierte Datenstrom den ersten als rechten Nachbarn hat). Ist *A* der bearbeitete Datenstrom und *B* der rechte Nachbar, dann gilt: $A = A + K_n$, wobei *n* die aus den sechs niederwertigsten Bit von *B* gebildete Zahl ist.

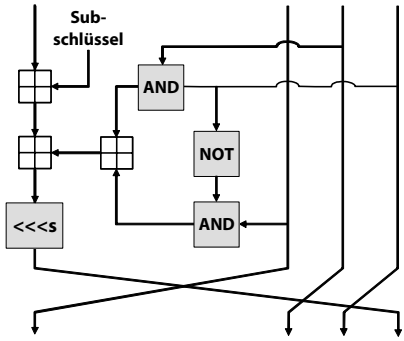


Abb. 7–5 *Eine Mixing-Operation von RC2*

RC2 arbeitet in 18 Runden. Eine Besonderheit ist, dass es zwei unterschiedliche Rundentypen gibt (*Mixing-Runde* und *Mashing-Runde*). Eine Mixing-Runde hat folgenden Ablauf:

1. Mixing-Operation mit $s = 1$
2. Mixing-Operation mit $s = 2$
3. Mixing-Operation mit $s = 3$
4. Mixing-Operation mit $s = 5$

Eine Mashing-Runde hat folgenden Ablauf:

1. Mashing-Operation auf dem ersten Datenstrom
2. Mashing-Operation auf dem zweiten Datenstrom
3. Mashing-Operation auf dem dritten Datenstrom
4. Mashing-Operation auf dem vierten Datenstrom

Die 18 Runden von RC2 laufen wie folgt ab:

1. 5 Mixing-Runden
2. 1 Mashing-Runde
3. 6 Mixing-Runden
4. 1 Mashing-Runde
5. 5 Mixing-Runden

Die Schlüsselaufbereitung von RC2 hat die Aufgabe, 64 Subschlüssel zu generieren. In jede der 16 Mixing-Runden gehen jeweils vier davon ein (es gibt also genau einen Subschlüssel pro Mixing-Operation). In den beiden Mashing-Runden werden insgesamt acht Subschlüssel benötigt, von denen jeder aus K_0, ..., K_{63} datenabhängig ausgewählt wird. Wie genau die Schlüsselaufbereitung von RC2 funktioniert, soll uns an dieser Stelle nicht interessieren. Wer das Verfahren implementieren will, findet in [RFC2268] alle notwendigen Informationen.

Interessant an der Funktionsweise von RC2 ist, dass das Verfahren Konfusions- und Diffusionselemente nicht trennt und daher auch ohne S-Boxen auskommt. Die Aufgabe der Konfusion (also den nichtlinearen Teil) übernehmen die Und-Funktionen, während die Diffusion in den restlichen Bestandteilen der Mixing- und Mashing-Operationen realisiert wird. Beachten Sie, dass RC2 keine datenabhängigen Rotationen verwendet – im Gegensatz zu den beiden Nachfolgern RC5 und RC6.

Bewertung von RC2

RC2 spielte in den neunziger Jahren eine wichtige Rolle, als der Export von Kryptografie aus den USA noch stark eingeschränkt war. Produkte, die RC2 mit einem 40-Bit-Schlüssel unterstützten, erhielten damals recht einfach eine Exportgenehmigung (unnötig zu erwähnen, dass 40 Bit bereits damals keine allzu hohe Sicherheit boten). Seit dem Jahr 2000 spielt dieser Umstand aufgrund einer geänderten

Rechtslage in den USA jedoch keine Rolle mehr. Seitdem wird RC2 zunehmend von anderen Verfahren verdrängt. Setzt man RC2 mit einer angemessenen Schlüssellänge ein (z. B. 128 Bit), dann gilt das Verfahren nach wie vor als sicher. Es gibt zwar einen Angriff, der zur Familie der Related-Key-Attacken gehört und 2^{34} gewählte Klartexte benötigt [KeSW97]. Für die Praxis hat diese Schwachstelle allerdings keine allzu große Bedeutung. Ein Pluspunkt ist neben der Einfachheit die hohe Verschlüsselungsgeschwindigkeit von RC2, die etwa zwei- bis dreimal höher liegt als beim DES. Da es mit RC5 und RC6 zwei modernere Nachfolger gibt, kann man allerdings auf RC2 inzwischen verzichten.

7.4.3 RC5

Mit **RC5** veröffentlichte Ron Rivest 1994 einen weiteren symmetrischen Verschlüsselungsalgorithmus [Rivest]. Dieser hat zwar einige Gemeinsamkeiten mit dem Vorgänger RC2, ist jedoch im Wesentlichen ein neues Verfahren. Das Interessante an RC5 ist seine Einfachheit verbunden mit seiner Flexibilität. Außerdem handelt es sich um ein sehr schnelles Verfahren – sowohl in Hardware als auch in Software.

Funktionsweise von RC5

Sieht man von den Stromchiffren (Kapitel 16) ab, dann hat RC5 von allen in diesem Buch vorgestellten symmetrischen Verschlüsselungsverfahren den einfachsten Aufbau. Zudem bietet RC5 einiges an Flexibilität: Sowohl die Blocklänge als auch die Schlüssellänge als auch die Rundenzahl sind variabel. Dabei gelten folgende Einschränkungen:

- *Blocklänge*: Diese beträgt 32, 64 oder 128 Bit.
- *Schlüssellänge*: Diese muss ein Vielfaches von 8 betragen. Der Höchstwert ist 2.040 (= 255 Bytes).
- *Rundenzahl*: Diese kann zwischen 0 und 255 liegen.

Die Funktionsweise von RC5 ist in Abbildung 7–6 ersichtlich (»<<<« steht wiederum für eine Linksrotation). Nachdem RC2 mit vier Datenströmen arbeitete, reduzierte Rivest deren Anzahl für RC5 auf zwei. Dadurch ähnelt RC5 einer Feistel-Chiffre. Verwirrenderweise umfasst eine RC5-Runde zwei Teilrunden, die jeweils einer Runde in einer Feistel-Chiffre entsprechen (dies ist ein Erbe von RC2). Vergleicht man RC5 mit einer Feistel-Chiffre (etwa dem DES), dann muss man daher in Gedanken die RC5-Rundenzahl verdoppeln.

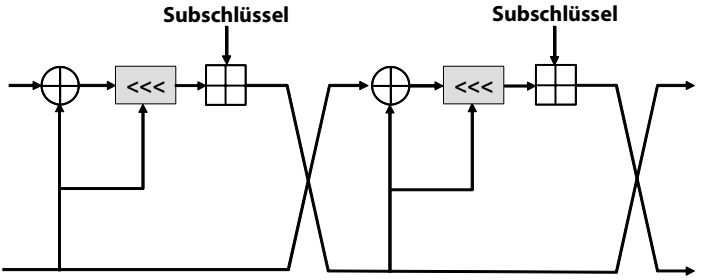


Abb. 7–6 *Eine Runde von RC5 entspricht zwei Feistel-Runden. Die Sicherheit des Verfahrens hängt zu einem wesentlichen Teil von datenabhängigen Rotationen ab.*

RC5 sieht Rotationen vor. Im Gegensatz zu RC2 sind diese Rotationen jedoch nicht von vornherein festgelegt, sondern datenabhängig. Wie in Abbildung 7–6 zu sehen, wird jeweils der linke Datenstrang in Abhängigkeit vom rechten nach links rotiert, wobei sich die Zahl der Einheiten, um die rotiert wird, direkt aus dem Wert des rechten Datenstrangs ergibt (je nach Blocklänge sind dabei nur dessen vier bis sechs niederwertigste Daten-Bits relevant). RC5 nutzt diese Rotationen anstelle von S-Boxen (datenabhängige Rotationen sind nicht linear). Rotationen sind vor allem in Hardware sehr performant zu realisieren und benötigen deutlich weniger Speicherplatz als eine Ersetzungstabelle. Dafür bieten sie nicht ganz die Sicherheit einer gut designten S-Box, weshalb Rivest 1994 eine Rundenzahl von 12 vorschlug. Dies entspricht 24 Feistel-Runden, während der DES mit 16 auskommt. Die RC5-Schlüsselaufbereitung muss pro Runde zwei Subschlüssel zur Verfügung stellen. Ein Subschlüssel hat jeweils die halbe Länge eines Blocks und ist damit 16, 32 oder 64 Bit lang. Der Ablauf der Schlüsselaufbereitung ist deutlich komplexer als beim DES. Auf deren genaue Betrachtung will ich an dieser Stelle verzichten.

Bewertung von RC5

Der beste bekannte Angriff gegen RC5 ist eine differenzielle Kryptoanalyse, für die Mallory bei 12 Runden und 64 Bit Blocklänge 2^{44} gewählte Klartexte benötigt [BirKus]. Bei mehr als 12 Runden bleibt nur die vollständige Schlüsselsuche. Es gibt einige schwache Schlüssel, die Mallory eine differenzielle Kryptoanalyse ermöglichen, doch deren Anteil an der Gesamtzahl der möglichen Schlüssel ist verschwindend gering. An der Sicherheit von RC5 gibt es daher nichts auszusetzen, sofern Alice und Bob die Rundenzahl nicht allzu knapp bemessen.

Dennoch gibt es einige Kritikpunkte an RC5. Zum einen erscheint es uneffektiv, eine Rotation jeweils nur von den vier bis sechs niederwertigsten Daten-Bits eines Ablaufstrangs steuern zu lassen. Mehrere Kryptoanalyse-Arbeiten deuten darauf hin, dass Alice und Bob RC5 bei gleicher Sicherheit mit weniger Runden betreiben könnten, wenn sich das Verfahren in dieser Hinsicht anders verhielte.

Der zweite wichtige Kritikpunkt ist die Blocklänge. Auf einem 32-Bit-Prozessor (dies ist die derzeit häufigste Variante) erreicht RC5 die maximale Verschlüsselungsgeschwindigkeit bei einer Blocklänge von 64 Bit. Heutzutage gelten jedoch 128 Bit Blocklänge als wünschenswert.

7.4.4 Blowfish

Von Bruce Schneier, einem amerikanischen Kryptografen und Buchautor, ist in diesem Buch mehrfach die Rede. Neben seinen äußerst erfolgreichen Veröffentlichungen ist er durch einige Krypto-Verfahren bekannt geworden, die er (mit-)entwickelt hat. **Blowfish** ist ein Algorithmus, den Schneier 1993 entwarf und ein Jahr später veröffentlichte [Schn07/1]. Der Name bedeutet auf Deutsch »Kugelfisch« (dies ist eine Fischart) und bezieht sich auf das Maskottchen des Unix-Betriebssystems Open BSD. Schneier wollte mit Blowfish eine Alternative für den damals bereits in die Jahre gekommenen DES schaffen. Im Gegensatz zu vielen anderen frühen DES-Alternativen bietet Blowfish eine Schlüssellänge, die weit über 128 Bit hinausgeht. Dies war offensichtlich ein wichtiges Designziel von Schneier, denn dieser war damals wie heute ein Freund besonders langer Schlüssel (in [FeScKo] empfiehlt er, generell eine Schlüssellänge von 256 Bit zu verwenden).

Funktionsweise von Blowfish

Blowfish verwendet hauptsächlich Designtechniken, die bereits 1993 nicht mehr neu waren. Das Verfahren ist eine Feistel-Chiffre, die wie der DES 64-Bit-Blöcke in 16 Runden bearbeitet. Die Schlüssellänge ist variabel und liegt zwischen 32 und 448 Bit. Im Gegensatz zum DES gibt es keine Permutationen vor und nach den 16 Runden. Stattdessen sieht Blowfish zwei zusätzliche Subschlüssel vor, die nach den 16 Runden mit je einer Blockhälfte exklusiv-oder-verknüpft werden. Dies ist eine Form von Whitening. Die Rundenfunktion von Blowfish bildet einen 32-Bit-Eingabewert auf einen 32-Bit-Ausgabewert ab (bei einer Feistel-Chiffre mit 64 Bit Blocklänge ist dies der Normalfall). Der Eingabewert wird in vier Bytes zerlegt, die jeweils einer S-Box zugeführt werden. Jede S-Box gibt 32 Bit aus. Die vier resultierenden 32-Bit-Blöcke werden in der Form $(((B_1+B_2)\oplus B_3)+B_4)$ miteinander verknüpft. Das Resultat ist der Ausgabewert der Rundenfunktion. Die Rundenschlüssel, die ebenfalls 32 Bit lang sind, werden jeweils am Ende einer Runde per Exklusiv-oder-Verknüpfung eingebracht.

Das Besondere an den S-Boxen von Blowfish ist, dass ihr Inhalt nicht fest vorgegeben ist. Stattdessen werden sie im Rahmen der Schlüsselaufbereitung mit einem aus dem Schlüssel abgeleiteten Zufallsmuster gefüllt. Solche zufällig generierten S-Boxen haben einerseits den Vorteil, dass Mallory sie nicht analysieren kann, was unter anderem die differenzielle und lineare Kryptoanalyse erschwert. Andererseits besteht vor allem bei kleineren S-Boxen die Gefahr, dass durch eine zufällige Festlegung ungünstige Eigenschaften entstehen. Zudem ist es bei Hard-

wareimplementierungen vorteilhaft, feste S-Boxen zu haben, da diese im nichtflüchtigen Speicher gehalten werden können.

Schlüsselaufbereitung von Blowfish

Die Schlüsselaufbereitung von Blowfish ist gleichzeitig auch die S-Boxen-Generierung. Im Rahmen dieses Vorgangs müssen zum einen 18 Subschlüssel (16 für die Runden, zwei für das Whitening) und zum anderen vier zufällige S-Boxen generiert werden. Dieser Vorgang ist recht aufwendig. Er sieht vor, dass zunächst alle Subschlüssel und alle S-Boxen mit einem vorgegebenen Wert (es handelt sich um die Nachkommastellen der Zahl Pi, die als Leerer-Ärmel-Zahl verwendet werden) initialisiert und dann mit dem Schlüssel exklusiv-oder-verknüpft werden. Ist der Schlüssel kürzer als 448 Bit, dann wird er bei diesem Vorgang mehrfach hintereinander verwendet. In dieser Konstellation wird ein Block aus lauter Null-Bits verschlüsselt. Das Ergebnis bildet die ersten beiden Subschlüssel und wird dann erneut verschlüsselt. Das Ergebnis bildet die Subschlüssel 3 und 4 und so weiter. Auf diese Weise werden nach und nach alle Rundenschlüssel und S-Boxen aufgefüllt.

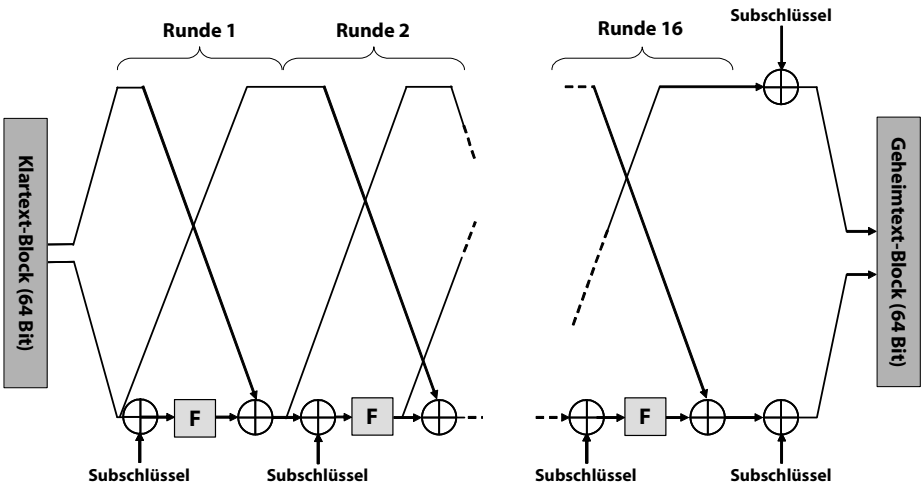


Abb. 7–7 *Blowfish arbeitet in 16 Runden. Das Design des Verfahrens ähnelt in einigen Punkten dem DES, wobei jedoch schlüsselabhängige S-Boxen zum Einsatz kommen.*

Bewertung von Blowfish

Blowfish ist zweifellos weder innovativ noch elegant. Die Schlüsselaufbereitung wirkt reichlich hausbacken und ist obendrein recht aufwendig. Sie verbraucht etwa so viel Rechenzeit wie das Verschlüsseln von 4 Kilobyte an Daten. Daher ist Blowfish nicht für Anwendungsfälle geeignet, in denen der Schlüssel oft wechselt

(wird dagegen derselbe Schlüssel mehrfach verwendet, kann man die Teilschlüssel zwischenspeichern). Dafür ist Blowfish recht schnell und gilt als sicher. Alle bisher gefundenen Schwächen betreffen lediglich Blowfish-Varianten mit weniger als den 16 vorgesehenen Runden. Darüber hinaus gibt es einige schwache Schlüssel (diese generieren schlechte S-Boxen), was jedoch bisher noch niemand zu einem Angriff ausbauen konnte.

Neben Sicherheit und Schnelligkeit trug vor allem die Herkunft zur Popularität von Blowfish bei. Sein Erfinder Bruce Schneier ist weit über die Krypto-Szene hinaus bekannt und als unabhängiger Experte gefragt. Viele Softwareentwickler bevorzugten ein von Schneier entwickeltes Verfahren gegenüber einem, hinter dem ein anonymer Konzern stand (bei SAFER, RC2, RC4, RC5, RC6 und einigen anderen Algorithmen ist dies der Fall). Darüber hinaus verzichtete Schneier auf eine Patentierung und machte sein Verfahren dadurch noch attraktiver.

Da Blowfish keine mächtige Lobby besitzt, schaffte es das Verfahren nur selten, Teil eines Standards zu werden. Dafür setzten viele Hersteller proprietärer Lösungen auf das Verfahren und machten es Ende der neunziger Jahre zu einem stillen Star der Krypto-Welt. Die Blowfish-Webseite nennt heute fast 200 Produkte, die das Verfahren nutzen, darunter Datei-Verschlüsselungslösungen, Betriebssysteme (z.B. Linux) und E-Mail-Verschlüsselungsprogramme. Inzwischen gibt es zwar bessere Algorithmen, doch für die Verschlüsselung von 64-Bit-Blöcken ohne häufigen Schlüsselwechsel ist Blowfish nach wie vor eine gute Wahl.

7.4.5 IDEA und IDEA NXT

IDEA (International Data Encryption Algorithm) ist ein symmetrisches Verschlüsselungsverfahren, das in den neunziger Jahren sehr populär war [LaiMas]. Die beiden Erfinder des Verfahrens, James Massey und Xueija Lai, arbeiteten mit dem Schweizer Elektronikkonzern Ascom zusammen, der IDEA patentieren ließ. Das Verfahren, das 1991 erstmals veröffentlicht wurde, war als Alternative zum DES gedacht und erwies sich in dieser Rolle als sehr erfolgreich. Als erstes Verfahren nach dem DES war IDEA gegenüber der differenziellen Kryptoanalyse abgesichert. Auch die lineare Kryptoanalyse blieb eine zahnlose Waffe gegen das Verfahren. Im Gegensatz zum Vorbild DES bietet IDEA eine längere Schlüssellänge (128 Bit) und eine höhere Performanz in Software. Da zudem keine Sicherheitslücken entdeckt wurden, entwickelte sich IDEA hinter dem DES Mitte der Neunziger zur Nummer zwei unter den symmetrischen Verfahren. Mit PGP setzte die wichtigste Krypto-Software überhaupt einige Jahre ausschließlich auf IDEA. Zudem bezeichnete Krypto-Papst Bruce Schneier IDEA in seinem einflussreichen Buch *Angewandte Kryptographie* als den besten öffentlich verfügbaren symmetrischen Verschlüsselungsalgorithmus [Schn96]. Kein Wunder, dass IDEA als neuer De-facto-Standard gehandelt wurde.

Die Blocklänge von IDEA beträgt (wie beim DES) 64 Bit. Das Verfahren arbeitet in acht Runden und sieht vier Ablaufstränge vor. Es handelt sich weder um eine Feistel- noch um eine SP-Chiffre. Zu den Grundgedanken von IDEA gehört die abwechselnde Nutzung von Operationen aus unterschiedlichen mathematischen Gruppen. Als nichtlineare Bestandteile werden Multiplikationen modulo $2^{16}+1$ verwendet. Das aus heutiger Sicht ungewöhnliche Designprinzip erwies sich als gelungen, und so gilt IDEA bis heute als sicher. Allerdings wurden mehrere Arten schwacher Schlüssel entdeckt, die zumindest unschön wirken.

Seine einst überragende Stellung hat IDEA in den letzten zehn Jahren wieder verloren. Schuld daran ist zum einen der Patentschutz, den der Algorithmus genießt – angesichts der immer zahlreicher werdenden kostenlosen Alternativen hatte offensichtlich niemand Lust, Lizenzgebühren für IDEA zu bezahlen. Ein weiterer Nachteil besteht darin, dass das Verfahren mit 16-Bit-Einheiten arbeitet, obwohl heute 32 Bit auf vielen Prozessoren passender wären. Eine Umstellung von 16 auf 32 Bit würde allerdings erfordern, dass die Multiplikation modulo $2^{16}+1$ durch eine Multiplikation modulo $2^{32}+1$ ersetzt wird. Im Gegensatz zu $2^{16}+1$ ist $2^{32}+1$ jedoch keine Primzahl und hat daher andere kryptografische Eigenschaften. Eine Umstellung auf 32-Bit-Einheiten würde daher fast alle bisherigen Analysen des Verfahrens hinfällig machen.

Der AES-Wettbewerb, der zahlreiche neue Verfahren populär machte, sorgte schließlich dafür, dass IDEA immer mehr von der Bildfläche verschwand. Die IDEA-Entwickler reichten ihr Verfahren nicht als AES-Kandidat ein, was zum einen daran lag, dass sie dafür den Patentschutz hätten auflösen müssen. Zum anderen erfüllte IDEA die AES-Kriterien – 128 Bit Blocklänge und 128/192/256 Bit Schlüssellänge – nicht und hätte daher ein Redesign benötigt.

Seit 2003 gibt es einen IDEA-Nachfolger, der **IDEA NXT** genannt wird. Sein ursprünglicher Name war FOX. Die beiden Entwickler des Verfahrens, Pascal Junod und Serge Vaudenay, verpassten dem Algorithmus zahlreiche Neuerungen, wobei insbesondere die Diffusions- und Konfusionsbestandteile nun deutlich erkennbar sind. Die Multiplikationen modulo $2^{16}+1$ gibt es nicht mehr, auch keine ähnlichen Operationen. IDEA NXT existiert in mehreren Varianten mit unterschiedlichen Block- und Schlüssellängen. Obwohl es sich dabei um ein gutes Verfahren handeln dürfte, wird es IDEA NXT schwer haben, sich gegen den AES, Serpent und Twofish zu behaupten.

7.4.6 Skipjack

Zu den meistdiskutierten symmetrischen Verfahren seiner Zeit gehörte **Skipjack** [Skipja]. Skipjack wurde von niemand Geringerem als der NSA entwickelt. Die Arbeit an diesem Verfahren begann bereits 1985 und wurde 1990 abgeschlossen. Skipjack war ein Teil des gescheiterten Clipper-Projekts (siehe Abschnitt 40.5.9). Wie Skipjack funktioniert, hielt die NSA zunächst einmal geheim. Bekannt war nur, dass es sich um eine Blockchiffre handelte, die 64-Bit-Blöcke in 32 Runden mit einem 80-Bit-Schlüssel verschlüsselt. Dabei arbeitet Skipjack etwa doppelt so schnell wie der DES. Über alle anderen Details des Verfahrens konnte man nur spekulieren. Da Skipjack nur in Form von Hardware auf den Markt kam, gab es keine Möglichkeit, die Funktionsweise des Verfahrens zu analysieren.

1998, als von Clipper keine Rede mehr war, veröffentlichte die NSA dann doch die Funktionsweise des Verfahrens. Dabei bewahrheitete sich, was viele Kryptografen erwartet hatten: Obwohl die NSA zweifellos kryptografische Kenntnisse besitzt, die über den Stand der akademischen Forschung hinausgehen, verwendet Skipjack lediglich bereits bekannte Techniken (diese allerdings sehr gut). Die Hoffnung, über Skipjack würde die NSA etwas von ihrem geheimen Wissen preisgeben, erfüllte sich damit nicht.

Natürlich stürzten sich sofort Experten in aller Welt auf Skipjack. Dabei machten die Kryptografen Eli Biham, Adi Shamir und Alex Biryukov eine interessante Entdeckung: Sie fanden eine Schwachstelle für den Fall, dass statt 32 nur 31 Runden verwendet werden [BiBiSh]. Ein derart geringer Sicherheitspuffer ist beim Chiffren-Design völlig unüblich und gilt daher als Designfehler. Dieses unerwartete Ergebnis lässt nur zwei Schlüsse zu: Entweder war sich die NSA sicher, dass der besagte Angriff nicht auf 32 Runden ausgedehnt werden kann, oder man hatte diese Schwachstelle bei der US-Geheimbehörde gar nicht entdeckt. Sowohl das eine als auch das andere Ergebnis wäre höchst interessant. Obwohl bisher keine praxisrelevante Schwachstelle entdeckt wurde, wird Skipjack kaum eingesetzt. Vielleicht liegt dies am generellen Misstrauen, das viele der NSA entgegenbringen.

7.4.7 TEA

TEA (**Tiny Encryption Algorithm**) ist ein symmetrisches Verschlüsselungsverfahren, das von den inzwischen verstorbenen Briten David Wheeler und Roger Needham entwickelt wurde [WheNee]. Die beiden stellten ihr Verfahren 1994 am Rande einer Fachkonferenz vor. Der Name von TEA (»tiny« heißt »winzig«) ist Programm: Der Algorithmus ist neben RC5 der einfachste unter den gebräuchlichen Blockchiffren. Es gehörte zu den Designzielen, dass ein Entwickler die Funktionsweise ohne größeren Aufwand auswendig lernen kann. Ein C-Programm, das eine TEA-Verschlüsselung realisiert, hat kaum mehr als zehn Zeilen.

TEA ist eine Feistel-Chiffre, die 64-Bit-Blöcke mit einem 128-Bit-Schlüssel verschlüsselt. Die Rundenfunktion ist besonders simpel und besteht nur aus Additionen, Verschiebungen und Exklusiv-oder-Verknüpfungen (siehe Abbildung 7–8). Der recht einfache Aufbau wird durch eine hohe Zahl von Runden kompensiert (64 Runden werden empfohlen, sind aber nicht zwingend vorgeschrieben). Auch die Schlüsselaufbereitung ist ausgesprochen einfach: In Runde 1 werden die ersten 64 Bit des Schlüssels verwendet, in Runde 2 die zweiten 64 Bit, in Runde 3 wieder die ersten 64 Bit und so weiter. Damit sich die Subschlüssel nicht wiederholen, wird in jeder Runde noch eine wechselnde Konstante dazuaddiert.

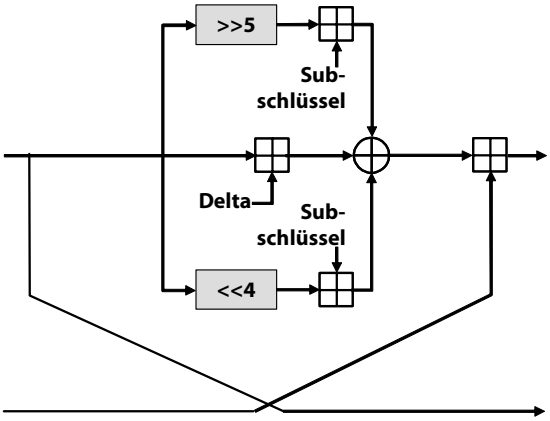


Abb. 7–8 *Eine TEA-Runde. Die Zeichen »<<« und »>>« stehen für eine bitweise Verschiebung nach links bzw. rechts. Delta ist eine rundenabhängige Konstante.*

Die Einfachheit und Schnelligkeit von TEA sind zwar vorteilhaft, haben jedoch auch ihren Preis. So sind bereits einige Angriffe auf das Verfahren bekannt geworden, die unter geeigneten Bedingungen auch praktisch verwertbar sind. Needham und Wheeler stellten daher 1997 **XTEA** vor, eine leicht verbesserte (aber langsamere) Version des Verfahrens. Gleichzeitig veröffentlichten sie auch **Block TEA**, das eine variable Blocklänge ermöglicht. Nachdem eine Schwachstelle von Block TEA entdeckt wurde, kam 1998 dann noch **XXTEA** auf den Markt.

Angesichts der zahlreichen Kryptoanalyse-Ergebnisse ist TEA in seinen diversen Varianten sicherlich kein Verfahren für Hochsicherheitsanwendungen oder Langzeitarchivierungen. Es handelt sich dabei aber um einen brauchbaren Algorithmus für weniger sicherheitskritische Anwendungen, der durch seine Einfachheit und Schnelligkeit durchaus seine Berechtigung hat.

7.4.8 GOST

GOST ist der umgangssprachliche Name eines russischen Verschlüsselungsverfahrens. Es wird im russischen Standard GOST 28147-89 beschrieben (GOST ist eine Abkürzung, die etwa dem deutschen DIN entspricht). Die korrekte Aussprache ist »gost«, auch wenn vor allem im englischen Sprachraum oft »gaust« zu hören ist. Es gibt auch eine kryptografische Hashfunktion, die in einem GOST-Standard beschrieben wird. Im Folgenden ist mit GOST jedoch das Verschlüsselungsverfahren gemeint.

GOST stammt aus den achtziger Jahren und damit aus der Zeit der Sowjetunion. Die Designkriterien sind nicht öffentlich bekannt, lassen sich aber leicht nachvollziehen. Da in den achtziger Jahren der DES als das Maß aller Dinge galt, übernahmen die GOST-Designer dessen Aufbau – es handelt sich also um eine DES-ähnliche Feistel-Chiffre. Die von der NSA mitgestalteten S-Boxen wollten die sowjetischen Kryptologen allerdings nicht übernehmen, stattdessen überließen sie es dem jeweiligen Anwender, eigene S-Boxen zu entwickeln. Die Schlüssellänge erhöhten sie auf 256 Bit, um von den mageren 56 Bit des DES wegzukommen. Um eventuelle Schwächen im Verfahren auszugleichen, erhöhten die GOST-Entwickler die Rundenzahl auf 32 (der DES hat 16), was das Verschlüsseln relativ langsam macht.

Insgesamt zeugt der Aufbau von GOST nicht gerade von Kreativität. Die Entwickler nahmen einfach den DES und änderten ihn auf eine Weise ab, die bei geringstem Risiko dessen tatsächliche oder vermutete Sicherheitsschwachstellen aufhob. Weitergehendes Know-how im Chiffren-Design, das es damals in der Sowjetunion zweifellos gab, ist nicht in GOST eingeflossen. Mehrere Kryptoanalyse-Ergebnisse aus den letzten Jahren zeigen, dass die vollständige Schlüsselsuche bei weitem nicht der beste Angriff auf GOST ist [Courto]. Eine praktisch verwertbare Schwäche wurde bisher jedoch nicht entdeckt, was nicht zuletzt an der hohen Rundenzahl liegt. Da GOST vor allem in Osteuropa nach wie vor im Einsatz ist, lohnt es sich, die weitere Entwicklung der Kryptoanalyse zu verfolgen.

7.4.9 Weitere Beispiele

Weitere symmetrische Verschlüsselungsverfahren aus den Achtzigern und frühen Neunzigern heißen **REDOC**, **Madryga**, **Khufu** und **Khafre**. Diese Algorithmen sind heute nur noch historisch interessant, da sie sich nicht gegen den DES behaupten konnten. Auch der DES-Vorläufer **Lucifer** von IBM soll nicht unerwähnt bleiben. Zu einer gewissen Popularität als Negativbeispiel hat es **FEAL** gebracht – FEAL ist so unsicher, dass so ziemlich jede Kryptoanalyse-Methode funktioniert. Wer sich für diese frühen Verfahren interessiert, findet in [Schn06] gute Beschreibungen.

8 Der Advanced Encryption Standard (AES)

Das momentan wohl bedeutendste symmetrische Verschlüsselungsverfahren ist der **Advanced Encryption Standard** (**AES**). Unter dem Namen Rijndael hat dieses Verfahren den Wettbewerb um die Nachfolge des DES als US-Verschlüsselungsstandard gewonnen (siehe Abschnitt 18.5.2). Seit November 2001 ist es in den USA offiziell standardisiert und wird dort seitdem für die Verschlüsselung staatlicher Dokumente verwendet. Zwei Jahre später ließ die US-Regierung den AES sogar für Daten bis zur höchsten Geheimhaltungsstufe (Top Secret) zu, was beim Vorgänger DES nicht der Fall gewesen war. Zum ersten Mal ist damit ein Verschlüsselungsverfahren, das für derart wichtige Informationen eingesetzt wird, öffentlich bekannt. Zahlreiche staatliche Institutionen in anderen Ländern nutzen den AES ebenfalls. Gleiches gilt für Hersteller von Hard- und Software in aller Welt. Das Verfahren ist nicht patentiert und damit frei verwendbar.

Entwickelt wurde der AES (damals noch unter der Bezeichnung Rijndael) von den beiden belgischen Kryptografen Vincent Rijmen und Joan Daemen (letzterer

ist übrigens ein Mann, auch wenn der Vorname diesbezüglich in die Irre führt). Die Bezeichnung »Rijndael« ist an die Nachnamen der beiden Erfinder angelehnt. Die korrekte Aussprache lautet »Reindahl«. Für die Namenswahl nannten Rijmen und Daemen einen einleuchtenden Grund: Sie wollten, dass ihre Nachnamen endlich einmal richtig ausgesprochen werden. Wer sich intensiv mit dem AES beschäftigen will, sollte sich das Buch besorgen, das Daemen und Rijmen darüber geschrieben haben [DaeRij01].

8.1 Funktionsweise des AES

Der Funktionsweise des AES liegen einige Überlegungen zugrunde, die nur mit mathematischem Grundlagenwissen zu verstehen sind. Dabei spielen endliche Körper (namentlich der Körper GF(2^8), der auch bei Twofish zum Einsatz kommt) eine zentrale Rolle. Diese sind ansonsten vor allem in der asymmetrischen Kryptografie von Bedeutung und werden daher im späteren Verlauf dieses Buchs noch näher betrachtet. Um den Ablauf einer AES-Verschlüsselung nachzuvollziehen, muss man diese mathematischen Grundlagen jedoch nicht unbedingt kennen. Ich werde die Funktionsweise des AES daher zunächst erklären, ohne auf endliche Körper einzugehen. In Abschnitt 8.2 gibt es dann einige Erklärungen zur Mathematik des Verfahrens.

Die Blocklänge des AES beträgt 128 Bit. Das Verfahren Rijndael unterstützt zwar auch 192 und 256 Bit, doch diese Blocklängen wurden nicht in die AES-Standardisierung aufgenommen (genau genommen muss man den AES daher als Spezialfall von Rijndael bezeichnen). Die AES-Schlüssellänge kann – wie bei allen AES-Kandidaten – wahlweise auf 128, 192 oder 256 Bit festgelegt werden. Die Rundenzahl ist, wie in der folgenden Tabelle dargestellt, von der Schlüssellänge abhängig:

Schlüssellänge	Rundenzahl
128 Bit	10
192 Bit	12
256 Bit	14

Der AES ist eine SP-Chiffre. Das Verfahren sieht im Wechsel Schritte zur Substitution und zur Permutation vor, ohne dass es dabei die von Feistel-Chiffren bekannte Aufteilung in linke und rechte Hälfte gibt. Ein Klartextblock (128 Bit) ist stets in Form einer 4×4-Matrix aus Byte-Variablen gegeben, wobei die 16 Bytes eines Blocks spaltenweise in die Matrix geschrieben werden. Ein Klartext (a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3, c_0, c_1, c_2, c_3, d_0, d_1, d_2, d_3) wird daher wie folgt in eine Matrix umgewandelt:

$$\begin{matrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{matrix}$$

8.1.1 Rundenaufbau

Der AES besteht aus vier Funktionen, die jeweils mehrfach durchlaufen werden: SubBytes, ShiftRow, MixColumn und AddRoundKey. Alle vier Funktionen bearbeiten eine 4×4-Matrix, in der am Anfang des Verfahrens der Klartext und am Ende der Geheimtext steht. Der erste Schritt einer AES-Verschlüsselung besteht aus einem Aufruf von AddRoundKey. Anschließend werden alle Runden bis auf eine (also 9 bzw. 11 bzw. 13 Runden) nach folgendem Schema abgearbeitet:

1. *SubBytes*
2. *ShiftRow*
3. *MixColumn*
4. *AddRoundKey*

In der letzten Runde fällt der dritte Schritt (MixColumn) weg, ansonsten verläuft sie gleich. Man kann das Verfahren übrigens auch anders erklären: Alle Runden bis auf die letzte bestehen aus den Schritten 1) AddRoundKey 2) SubBytes 3) ShiftRow 4) MixColumn. In der letzten Runde wird MixColumn durch ein weiteres AddRoundKey ersetzt.

SubBytes

Die Funktion *SubBytes* realisiert eine S-Box. Sie führt also eine nichtlineare Substitution durch. Nach den im Chiffren-Design üblichen Grundsätzen (Abschnitt 7.1) ist SubBytes damit für die Konfusion zuständig. Die Arbeitsweise von SubBytes ist recht einfach: Jedes der 16 Bytes in der 4×4-Matrix wird nach derselben Ersetzungstabelle (S-Box) durch ein neues ersetzt. Die Ersetzungstabelle hat demnach für jeden Bytewert von 0 bis 255 einen Eintrag und sieht so aus (Einträge in hexadezimaler Schreibweise):

00-0F	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
10-1F	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
20-2F	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
30-3F	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
40-4F	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
50-5F	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
60-6F	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
70-7F	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
80-8F	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
90-9F	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A0-AF	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B0-BF	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C0-CF	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D0-DF	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E0-EF	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F0-FF	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Der Aufbau der S-Box ist so gestaltet, dass der AES gegenüber differenzieller und linearer Kryptoanalyse nicht anfällig ist. Eine Besonderheit der AES-S-Box besteht darin, dass man jeden der 256 Einträge durch eine einfache mathematische Formel berechnen kann. Mehr dazu gibt es im Unterkapitel über die mathematischen Hintergründe des AES (Abschnitt 8.2).

ShiftRow

In *ShiftRow* werden die Zeilen der 4×4-Matrix durcheinandergemischt. Der Sinn dieser Funktion dürfte klar sein: Nachdem SubByte als nichtlineare Funktion für Konfusion zuständig war, sorgt ShiftRow für Diffusion durch eine (lineare) Permutation.Die folgende Tabelle zeigt links die Matrix vor dem Mischen und rechts danach:

$$\begin{matrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{matrix} \quad => \quad \begin{matrix} a_0 & b_0 & c_0 & d_0 \\ b_1 & c_1 & d_1 & a_1 \\ c_2 & d_2 & a_2 & b_2 \\ d_3 & a_3 & b_3 & c_3 \end{matrix}$$

In Worten beschrieben bedeutet dies: Die erste Zeile wird um null, die zweite um eine, die dritte um zwei und die vierte um drei Stellen nach links rotiert. An dieser Stelle ist zu beachten, dass Rijndael bei einer Blocklänge von 192 oder 256 Bit geringfügig andere Verschiebungen vorsieht. Da wir jedoch nur den AES betrachten, der eine Blocklänge von 128 Bit vorschreibt, soll uns das nicht weiter interessieren.

MixColumn

Die Funktion *Mix-Column* trägt wie ShiftRow zur Diffusion bei und ist wie diese linear. Dabei gibt es eine klare Aufteilung: Während ShiftRow für eine Durchmischung der Zeilen sorgt, führt MixColumn ein spaltenweises Mischen durch. Im Vergleich zum sehr simplen ShiftRow ist MixColumn etwas komplexer. Um es zu verstehen, benötigen wir eine spezielle Form der Multiplikation, die wir mit dem Zeichen »•« notieren. Ist b ein Byte, dann ist diese Multiplikation wie folgt definiert:

$2 \bullet b$:= Linksshift von b, falls $b<128$
:= (Linksshift von b) $\oplus$ 00011011, falls $b>=128$
$3 \bullet b := (2 \bullet b) \oplus b$

Damit haben wir zwar nur die Multiplikation eines Bytes mit den Zahlen 2 und 3 definiert, doch das reicht aus. Beachten Sie, dass ein Linksshift keine Linksrotation ist; das erste Bit eines Bytes geht bei einem Linksshift verloren und wird nicht in das letzte Bit geschoben (stattdessen wird das letzte Bit Null gesetzt). Zur Erklärung von MixColumn gehen wir davon aus, dass a_0, a_1, a_2 und a_3 die erste Spalte der 4×4-Matrix vor dem MixColumn-Aufruf bilden. Nach Abarbeiten der Funktion bezeichnen wir die entsprechenden Werte als a_0', a_1', a_2' und a_3'. MixColumn sieht folgende Formeln vor:

$a_0' = (2 \bullet a_0) \oplus (3 \bullet a_1) \oplus a_2 \oplus a_3$
$a_1' = a_0 \oplus (2 \bullet a_1) \oplus (3 \bullet a_2) \oplus a_3$
$a_2' = a_0 \oplus a_1 \oplus (2 \bullet a_2) \oplus (3 \bullet a_3)$
$a_3' = (3 \bullet a_0) \oplus a_1 \oplus a_2 \oplus (2 \bullet a_3)$

Die entsprechenden Formeln für die zweite, dritte und vierte Spalte sehen genau gleich aus, nur dass Sie den Buchstaben a durch b, c bzw. d ersetzen müssen. Die vier MixColumn-Formeln sind im Übrigen nach einem mathematischen Modell so gewählt, dass sie eine maximale Durchmischung der vier Matrix-Spalten garantieren. Mehr dazu gibt es im Unterkapitel über die mathematischen Grundlagen des AES.

Wie oben beschrieben, ist MixColumns die einzige AES-Funktion, die nicht in allen Runden vorkommt (sie fehlt in der letzten Runde). Dafür gibt es eine einfache Begründung: Die beiden Funktionen MixColumns und AddRoundKey kann man in der Reihenfolge vertauschen, ohne dass sich am Ergebnis der Verschlüsselung etwas ändert. Gäbe es in der letzten Runde ein MixColumns, dann könnte man dieses mit dem letzten AddRoundKey vertauschen, wodurch der letzte Schritt der Verschlüsselung nicht vom Schlüssel abhängen würde. Es wäre damit kryptografisch wertlos und würde unnötige Rechenzeit verbrauchen.

AddRoundKey

In der Funktion *AddRoundKey* wird ein Subschlüssel zur 4×4-Matrix addiert (bitweises exklusives Oder). Da sowohl der erste als auch der letzte Schritt einer AES-Verschlüsselung aus einem AddRoundKey besteht, ist ein Whitening gegeben.

8.1.2 Entschlüsselung mit dem AES

Bei einer SP-Chiffre ist die Entschlüsselung nie so elegant wie bei einer Feistel-Chiffre. Dies gilt auch für den AES. Eine AES-Entschlüsselung sieht notwendigerweise vor, dass dieselben Schritte wie bei der Verschlüsselung durchlaufen werden, nur invertiert und in umgekehrter Reihenfolge. Bei ShiftRow und AddRoundKey ist die Invertierung trivial. Bei SubByte muss Alice eine inverse Ersetzungstabelle aufbauen, was ebenfalls kein Problem darstellt, weil die S-Box entsprechend konstruiert wurde. Für MixColumns ist die Invertierung nicht ganz so naheliegend, aber durch das zugrunde liegende mathematische Modell ebenfalls nicht schwierig. Die Umkehrung der MixColumns-Formeln für eine Spalte lautet:

$$a_0' = 14 \bullet a_0 \oplus 11 \bullet a_1 \oplus 13 \bullet a_2 \oplus 9 \bullet a_3$$
$$a_1' = 9 \bullet a_0 \oplus 14 \bullet a_1 \oplus 11 \bullet a_2 \oplus 13 \bullet a_3$$
$$a_2' = 13 \bullet a_0 \oplus 9 \bullet a_1 \oplus 14 \bullet a_2 \oplus 11 \bullet a_3$$
$$a_3' = 11 \bullet a_0 \oplus 13 \bullet a_1 \oplus 9 \bullet a_2 \oplus 14 \bullet a_3$$

Da wir die Multiplikation »•« bisher nur für die Zahlen 2 und 3 definiert haben, benötigen wir nun einige weitere Werte:

$$9 \bullet b := 2 \bullet (2 \bullet (2 \bullet b)) \oplus b$$
$$11 \bullet b := 8 \bullet b \oplus 2 \bullet b \oplus b$$
$$13 \bullet b := 8 \bullet b \oplus 4 \bullet b \oplus b$$
$$14 \bullet b := 8 \bullet b \oplus 4 \bullet b \oplus 2 \bullet b$$

8.1.3 Schlüsselaufbereitung

Wie man leicht nachrechnet, benötigt der AES einen Subschlüssel mehr, als es Runden gibt. Jeder Subschlüssel besteht aus 16 Bytes und hat dieselbe Form wie die 4×4-Matrix, die von den vier AES-Funktionen verarbeitet wird. Wir gehen an dieser Stelle von 128 Schlüssel-Bits und damit von 10 Runden aus. Wir benötigen 11 Subschlüssel zu je 16 Bytes, insgesamt also 176 Bytes.

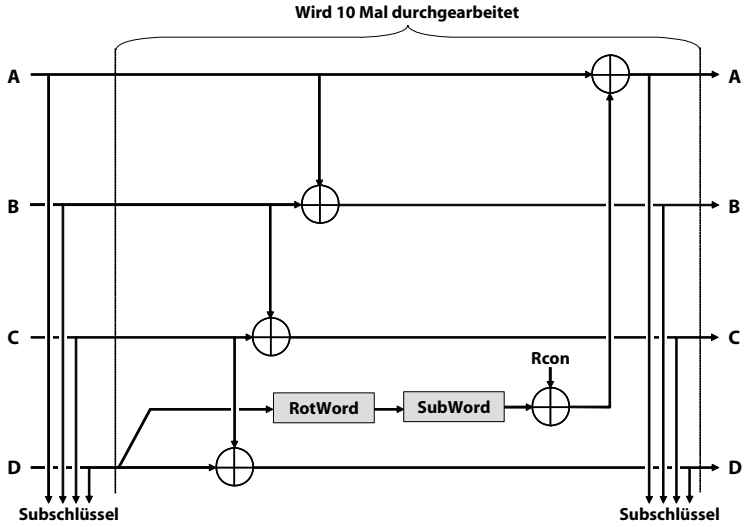


Abb. 8–1 *Die AES-Schlüsselaufbereitung generiert für die 128-Bit-Variante 11 Subschlüssel.*

Die AES-Schlüsselaufbereitung lässt sich am einfachsten an einem Schaubild erklären (siehe Abbildung 8–1). Die 128 Schlüssel-Bits werden zunächst in vier 32-Bit-Variablen aufgeteilt (*A*, *B*, *C* und *D*). Der erste Subschlüssel ist mit dem AES-Schlüssel identisch. Danach wird in zehn Runden jeweils ein weiterer Subschlüssel generiert. Wie in der Abbildung ersichtlich, kommen dabei zwei Funktionen (*SubWord*, *RotWord*) und eine Konstante (*Rcon*) zur Anwendung. Die beiden Funktionen nehmen jeweils vier Bytes (also 32 Bit) entgegen und geben vier Bytes wieder aus. Hier die Beschreibung der Funktionen und der Konstante:

- *SubWord*: Diese Funktion ersetzt jedes der vier Bytes durch den entsprechenden Eintrag in der S-Box des AES (dies ist die Ersetzungstabelle, die auch in SubBytes zum Einsatz kommt).
- *RotWord*: Diese Funktion führt eine byteweise Linksrotation der vier Bytes durch. Lautet die Eingabe b_0,b_1,b_2,b_3, dann ist die Ausgabe b_1,b_2,b_3,b_0.
- *Rcon*: Diese Konstante besteht aus vier Bytes, von denen die letzten drei stets Null sind (nur das erste Byte enthält also einen Wert). Der Wert des ersten Bytes hängt von der Runde ab (das Zeichen »•« bezeichnet die oben definierte Multiplikation):

Runde	Wert
1	1
2	2
3	2•2
4	2•2•2
5	2•2•2•2
...	...

Auch für diese Vorgehensweise gibt es mathematische Hintergünde, die Sie im folgenden Unterkapitel nachlesen können.

8.2 Mathematische Betrachtung des AES

Die Funktionsweise des AES wirkt an einigen Stellen recht willkürlich. Wenn man jedoch weiß, dass dem Verfahren einige mathematische Überlegungen zugrunde liegen, auf die ich bisher nicht eingegangen bin, lässt sich alles recht schlüssig erklären. Um die mathematischen Grundlagen des AES zu verstehen, benötigen Sie ein Grundwissen zum Thema Galois-Felder (siehe Abschnitt 11.1 und 13.1.1).

Für die Funktionsweise des AES spielt der endliche Körper GF(2^8) eine wichtige Rolle. Die Elemente von GF(2^8) sind mit den 256 Werten eines Bytes identisch. Ein Byte $b_0,b_1,b_2,b_3,b_4,b_5,b_6,b_7$ kann man auch als Polynom der Form $b_0x^7+b_1x^6+b_2x^5+b_3x^4+b_4x^3+b_5x^2+b_6x+b_7$ schreiben. Um im Körper GF(2^8) multiplizieren zu können, benötigen wir ein irreduzibles Polynom des Grads 8. Die AES-Entwickler haben $m(x)=x^8+x^4+x^3+x+1$ hierfür vorgesehen. Eine Multiplikation zweier Bytes entspricht daher immer der Multiplikation der beiden zugehörigen Polynome modulo $m(x)$. Eine Byte-Multiplikation modulo $m(x)$ wird mit dem Zeichen »•« notiert. Damit wäre geklärt, wie die in MixColumns und Rcon verwendete Multiplikation zustande kommt. Das Praktische daran ist, dass man für die Durchführung einer solchen Multiplikation keine Modulo-Polynom-Rechnung durchführen muss. Dies liegt daran, dass die AES-Entwickler das Polynom $m(x)$ so gewählt haben, dass man das Multiplizieren mit der oben beschriebenen Methode elegant abkürzen kann.

Auch die Funktion SubBytes lässt sich mithilfe der Polynom-Multiplikation modulo $m(x)$ beschreiben. Ist b der Bytewert aus der 4×4-Matrix, dann hat der zugehörige Tabelleneintrag den Wert $A \cdot b^{-1} \oplus 01100011$. Hierbei ist b^{-1} das inverse Element zu b in GF(2^8), es gilt also: $b \bullet b^{-1}=1$. A ist eine Matrix mit folgendem Inhalt:

$$\begin{matrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{matrix}$$

Beachten Sie, dass sowohl die Matrix-Multiplikation als auch die Addition der Konstante 01100011 lineare Operationen sind. Der nichtlineare Anteil entsteht ausschließlich durch b^{-1}. Wenn Sie mehr über das S-Boxen-Design des AES wissen wollen, empfehle ich Ihnen einen Blick in [DaeRij01].

8.3 Sicherheit des AES

Angesichts einer Schlüssellänge von mindestens 128 Bit ist eine vollständige Schlüsselsuche beim AES ein aussichtsloses Unterfangen. Dafür war die relativ geringe Rundenzahl von Anfang an umstritten. Die zunächst bekannten Angriffe funktionierten gegen sieben Runden bei 128 Bit Schlüssellänge, gegen acht Runden bei 192 Bit Schlüssellänge sowie gegen neun Runden bei 256 Bit Schlüssellänge [KLSSWW]. Das alles nutzt Mallory zwar nichts, da der AES 10, 12 bzw. 14 Runden vorsieht, doch der Puffer beträgt nur drei bis fünf Runden. Für ein Verfahren, das einige Jahrzehnte lang verwendet werden soll, ist dies zweifellos etwas knapp bemessen. Auch die Tatsache, dass der AES einige damals noch wenig erforschte Designprinzipien (z.B. durch eine Formel berechenbare S-Boxen) verwendet, sorgte für Bedenken. Über ein Jahrzehnt später ist jedoch nach wie vor keine Schwäche des AES bekannt, die auch nur annähernd eine praktische Bedeutung hat. Dafür sind einige recht interessante theoretische Angriffe bekannt geworden, die wir uns im Folgenden genauer anschauen wollen.

8.3.1 AES als algebraische Formel

Ein Kritikpunkt ist das mathematische Modell, das dem AES zugrunde liegt. Dieses macht das Verfahren zwar sehr elegant, es hat aber auch Nachteile. 2001 zeigten die drei Kryptografen Niels Ferguson, Richard Schroeppel und Doug Whiting, dass sich der AES vergleichsweise gut als algebraische Formel darstellen lässt [FeScWh]. Bei einer Schlüssellänge von 128 Bit enthält die entsprechende Formel 2^{50} Bestandteile, bei 256 Schlüssel-Bits sind es 2^{70}. Diese Zahlen sind zwar nicht gerade klein. Es gibt jedoch kein anderes modernes symmetrisches Verschlüsselungsverfahren, für das eine vergleichbare Darstellung bekannt ist. 2^{50} ist eine Größenordnung, die man mit sehr viel Computeraufwand gerade noch in den Griff bekommen kann, zumal die Leistungsfähigkeit der verfügbaren Computer ständig zunimmt. Außerdem könnten neue Erkenntnisse dafür sorgen, dass man Vereinfachungsmöglichkeiten entdeckt.

$$a_{i,j}^{(3)} = k_{i,j}^{(2)} + \sum_{\substack{e_2 \in \mathcal{E} \\ d_2 \in \mathcal{D}}} \frac{w_{i,e_2,d_2}}{\left(k_{e_2,e_2+j}^{(1)} + \sum_{\substack{e_1 \in \mathcal{E} \\ d_1 \in \mathcal{D}}} \frac{w_{e_2,e_1,d_1}}{(a_{e_1,e_1+e_2+j}^{(1)})^{2^{d_1}}} \right)^{2^{d_2}}} \tag{4}$$

and the three-round version is

$$a_{i,j}^{(4)} = k_{i,j}^{(3)} + \sum_{\substack{e_3 \in \mathcal{E} \\ d_3 \in \mathcal{D}}} \frac{w_{i,e_3,d_3}}{\left(k_{e_3,e_3+j}^{(2)} + \sum_{\substack{e_2 \in \mathcal{E} \\ d_2 \in \mathcal{D}}} \frac{w_{e_3,e_2,d_2}}{\left(k_{e_2,e_2+e_3+j}^{(1)} + \sum_{\substack{e_1 \in \mathcal{E} \\ d_1 \in \mathcal{D}}} \frac{w_{e_2,e_1,d_1}}{(a_{e_1,e_1+e_2+e_3+j}^{(1)})^{2^{d_1}}} \right)^{2^{d_2}}} \right)^{2^{d_3}}}$$

$$a_{i,j}^{(3)} = K + \sum_{\substack{e_2 \in \mathcal{E} \\ d_2 \in \mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_1 \in \mathcal{E} \\ d_1 \in \mathcal{D}}} \frac{C}{p_*^* + K^*}}$$

We can now write the five-round formula

$$a_{i,j}^{(6)} = K + \sum_{\substack{e_5 \in \mathcal{E} \\ d_5 \in \mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_4 \in \mathcal{E} \\ d_4 \in \mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_3 \in \mathcal{E} \\ d_3 \in \mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_2 \in \mathcal{E} \\ d_2 \in \mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_1 \in \mathcal{E} \\ d_1 \in \mathcal{D}}} \frac{C}{K^* + p_*^*}}}}} \tag{5}$$

Abb. 8–2 *Der AES lässt sich (in sehr aufwendiger Form) als mathematische Formel darstellen. Die Abbildung zeigt lediglich einen Auszug. Noch hat niemand eine Möglichkeit gefunden, diese Formel nach dem Schlüssel aufzulösen.*

Die entscheidende Frage lautet nun: Gibt es eine Möglichkeit, die algebraische AES-Formel mit vertretbarem Aufwand nach dem Schlüssel aufzulösen? Wäre dies der Fall, dann würde möglicherweise schon ein einziges Klartext-Geheimtext-Paar genügen, um den Schlüssel zu ermitteln. Bisher lautet die Antwort jedoch eindeutig nein. Noch gibt es nicht einmal ansatzweise eine Methode, um die AES-Formel zu lösen. Die Sicherheit des AES ist von dieser Seite also derzeit nicht gefährdet. Dennoch ist Vorsicht geboten. Die Lösung der AES-Formel betrifft Teilgebiete der Mathematik, die bisher keine Berührungspunkte mit der Kryptografie hatten. Es ist daher nicht auszuschließen, dass eines Tages ein kluger Kopf, der mit Kryptografie bis dahin nichts am Hut hatte, die passende Lösungsmethode präsentiert. Gut möglich, dass dieser kluge Kopf auf mathematische Erkenntnisse zurückgreift, die derzeit keiner mit diesem Problem in Verbindung bringt. Bisher hat sich allerdings noch kein kluger Kopf gemeldet.

8.3.2 Quadratische Kryptoanalyse

Die einfache mathematische Darstellbarkeit des AES macht das Verfahren besonders interessant für die quadratische Kryptoanalyse (XSL-Angriff). Diese 2002 von Nicolas Courtois und Josef Pieprzyk veröffentlichte Methode sieht vor, ein Verschlüsselungsverfahren mithilfe eines quadratischen Gleichungssystems nachzubilden und dieses anschließend zu lösen (beispielsweise mit dem sogenannten XSL-Verfahren). In der Tat lässt sich der AES besser mit quadratischen Gleichungen nachbilden als die meisten anderen symmetrischen Verschlüsselungsverfahren [CouPie]. Bei einer Schlüssellänge von 128 Bit sind 8.000 Gleichungen mit 1.600 Variablen notwendig. Der Bedarf an Klartext-Geheimtext-Paaren ist gering.

Die quadratische Kryptoanalyse gibt jedoch bisher noch keinen Anlass zur Panik. Noch ist völlig unklar, ob die Methode überhaupt so funktioniert, wie es sich die beiden Erfinder vorstellen (nicht wenige Fachleute zweifeln daran). Falls ja, ist man bisher noch weit davon entfernt, ein Gleichungssystem dieser Komplexität lösen zu können. Bisher ist die quadratische Kryptoanalyse daher ein sehr spekulatives Thema, das noch weiterer Forschung bedarf. AES-Miterfinder Vincent Rijmen ließ sich jedenfalls erst einmal nicht beeindrucken. Sein Kommentar: »Der XSL-Angriff ist kein Angriff. Er ist ein Traum.«

8.3.3 Biclique-Kryptoanalyse

Den bisher besten Angriff auf den AES fanden Andrey Bogdanov, Dmitry Khovratovich und Christian Rechberger im Jahr 2011 – also 14 Jahre nach Veröffentlichung des Verfahrens [BoKhRe]. Der Angriff ist eine Biclique-Kryptoanalyse, für den Mallory nur einige wenige Klartext-Geheimtext-Paare benötigt. Er ist etwa um das Vierfache schneller als die vollständige Schlüsselsuche. Präziser ausgedrückt benötigt Mallory für den Angriff $2^{126.1}$ Schritte bei einem 128-Bit-Schlüssel, $2^{189.7}$ Schritte bei 192 Bits und $2^{254.4}$ Schritte bei 256 Bits. Die vollständige Schlüsselsuche ist somit nicht mehr der beste bekannte Angriff auf den AES. Müssen wir uns deshalb Sorgen machen? Zunächst einmal nicht, denn der Angriff von Andrey Bogdanov, Dmitry Khovratovich und Christian Rechberger ist noch weit davon entfernt, praktisch verwertbar zu sein.

8.3.4 Weitere Angriffe

In den letzten Jahren wurden einige weitere theoretische Angriffe auf den AES veröffentlicht. Dazu gehört insbesondere eine Related-Key-Attacke aus dem Jahr 2009 von Alex Biryukov and Dmitry Khovratovich [BirKho]. Dieser Angriff betrifft die 192-Bit und die 256-Bit-Version des AES. Er benötigte zunächst einen Aufwand von 2^{119} Schritten, wurde jedoch auf $2^{99.5}$ verbessert. Zwar sind diese Angriffe nicht praxisrelevant, sie sorgenten jedoch wieder einmal für Diskussio-

nen über das zwar elegante, aber nicht mit viel Sicherheitspielraum ausgestattete Design des AES. In diesem Fall war die vergleichsweise einfache Schlüsselaufbereitung des AES die Ursache.

8.4 Bewertung des AES

Kein Zweifel, der Sieg von Rijndael beim AES-Wettbewerb kam nicht von ungefähr. Das Verfahren verfügt über ein elegantes Design, offenbarte in der Wettbewerbsphase keine Sicherheitslücken, und die Verschlüsselungsgeschwindigkeit ließ auf unterschiedlichen Plattformen nichts zu wünschen übrig. So setzte sich Rijndael in der Gunst der Experten gegen 14 Konkurrenten durch. Kaum jemand stellte diesen Sieg infrage. Allerdings bestand Einigkeit darüber, dass die Verfahren Serpent und Twofish ebenfalls würdige Gewinner gewesen wären. Am Ende war es eher eine Frage des Geschmacks, welchen der drei Algorithmen man bevorzugte. Serpent galt als eher konservativ – ohne neue Designelemente, mit einem großen Sicherheitspuffer ausgestattet, dafür relativ langsam. Rijndael wirkte im Vergleich dazu deutlich innovativer und eleganter, verzichtete auf eine allzu große Sicherheitsmarge und überzeugte durch eine hohe Verschlüsselungsgeschwindigkeit. Twofish lag zwischen diesen beiden Extremen.

Über ein Jahrzehnt nach Ende des AES-Wettbewerbs stellt sich die Situation etwas anders dar. Zwar gilt der AES nach wie vor als sehr sicher. Doch seit Bekanntwerden der Biclique-Kryptoanalyse weiß man, dass die vollständige Schlüsselsuche nicht die beste Methode ist, um ihn zu knacken. Zusammen mit den Überlegungen zur quadratischen Kryptoanalyse und der Darstellbarkeit als algebraische Formel hätte dies vermutlich ausgereicht, um den Sieg von Rijndael zu verhindern.

Interessant ist nun folgende Frage: Haben die Experten die Sicherheit von Rijndael überschätzt, als sie das Verfahren zum AES kürten, oder sind die vielen Kryptoanalyse-Ergebnisse darauf zurückzuführen, dass sich so viele mit dem Verfahren beschäftigten. Zweifellos spielt der letztgenannte Aspekt eine wichtige Rolle, denn seit Ende des Wettbewerbs haben Heerscharen von Kryptografen mit großem Aufwand nach Schwachstellen im AES gesucht. Niemand weiß, wie Serpent und Twofish heute dastehen würden, wenn sie so intensiv untersucht worden wären. Andererseits ist zu beachten: Serpent und Twofish haben ein konservativeres Design als der AES. Es ist daher durchaus möglich, dass diese beiden Verfahren den Kryptoanalytikern weniger Angriffsfläche geboten hätten. Interessant ist auf jeden Fall eine Lehre, die man aus der Geschichte des AES ziehen kann: Selbst wenn sich zahlreiche, weltweit führende Experten über 15 Jahre lang mit einem Verfahren beschäftigt haben, können noch interessante Sicherheitslücken entdeckt werden.

9 AES-Kandidaten

In diesem Kapitel schauen wir uns an, welche symmetrischen Verschlüsselungsverfahren neben dem Sieger Rijndael am AES-Wettbewerb beteiligt waren. Insgesamt nahmen 15 Verfahren teil. Neben Rijndael kamen in die engere Wahl: Serpent, Twofish, RC6 und MARS. Genauere Informationen zum Wettbewerb gibt es in Abschnitt 18.5.2.

9.1 Serpent

Serpent ist ein symmetrisches Verschlüsselungsverfahren [AnBiKn]. Es gehörte neben Rijndael und Twofish zu den drei Algorithmen, unter denen die Entscheidung im AES-Wettbewerb fiel. Entwickelt wurde Serpent von den drei bekannten Kryptografen Ross Anderson, Eli Biham und Lars Knudsen. Diese argumentierten, der AES-Sieger müsse mindestens ein Jahrhundert lang gegen alle Fortschritte der Kryptoanalyse gewappnet sein. Deshalb käme nur ein Design ohne

Experimente und mit hohen Sicherheitspuffern infrage. In der Tat gilt Serpent als der konservativste unter allen AES-Kandidaten.

9.1.1 Funktionsweise von Serpent

Serpent ist eine SP-Chiffre, die 128-Bit-Blöcke verarbeitet und Schlüssel der Länge 128, 192 und 256 Bit unterstützt. Da die drei Designer keine Experimente eingehen wollten, sahen sie lediglich S-Boxen und einige lineare Funktionen als Designbestandteile vor. Diese Werkzeuge sind seit den siebziger Jahren bekannt, kommen in zahlreichen Verfahren zur Anwendung und galten schon damals als gut erforscht.

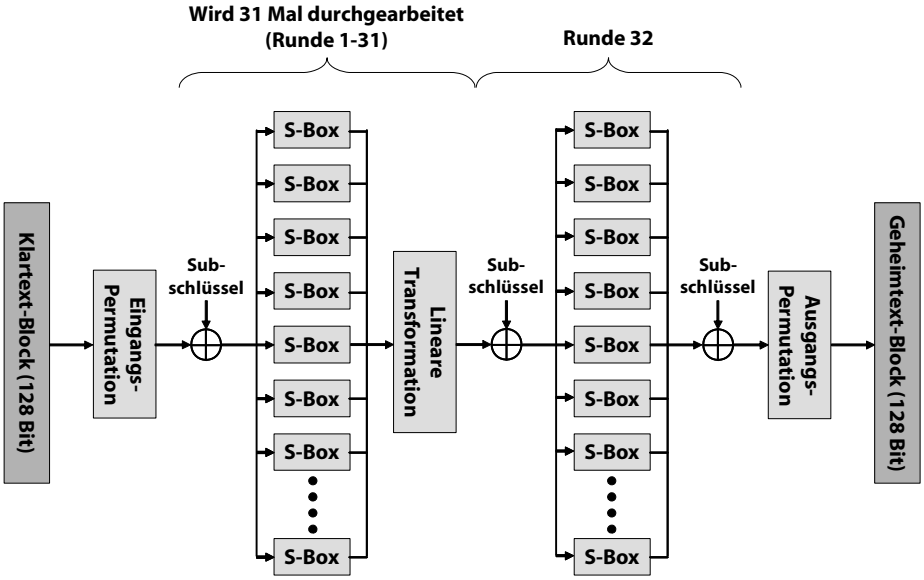


Abb. 9–1 *Serpent arbeitet in 32 Runden, wobei die letzte einen leicht geänderten Ablauf hat.*

Dass die drei Serpent-Entwickler kein Risiko eingehen wollten, zeigt sich auch daran, dass das Verfahren 32 Runden vorsieht (vergleichbare Verfahren verwenden meist acht bis 16 Runden). Wie diese 32 Runden ablaufen, zeigt Abbildung 9–1. Am Anfang einer Serpent-Verschlüsselung steht eine Permutation. Dabei werden die Klartext-Bits b_0, b_1, ..., b_{127} in folgende Reihenfolge gebracht: b_0, b_{32}, b_{64}, b_{96}, b_1, b_{33}, b_{65}, b_{97}, b_2, b_{34}, b_{66}, b_{98}, b_3, ... Danach folgen 31 Runden mit folgendem Ablauf:

1. *Schlüsseladdition*: Ein 128-Bit-Subschlüssel wird mit dem Eingabewert exklusiv-oder-verknüpft.
2. *S-Box*: Das Resultat wird in 32 S-Boxen verarbeitet, die jeweils vier Bit auf vier Bit abbilden (siehe unten).
3. *Lineare Transformation*: Die 128 Bit aus dem vorhergehenden Schritt werden in vier 32-Bit-Blöcke aufgeteilt und wie in Abbildung 9–2 zu sehen bearbeitet.

Die 32. Runde läuft gleich ab, allerdings wird die lineare Transformation durch eine zusätzliche Schlüsseladdition ersetzt. Der Grund für diese Maßnahme: Eine lineare Transformation am Ende des Ablaufs lässt sich leicht herausrechnen und ist damit kryptografisch wirkungslos. Am Ende des Ablaufs wird die initiale Permutation wieder aufgehoben. Da der (abgesehen von den Permutationen) erste und letzte Schritt von Serpent die Exklusiv-oder-Verknüpfung eines Subschlüssels vorsieht, ist ein Whitening gegeben. Serpent ist außerdem so aufgebaut, dass sich alle Schritte des Verfahrens gut umkehren lassen – eine Entschlüsselung ist daher auf naheliegende Weise möglich.

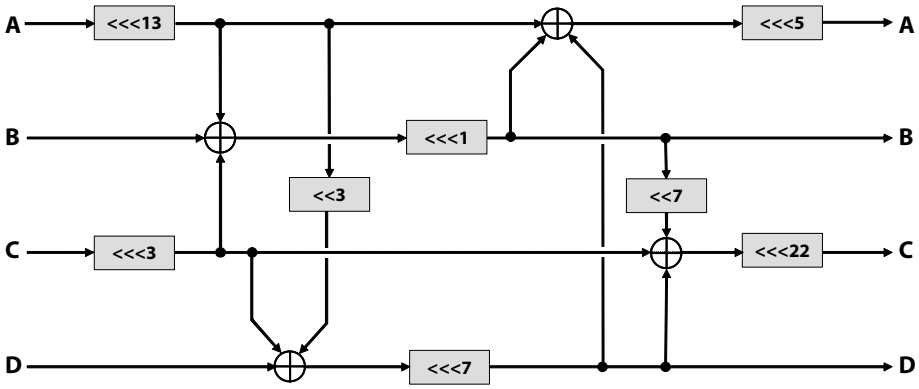


Abb. 9–2 *Die lineare Transformation von Serpent arbeitet mit Linksrotationen (»<<<«) und Linksverschiebungen (»<<«).*

9.1.2 S-Box-Design

In jeder Serpent-Runde kommen 32 S-Boxen zur Anwendung. Jede S-Box nimmt 4 Bit entgegen und gibt 4 Bit aus. Alle 32 S-Boxen innerhalb einer Runde sind identisch aufgebaut. Es gibt jedoch acht verschiedene S-Boxen (S_0, S_1, ..., S_7). In Runde 1 kommt ausschließlich S_0 zum Einsatz, in Runde 2 S_1 und so weiter. In Runde 8 wird S_7 eingesetzt, anschließend beginnt die Reihenfolge von vorn: Runde 9 setzt S_0 ein, Runde 10 setzt S_1 ein usw. Die folgende Tabelle zeigt die Ein- und Ausgabewerte der Serpent-S-Boxen:

Eingabe	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	3	15	8	0	1	15	7	1
1	8	12	6	15	15	5	2	13
2	15	2	7	11	8	2	12	15
3	1	7	9	8	3	11	5	0
4	10	9	3	12	12	4	8	14
5	6	0	12	9	0	10	4	8
6	5	5	10	6	11	9	6	2
7	11	10	15	3	6	12	11	11
8	14	1	13	13	2	0	14	7
9	13	11	1	1	5	3	9	4
10	4	14	14	2	4	14	1	12
11	2	8	4	4	10	8	15	10
12	7	6	0	10	9	13	13	9
13	0	13	11	7	14	6	3	3
14	9	3	5	5	7	7	10	5
15	12	4	2	14	13	1	0	6

Interessant ist, wie die drei Serpent-Designer diese S-Boxen entwickelten. Zunächst legten sie fest, welche Kriterien die S-Boxen erfüllen sollten. Dabei ging es vor allem darum, differenzielle und lineare Kryptoanalyse zu verhindern. Da diese beiden Angriffsmethoden inzwischen gut untersucht sind, ist es relativ einfach, Kriterien aufzustellen, bei deren Einhaltung der gewünschte Schutz gewährleistet ist. Auf Basis dieser Vorgaben nahmen Anderson, Knudsen und Biham die S-Boxen des DES und unterzogen diese einer fortwährenden Änderung nach einem zuvor definierten Ablauf. Diesen Ablauf setzten sie so lange fort, bis die gestellten Kriterien erstmals erfüllt waren. So entstand die erste S-Box. Anschließend wiederholten sie dieses Vorgehen sieben Mal, um die restlichen S-Boxen zu generieren. Die S-Boxen des DES dienten also als Leerer-Ärmel-Zahl.

9.1.3 Schlüsselaufbereitung von Serpent

Serpent benötigt 33 Subschlüssel zu je 128 Bit. Man kann auch von 132 Subschlüsseln zu je 32 Bit zu sprechen. Diese Subschlüssel werden mit K_0, K_1, ..., K_{131} bezeichnet. Jeweils vier Subschlüssel gehen in eine Runde ein – außer in der letzten Runde, dort sind es acht. Zur Schlüsselgenerierung benötigen wir außerdem 140 weitere 32-Bit-Blöcke, die W_{-8}, W_{-7}, W_{-6}, W_{-5}, ..., W_{131} genannt werden. In W_{-8} bis W_{-1} wird der Schlüssel geschrieben (falls der Schlüssel kürzer als 256 Bit ist, wird der Rest mit einer 1 gefolgt von lauter Nullen aufgefüllt). Anschließend gilt:

$$W_i = (W_{i-8} \oplus W_{i-5} \oplus W_{i-3} \oplus W_{i-1} \oplus 9E3779B9 \oplus i) <<< 11$$

Die Subschlüssel werden nun wie folgt generiert (S_0, S_1, ..., S_7 sind die S-Boxen):

$K_0, K_1, K_2, K_3 = S_3(W_0, W_1, W_2, W_3)$
$K_4, K_5, K_6, K_7 = S_2(W_4, W_5, W_6, W_7)$
$K_8, K_9, K_{10}, K_{11} = S_1(W_8, W_9, W_{10}, W_{11})$
$K_{12}, K_{13}, K_{14}, K_{15} = S_0(W_{12}, W_{13}, W_{14}, W_{15})$
$K_{16}, K_{17}, K_{18}, K_{19} = S_7(W_{16}, W_{17}, W_{18}, W_{19})$
...
$K_{124}, K_{125}, K_{126}, K_{127} = S_4(W_{124}, W_{125}, W_{126}, W_{127})$
$K_{128}, K_{129}, K_{130}, K_{131} = S_3(W_{128}, W_{129}, W_{130}, W_{131})$

9.1.4 Bewertung von Serpent

Der beste bekannte Angriff gegen Serpent ist die vollständige Schlüsselsuche. Reduziert man die Rundenzahl des Verfahrens, dann findet sich nach aktuellem Stand der Forschung erst bei einem Neun-Runden-Serpent eine Möglichkeit, schneller als mit Brute Force zum Ziel zu kommen [KeKoSc]. Der Sicherheitspuffer ist also groß. Abgesehen davon gehört Serpent zu den Verfahren, die gegenüber einer quadratischen Kryptoanalyse anfällig sein könnten [CouPie]. Es gibt jedoch ernsthafte Zweifel daran, dass auf diese Weise jemals ein Verfahren schneller als mit vollständiger Schlüsselsuche geknackt werden wird.

Aufgrund des großen Sicherheitspuffers von Serpent kann man vermuten, dass die zahlreichen theoretischen Schwächen, die beim AES inzwischen entdeckt wurden, bei Serpent nicht festgestellt worden wären. Das heutige Wissen vorausgesetzt, hätte Serpent den AES-Wettbewerb daher vermutlich gewonnen oder wäre hinter Twofish Zweiter geworden. Durch den großen Spielraum ist Serpent allerdings langsamer als viele andere Verfahren. Serpent ist damit vor allem für Anwendungen geeignet, in denen über Jahrzehnte hinweg eine hohe Sicherheit notwendig ist.

9.2 Twofish

Twofish [Schn07/2] ist neben Rijndael und Serpent das dritte Verfahren, das es in die Top 3 beim AES-Wettbewerb brachte. Der Name soll ausdrücken, dass es sich um eine Weiterentwicklung von Blowfish (Abschnitt 7.4.4) handelt. Zu Twofish gibt es ein ganzes Buch [SKWHFW]. Da dieses auch ausführlich auf den Designprozess eingeht, ist es eine Pflichtlektüre für jeden, der sich mit der Entwicklung neuer symmetrischer Verfahren beschäftigt. Wie alle AES-Kandidaten verschlüsselt Twofish 128-Bit-Blöcke und unterstützt Schlüssel der Länge 128, 192 und 256 Bit.

9.2.1 Funktionsweise von Twofish

Twofish ist eine Feistel-Chiffre, deren Verwandtschaft zum Vorgänger Blowfish deutlich erkennbar ist. Das Verfahren arbeitet in 16 Runden. Twofish sieht vier aus dem Schlüssel generierte S-Boxen vor, die jeweils zweimal pro Runde eingesetzt werden. Darüber hinaus gibt es einige im Vergleich zu Blowfish zusätzliche Operationen, die für eine stärkere Durchmischung der Zwischenergebnisse sorgen (Diffusion). Wie Blowfish nutzt das Verfahren Whitening: Jeweils vier 32-Bit-Rundenschlüssel werden am Anfang und am Ende mit den 128 Bit exklusiv-oder-verknüpft. Die Schlüsselaufbereitung ist im Vergleich zu Blowfish komplett neu gestaltet und deutlich schneller.

Die Funktionsweise von Twofish ist in Abbildung 9–3 dargestellt. Die vier Ablaufstränge verbergen, dass es sich um eine Feistel-Chiffre handelt (Feistel-Chiffren haben nur zwei Ablaufstränge). Fasst man jedoch jeweils zwei Stränge zusammen, dann ergibt sich (bis auf die Rotationen) das gewohnte Feistel-Schema. Die einzelnen Bestandteile haben folgende Bedeutung:

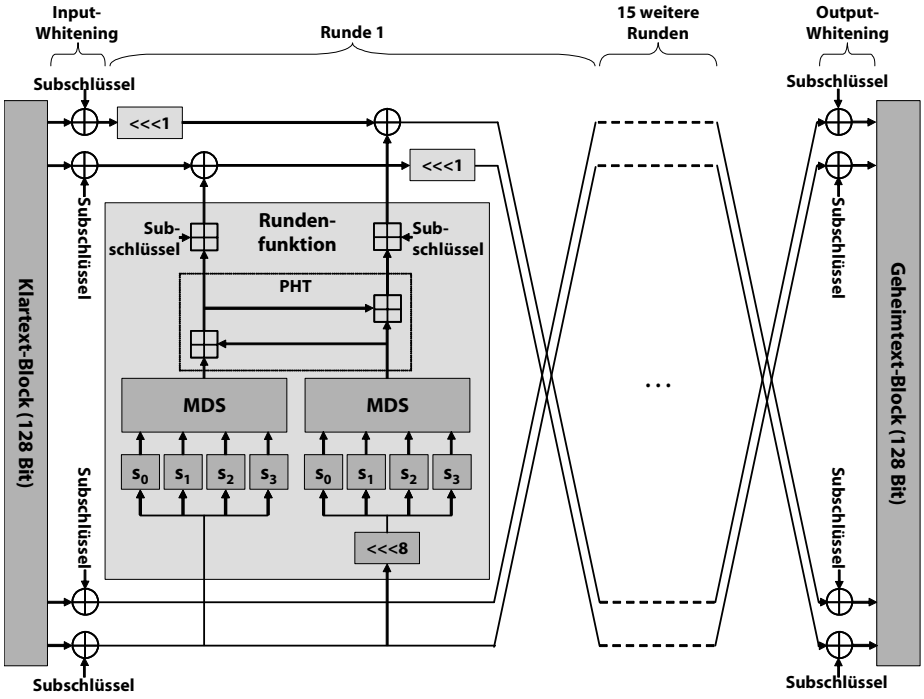


Abb. 9–3 *Twofish ist eine Feistel-Chiffre mit 16 Runden. Die Schlüssellänge beträgt 128, 192 oder 256 Bit.*

- *S-Boxen*: Es gibt vier S-Boxen, die jeweils 8 Bit auf 8 Bit abbilden. Deren Inhalt ist schlüsselabhängig und wird im Rahmen der Schlüsselaufbereitung festgelegt.

- *MDS*: Diese Komponente nimmt vier Bytes entgegen und führt eine Multiplikation mit einer festgelegten 4×4-Matrix durch. Das Ergebnis sind wiederum vier Bytes. Die Matrix-Multiplikation wird im Körper GF(2^8) durchgeführt (siehe Abschnitt 12.1.1). Den genauen Ablauf dieser Rechnung können Sie in [Schn07/2] nachlesen.
- *PHT*: Diese Komponente nimmt zwei 32-Bit-Werte entgegen und führt eine Pseudo-Hadamard-Transformation (PHT) durch. Das klingt kompliziert, läuft aber nach einer einfachen Formel ab (A und B seien die Eingabewerte, A' und B' die Ausgabewerte): $A' = A + B \bmod 256$; $B' = A + 2 \cdot B \bmod 256$.
- <<<: Dieses Zeichen steht für eine bitweise Linksrotation.

Abgesehen von den S-Boxen sind alle Schritte linear. Wie Sie leicht nachrechnen können, benötigt Twofish 40 Subschlüssel der Länge 32 Bit. Außerdem müssen die vier S-Boxen im Rahmen der Schlüsselaufbereitung generiert werden. Die Schlüsselaufbereitung sieht einige Multiplikationen in GF(2^8) vor und soll uns nicht im Detail interessieren. In [Schn07/2] finden Sie die genaue Spezifikation.

9.2.2 Bewertung von Twofish

In ihren Veröffentlichungen betonen die Twofish-Entwickler um Bruce Schneier, dass sie sich auf bewährte Designelemente verlassen und dadurch ein konservatives Verfahren geschaffen haben. Kryptografen ordnen Twofish meist zwischen dem eleganteren AES und dem noch konservativeren Serpent ein. Abgesehen von einer theoretischen Schwachstelle [MorYin] sind bisher keine Twofish-Sicherheitsprobleme bekannt. Wer den AES aus irgendwelchen Gründen nicht einsetzen möchte, hat mit Twofish eine gute Alternative zur Verfügung.

9.3 RC6

RC6 ist der Nachfolger von RC5 (Abschnitt 7.4.3). Ron Rivest entwickelte dieses Verfahren zusammen mit drei Kollegen, um damit am AES-Wettbewerb teilzunehmen [RiRoSY]. RC6 kam dabei unter die besten fünf, gehörte am Ende jedoch nicht zu den besten dreien. Dies lag zum einen an den datenabhängigen Rotationen, die als noch nicht genügend erforscht gewertet wurden, um sie in einem Verschlüsselungsstandard einzusetzen, zumal es in diesem Bereich eine unklare Patentlage gab. Außerdem erschien den AES-Juroren das Design von RC6 zu sehr auf 32-Bit-Prozessoren zugeschnitten. Werden 64-Bit-Prozessoren eingesetzt, dann ergeben sich Performanznachteile. Gleiches gilt in noch stärkerem Maße für 8- und 16-Bit-Prozessoren. Da das Verfahren MARS (siehe Abschnitt 9.4), das ebenfalls unter die ersten fünf kam, deutlich mehr Schwächen zeigte, kann man RC6 als Viertplatzierten des AES-Wettbewerbs betrachten (offiziell wurden keine Platzierungen vergeben).

9.3.1 Funktionsweise von RC6

Wie der Vorgänger RC5 unterstützt auch RC6 eine variable Blocklänge, Schlüssellänge und Rundenzahl. Für den AES-Wettbewerb waren jedoch die Bit- und Blocklängen in der besagten Form vorgegeben. Da der AES-Wettbewerb zudem keine variable Rundenzahl zuließ, legten die RC6-Entwickler um Ron Rivest diese auf 20 fest. Im Folgenden wollen wir uns auf die genannten Parameter beschränken. Die Funktionsweise von RC6 ist in Abbildung 9–4 zu sehen. Wie Vorgänger RC5 basiert auch RC6 in einem hohen Maß auf datenabhängigen Rotationen. Diese ersetzen die in anderen Verfahren üblichen S-Boxen. Folgende Unterschiede zu RC5 sind erwähnenswert:

- Nachdem RC2 mit vier und RC5 mit zwei Datenströmen arbeiteten, heißt es bei RC6 »back to the roots«. Das Verfahren sieht also wieder vier Datenströme vor (repräsentiert durch die vier gleich großen Teilblöcke *A*, *B*, *C* und *D*), die jeweils aus 32 Bit bestehen. Der Grund dafür: 32-Bit-Blöcke sind auf gegenwärtigen Prozessorarchitekturen am besten und schnellsten zu verarbeiten.
- Bei RC5 entsprechen zwei Runden einer Feistel-Runde. Bei RC6 kehrten Rivest und seine Kollegen von dieser etwas verwirrenden Definition ab. Eine RC6-Runde entspricht daher dem, was man als Betrachter intuitiv erwartet.

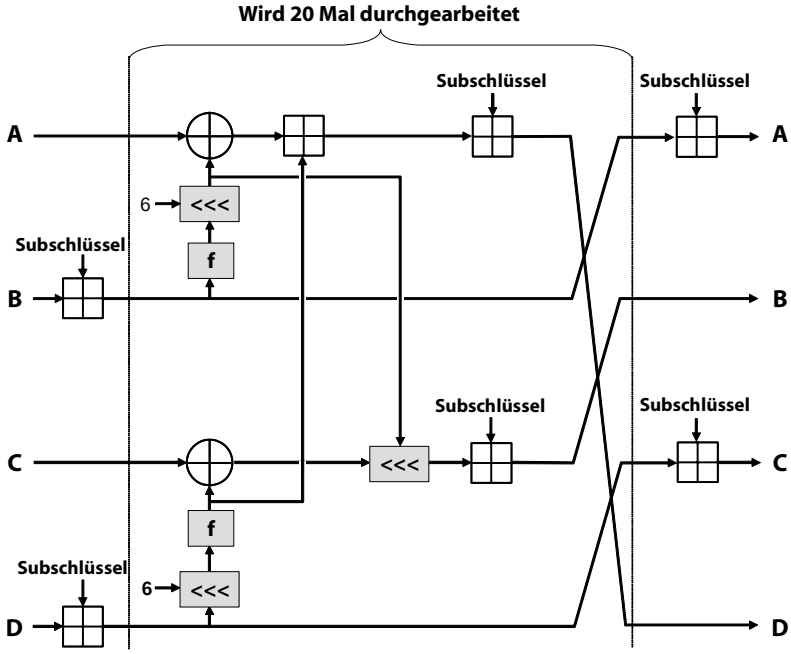


Abb. 9–4 *RC6 arbeitet in 20 Runden, wobei in jede Runde zwei Subschlüssel eingehen.*

- Achten Sie auf die Datenrotation im ersten Datenstrom. Diese wird von Daten aus dem dritten Datenstrom gesteuert (ähnlich wie bei RC5). Im Gegensatz zu RC5 werden die zur Steuerung verwendeten Daten jedoch durch die Funktion *f* und eine weitere Rotation bearbeitet. Dies hat zur Folge, dass nicht nur ein paar niederwertige Bits (wie bei RC5), sondern der gesamte Inhalt des Datenstroms die Rotation beeinflussen.
- Die Funktion *f* enthält eine Multiplikation modulo 2^{32}. Eine solche Operation kommt in RC5 nicht vor. Rivest und seine Kollegen führten sie ein, da Multiplikationen im Allgemeinen gute Diffusionseigenschaften besitzen.
- Vor Abarbeitung der 20 Runden sieht RC6 vor, dass zwei Subschlüssel zu den Datenströmen *B* und *D* addiert werden. Dies ist eine Form von Whitening. Damit soll verhindert werden, dass Angreifer Mallory den Wert von *A* und *C* am Anfang der zweiten Runde kennt. Eine analoge Funktion haben die beiden Subschlüssel-Additionen am Ende des Verfahrens.

Wie in Abbildung 9–4 erkennbar ist, arbeitet RC6 mit vier unterschiedlichen Funktionseinheiten. Die Rotationsfunktion (*ROT*) ist die einzige nichtlineare Einheit und erfüllt daher die Funktion einer S-Box. Es ist erstaunlich, dass eine so einfache Operation die Sicherheit des gesamten Verfahrens trägt. Auch die Funktion *f* und die Modulo-Addition, die für die Durchmischung sorgen, sind einfach gehalten. Das Einbringen der Subschlüssel durch eine Exklusiv-oder-Verknüpfung ist dagegen Standard.

9.3.2 Schlüsselaufbereitung von RC6

Insgesamt sieht RC6 44 Subschlüssel zu jeweils 32 Bit vor. Zwei davon werden am Anfang zu den Teilblöcken *B* und *D* gezählt, zwei weitere am Ende zu *A* und *C*. Darüber hinaus gehen in jeder der 20 Runden zwei Subschlüssel in die Verschlüsselung ein. Der Schlüssel ist als Array von 32-Bit-Wörtern gegeben und wird K_0, ..., K_{n-1} notiert, wobei *n* je nach Schlüssellänge den Wert 4, 6 oder 8 annimmt. Die 44 Subschlüssel werden als S_0, S_1, ..., S_{43} bezeichnet und in dieser Reihenfolge verwendet (S_0, S_1, S_{42} und S_{43} kommen also nicht in den 20 Runden, sondern davor bzw. danach zum Einsatz).

S_0 wird mit dem Wert B7E15163 initialisiert. Außerdem gilt: $S_1 = S_0 + 9E3779B9$, $S_2 = S_1 + 9E3779B9$, $S_3 = S_2 + 9E3779B9$, $S_4 = S_3 + 9E3779B9$ und so weiter. Die beiden (in hexadezimaler Schreibweise notierten) Konstanten sind Leerer-Ärmel-Zahlen: B7E15163 ist von den Nachkommastellen der Zahl *e* (2,7182...) abgeleitet, 9E3779B9 von den Nachkommastellen des Goldener-Schnitt-Verhältnisses (1,6180...). Nach der Initialisierung wird folgender Algorithmus abgearbeitet (»<<<« bezeichnet eine Linksrotation):

$X = Y = i = j = 0$
Für *Zähler* = 1 bis 132
$X = S[i] = (S[i] + X + Y) <<< 3$
$Y = K[j] = (K[j] + X + Y) <<< (X + Y)$
$i = (i + 1) \bmod 44$
$j = (j + 1) \bmod n$

9.3.3 Bewertung von RC6

Die Ähnlichkeit zu RC5 ist zwar noch zu erkennen, doch es ist klar, dass das RC6 im Vergleich zum Vorgänger an Komplexität zugenommen hat. Dennoch gehört RC6 zu den einfacheren der modernen symmetrischen Verschlüsselungsverfahren. Auch die Sicherheit ist nach aktuellem Kenntnisstand ausreichend hoch – es gibt keinen besseren bekannten Angriff als Brute Force. Der Sicherheitspuffer ist jedoch vergleichsweise knapp bemessen, denn man kennt immerhin einen erfolgreichen Angriff auf eine 17-Runden-Version des Verfahrens [ShKiHa]. RC6 ist zweifellos ein hervorragendes Verfahren, das modernen Ansprüchen genügt. Es steht jedoch im Schatten der erfolgreicheren AES-Kandidaten AES, Twofish und Serpent. Zudem ist es patentiert – nur im Falle eines Sieges beim AES-Wettbewerb wäre es zum Public-Domain-Verfahren geworden.

9.4 MARS

MARS ist nicht etwa ein Schokoriegel, sondern ein weiterer AES-Kandidat [BuCoDA]. Es zählte zu den fünf Finalisten des Wettbewerbs, schnitt von diesen aber am schlechtesten ab. Das Verfahren wurde von der Firma IBM entwickelt, die drei Jahrzehnte zuvor mit dem DES Pionierarbeit geleistet hatte. Zum MARS-Entwicklerteam gehörte Don Coppersmith, einer der DES-Väter. MARS hat eine Blocklänge von 128 Bit und unterstützt Schlüssel der Länge 128 bis 448 Bit (in 32-Bit-Schritten). Das Verfahren arbeitet mit vier Ablaufsträngen (wie RC6) und verwendet eine S-Box, die bei Bedarf in zwei Teile geteilt wird. MARS hat einen äußerst ungewöhnlichen Aufbau, der 32 Runden vorsieht – ungewöhnlich dabei ist, dass es vier unterschiedliche Rundentypen gibt. Eine Verschlüsselung läuft wie folgt ab:

1. Ein Subschlüssel wird addiert (Whitening).
2. Acht Runden *Forward Mixing*: Eine Runde dieses Typs verwendet die S-Box (in zweigeteilter Form) und ist damit nicht linear. In eine Forward-Mixing-Runde gehen keine Subschlüssel ein.
3. Acht Runden *Forward Transformation*: Eine Forward-Transformation-Runde bringt einen 64-Bit-Subschlüssel ein und wendet die S-Box (dieses Mal als Ganzes) an. Außerdem kommt eine datenabhängige Rotation (wie bei RC6) zum Einsatz.

4. Acht Runden *Backward Transformation*: Eine Runde dieses Typs läuft ab wie die Forward Transformation, allerdings in invertierter Form.
5. Acht Runden *Backward Mixing*: Eine Runde dieses Typs läuft ab wie das Forward Mixing, allerdings in invertierter Form.
6. Ein Subschlüssel wird addiert (Whitening).

Da die zweite Hälfte des Verfahrens die erste invertiert, entspricht das Entschlüsseln dem Verschlüsseln mit umgekehrter Reihenfolge der Subschlüssel. Die Schlüsselaufbereitung muss 16 64-Bit-Subschlüssel für die Runden zur Verfügung stellen, außerdem zwei 128-Bit-Subschlüssel für das Whitening. Zu diesem Zweck wird ein vergleichsweise aufwendiges Verfahren abgearbeitet, das die MARS-S-Box nutzt.

Die Mitwirkung von IBM und Don Coppersmith trug dazu bei, dass MARS anfangs zu den Favoriten des AES-Wettbewerbs gezählt wurde. Zunächst schienen diese Vorschusslorbeeren gerechtfertigt, denn das Verfahren überstand die erste Runde ohne Blessuren. Als es jedoch in die Endauswahl ging und die Experten die Finalisten genau unter die Lupe nahmen, kamen mehrere Ungereimtheiten ans Licht. Schneier und Kelsey fanden einen Angriff auf MARS mit reduzierter Rundenzahl, der die Vermutung nahelegte, dass die schlüsselunabhängige Gestaltung der Mixing-Runden ein Fehler war [KelSch]. Biham und Furman entdeckten eine theoretische Schwäche in den Transformationsrunden [BihFur]. Als äußerst peinlich für das MARS-Team erwies sich eine weitere Beobachtung: Die S-Box von MARS entspricht nicht den von den Entwicklern selbst angegebenen Kriterien [BuCaDM].

Keiner der genannten Mängel zog bis heute einen praktisch verwertbaren Angriff nach sich. Die Chancen im AES-Wettbewerb waren angesichts dieser Ungereimtheiten jedoch verspielt. Viele Beobachter hatten den Eindruck, dass das Design von MARS zwar gute Ideen enthielt, jedoch noch nicht ausgereift war. So ging das Verfahren als schlechtester der fünf AES-Finalisten in die Krypto-Geschichte ein.

9.5 SAFER

SAFER ist eine Abkürzung für »Secure And Fast Encryption Routine«. Es handelt sich dabei um ein symmetrisches Verschlüsselungsverfahren, dessen erste Version 1994 veröffentlicht wurde. Erfinder ist der bekannte Kryptograf James Massey. Dessen Ziel war es, mit SAFER eine Alternative zum in die Jahre gekommenen DES zu schaffen, was ihm auch gelungen ist. SAFER zählt inzwischen schon zu den älteren symmetrischen Verschlüsselungsverfahren, wird aber – nach mehreren Änderungen im Design – auch heute noch eingesetzt. SAFER gibt es in sieben unterschiedlichen Varianten. Die erste Variante mit dem Namen *SAFER K-64* (die Zahl 64 steht für die Schlüssellänge) entspricht dem ursprünglichen Design aus dem Jahr 1994. Ein Jahr später veröffentlichte Massey *SAFER K-128*,

um der Forderung nach einem 128-Bit-Schlüssel nachzukommen. Nachdem Kollegen Masseys nachgewiesen hatten, dass die Schlüsselaufbereitung des Verfahrens einige Schwächen aufwies, besserte der SAFER-Erfinder nach und veröffentlichte *SAFER SK-64* sowie *SAFER SK-128*. Zusätzlich veröffentlichte er dann noch *SAFER SK-40*, um den damaligen Exportrestriktionen der USA zu genügen. Die drei SK-Varianten unterscheiden sich vom Original nur durch die unterschiedliche Schlüsselaufbereitung. SK steht für »Secure Key Schedule«.

1998 erschien mit **SAFER+** die nächste SAFER-Version [KhKuMa]. An der Entwicklung waren neben Massey die beiden armenischen Kryptografen Gurgen Khachatrian und Melsik Kuregian beteiligt. SAFER+ nahm am AES-Wettbewerb teil, kam allerdings nicht in die engere Auswahl. Im Jahr 2000 erblickte schließlich **SAFER++** als NESSIE-Kandidat (siehe Abschnitt 18.5.4) das Licht der Krypto-Welt. Sowohl SAFER+ als auch SAFER++ weisen im Vergleich zum Ursprungsverfahren einige Änderungen auf, die über die Schlüsselaufbereitung hinausgehen. Im Folgenden will ich mich auf die Variante SAFER+ beschränken. Diese ist besonders wichtig, weil sie im weit verbreiteten Bluetooth-Standard zur Anwendung kommt (siehe Abschnitt 33.3).

9.5.1 Funktionsweise von SAFER+

SAFER+ bietet die im AES-Wettbewerb geforderten Schlüssellängen von 128, 192 und 256 Bit. Die Rundenzahl ist von der Schlüssellänge abhängig. Die Länge eines Klartextblocks beträgt 128 Bit. Das Verfahren zählt zu den SP-Chiffren. Eine SAFER+-Runde hat folgenden Ablauf (siehe auch Abbildung 9–5):

1. Jedes der 16 Bytes eines Blocks wird mit jeweils einem Byte aus dem ersten Subschlüssel verknüpft (ein Subschlüssel besteht ebenfalls aus 16 Bytes). Die Verknüpfung ist in acht von 16 Fällen eine Exklusiv-oder-Verknüpfung, ansonsten eine Addition.
2. Acht der 16 resultierenden Bytes werden der Funktion *EXP* zugeführt, die anderen der Funktion *LOG*. *EXP* und *LOG* sind S-Boxen, die jeweils 8 Bit auf nichtlineare Weise auf 8 Bit abbilden. Beide S-Boxen lassen sich mit einer mathematischen Formel beschreiben. Für *EXP* gilt: $y=45^x \bmod 257$. Für *LOG* gilt: $y=\log_{45} x \bmod 257$. x und y sind Byte-Variablen. In beiden Formeln wird für den Spezialfall $x=0$ als Ergebnis $y=0$ festgelegt. *EXP* und *LOG* verwenden Modulo-Rechnungen, wie sie in Abschnitt 10.1.1 beschrieben werden. Wie man leicht nachrechnet, ist *LOG* die Umkehrung von *EXP* (und umgekehrt). In der Praxis werden die beiden Formeln fast immer vorberechnet und als Ersetzungstabellen implementiert.

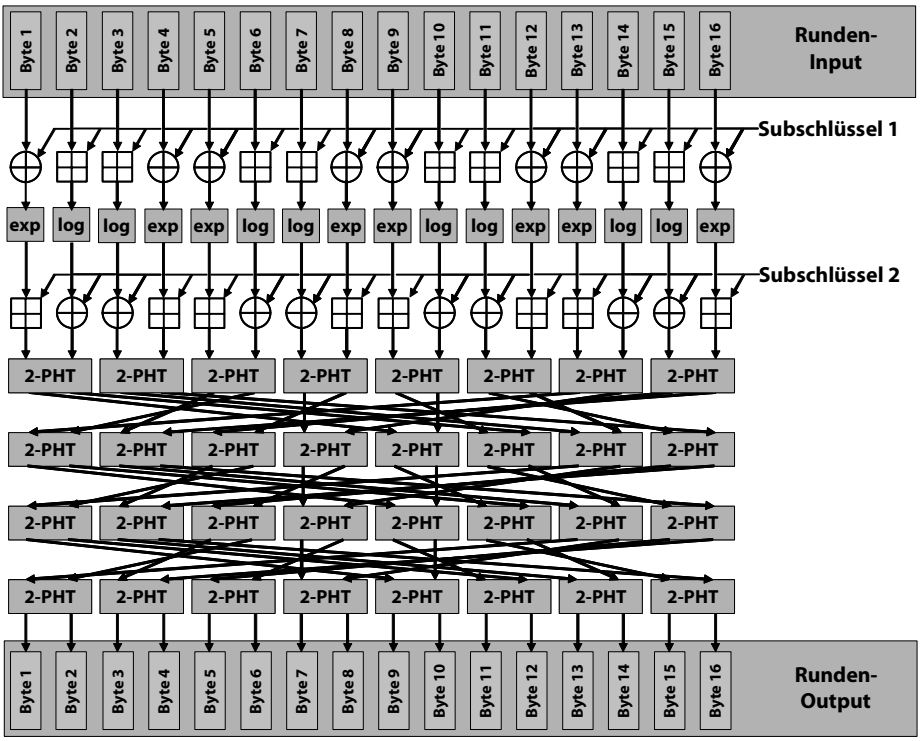


Abb. 9–5 *Eine Runde des Verschlüsselungsverfahrens SAFER+. Die Rundenzahl beträgt je nach Schlüssellänge 8, 12 oder 16.*

3. Jedes der 16 resultierenden Bytes wird mit einem Byte aus dem zweiten Subschlüssel verknüpft. Die Verknüpfung ist erneut entweder eine Exklusiv-oder-Verknüpfung oder eine Addition.
4. Jeweils zwei der resultierenden Bytes (b_1 und b_2) werden nach folgender Formel bearbeitet: $b_1=2b_1+b_2$ (mod 256); $b_2=b_1+b_2$ (mod 256). Diese Berechnung ist linear und wird auch als *Pseudo-Hadamard-Transformation* bezeichnet (sie spielt auch bei Twofish eine Rolle). Da in diesem Fall zwei Variablen bearbeitet werden, lautet die Abkürzung dafür *2-PHT*.
5. Die resultierenden Bytes werden durch eine Permutation in folgende Reihenfolge gebracht: 9, 12, 13, 16, 3, 2, 7, 6, 11, 10, 15, 14, 1, 8, 5, 4. Diese Umordnung wird als *Armenian Shuffle* (armenisches Mischen) bezeichnet, da sie von den beiden besagten armenischen Kryptografen eingeführt wurde.
6. Die armenisch gemischten Bytes werden wiederum jeweils im Zweierpack einer zweiten Pseudo-Hadamard-Transformation unterzogen.
7. Es folgt ein zweites armenisches Mischen.
8. Es folgt eine dritte Pseudo-Hadamard-Transformation.
9. Es folgt ein drittes armenisches Mischen.
10. Es folgt eine vierte Pseudo-Hadamard-Transformation.

Die 16 resultierenden Bytes sind der Runden-Output. Die Schritte 4 bis 10 sind alle linear und lassen sich daher zu einer einzigen Matrix-Multiplikation zusammenfassen. Die Rundenzahl von SAFER+ ist von der Schlüssellänge abhängig. Sie beträgt acht Runden für 128 Bit, 12 Runden für 192 Bit sowie 16 Runden für 256 Bit. Nach der letzten Runde folgt eine Teilrunde (*Output Transformation*), die nur aus Schritt 2 einer normalen Runde besteht (also aus einer Exklusiv-oder-Verknüpfung bzw. Addition eines Rundenschlüssels). Speziell für die Output-Transformation wird ein zusätzlicher Subschlüssel generiert. Da sowohl der erste als auch der letzte Schritt von SAFER+ aus der Addition bzw. Exklusiv-oder-Verknüpfung eines Subschlüssels besteht, liegt eine Form von Whitening vor.

Ein bekannter Nachteil bei einem SP-Verfahren besteht darin, dass die Entschlüsselung nicht identisch wie die Verschlüsselung abläuft. Vielmehr muss man bei einem SP-Verfahren jeden Teilschritt einzeln invertieren, um zurück zum Klartext zu kommen. Natürlich ist SAFER+ so konstruiert, dass alle Teilschritte umkehrbar sind. Die Entschlüsselung startet daher mit einer *Input Transformation* (Umkehrung der Output Transformation), gefolgt von acht bis 16 Runden, die jeweils den Verschlüsselungsvorgang invertieren.

9.5.2 Schlüsselaufbereitung von SAFER+

Die Schlüsselaufbereitung von SAFER+ ist vergleichsweise einfach und verzichtet auf S-Boxen und sonstige Verkomplizierungen. SAFER+ verarbeitet pro Runde zwei 16-Byte-Subschlüssel zuzüglich eines weiteren in der Output Transformation. Dies ergibt insgesamt 17, 25 oder 33 Subschlüssel. Die Schlüsselaufbereitung von SAFER+ sieht vor, dass an den (16, 24 oder 32 Byte langen) Schlüssel ein weiteres Byte gehängt wird. Aus diesem erweiterten Schlüssel werden nach einem festgelegten Schema jeweils 16 Byte entnommen und zu einer rundenabhängigen Konstante gezählt. Das Ergebnis ist der jeweilige Subschlüssel. Vor der Entnahme des nächsten Subschlüssels erfolgt jeweils eine interne Rotation der Schlüssel-Bytes.

9.5.3 Bewertung von SAFER+

Im Rahmen des AES-Auswahlprozesses entdeckten John Kelsey, Bruce Schneier und David Wagner eine theoretische Schwäche von SAFER+ [KeSW99]. Den dreien war aufgefallen, dass die Diffusionseigenschaft der Schlüsselaufbereitung nicht optimal ist. Dadurch verteilen sich die Schlüssel-Bits vergleichsweise langsam in die einzelnen Runden. Kelsey, Schneier und Wagner konnten diese Schwäche in eine Meet-in-the-Middle-Attacke auf die 256-Bit-Version von SAFER+ umwandeln. Zwar benötigt dieser Angriff 2^{240} Verschlüsselungsoperationen und ist damit für die Praxis völlig irrelevant. Die Forderung, dass die vollständige Schlüsselsuche der wirksamste Angriff sein soll, war damit jedoch verletzt. Dies war ein Grund für die Entwicklung von SAFER++.

Sieht man von diesem theoretischen Angriff ab, dann gilt SAFER+ als sicher. Mit seinen 128, 192 oder 256 Byte langen Schlüsseln bietet das Verfahren einen ausreichenden Schutz gegenüber einer vollständigen Schlüsselsuche. Die Blocklänge von 128 Bit ist ebenfalls sicher und zeitgemäß. Man kann also davon ausgehen, dass Mallory keine Chance hat, eine SAFER+-Nachricht zu knacken. Die Entwickler des Bluetooth-Sicherheitsmodells, die auf SAFER+ gesetzt haben, können ihre Entscheidung bisher als richtig betrachten.

9.6 CAST

CAST ist ein symmetrischer Verschlüsselungsalgorithmus von Carlisle Adams und Stafford Tavares. Die beiden Kryptografen entwickelten das Verfahren für die kanadische Firma Entrust, die es 1996 patentieren ließ. Trotz des Patents ist CAST ohne Einschränkungen kostenlos nutzbar. Neben zahlreichen Entrust-Produkten unterstützt auch PGP CAST. Die kanadische Regierung hat das Verfahren zur Nutzung für bestimmte staatliche Anwendungen freigegeben. Das Design von CAST setzt auf Variabilität: Sowohl die S-Boxen als auch die Schlüssellänge sind variabel. Die ursprüngliche Version wird auch als **CAST-128** bezeichnet. Dabei handelt es sich um eine Feistel-Chiffre, die eine Schlüssellänge zwischen 40 und 128 Bit vorsieht. Die Blocklänge beträgt 64 Bit. In [RFC2144] wird ein Spezialfall von CAST-128 beschrieben, der 128 Bit Schlüssellänge und feste S-Boxen verwendet.

Für den AES-Wettbewerb überarbeiteten Adams und Tavares ihr Verfahren (zusammen mit Howard Heys und Michael Wiener), wodurch CAST-256 entstand. Dieses wird unter anderem in [RFC2612] beschrieben. Gemäß den AES-Vorgaben hat CAST-256 eine Blocklänge von 128 Bit und unterstützt Schlüssel der Länge 128, 192 und 256 Bit. Außerdem können 160-Bit- und 224-Bit-Schlüssel eingesetzt werden, was jedoch im AES-Wettbewerb nicht gefordert war. CAST-256 ist keine Feistel-Chiffre, da es keine zwei, sondern vier Ablaufstränge gibt (ähnlich wie bei RC6). Man kann jedoch von einer verallgemeinerten Feistel-Struktur sprechen. Es gibt vier S-Boxen, die nicht variabel sind. Die Rundenzahl beträgt 48, ist also sehr hoch angesetzt.

CAST-128 gilt als sicher, wenn die entsprechenden Parameter richtig gewählt werden. Auch an CAST-256 gibt es diesbezüglich wenig auszusetzen. Das Verfahren hat den AES-Auswahlprozess überstanden, ohne dass irgendwelche Sicherheitslücken entdeckt wurden. Die guten Sicherheitseigenschaften sind vor allem auf die hohe Rundenzahl zurückzuführen, die für einen ordentlichen Puffer sorgt. Andererseits ist CAST-256 aus demselben Grund relativ langsam, was neben einem hohen Speicherbedarf den Ausschlag dafür gab, dass das Verfahren die zweite AES-Runde nicht erreichte. Vielleicht hätte die Sache anders ausgesehen, wenn Serpent, das ebenfalls ein konservatives Design mit hohem Sicherheitsspielraum vorsieht, nicht zu den AES-Kandidaten gezählt hätte.

9.7 MAGENTA

MAGENTA ist ein symmetrisches Verschlüsselungsverfahren, das von zwei Mitarbeitern der Deutschen Telekom entwickelt wurde (die Farbe Magenta ist Teil des Unternehmenslogos) [HubJac]. Der einzige deutsche AES-Kandidat war auch der einzige, dessen Funktionsweise bis zur Präsentation beim Wettbewerbsauftakt geheim gehalten wurde. Noch während MAGENTA-Miterfinder Michael Jacobson das Verfahren dem Auditorium vorstellte, entdeckten einige seiner Zuhörer eine mögliche Schwachstelle in der Schlüsselaufbereitung. Schon wenige Tage danach veröffentlichten vier Kryptografen (darunter Bruce Schneier und Adi Shamir) einen Angriff, der diese Schwäche nutzt [BBFKSS].

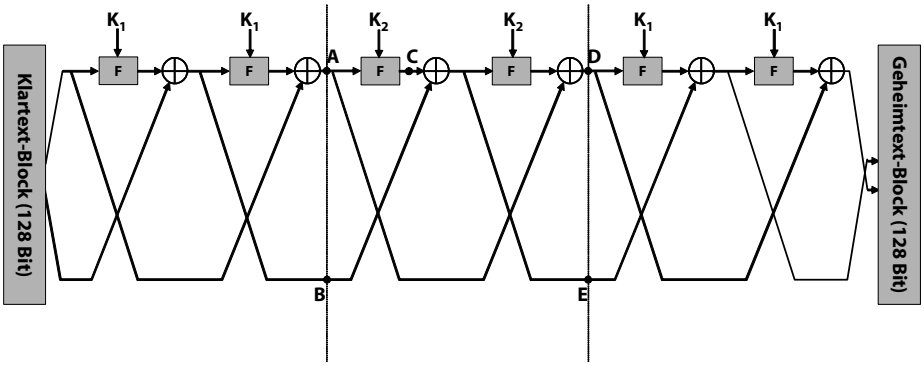


Abb. 9–6 *MAGENTA ist eine Feistel-Chiffre mit einer Schwäche in der Schlüsselaufbereitung.*

Der besagte Angriff funktioniert am besten bei der MAGENTA-Variante mit 128 Bit Schlüssellänge (nur diese Variante wollen wir im Folgenden betrachten). Der Angriff ist als Chosen-Plaintext-Attacke und (mit größerem Aufwand) als Known-Plaintext-Attacke realisierbar. Er handelt sich außerdem um eine Meet-in-the-Middle-Attacke. Die dem Angriff zugrunde liegende Beobachtung besteht darin, dass die Schlüsselaufbereitung von MAGENTA sehr simpel ist und keine ausreichende Diffusion bietet. Dadurch gehen bestimmte Schlüssel-Bits nur in bestimmte Runden ein.

Um den Angriff zu verstehen, muss man wissen, dass MAGENTA eine Feistel-Chiffre mit (in der 128-Bit-Variante) sechs Runden ist (siehe Abbildung 9–6). Das Design der Rundenfunktion F ist für den Angriff unerheblich. Pro Runde verarbeitet das Verfahren einen Subschlüssel der Länge 64 Bit. Die Schlüsselaufbereitung generiert in der 128-Bit-Variante zwar nur zwei Subschlüssel (K_1 und K_2), lässt diese jedoch mehrfach zum Einsatz kommen. K_1 geht in die Runden 1, 2, 5 und 6 ein, K_2 in die Runden 3 und 4.

Wie der MAGENTA-Angriff (als Chosen-Plaintext-Attacke) funktioniert, lässt sich am besten mit einem Blick auf die Abbildung erklären. Die dort eingezeichneten Variablen A, B, C, D und E spielen nur für die Beschreibung des

Angriffs eine Rolle. Da wir von einer Chosen-Plaintext-Attacke ausgehen, nehmen wir an, dass Angreifer Mallory eine Implementierung (Blackbox) zur Verfügung steht, die einen beliebigen Klartextblock M in den zugehörigen Geheimtextblock X verschlüsselt. Mallorys Ziel ist es, den von der Backbox verwendeten Schlüssel zu ermitteln. Zur Vorbereitung des Angriffs schiebt Mallory einen beliebigen Block M_1 in die Blackbox und erhält als Ergebnis den Geheimtext X_1. Für jeden möglichen Wert von K_1 (also insgesamt 2^{64} Mal) gilt nun folgender Ablauf (K_1' sei der jeweilige Kandidat für K_1):

1. Aus M_1 und K_1' berechnet Mallory die zugehörigen Werte von A und B (ein Blick auf die Abbildung macht klar, dass diese Berechnung möglich ist, ohne K_2 zu kennen). Diese beiden Werte werden A_1 und B_1 genannt.
2. Aus X_1 und K_1' berechnet Mallory die zugehörigen Werte von D und E (auch dies ist möglich, ohne K_2 zu kennen). Diese heißen D_1 und E_1.
3. Aus B_1 und E_1 berechnet Mallory den zugehörigen Wert von C. Dieser Wert heißt C_1.
4. Mallory belegt nun A und B neu mit den Werten A_2 und B_2. Es gilt A_2=A_1. B_2 erhält einen beliebigen Wert ungleich B_1. Mithilfe von K_1' berechnet Mallory anschließend den neuen Wert von M. Dieser heißt M_2.
5. Mallory schiebt M_2 in die Blackbox und erhält X_2.
6. Aus X_2 berechnet Mallory die neuen Werte von D und E. Diese heißen D_2 und E_2.
7. Aus B_2 und E_2 berechnet Mallory den neuen Wert von C. Dieser heißt C_2.
8. Mit C_1 und C_2 hat Mallory nun zwei Werte für C berechnet. Da A_1=A_2 gilt, muss auch C_1=C_2 gelten, falls K_1' der passende Schlüssel ist (ein Blick auf die Abbildung verrät, dass dies so sein muss). Gilt C_1=C_2, dann setzt Mallory den aktuellen Wert von K_1' auf seine Merkliste.

Hat Mallory dieses Prozedere für alle 2^{64} möglichen Subschlüssel durchgespielt, dann muss mindestens ein Wert auf seiner Merkliste stehen – dieser ist dann der richtige. Sollten mehrere Einträge vorhanden sein (viele können es nicht sein), dann ergibt ein kurzes Ausprobieren den richtigen Kandidaten. Um auf die beschriebene Weise zum Erfolg zu kommen (um also K_1 zu finden), muss Mallory 2^{64} Klartexte in die Blackbox schieben. Den Schlüssel K_2 findet er anschließend durch Brute Force. Insgesamt benötigt Mallory also maximal 2^{65} Arbeitsschritte. Dies ist zweifellos ein enormer Aufwand. Selbst wenn Mallory 1.000 Klartexte pro Sekunde durch die Blackbox schleusen kann, benötigt er über eine Milliarde Jahre zum Ermitteln des Schlüssels. Trotzdem machte diese Attacke MAGENTA zum mit Abstand unsichersten AES-Kandidaten. Eigentlich schade, denn das Verfahren hätte bei einer besseren Schlüsselaufbereitung sicherlich mehr erreichen können.

9.8 Die restlichen AES-Kandidaten

Werfen wir nun noch einen Blick auf die verbleibenden AES-Kandidaten. Fünf der 15 AES-Kandidaten traf die Höchststrafe: Sie offenbarten ernst zu nehmende Sicherheitslücken und schieden deshalb aus. Zwar ist bis heute kein Angriff auf eines der Verfahren bekannt, der sich in der Praxis nutzen ließe, doch die Anforderungen in der Kryptografie sind nun einmal hart. Hier die Liste der Unsicheren:

- *LOKI97*: Diese Feistel-Chiffre mit 16 Runden zählte zu den größten Enttäuschungen des AES-Wettbewerbs [BroPip]. Obwohl mit Jennifer Seberry und Josef Pieprzyk zwei namhafte Kryptografen an der Entwicklung mitgewirkt hatten, zeigten sich bei der Sicherheit einige Schwächen. Mit linearer Kryptoanalyse gelang eine Known-Plaintext-Attacke mit 2^{56} Klartextblöcken.
- *FROG*: Auch dieses Verfahren kann per linearer Kryptoanalyse mit einer Known-Plaintext-Attacke mit 2^{56} Klartextblöcken gebrochen werden [GeLeCh]. Die Performanz erwies sich ebenfalls als nicht berauschend.
- *DEAL*: Diesen Algorithmus kann man als Anpassung des DES an die AES-Kriterien bezeichnen [Knudse]. Dennoch zeigten sich einige potenzielle Sicherheitslücken in der Schlüsselaufbereitung. Die Performanz entspricht etwa der von Triple-DES, was sich als zu langsam erwies.
- *HPC*: Der kuriose Name der »Hasty Pudding Cipher« [Schroe] konnte nicht verhindern, dass das Verfahren in der ersten Runde ausschied. Dies lag unter anderem daran, dass HPC-Erfinder Richard Schroeppel einige exotische Funktionselemente in sein Verfahren eingebaut hatte, deren kryptografische Eigenschaften noch nicht ausreichend analysiert sind. Zudem erwies sich die Schlüsselaufbereitung als unsicher, da für eine große Anzahl an Schlüsseln jeweils 2^{30} äquivalente Schlüssel existieren.
- *MAGENTA*: siehe Abschnitt 9.7

Ebenfalls in der ersten Runde mussten fünf weitere Algorithmen die Segel streichen, obwohl keine nennenswerten Sicherheitsmängel bekannt wurden. Grund für das Ausscheiden war die geringe Performanz auf unterschiedlichen Plattformen oder der zu hohe Speicherverbrauch. Folgende Verfahren waren betroffen:

- *E2*: E2 ist eine Feistel-Chiffre und stammt aus Japan [KMAUOT]. E2 gilt als das beste Verfahren, das die zweite Runde nicht erreichte. Eine etwas zu geringe Performanz sowie ein knapp bemessener Sicherheitsspielraum gaben den Ausschlag.
- *CAST-256*: siehe Abschnitt 9.6
- *SAFER+*: siehe Abschnitt 9.5

- *CRYPTON*: CRYPTON stammt von einem südkoreanischen Kryptologen [LimHwa]. Es hat einige Ähnlichkeit mit Rijndael, erreicht jedoch dessen Verschlüsselungsgeschwindigkeit nicht.
- *DFC*: Diese aus Frankreich stammende Feistel-Chiffre schied ebenfalls wegen einer nicht ausreichenden Performanz aus [GGHNPP]. Vor allem auf Nicht-64-Bit-Architekturen erschien das Verfahren zu langsam.

9.9 Fazit

Zwölf Jahre nach Abschluss des AES-Wettbewerbs sind neben den fünf Finalisten nur noch die beiden anderweitig relevanten Verfahren CAST-256 und SAFER+ von Interesse. Unabhängig davon hat der Wettbewerb der kryptografischen Forschung zahlreiche neue Impulse verliehen und hat einige neue Erkenntnisse gebracht. So gesehen bedeutet der AES sicherlich mehr als nur ein besonders gutes Verschlüsselungsverfahren. Großen Anklang fand nicht zuletzt auch der AES-Wettbewerb an sich. Praktisch alle Beteiligten lobten die Fairness und die Transparenz, in der die Auswahl ablief.

10 Symmetrische Verschlüsselungsverfahren, die nach dem AES entstanden sind

Nach Abschluss des AES-Wettbewerbs im Jahr 2000 wurden einige Jahre lang nur wenige neue symmetrische Verschlüsselungsverfahren vorgestellt – der Bedarf war schließlich durch den AES und die anderen Teilnehmer erst einmal gedeckt. Da sich der AES bewährte und zum wichtigsten Verfahren seiner Art überhaupt wurde, stellt sich bis heute bei jedem neuen symmetrischen Verschlüsselungsalgorithmus erst einmal die Frage nach der Existenzberechtigung. In diesem Kapitel schauen wir uns die wichtigsten Verfahren an, die nach dem AES entstanden sind und aus dessen Schatten heraustreten konnten.

10.1 MISTY1, KASUMI und Camellia

MISTY1, KASUMI und Camellia sind drei aus Japan stammende symmetrische Verschlüsselungsverfahren, die besonders gut für Hardwareimplementierungen geeignet sind. Eine wichtige Rolle spielt vor allem KASUMI, das in allen Handys

eingesetzt wird, die den Mobilfunk-Standard UMTS (siehe Abschnitt 32.3.2) unterstützen.

10.1.1 MISTY1

MISTY1 ist ein 1996 veröffentlichtes symmetrisches Verschlüsselungsverfahren, das von der Firma Mitsubishi entwickelt wurde [Mats97]. »MISTY« steht einerseits für »Mitsubishi Improved Security Technology«, lässt sich jedoch auch aus den Initialen der fünf Entwickler des Verfahrens zusammenwürfeln: Matsui Mitsuru, Ichikawa Tetsuya, Sorimachi Jun, Tokita Toshio und Yamagishi Atsuhiro. Neben MISTY1 gibt es auch das ähnlich aufgebaute MISTY2, das jedoch keine große Rolle spielt. MISTY1 ist patentiert, allerdings für den nichtkommerziellen akademischen Einsatz kostenlos nutzbar. Das Verfahren wird im Internetstandard RFC 2994 beschrieben [RFC2994]. MISTY1 zählt zu den Feistel-Chiffren. Es verwendet einen Schlüssel der Länge 128 Bit und arbeitet wie der DES mit 64-Bit-Blöcken. Die Anzahl der Runden ist variabel, muss aber ein Vielfaches von Vier betragen (acht Runden werden empfohlen). Das Verfahren ist für Hardwareimplementierungen gedacht und in dieser Form besonders schnell. Da MISTY1 inzwischen vom recht ähnlichen KASUMI abgelöst wurde, will ich an dieser Stelle nicht näher darauf eingehen.

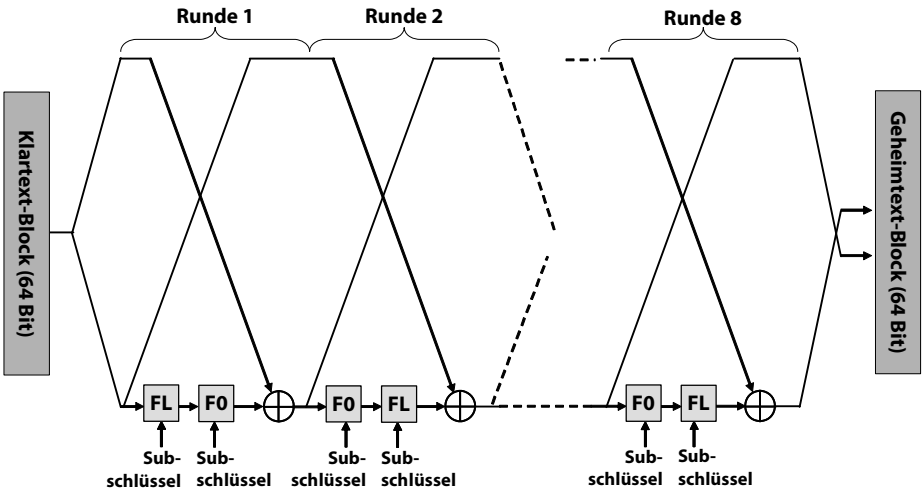


Abb. 10–1 *KASUMI ist eine Feistel-Chiffre, die vor allem für Hardwareimplementierungen geeignet ist.*

10.1.2 KASUMI

KASUMI ist eine Weiterentwicklung von MISTY1 aus dem Jahr 1999 [KASUMI]. Das Verfahren wurde speziell für den Einsatz im UMTS-Standard geschaffen. Die Änderungen gegenüber MISTY1 haben den Zweck, KASUMI für die UMTS-Rahmenbedingungen zu optimieren. KASUMI ist daher etwas schneller als MISTY, besser an die gängige Hardware angepasst und hat eine einfachere Schlüsselaufbereitung. Der Name ist von einer Figur aus der Zeichentrickserie Pokémon abgeleitet, die in der deutschen und englischen Fassung Misty, im japanischen Original jedoch Kasumi heißt. Durch die Nutzung im UMTS-Standard zählt KASUMI derzeit zu den am weitesten verbreiteten symmetrischen Verschlüsselungsverfahren überhaupt.

Funktionsweise von KASUMI

Wie in Abbildung 10–1 zu sehen, ist KASUMI eine Feistel-Chiffre. Der Aufbau ähnelt dem DES, es gibt jedoch nur acht Runden sowie keine Permutationen am Anfang und am Ende. Die Rundenfunktion von KASUMI setzt sich aus den beiden Einheiten *FL* und *FO* zusammen, deren Reihenfolge sich von Runde zu Runde umkehrt. Beide Einheiten nehmen einen jeweils 32 Bit langen Subschlüssel entgegen. Das Interessante am Design von KASUMI (ebenso von MISTY1) ist, dass der Inhalt von FL und FO erneut wie eine Feistel-Chiffre aussieht. Im Falle von FO ist dies ein Ablauf mit drei Runden, der einem Drei-Runden-DES ähnelt. Die Rundenfunktion bildet eine *FI* genannte Einheit, die einer weiteren Feistel-Chiffre (dieses Mal mit zwei Runden) entspricht. Deren Rundenfunktion ist eine einfache Funktion, die aus zwei einfachen Bit-Operationen besteht (diese sind nicht dieselbe für beide Runden). Der Inhalt von FI ist eine Feistel-Chiffre mit vier Runden, in deren Rundenfunktion jeweils eine von zwei S-Boxen zum Einsatz kommt.

Schlüsselaufbereitung

KASUMI benötigt in jeder der acht Runden einen Subschlüssel der Länge 128 Bit. Da die Schlüssellänge selbst ebenfalls 128 Bit beträgt, muss die Schlüsselaufbereitung eine Ausdehnung des Schlüssels auf das Achtfache bewerkstelligen. Aus der Funktionsweise von KASUMI ergibt sich zudem, dass jeder Subschlüssel in kleinere Einheiten aufgeteilt werden muss. Der in FL eingehende Teil heißt *KL* und besteht aus zwei 16-Bit-Blöcken. Die Schlüsselteile, die in FO und FI einfließen, heißen *KO* bzw. *KI* und haben jeweils eine Länge von dreimal 16 Bit. Der erste Schritt der Schlüsselaufbereitung sieht vor, dass der Schlüssel in acht 16-Bit-Blöcke aufgeteilt wird, die K_1, K_2,...,K_8 genannt werden. Durch die Exklusiv-oder-Verknüpfung von K_i mit jeweils einer Konstante C_i ($i=1, 2,..., 8$) entstehen acht weitere 16-Bit-Blöcke, die K_1', K_2',...,K_8' genannt werden. Es gilt also $K_i' = K_i \oplus C_i$. Die Konstanten nehmen folgende Werte an (in hexadezimaler Schreibweise):

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
0123	4567	89AB	CDEF	FEDC	BA98	7654	3210

Die folgende Tabelle gibt an, wie in den acht KASUMI-Runden die Schlüssel *KL*, *KO* und *KI* auf Basis der soeben definierten Werte generiert werden:

	KL		KO			KI		
Runde	**Block 1**	**Block 2**	**Block 1**	**Block 2**	**Block 3**	**Block 1**	**Block 2**	**Block 3**
1	K_1<<<1	K_3'	K_2<<<5	K_6<<<8	K_7<<<13	K_5'	K_4'	K_8'
2	K_2<<<1	K_4'	K_3<<<5	K_7<<<8	K_8<<<13	K_6'	K_5'	K_1'
3	K_3<<<1	K_5'	K_4<<<5	K_8<<<8	K_1<<<13	K_7'	K_6'	K_2'
4	K_4<<<1	K_6'	K_5<<<5	K_1<<<8	K_2<<<13	K_8'	K_7'	K_3'
5	K_5<<<1	K_7'	K_6<<<5	K_2<<<8	K_3<<<13	K_1'	K_8'	K_4'
6	K_6<<<1	K_8'	K_7<<<5	K_3<<<8	K_4<<<13	K_2'	K_1'	K_5'
7	K_7<<<1	K_1'	K_8<<<5	K_4<<<8	K_5<<<13	K_3'	K_2'	K_6'
8	K_8<<<1	K_2'	K_1<<<5	K_5<<<8	K_6<<<13	K_4'	K_3'	K_7'

Bewertung von KASUMI

KASUMI ist schnell in Hardware, und es gibt bisher keine praxisrelevanten Angriffe. 2005 veröffentlichten drei israelische Kryptografen jedoch eine theoretisch interessante Schwäche, die zeigt, dass die vollständige Schlüsselsuche in bestimmten Szenarien nicht der beste Angriff auf KASUMI ist [BiDuKe]. Es handelt sich dabei um eine Related-Key-Attacke, die neben einer speziellen Schlüsselsituation etwa 2^{54} gewählte Klartexte und 2^{76} Verschlüsselungsoperationen erfordert. Im Jahr 2010 wurde eine weitere Related-Key-Attacke bekannt [DuKeSh]. Die Autoren zeigen in ihrer Arbeit, dass die im Vergleich zu MISTY1 vereinfachte Schlüsselaufbereitung von KASUMI weniger sicher ist. Wer KASUMI einsetzen will, sollte die weitere Entwicklung der Kryptoanalyse zu diesem Verfahren im Auge behalten.

10.1.3 Camellia

Camellia ist der Nachfolger von KASUMI [AIKMMN]. Das im Jahr 2000 von Mitsubishi und NEC veröffentlichte Verfahren hat ein ähnliches Design wie der Vorgänger, bietet jedoch an den AES angelehnte Schlüssel- und Blocklängen (128, 192 oder 256 Bit Schlüssellänge, 128 Bit Blocklänge). Seine Entwickler reichten Camellia bei den beiden Krypto-Wettbewerben NESSIE und CRYPTREC ein, wo es gut abschnitt (siehe Abschnitt 38.2.3). Camellia ist eine Feistel-Chiffre mit 18 (bei 128 Bit Schlüssellänge) oder 24 Runden (bei 192 oder 256 Bit Schlüssellänge). Es verwendet Whitening. Camellia ist patentiert, aber für nichtkommerzielle Zwecke kostenlos einsetzbar.

10.2 CLEFIA

CLEFIA ist ein symmetrisches Verschlüsselungsverfahren, das von der Firma Sony entwickelt wurde [Sony]. Der Name ist vom französischen Wort »clef« (»Schlüssel«) abgeleitet. Mit einer Blocklänge von 128 Bit und einer Schlüssellänge von 128, 192 oder 256 Bit hat es die gleichen Parameter wie der AES. Es soll im Bereich Digital Rights Management (siehe Abschnitt 38.7) eingesetzt werden. Clefia wurde von der internationalen Standardisierungsorganisation ISO standardisiert [ISO29192-2]. Im gleichen Standard wird auch das Verschlüsselungsverfahren PRESENT (siehe Abschnitt 10.3.2) zur Norm erhoben.

10.2.1 Funktionsweise von CLEFIA

CLEFIA ist eine verallgemeinerte Feistel-Chiffre. Es gibt vier Ablaufstränge, in denen jeweils ein 32-Bit-Wert verarbeitet wird. Die Rundenzahl ist von der Schlüssellänge abhängig und beträgt 18 (128 Schlüsselbits), 22 (192 Schlüsselbits) oder 26 (256 Schlüsselbits). Im Folgenden wollen wir uns auf die einfachste Variante (18 Runden, 128 Bit Schlüssellänge) beschränken. Der Ablauf einer CLEFIA-Verschlüsselung ist in Abbildung 10–2 zu sehen. Pro Runde werden je einmal die Funktionen F_0 und F_1 aufgerufen. Alle Bestandteile außerhalb dieser beiden Funktionen sind linear. Am Ende jeder Runde (außer der letzten) findet eine Permutation der vier Ablaufstränge statt.

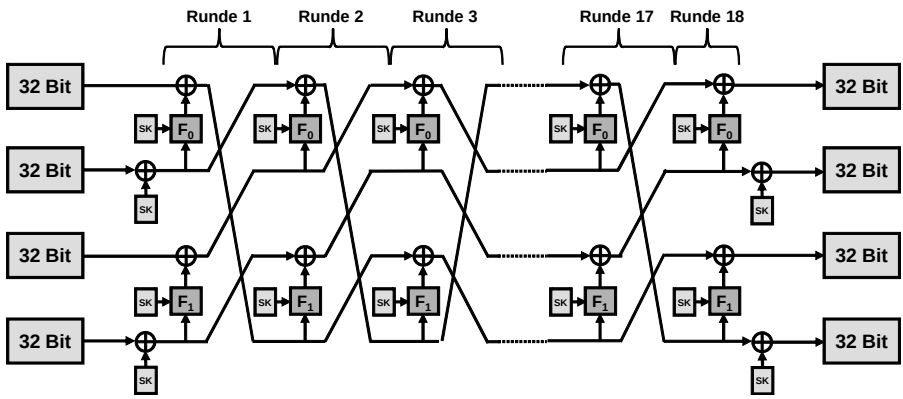


Abb. 10–2 *CLEFIA ist eine verallgemeinerte Feistel-Chiffre, die für den Einsatz im Digital Rights Management entwickelt wurde.*

In jeden Aufruf von F_0 und F_1 geht je ein Subschlüssel der Länge 32 Bit ein (auf die Schlüsselaufbereitung will ich nicht näher eingehen). Vier zusätzliche Subschlüssel gleicher Länge werden für ein Whitening verwendet. Zwei davon werden am Anfang, die beiden anderen am Ende mit zweien der Ablaufstränge exklusiv-oder-verknüpft. Man beachte, dass das Whitening jeweils nur die Hälfte

der Bits eines Blocks beeinflusst (partielles Whitening). Nach Angaben der CLEFIA-Designer reicht ein partielles Whitening für die auf vier Ablaufsträngen basierende Struktur von CLEFIA aus, um den gewünschten Effekt (Unkenntlichmachen der Ein- und Ausgabe) zu erzielen.

Die Entschlüsselung mit CLEFIA funktioniert ähnlich wie die Verschlüsselung. Es wird der gleiche Ablauf abgearbeitet, nur die Subschlüssel kommen in umgekehrter Reihenfolge zum Einsatz, und die Permutation der Ablaufstränge wird umgedreht. Der Aufbau der Funktionen F_0 und F_1 ist in Abbildung 10–3 dargestellt. Beide sehen zunächst die Addition des Subschlüssels vor, anschließend wird eine S-Box aufgerufen und schließlich folgt eine Matrix-Multiplikation (M_0 bzw. M_1). Dieser Ablauf ähnelt einer Runde einer SP-Chiffre – erst wird der Schlüssel eingebracht, dann wird mit S-Boxen Konfusion erzeugt, dann folgt eine lineare Operation (die Matrix-Multiplikation) für die Diffusion.

Der Unterschied zwischen F_0 und F_1 besteht zum einen in einer unterschiedlichen Matrix, mit der multipliziert wird, und zum anderen in der unterschiedlichen Reihenfolge der S-Boxen. Es werden zwei S-Boxen unterschieden: s_0 und s_1. Beide lassen sich in Form von Ersetzungstabellen angeben (für jeden 8-Bit-Wert liefert eine S-Box eine 8-Bit-Ausgabe). Die Tabellen der beiden S-Boxen wurden jedoch auf völlig unterschiedliche Weise generiert. Während sich s_0 auf vier kleinere S-Boxen (jeweils 4 Bit Ein- und Ausgabe) zurückführen lässt, basiert s_1 (ähnlich wie die AES-S-Box) auf einer Multiplikation in GF(2^8).

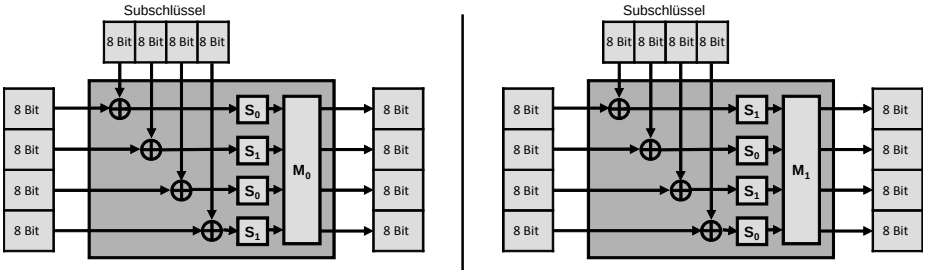


Abb. 10–3 *Die Funktionen F_0 (links) und F_1 (rechts) sind die zentralen Bausteine von CLEFIA.*

10.2.2 Bewertung von CLEFIA

CLEFIA ist noch ein recht junges Verfahren. Interessant daran ist, dass es einige altbekannte Designelemente (verallgemeinerte Feistel-Chiffre, S-Boxen) mit einigen neuen (partielles Whitening, unterschiedlich generierte S-Boxen) kombiniert. Sicherheitsmängel sind bisher nicht bekannt. Die nächsten Jahre werden zeigen, ob CLEFIA sich durchsetzt.

10.3 Schlanke Verschlüsselungsverfahren

Es gibt eine erhebliche Nachfrage nach Verschlüsselungsverfahren, die in extrem ressourcenbeschränkten Umgebungen zum Einsatz kommen – auf Chipkarten, Sensoren, RFID-Chips und ähnlichen Plattformen (siehe Kapitel 23). Praktisch alle praxisrelevanten symmetrischen Algorithmen wurden daher mit dem Ziel entwickelt, dass sie auch in ressourcenschwachen Umgebungen einsetzbar sind. Inzwischen gibt es auch Verschlüsselungsverfahren, die speziell für solche Low-End-Plattformen geschaffen wurden und auf diesen gegenüber Allroundern wie dem AES verschiedene Vorteile bieten. Diese will ich als **schlanke Verschlüsselungsverfahren** bezeichnen. Für schlanke Verschlüsselungsverfahren werden meist folgende Anforderungen genannt:

- *Geringe Hardwareanforderungen*: Der Hardwarebedarf wird in diesem Zusammenhang meist in logischen Gattern angegeben. Im Jahr 2002 forderte der einflussreiche RFID-Experte Sanjay Sarma ein Verschlüsselungsverfahren, das sich mit maximal 2.000 Gattern implementieren lässt [Schm12/2]. Der AES benötigt etwa 3.000 Gatter (es geht auch mit weniger, das Verfahren wird dadurch aber deutlich langsamer).
- *Geringer Energieverbrauch*: Ein Batteriewechsel ist auf vielen ressourcenschwachen Plattformen aufwendig oder sogar unmöglich. Der Vorteil eines energiesparenden Verfahrens ist daher offensichtlich. Der Energieverbrauch steigt etwa proportional zur Anzahl der Gatter. Bei der Forderung nach möglichst wenig Gattern ist das Energiesparen meist ein wichtigeres Argument als die Hardwarekosten. Außerdem gilt naturgemäß: Je performanter ein Verfahren arbeitet (also je weniger Taktzyklen es benötigt), desto weniger Energie wird verbraucht.
- *Hardwareorientierung*: Schlanke Verschlüsselungsverfahren werden meist direkt in Hardware implementiert. Dies muss beim Design berücksichtigt werden.

Nicht ganz so entscheidend ist bei schlanken Verfahren dagegen eine hohe Sicherheitsstufe. Zwar muss auch die Verschlüsselung eines ressourcenschwachen Mini-Computers sicher sein, doch Staatsgeheimnisse werden auf Chipkarten und RFID-Tags normalerweise nicht verschlüsselt. Auch die Performanz steht oft nicht im Mittelpunkt, da die verarbeiteten Datenmengen auf beschränkten Plattformen meist klein sind.

Die obige Liste und die Einschränkungen sind jedoch nur als Anhaltspunkte zu verstehen. In der Praxis können die Anforderungen an ein schlankes Verschlüsselungsverfahren durchaus unterschiedlich sein. So ist auf einer kontaktbehafteten Chipkarte der Energieverbrauch kein großes Thema, da es eine externe Energieversorgung gibt. Genauso wenig spielen auf einem Herzschrittmacher Hardwarekosten im Cent-Bereich eine große Rolle. Bereits jetzt ist daher zu erkennen, dass es innerhalb der schlanken Verschlüsselungsverfahren Spezialisie-

rungen gibt – beispielsweise kann sich ein Chiffren-Designer einen geringen Energieverbrauch bei moderater Verschlüsselungsgeschwindigkeit zum Ziel setzen. In den nächsten Jahren wird sich dieser Trend sicherlich fortsetzen.

10.3.1 SEA

Zu den ältesten schlanken Verschlüsselungsverfahren zählt **SEA** (**Scalable Encryption Algorithm**) [GePiQS]. SEA wurde von dem bekannten belgischen Kryptografen Jean-Jacques Quisquater zusammen mit drei Kollegen entwickelt. Man beachte, dass »SEA« rückwärts geschrieben die Buchstabenfolge »AES« ergibt. SEA lässt eine variable Block- und Schlüssellänge sowie eine variable Rundenzahl zu. Die Block- und die Schlüssellänge müssen jedoch gleich sein. Im Folgenden will ich mich auf den Spezialfall $SEA_{96,8}$ beschränken, der eine Block- und Schlüssellänge von je 96 Bit vorsieht (die Zahl 8 im Namen zeigt an, dass das Verfahren mit 8-Bit-Wörtern arbeitet). Die betrachtete Rundenzahl beträgt 92 und folgt damit einer Empfehlung der SEA-Entwickler für die genannten Parameter.

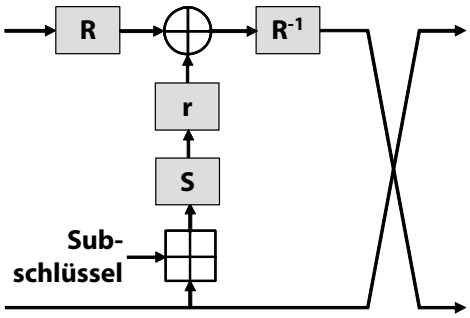


Abb. 10–4 *Eine Runde des symmetrischen Verschlüsselungsverfahrens SEA*

Die Funktionsweise von SEA ähnelt einer Feistel-Chiffre. Der Ablauf einer SEA-Runde ist in Abbildung 10–4 zu sehen. Die Konfusionseigenschaft des Verfahrens wird durch die (nichtlineare) Funktion *S* sichergestellt. *S* enthält 16 identisch aufgebaute S-Boxen, die jeweils drei Eingabe-Bit auf drei Ausgabe-Bit abbilden. Die 48 Eingabe-Bits der Funktion *S* werden in Dreierblöcke aufgeteilt und diesen S-Boxen zugeführt. Eine SEA-S-Box hat folgende Ein- und Ausgabewerte:

Eingabewert	000	001	010	011	100	101	110	111
Ausgabewert	000	101	110	111	100	011	001	010

Die S-Box-Architektur von SEA ist sehr spartanisch. Während beispielsweise der DES acht unterschiedliche S-Boxen der Größe 6×4 vorsieht, benötigt SEA nur einen S-Box-Typ, der zudem kleiner als beim DES ist. Auch die anderen Funktionselemente von SEA (sie sind alle linear) sind sehr schlicht aufgebaut:

- *R*: Dies ist eine byteweise Rechtsrotation des Sechs-Byte-Werts, den die Funktion entgegennimmt.
- *r*: Diese Funktion nimmt ebenfalls sechs Bytes entgegen. Das erste und das vierte Byte werden jeweils um eins nach rechts rotiert. Das zweite und das fünfte Byte bleiben unverändert. Das dritte und das sechste Byte werden jeweils um eins nach links rotiert.

Die Schlüsselaufbereitung von SEA ist ebenfalls sehr ressourcensparend realisiert. 92 Subschlüssel werden benötigt (einer pro Runde). Zur Schlüsselaufbereitung wird der 96-Bit-Schlüssel in zwei Teile geteilt, wobei die ersten zwei Subschlüssel entstehen. Abbildung 10–5 zeigt, nach welchem Ablauf in 91 Aufbereitungsrunden die weiteren Subschlüssel generiert werden. Die ersten 46 Subschlüssel ergeben sich aus den oberen 48 Bit des jeweiligen Blocks, danach kommen jeweils die unteren 48 Bit zum Einsatz. Beachten Sie, dass die Schlüsselaufbereitung einen sehr ähnlichen Ablauf hat wie die Verschlüsselung. Dies spart Speicherplatz. Der Wert *C*, der in jede Aufbereitungsrunde eingeht, besteht aus fünf Nullbytes gefolgt von einem Byte, das der Rundennummer entspricht. Andere symmetrische Verschlüsselungsverfahren verwenden für vergleichbare Konstanten kompliziertere Werte. SEA verzichtet auf eine solche Verkomplizierung, um Speicherplatz einzusparen.

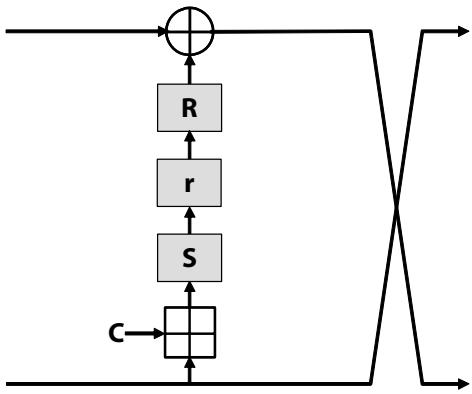


Abb. 10–5 *Die SEA-Schlüsselaufbereitung generiert für jede Runde einen Subschlüssel.*

Auffällig an der Schlüsselaufbereitung ist, dass der erste und der letzte Subschlüssel gleich sind. Gleiches gilt für den zweiten und den vorletzten sowie für den dritten und den drittletzten Subschlüssel usw. Auch diese Besonderheit macht SEA zu einem besonders ressourcenschonenden Verfahren, und zwar aus zwei Gründen. Zum einen läuft die SEA-Entschlüsselung genauso ab wie die Verschlüsselung, außer dass R und R^{-1} vertauscht werden – beim AES beispielsweise ist die Unterscheidung zwischen Ver- und Entschlüsselung aufwendiger. Zum anderen muss eine SEA-Implementierung den ursprünglichen Schlüssel nicht zwischenspeichern, was ebenfalls Speicherplatz spart.

Die Einfachheit von SEA wird teilweise durch eine vergleichsweise geringe Verschlüsselungsgeschwindigkeit erkauft, die wiederum eine Folge der hohen Rundenzahl ist. Über die Sicherheit von SEA lässt sich bisher wenig sagen, da das Verfahren noch zu neu ist. Bisher sind keine Schwachstellen bekannt. Nach Angaben der Erfinder ist SEA nicht gegenüber differenzieller und linearer Kryptoanalyse anfällig.

10.3.2 PRESENT

PRESENT ist eine »Ultra-Leichtgewichts-Blockchiffre«, die von acht Wissenschaftlern aus drei Ländern entwickelt wurde [BKLPPR, Schm12/2]. Dazu gehören Andrey Bogdanov, Gregor Leander, Christof Paar und Axel Poschmann von der Ruhr-Universität Bochum. Der Name PRESENT erklärt sich dadurch, dass das Verfahren überall präsent sein soll – beispielsweise auf RFID-Chips und anderen ressourcenschwachen Plattformen. PRESENT ist auf eine Implementierung in Hardware ausgelegt. Neben der Sicherheit stand bei der Entwicklung vor allem ein geringer Energieverbrauch im Vordergrund. Auch die Performanz ist – nach Angaben der Entwickler – ausgesprochen hoch.

PRESENT ist eine SP-Chiffre mit 31 Runden. Die Schlüssellänge beträgt wahlweise 80 oder 128 Bit, die Blocklänge 64 Bit. Die Entwickler empfehlen die Verwendung von 80-Bit-Schlüsseln. Eine PRESENT-Runde läuft in drei Schritten ab:

1. *AddRoundKey*: In diesem Schritt wird ein Subschlüssel der Länge 64 Bit mit dem Klartextblock bzw. mit dem Ergebnis der vorhergehenden Runde exklusiv-oder-verknüpft.
2. *sBoxlayer*: In diesem Schritt werden die 64 Bit in 16 Einheiten zu je vier Bit aufgeteilt. Jede Vier-Bit-Einheit wird einer 4×4-S-Box zugeführt.
3. *pLayer*: In diesem Schritt wird die Reihenfolge der 64 Bit geändert.

Nach der 31. und letzten Runde erfolgt ein weiterer Aufruf von AddRoundKey. Sowohl der erste als auch der letzte Schritt des Verfahrens besteht somit aus der Exklusiv-oder-Verknüpfung eines Subschlüssels – dies ist eine Form von Whitening. Die Schlüsselaufbereitung muss in der 80-Bit-Variante 32 Subschlüssel der Länge 64 Bit bereitstellen. Der erste Subschlüssel besteht aus den ersten 64 Bit des Schlüssels. Anschließend wird der Schlüssel um 61 Positionen nach links rotiert, wobei die vier ganz links stehenden Bits durch die S-Box geschickt werden. Zusätzlich wird ein 5-Bit-Rundenzähler mit den Bits Nummer 19 bis 15 (die Bits sind von links beginnend von 63 bis 0 durchnummeriert) exklusiv-oder-verknüpft.

Inzwischen gibt es über zehn Forschungsarbeiten, die eine Kryptoanalyse von PRESENT zum Inhalt haben. Beunruhigende Ergebnisse haben sich bisher keine ergeben – PRESENT kann also als sicher gelten. Dies sah auch die ISO (siehe Abschnitt 18.1.1) so und machte das Verfahren zu einem offiziellen Standard

[ISO/IEC 29192]. Diese Standardisierung könnte dazu beitragen, dass PRESENT in den nächsten Jahren deutlich an Bedeutung gewinnt, denn schlanke Verschlüsselungsverfahren sind gefragt, und die meisten Konkurrenten von PRESENT haben keinen Standardstatus und sind zudem nicht so gut untersucht. PRESENT könnte dadurch zu einem der wenigen praktisch eingesetzten Krypto-Verfahren werden, die unter deutscher Beteiligung entstanden sind.

10.3.3 Bewertung schlanker Verfahren

Die Forschung zu schlanken symmetrischen Verschlüsselungsverfahren befindet sich noch in einer frühen Phase. Bisher wurden noch nicht allzu viele Verfahren dieser Art vorgeschlagen, und es gibt vergleichsweise wenige Kryptoanalyse-Resultate. Nachdem es bereits mehrere Wettbewerbe zur Ermittlung guter kryptografischer Verfahren gegeben hat (AES-Wettbewerb, eSTREAM, SHA-3-Wettbewerb und andere, siehe Abschnitt 18.5), wäre es nun an der Zeit, einen Wettbewerb zum Thema schlanke Verschlüsselungsverfahren durchzuführen. Ich hoffe, dass dies in den nächsten Jahren in irgendeiner Form passieren wird.

Eine interessante Frage lautet, wie man schlanke Blockchiffren im Vergleich zu Stromchiffren (siehe Kapitel 16) einordnen soll. Bisher werden in ressourcenschwachen Umgebungen meist Letztere eingesetzt. Die Entwicklung von Verfahren wie PRESENT oder SEA zeigt jedoch, dass auch Blockchiffren für diesen Zweck geeignet sind. Da Stromchiffren meist eine Initialisierungsphase benötigen, haben diese vor allem bei der Verschlüsselung kleinerer Datenmengen einen Nachteil. Da Blockchiffren zudem besser untersucht sind und insgesamt häufiger verwendet werden, könnte es durchaus sein, dass Stromchiffren in den nächsten Jahren an Bedeutung verlieren werden.

Bleibt noch die Frage, welches schlanke Verschlüsselungsverfahren derzeit das empfehlenswerteste ist. Durch die ISO-Standardisierung spricht derzeit vieles für PRESENT. Die bisherigen Untersuchungen lassen vermuten, dass PRESENT sicher ist – sollte sich diese Einschätzung als Irrtum herausstellen, dann kann man als Verantwortlicher immer noch die Schuld auf die ISO schieben. Auch der geringe Hardwareverbrauch ist ein Argument für PRESENT. Die Entwickler des Verfahrens (diese sind natürlich keine neutrale Quelle) geben folgende Zahlen an [BKLPPR]:

- PRESENT: 1.570 Gatter
- SEA (mit Parameter, die mit PRESENT vergleichbar sind): 2.280 Gatter
- Stromchiffren aus dem eSTREAM-Projekt (siehe Abschnitt 16.6): 1.300–2.600 Gatter
- TEA: 2.100 Gatter
- DES: 3.000 Gatter
- AES: 3.600 Gatter (nach neueren Überlegungen wären hier eher 3.000 Gatter angebracht)

Auf jeden Fall lohnt es sich, die Augen offen zu halten. Es wird zweifellos in den nächsten Jahren weitere interessante schlanke Verschlüsselungsverfahren geben.

10.4 Tweak-Verfahren

Im Jahr 2002 schlugen die drei Kryptografen Liskov, Rivest und Wagner eine Variante von Verschlüsselungsverfahren vor, die sie »tweakable« (wörtlich: »zwickbar«) nannten [LiRiWa]. Die Idee war zwar nicht ganz neu, die drei waren jedoch die Ersten, die eine Theorie dazu entwickelten und einen Namen dafür vorschlugen. Da »zwickbar« auf Deutsch nicht besonders gut klingt, werde ich in diesem Buch von **Tweak-Verfahren** reden.

Die Idee hinter einem Tweak-Verfahren ist recht einfach: Während herkömmliche Verschlüsselungsverfahren nur den Schlüssel und den Klartext als Eingabe kennen, kommt bei einem Tweak-Verfahren noch ein dritter Eingabewert (**Tweak**) hinzu. Der Tweak muss nicht geheim gehalten werden. Sein Zweck ist es, dass der gleiche Klartext bei gleichem Schlüssel in einen unterschiedlichen Geheimtext verschlüsselt wird. Oder anders ausgedrückt: Jeder Tweak definiert ein eigenständiges Verschlüsselungsverfahren. Ein Tweak-Verfahren sollte folgende Anforderungen erfüllen:

- Das Verfahren muss inklusive des Tweaks ein Zufallsorakel sein. Das bedeutet insbesondere: Wenn Mallory Klartext, Geheimtext und Tweak kennt, darf ihm dies keine Rückschlüsse auf einen Geheimtext bei Verwendung eines anderen Tweaks ermöglichen. Nur so ist gewährleistet, dass jeder Tweak ein eigenständiges Verschlüsselungsverfahren definiert.
- Die Performanz des Verfahrens darf durch die zusätzliche Verarbeitung des Tweaks nicht wesentlich schlechter werden.
- Das Ändern eines Tweaks darf nur geringen Aufwand erfordern. Insbesondere muss dieser Vorgang einfacher sein als die Schlüsselaufbereitung des Verfahrens.
- Die Schlüssellänge und die Blocklänge dürfen durch den Tweak nicht verändert werden.

Tweak-Verfahren spielen vor allem im Zusammenhang mit Betriebsarten (siehe Abschnitt 19.2) seine wichtige Rolle. Außerdem lassen sie sich als kryptografische Hashfunktion (siehe Abschnitt 14.4.5) nutzen.

10.4.1 Beispiele

Alice und Bob können versuchen, ein bestehendes Verfahren (etwa den AES) in ein Tweak-Verfahren umzuwandeln. Dies ist jedoch deutlich schwieriger, als es scheint. Ein naheliegender Versuch besteht darin, den Tweak per Exklusiv-oder-

Verknüpfung mit dem Klartext zu verknüpfen (m ist der Klartext, k der Schlüssel, t der Tweak):

$$e_{k,t}(m) = e_k(m \oplus t)$$

Allerdings ist die Zufallsorakel-Eigenschaft hier nicht erfüllt, denn wenn im Klartext und im Tweak jeweils das gleiche Bit kippt, dann ändert sich der Geheimtext nicht. Ähnliche Probleme ergeben sich, wenn Alice und Bob den Geheimtext mit dem Tweak verknüpfen ($e_{k,t}(m) = e_k(m) \oplus t$) oder gleich zwei Verknüpfungen einbauen ($e_{k,t}(m) = e_k(m \oplus t) \oplus t$) – in beiden Fällen ist kein Zufallsorakel gegeben. Wenn die beiden den Tweak mit dem Schlüssel exklusiv-oder-verknüpfen ($e_{k,t}(m) = e_{k \oplus t}(m)$), sind Related-Key-Angriffe möglich. Sicher ist dagegen folgende Variante [LiRiWa]:

$$e_{k,t}(m) = e_k(e_k(m) \oplus t)$$

Diese Methode sieht jedoch pro verschlüsseltem Block zwei Verschlüsselungsfunktionen vor, wodurch das Verschlüsseln deutlich langsamer wird. Gleiches gilt für das in [P1619] standardisierte Tweak-Verfahren.

10.4.2 Threefish

Befriedigende Ergebnisse erhalten Alice und Bob nur, wenn sie ein Tweak-Verfahren nutzen, das speziell für diesen Zweck geschaffen und nicht etwa aus einem bestehenden Algorithmus abgeleitet wurde. Das derzeit bekannteste Verfahren dieser Art ist interessanterweise eines, das nicht in erster Linie zum Verschlüsseln, sondern für die Nutzung innerhalb der kryptografischen Hashfunktion Skein (siehe Abschnitt 14.4.5) entwickelt wurde. Es nennt sich **Threefish** und wurde von einem internationalen Kryptografen-Team geschaffen, dem unter anderem der Deutsche Stefan Lucks und der in diesem Buch öfter erwähnte Bruce Schneier angehörten. Der Name des Verfahrens ist an Twofish (siehe Abschnitt 9.2) angelehnt, auch wenn letzterer Algorithmus völlig anders funktioniert.

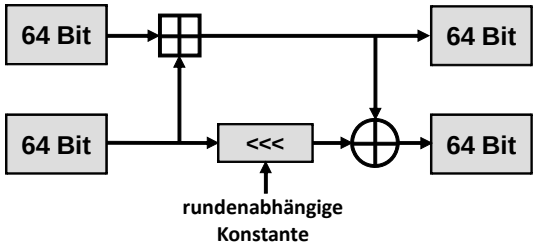


Abb. 10–6 *Die Funktion MIX ist ein Bestandteil von Threefish.*

Threefish unterstützt drei unterschiedliche Schlüssellängen: 256, 512 und 1.024 Bit. Die Blocklänge entspricht jeweils der Schlüssellänge. Der Tweak hat eine

Länge von 128 Bit. Der wichtigste Bestandteil von Threefish ist eine Funktion, die sich MIX nennt und die zwei 64-Bit-Blöcke bearbeitet (siehe Abbildung 10–6). MIX sieht eine Addition modulo 2^{64}, eine Linksrotation sowie eine Exklusiv-oder-Verknüpfung vor. Die Modulo-Addition ist der einzige nichtlineare Bestandteil des Verfahrens, sie ersetzt also eine S-Box (man beachte, dass die Modulo-Addition nicht linear ist). Um wie viele Einheiten bei der Linksrotation von MIX rotiert wird, ändert sich von Aufruf zu Aufruf. Aus einer Tabelle, welche die Spezifikation vorgibt, kann man den jeweiligen Wert in Abhängigkeit von der Runde, der Schlüssellänge und des Parallelaufrufs auslesen.

Bei einer Schlüssellänge von 1.024 Bit arbeitet Threefish in 80 Runden, ansonsten in 72. Eine Runde sieht jeweils mehrere parallele Aufrufe von MIX und eine anschließende Permutation vor (siehe Abbildung 10–7). Bei einer Schlüssellänge von 256 sind es zwei parallele Aufrufe, bei 512 bzw. 1.024 entsprechend doppelt oder viermal so viele. Die Permutation ändert die Reihenfolge der eingegebenen 64-Bit-Wörter, wobei die Threefish-Spezifikation wiederum eine Tabelle vorgibt, in der der Ablauf der jeweils durchzuführenden Permutation in Abhängigkeit von der Runde und der Blockgröße festgeschrieben ist.

Am Anfang sowie nach jeweils vier Runden wird per Modulo-Addition ein Subschlüssel eingebracht. Da somit sowohl am Anfang als auch am Ende ein Subschlüssel addiert wird, ist eine Form von Whitening gegeben. Die Schlüsselaufbereitung muss dementsprechend 19 bzw. 21 Subschlüssel generieren. Auf die Details will ich an dieser Stelle nicht eingehen. Eines ist jedoch wichtig: Die Schlüsselaufbereitung verarbeitet zunächst den Schlüssel und addiert anschließend den Tweak dazu (es wird modulo 2^{64} gerechnet). Dadurch ist es möglich, den Tweak zu ändern, ohne eine neue Schlüsselaufbereitung durchführen zu müssen.

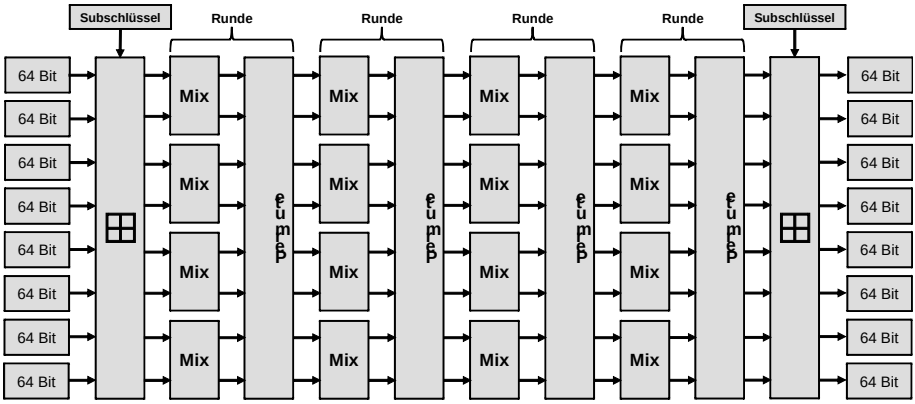


Abb. 10–7 *Threefish arbeitet in 72 oder 80 Runden. Vier davon sind hier zu sehen.*

10.4.3 Bewertung von Tweak-Verfahren

Obwohl Tweak-Verfahren seit über zehn Jahren bekannt sind, gibt es bisher nur wenige Forschungsarbeiten zu diesem Thema. Insbesondere wurden nur wenige Verfahren dieser Art entwickelt. Threefish ist eines der wenigen davon. Es gilt als sicher, wurde aber nicht in erster Linie zum Verschlüsseln, sondern für die Nutzung in Form einer kryptografischen Hashfunktion geschaffen. Es gibt also noch einigen Forschungsbedarf in diesem Bereich.

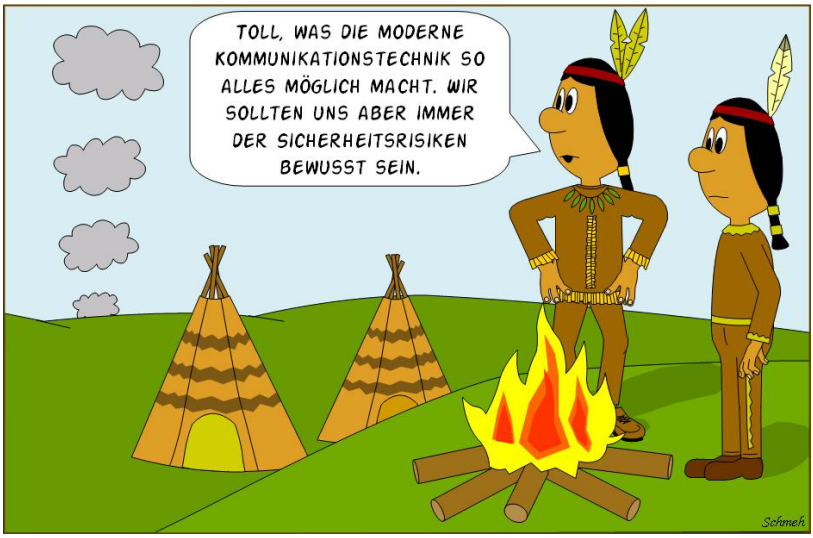
10.5 Weitere symmetrische Verschlüsselungsverfahren

Die in diesem Buch vorgestellte Übersicht über symmetrische Verschlüsselungsverfahren ist bei weitem nicht vollständig. Nicht behandelt habe ich beispielsweise **Chiasmus**, ein symmetrisches Verfahren, das vom Bundesamt für Sicherheit in der Informationstechnik (BSI) entwickelt wurde. Seine Funktionsweise ist geheim. Bekannt ist lediglich, dass es sich dabei um ein speziell für Softwareimplementierungen entwickeltes Verfahren handelt. Chiasmus verschlüsselt 64-Bit-Blöcke mit einem 160-Bit-Schlüssel. Die effektive Schlüssellänge beträgt jedoch lediglich 128 Bit, die restlichen 32 Bit bilden eine Prüfsumme. Chiasmus wurde vom BSI vor allem für deutsche Behörden entwickelt. Die bekannteste Anwendung von Chiasmus ist derzeit die Software *Chiasmus für Windows*, mit der Dateien und Verzeichnisse verschlüsselt werden können.

Ein weiteres geheimes symmetrisches Verschlüsselungsverfahren des BSI ist **Libelle**. Dieses wird von dem Hardwarechip Pluto eingesetzt, der bei zahlreichen Behörden in Verwendung ist. Auch Libelle unterstützt eine Schlüssellänge von 160 Bit. Vermutlich haben Chiasmus und Libelle eine ähnliche Funktionsweise, wobei ersteres Verfahren für Software- und letzteres für Hardwareimplementierungen optimiert ist.

Weitere symmetrische Verschlüsselungsverfahren, die dem Kryptografie-Interessierten ab und zu über den Weg laufen, sind **Anubis**, **ICE** (Information Concealment Engine), **KHAZAD** und **SHACAL**. In Japan haben **CIPHERUNICORN-E**, **CIPHERUNICORN-A**, **Hierocrypt-L1**, **Hierocrypt-3** und **SC2000** eine gewisse Popularität. Nicht zu vergessen ist der **Cellular Message Encryption Algorithm** (**CMEA**), der in den USA zur Verschlüsselung im Mobilfunk eingesetzt wird. Das Verfahren **Red Pike** wurde vom britischen Geheimdienst GCHQ entwickelt, die Funktionsweise ist geheim. **SEED** ist ein in Südkorea sehr populäres Verfahren, das von der dortigen Behörde für Informationssicherheit stammt. Zudem existieren einige Algorithmen, die von der NSA entwickelt wurden. Die bekanntesten davon heißen **JUNIPER** und **BATON**. Daneben gibt es eine ganze Klasse weiterer symmetrischer Verschlüsselungsverfahren, die man als Stromchiffren bezeichnet (die in diesem Kapitel beschriebenen Verfahren sind Blockchiffren). Um Stromchiffren geht es in Abschnitt 16.

11 Asymmetrische Verschlüsselung



Betrachten Sie folgende E-Mail, die Bob an Alice schreibt:

```
Liebe Alice,
ich schlage vor, dass wir von nun an alle unsere Mails mit dem AES verschlüsseln.
Als Schlüssel verwenden wir D6 FE 76 C9 8A 76 AF 6F E6 7F D9 EF A3 68 BB 5F.
Bob
```

Über eine solche E-Mail freut sich Mallory ganz besonders. Da er den AES kennt und aus Alices E-Mail den verwendeten Schlüssel erfährt, hat er keine Probleme, fortan die verschlüsselten Nachrichten von Alice und Bob zu lesen. Damit sind wir bei einem fundamentalen Problem der Kryptografie angelangt: Es hat keinen Sinn, wenn Alice und Bob sich über denjenigen Übertragungskanal auf einen gemeinsamen Schlüssel verständigen, über den sie anschließend auch die verschlüsselten Nachrichten senden. Wenn nämlich Mallory den Übertragungskanal abhört, dann gelangt er dadurch an den Schlüssel. Wird der Übertragungskanal dagegen nicht abgehört, dann können Alice und Bob gleich auf die Verschlüsse-

lung verzichten. Um diesem Dilemma – es wird **Schlüsselaustauschproblem** genannt – zu entgehen, gibt es folgende Möglichkeiten:

- Alice übergibt Bob den Schlüssel bei einem persönlichen Treffen oder per Telefon (dies wird auch als **manueller Schlüsselaustausch** oder **Out-of-Band-Schlüsselaustausch** bezeichnet). Wollte Alice jedoch jeden, dem sie eine verschlüsselte Mail schicken will, vorher treffen, dann hätte sie einiges zu tun. Der manuelle Schlüsselaustausch hat zudem den Nachteil, dass es recht unübersichtlich wird, wenn eine größere Anwendergruppe untereinander verschlüsselt kommunizieren will. Soll beispielsweise jeder der 50 Mitarbeiter der Firma Krypt & Co. einen AES-Schlüssel mit jeweils jedem anderen Mitarbeiter gemeinsam haben, dann werden insgesamt nicht weniger als 1.225 Schlüssel benötigt. Allgemein sind bei n Mitarbeitern $n/2 \cdot (n-1)$ Schlüssel erforderlich (siehe Abbildung 11–1).
- Alice und Bob verwenden einen Authentifizierungsserver. Ein Beispiel dafür ist Kerberos (Abschnitt 22.3).
- Alice und Bob verwenden **asymmetrische Kryptografie**. Genau darum geht es in diesem Kapitel.

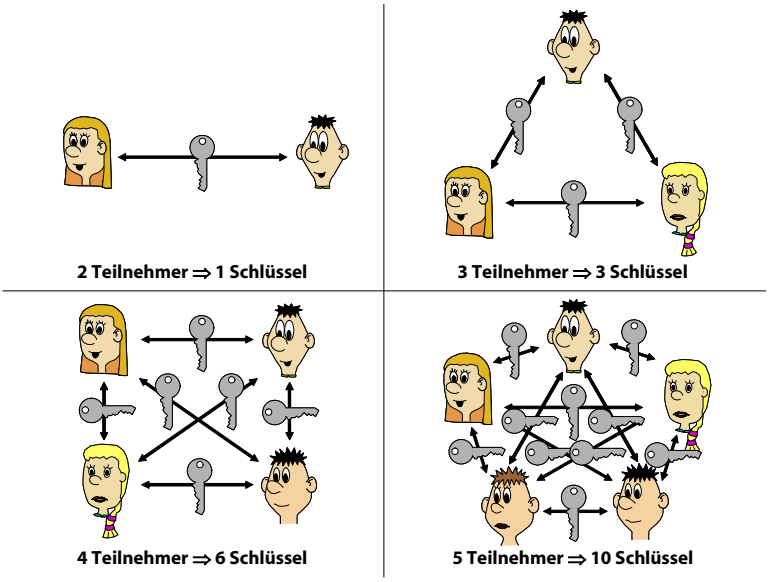


Abb. 11–1 *Um ein symmetrisches Verschlüsselungsverfahren einzusetzen, müssen je zwei Teilnehmer einen gemeinsamen geheimen Schlüssel besitzen. Dies ist nur bei kleineren Teilnehmerzahlen sinnvoll, da sonst die Anzahl der Schlüssel zu groß wird.*

Wie funktioniert nun die asymmetrische Kryptografie (sie wird auch **Public-Key-Kryptografie** genannt)? Die Verschlüsselungsverfahren, die wir bisher besprochen haben (zum Beispiel DES und AES), bezeichnet man, wie erwähnt, als symmetri-

sche Verfahren (oder Secret-Key-Verfahren). Symmetrisch deshalb, weil Alice und Bob denselben Schlüssel verwenden. Die Verfahren der asymmetrischen Kryptografie (die Public-Key-Verfahren) verwenden dagegen stets zwei Schlüssel pro verschlüsselter Nachricht (siehe Abbildung 11–2): einen, den nur Empfängerin Alice kennt (den **privaten Schlüssel**), und einen, der öffentlich bekannt ist (den **öffentlichen Schlüssel**). Die beiden Schlüssel sind voneinander abhängig. In der Regel sind sie zudem einer Person zugeordnet, weshalb wir auch von Alices öffentlichem Schlüssel und Alices privatem Schlüssel sprechen können.

Will Bob eine verschlüsselte Nachricht an Alice schicken, dann benötigt er deren öffentlichen Schlüssel. Diesen kann er sich guten Gewissens von Alice über das Netz schicken lassen, denn Abhörer Mallory hat nichts gewonnen, wenn er ihn erfährt – schließlich ist es ja ein öffentlicher Schlüssel und damit kein Geheimnis. Mit Alices öffentlichem Schlüssel verschlüsselt Bob nun die Nachricht und schickt sie an Alice. Alice verwendet anschließend ihren privaten Schlüssel, um die Nachricht wieder zu entschlüsseln.

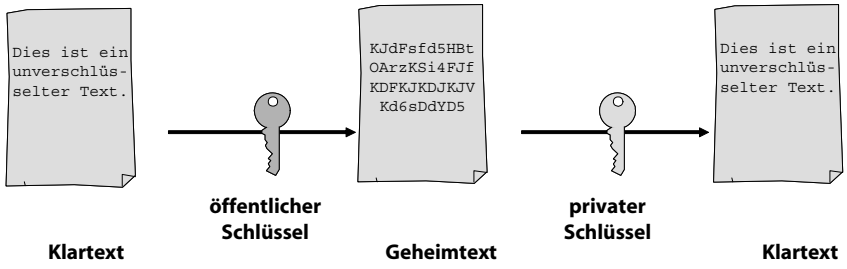


Abb. 11–2 *So funktioniert eine Public-Key-Verschlüsselung: Zum Verschlüsseln wird ein anderer Schlüssel verwendet als zum Entschlüsseln. Ersterer ist öffentlich bekannt. Letzterer wird vom Empfänger geheim gehalten, weshalb man ihn privaten Schlüssel nennt.*

Der öffentliche und der private Schlüssel hängen zwar voneinander ab, man kann aus dem öffentlichen jedoch nicht den privaten bestimmen. In der Schreibweise der Mathematik sieht die Public-Key-Verschlüsselung so aus: Alice hat einen öffentlichen Schlüssel a und einen privaten Schlüssel x. Ist m die Nachricht, c der Geheimtext, e die Verschlüsselungsfunktion und d die Entschlüsselungsfunktion, dann gilt:

$c = e(a,m)$ und $m = d(x,c)$.

Wenn anschließend Alice etwas an Bob zurückschicken will, dann benötigt sie Bobs öffentlichen Schlüssel, um die Nachricht zu verschlüsseln. Die Verwirrung, die an dieser Stelle oft entsteht, ist auch deshalb verständlich, weil es alles andere als einfach ist, Funktionen e und d zu finden, die die geforderten Eigenschaften haben. Es gibt sie jedoch, und um sie zu betrachten, benötigen wir zunächst etwas Hilfe aus der Mathematik.

11.1 Ein bisschen Mathematik

Ich wünschte, es gäbe ein triviales Beispiel für ein asymmetrisches Verschlüsselungsverfahren. Doch leider ist dies nicht der Fall. Es gibt allgemein nur relativ wenige asymmetrische Verfahren und noch weniger, die wirklich sicher sind. Alle haben sie gemeinsam, dass man mathematische Vorkenntnisse benötigt, um sie zu verstehen. Einige dieser Vorkenntnisse will ich Ihnen in diesem Kapitel vermitteln. Über meine Betrachtungen wird ein Mathematiker vielleicht nur milde lächeln. Mir geht es jedoch nicht darum, streng mathematisch vorzugehen. Vielmehr will ich die wichtigsten mathematischen Zusammenhänge auf möglichst einfache Weise erklären und verzichte daher weitgehend auf Sätze und Beweise.

11.1.1 Modulo-Rechnen

Beginnen wir unsere Betrachtungen mit dem sogenannten **Modulo-Rechnen**. Modulo-Rechnen ist das Rechnen mit natürlichen Zahlen von 0 bis zu einer bestimmten Größe n (wir zählen die Null hierbei zu den natürlichen Zahlen, auch wenn das nicht überall üblich ist). Zahlen, die größer oder gleich n sind, gibt es in diesem Zusammenhang nicht. Nach n-1 wird wieder bei Null angefangen zu zählen. Beispielsweise kann man die Stundenanzeige einer Digitaluhr als Modulo-24-Zähler betrachten. Eine solche Anzeige zählt die Stunden von 0 bis 23 und fängt anschließend wieder bei 0 an. Im Folgenden bezeichnet n immer eine natürliche Zahl. a und b sind ebenfalls natürliche Zahlen, die stets zwischen 0 und n-1 liegen.

Modulo-Addition und -Subtraktion

Die beiden wichtigsten Modulo-Rechenarten, die **Modulo-Addition** und die **Modulo-Subtraktion**, entsprechen dem herkömmlichen Abziehen und Zusammenzählen, allerdings mit einer kleinen Änderung: Ist bei der Addition das Ergebnis größer oder gleich n, dann zieht man n davon ab. Ist analog bei einer Subtraktion das Ergebnis kleiner null, dann zählt man n dazu. Bei einer Modulo-Addition oder -Subtraktion erhält man also in jedem Fall eine Zahl zwischen 0 und n-1. Um Modulo-Rechenarten von den bekannten zu unterscheiden, schreibt man am Ende einer Gleichung jeweils »(mod n)«.

Beispiele:

$3+5=1 \pmod 7$

$2+2=4 \pmod{13}$

$3-6=6 \pmod 9$

$3+6=0 \pmod 9$

Modulo-Multiplikation und -Division

Aus der Modulo-Addition können wir auf einfache Weise eine **Modulo-Multiplikation** zweier Zahlen herleiten. Wie gewohnt schreibt man dabei eine Multiplikation als mehrfache Addition.

Beispiel:

$$\begin{aligned} 4\cdot 5 &= 4+4+4+4+4 \pmod 7 \\ &= 1 \quad +4+4+4 \pmod 7 \\ &= 5 \qquad +4+4 \pmod 7 \\ &= 2 \qquad\quad +4 \pmod 7 \\ &= 6 \pmod 7 \end{aligned}$$

Man kann das Ergebnis einer Modulo-Multiplikation jedoch auch etwas schneller berechnen: Man multipliziert dazu einfach die beiden Zahlen und zieht dann so lange n ab, bis man schließlich eine Zahl zwischen 0 und (n-1) erhält.

Beispiel:

$4\cdot 5 = 20\text{-}7\text{-}7 = 6 \pmod 7$

Als Nächstes fragen wir uns, ob es zu jeder Zahl a eine Zahl b gibt mit der Eigenschaft $a\cdot b=1 \pmod n$. Gibt es eine solche Zahl, dann nennt man sie das **inverse Element** von a und schreibt dafür a^{-1}. Da eine Multiplikation mit dem inversen Element einer Zahl stets das Gleiche ist wie das Teilen durch diese Zahl, haben wir somit auch eine **Modulo-Division** definiert. »$b\cdot a^{-1} \pmod n$« kann man daher auch als »$b/a \pmod n$« schreiben. Wann aber existiert überhaupt ein inverses Element zu a, wenn modulo n gerechnet wird? Die Antwort lautet: wenn a und n teilerfremd sind, also wenn sie außer 1 keinen gemeinsamen Teiler haben.

Beispiele:

- $5^{-1} \pmod 7$ existiert, da 5 und 7 teilerfremd sind.
  Es gilt: $5^{-1}=3$.
- $4^{-1} \pmod 8$ existiert nicht, da 4 und 8 nicht teilerfremd sind
  (4 teilt beide Zahlen).
- $7^{-1} \pmod{10}$ existiert, da 10 und 7 teilerfremd sind.
  Es gilt: $7^{-1}=3$.

Ob zu einer Zahl a ein inverses Element (mod n) existiert, ist also einfach nachzuprüfen. Etwas schwieriger ist es da schon, dieses inverse Element auch zu berechnen. Auch dieses Problem ist jedoch in den Griff zu bekommen, und zwar mit dem sogenannten **euklidischen Algorithmus** (den ich aus Platzgründen nicht beschreiben will). Dieser wird im Normalfall verwendet, um den größten gemeinsamen Teiler zweier Zahlen a und n zu finden. Man kann den Algorithmus jedoch so modifizieren, dass er $a^{-1} \pmod n$ berechnet, falls diese Zahl existiert. Für

Experten sei noch Folgendes erwähnt: Die Komplexität des euklidischen Algorithmus beträgt O(log(n)). Er ist also etwa so schnell wie das Auffinden eines Elements in einem sortierten Feld.

Modulo-Exponentiation und deren Umkehrungen

Wie Sie sicherlich wissen, kann man mithilfe einer Multiplikation eine Exponentiation (Potenzfunktion) definieren, indem man mehrere Multiplikationen zusammenfasst. Genau das funktioniert auch beim Modulo-Rechnen.

Beispiel:

$$\begin{aligned} 3^4 &= 3 \cdot 3 \cdot 3 \cdot 3 \pmod 7 \\ &= 2 \cdot 3 \cdot 3 \pmod 7 \\ &= 6 \cdot 3 \pmod 7 \\ &= 4 \pmod 7 \end{aligned}$$

Damit haben wir also auch eine **Modulo-Exponentiation** definiert. In der Kryptografie spielen vor allem die beiden Umkehrungen dieser Rechenart eine Rolle. Eine davon ist der **Modulo-Logarithmus**, der auch als **diskreter Logarithmus** bezeichnet wird. Sind a, b und n gegeben, dann ist der diskrete Logarithmus die Zahl x, für die gilt: $a^x=b \pmod n$. Die Frage, wann zu gegebenen Zahlen a, b und n der diskrete Logarithmus existiert, wird uns weiter unten begegnen. Die zweite Umkehrung der Modulo-Exponentiation ist das **Modulo-Wurzelziehen**. Dessen Definition dürfte nun klar sein: Zu gegebenem a, b und n wird eine Zahl x gesucht, für die gilt: $x^a=b \pmod n$. x heißt dann die a-te Wurzel von $b \pmod n$. Um die Frage zu beantworten, wann eine Modulo-Wurzel existiert, benötigen wir die sogenannte φ-Funktion, die angibt, wie viele natürliche Zahlen, die größer als 0 und kleiner als eine Zahl n sind, zu n teilerfremd sind.

Beispiele:

φ(3)=2, da 1 und 2 teilerfremd zu 3 sind.

φ(6)=2, da 1 und 5 teilerfremd zu 6 sind.

φ(7)=6, da 1, 2, 3, 4, 5 und 6 teilerfremd zu 7 sind.

Wie Sie vielleicht bemerkt haben, ist $\varphi(n)=n-1$, wenn n eine Primzahl ist. Ist dagegen n das Produkt zweier Primzahlen p und q, dann gilt $\varphi(n)=(p-1)\cdot(q-1)$. Bei der Berechnung von Modulo-Wurzeln hilft uns nun der folgende Satz:

Satz: Sind a und n natürliche Zahlen ($a<n$),
dann gilt: $a^{1 \pmod{\varphi(n)}} = a \pmod n$.

Mit anderen Worten heißt dies: $a^b=a \pmod n$ gilt, wenn $b-1$ ein Vielfaches von $\varphi(n)$ ist.

Beispiel:

$3^3=3 \pmod 6$, da $3=1 \pmod{(\varphi(6))}$

Mit diesem Satz können wir nun die *a*-te Modulo-Wurzel einer Zahl berechnen. Gesucht ist zu gegebenem *a*, *b* und *n* eine Zahl *x*, für die gilt:

$x^a=b \pmod n$.

Wenn *a* und $\varphi(n)$ teilerfremd sind, dann kann man mit dem euklidischen Algorithmus die Zahl $c=a^{-1} \pmod{\varphi(n)}$ berechnen. Mit dieser Zahl exponieren wir jetzt beide Seiten der Gleichung:

$(x^a)^{c(\mathrm{mod}\ \varphi(n))} = b^{\,c(\mathrm{mod}\ \varphi(n))} \pmod n$.

Durch die Anwendung des obigen Satzes können wir die rechte Seite vereinfachen:

$(x^a)^{c(\mathrm{mod}\ \varphi(n))} = b^c \pmod n$.

Da $a \cdot c = 1 \pmod{\varphi(n)}$, erhalten wir auf der linken Seite:

$x^{1(\mathrm{mod}\ \varphi(n))} = b^c \pmod n$.

Dies ist nach obigem Satz wiederum:

$x=b^c \pmod n$

Nun kennen wir die Zahl *x*, die ja die gesuchte *a*-te Wurzel von *b* ist. Damit *x* existiert, müssen *a* und $\varphi(n)$ teilerfremd sein. Ist dies der Fall und kennt man $\varphi(n)$, so ist die Wurzel problemlos zu berechnen. Dass $\varphi(n)$ jedoch oftmals alles andere als einfach zu berechnen ist, werden Sie im nächsten Kapitel sehen.

Gruppen und Körper

In der Mathematik und in der Kryptografie sind solche Werte von *n* besonders interessant, für die alle Zahlen zwischen 1 und *n*-1 ein inverses Element (mod *n*) haben. Damit dies der Fall ist, müssen alle Zahlen zwischen 1 und *n*-1 teilerfremd zu *n* sein. Diese Anforderung ist genau dann erfüllt, wenn *n* eine Primzahl ist (ich schreibe in einem solchen Fall *p* statt *n*). Ist *p* also eine Primzahl, dann ist jede Zahl zwischen 0 und *p*-1 durch jede Zahl zwischen 1 und *p*-1 modulo *p* teilbar (die Division durch 0 ist wie üblich nicht definiert).

Die Zahlen zwischen 1 und *p*-1 bilden mit der Modulo-Multiplikation eine sogenannte **Gruppe**, die wir als $\mathbf{Z(p,\cdot)}$ bezeichnen. Eine Gruppe ist in der Mathematik eine Menge, für die folgende Voraussetzungen gegeben sind:

- Es ist eine Verknüpfung definiert (in diesem Fall ist dies die Modulo-Multiplikation). Dass eine Verknüpfung definiert ist, bedeutet, dass man zwei beliebige Elemente der Gruppe miteinander verknüpfen kann und dabei stets ein

weiteres Element der Gruppe erhält (Abgeschlossenheit). Die Verknüpfung muss zudem assoziativ sein, was bei der Modulo-Multiplikation gegeben ist.
- Es gibt ein neutrales Element (in diesem Fall die Zahl 1). Ein neutrales Element hat die Eigenschaft, dass man bei der Verknüpfung mit einem anderen Element dieses andere Element als Ergebnis erhält.
- Zu jedem Element gibt es ein inverses Element. Da p eine Primzahl ist, ist diese Voraussetzung in unserem Fall gegeben.

Beachten Sie, dass p auch wirklich eine Primzahl sein muss, ansonsten haben nicht alle Elemente ein inverses Element, und es handelt sich dann nicht um eine Gruppe. Außerdem gehört die Null nicht zur Gruppe $Z(p,\cdot)$, da die Null kein inverses Element besitzt. Übrigens ist die Menge der Zahlen zwischen 0 und p-1 auch bezüglich der Modulo-Addition eine Gruppe – wir nennen sie $\mathbf{Z}(\boldsymbol{p},+)$. Dieses Mal ist die Null Bestandteil der Gruppe (sie ist das neutrale Element).

Die Menge der Zahlen zwischen 0 und p-1 bilden damit sowohl mit der Modulo-Addition als auch mit der Modulo-Multiplikation (ohne die Null) eine Gruppe. Eine solche »Doppelgruppe« wird in der Mathematik **Körper** genannt. Einen Körper, der auf die beschriebene Weise aus den Zahlen 0 bis p-1 entsteht, nennt man Körper der Größe p oder Galois-Feld der Größe p. Wir schreiben dafür $\mathbf{GF}(\boldsymbol{p})$.

Untergruppen

Eine **Untergruppe** ist eine Teilmenge einer Gruppe, die (mit derselben Verknüpfung und demselben neutralen Element) selbst wieder eine Gruppe bildet. Nicht jede Teilmenge einer Gruppe ist automatisch eine Untergruppe. Eine Minimalanforderung ist, dass das neutrale Element in der Teilmenge enthalten ist. Darüber hinaus fehlt bei vielen Teilmengen die Abgeschlossenheit – dies bedeutet, dass bei der Verknüpfung zweier Elemente der Teilmenge das Ergebnis nicht zur Teilmenge gehört.

Beispiel: Betrachten wir $Z(5,\cdot)$, dann gilt:
- {2, 3} ist keine Untergruppe, da das neutrale Element fehlt.
- {1, 3} ist keine Untergruppe, da gilt: $3\cdot3=4 \pmod 5$. 4 ist jedoch kein Element der Teilmenge.
- {1, 4} ist eine Untergruppe, da gilt: $1\cdot1=1 \pmod 5$, $1\cdot4=4 \pmod 5$, $4\cdot1=4 \pmod 5$ und $4\cdot4=1 \pmod 5$. Abgeschlossenheit ist also gegeben, da bei allen möglichen Verknüpfungen stets ein Element der Teilmenge entsteht.

Man kann beweisen, dass die Anzahl der Elemente in einer Untergruppe ein Teiler der Anzahl der Elemente der zugehörigen Gruppe ist. Da die Gruppe $Z(p,\cdot)$ genau p-1 Elemente besitzt, bedeutet dies, dass die Anzahl der Elemente in einer Untergruppe stets p-1 teilt. Hier ein Beispiel: Die Gruppe $Z(13,\cdot)$ hat 12 Elemente. Die Untergruppen von $Z(13,\cdot)$ haben daher 1, 2, 3, 4, 6 oder 12 Elemente.

Man kann außerdem zeigen, dass es zu $Z(p,\cdot)$ für jeden Teiler von p-1 genau eine Untergruppe mit dieser Anzahl von Elementen gibt. Hier ein Beispiel: Zu $Z(13,\cdot)$ gibt es jeweils eine Untergruppe mit 1, 2, 3, 4, 6 und 12 Elementen. Die Untergruppe mit 12 Elementen ist die Gruppe selbst. Die Untergruppe mit einem Element enthält nur die Zahl 1.

Generatoren

Eine weitere Aussage, die sich beweisen lässt: Nimmt man ein beliebiges Element a aus $Z(p,\cdot)$ und betrachtet die Menge $\{a, a^2, a^3, a^4, ..., a^{p-1} \pmod p\}$, dann erhält man eine Untergruppe. a heißt **Generator** der Untergruppe. Jede Untergruppe von $Z(p,\cdot)$ hat mindestens einen Generator, und damit auch $Z(p,\cdot)$ selbst. Die folgende Tabelle zeigt die Untergruppen von $Z(13,\cdot)$:

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}
1											
2	4	8	3	6	12	11	9	5	10	7	1
3	9	1									
4	3	12	9	10	1						
5	12	8	1								
6	10	8	9	2	12	7	3	5	4	11	1
7	10	5	9	11	12	6	3	8	4	2	1
8	12	5	1								
9	3	1									
10	9	12	3	4	1						
11	4	5	3	7	12	2	9	8	10	6	1
12	1										

Aus der Tabelle lässt sich beispielsweise herauslesen, dass die Zahl 1 der einzige Generator der Untergruppe von $Z(13,\cdot)$ mit einem Element ist. Die Untergruppe mit zwei Elementen hat den Generator 12. Für die Untergruppe mit drei Elementen gibt es zwei Generatoren: 3 und 9. Die Zahlen 5 und 8 generieren die Untergruppe mit vier Elementen. Die Untergruppe mit sechs Elementen wird von 4 und 10 generiert. 2, 6, 7 und 11 sind Generatoren der Untergruppe mit 12 Elementen, also $Z(13,\cdot)$ selbst.

Dass es auch Generatoren für $Z(p,\cdot)$ selbst gibt, spielt in der Kryptografie eine wichtige Rolle. Wie Sie leicht nachvollziehen können, ist die Gleichung $a^x=b \pmod p$ immer lösbar, wenn a ein Generator von $Z(p,\cdot)$ ist (b ist dabei ein Element von $Z(p,\cdot)$ und darf daher nicht Null sein). Mit anderen Worten heißt dies: Der diskrete Logarithmus in $Z(p,\cdot)$ existiert immer, wenn die Basis ein Generator von $Z(p,\cdot)$ ist.

11.1.2 Einwegfunktionen und Falltürfunktionen

Der Grund, warum die im letzten Kapitel beschriebene Modulo-Rechnung in der Kryptografie so wichtig ist, ist folgender: Einige der Modulo-Rechenarten sind sehr einfach durchführbar, ihre Umkehrung ist dagegen recht aufwendig. Dies lässt sich in der Kryptografie hervorragend nutzen, wenn man eine einfache Rechnung als Verschlüsselung und die komplizierte Umkehrung als Entschlüsselung verwendet. Voraussetzung ist jedoch, dass es bei der komplizierten Umkehrung eine »versteckte Abkürzung« gibt, die als Schlüssel verwendet wird. Eine Funktion, die einfach zu berechnen ist, deren Umkehrung jedoch nur mit großem Aufwand berechnet werden kann, nennt man **Einwegfunktion**. Existiert dagegen eine »versteckte Abkürzung« also eine Zusatzinformation, mit der die ansonsten schwierige Umkehrung einfach gemacht wird, dann spricht man von einer **Falltürfunktion**.

Der diskrete Logarithmus

Mit einem geeigneten Computerprogramm ist das Ergebnis der Formel $a^b \pmod n$ selbst für sehr große Werte von a, b und n relativ leicht berechenbar. Selbst für über hundert Bit lange Zahlen ist bei geschickter Implementierung eine Sekunde auf einem PC ausreichend. Vollkommen anders sieht es mit der Umkehrung, dem diskreten Logarithmus, aus. Selbst mit größtem Hardwareaufwand und den besten bekannten Algorithmen erreicht man bei mehreren Tausend Bit schnell Berechnungszeiten, die über die Lebensdauer unseres Universums hinausgehen.

Damit haben wir das erste Beispiel für eine Einwegfunktion: $f(x)=a^x \pmod n$ ist nach heutigem Kenntnisstand eine solche. Einen mathematischen Beweis dafür gibt es allerdings nicht. Es ist daher theoretisch möglich, dass eines Tages ein schnelles Verfahren zur Berechnung des diskreten Logarithmus entdeckt wird. Da dies jedoch schon seit längerem erfolglos versucht wird, können wir davon ausgehen, dass dieser Fall nie eintreten wird.

Leider ist die Erkenntnis, dass die Modulo-Exponentiation eine Einwegfunktion ist, zunächst einmal recht nutzlos. Denn Alice kann einen Klartext durch eine Modulo-Exponentiation (wir stellen uns den Klartext in diesem Kapitel als Zahl vor) zwar gut verschlüsseln – Empfänger Bob kann den Text jedoch nicht mehr entschlüsseln, da es sich ja um eine Einwegfunktion handelt. Trotzdem gilt die Modulo-Exponentiation als wichtigste Einwegfunktion der Kryptografie. Was Alice und Bob damit machen können, werden Sie in späteren Kapiteln sehen.

Das Faktorisierungsproblem

Eine weitere wichtige Einwegfunktion ist das Multiplizieren zweier Primzahlen (in diesem Fall ist zur Abwechslung nicht die Modulo-Multiplikation gemeint). Eine solche Primzahl-Multiplikation ist heutzutage mit Computerunterstützung einfach realisierbar, auch bei größeren Zahlen entstehen so schnell keine Prob-

leme. Im Gegensatz dazu sind keine effizienten Verfahren bekannt, mit denen aus dem Produkt zweier Primzahlen die beiden Faktoren bestimmt werden können. Um dies nachvollziehen zu können, hier ein kleiner Test: Berechnen Sie im Kopf das Resultat der Multiplikation 13·17. Dann bestimmen Sie zum Vergleich die beiden Primzahlen, die miteinander multipliziert 217 ergeben (natürlich auch im Kopf). Damit können Sie sich in etwa vorstellen, warum die Primzahl-Multiplikation als Einwegfunktion gilt.

Es gibt auch tiefer gehende Untersuchungen, und diese haben ähnliche Resultate hervorgebracht wie bei der Modulo-Exponentiation: Auch wenn sich nicht beweisen lässt, dass die Primzahl-Multiplikation eine Einwegfunktion ist, so spricht doch alles dafür. Man bezeichnet das Zerlegen eines Primzahlprodukts auch als Faktorisierung und spricht in diesem Zusammenhang vom **Faktorisierungsproblem**. Dieses spielt in der Kryptografie eine wichtige Rolle (ein interessanter Artikel dazu ist [Buch96]). Einen Computer dürfte die Faktorisierung einer Zahl wie 217 zwar noch nicht in Verlegenheit bringen. Bei einigen hundert Bit jedoch geht selbst der stärkste Superrechner in die Knie. Wie sich die Einwegfunktion Primzahl-Multiplikation in eine Falltürfunktion umwandeln lässt, erfahren Sie im Abschnitt über RSA (11.3).

11.2 Der Diffie-Hellman-Schlüsselaustausch

Kehren wir nach diesem mathematischen Exkurs wieder zum Ausgangspunkt dieses Kapitels zurück: Alice und Bob wollen verschlüsselt kommunizieren, ohne dabei einen geheimen Schlüssel über das Netz schicken zu müssen (Schlüsselaustauschproblem). Einweg- und Falltürfunktionen sind in der Lage, dieses Problem zu lösen. Ein Verfahren, das den diskreten Logarithmus zur Lösung des Schlüsselaustauschproblems verwendet, wurde von den Kryptografen Whitfield Diffie und Martin Hellman erfunden und wird deshalb als **Diffie-Hellman-Schlüsselaustausch** (oder auch nur als **Diffie-Hellman-Verfahren**) bezeichnet. Mit ihrem Verfahren begründeten Diffie und Hellman die asymmetrische Kryptografie und sorgten für einen gewaltigen Schub in einer Wissenschaft, die damals noch nicht sehr lange öffentlich betrieben wurde. Kein Wunder, dass die beiden ihre 1976 veröffentlichte Forschungsarbeit »New Directions in Cryptography« nannten [DifHel].

11.2.1 Funktionsweise von Diffie-Hellman

Um einen Diffie-Hellman-Schlüsselaustausch durchzuführen, einigen sich Alice und Bob auf eine Primzahl p und auf eine natürliche Zahl g. g sollte idealerweise ein Generator der Gruppe $Z(p,\cdot)$ sein, das Verfahren funktioniert aber auch, wenn g einen anderen Wert kleiner als p annimmt. Die Zahlen p und g können Alice und Bob guten Gewissens über das Internet schicken, denn Mallory darf sie

ruhig erfahren. In der Praxis ist es ohnehin meist so, dass g und p vorgegeben sind und von vielen Anwendern verwendet werden. Außerdem wählt Alice eine natürliche Zahl $x<p$, Bob wählt eine natürliche Zahl $y<p$. Diese Zahlen behalten Alice und Bob jeweils für sich. Nun beginnt folgender Ablauf:

1. Alice berechnet die Zahl $a=g^x \pmod p$. Sie schickt a an Bob.
2. Bob berechnet die Zahl $b=g^y \pmod p$. Er schickt b an Alice.
3. Alice berechnet aus dem erhaltenen b die Zahl $k_1:=b^x \pmod p$.
4. Bob berechnet aus dem erhaltenen a die Zahl $k_2:=a^y \pmod p$.

Jetzt gilt $k_1 = k_2$ (warum, das können Sie selbst nachrechnen), weshalb wir dafür k schreiben. Mallory, der die Kommunikation abhört, kennt dieses k nicht, er kann es auch nicht berechnen. Dazu müsste er nämlich den diskreten Logarithmus lösen, und das ist nach heutigem Kenntnisstand nicht möglich, wenn die Zahlen groß genug sind. Alice und Bob können also k (z.B. in eine 128-Bit-Zahl umgewandelt) gefahrlos für eine symmetrische Verschlüsselung nutzen.

Und warum ist das Ganze ein Public-Key-Verfahren? Weil a der öffentliche und x der private Schlüssel von Alice ist. Analog hat Bob den privaten Schlüssel y und den öffentlichen Schlüssel b. Man beachte, dass erwartungsgemäß in beiden Fällen der öffentliche vom privaten Schlüssel abhängt. Auch wenn das Diffie-Hellman-Verfahren selbst nicht zum Verschlüsseln verwendet werden kann, so löst es doch immerhin das Schlüsselaustauschproblem. Daher gilt es zu Recht als asymmetrisches Verfahren und als einfachster Vertreter dieser Zunft obendrein.

Ein Beispiel

Schauen wir uns einmal ein Beispiel für den Einsatz des Diffie-Hellman-Verfahrens an, bei dem wir kleine Zahlen verwenden. Alice und Bob einigen sich (wobei Mallory mithören darf) auf $g=3$ und $p=7$. Wie Sie nachrechnen können, ist 3 ein Generator von $Z(7,\cdot)$. Alice wählt als privaten Schlüssel $x=2$, Bob legt für seinen privaten Schlüssel $y=5$ fest. Nun können die beiden einen Diffie-Hellman-Schlüsselaustausch abarbeiten:

1. Alices öffentlicher Schlüssel ist $a = g^x \pmod p = 3^2 = 2 \pmod 7$.
2. Bobs öffentlicher Schlüssel ist $b = g^y \pmod p = 3^5 = 5 \pmod 7$.
3. Alice berechnet nun k_1: $k_1 = b^x = 5^2 = 4 \pmod 7$.
4. Bob kann nun k_2 berechnen: $k_2 = a^y = 2^5 = 4 \pmod 7$.

Wie gewünscht, ist $k_1 = k_2 = k$. k hat den Wert 4. Mit diesem Schlüssel können Alice und Bob nun ihre Nachrichten verschlüsseln.

Bewertung von Diffie-Hellman

Diffie-Hellman ist nur eines von mehreren Verfahren, das den diskreten Logarithmus als Einwegfunktion nutzt. Man spricht in diesem Zusammenhang auch von **Krypto-Systemen auf Basis des diskreten Logarithmus** oder schlicht von **DL-Verfahren**. Im Gegensatz zum AES und anderen gängigen symmetrischen Verfahren ist bei Diffie-Hellman und anderen DL-Verfahren die vollständige Schlüsselsuche nicht die effektivste Angriffsmethode. Dies liegt daran, dass es Algorithmen zur Berechnung des diskreten Logarithmus gibt, die – trotz eines großen Aufwands – immer noch deutlich schneller sind als ein bloßes Durchprobieren. Dieser Nachteil fällt jedoch nicht ins Gewicht, wenn Alice und Bob die Bit-Länge von g, x und y geeignet wählen. Dabei gilt logischerweise: Je größer die verwendeten Zahlen, desto sicherer das Verfahren, aber auch desto aufwendiger. Wählen Alice und Bob beispielsweise für alle drei Werte 1.024-Bit-Zahlen (dies entspricht einer Schlüssellänge von 1.024 Bit), dann sind sie gegenüber allen bekannten Angriffen auf Diffie-Hellman gewappnet, ohne dass die Performanz zu sehr in den Keller geht. Neuere Implementierungen verwenden häufig 2.048 Bit oder mehr. Darüber hinaus gibt es einen offensichtlichen Angriff auf Diffie-Hellman, der zur Familie der Man-in-the-Middle-Attacken gehört. Er wird in Abschnitt 20.3.3 beschrieben. Durch den Einsatz einer Public-Key-Infrastruktur (siehe Teil 4 dieses Buchs) können Alice und Bob diesen Angriff verhindern.

Bleibt noch die Frage, warum g ein Generator der Gruppe $Z(p,\cdot)$ sein sollte. Die Antwort erkennen Sie, wenn Alice und Bob beispielsweise g=1 wählen. Das Diffie-Hellman-Verfahren funktioniert bei dieser Wahl zwar nach wie vor, doch der Schlüssel k ist in jedem Fall 1 – und das ist natürlich nicht erwünscht. Der Grund für dieses Ergebnis ist, dass 1 der Generator der Untergruppe von $Z(p,\cdot)$ mit einem Element ist. Ähnlich unsicher ist das Verfahren, wenn Alice und Bob für g den Generator einer anderen kleinen Untergruppe wählen. Handelt es sich dagegen um einen Generator einer großen Untergruppe (z.B. der Größe $p/2$), dann kann nichts passieren. Am besten ist es aber allemal, wenn sich Alice und Bob gleich auf einen Generator der ganzen Gruppe festlegen – da die Hälfte aller Zahlen kleiner als p solche Generatoren sind, gibt es genug Auswahl.

Bei richtigem Einsatz ist das Diffie-Hellman-Verfahren ein äußerst wirksames Werkzeug für den Schlüsselaustausch in einem Computernetz. Kein Wunder, dass diese Methode in unterschiedlichen Versionen zu den wichtigsten Verfahren der Kryptografie überhaupt gehört. Es gibt unzählige Implementierungen und zahlreiche Netzwerkprotokolle, die es verwenden. Neben dem RSA-Verfahren ist Diffie-Hellman das zweite große Standbein der asymmetrischen Kryptografie.

11.2.2 MQV

Der Diffie-Hellman-Schlüsselaustausch ist zwar eine schöne Sache, doch er ist nicht perfekt. Insbesondere bietet er in seiner bisher beschriebenen Form keine Sicherheit gegen einige Angriffe, zu denen wir erst in Abschnitt 20.3.5 kommen. Diese Angriffe nennen sich Known-Key-Attacken und gliedern sich in die Teilbereiche Forward Security, Backward Security und Key Compromise Impersonation Security. Verkürzt ausgedrückt besteht das Problem darin, dass Mallory einen fast unbegrenzten Schaden anrichten kann, sobald er Alices oder Bobs privaten Diffie-Hellman-Schlüssel kennt. Diese Gefahr lässt sich etwas abmildern, indem Alice und Bob jeweils mit mehreren Schlüsselpaaren arbeiten. Erfährt Mallory eines davon, dann ist damit noch nicht die gesamte Sicherheit infrage gestellt. Das bekannteste Verfahren, das Diffie-Hellman durch die Verwendung mehrerer Schlüsselpaare sicherer macht, hat den Namen **MQV** [MeQuVa]. Es ist nach den drei Kryptografen Menezes, Qu und Vanstone benannt, die es entwickelten und 1995 der Öffentlichkeit vorstellten. 1998 präsentierten Laurie Law und Jerry Solinas eine Weiterentwicklung [LMQSV]. Diese ist heute meist gemeint, wenn von MQV die Rede ist.

Funktionsweise von MQV

Bei MQV arbeiten Alice und Bob jeweils mit zwei Diffie-Hellman-Schlüsselpaaren, wobei jeder der beiden ein *dauerhaftes* und ein *temporäres Schlüsselpaar* besitzt. In der Praxis stellt oft eine Public-Key-Infrastruktur (siehe Kapitel 26) die dauerhaften öffentlichen Schlüssel zur Verfügung. Das temporäre Schlüsselpaar wird dagegen vor jeder Kommunikation von Alice und Bob neu generiert. Der Vorteil dieser Vorgehensweise ist offensichtlich: Selbst wenn Mallory sowohl Alices temporären als auch ihren dauerhaften privaten Schlüssel erfährt, ist der Schaden begrenzt, weil bei der nächsten Kommunikation ein neuer temporärer privater Schlüssel zum Einsatz kommt.

MQV hat ähnliche Voraussetzungen wie Diffie-Hellman. Wir benötigen eine Primzahl p und eine natürliche Zahl g. g sollte idealerweise ein Generator der Gruppe $Z(p,\cdot)$ sein, das Verfahren funktioniert aber auch, wenn g einen anderen Wert kleiner als p annimmt. Alice hat den dauerhaften privaten Schlüssel x, Bob den dauerhaften privaten Schlüssel y. x und y sind natürliche Zahlen. Alice hat den dauerhaften öffentlichen Schlüssel $a=g^x \pmod p$, Bob hat den dauerhaften öffentlichen Schlüssel $b=g^y \pmod p$. Im Falle von Alice lauten die temporären Schlüssel t und $g^t \pmod p$, im Falle von Bob u und $g^u \pmod p$.

In seiner Standard-Version (*Two-Pass-Variante*) sieht MQV vor, dass Alice ihren öffentlichen temporären Schlüssel an Bob schickt sowie umgekehrt. Nun benötigen wir folgende zusätzliche Werte:

- m = halbe Bit-Länge von p (bei einer Schlüssellänge von 1.024 Bit hat m also den Wert 512).
- $d = 2^m + (g^t \bmod 2^m)$
- $e = 2^m + (g^u \bmod 2^m)$

Alice berechnet nun $k=(g^b \cdot g^{xe})^{a+d \cdot x} \bmod (p)$. Bob berechnet $k=(g^a \cdot g^{yd})^{b+ey} \pmod p$. k hat in beiden Fällen denselben Wert. Mallory kann diesen Wert aus g^a, g^b, g^x und g^y nicht berechnen, ohne einen diskreten Logarithmus zu lösen (was wir als unmöglich annehmen können). Alice und Bob können also k als geheimen Schlüssel verwenden. Neben der Two-Pass-Variante gibt es auch eine *One-Pass-* und eine *Three-Pass-Variante*. Die Varianten sind nach der Anzahl der übermittelten Nachrichten benannt. One-Pass ist vor allem für E-Mails gedacht, Three-Pass sieht die zusätzliche Nutzung einer kryptografischen Hashfunktion vor.

Bewertung von MQV

MQV mag Ihnen auf den ersten Blick etwas kompliziert vorkommen. In der Tat könnte man die Sicherheitsziele des Verfahrens auch erreichen, wenn Alice und Bob das Diffie-Hellmann-Verfahren doppelt verwenden und dabei die Berechnungen mit den dauerhaften und den temporären Schlüsseln voneinander trennen. Dies wäre zwar übersichtlicher, dafür aber weniger performant. Der große Vorteil von MQV liegt nämlich darin, dass für Alice bzw. Bob jeweils nur eine Berechnung notwendig ist, um den Schlüssel k zu ermitteln. Diese Berechnung ist nur um 25 Prozent aufwendiger als beim gewöhnlichen Diffie-Hellman (die 25 Prozent kommen durch die halbe Bit-Länge zustande). Würden Alice und Bob dagegen ihre temporären und dauerhaften Schlüssel jeweils separat einsetzen, dann läge der Zusatzaufwand bei 100 Prozent.

Nach Angaben seiner Erfinder verhindert MQV nicht nur Known-Key-Attacken, sondern erfüllt zusätzlich einige weitere subtile Sicherheitsziele, die in diesem Buch nicht näher beschrieben werden. Durch diese Sicherheit bei hoher Performanz ist MQV zu einem recht populären Verfahren geworden. Es ist beispielsweise in P1363 und in mehreren US-Standards standardisiert. Einige Varianten sind patentiert.

HMQV

In Abschnitt 20.2 werde ich die Problematik ansprechen, dass ein gutes Krypto-Protokoll oft schwieriger zu entwickeln ist als ein gutes Krypto-Verfahren. MQV ist ein typisches Beispiel dafür. Obwohl MQV in zahlreichen Standards enthalten ist und häufig eingesetzt wird, ist es durchaus umstritten. Der Grund dafür sind die zahlreichen unterschiedlichen Sicherheitsziele, die das Verfahren erfüllen soll (oder muss). 2005 veröffentlichte der Kryptograf Hugo Krawczyk eine Arbeit, in der er MQV mit einem von ihm entwickelten Protokoll-Analyseverfahren untersuchte. Er kam zu dem Schluss, dass MQV einige der angestrebten Sicherheits-

ziele verfehlte und zudem auch einige grundsätzliche Sicherheitsschwächen aufwies. Als Alternative schlug Krawczyk eine von ihm geänderte MQV-Version vor, die er **HMQV** nannte [Krawcz]. Das *H* steht für »Hashed« und erklärt sich dadurch, dass HMQV den Einsatz einer Hashfunktion vorsieht. MQV-Miterfinder Alfred Menezes veröffentlichte daraufhin fast postwendend einen Fachaufsatz, indem er die Vorwürfe Krawczyks zurückwies. Menezes bezeichnete die Beweisführung Krawczyks als fehlerhaft und wies auf einen »verhängnisvollen Sicherheitsfehler«in HMQV hin. Bisher hat sich HMQV nicht durchgesetzt, während MQV immer mehr an Bedeutung gewinnt.

11.3 RSA

Ron Rivest, den gegenwärtig wohl bedeutendsten Kryptografen, habe ich Ihnen bereits als Erfinder diverser symmetrischer Verfahren wie RC2, RC4, RC5 und RC6 vorgestellt. Auch seinen Kollegen Adi Shamir habe ich bereits mehrfach erwähnt. Zusammen mit Leonard Adleman entwickelten die beiden Ausnahmekönner ein Krypto-Verfahren, das nach den Nachnamen der Erfinder als **RSA** bezeichnet wird. Es handelt sich dabei nicht nur um eines der ältesten, sondern um das mit Abstand wichtigste und bekannteste asymmetrische Verschlüsselungsverfahren. Im Vergleich zum simplen Diffie-Hellman-Schlüsselaustausch ist RSA vielseitiger: Es kann nicht nur zum Schlüsselaustausch, sondern auch zur asymmetrischen Verschlüsselung verwendet werden.

11.3.1 Funktionsweise des RSA-Verfahrens

Im Gegensatz zur symmetrischen Kryptografie sollten Sie sich Klartext, Geheimtext und Schlüssel in diesem Zusammenhang nicht als Bit-Folgen, sondern als natürliche Zahlen vorstellen. Für den Computer macht dies ohnehin keinen Unterschied, da dieser alle Daten als Bit-Folge abspeichert. Bei einer zu großen Zahl wird der Klartext in Blöcke aufgeteilt, die einzeln verschlüsselt werden.

Um die Funktionsweise von RSA zu verstehen, sollten Sie sich noch einmal an Abschnitt 11.1.2 erinnern. Dort habe ich beschrieben, dass die Primzahl-Multiplikation eine Einwegfunktion ist. Der entscheidende Schritt besteht nun darin, dass wir aus der besagten Einwegfunktion eine Falltürfunktion basteln. Dazu müssen Sie sich an das Modulo-Wurzelziehen aus Abschnitt 11.1 erinnern: Die *a*-te Wurzel der Zahl *b* modulo *n* lässt sich leicht berechnen, wenn man $\varphi(n)$ kennt. Wie ich ebenfalls bereits angesprochen habe, kann man $\varphi(n)$ wiederum einfach berechnen, wenn es sich dabei um das Produkt zweier Primzahlen *p* und *q* handelt. Dann gilt nämlich $\varphi(n)=(p-1)\cdot(q-1)$. Die Frage ist nun: Was ist, wenn man *p* und *q* nicht kennt? Dann lässt sich $\varphi(n)$ auf diese Weise nicht berechnen und damit auch die Wurzel nicht. Und wie ich gerade ausgeführt habe, ist es ausgesprochen schwierig, zwei Primzahlen *p* und *q* zu bestimmen, von denen man nur das Produkt *n* kennt.

Nun stellt sich die Frage, ob es einen anderen praktikablen Weg zur Berechnung der Modulo-Wurzel gibt. Die Antwort: nach heutigem Kenntnisstand nein. Der Weg über die φ-Funktion ist der einzige bekannte. Das Modulo-Potenzieren ist damit eine Falltürfunktion, wobei die Falltürinformation die Faktorisierung des Modulus ist. Mit dieser Vorarbeit lässt sich das RSA-Verfahren leicht erklären. Will Bob eine Nachricht an Alice mit dem RSA-Verfahren verschlüsseln, dann ergibt sich folgender Ablauf:

1. Zunächst ist etwas Vorarbeit von Alice notwendig: Sie wählt zufällig zwei große Primzahlen p und q und berechnet daraus $n:=p\cdot q$.
2. Alice wählt wiederum zufällig eine natürliche Zahl e, die teilerfremd zu $\varphi(n)$ ist (Sie erinnern sich: $\varphi(n)=(p-1)\cdot(q-1)$). Die Zahlen n und e bilden zusammen den öffentlichen Schlüssel, den Alice öffentlich bekannt macht und den ruhig auch Mallory erfahren darf.
3. Alice berechnet $d:=e^{-1} (\text{mod } \varphi(n))$. d ist ihr geheimer Schlüssel, den sie natürlich für sich behalten muss.
4. Kennt Bob Alices öffentlichen Schlüssel e, dann verschlüsselt er damit seine Nachricht m, die er wie oben gezeigt als Zahl betrachtet. Dazu berechnet er $c:=m^e (\text{mod } n)$. c ist der Geheimtext, den er an Alice schickt. Die Verschlüsselung entspricht einer Modulo-Exponentiation.
5. Die von Bob erhaltene Nachricht c kann Alice nun entschlüsseln, indem sie $c^d (\text{mod } n)$ berechnet. Das Ergebnis ist dann genau der Klartext m, den Bob abgeschickt hat. Dieses Entschlüsseln entspricht einem Modulo-Wurzelziehen: Alice zieht die e-te Modulo-Wurzel von c, indem sie die d-te Potenz berechnet. Mallory kann d nicht ermitteln, weil er die Faktorisierung von n nicht kennt.

Dass Alice nun tatsächlich den Klartext in den Händen hält, zeigt folgende Rechnung:

$$c^d = (m^e)^d = m^{1(\text{mod } \varphi(n))} = m \text{ (mod n)}$$

Da Bob die Verschlüsselungsfunktion m^e (mod n) für jede Nachricht m an Alice berechnen muss und das öffentlich bekannte e dabei stets gleich ist, liegt es nahe, für e einen vorteilhaften Wert zu wählen. Als besonders praktisch haben sich hierfür die Primzahlen 3, 17 und 65.537 erwiesen, da diese in ihrer Binärdarstellung nur wenige Einsen aufweisen und daher eine schnelle Exponentiation ermöglichen. Wird eine dieser Zahlen verwendet, dann ist eine RSA-Verschlüsselung logischerweise um ein Vielfaches schneller als die Exponentiation mit einem 1.024-Bit-Exponenten, wie sie bei Diffie-Hellman benötigt wird. Da e somit meist klein und konstant ist, ist mit Schlüssellänge im Zusammenhang mit dem RSA-Verfahren nahezu immer die Länge der Zahl n gemeint.

11.3.2 Ein Beispiel

Schauen wir uns nun auch zum RSA-Verfahren ein Beispiel an, bei dem wir kleine Zahlen verwenden. Wählt Alice p=5 und q=17, dann ergibt sich:

$n = p \cdot q = 5 \cdot 17 = 85$

$\varphi(n) = (p-1) \cdot (q-1) = 4 \cdot 16 = 64$

Wenn Alice außerdem e=3 wählt, dann besteht ihr öffentlicher Schlüssel aus den Zahlen 85 und 3. Ihren privaten Schlüssel kann sie einfach berechnen, da sie die Faktorisierung von n und damit auch $\varphi(n)$ kennt. Es gilt $d = e^{-1} = 43 \pmod{64}$, da $3 \cdot 43 = 1 \pmod{64}$. Alices privater Schlüssel ist damit 43.

Nehmen wir nun an, die Nachricht, die Bob an Alice schicken will, sei im Klartext die Zahl m=2. Bob berechnet nun den Geheimtext mithilfe von Alices öffentlichem Schlüssel:

$c = m^e \pmod{n} = 2^3 = 8 \pmod{85}$

Der Geheimtext, den Bob an Alice schickt, lautet damit also 8. Bob hat ihn berechnet, ohne Alices privaten Schlüssel d=43 zu kennen. Zur Entschlüsselung berechnet Alice:

$m = c^d \pmod{n} = 8^{43} \pmod{85} = 2$

Mit ihrem privaten Schlüssel c=43 kann Alice damit die Nachricht entschlüsseln.

11.3.3 Sicherheit des RSA-Verfahrens

Da Alice die Zahl n (und damit die Schlüssellänge) beliebig groß wählen kann, ist RSA ein Verfahren mit variabler Schlüssellänge. Wie Sie sich denken können, gilt dabei ein Grundsatz, der für alle guten Krypto-Verfahren gilt: Je länger der Schlüssel, desto sicherer das Verfahren. Wie beim Diffie-Hellman-Schlüsselaustausch gibt es auch beim RSA-Verfahren (im Gegensatz zum DES) deutlich schnellere Angriffe als die vollständige Schlüsselsuche, weshalb n erheblich länger sein muss als die 128 Bit eines AES-Schlüssels. 1.024 Bit und 2.048 Bit sind momentan die gängigsten Werte.

Im Vergleich zum AES gibt es beim RSA-Verfahren deutlich mehr Ansätze für die Kryptoanalyse. Dies liegt zum einen daran, dass die Werte p, q, e und n frei wählbar sind. Eine falsche Wahl kann Mallory das Leben erheblich erleichtern. Zum anderen zeigt die Erfahrung, dass komplizierte mathematische Verfahren mehr Angriffsfläche bieten als die einfachen Bit-Operationen der gängigen symmetrischen Verfahren. In den folgenden Unterkapiteln werde ich Ihnen die wichtigsten Kryptoanalyse-Ergebnisse vorstellen, die bisher gegen das RSA-Verfahren vorliegen. Ein Ergebnis kann ich jedoch vorwegnehmen: Bei richtiger Implementierung ist RSA nach gegenwärtigem Stand der Forschung ein sehr sicheres Verfahren.

Allgemeines zum Thema RSA-Kryptoanalyse

Bei asymmetrischen Verschlüsselungsverfahren ist zwar vieles anders als bei symmetrischen. Das Ziel von Abhörer Mallory ist jedoch weitgehend das gleiche: Er will an den Klartext oder einen privaten Schlüssel herankommen. Während es bei symmetrischen Verfahren Ciphertext-Only-, Known-Plaintext- und Chosen-Plaintext-Attacken gibt (Abschnitt 4.1.2), spielen bei asymmetrischen Verschlüsselungsalgorithmen nur zwei Angriffstypen eine wesentliche Rolle:

- **Public-Key-Only-Attacke**: In diesem Fall steht Mallory nur Alices öffentlicher Schlüssel zur Verfügung. Diesen kann er nutzen, um beliebige Klartexte zu verschlüsseln. Eine Public-Key-Only-Attacke schließt daher immer ein, dass Mallory auch Klartext-Geheimtext-Paare analysieren kann.
- **Chosen-Ciphertext-Attacke**: In diesem Fall kann Mallory einen Geheimtext frei wählen und ihn von Alice entschlüsseln lassen.

Eine Public-Key-Only-Attacke ist in unserem Alice-Bob-Mallory-Modell immer möglich, da der öffentliche Schlüssel per Definition allen bekannt ist. Die Voraussetzungen für eine Chosen-Ciphertext-Attacke sind dagegen nur in bestimmten Fällen gegeben, etwa wenn Mallory Zugriff auf ein RSA-Hardwaremodul hat. Die Chosen-Ciphertext-Attacke bietet zwar bessere Voraussetzungen als eine Public-Key-Only-Attacke, doch die Vorteile sind nicht wirklich dramatisch. Zu den Besonderheiten asymmetrischer Verschlüsselungsverfahren gehört daher, dass Mallory fast alle seine Angriffe ohne Insiderinformationen (bekannte oder gewählte Klartexte) starten kann.

Vollständige Schlüsselsuche

Theoretisch kann Mallory das RSA-Verfahren durch eine vollständige Schlüsselsuche brechen (dies wäre eine Public-Key-Only-Attacke). Praktisch ist das jedoch schon bei einer bescheidenen Schlüssellänge von 256 Bit ein mehr als aussichtsloses Unterfangen. Mallory müsste dazu im Schnitt 2^{255} Schlüssel durchprobieren (das ist die Hälfte der 2^{256} möglichen Schlüssel). Dies entspricht 10^{76} Schlüsseln – eine Zahl, die größer ist als die Anzahl der Atome im Universum. Da das Universum »nur« 10^{18} Sekunden alt ist, müsste Mallory schon 10^{58} Schlüssel pro Sekunde durchprobieren, um überhaupt in einem erfassbaren Zeitraum einen Erfolg erwarten zu können. Wenn Mallory also nichts Besseres einfällt als die vollständige Schlüsselsuche, dann kann Alice und Bob nichts passieren.

Faktorisierungsangriff

Wie bereits erwähnt, ist die vollständige Schlüsselsuche bei weitem nicht der beste Angriff auf das RSA-Verfahren. Deutlich besser ist folgende naheliegende Public-Key-Only-Attacke: Mallory nimmt die Zahl n aus Alices öffentlichem Schlüssel und versucht, diese in die zwei Primzahlen p und q zu zerlegen. Mit p und q kann

er problemlos Alices privaten Schlüssel berechnen. Dieser Angriff wird als **Faktorisierungsangriff** bezeichnet. Die Funktionsweise des RSA-Verfahrens beruht jedoch gerade darauf, dass die Berechnung von p und q aus n besonders schwierig ist – es handelt sich ja um eine Einwegfunktion. Eine Faktorisierungsattacke läuft daher auf das Umkehren einer Einwegfunktion hinaus und ist äußerst aufwendig.

Wie aufwendig eine Faktorisierungsattacke ist, hängt von der Methode ab, die zur Faktorisierung von n verwendet wird. Möglichkeiten gibt es dazu viele – man könnte problemlos ein ganzes Buch über Faktorisierungsverfahren schreiben. Uns interessiert an dieser Stelle jedoch nur eines: Keine der bisher bekannten Methoden zur Faktorisierung ist auch nur annähernd leistungsfähig genug, um mit realistischem Aufwand eine 1.024-Bit-Zahl zu zerlegen (wäre es anders, dann hätte sich RSA nie durchgesetzt). Vermutlich wird das auch in Zukunft so bleiben. Es wird zwar weiterhin geforscht, und es sind weitere Fortschritte zu erwarten – ein Quantensprung auf dem Gebiet der Faktorisierungsverfahren ist derzeit jedoch nicht in Sicht.

Sicherlich fragen Sie sich nun, bis zu welcher Größe Zahlen heutzutage faktorisiert werden können. Der gegenwärtige Weltrekord liegt bei 768 Bit, was 232 Dezimalstellen entspricht (siehe Abschnitt 40.4.1). Dieser Rekord bezieht sich natürlich nur auf bekannt gewordene Faktorisierungen. Zweifellos sind die NSA und andere Geheimdienste in der Lage, auch längere Zahlen zu zerlegen. Daraus können wir schließen, dass 512 Bit als RSA-Schlüssellänge für besonders sicherheitskritische Daten nicht mehr ausreichen. Auch 1.024 Bit gelten inzwischen als knapp bemessen. 2.048 Bit dürften dagegen vorläufig nicht zu knacken sein.

Low-Exponent-Attacke

Wie in Abschnitt 11.3.1 bereits erwähnt, verwenden viele RSA-Implementierungen kleine Werte für die Zahl e (meist 3 oder 17), da diese eine besonders schnelle Verschlüsselung ermöglichen. Dabei ist jedoch Vorsicht geboten: Wird die gleiche Nachricht an e Empfänger gesendet und jeweils mit der gleichen Zahl e verschlüsselt, dann gibt es einen Angriff, mit dem Mallory den Geheimtext wiederherstellen kann (**Low-Exponent-Attacke**). Auch wenn nur Teile der Nachricht gleich sind, ist dies teilweise schon möglich. Natürlich gibt es gegen eine Low-Exponent-Attacke wirksame Gegenmaßnahmen. So können Alice und Bob etwa die Zahl e einfach etwas größer wählen. 65.537 ist etwa ähnlich günstig für die Performanz wie 3 und 17, aber deutlich weniger anfällig. Der PKCS#1-Standard (Abschnitt 19.4.1) geht dagegen einen anderen Weg: Er sieht vor, dass jede zu verschlüsselnde Nachricht in einer bestimmten Weise aufbereitet wird, wodurch gleiche Nachrichten nicht mehr vorkommen.

RSA und Quantencomputer

In Abschnitt 7.3.3 haben Sie erfahren, was ein Quantencomputer ist. Das Tolle an diesen Geräten ist, dass sie nicht an die klassische Physik gebunden sind und daher Rechenoperationen durchführen können, die für einen herkömmlichen Computer unmöglich sind. Der Nachteil an Quantencomputern besteht darin, dass man noch weit davon entfernt ist, praxistaugliche Exemplare bauen zu können. Möglicherweise wird es nie Quantencomputer geben, die mehr als ein paar einzelne Bits verarbeiten.

Sollten irgendwann doch brauchbare Quantencomputer verfügbar sein, dann dürfte das Knacken von Verschlüsselungen die wichtigste Anwendung werden. Ein Beispiel ist die in Abschnitt 7.3.3 beschriebene vollständige Schlüsselsuche bei symmetrischen Verfahren mit dem Grover-Algorithmus – sie kommt dem Halbieren der Schlüssellänge gleich. Noch deutlich wirkungsvoller ist ein Quantencomputer, wenn es um das Faktorisieren geht. Mit dem sogenannten Shor-Algorithmus kann ein Quantencomputer ein Primzahlprodukt in vergleichsweise kurzer Laufzeit in seine Faktoren zerlegen [Shor]. Das bedeutet: Sollte es einmal größere Quantencomputer geben, dann ist das RSA-Verfahren erledigt. Die Verfahren auf Basis des diskreten Logarithmus wären ebenfalls hinfällig, da sich der diskrete Logarithmus auf das Faktorisierungsproblem zurückführen lässt. Auch Verfahren auf Basis elliptischer Kurven würden in einem solchen Fall ihre Wirkung verlieren. Schlaflose Nächte sind bisher jedoch nicht angebracht, denn noch sind Quantencomputer eher Science Fiction als Realität.

TWINKLE und TWIRL

Das bisher kurioseste Kapitel in der Geschichte der RSA-Kryptoanalyse schlug ausgerechnet RSA-Miterfinder Adi Shamir auf. 1999 präsentierte er bei der Eurocrypt-Konferenz in Prag die Idee zu einer Maschine, die zu einem Primzahlprodukt die beiden zugehörigen Primzahlen berechnen konnte [Sham99, Luck99]. Das Besondere daran war, dass es sich dabei um keinen Computer, sondern um einen optisch-elektronischen Apparat handelte. Shamir gab seiner Maschine den Namen **TWINKLE** (The Weizmann Institute Key Locating Engine). Shamir schätzte, dass eine TWINKLE-Maschine für etwa 5.000 US-Dollar pro Stück realisierbar wäre und bei Faktorisierungsvorgängen 100 bis 1.000 PCs ersetzen könnte. Bisher hat zwar noch niemand ein solches Gerät gebaut (jedenfalls ist nichts bekannt geworden), doch Experten gehen davon aus, dass es möglich wäre.

2003 stellte Shamir zusammen mit einem Kollegen einen TWINKLE-Nachfolger vor, den er **TWIRL** (The Weizmann Institute Relation Locator) nannte [ShaTro]. Auch TWIRL ist bisher eine fiktive Maschine geblieben. Shamir geht davon aus, dass sich damit ein 1.024-Bit-RSA-Schlüssel innerhalb eines Jahres faktorisieren ließe – bei Kosten von einigen Millionen Euro. Dies erscheint zwar immer noch reichlich viel, ist aber für Geheimdienste und Militärorganisationen

durchaus realistisch. TWINKLE und TWIRL legen jedenfalls nahe, dass man für besonders wertvolle Informationen mindestens 2.048 Bit RSA-Schlüssellänge einsetzen sollte.

Wann ist RSA sicher?

Neben den genannten gibt es noch andere wirkungsvolle Angriffe auf das RSA-Verfahren, die jedoch ebenfalls nicht das Verfahren selbst betreffen, sondern lediglich bestimmte Implementierungen. Mehr darüber gibt es in den Abschnitten 17.1 und 19.4. Die zahlreichen Kryptoanalyse-Ergebnisse machen zwei Dinge deutlich: Zum einen ist das RSA-Verfahren anfällig gegenüber falschen Implementierungen. Eine zu kurze Schlüssellänge, falsch gewählte Parameter und andere Fehler können schnell zu erheblichen Sicherheitslücken führen. Andererseits ist RSA nach wie vor ein sehr gutes Verfahren, wenn die genannten Fehler vermieden werden. Wie bei Diffie-Hellman, so ist auch bei RSA eine Schlüssellänge von 1.024 bis 2.048 Bit üblich. Mit Paranoiazuschlag sind es 4.096 Bit. 512 Bit liegen bereits eindeutig diesseits der NSA-Grenze. Allgemein gilt, dass RSA bei gleicher Schlüssellänge um einige Prozent schneller zu knacken ist als Diffie-Hellman.

11.3.4 RSA und der Chinesische Restsatz

Bei einem Umzug in Kryptoland marschiert eine Gruppe in Viererreihen (siehe Abbildung 11–3). Eine Person bleibt übrig und läuft hinterher. An einer engen Stelle stellt die Gruppe auf Dreierreihen um – jetzt bleiben zwei Personen übrig. Nun lautet die Frage: Wie viele Mitglieder hat die Gruppe?

Mathematisch gesprochen kann man diese Aufgabenstellung wie folgt formulieren: $x = 2 \pmod 3$ und $x = 1 \pmod 4$. Es ist nicht besonders schwierig, die Lösung zu ermitteln. Durch Probieren findet man schnell heraus, dass $x = 5$ die Anforderungen erfüllt – demnach könnten fünf Personen zur Gruppe gehören. Weitere mögliche Lösungen sind 17, 29, 41, 53 und so weiter.

Um die Fragestellung etwas allgemeiner zu formulieren, gehen wir davon aus, dass x Personen zur Gruppe gehören. Marschieren die Mitglieder in Reihen der Größe p, dann bleiben a Personen übrig. Bei Reihen der Größe q bleiben b übrig. Mathematisch gesprochen gilt: $x = a \pmod p$ und $x = b \pmod q$. Eine offensichtliche Frage lautet nun: Unter welchen Umständen ist eine solche Aufgabe überhaupt lösbar? Die Antwort liefert ein mathematischer Satz, der bereits den alten Chinesen bekannt war und der daher **Chinesischer Restsatz** genannt wird. Die englische Bezeichnung lautet **CRT** (Chinese Remainder Theorem). Der Chinesische Restsatz besagt, dass das Gleichungssystem bestehend aus $x = a \pmod p$ und $x = b \pmod q$ lösbar ist, wenn p und q teilerfremd sind (handelt es sich bei p und q um Primzahlen, dann ist diese Anforderung erfüllt). Wenn eine Lösung existiert, dann gibt es unendlich viele weitere, allerdings ist stets nur eine Lösung kleiner

als $p \cdot q$. Im obigen Beispiel ist dies leicht zu erkennen: Die unendlich vielen Lösungen heißen 5, 17, 29, 41, 53, ..., wobei nur die erste dieser Zahlen kleiner als $3 \cdot 4 = 12$ ist.

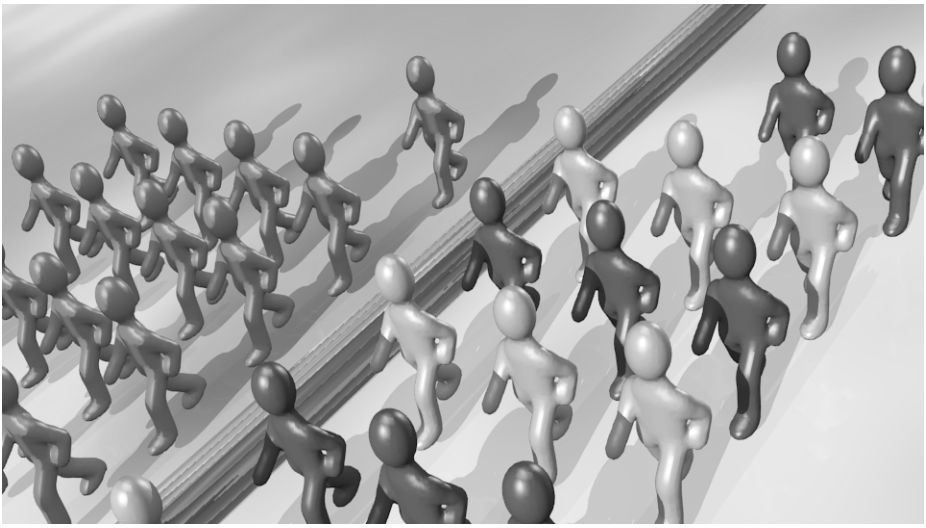


Abb. 11–3 *Eine Gruppe läuft in Viererreihen (links), wobei eine Person übrig bleibt. Läuft sie in Dreierreihen (rechts), dann bleiben zwei Personen übrig. Wie viele Personen zur Gruppe gehören, lässt sich mit dem Chinesischen Restsatz ermitteln.*

Nun ist die Frage, wie wir die Lösung (also den Wert von x) finden. Hierfür gibt es eine Formel. Sie liefert immer die kleinste Lösung und lautet (a muss größer oder gleich b sein):

$$x = (((a - (b \bmod p)) \cdot (q^{-1} (\bmod p))) \bmod p) \cdot q + b$$

Machen wir die Probe aufs Exempel und setzen die oben genannten Werte für a, b, p und q ein. Dann ergibt sich:

$$\begin{aligned} x &= (((2 - (1 \bmod 3)) \cdot (4^{-1} (\bmod 3))) \bmod 3) \cdot 4 + 1 \\ &= (((2 - 1) \cdot 1) \bmod 3) \cdot 4 + 1 \\ &= ((1 \cdot 1) \bmod 3) \cdot 4 + 1 \\ &= 1 \cdot 4 + 1 \\ &= 5 \end{aligned}$$

Spätestens jetzt werden Sie sich fragen, was das alles mit dem RSA-Verfahren zu tun hat. Um die Frage zu beantworten, stellen wir uns c (also den RSA-Geheimtext) als Anzahl der Personen in der Gruppe vor. Wir gehen davon aus, dass sich die Gruppe zunächst in Reihen zu je p Personen aufstellt, wobei a Personen übrig bleiben. Danach stellen sich die Mitglieder in q-Reihen auf – mit b übrigen Personen. p und q sind im Zusammenhang mit RSA immer Primzahlen. Durch den

Chinesischen Restsatz wissen wir nun: Wenn a, b, p und q bekannt sind, können wir daraus c berechnen. Oder anders ausgedrückt: c ist durch a, b, p und q eindeutig bestimmt (man spricht auch von der **CRT-Darstellung** von c). Mit einer ähnlichen Überlegung kann Alice auch die Zahl e mit zwei Modulowerten sowie mit p und q in eine CRT-Darstellung überführen.

Was sich anhört wie eine nutzlose Spielerei, hat in Wirklichkeit einen interessanten Vorteil: Es gibt Methoden, die die Funktion $c^d \pmod n$ besonders schnell berechnen können, wenn c und d in der CRT-Darstellung vorliegen. Mit anderen Worten: Mithilfe des Chinesischen Restsatzes und speziell der CRT-Darstellung kann Alice schneller entschlüsseln. Oft ist die CRT-Variante um das Drei- oder Vierfache schneller als derselbe Vorgang ohne diesen Trick.

Es gibt sehr viele RSA-Implementierungen, die den Chinesischen Restsatz auf diese Weise anwenden. Da sich der Wert von d nur selten ändert, kann Alice dessen CRT-Darstellung speichern und wiederverwenden. Allerdings muss sie alle CRT-Darstellungen (also die von d und c) geheim halten, da Mallory aus diesen die Zahlen p und q und damit Alices privaten Schlüssel berechnen kann. Der Chinesische Restsatz ist also nur für das Entschlüsseln einer Nachricht interessant. Absender Bob kann den CRT-Trick zum Verschlüsseln nicht nutzen, da der die CRT-Darstellungen von m und e nicht kennt.

11.4 Symmetrisch und asymmetrisch im Zusammenspiel

Symmetrische und asymmetrische Verschlüsselung sind die beiden wichtigsten Standbeine der Kryptografie. Obwohl beide einen ähnlichen Zweck erfüllen, gibt es zahlreiche interessante Unterschiede. Diese Unterschiede führen dazu, dass sich symmetrische und asymmetrische Kryptografie keine Konkurrenz machen, sondern sich gegenseitig ergänzen.

11.4.1 Unterschiede zwischen symmetrisch und asymmetrisch

Ich hoffe, dass Ihnen der wesentliche Unterschied zwischen asymmetrischen und symmetrischen Verfahren in der Zwischenzeit klar geworden ist: Bei asymmetrischen Verfahren entfällt das Problem des Schlüsselaustauschs, da der Schlüssel zum Verschlüsseln ohnehin öffentlich bekannt ist. Doch auch im Umgang mit den Verfahren gibt es bedeutende Unterschiede:

- Alle gängigen symmetrischen Verfahren basieren ausschließlich auf einfachen, bit-orientierten Funktionen. Public-Key-Verfahren können dagegen nur mit mathematisch anspruchsvolleren Funktionen realisiert werden.
- Die im ersten Punkt aufgeführte Tatsache führt dazu, dass es relativ schwierig ist, ein neues asymmetrisches Verschlüsselungsverfahren zu erfinden. Symmetrische Verfahren können dagegen in beliebiger Zahl neu entwickelt werden.

- Der in der Praxis wichtigste Unterschied hängt mit dem ersten Punkt zusammen: Eine asymmetrische Verschlüsselung erfordert wesentlich mehr an Rechenkraft als eine symmetrische. Nimmt man RSA und AES als Beispiel, so ist RSA etwa um den Faktor 1.000 langsamer als der AES.
- Asymmetrische Verfahren sind anfälliger gegenüber Implementierungsfehlern als symmetrische Verfahren. Auch dieser Unterschied ist auf den ersten Punkt zurückzuführen.
- Bei symmetrischen Verfahren sollte man sich Schlüssel, Klartext und Geheimtext als Bit-Folgen vorstellen. Bei asymmetrischen Algorithmen dagegen als große Zahlen beziehungsweise Zahlenpaare.
- Bei symmetrischen Verfahren ist die Schlüssellänge meist vorgegeben oder aus verschiedenen Vorgaben wählbar. Bei asymmetrischen Verfahren ist die Schlüssellänge dagegen völlig variabel, weil die als Schlüssel verwendete Zahl eine beliebige Größe annehmen kann.

Die genannten Unterschiede sind allesamt Erfahrungstatsachen, die nicht mathematisch bewiesen sind. Es zählt zu den offenen Fragen der Kryptografie, ob es nicht vielleicht doch asymmetrische Verfahren gibt, die nur einfache Bit-Operationen verwenden und mit einer Schlüssellänge von 128 Bit auskommen.

11.4.2 Hybridverfahren

RSA und alle anderen Public-Key-Verschlüsselungsverfahren haben einen bereits erwähnten Nachteil: Sie sind vergleichsweise langsam. Eine RSA-Entschlüsselung ist etwa um den Faktor 1.000 langsamer als der gleiche Vorgang beim AES. Auch eine RSA-Verschlüsselung mit einer kleinen Primzahl ist noch deutlich aufwendiger als eine AES-Verschlüsselung. Vor allem bei längeren Nachrichten, die in einzelne Blöcke aufgeteilt werden müssen, ist eine solche Verzögerung sehr lästig. RSA und andere Public-Key-Verschlüsselungsverfahren werden daher in der Praxis fast nie verwendet, um ganze Nachrichten zu verschlüsseln. Stattdessen wird damit in der Regel ein Schlüssel für ein symmetrisches Verfahren übermittelt, um damit den Rest der Kommunikation zu verschlüsseln. Somit wird RSA in der Praxis nicht als Verschlüsselungsverfahren, sondern als Schlüsselaustauschverfahren eingesetzt (wie Diffie-Hellman). Die gemeinsame Verwendung von Secret Key und Public Key auf diese Weise wird **Hybridverfahren** genannt

Ein Großteil aller Krypto-Implementierungen arbeitet mit Hybridverfahren. Zum überwiegenden Teil ist RSA dabei der asymmetrische Vertreter, während der AES oder ein anderes Verfahren den symmetrischen Part übernimmt. Wenn nicht RSA, dann wird fast immer Diffie-Hellman oder ein Verfahren auf Basis elliptischer Kurven (siehe Abschnitt 13.1.2) verwendet. Hybridverfahren sind auch der Grund, warum symmetrische und asymmetrische Verfahren sich gegenseitig keine Konkurrenzkämpfe liefern. Vielmehr ergänzen sich beide Mitglieder der Krypto-Familie gegenseitig..

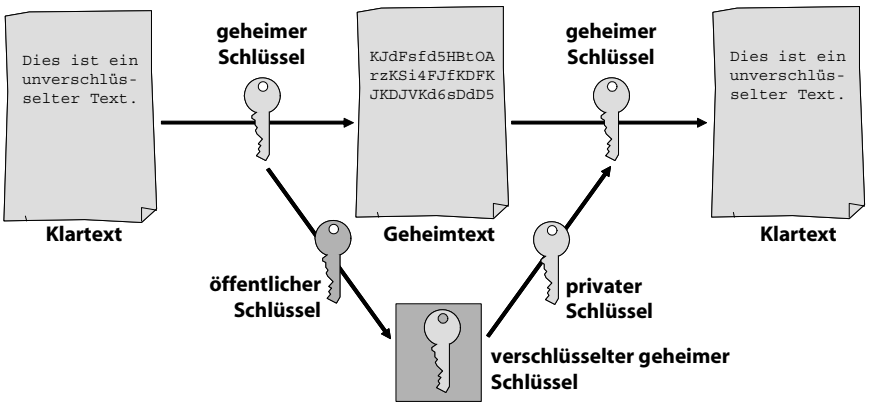


Abb. 11–4 *Symmetrische und asymmetrische Krypto-Verfahren werden oft zusammen eingesetzt (Hybridverfahren). Dabei wird mit dem asymmetrischen Verfahren lediglich ein Schlüssel übertragen, mit dem die restliche Nachricht symmetrisch verschlüsselt ist.*

12 Digitale Signaturen

Alice hat von Online-Händler Otto einen Staubsauger geliefert bekommen, den sie nie bestellt hat. Der Grund: Bösewicht Mallory hat eine E-Mail an Otto geschickt und sich dabei als Alice ausgegeben. Leider ist es nur zu einfach, den Absender einer E-Mail zu fälschen (siehe Abschnitt 3.4.5), und man braucht nicht viel Fantasie, um sich auszumalen, welchen Unfug Mallory damit sonst noch anstellen kann. Diese Umstände sind jedoch bekannt, und daher ist das Vertrauen, das einer E-Mail entgegengebracht wird, im Allgemeinen nicht sehr hoch. Das Problem falscher Absender ist allerdings nicht auf den Bereich der Computernetzwerke beschränkt. Schließlich kann auch per Post jeder, der dazu Lust und Laune hat, einen Brief mit beliebiger Absenderangabe durch die Gegend schicken. Im richtigen Leben löst man dieses Problem mit einer Unterschrift. In der digitalen Welt verwendet man eine **digitale Signatur**.

12.1 Was ist eine digitale Signatur?

Unter einer digitalen Signatur darf man sich keine Unterschrift von Hand vorstellen, die eingescannt und digital gespeichert wird. Eine solche könnte schließlich leicht kopiert und zweckentfremdet werden. Vielmehr versteht man darunter eine spezielle, schlüsselabhängige Prüfsumme, die im Zusammenhang mit einem digitalen Dokument ähnliche Eigenschaften aufweist wie eine Unterschrift von Hand. Damit die Bezeichnung Signatur gerechtfertigt ist, muss diese Prüfsumme folgende Voraussetzungen erfüllen:

- Sie darf nicht zu fälschen sein.
- Ihre Echtheit muss überprüfbar sein.
- Sie darf nicht unbemerkt von einem Dokument zum anderen übertragen werden können.
- Das dazugehörende Dokument darf nicht unbemerkt verändert werden können.

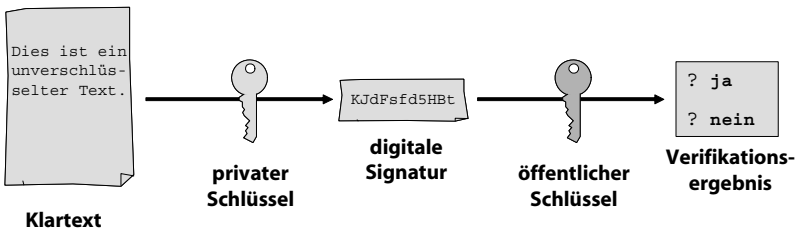


Abb. 12–1 *Eine digitale Signatur wird mit einem privaten Schlüssel erzeugt. Die Überprüfung der Echtheit erfolgt mit dem zugehörigen öffentlichen Schlüssel.*

Der erste Punkt wird sinnvollerweise wiederum dadurch gelöst, dass ein Passwort (auch in diesem Fall privater Schlüssel genannt) in die digitale Signatur eingeht. Damit kann jeder seine eigene, eindeutige digitale Unterschrift generieren, sofern niemand sonst den Schlüssel kennt. Die Tücke bei der Sache ist jedoch, dass es auch ohne Kenntnis des Schlüssels möglich sein muss, die Echtheit der Unterschrift zu überprüfen. Dazu wird wiederum ein öffentlicher Schlüssel verwendet (auch diese Bezeichnung kennen Sie bereits aus der asymmetrischen Verschlüsselung).

Das Ganze wollen wir noch einmal mathematisch formulieren: Gegeben sei eine Nachricht m, die Alice signieren will. Dazu hat sie einen öffentlich bekannten Schlüssel a und einen geheimen Schlüssel x. Das Unterschreiben entspricht der Berechnung einer Funktion $u(m,x)$, die Unterschrift (Signatur) ist $s=u(m,x)$. Da nur Alice x kennt, kann nur sie s berechnen. Will Bob nun prüfen (**verifizieren**), ob die Unterschrift echt ist, dann verwendet er eine andere Funktion v. Er berechnet $m'=v(a,s)$. Falls $m=m'$ gilt, ist die Unterschrift echt.

12.2 RSA als Signaturverfahren

Klar ist, dass a und x voneinander abhängen. Weniger klar ist dagegen, wie man diese Funktionen wählt, damit das Ganze nicht nur funktioniert, sondern auch sicher ist. Es ist nicht einmal offensichtlich, dass es überhaupt brauchbare digitale Signaturen gibt.

12.2.1 Funktionsweise

Zum Glück gibt es brauchbare Verfahren für digitale Signaturen, und verblüffenderweise muss ich Ihnen dazu nicht einmal ein neues Verfahren vorstellen. Für digitale Signaturen können Alice und Bob nämlich das in Abschnitt 11.3 beschriebene RSA-Verfahren einsetzen, wenn sie dazu lediglich die Verwendung der Schlüssel vertauschen. Will Alice eine Nachricht signieren, so »entschlüsselt« sie diese mit ihrem geheimen RSA-Schlüssel d (obwohl die Nachricht unverschlüsselt ist). Der resultierende »Klartext« ist die digitale Signatur. Bob verifiziert diese, indem er sie mit den Werten e und n per RSA »verschlüsselt«. Erhält er so die ursprüngliche Nachricht zurück, dann ist die digitale Signatur echt.

Es ist nicht ohne weiteres einzusehen, dass diese Anwendung von RSA als Signaturverfahren funktioniert. Den Beweis dafür möchte ich Ihnen an dieser Stelle jedoch ersparen, es genügt, wenn Sie sich merken: RSA kann sowohl zum Verschlüsseln als auch zum Signieren eingesetzt werden. Auch unabhängig von RSA hängen Signaturverfahren eng mit Public-Key-Verschlüsselungsverfahren zusammen, weshalb sie auch zur asymmetrischen Kryptografie gezählt werden. Leider haben Signaturverfahren auch die gleichen Nachteile wie asymmetrische Verschlüsselungsverfahren: Es gibt nur wenige, und diese sind recht langsam.

RSA hat unter den Signaturverfahren eine ähnliche Vormachtstellung wie unter den asymmetrischen Verschlüsselungsverfahren. In den über 30 Jahren seines Bestehens hat sich das RSA-Verfahren damit zum wichtigsten Eckpfeiler der Kryptografie überhaupt entwickelt. Kein anderes asymmetrisches Verfahren ist gleichzeitig so vielseitig verwendbar, so gut erforscht und so einfach zu implementieren. RSA wird uns daher im Verlauf dieses Buchs noch öfter begegnen.

12.2.2 Sicherheit von RSA-Signaturen

Was ich in Abschnitt 11.3 über die Sicherheit des RSA-Verfahrens berichtet habe, gilt auch, wenn dieses zur digitalen Signatur verwendet wird. Mallory kann es also mit einer vollständigen Schlüsselsuche oder mit einem Faktorisierungsangriff versuchen, um Alices privaten Schlüssel zu ermitteln. Wenn allerdings mindestens 1.024 Bit für den Modulus zum Einsatz kommen, dann sollte dies nicht funktionieren. Für Paranoide sind 2.048 Bit angebracht. Auch gegen eine Low-Exponent-Attacke können RSA-Signaturen anfällig sein.

Daneben gibt es auch einige Angriffe, die nur beim Signieren mit RSA funktionieren. Bei diesen Angriffen versucht Mallory, Alices digitale Signatur zu fälschen, ohne Alices Schlüssel zu kennen. Beispielsweise kann Mallory Alices Signatur-Software so verändern, dass das, was Alice am Bildschirm sieht, nicht das ist, was sie signiert (Darstellungsangriff). Dieser Angriff ist zweifellos die größte Schwachstelle digitaler Signaturen überhaupt. In Abschnitt 17.2 gibt es mehr dazu.

Unterschiebe-Angriff

Gefährlich kann es werden, wenn Alice zum Verschlüsseln und Signieren denselben Schlüssel verwendet. Mallory kann in diesem Fall beispielsweise einen Geheimtext nehmen, den Bob mit Alices öffentlichem Schlüssel verschlüsselt hat, und diesen von Alice signieren lassen (**Unterschiebe-Angriff**). Da das RSA-Signieren dem RSA-Entschlüsseln entspricht, entsteht hierbei der Klartext – möglicherweise ohne dass Alice dies merkt. Um einen solchen Angriff zu verhindern, sollten Alice und Bob zum Verschlüsseln und Signieren jeweils unterschiedliche Schlüsselpaare verwenden. Praktisch alle gängigen Krypto-Implementierungen sehen eine solche Schlüsseltrennung vor.

RSA und Zufallsorakel

Das RSA-Verfahren ist kein Zufallsorakel. Dies zeigt folgender Angriff, den Mallory starten kann:

1. Mallory generiert drei Nachrichten m_1, m_2 und m_3, die folgende Eigenschaft erfüllen: $m_1 \cdot m_2 = m_3$.
2. Mallory lässt Alice m_1 signieren und erhält die Signatur s_1.
3. Mallory lässt Alice m_2 signieren und erhält die Signatur s_2.

Die Eigenschaften des RSA-Verfahrens bringen es mit sich, dass nun $s_3 = s_1 \cdot s_2 \bmod n$ gilt. Mallory besitzt nun eine gültige Signatur von Alice für m_3. Es gibt noch weitere Beispiele, die belegen, dass RSA kein Zufallsorakel ist. Abschnitt 19.4 zeigt, was Alice und Bob tun können, um dies zu ändern.

12.3 Signaturen auf Basis des diskreten Logarithmus

Neben RSA gibt es derzeit nur noch eine weitere Gruppe von Signaturverfahren, die praktische Relevanz hat: die recht vielfältige Familie der Signaturen auf Basis des diskreten Logarithmus. Man bezeichnet diese auch als **DLSS**s (Discrete Logarithm Signature Systems). Die DLSSs sind eine Weiterentwicklung des Diffie-Hellman-Schlüsselaustauschs und basieren somit auf dem Problem des diskreten Logarithmus. Dies bedeutet, dass Bobs privater Schlüssel eine natürliche Zahl x und der öffentliche Schlüssel $g^x \pmod p$ ist. Insgesamt gibt es über 13.000 Variationen der Signaturverfahren auf Basis des diskreten Logarithmus. Wir wollen

uns zwei davon anschauen: ElGamal und der Digital Signature Algorithm (DSA). Weitere DLSSs, auf die ich nicht näher eingehen werde, sind am Ende dieses Unterkapitels aufgeführt.

12.3.1 ElGamal-Verfahren

Das einfachste DLSS ist das von Taher ElGamal entwickelte **ElGamal-Verfahren** [ElGama]. Dieses wird zwar in seiner ursprünglichen Form in der Praxis so gut wie nie eingesetzt, es bildet jedoch die Basis des DSA-Verfahrens, zu dem wir später kommen.

Funktionsweise des ElGamal-Verfahrens

Für eine ElGamal-Signatur benötigen wir erst einmal eine große Primzahl p und eine Zahl g, die ein Generator von $Z(p,\cdot)$ ist (das alles kommt Ihnen hoffentlich noch von Diffie-Hellman bekannt vor). Alice, die ihre Nachrichten signieren will, benötigt einen geheimen Schlüssel $x<p$, aus dem sie den öffentlichen Schlüssel $g^x \pmod p$ berechnet. Diesen öffentlichen Schlüssel lässt Alice Bob zukommen, er benötigt ihn zum Verifizieren von Alices Signaturen. Immer wenn Alice nun eine Nachricht m unterschreiben will, berechnet sie eine Zufallszahl y, die zu p-1 relativ prim ist (also außer 1 keinen gemeinsamen Teiler hat). Aus y berechnet sie die Zahl $r=g^y \pmod p$ und stellt anschließend folgende Gleichung auf:

$$m = x \cdot r + y \cdot s \pmod{p-1}$$

Hierbei sind x, y, m und g bekannt. Die neu eingeführte Variable s ist von diesen Größen abhängig und kann von Alice durch eine algebraische Umformung berechnet werden. Die beiden Zahlen r und s bilden zusammen die digitale Signatur zur Nachricht m. Wie Bob die Unterschrift verifizieren kann, sieht man, wenn man die obige Gleichung als Exponent zur Basis g nimmt. Dabei muss der in Abschnitt 11.1.1 beschriebene Satz beachtet werden, der besagt, dass $a^{1 \pmod{\varphi(n)}} = a \pmod n$ gilt, wenn a und n natürliche Zahlen ($a<n$) sind. Aus diesem folgt, dass $a^{k \pmod{\varphi(n)}} = a^k \pmod n$ gilt (k ist eine natürliche Zahl). Nun ergibt sich:

$$m = x \cdot r + y \cdot s \pmod{p-1}$$
$$\Rightarrow g^m = g^{x \cdot r + y \cdot s \pmod{p-1}}$$
$$\Rightarrow g^m = g^{x \cdot r + y \cdot s} \pmod p \quad \text{(dies ergibt sich aus dem genannten Satz)}$$
$$\Rightarrow g^m = g^{x \cdot r} \cdot r^s \pmod p$$

In der letzten Gleichung sehen Sie, dass Bob beide Seiten berechnen kann, ohne die geheimen Zahlen x und y zu kennen. Er benötigt dazu lediglich die ihm bekannten Zahlen g, m, den öffentlichen Schlüssel g^x sowie die Variablen r und s, die ja die Signatur bilden. Genau mit dieser Gleichung verifiziert Bob die digitale

Signatur: Er berechnet die linke und die rechte Seite und vergleicht die beiden Ergebnisse. Stimmen sie überein, dann ist die Signatur echt.

Beispiel

Betrachten wir nun ein Beispiel mit kleinen Zahlen. Nehmen wir an, Alice wählt p=13 und g=7. 7 ist ein Generator von Z(13,·) und daher ein zulässiger Wert für g. p=13 und g=7 sind öffentlich bekannte Werte. Außerdem wählt Alice x=3 als privaten Schlüssel. Daraus berechnet sie ihren öffentlichen Schlüssel $g^x \pmod p = 7^3 = 5 \pmod{13}$.

Will Alice nun die Nachricht m=10 signieren, dann wählt sie einen weiteren Zufallswert. Dieser sei y=5 (5 ist relativ prim zu p-1=12). Alice erhält damit $r = g^y \pmod p = 7^5 = 11 \pmod{13}$. Damit stellt sie folgende Gleichung auf:

$$m = x \cdot r + y \cdot s \pmod{p\text{-}1}$$
$$\Leftrightarrow 10 = 3 \cdot 11 + 5 \cdot s \pmod{12}$$
$$\Leftrightarrow 10 = 9 + 5 \cdot s \pmod{12}$$
$$\Leftrightarrow s = 5 \pmod{12}$$

Die Signatur wird damit durch die Zahlen 11 und 5 gebildet. Schauen wir uns nun an, wie Bob diese Signatur verifizieren kann. Bob stellt folgende Gleichung auf:

$$g^m = g^{x \cdot r} \cdot r^s \pmod p$$
$$7^{10} = 5^{11} \cdot 11^5 \pmod{13}$$
$$4 = 8 \cdot 7 \pmod{13}$$
$$4 = 4 \pmod{13}$$

Da die Gleichung stimmt, weiß Bob, dass die Signatur echt ist.

12.3.2 DSA

Dass es so viele DLSS-Varianten gibt, liegt daran, dass man die zur Verifikation verwendeten Gleichungen, die Sie vom ElGamal-Verfahren kennen, auf vielfältige Weise variieren kann. Praktische Relevanz haben derzeit jedoch nur wenige dieser Variationen. Die wichtigste Gleichungsvariante wird in einem DLSS verwendet, das **Digital Signature Algorithm** (**DSA**) genannt wird [FIPS-186]. Wie der DSA funktioniert, ist im Digital Signature Standard (DSS) festgelegt. Der DSS ist ein Werk der NSA und wurde von der amerikanischen Standardisierungsbehörde NIST 1991 veröffentlicht sowie 1994 standardisiert. Dass statt dem damals bereits weit verbreiteten RSA ein No-Name-Verfahren zum Standard gemacht wurde, war für viele unverständlich und führte zu heftigen Diskussionen. Daher ist es auch nicht verwunderlich, dass der DSA nie die Popularität von RSA erreichte. Er ist jedoch immerhin die unumstrittene Nummer zwei.

Mathematik des DSA

Um die Funktionsweise des DSA zu verstehen, müssen wir noch einmal auf die in Abschnitt 11.1.1 behandelten mathematischen Grundlagen zurückgreifen. Dort haben Sie erfahren, dass jede Gruppe vom Typ $Z(p,\cdot)$ Untergruppen besitzt. Zu jedem Teiler von p-1 gibt es genau eine Untergruppe. Zu jeder Untergruppe gibt es mindestens einen Generator. Für uns ist nun folgende Fragestellung wichtig: Es sei g ein Generator einer Untergruppe von $Z(g,\cdot)$, b sei ein Element der von g generierten Untergruppe und es gelte $g^x=b \pmod p$. Ist unter diesen Voraussetzungen x (der diskrete Logarithmus) einfacher zu berechnen, als wenn – wie bei Diffie-Hellman und ElGamal gefordert – g ein Generator von $Z(g,\cdot)$ selbst ist? Man kann zeigen, dass die Antwort nein lautet. Die Kenntnis der generierten Untergruppe bietet keinen Vorteil bei der Berechnung des diskreten Logarithmus – lediglich eine vollständige Schlüsselsuche wird einfacher, aber das fällt nicht ins Gewicht, wenn die Untergruppe groß genug ist. Diese Tatsache wird beim DSA ausgenutzt.

Funktionsweise des DSA

Für eine DSA-Signatur benötigen Alice und Bob wiederum eine große Primzahl p. Außerdem wählt Alice eine Primzahl q, die ein Teiler von p-1 ist. q hat laut DSS-Spezifikation eine Länge von 160 Bit, allerdings wäre auch eine andere Länge sinnvoll möglich. Der Trick beim DSA gegenüber dem ElGamal-Verfahren besteht darin, dass in der Untergruppe mit q Elementen gerechnet wird (es gibt nur eine solche). Aus dieser Untergruppe wählt Alice einen Generator g. Nun berechnet Alice ihr Schlüsselpaar. Sie wählt dazu einen privaten Schlüssel $x<q$, aus dem sie den öffentlichen Schlüssel $g^x \pmod p$ berechnet. Diese Vorgehensweise entspricht der des ElGamal-Verfahrens, nur dass $x<q$ gelten muss. Den öffentlichen Schlüssel lässt Alice Bob zukommen, er benötigt ihn zum Verifizieren von Alices Unterschriften.

Immer wenn Alice eine Nachricht m unterschreiben will, berechnet sie eine Zufallszahl y, die kleiner als q ist. Aus y berechnet sie die Zahl $r=(g^y \pmod p) \pmod q$ und stellt anschließend folgende Gleichung auf:

$$m = y \cdot s - x \cdot r \pmod q$$

Hierbei sind x, y, m, r und g bekannt. Die neu eingeführte Variable s ist von diesen Größen abhängig und kann durch eine algebraische Umformung berechnet werden. Die beiden Zahlen r und s bilden zusammen die digitale Signatur zur Nachricht m. Beachten Sie, dass diese Gleichung der des ElGamal-Verfahrens ähnelt. Es wird jedoch modulo q gerechnet, um in der Untergruppe der Größe q zu bleiben. Außerdem sieht der DSA ein Minuszeichen statt einem Pluszeichen vor – dieser Unterschied hat jedoch keine tiefere Bedeutung, sondern ist eine von vielen Variationsmöglichkeiten für die Verifikationsgleichung. Die Verifikation führt Bob dann mit folgender Gleichung durch:

$$g^m = r^s / g^{x \cdot r} \pmod p$$

Stimmt diese Gleichung, dann ist die Signatur echt. Der DSS, in dem der DSA beschrieben wird, macht einige Einschränkungen bezüglich der Parameter: p muss demgemäß 512, 1.024, 2.048, 3.078, 7.680 oder 15.360 Bit lang sein. Die Länge von q muss 140, 224, 256, 384 oder 512 Bit betragen. Natürlich funktioniert das Verfahren auch mit anderen Bit-Längen.

Ein Beispiel

Betrachten wir nun noch ein Beispiel für eine DSA-Signatur. Wir nehmen an, Alice wähle p=13 und q=3. Da p und q Primzahlen sind und außerdem q=3 die Zahl p-1=12 teilt, sind die geforderten Voraussetzungen erfüllt. Nun benötigt Alice noch einen Generator g der Untergruppe von Z(13,·) mit q=3 Elementen. Wie Sie in Abschnitt 11.1.1 erfahren haben, gibt es mit den Zahlen 3 und 9 zwei infrage kommende Generatoren. Alice wählt g=9.

Nun generiert Alice ihr Schlüsselpaar. Sie wählt dazu x=1 als privaten Schlüssel. Für den öffentlichen Schlüssel g^x ergibt sich $g^x \pmod p = 9^1 = 9 \pmod{13}$. Nun signiert Alice die Nachricht m=1. Dazu wählt sie noch zufällig den Wert y=2. Damit berechnet sie $r = (g^y \pmod p) \pmod q = 9^2 = (3 \pmod{13}) \pmod 3 = 0$. Damit stellt sie folgende Gleichung auf:

$$m = y \cdot s - x \cdot r \pmod q$$
$$\Leftrightarrow 1 = 2 \cdot s - 1 \cdot 0 \pmod 3$$
$$\Leftrightarrow 1 = 2 \cdot s \pmod 3$$

Nach s aufgelöst ergibt sich der Wert s=2. Die Signatur ist damit das Zahlenpaar bestehend aus 0 und 2.

Wenn Bob diese Signatur verifizieren will, dann berechnet er:

$$g^m = r^s / g^{x \cdot r} \pmod p$$
$$\Leftrightarrow 9^1 = 0^2 / 9^0 \pmod 3$$
$$\Leftrightarrow 0 = 0 \pmod 3$$

Da die linke Seite der Gleichung mit der rechten übereinstimmt, ist die Signatur echt.

Bewertung des DSA

Der DSA hat gegenüber RSA zwei Nachteile: Zum einen ist eine DSA-Verifikation deutlich langsamer als eine RSA-Verifikation. Dies ist vor allem dann der Fall, wenn Alice und Bob einen niedrigen RSA-Modulus verwenden. Zum anderen benötigt der DSA für jede Signatur-Generierung eine Zufallszahl, die Mallory nicht erraten darf. Dies setzt voraus, dass Alice ein guter Zufallsgenerator zur Verfügung steht, was nicht selbstverständlich ist (siehe Kapitel 15). Trotz dieser Nachteile ist der DSA ein praxistaugliches Verfahren und eine interessante Alternative zum RSA.

12.3.3 Weitere DLSSs

Weitere DLSSs mit einer gewissen Bedeutung sind **Nyberg-Rueppel** [NybRue], das **GOST-Signaturverfahren** (dieses stammt aus Russland und wird im selben Standard beschrieben wie das gleichnamige symmetrische Verschlüsselungsverfahren; eine Beschreibung gibt es in [Schn96]) und **KCDSA** (*Korean Certificate-based Digital Signature Algorithm*) [BHJKKL]. Ein ISO-Standard beschreibt drei weitere Varianten [ISO9796].

12.4 Unterschiede zwischen DLSSs und RSA

RSA und DLSSs weisen einige interessante Unterschiede auf. Hier die wichtigsten:

- Aus einer RSA-Signatur lässt sich mit dem öffentlichen Schlüssel die signierte Nachricht berechnen. Durch diese Eigenschaft zählt RSA zu den **Signaturverfahren mit Message Recovery**. Diese Bezeichnung ist etwas unglücklich gewählt, da eine Verwechslungsgefahr mit dem Thema Recovery (Abschnitt 24.1.8) besteht – damit hat es aber nichts zu tun. Der DSA ist dagegen keine Signaturverfahren mit Message Recovery, da Bob aus der Signatur nicht die signierte Nachricht berechnen kann. Es gibt jedoch RSA-Varianten ohne Message Recovery sowie DLSSs mit Message Recovery.
- RSA kann gleichermaßen verschlüsseln und signieren. Ein DLSS kann dagegen nur für Signaturen eingesetzt werden. Es gibt zwar auch Verschlüsselungsverfahren auf Basis des diskreten Logarithmus, doch diese haben einen anderen Ablauf als eine DLSS-Signatur. Die Trennung zwischen Signatur- und Verschlüsselungsschlüsseln ist bei RSA daher wichtiger als bei DLSSs.
- Während die Generierung einer Signatur bei beiden Verfahren etwa gleich lange dauert, ist die Verifikation beim RSA-Verfahren etwa zehnmal schneller als bei einem DLSS. Dies gilt insbesondere dann, wenn für den Wert e beim RSA kleine Standardwerte verwendet werden.
- Die Länge des öffentlichen Schlüssels beim RSA beträgt in der Praxis meist 1.024 oder 2.048 Bit. Die jeweils gleiche Sicherheit kann bei einem DLSS mit etwas kürzeren Schlüsseln erreicht werden.
- Die gängigen DLSSs benötigen für jede Signatur eine eigene Zufallszahl, beim RSA-Verfahren muss dagegen nur der Schlüssel zufällig generiert werden. Dieser Nachteil der DLSSs ist nicht zu unterschätzen, da das Generieren von Zufallszahlen deutlich schwieriger ist, als es scheint.

Sie sehen, abgesehen von der unwesentlich kürzeren Schlüssellänge spricht scheinbar alles gegen DLSSs und für das RSA-Verfahren. Dass DLSSs trotzdem ihren Marktanteil haben, hat mehrere Gründe:

- DLSSs können mithilfe elliptischer Kurven implementiert werden, wodurch sie entscheidend schneller als RSA werden (es gibt zwar auch eine RSA-Vari-

ante auf Basis elliptischer Kurven, diese wird aber kaum eingesetzt). Mehr darüber in Abschnitt 13.1.2.

- Das RSA-Verfahren stand bis zum Jahr 2000 in den USA unter Patentschutz. Für DSA und ElGamal gilt dies nicht. Der Patentschutz ist sicher ein Grund, weshalb sich die amerikanische Standardisierungsbehörde NIST für ein DLSS und gegen RSA entschied.

13 Weitere asymmetrische Krypto-Verfahren

Die Auswahl an asymmetrischen Krypto-Verfahren ist deutlich kleiner als in der symmetrischen Kryptografie. Wer RSA und die weit verzweigte Familie der Krypto-Systeme auf Basis des diskreten Logarithmus kennt, wird in der Praxis nur selten auf Unbekanntes stoßen. Dennoch gibt es auch in der zweiten Reihe einige hochinteressante asymmetrische Verfahren, die es zu betrachten lohnt. Dies gilt vor allem für die sogenannten Krypto-Systeme auf Basis elliptischer Kurven (ECC-Verfahren), die derzeit einen immer größer werdenden Teil des Marktes erobern. Alle weiteren asymmetrischen Verfahren spielen zwar in der Praxis bisher keine nennenswerte Rolle, bieten jedoch teilweise interessante Perspektiven.

Die folgenden Betrachtungen sind die mathematisch anspruchsvollsten in diesem Buch. Ich habe mich bemüht, die jeweiligen Verfahren dennoch einigermaßen verständlich zu erklären. Dies war jedoch nur dadurch möglich, dass ich einige Aspekte vereinfacht oder schlicht weggelassen habe. Ein Mathematiker wird die folgenden Ausführungen daher vermutlich oberflächlich finden, doch

dieses Buch ist nicht in erster Linie für Mathematiker geschrieben. Wer mehr Tiefgang sucht, den verweise ich auf die jeweils angegebene Literatur.

13.1 Krypto-Systeme auf Basis elliptischer Kurven

Der größte Nachteil von asymmetrischen Verfahren ist, dass sie im Vergleich zu den symmetrischen recht aufwendig sind. Daran wird sich zwar so schnell nichts ändern, doch es gibt immerhin Möglichkeiten, den Vorsprung der symmetrischen Verfahren etwas zu verkleinern. Ein Ansatz dazu sind die **Krypto-Systeme auf Basis elliptischer Kurven**. Diese werden auch als **ECC-Verfahren** bezeichnet, wobei ECC für *Elliptic Curve Cryptography* steht. ECC-Verfahren sind keine eigenständigen kryptografischen Algorithmen. Vielmehr handelt es sich dabei um Vertreter der Ihnen bereits bekannten Verfahren auf Basis des diskreten Logarithmus (etwa Diffie-Hellman, MQV oder DSA), die jedoch auf eine besondere Weise implementiert werden. Um das vorliegende Kapitel verstehen zu können, sollten sie daher mit dem diskreten Logarithmus und den darauf basierenden Verfahren vertraut sein.

13.1.1 Mathematische Grundlagen

Wie Sie aus Abschnitt 11.1.1 bereits wissen, ist ein Körper eine Menge, auf der zwei Verknüpfungen definiert sind. Diese werden meist als Addition und Multiplikation bezeichnet. Ein Körper ist bezüglich der Addition eine Gruppe, außerdem bildet er ohne die Null auch eine Gruppe bezüglich der Multiplikation. Der bekannteste Körper ist die Menge der reellen Zahlen, auf der Addition und Multiplikation auf bekannte Weise definiert sind. Wie Ihnen ebenfalls bereits bekannt ist, gibt es zu jeder Primzahl p einen Körper mit p Elementen, der GF(p) genannt wird. Dabei fungiert die Modulo-Addition als Addition und die Modulo-Multiplikation als Multiplikation.

Eine weitere beweisbare Tatsache habe ich jedoch noch nicht erwähnt: Für jede Primzahl p und jede natürliche Zahl n gibt es genau einen Körper mit p^n Elementen. Ist n=1, dann haben wir den Spezialfall, in dem die Modulo-Rechnung angewendet werden kann. Im Folgenden werden wir nur die beiden Spezialfälle n=1 und p=2 betrachten. Es geht also um GF(p) und GF(2^n). Mit diesen Galois-Feldern lassen sich ECC-Verfahren am besten realisieren. Mit GF(m) bezeichne ich im Folgenden einen beliebigen Körper der Form GF(p) oder GF(2^n).

Rechnen in GF (2^n)

Wie in GF(p) gerechnet wird, wissen Sie bereits. In GF(2^n) ist die Sache etwas komplizierter. Die Elemente in GF(2^n) werden nicht als natürliche Zahl, sondern als Binärzahl oder als Polynom mit den Koeffizienten 0 und 1 geschrieben. Es folgen einige Beispiele aus GF(2^4):

$$0101 \equiv x^2+1$$
$$1111 \equiv x^3+x^2+x+1$$
$$1010 \equiv x^3+x$$

Die Addition in GF(2^n) ist als bitweise Exklusiv-oder-Verknüpfung zweier Binärzahlen definiert. Bei der Polynomschreibweise entspricht dies einer Polynomaddition mit Koeffizienten aus GF(2). Hier einige Beispiele aus GF(2^4):

$$1001+1111=0110 \equiv (x^3+1)+(x^3+x^2+x+1)=x^2+x$$
$$1000+0001=1001 \equiv (x^3)+(1)=x^3+1$$

Mit Polynom ist im Folgenden stets ein Polynom mit Koeffizienten aus GF(2) gemeint. Die Multiplikation in GF(2^n) ist als Polynommultiplikation modulo einem irreduziblen Polynom vom Grad n definiert. Irreduzibel ist ein Polynom dann, wenn es sich nicht als Produkt aus Polynomen kleineren Grades darstellen lässt. Es gibt verschiedene Algorithmen, mit denen festgestellt werden kann, ob ein Polynom irreduzibel ist. Bei unserer Betrachtung genügt es jedoch, ein Polynom des gewünschten Grades einer Tabelle zu entnehmen. Es gibt zu jedem Grad $n>1$ irreduzible Polynome. In den folgenden Beispielen wird das irreduzible Polynom x^4+x+1 verwendet, um die Multiplikation in GF(2^4) zu definieren:

$$(x^3+1)\cdot(x^3+x^2+1)=x^6+x^5+x^2+1=x^3+x^2+x+1 \pmod{x^4+x+1},$$
daher $1001 \cdot 1101 = 1111$

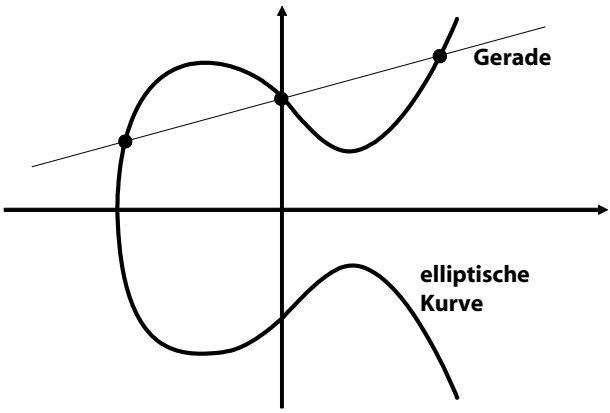


Abb. 13–1 *Zählt man Berührpunkte doppelt und nimmt man einen Punkt im Unendlichen hinzu, dann gilt: Eine Gerade schneidet eine elliptische Kurve immer in genau drei Punkten.*

Elliptische Kurven

Eine elliptische Kurve ist definiert als eine Kurve auf einem Körper, die folgende Gleichung erfüllt:

$$y^2+a_1xy+a_2y = x^3+a_3x^2+a_4x+a_5$$

Mit hinzugenommen wird zu dieser Definition ein Punkt, der im Unendlichen liegt und der als *0* bezeichnet wird (nicht zu verwechseln mit dem Nullpunkt des Koordinatensystems). Man kann zeigen, dass elliptische Kurven (und nur diese) die folgende interessante Eigenschaft haben: Schneidet eine Gerade eine solche Kurve, dann gibt es genau drei Schnittpunkte. Dabei treten folgende Fälle auf (siehe auch Abbildung 13–1):

1. Bei einer Geraden, die parallel zur *y*-Achse verläuft, ist einer der drei Schnittpunkte *0*.
2. Bei einer Geraden, welche die Kurve berührt, wird der Berührpunkt als doppelter Schnittpunkt gezählt.
3. Bei allen anderen Geraden sind die drei Schnittpunkte offensichtlich.

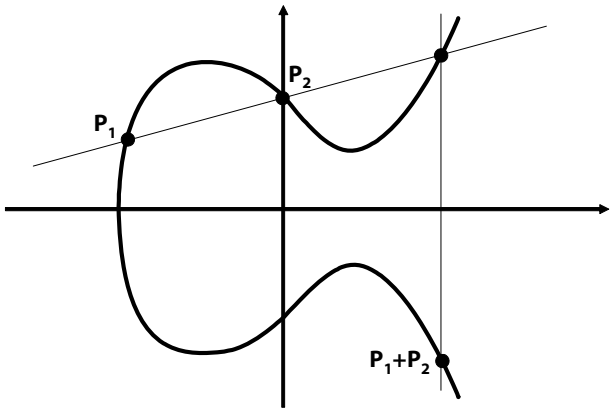


Abb. 13–2 *Zwei Punkte einer elliptischen Kurve werden addiert, indem eine Gerade durch sie gezogen wird. Das Ergebnis der Addition ist die Spiegelung des dritten Schnittpunkts der Gerade an der x-Achse.*

Durch diese Eigenschaften lässt sich mithilfe elliptischer Kurven eine Gruppe definieren, die $E(K)$ genannt wird. Deren Elemente sind die Punkte einer elliptischen Kurve (inklusive *0*). Die Addition ist wie folgt definiert: Zwei Punkte werden addiert, indem eine Gerade durch sie gelegt wird (sind die beiden Punkte gleich, dann ist es die Tangente). Gemäß obiger Eigenschaft muss es nun einen dritten Punkt auf der Kurve geben, den diese Gerade schneidet. Die Spiegelung dieses Punktes an der *x*-Achse ist das Resultat der Addition, wobei die Spiegelung von *0* wiederum *0* ist. *0* ist das neutrale Element der Gruppe: Es gilt $P+0 = 0+P = P$

für alle Punkte P der elliptischen Kurve. Das inverse Element eines Punktes erhält man, indem man an ihn eine Gerade anlegt, die parallel zur y-Achse verläuft. Ist diese Gerade eine Tangente, dann ist der Punkt selbst sein inverses Element. Ist die Gerade keine Tangente, so gibt es nach obigen Eigenschaften genau einen weiteren Schnittpunkt. Dieser ist das inverse Element.

In der Kryptografie spielen nur die elliptischen Kurven der Form $E(\mathrm{GF}(m))$ (also $E(\mathrm{GF}(p))$ und $E(\mathrm{GF}(2^n))$) eine Rolle, obwohl alle der im Folgenden genannten Verfahren auch bei Verwendung von $E(\mathrm{GF}(p^n))$ funktionieren.

13.1.2 ECC-Verfahren

Die Verknüpfung einer Gruppe $E(\mathrm{GF}(m))$ wird aus historischen Gründen »Addition« genannt. Wie erwähnt, wäre auch die Bezeichnung »Multiplikation« möglich, weshalb für die mehrfache Addition von Punkten aus $E(\mathrm{GF}(m))$ der Begriff Potenzfunktion gerechtfertigt ist. Die Umkehrung dieser Rechenart kann aus den gleichen Gründen als Logarithmus bezeichnet werden.

Folgende Eigenschaft von $E(\mathrm{GF}(m))$ ist für die Kryptografie von zentraler Bedeutung: Für die Berechnung einer Potenzfunktion existieren effektive Algorithmen; für die Berechnung des Logarithmus dagegen nicht. Die Folge ist, dass Alice und Bob alle kryptografischen Verfahren, die auf dem diskreten Logarithmus beruhen, auch mithilfe von $E(\mathrm{GF}(m))$ einsetzen können. Ein Krypto-System auf Basis elliptischer Kurven ist somit ein asymmetrisches Verfahren auf Basis des diskreten Logarithmus, bei dem statt Rechenoperationen in $\mathrm{GF}(p)$ Rechenoperationen in $E(\mathrm{GF}(m))$ verwendet werden. Statt auf einen Modulus müssen sich Alice und Bob hierbei auf einen Körper $\mathrm{GF}(m)$ und eine darauf aufbauende Gruppe $E(\mathrm{GF}(m))$ – und damit auf eine bestimmte elliptische Kurve – verständigen. Alle Parameter, die im Exponenten stehen, sind (wie bisher) natürliche Zahlen, während die Basis einer Potenz ein Element von $\mathrm{E}(\mathrm{GF}(m))$ ist.

Die Addition zweier Punkte aus $E(\mathrm{GF}(m))$ setzt sich aus mehreren Rechenoperationen in $\mathrm{GF}(m)$ zusammen. Damit ist eine Exponentiation über $\mathrm{GF}(m)$ weniger aufwendig als eine Exponentiation über $E(\mathrm{GF}(m))$. Die zum Berechnen des diskreten Logarithmus über $\mathrm{GF}(p)$ bekannten Verfahren sind jedoch wesentlich effektiver als diejenigen, die den Logarithmus über elliptischen Kurven berechnen. Der zentrale Vorteil bei der Verwendung von $E(\mathrm{GF}(m))$ ist daher, dass Alice und Bob bei gleicher Sicherheit eine Gruppe kleinerer Kardinalität verwenden können. Dies hat kürzere Schlüssellängen, kürzere Signaturen und kürzere Rechenzeiten zur Folge.

Die Komplexität des diskreten Logarithmus in $E(\mathrm{GF}(m))$ nimmt mit m linear zu, während die Komplexität in $\mathrm{GF}(m)$ nur logarithmisch ansteigt. Eine Schlüssellänge von 1.024 Bit kann bei einem Krypto-System auf Basis des diskreten Logarithmus durch die Verwendung elliptischer Kurven auf 200 Bit verkürzt werden, ohne dass dabei Sicherheitseinbußen entstehen. Für die Einsparung an Rechenzeit wird meist der Faktor zehn genannt.

Gegenüber dem RSA-Verfahren haben ECC-Verfahren in puncto Schlüssellänge noch größere Vorteile als DL-Verfahren ohne elliptische Kurven. Einer RSA-Schlüssellänge von 1.024 Bit entspricht eine ECC-Schlüssellänge von 160 bei vergleichbarer Sicherheit. Auch hier ist ein ECC-Verfahren etwa um den Faktor zehn schneller. Zudem ist die resultierende Signatur (oder der Geheimtext) kürzer.

Durch diese Ersparnis an Rechenzeit und Bandbreite sind ECC-Verfahren besonders beim Einsatz von Smartcards interessant (siehe Abschnitt 23.2). Sollen Verfahren wie RSA oder DL-Verfahren (ohne elliptische Kurven) auf einer Smartcard realisiert werden, so ist dies oft nur durch den Einsatz eines Coprozessors möglich. Ein solcher verursacht jedoch zusätzliche Kosten, was bei den hohen Stückzahlen, in denen Smartcards meist ausgegeben werden, erheblich ins Gewicht fällt. ECC-Verfahren sind dagegen auch ohne Coprozessor realisierbar, ohne dass die Performanz dabei zu sehr leidet.

13.1.3 Die wichtigsten ECC-Verfahren

Da es sich bei den ECC-Verfahren im Wesentlichen um Algorithmen handelt, deren Funktionsweise Sie bereits aus früheren Kapiteln kennen (nur dass dort in GF(p) gerechnet wurde), bietet ein Blick auf die wichtigsten Vertreter dieser Gattung wenig Überraschendes. Hier ist eine Liste der bedeutendsten ECC-Verfahren:

- **ECDH**: Diese Abkürzung für **Elliptic Curve Diffie-Hellman** bezeichnet das Diffie-Hellman-Verfahren auf Basis elliptischer Kurven. Alice und Bob einigen sich hierbei statt auf p auf eine Gruppe $E(\mathrm{GF}(m))$ und damit auf eine elliptische Kurve. Die öffentlichen Schlüssel x und y sind weiterhin natürliche Zahlen, während g ein Element von $E(\mathrm{GF}(m))$ ist, auf das sich Alice und Bob verständigen.
- **ECMQV**: Dies ist die ECC-Version von MQV.
- **ECDSA**: Auch der DSA lässt sich mit elliptischen Kurven realisieren. Man spricht in diesem Fall vom ECDSA (**Elliptic Curve DSA**). Das einzige Problem bei der Übertragung des DSA-Verfahrens in die ECC-Welt liegt darin, dass die Zahl g^y, die ein Punkt auf einer elliptischen Kurve ist, auch im Exponenten verwendet wird – dort muss jedoch eine natürliche Zahl stehen. Alice kann dieses Problem lösen, indem sie im Exponenten eine aus g^y abgeleitete natürliche Zahl einsetzt. In der Regel wird die x-Koordinate des Punktes g^y dazu verwendet.
- **EC-NR**: Dies ist die ECC-Variante des Nyberg-Rueppel-Signaturverfahrens.
- **EC-KCDSA**: Dies ist die ECC-Variante des aus Korea stammenden Signaturverfahrens KCDSA.

- **ECGDSA:** Dieses ECC-Signaturverfahren wurde von der Firma Siemens vorgeschlagen. Sein vollständiger Name lautet *Elliptic Curve German Digital Sgnature Standard*. Die Erfinder heißen Erwin Hess, Marcus Schafheutle und Pascale Serf [HeScSe]. ECGDSA wurde von Anfang an als ECC-Verfahren konzipiert, wodurch es kein Gegenstück ohne elliptische Kurven gibt.

Für den praktischen Einsatz der genannten Verfahren ist natürlich eine Standardisierung notwendig. Der P1363-Standard spezifiziert insgesamt fünf ECC-Verfahren [P1363, P1363a]. Dabei handelt es sich um zwei ECDH-Varianten sowie um je eine Version von ECMQV, ECDSA und EC-NR. Zu erwähnen ist außerdem der Standard **PKCS#13**, der sich ebenfalls mit elliptischen Kurven beschäftigt [PKCS#13]. Dieser noch in der Entstehung befindliche Standard wird sich an P1363 anlehnen und dabei voraussichtlich etwas restriktiver sein, weshalb man von einem P1363-Profil reden kann. Es gibt noch weitere Standards, in denen ECC-Verfahren spezifiziert werden. Dazu gehören ANSI X.9 und mehrere ISO-Standards.

Wenn Sie mehr über Krypto-Systeme auf Basis elliptischer Kurven wissen wollen, dann stehen Ihnen mehrere Bücher zur Verfügung, die sich nur mit diesem Thema beschäftigen. [HaMeVa] und [CoFrAv] sind meiner Meinung nach die besten davon, etwas kostengünstiger ist das deutschsprachige [Werner].

13.2 Weitere asymmetrische Verfahren

Kommen wir nun zu einigen asymmetrischen Krypto-Verfahren, die in der Praxis bisher keine nennenswerte Rolle spielen. Die komplexe Mathematik dieser Algorithmen kann ich leider nur sehr oberflächlich behandeln.

13.2.1 NTRU

NTRU ist ein asymmetrisches Krypto-Verfahren, das seit Jahren von einer gleichnamigen US-Firma propagiert wird. Es wurde von den Kryptografen Jeffrey Hoffstein, Jill Pipher und Joseph H. Silverman erfunden [HoPiSi]. Praxistaugliche NTRU-Implementierungen sind am Markt verfügbar (wenn auch kaum verbreitet), was in der asymmetrischen Kryptografie ansonsten nur für RSA, die DL-Verfahren und die ECC-Verfahren gilt. Der Name des Verfahrens leitet sich von »n-truncated« (»n-abgeschnitten«) ab und bezieht sich auf die Polynome, die darin zum Einsatz kommen. Es gibt sowohl *NTRU-Encrypt* (für die Verschlüsselung) als auch *NTRU-Sign* (für digitale Signaturen). Die Unterschiede zwischen diesen beiden Varianten sind so groß, dass man von zwei verschiedenen Verfahren reden kann. Ich will mich im Folgenden auf die Verschlüsselung beschränken.

Mathematische Grundlagen

Die Funktionsweise von NTRU lässt sich mithilfe mathematischer Strukturen erklären, die man als *Gitter* bezeichnet. Die meisten Beschreibungen des Verfahrens beginnen daher mit einer Einführung in die Gittertheorie. Ich folge dagegen an dieser Stelle der originalen Beschreibung von NTRU aus der Arbeit von Hoffstein, Pipher und Silverman, die ohne die Betrachtung von Gittern auskommt.

In NTRU spielen Polynome der Form $f(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+...+a_1x+a_0$ eine Rolle. Die Koeffizienten a_i sind natürliche Zahlen, mit denen modulogerechnet wird. Werden zwei Polynome miteinander multipliziert und entsteht dabei ein Polynom vom Grad n oder größer, dann erfolgt eine Reduzierung (»Abschneiden«) in Form einer Teilung durch das Polynom x^n-1. Dieses Vorgehen, das dem Verfahren seinen Namen gab, lässt sich vereinfacht so ausdrücken: Durch eine Multiplikation entsteht immer ein Polynom vom Grad n-1 oder niedriger. Außerdem benötigen wir für die Nutzung von NTRU noch eine kleine natürliche Zahl p und eine deutlich größere natürliche Zahl q. Typische Werte sind n=503, p=3 und q=256.

Neben der Multiplikation zweier Polynome gibt es auch eine Teilung (Polynomdivision). Diese funktioniert jedoch nur, wenn zum Polynom f, das als Teiler dient, ein inverses Polynom f^{-1} existiert, für das gilt: $f \cdot f^{-1} = 1$. Diese Bedingung ist zwar nicht immer erfüllt, doch es gibt für unsere Zwecke ausreichend viele invertierbare Polynome, die zudem leicht zu ermitteln sind.

Funktionsweise von NTRU

Der öffentliche NTRU-Schlüssel wird wie folgt berechnet: Alice wählt zwei Polynome f und g, wobei f invertierbar sein muss, und berechnet daraus das Polynom h nach folgender Formel:

$$h = g/f \pmod q$$

h ist Alices öffentlicher Schlüssel, f (bzw. f^{-1}) der private. Man kann zeigen, dass unter den gegebenen Rahmenbedingungen g und f eindeutig bestimmt sind, wenn h und q gegeben sind (wäre es nicht so, dann gäbe es zu einem öffentlichen Schlüssel mehrere private, was nicht erwünscht ist). Die Formel, nach der h berechnet wird (Teilen zweier Polynome modulo q), gilt als Einwegfunktion. Wäre dies nicht der Fall, dann wäre das Verfahren leicht zu knacken. Wie bei anderen asymmetrischen Verfahren ist diese Einwegfunktion-Eigenschaft jedoch nicht bewiesen, sondern nur eine begründete Vermutung.

Zur Verschlüsselung benötigt Bob zunächst Alices öffentlichen Schlüssel h. Seine Nachricht m kodiert er als Polynom. Außerdem generiert er ein zufälliges Polynom b, das als *Blendpolynom* bezeichnet wird (einen zufälligen Bestandteil gibt es auch bei DSA und ElGamal, jedoch nicht bei RSA). Die Verschlüsselung erfolgt nach folgender Formel:

$$e = p \cdot b \cdot h + m \pmod{q}$$

Die Entschlüsselung erledigt Alice in zwei Schritten. Zunächst berechnet sie das Zwischenergebnis a wie folgt

$$a = f \cdot e \pmod{q}$$

Nun erhält sie den Klartext nach folgender Formel:

$$m = a/f \pmod{p}$$

Warum die Entschlüsselung tatsächlich den Klartext ergibt, soll uns an dieser Stelle nicht interessieren. Wichtig ist allerdings ein Blick auf die Verschlüsselungsfunktion. Diese sieht vor, dass (unter geeigneter Einbeziehung der Parameter p und q) der Klartext zu zwei Polynomen gezählt wird, nachdem diese miteinander multipliziert worden sind. Nur eines der Polynome ist bekannt. Diese Operation ist eine Falltürfunktion. Im Gegensatz zu RSA gibt es zwei Informationen zum »Öffnen« der Falltür: das geheime Blendpolynom b und den privaten Schlüssel f.

Es bietet sich an, an dieser Stelle einen tabellarischen Vergleich zwischen NTRU und dem RSA-Verfahren vorzunehmen:

	NTRU	RSA
vertrauliche Parameter für die Schlüsselgenerierung	Polynome f, g	Primzahlen p, q
vertraulicher Parameter, der zur schnelleren Entschlüsselung aufbewahrt wird	f^{-1}	d (natürliche Zahl)
privater Schlüssel	f	p, q
öffentliche Parameter	p, q (natürliche Zahlen)	e (natürliche Zahl)
öffentlicher Schlüssel	$h = g/f \pmod{q}$	$n = p \cdot q$

Bewertung von NTRU

Die Schlüssellängen von NTRU entsprechen bei gleicher Sicherheitsstufe etwa denen von RSA und Diffie-Hellman. Meist wird ein Wert zwischen 1.000 und 2.000 Bit eingesetzt. Der große Vorteil des Verfahrens liegt in der sehr hohen Ver- und Entschlüsselungsgeschwindigkeit. Nach Angaben der Entwickler kann NTRU im Vergleich zu RSA eine zehn- bis 500-fache Performanz erreichen. Damit liegt das Verfahren in einer Größenordnung, die sonst nur symmetrische Algorithmen erreichen.

Trotz dieser Vorteile hat NTRU bisher nicht den großen Durchbruch geschafft. Auch der missionarische Eifer des gleichnamigen Unternehmens konnte dies nicht ändern. Zu den Nachteilen von NTRU gehört zweifellos die im Vergleich zu RSA größere Komplexität (diese wird noch größer, wenn man NTRU-Sign einbezieht). Darüber hinaus ist bei NTRU die richtige Wahl der Parameter noch wichtiger und komplizierter als bei RSA. Ein weiteres Problem besteht darin, dass NTRU auch bei korrektem Einsatz in manchen Fällen falsche Ergebnisse liefert.

Auch die Sicherheit von NTRU ist nicht über jeden Zweifel erhaben. Es sind mehrere Angriffe auf das Verfahren bekannt, die zwar alle durch geeignete Parameterwahl abzuwehren sind, doch ein ungutes Gefühl bleibt. So muss das Fazit lauten: NTRU ist ein äußerst interessantes Verfahren, das für den massenweisen Einsatz in der Praxis geeignet sein könnte. Bevor es so weit kommt, ist jedoch noch einiges an Analysearbeit zu leisten.

13.2.2 XTR

XTR ist neben den ECC-Verfahren eine weitere Familie von asymmetrischen Krypto-Systemen, die auf einer speziellen Nutzung des diskreten Logarithmus basiert. Streng genommen handelt es sich dabei nicht einmal um eigenständige Verfahren, sondern um eine Verbesserung bestehender Algorithmen. Der Name XTR steht für die Buchstabenkombination »ECSTR«, die wiederum eine Abkürzung für »Efficient and Compact Subgroup Trace Representation« ist. Entwickelt wurde XTR von Arjen Lenstra und Eric R. Verheul [LenVer]. Die Grundlage von XTR bilden DL-Verfahren wie Diffie-Hellman oder DSA, die mit Parametern aus dem endlichen Körper GF(p^6) implementiert werden (p ist eine Primzahl). Insbesondere wird auch der diskrete Logarithmus in GF(p^6) genutzt. Der Trick dabei ist, dass XTR eine besonders effektive Art des Rechnens in diesem Körper nutzt. Sowohl die Darstellung der Parameter als auch die Durchführung der Rechenoperationen sind dadurch deutlich simpler als in der herkömmlichen Form. Der besagte Trick sieht vor, dass ein großer Teil der Berechnungen statt in GF(p^6) in GF(p^2) erledigt werden kann.

Praktisch alle Krypto-Verfahren, die auf dem diskreten Logarithmus basieren, lassen sich mit XTR realisieren. Dementsprechend gibt es einen XTR-Diffie-Hellman, ein XTR-MQV und einen XTR-DSA. Nach aktuellem Stand der Forschung ist XTR genauso sicher wie die jeweiligen Krypto-Systeme in herkömmlicher Form. Die Schlüssellänge und die Verschlüsselungsgeschwindigkeit sind mit ECC-Verfahren vergleichbar und damit deutlich besser als etwa beim RSA-Verfahren. Auch die Schlüsselgenerierung ist sehr performant. Da XTR bisher kaum verwendet wird, ist es schwierig, eine belastbare Aussage zur Sicherheit und zur Praxistauglichkeit zu machen. In jedem Fall zählt diese Familie von Verfahren zu den interessantesten aymmetrischen Krypto-Systemen aus der zweiten Reihe und könnte in den nächsten Jahren an Bedeutung gewinnen.

13.2.3 Krypto-Systeme auf Basis hyperelliptischer Kurven

Hyperelliptische Kurven sind eine Verallgemeinerung der elliptischen Kurven. Während Letztere durch die Gleichung $y^2+a_1xy+a_2y = x^3+a_3x^2+a_4x+a_5$ definiert sind, können bei hyperelliptischen Kurven zusätzliche Terme mit höheren Potenzen vorkommen. Um es nicht zu kompliziert zu machen, beschränkt man sich in

der Kryptografie auf solche Fälle, in denen die Formel wie folgt aufgebaut ist (die Zahl *n* muss ungerade und größer als 3 sein, meist ist ihr Wert 5 oder 7):

$$y^2 = x^n + a_{n-1}x^{n-1} + \ldots + a_1x + a_0$$

Mithilfe von hyperelliptischen Kurven lassen sich genauso eine Potenzfunktion und ein Logarithmus definieren wie mit elliptischen Kurven. Dementsprechend funktionieren alle Verfahren auf Basis des diskreten Logarithmus auch mit hyperelliptischen Kurven. Man spricht in diesem Zusammenhang von **Krypto-Systemen auf Basis hyperelliptischer Kurven** oder von **HEC-Verfahren**. Die Idee dazu stammt aus den achtziger Jahren und ist nur unwesentlich jünger als die Einführung elliptischer Kurven in die Kryptografie.

HEC-Verfahren sind noch komplexer als ECC-Verfahren. Es erscheint jedoch möglich, mit derartigen Algorithmen bei gleicher Sicherheit kürzere Schlüssellängen zu realisieren, um so auf kürzere Rechenzeiten zu kommen. In absehbarer Zeit werden HEC-Verfahren jedoch kaum zur Praxisreife gelangen. Bevor wir uns um die hyperelliptischen Kurven kümmern, sollten erst einmal die elliptischen im Vordergrund stehen.

13.2.4 HFE

HFE steht für **Hidden Field Equations** und bezeichnet eine Familie asymmetrischer Krypto-Verfahren, die einige interessante Eigenschaften haben. Es gibt mehrere Vertreter, die nach aktuellem Stand der Forschung für den Praxiseinsatz geeignet sind.

Mathematische Grundlagen

HFE arbeitet auf Basis von endlichen Körpern, die mit $GF(p^{m \cdot n})$ notiert werden. (p ist eine Primzahl, m und n sind natürliche Zahlen). Der Einfachheit halber beschränke ich mich im Folgenden auf p=2 und m=1, damit wir im endlichen Körper $GF(2^n)$ rechnen können. n kann beispielsweise den Wert 128 annehmen, wodurch wir es mit 128-Bit-Werten zu tun haben. In HFE spielen vor allem Polynome über $GF(2^n)$ eine Rolle (also Polynome, deren Koeffizienten und Variablen aus $GF(2^n)$ stammen), deren Grad kleiner als n ist.

Im Mittelpunkt von HFE steht die mathematisch beweisbare Tatsache, dass sich bestimmte Polynome über $GF(2^n)$ in quadratische Gleichungssysteme umwandeln lassen. So ist beispielsweise die Polynomfunktion $f(x)=x^5+x^3+x$ äquivalent zu folgendem Gleichungssystem (wie diese Umwandlung erfolgt, soll uns an dieser Stelle nicht interessieren):

$$f_1(x_0,x_1,x_2) = x_2+x_2 \cdot x_1+x_2 \cdot x_0+x_1$$
$$f_2(x_0,x_1,x_2) = x_2 \cdot x_1+x_1 \cdot x_0+x_2$$
$$f_3(x_0,x_1,x_2) = x_0+x_2+x_1 \cdot x_0+x_2 \cdot x_0$$

Beachten Sie, dass im Gleichungssystem immer nur zwei x_i-Variablen miteinander multipliziert werden (es kommt also kein Ausdruck wie $x_1 \cdot x_2 \cdot x_3$ vor). Deshalb gilt das Gleichungssystem als quadratisch. Diese Eigenschaft hat den Vorteil, dass sich die jeweiligen Formeln schnell berechnen lassen. Quadratische Gleichungssysteme spielen im Übrigen auch bei der quadratischen Kryptoanalyse eine Rolle, die Sie aus Abschnitt 7.3.4 kennen.

Nun benötigen wir noch ein weiteres Werkzeug: die *affine Abbildung*. Ist x eine Variable mit Werten aus GF(2^n), M eine $n \times n$-Matrix und C eine Konstante aus GF(2^n), dann ist eine affine Abbildung A wie folgt definiert:

$$A(x) = Mx + C$$

Mit anderen Worten: Eine affine Abbildung multipliziert eine Matrix mit einem Vektor und zählt einen weiteren Vektor dazu. Wichtig für uns ist folgende Tatsache, die für jede affine Abbildung $A(x)$ gültig ist: Ist $f(x)$ ein Polynom, dann sind sowohl $A(f(x))$ als auch $f(A(x))$ ebenfalls Polynome. Außerdem gilt: Ist $f(x)$ als quadratisches Gleichungssystem darstellbar, dann geht diese Eigenschaft durch eine affine Abbildung nicht verloren.

Das Verfahren

Um ein HFE-Schlüsselpaar zu generieren, benötigt Alice zunächst ein Polynom f über GF(2^n). f muss als quadratisches Gleichungssystem darstellbar sein. Außerdem braucht Alice zwei affine Abbildungen S und T, die ebenfalls auf GF(2^n) arbeiten. Die Matrizen und Konstanten dieser affinen Abbildungen sind zufällig gewählt. Der öffentliche Schlüssel des HFE-Verfahrens ist das Polynom $g(x)=T(f(S(x)))$ als quadratisches Gleichungssystem dargestellt. Der private Schlüssel besteht aus f sowie den beiden affinen Abbildungen S und T. Man kann zeigen, dass es zu einem $g(x)$ jeweils nur einen passenden Wert von S, T und f gibt (wäre es anders, dann gäbe es mehrere private Schlüssel zu einem öffentlichen).

Dass Mallory den privaten HFE-Schlüssel nicht aus dem öffentlichen berechnen kann, liegt daran, dass das Umwandeln von f mithilfe von S und T in das besagte Gleichungssystem, das mit g äquivalent ist, eine Einwegfunktion ist. Wie bei anderen asymmetrischen Verfahren ist dies allerdings nicht bewiesen, sondern nur eine Vermutung.

Das Verschlüsseln geschieht wie folgt: Bob nimmt einen n-Bit-Wert als Klartext w und berechnet daraus den Schlüsseltext c mit der einfachen Formel $c = g(w)$. Dieser Vorgang ist im Vergleich zu anderen asymmetrischen Verfahren ausgesprochen performant zu realisieren – es kommen ja nur Quadrierungen und Additionen zum Einsatz – und erreicht nahezu die Geschwindigkeit symmetrischer Verfahren.

Zur Entschlüsselung berechnet Alice $w = S^{-1}(f^{-1}(T^{-1}))$. Dieser Vorgang dauert länger als das Verschlüsseln und benötigt zudem etwas Vorarbeit: Alice muss aus f die Invertierung f^{-1} berechnen, was auf einem PC einige Sekunden dauern kann.

Wenn sie dies einmal gemacht hat, kann sie f^{-1} für spätere Entschlüsselungen aufbewahren. Die Invertierung von S und T ist dagegen trivial. HFE funktioniert, weil g in der Darstellung als quadratisches Gleichungssystem eine **Falltürfunktion** ist. Die zum »Öffnen« der Falltür benötigte Information ist das Polynom f.

HFE lässt sich auch für digitale Signaturen nutzen. Die entsprechenden Varianten davon heißen **Quartz**, **FLASH** und **SFLASH**. Das wichtigste davon ist SFLASH [CoGoPa].

Bewertung von HFE

Der wichtigste Vorteil von HFE ist die sehr hohe Verschlüsselungsgeschwindigkeit. Die Entschlüsselung ist dagegen deutlich langsamer. Auch die zugehörigen Signaturverfahren können eine hohe Performanz erreichen. Von Vorteil ist außerdem die geringe Größe der Signaturen, die bereits ab etwa 200 Bit ausreichend sicher sind. Der entscheidende Nachteil von HFE besteht dagegen darin, dass die öffentlichen Schlüssel sehr lang sind – über 100 Kilobyte sind bei hoher Sicherheitsstufe durchaus realistisch.

13.2.5 Weitere asymmetrische Verfahren

Hier sind noch einige weitere asymmetrische Krypto-Verfahren:

- **Cailey-Purser**: Von der Erfindung eines neuen Public-Key-Verfahrens nehmen die Massenmedien im Normalfall so gut wie keine Notiz. Anders war dies bei einem 1998 entstandenen Algorithmus namens Cailey-Purser [Flanne]. Dies lag jedoch weniger am Verfahren selbst als an dessen Erfinderin. Diese heißt Sarah Flannery und war bei Bekanntwerden des Verfahrens eine 15-jährige Schülerin aus Dublin. Das von Flannery entdeckte Verfahren beruht wie RSA auf dem Faktorisierungsproblem, ist aber deutlich schneller. Eine größere Verbreitung wird es kaum finden, denn es wurde inzwischen von der Erfinderin selbst gebrochen. An der außergewöhnlichen Leistung der jungen Kryptografin ändert das natürlich wenig. Leider hat man von der jungen Irin in der Zwischenzeit nichts mehr gehört.
- **McEliece**: Das asymmetrische Verschlüsselungsverfahren von McEliece stammt aus dem Jahr 1978 und gehört damit zu den ältesten seiner Art [McElie]. Deutlich jünger ist ein dazugehörendes Signaturverfahren [CoFiSe]. Der McEliece-Algorithmus basiert auf Fehlerkorrekturcodes und gilt als sicher. Leider sind die öffentlichen Schlüssel mit einer Größe von einem Megabyte oder mehr viel zu groß für eine praktische Nutzung.
- **Knapsack**: Dieses Verfahren gehört ebenfalls zu den ältesten asymmetrischen Verfahren. Es ist vergleichsweise einfach zu verstehen. Es gibt mehrere Knapsack-Varianten, die meisten wurden gebrochen. [Esslin] gibt eine gute Einführung.

Es gibt noch zahlreiche weitere asymmetrische Krypto-Verfahren, die in der Fachliteratur vorgeschlagen wurden. Sie alle sind jedoch noch zu wenig untersucht, unpraktikabel oder unsicher.

14 Kryptografische Hashfunktionen

Aus dem Kapitel über asymmetrische Verschlüsselung ist Ihnen sicherlich noch folgende Tatsache bekannt: Asymmetrische Verschlüsselungsverfahren wie RSA sind deutlich langsamer als symmetrische. In der Praxis verwenden Alice und Bob solche Verfahren daher nie für ganze Nachrichten, sondern stets nur für die Übertragung eines Schlüssels. Ein ähnliches Problem stellt sich bei digitalen Signaturen, wo ähnliche oder gar die gleichen Verfahren eingesetzt werden. Das blockweise Signieren einer längeren Nachricht kann für die Absenderin zu einer aufwendigen Angelegenheit werden, das spätere Verifizieren für Empfänger Bob ebenso. Von Nachteil ist zudem, dass blockweise erstellte Signaturen recht unhandlich sind, da sie zusammen mindestens die gleiche Länge haben wie die Nachricht selbst.

Um die beschriebenen Probleme zu vermeiden, sollte Alice nie eine ganze Nachricht signieren. Stattdessen sollte sie aus der Nachricht eine kurze Prüfsumme bilden und das Signaturverfahren nur auf diese anwenden. Eine solche Prüfsumme wird als **Hashwert** bezeichnet. Ein Verfahren zur Berechnung eines

Hashwerts heißt **Hashfunktion**. Mit Hashwerten, Hashfunktionen und deren Rolle in der Kryptografie beschäftigen wir uns in diesem Kapitel.

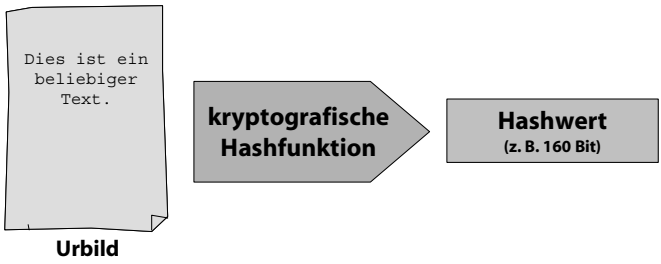


Abb. 14–1 *Statt eines ganzen Texts wird in der Praxis nur ein Hashwert signiert. Dieser wird mit einer kryptografischen Hashfunktion gebildet.*

14.1 Was ist eine kryptografische Hashfunktion?

Die Berechnung eines Hashwerts mithilfe einer Hashfunktion ist in der Informatik eine alltägliche Angelegenheit. Mit Kryptografie hat das zunächst einmal nichts zu tun. Mathematisch gesprochen sieht die Sache so aus: Ist H die Hashfunktion, h der Hashwert und m die zu signierende Nachricht (diese wird in diesem Zusammenhang auch **Urbild** genannt), dann gilt die Beziehung: $h=H(m)$. Der Hashwert h kann dabei nur eine begrenzte Menge von Werten annehmen, während m eine beliebige Größe hat.

14.1.1 Nichtkryptografische Hashfunktionen

Ein typischer, nicht aus der Kryptografie stammender Anwendungsfall für eine Hashfunktion ist folgender: Die Kryptobank (das führende Bankhaus in Kryptoland) vergibt Konten mit vierstelligen Nummern. Da sich die Kunden häufig mit der einen oder anderen Ziffer vertun, beschließt die Kryptobank, eine fünfte Stelle einzuführen. Die letzte Ziffer ist ein Hashwert der ersten vier. Täuscht sich Kundin Alice in einer Ziffer, dann stimmt in der Regel der Hashwert nicht mehr, und es fällt sofort auf, dass die Nummer falsch ist. Der Hashwert nimmt dabei also die Funktion einer Prüfsumme ein.

Im beschriebenen Fall gibt es 10.000 mögliche Urbilder (alle vierstelligen Zahlen) und zehn mögliche Hashwerte. Dabei kann es vorkommen, dass zwei unterschiedliche Urbilder denselben Hashwert erzeugen. Dies wird **Kollision** genannt. Bei einer guten Hashfunktion sollten Kollisionen möglichst selten vorkommen. Damit dies der Fall ist, muss die Hashfunktion folgende zwei Voraussetzungen erfüllen:

- Probiert man alle möglichen Urbilder durch, dann sollte jeder Hashwert etwa gleich oft vorkommen.
- Der Hashwert sollte sich auch bei kleineren Änderungen des Urbilds ändern. Im vorliegenden Beispiel wäre es schlecht, wenn etwa 4711 und 4712 denselben Hashwert lieferten.

In unserem Fall wäre es für die Kryptobank am einfachsten, die vier Ziffern des Urbilds zusammenzuzählen (Quersumme). Ist die Quersumme größer als 9, dann wird nur die letzte Ziffer genommen. Bei 4711 ergibt sich $4+7+1+1=13$, der Hashwert ist also 3. Alices vollständige Kontonummer lautet bei Verwendung dieser einfachen Hashfunktion 47113. Das Urbild 1147 liefert ebenfalls den Hashwert 3 – es liegt also eine Kollision vor.

14.1.2 Kryptografische Hashfunktionen

Kommen wir zurück zum Ausgangsproblem: Alice möchte statt einer ganzen Nachricht nur einen Hashwert signieren, damit das Signieren nicht zu einer abendfüllenden Betätigung ausartet. Die im vorhergehenden Kapitel beschriebene Quersummen-Hashfunktion wird sie dazu sicherlich nicht verwenden. Für Mallory ist es nämlich kein größeres Problem, eine Kollision (und damit eine zweite Nachricht mit dem gleichen Hashwert) zu erzeugen, wodurch er Alices Signatur auf einen Text übertragen kann, den sie nicht signiert hat.

Nimmt Alice statt der Quersummenmethode eine der zahlreichen anderen Hashfunktionen, die in der Literatur für nichtkryptografische Anwendungen vorgeschlagen wurden, dann bleibt das Problem in aller Regel bestehen. Dies liegt daran, dass Hashfunktionen, die nicht speziell für kryptografische Anwendungen entwickelt worden sind, für diese auch nicht geeignet sind. Wir haben hier also eine Situation, die Ihnen in diesem Buch noch häufiger begegnen wird (etwa, wenn es um Zufallsgeneratoren oder Protokolle geht): Die Kryptografie stellt im Vergleich zu anderen Anwendungen höhere Anforderungen.

Um eine Hashfunktion für digitale Signaturen verwenden zu können, müssen Alice und Bob an diese folgende Zusatzanforderung stellen: Es muss für Angreifer Mallory unmöglich sein, Kollisionen herbeizuführen. Diese Anforderung ist nicht trivial, weil Kollisionen immer existieren, sofern es mehr Urbilder als Hashwerte gibt. Eine Hashfunktion, die die genannte Zusatzanforderung erfüllt, wird als **kryptografische Hashfunktion** bezeichnet. Ein Hashwert, der mit einer kryptografischen Hashfunktion generiert wurde, heißt entsprechend **kryptografischer Hashwert**. Kryptografische Hashfunktionen bilden längst ein eigenes Teilgebiet der Kryptografie, in dem aktiv geforscht wird.

Leider ist die Namensgebung im Bereich der kryptografischen Hashfunktionen äußerst unübersichtlich. Allein mit den Bezeichnungen, die statt »kryptografische Hashfunktion« verwendet werden, könnte man problemlos eine halbe Seite füllen. So sind etwa die Begriffe Footprint-Funktion, Manipulation Detection

Code (MDC), sichere Hashfunktion oder Message Integrity Code (MIC) genauso gebräuchlich wie kryptografisches Prüfsummenverfahren oder Einweg-Hashfunktion. Gemeint ist damit in aller Regel das Gleiche. Entsprechend gibt es auch für den kryptografischen Hashwert unzählige Zweitnamen wie etwa Footprint, digitaler Fingerabdruck, Message Digest (MD) oder kryptografische Prüfsumme. Einfacher ist die Sache bei der Nachricht, die mit einer kryptografischen Hashfunktion bearbeitet wird. Diese wird (wie bereits erwähnt) als Urbild bezeichnet.

Alle gängigen kryptografischen Hashfunktionen verwenden Techniken, die Sie bereits aus der symmetrischen Kryptografie kennen. Sie teilen eine Nachricht in Blöcke auf und bearbeiten jeden Block in mehreren Runden. Dabei werden stets nur vergleichsweise einfache Bit-Operationen wie die Exklusiv-oder-Verknüpfung oder die Modulo-Addition verwendet. Wie bei einem symmetrischen Verschlüsselungsverfahren spielen lineare Bestandteile (für die Diffusion) sowie nichtlineare (für die Konfusion) eine zentrale Rolle. Typische Hashwert-Längen liegen zwischen 128 und 512 Bit. Das Urbild kann beliebig lang sein oder ist auf ein sehr großes Maximum begrenzt.

Wichtig ist die Tatsache, dass bei kryptografischen Hashfunktionen nichts verschlüsselt wird und dass es deshalb keine Schlüssel gibt. Jeder (auch Mallory) kann also den kryptografischen Hashwert zu einer gegebenen Nachricht berechnen. Es gibt allerdings einige Ausnahmen zu dieser Regel, die Sie in Abschnitt 14.5 kennenlernen werden.

14.1.3 Angriffe auf kryptografische Hashfunktionen

Schauen wir uns nun an, wie ein Angriff auf eine kryptografische Hashfunktion aussehen kann. Ziel von Angreifer Mallory ist es dabei stets, Kollisionen zu finden. Ähnlich wie man Angriffe auf ein Verschlüsselungsverfahren in Ciphertext-Only-, Known-Plaintext- und Chosen-Plaintext-Attacken einteilen kann, gibt es auch bei Angriffen auf kryptografische Hashfunktionen verschiedene Abstufungen:

- *Kollision*: Am einfachsten ist es für Mallory, eine Kollision zweier beliebiger Nachrichten zu finden. Dabei bedeutet »beliebig«, dass keine der beiden Nachrichten einen Sinn hat oder sonst irgendwelche Vorgaben erfüllen muss. Kryptografische Hashfunktionen, bei denen das Finden beliebiger Kollisionen nach aktuellem Kenntnisstand nicht möglich ist, werden **stark kollisionssicher** genannt.
- *Zweites Urbild*: Schwieriger wird es, wenn Mallory eine Kollision finden muss, bei der eine der beiden Nachrichten (oder ein Hashwert) vorgegeben ist. Einen solchen Fall bezeichnet man als **zweites Urbild**. Eine kryptografische Hashfunktion wird **schwach kollisionssicher** genannt, wenn sie das Auffinden zweiter Urbilder nach aktuellem Kenntnisstand nicht zulässt.

Ihnen ist sicherlich klar, dass es Mallory meist nicht reicht, beliebige Kollisionen zu finden, denn in der Praxis ist ja ein Urbild vorgegeben (zum Beispiel die signierte Nachricht, die Mallory von Alice abgefangen hat). Im Normalfall sind daher schwach kollisionssichere Hashfunktionen zur Bildung von Hashwerten für Signaturen ausreichend. Trotzdem gibt sich in der Praxis niemand mit einer schwach kollisionssicheren Hashfunktion zufrieden. Dies liegt zum einen daran, dass Kryptografen immer einen Sicherheitspuffer haben wollen, und ein solcher ist bei starker Kollisionsresistenz zweifellos gegeben. Es hat aber auch damit zu tun, dass mehrere gute, stark kollisionssichere kryptografische Hashfunktionen verfügbar sind – weshalb sollte man dann etwas Schwächeres einsetzen? Davon abgesehen kann man noch einen weiteren Grund ins Feld führen: Es gibt Anwendungen für kryptografische Hashfunktionen, bei denen Kollisionen grundsätzlich nicht möglich sein dürfen (in diesem Buch spielen solche Anwendungen jedoch keine Rolle).

Nimmt man es genau, dann reicht die Aufteilung in Kollisionen und zweite Urbilder nicht aus. In der Praxis genügt es Mallory nämlich nicht, *irgendein* zweites Urbild zu finden. Es muss vielmehr ein zweites Urbild sein, das einen sinnvollen Inhalt hat und Mallory einen Vorteil verschafft (etwa die Bestellung eines Staubsaugers). Diese Unterscheidung lässt sich jedoch nicht genau definieren – was ist ein sinnvoller Inhalt? – und wird von Kryptografen daher meist nicht näher erörtert.

Sehr wohl interessant ist jedoch eine andere Anforderung, die Kryptografen gemeinhin stellen. Diese lautet: Eine kryptografische Hashfunktion sollte ein Zufallsorakel bilden. Zwischen Urbild und Hashwert darf es also keinen erkennbaren Zusammenhang geben. Dies bedeutet insbesondere, dass auch für kryptografische Hashfunktionen der Avalanche-Effekt gelten muss – ändert sich ein Bit im Urbild, dann muss sich im Schnitt die Hälfte der Bits im Hashwert ändern.

Substitutionsattacke

Überraschenderweise gibt es einen sehr einfachen Angriff auf kryptografische Hashfunktionen, mit dem Mallory ein zweites Urbild findet, dessen Inhalt er sogar weitgehend bestimmen kann. Dieser Angriff funktioniert grundsätzlich bei jeder kryptografischen Hashfunktion. Er wird als **Substitutionsattacke** bezeichnet. Um zu verstehen, wie eine Substitutionsattacke abläuft, nehmen wir an, Mallory habe eine Nachricht von Alice mit signiertem kryptografischen Hashwert abgefangen. Mallory sucht nun nach einem Text mit gleichem Hashwert, in dem ein Staubsauger bestellt wird. Dazu setzt er folgenden Text auf:

```
Ich bestelle hiermit einen Staubsauger vom Typ Cryptosaug Plus.
Bitte liefern Sie bis Samstag 16 Uhr.
Alice Onliner
```

In diesem Text sucht Mallory nach Wörtern, die er ersetzen kann, ohne dass sich der Sinn verändert. Hier einige Beispiele:

- »Samstag« kann Mallory durch »Sonnabend« ersetzen.
- »16 Uhr« kann er durch »16.00 Uhr« ersetzen.
- »Alice Onliner« kann er durch »A. Onliner« ersetzen.
- »hiermit« kann er ganz weglassen.

Nehmen wir an, Mallory habe *n* Ersetzungsmöglichkeiten gefunden. Durch die Kombination von Ersetzungen kann er 2^n Staubsauger-Bestellungen mit gleicher Bedeutung, aber meist unterschiedlichem Hashwert generieren. Mallory berechnet nun nacheinander alle 2^n Hashwerte, bis er irgendwann auf eine Staubsauger-Bestellnachricht stößt, die denselben Hashwert wie die abgefangene Nachricht hat. Damit ist ein zweites Urbild gefunden. Hat der Hashwert der kryptografischen Hashfunktion eine Länge, die kleiner als *n* ist (also kleiner als die Anzahl der Ersetzungsmöglichkeiten), dann ist die Wahrscheinlichkeit sehr groß, dass Mallory mit dieser Methode ein zweites Urbild findet.

Dabei gibt es für eine Substitutionsattacke oft bessere Möglichkeiten als das Ersetzen von Wörtern. So kann Mallory etwa ein Leerzeichen durch ein anderes nicht druckbares Zeichen ersetzen, ohne dass ein Unterschied auffällt. Oder er kann am Ende des Textes einige Leerzeichen hinzufügen. Bei vielen Formaten (etwa HTML) gibt es zudem die Möglichkeit, Kommentare einzufügen, was Mallory ebenfalls nutzen kann.

Eine Substitutionsattacke funktioniert bei jeder kryptografischen Hashfunktion und bei jeder Art von Nachricht. Die einzige Gegenmaßnahme besteht darin, dass Alice den Hashwert ausreichend lang wählt, denn je länger der Hashwert, desto größer die Anzahl der Nachrichten, die Mallory durchprobieren muss. Hat der Hashwert etwa die Länge 160 Bit, dann muss Mallory entsprechend etwa 160 Ersetzungsmöglichkeiten (oder mehr) suchen und daraus 2^{160} Hashwerte bilden, die er durchprobieren muss. Wenn wir davon ausgehen, dass er im Schnitt schon nach der Hälfte der durchprobierten Werte Erfolg hat, dann muss er immerhin $2^{159} = 7{,}3 \cdot 10^{47}$ Werte generieren, was selbst mit den besten jemals denkbaren Superrechnern nicht vor Ende dieses Universums zu schaffen ist.

Geburtstagsangriff

Versuchen Sie einmal, folgende Frage zu beantworten: Wie viele Menschen müssen in einem Raum versammelt sein, damit mit einer Wahrscheinlichkeit von ½ mindestens zwei davon am gleichen Tag Geburtstag haben? Diese Frage wird auch **Geburtstagsproblem** genannt. Die Antwort: Schon wenn 22 Menschen in einem Raum versammelt sind, dann beträgt die Wahrscheinlichkeit etwa ½. Dies bedeutet Folgendes: Wenn wir eine größere Anzahl von 22-köpfigen Menschengruppen zusammenstellen, dann gibt es in etwa der Hälfte davon zwei Personen, die am gleichen Tag Geburtstag feiern. Das Geburtstagsproblem lässt sich auch

allgemeiner formulieren. Nehmen wir an, ein Jahr habe *t* Tage. Dann beträgt die Anzahl der Menschen, die zusammenkommen müssen, damit die Wahrscheinlichkeit eines gleichen Geburtstags ½ ist, stets etwas mehr als die Wurzel von *t*.

Was das alles mit kryptografischen Hashfunktionen zu tun hat? Ganz einfach: Will Mallory wissen, wie viele Nachrichten er im Durchschnitt durchprobieren muss, bis er durch Zufall eine Kollision gefunden hat, dann braucht er nur das Geburtstagsproblem zu bemühen. Die Zahl der möglichen Hashwerte ist dabei die Anzahl der Tage im Jahr, die Zahl der durchprobierten Nachrichten ist die Anzahl der Menschen. Hat der Hashwert die Länge *n*, dann gibt es 2^n mögliche Hashwerte. Die Wurzel aus 2^n beträgt $2^{n/2}$, was bedeutet, dass Mallory im Schnitt bereits nach $2^{n/2}$ Versuchen eine Kollision findet.

Ein Angriff, der das Geburtstagsphänomen ausnutzt, wird als **Geburtstagsangriff** bezeichnet. Die einfachste Form des Geburtstagsangriffs kennen Sie bereits: Das Anwenden einer kryptografischen Hashfunktion auf alle möglichen Nachrichten, bis eine Kollision gefunden ist. Bei einem Hashwert der Länge 128 Bit benötigt Mallory bei dieser Methode durchschnittlich 2^{64} (also etwa 10^{19}) Versuche. Um diese mit einem PC durchzuprobieren, ist Mallory einige zehntausend Jahre beschäftigt. Für die NSA mit ihren Supercomputern dürfte dies jedoch etwas schneller gehen, weshalb alle neueren kryptografischen Hashfunktionen 160 Bit oder mehr als Länge des Hashwerts verwenden.

Es gibt noch andere Formen des Geburtstagsangriffs. Kann Mallory Nachrichten, die Alice signiert, vorformulieren, dann funktioniert beispielsweise folgende Methode (*n* ist dabei die Länge des Hashwerts in Bit):

1. Mallory, der weiß, dass Alice eine Packung Büroklammern bestellen will, formuliert mit einer Substitutionsattacke $2^{n/2}$ Versionen einer unverdächtigen Bestell-Nachricht (etwa »Ich bestelle eine Packung Büroklammern. Alice«).
2. Mallory formuliert eine Nachricht mit dem von ihm gewünschten Inhalt (etwa »Ich bestelle einen Staubsauger. Alice«). Diese Nachricht verändert er fortlaufend durch die von der Substitutionsattacke bekannte Methode und berechnet jeweils den Hashwert. Diesen vergleicht er dann jeweils mit den Hashwerten der $2^{n/2}$ unverdächtigen Nachrichten. Das Ganze macht er so lange, bis er schließlich eine Kollision zwischen einer Büroklammer- und einer Staubsauger-Nachricht gefunden hat.
3. Die unverdächtige Büroklammer-Nachricht legt Mallory nun Alice zum Signieren vor (Alice schöpft keinen Verdacht, da sie ja ohnehin Büroklammern bestellen wollte).
4. Die Staubsauger-Nachricht mit gleichem Hashwert gibt Mallory an seinen Freund, den Staubsauger-Vertreter, weiter.

Damit eine solche Geburtstagsattacke Erfolg hat, muss Mallory durchschnittlich $2^{n/2}$ Staubsauger-Nachrichten durchprobieren. So etwas ist schon bei einem Hashwert der Länge 128 Bit kaum zu schaffen und bei 160 Bit absolut aussichtslos.

Wörterbuch-Angriff

Der in der Praxis mit Abstand häufigste und wirksamste Angriff auf kryptografische Hashfunktionen ist der **Wörterbuch-Angriff**. Ein solcher funktioniert dann, wenn das Urbild aus nur einem Wort oder einem kurzen Satz besteht. Diese Voraussetzung ist vor allem dann gegeben, wenn es sich beim Urbild um ein Passwort handelt. Viele Computersysteme speichern aus Sicherheitsgründen keine Passwörter ab, sondern begnügen sich mit dem jeweiligen Hashwert. Mit einem Wörterbuch-Angriff kann Mallory das zum Hashwert gehörige Passwort ermitteln.

Ein Wörterbuch-Angriff ist denkbar einfach. Mallory besorgt sich zunächst Wörterbuchdateien, in denen möglichst viele Wörter aus unterschiedlichen Sprachen enthalten sind. Wer sich im Internet umsieht, findet solche Dateien zuhauf. Viele davon enthalten auch Spezialausdrücke wie geografische Begriffe oder Personennamen. Anschließend berechnet Mallory zu jedem einzelnen Wort aus seinen Wörterbüchern den Hashwert und vergleicht ihn mit dem gesuchten Wert. Irgendwann ergibt sich ein Treffer, und Mallory hat das gesuchte Urbild gefunden. Ein Wörterbuch-Angriff ist also eine Brute-Force-Methode.

Wörterbuch-Angriffe funktionieren erstaunlich gut. Es gibt spezielle Computerprogramme wie *Crack* oder *John the Ripper*, die oft schon in wenigen Minuten das passende Passwort finden. Solche Programme prüfen nicht nur das jeweilige Wort aus dem Wörterbuch (z. B. »Alice«), sondern probieren gleich auch verschiedene Abwandlungen wie »alice«, »ALICE«, »aLICE«, »AlIcE«, »A1ice« oder »ecilA« aus. Am besten funktioniert eine derartige Software, wenn Mallory eine ganze Liste von gesuchten Hashwerten eingibt. Dann nämlich kann das Programm einen berechneten Hashwert nacheinander mit mehreren Werten vergleichen, was die Wahrscheinlichkeit auf einen Treffer erhöht.

Das beste Mittel gegen einen Wörterbuch-Angriff ist die Wahl eines Urbilds, das nicht – auch nicht in Abwandlungen – in einem Wörterbuch zu finden ist. Davon abgesehen gibt es keine Möglichkeit, einen Wörterbuch-Angriff zu verhindern – allerdings kann man einen solchen erschweren. Eine Möglichkeit dazu ist ein sogenanntes **Salt** (benannt nach dem englischen Wort für »Salz«). Ein Salt ist ein beliebiger Wert, der in die Hashwert-Berechnung eingeht und der für jedes Urbild verschieden ist. Es handelt sich dabei allerdings nicht um einen Schlüssel (einen solchen gibt es in diesem Fall nicht), sondern um einen öffentlich bekannten Wert. Da auch Mallory das Salt kennt, hindert dieses ihn nicht an einem Wörterbuch-Angriff. Es erschwert einen solchen jedoch, wenn Mallory eine ganze Liste von gesuchten Hashwerten bearbeitet. Ein Beispiel: Hat Mallory eine Liste von 100 Hashwerten, die er knacken will, dann genügt ihm ohne Salt eine Hashwert-Berechnung für 100 Vergleiche. Mit Salt muss er dagegen für jeden Vergleich einen neuen Hashwert berechnen.

Es gibt noch eine weitere Möglichkeit, Wörterbuch-Angriffe zu erschweren: Alice wendet zur Generierung des Hashwerts die Hashfunktion nicht einmal, sondern 100 Mal an (**Hash-Iteration**). Normalerweise ist so etwas unnötig, da

schon ein Hashfunktion-Durchlauf sicher genug ist. In diesem Fall führt die Hash-Iteration jedoch dazu, dass auch Mallory für jede Prüfung 100 Durchläufe benötigt. Seine Suche wird dadurch um den Faktor 100 langsamer. Natürlich kann Alice auch 1.000 oder 1.589 Durchläufe wählen – je nachdem, welchen Performanzverlust sie in Kauf nehmen will.

Regenbogentabellen

Kryptografische Hashfunktionen haben für Mallory einen entscheidenden Vorteil: Da es keinen Schlüssel gibt, erzeugt dasselbe Urbild bei Verwendung derselben Hashfunktion immer denselben Hashwert – überall und zu jeder Zeit. Hat Mallory erst einmal eine Kollision oder ein passendes Urbild gefunden, dann lässt sich dieser Erfolg nicht mehr aus der Welt schaffen. Bei einem Verschlüsselungsverfahren ist das anders: Hier können Alice und Bob eine erfolgreiche Schlüsselsuche durch einen Schlüsselwechsel zunichte machen.

Die beschriebene Tatsache kann Mallory nutzen, indem er eine möglichst große und ständig wachsende Sammlung von Urbild-Hashwert-Paaren anlegt (**Hashwert-Tabelle**). Falls es mehrere Sammlungen gibt, lassen sich diese zu einer besonders großen Hashwert-Tabelle vereinigen, die beispielsweise im Internet veröffentlicht wird. Sucht Mallory zu einem bestimmten Hashwert ein passendes Urbild, dann schaut er in der Hashwert-Tabelle nach. Dies ist ziemlich mühsam, da Mallory mit einer riesigen Tabelle hantieren muss. Allerdings ist das Lesen eines Urbild-Hashwert-Paares aus einer Tabelle immer noch deutlich performanter als das Berechnen des Hashwerts aus dem Urbild.

Wie Sie leicht nachvollziehen können, funktionieren Hashwert-Tabellen besonders gut im Zusammenhang mit Wörterbuch-Angriffen. So kann Mallory beispielsweise zu jedem Wörterbucheintrag vorab den passenden Hashwert berechnen und speichern. Hat Mallory eine geeignete Tabelle zur Hand, dann wird ein Wörterbuch-Angriff deutlich effektiver, wobei zudem Hash-Iterationen viel von ihrer Wirkung verlieren.

Als größter Nachteil einer Hashwert-Tabelle gilt, dass Mallory eine enorme Menge an Speicher benötigt. Mehrere Kryptografen haben daher in den letzten Jahren Verfahren vorgeschlagen, mit denen sich Speicher einsparen lässt, ohne dabei zu viel an Geschwindigkeit einzubüßen. Praktisch alle dieser Verfahren sind Varianten einer Methode, die ich im Folgenden an einem Beispiel erklären will. Wir nehmen an, dass das Urbild eine positive ganze Zahl ist (z.B. 311). Als Hashfunktion nehmen wir eine Funktion, die alle Stellen bis auf die letzten zwei streicht (dies ist zwar keine kryptografische Hashfunktion, reicht aber aus, um die Methode zu erklären). Der Hashwert von 311 ist also 11. Außerdem benötigen wir eine sogenannte **Reduktionsfunktion**. Als einfaches Beispiel verwenden wir hierfür eine Funktion, die den Eingabewert mit 13 multipliziert (aus 11 wird also 143).

Mallory geht nun wie folgt vor: Er nimmt ein beliebiges Urbild (*Urbild 1*) und berechnet dazu den Hashwert (*Hashwert 1*). Auf *Hashwert 1* wendet er die

Reduktionsfunktion an und erhält dadurch *Urbild 2*. Aus diesem berechnet er *Hashwert 2*, woraus durch die Reduktionsfunktion *Urbild 3* entsteht. Hat Mallory auf analoge Weise schließlich auch *Hashwert 3*, *Urbild 3* und *Hashwert 4* berechnet, dann legt er eine Tabellenzeile an, in die er die soeben berechneten Hashwerte und Urbilder einträgt. Anschließend nimmt Mallory ein neues *Urbild 1* und beginnt wieder von vorne. So entsteht eine Tabelle wie die folgende:

Urbild 1	Hashwert 1	Urbild 2	Hashwert 2	Urbild 3	Hashwert 3	Urbild 4	Hashwert 4
311	11	143	43	559	59	767	67
607	07	91	91	1.183	83	1.079	79
524	24	312	12	156	56	728	28
210	10	130	30	390	90	1.170	70
773	73	949	49	637	37	481	81
...	...	...	...	...	...	...	...

Von dieser Tabelle speichert Mallory nur die erste und letzte Spalte (also jeweils *Urbild 1* und *Hashwert 4*). Den Rest löscht er, um Speicherplatz zu sparen. Mallory kann die zweispaltige Tabelle nun wie folgt nutzen, um zu einem vorgegebenen Hashwert h ein passendes Urbild zu suchen:

1. Zunächst durchsucht Mallory die letzte Spalte (*Hashwert 4*) nach h. Wird er dort fündig (etwa bei h=70), dann muss er nur noch das zugehörige *Urbild 4* ermitteln, um ans Ziel zu kommen. *Urbild 4* ist zwar in der Tabelle nicht gespeichert, doch mithilfe von *Urbild 1* kann er es durch Anwendung der Hashfunktion und der Reduktionsfunktion einfach berechnen.
2. Findet Mallory h in der letzten Spalte nicht, dann müsste er als Nächstes in der drittletzten Spalte suchen (*Hashwert 3*). Diese Spalte hat er jedoch nicht gespeichert. Er kann sich allerdings behelfen, indem er auf h erst die Reduktionsfunktion und dann die Hashfunktion anwendet. Nach dem Ergebnis dieser Berechnung sucht er dann wiederum in der letzten Spalte. Wird er dort fündig, dann kann er mithilfe von *Hashwert 1* das passende *Urbild 3* bestimmen.
3. Findet sich h nicht in der drittletzten Spalte, dann schaut Mallory in der fünftletzten Spalte nach (*Hashwert 2*). Er sucht jedoch nicht direkt darin (er hat die Spalte ja nicht gespeichert), sondern wendet auf h die Reduktionsfunktion, dann die Hashfunktion, dann erneut die Reduktionsfunktion und schließlich erneut die Hashfunktion an. Nach dem Ergebnis dieser Berechnung sucht er in der letzten Spalte.
4. Nach der fünftletzten ist die siebtletzte Spalte an der Reihe. Die Vorgehensweise sollte klar sein.

Natürlich kann die Suche auch erfolglos verlaufen – dann ist die Hashwert-Tabelle zu klein. Als Urbilder kommen in der Praxis oft Buchstabenfolgen zum Einsatz, bei denen es sich um ein Passwort handeln könnte. Die Reduktionsfunk-

tion ist in diesem Fall eine Funktion, die einen Hashwert (z.B. 160 Bit) auf eine kürzere Buchstabenfolge abbildet. In dieser Form sind Hashwert-Tabellen eine wirksame Waffe gegen gehashte Passwörter. Der Vorteil der gezeigten Methode liegt auf der Hand: Statt acht Spalten muss Mallory nur zwei speichern, was 75 Prozent an Speicherplatz einspart. In der Praxis wird Mallory nicht nur vier Hashwerte pro Zeile speichern, sondern beispielsweise 5.000. Die Einsparung an Speicherplatz ist entsprechend groß.

Ein Problem bei dieser Art von Hashwert-Tabellen besteht darin, dass es Überschneidungen gibt. Tritt an einer Stelle ein Hashwert auf, der schon einmal vorkam, dann sind auch die folgenden Urbilder und Hashwerte dieser Zeile bereits bekannt und vergeuden Tabellenplatz. Philippe Oechslin schlug zur Behebung dieses Problems vor, eine Reduktionsfunktion zu verwenden, die sich in jeder Spalte ändert [Oechsl]. Dies verhindert zwar Überschneidungen nicht, sorgt jedoch dafür, dass ein doppelt vorkommender Hashwert nicht automatisch weitere doppelt vorkommende Hashwerte nach sich zieht. Hashwert-Tabellen, die so funktionieren, gelten derzeit als die besten ihrer Art. Sie werden **Regenbogentabellen** genannt. Es gibt derzeit zwei kostenlose Softwareprogramme, die Regenbogentabellen berechnen: *RainbowCrack* und *DistrRTgen*. Letzteres ermöglicht es, die Rechenzeit auf beliebig viele Rechner zu verteilen. Unter der Adresse www.freerainbowtables.com gibt es eine größere Auswahl an Regenbogentabellen, die mit DistrRTgen generiert wurden.

Regenbogentabellen sind eine wirksame Waffe, wenn Mallory die Menge der Urbilder eingrenzen kann (z.B. auf Buchstabenfolgen begrenzter Länge). Es gibt jedoch ein probates Gegenmittel: das bereits erwähnte Salt. Lassen Alice und Bob beispielsweise in jede Hashwert-Berechnung einen Vier-Byte-Wert als Salt einfließen, dann muss Mallory seine Regenbogentabelle um den Faktor 4 Milliarden vergrößern, um dieselbe Erfolgschance zu haben. Davon abgesehen hilft gegen Regenbogentabellen nur die Macht großer Zahlen. Wenn Alice und Bob eine Hashfunktion mit 160 Bit Hashwert-Länge verwenden und außerdem beliebige Urbilder vorkommen können (also nicht nur Buchstabenfolgen), dann kann Mallory mit einer Regenbogentabelle nicht viel ausrichten.

Fazit

Substitutionsattacke, Geburtstagsangriff, Wörterbuch-Angriff und Regenbogentabellen zeigen, dass Angriffe auf kryptografische Hashfunktionen einfacher sind als auf symmetrische Verschlüsselungsverfahren. Den beiden erstgenannten Angriffen kann man unabhängig vom jeweiligen Design der kryptografischen Hashfunktion nur durch eine ausreichende Hashwert-Länge begegnen. Diese sollte daher länger gewählt werden als die Schlüssellänge eines symmetrischen Verfahrens. 160 Bit gelten heutzutage als Minimum.

Natürlich sind die an dieser Stelle behandelten Angriffe nur die Spitze des kryptoanalytischen Eisbergs im Bereich der kryptografischen Hashfunktionen.

Die meisten weiteren Angriffe sind jedoch von der Funktionsweise des jeweiligen Verfahrens abhängig. So ist beispielsweise auch das aus der symmetrischen Kryptografie bekannte Prinzip der differenziellen Kryptoanalyse auf kryptografische Hashfunktionen anwendbar. Die lineare Kryptoanalyse funktioniert in diesem Bereich dagegen nicht.

14.2 MD4-artige Hashfunktionen

Nachdem wir uns mit der Theorie der kryptografischen Hashfunktionen beschäftigt haben, kommen wir nun zu den Verfahren selbst. Wir schauen uns die wichtigsten Vertreter genauer an.

14.2.1 SHA-1

Die lange Zeit wichtigste kryptografische Hashfunktion heißt **SHA-1** [RFC3174]. Die Abkürzung SHA steht für Secure Hash Algorithm, und die Eins kam in den Namen, nachdem die Funktionsweise leicht geändert wurde (vorher hieß das Verfahren einfach nur SHA). Entwickelt wurden SHA und SHA-1 von der NSA im Auftrag der US-Standardisierungsbehörde NIST. Die Entwicklung fand parallel zum ebenfalls bei der NSA entstandenen Digital Signature Algorithm (DSA) statt, den ich bereits in Abschnitt 12.3.2 beschrieben habe.

DSA und SHA wurden 1991 der Öffentlichkeit vorgestellt. Obwohl Entwicklungen der NSA von Kryptografen stets mit Misstrauen betrachtet werden, erwies sich SHA-1 als gute kryptografische Hashfunktion. Absichtlich eingebaute Schwächen, die der NSA immer zugetraut werden, sind nie entdeckt worden. Man kann allerdings auch nicht behaupten, dass die NSA bei der Entwicklung von SHA-1 vollkommen neue kryptografische Wege gegangen ist, denn SHA-1 ist eine Weiterentwicklung der kryptografischen Hashfunktion MD4, von der noch die Rede sein wird.

Funktionsweise von SHA-1

SHA-1 verarbeitet Blöcke der Länge 512 Bit und generiert einen Hashwert der Länge 160 Bit. Wenn Alice ein Urbild mit SHA-1 bearbeiten will, dann muss sie dieses zunächst in 512-Bit-Blöcke aufteilen, die wir als U_0, U_1, U_2, U_3, ... bezeichnen. Da die Länge eines Urbilds in der Regel nicht ein Vielfaches von 512 Bit beträgt, schreibt das SHA-1-Verfahren eine Methode für das Auffüllen des letzten Blocks vor (Padding). Diese funktioniert so:

1. Sind weniger als 64 Bit aufzufüllen, dann hänge einen zusätzlichen 512-Bit-Block an das Urbild.
2. Fülle das Urbild so auf, dass im letzten Block noch genau 64 Bit übrig bleiben. Die Auffülldaten beginnen mit einem Bit des Werts 1, anschließend folgen

lauter Null-Bits. Die Länge des Urbilds vor dem Auffüllen muss im Übrigen kein Vielfaches von 8 betragen. Die 1 kann daher auch mitten in einem Byte stehen.

3. In die verbleibenden 64 Bit schreibe die Bit-Länge der ursprünglichen Nachricht. Ist das Urbild länger als 2^{64} Bit (in der Praxis dürfte dies kaum vorkommen), dann werden nur die 64 niederwertigsten Bits verwendet.

Eine zentrale Rolle bei der Funktionsweise von SHA-1 spielen fünf 32-Bit-Variablen (**Kettenvariablen**), die zusammen auch als **Status** bezeichnet werden. Wir bezeichnen diese als *A*, *B*, *C*, *D* und *E*. Am Anfang stehen in den Kettenvariablen (in hexadezimaler Schreibweise) die Werte *A*=67452301, *B*=EFCDAB89, *C*=98BADCFE, *D*=10325476 und *E*=C3D2E1F0. Eine weitere wichtige Rolle spielt die sogenannte **Kompressionsfunktion**, die wir mit dem Buchstaben *f* bezeichnen wollen. Die Kompressionsfunktion nimmt den aktuellen Inhalt der fünf Kettenvariablen sowie einen 512-Bit-Block des Urbilds entgegen und liefert den neuen Wert der Kettenvariablen als Ausgabe.

Die Generierung eines SHA-1-Hashwerts läuft wie folgt ab: Die Kompressionsfunktion wird zunächst auf U_0 angewendet, dann auf U_1, dann auf U_2 usw. Der Inhalt der Kettenvariablen ändert sich dabei mit jedem Abarbeiten der Kompressionsfunktion. Die fünf Werte, die am Ende in den Kettenvariablen stehen, bilden zusammen den Hashwert. In Pseudocode sieht das so aus:

```
A=67452301, B=EFCDAB89
C=98BADCFE, D=10325476
E=C3D2E1F0

Für i:=0 bis (Urbildlänge−1):   (A,B,C,D,E):=f(U_i,A,B,C,D,E)
Hashwert := (A,B,C,D,E)
```

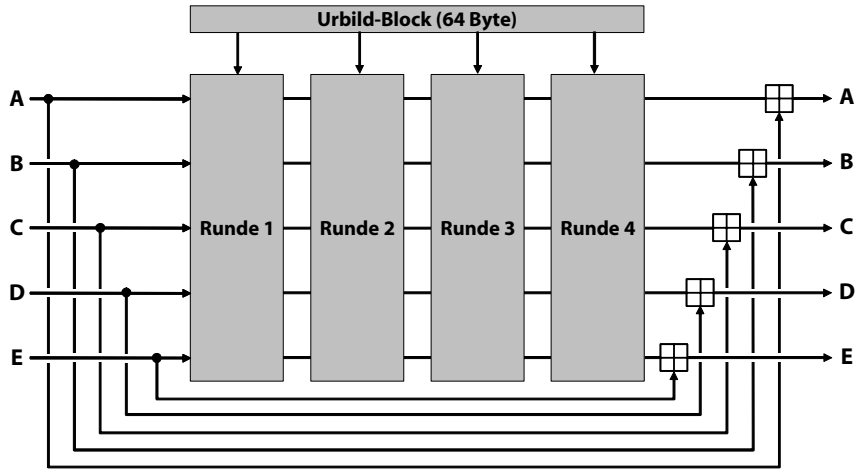


Abb. 14–2 *Die Kompressionsfunktion von SHA-1 arbeitet in vier Runden.*

Die SHA-1-Kompressionsfunktion läuft in vier Runden ab (siehe Abbildung 14–2). Nach der vierten Runde wird zu jeder Kettenvariable der Wert dazugezählt, den diese vor der ersten Runde hatte. Jede Runde besteht aus 20 Teilrunden. Den Ablauf einer Teilrunde sehen Sie in Abbildung 14–3. In jede Teilrunde geht eine Konstante K ein. Zudem spielt die Funktion g eine Rolle. Funktion und Konstante (in Hexadezimalschreibweise) sind abhängig von der jeweiligen Runde wie folgt definiert (g ist in Runde 1 und 3 nicht linear, dadurch entsteht die notwendige Konfusion):

	Funktion g	Konstante K
Runde 1	$(B \wedge C) \vee ((\neg B) \wedge D)$	5A827999
Runde 2	$B \oplus C \oplus D$	6ED9EBA1
Runde 3	$(B \wedge C) \vee (B \wedge D)$	8F1BBCDC
Runde 4	$B \oplus C \oplus D$	CA62C1D6

Zudem geht in jede Teilrunde der SHA-1-Kompressionsfunktion ein 32-Bit-Block aus dem (aufgefüllten) Urbild ein. Wie man leicht nachrechnet, werden zu diesem Zweck insgesamt 80 32-Bit-Blöcke benötigt. Da ein Urbildblock jedoch nur 16 32-Bit-Blöcke besitzt, ist eine *Expansion* notwendig (vergleichbar mit der Schlüsselaufbereitung bei einem symmetrischen Verschlüsselungsverfahren). Wenn wir die 16 Urbild-Teilblöcke mit T_0, T_1, ..., T_{15} und die expandierten Teilblöcke mit E_0, E_1, ..., E_{79} bezeichnen, dann funktioniert die Expansion nach der folgenden Formel:

$E_i = T_i$ für i von 0 bis 15

$E_i = (E_{i-3} \oplus E_{i-8} \oplus E_{i-14} \oplus E_{i-16}) <<< 1$ für i von 16 bis 79

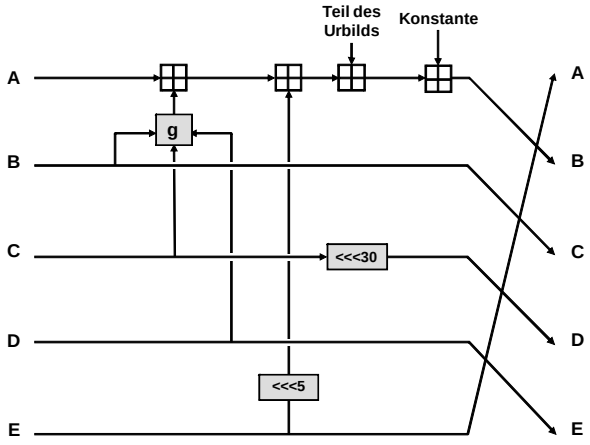


Abb. 14–3 *Diese Operation wird in jeder Runde von SHA-1 20-mal durchlaufen. Die Funktion g und die Konstante ändern sich mit jeder Runde.*

Bewertung von SHA-1

Ende der neunziger Jahre entwickelte sich SHA-1 zur bedeutendsten kryptografischen Hashfunktion überhaupt. Sie kommt unter anderem in den gängigen Webbrowsern, in PGP sowie in den Netzwerkprotokollen SSL, IPsec und S/MIME zur Anwendung. Bis zum Jahr 2004 konnte man sich bei der Verwendung von SHA-1 auf der sicheren Seite fühlen. Dann kamen jedoch erstaunliche Nachrichten aus China. Dort war es der Kryptografin Xiaoyun Wang zusammen mit zwei Kollegen gelungen, eine Methode zur Generierung von SHA-1-Kollisionen zu entwickeln, die schneller funktionierte als ein Geburtstagsangriff [WaYiYu]. Statt 2^{80} benötigte dieser Angriff »nur« 2^{69} Funktionsaufrufe. Noch im selben Jahr verbesserten Xiaoyun Wang, Andrew Yao und Frances Yao diesen Wert auf 2^{63} [WaYaYa]. Krypto-Papst Bruce Schneier bezeichnete SHA-1 bereits nach der ersten Veröffentlichung als gebrochen [Schn05]. In der Tat liegen 2^{63} Funktionsaufrufe gerade noch im Bereich des Möglichen. SHA-1 hat also einige ernsthafte Kratzer abbekommen. Noch ist zwar kein konkreter Fall einer Kollisionsberechnung bekannt, und der Aufwand für ein solches Unterfangen ist mit den derzeit verfügbaren Mitteln gewaltig. Schon die nächste Verbesserung des besagten Angriffs könnte die Situation jedoch ändern. SHA-1 sollte man daher in neuen Entwicklungen vermeiden.

14.2.2 Neue SHA-Varianten

Schon vor den Angriffen durch die chinesischen Kryptografen veröffentlichte die US-Standardisierungsbehörde NIST im Jahr 2000 vier neue SHA-Versionen: SHA-224, SHA-256, SHA-384 und SHA-512 (die Zahl im Namen steht jeweils für die Länge des Hashwerts). Die vier neuen SHA-Funktionen unterscheiden sich nicht nur in der Hashwert-Länge von SHA-1, sondern weisen auch eine ganze Reihe funktionaler Unterschiede auf [FIPS-180, RFC4634].

SHA-224 und SHA-256

SHA-256 unterscheidet sich wie folgt von SHA-1:

- Die Initialisierungswerte für die Kettenvariablen wurden geändert.
- Es gibt nur noch 16 statt 20 Teilrunden pro Runde.
- Für jede Teilrunde wird eine andere Konstante verwendet.
- Die Formel für die Funktion *g* wurde geändert und enthält nun einige zusätzliche Rotationen.
- Die Expansionsformel wurde geändert und enthält nun ebenfalls zusätzliche Rotationen sowie zwei Schiebeoperationen.

SHA-224 hat denselben Ablauf wie SHA-256, nur dass am Ende 32 Bit des Hashwerts abgeschnitten werden. SHA-224 und SHA-256 sind von dem aus China stammenden Angriff auf SHA-1 nicht betroffen.

SHA-384 und SHA-512

SHA-512 hat folgende funktionale Unterschiede zu SHA-256:

- Statt 32-Bit-Blöcken werden 64-Bit-Blöcke verwendet.
- Wie in SHA-1 gibt es 20 Teilrunden pro Runde.
- Die Initialisierungswerte für die Kettenvariablen und die Konstanten sind anders.
- Die Schiebeoperationen und die Rotationen sind anders.

Beachten Sie, dass SHA-512 im Gegensatz zu SHA-256 mit 20 Teilrunden pro Runde arbeitet. **SHA-384** hat denselben Ablauf wie SHA-512, nur dass am Ende 128 Bit des Hashwerts abgeschnitten werden. Auch SHA-384 und SHA-512 haben sich bisher als resistent gegenüber dem Angriff aus China erwiesen.

Bewertung der neuen SHA-Varianten

Durch die erfolgreichen Angriffe auf SHA-1 ist die Bedeutung der vier SHA-2-Varianten erheblich gestiegen. Zusammen mit Keccak, das den SHA-3-Wettbewerb gewonnen hat (siehe Abschnitt 14.3), sind sie momentan erste Wahl, wenn es um kryptografische Hashfunktionen geht.

14.2.3 MD4

MD4 (Message Digest 4) ist eine kryptografische Hashfunktion, die von RSA-Miterfinder Ron Rivest entwickelt wurde [RFC1320]. Dass MD4 eine große Ähnlichkeit mit SHA-1 aufweist, ist kein Zufall, denn SHA-1 ist eine Weiterentwicklung von MD4. Aufgrund dieser Ähnlichkeit beschränke ich mich an dieser Stelle auf eine Beschreibung der Unterschiede:

- MD4 liefert einen Hashwert der Länge 128 Bit. Bei SHA-1 sind es 160 Bit. Deshalb gibt es bei MD4 auch nur vier Kettenvariablen.
- MD4 begnügt sich mit drei Runden pro Urbildblock, bei SHA-1 sind es vier.
- MD4 begnügt sich mit 16 Teilrunden pro Runde, bei SHA-1 sind es 20.
- Die Teilrunden von MD4 sind etwas anders aufgebaut als bei SHA-1.

MD4 ist ohne Zweifel ein Verfahren, das einen großen Einfluss auf die Entwicklung der kryptografischen Hashfunktionen genommen hat. Neben den verschiedenen SHA-Verfahren sind auch die meisten anderen in diesem Kapitel vorgestellten kryptografischen Hashfunktionen Weiterentwicklungen von MD4. Überragend ist in jedem Fall die Geschwindigkeit, die sich mit MD4 erreichen lässt. Der Durchsatz des Verfahrens ist dank der geringeren Runden- und Teilrundenzahl sowie des kürzeren Hashwerts etwa viermal so hoch wie bei SHA-1. Das knappe Bemessen der genannten Parameter sowie das aus heutiger Sicht nicht optimale Design einiger Bestandteile haben jedoch dazu geführt, dass die Sicherheit von MD4 nicht mehr allzu hoch ist. Mit Angriffsmethoden, die in den letzten

Jahren entdeckt wurden, ist es inzwischen selbst mit geringen Mitteln möglich, Kollisionen zu finden. MD4 sollten Sie daher nicht mehr verwenden.

14.2.4 MD5

Die Mängel an MD4 veranlassten Rivest bereits 1991, eine überarbeitete Version des Verfahrens zu veröffentlichen. Das Ergebnis hieß **MD5** und wurde vorübergehend zur meistverwendeten kryptografischen Hashfunktion überhaupt. MD5 wurde inzwischen von der IETF standardisiert und ist in [RFC1321] beschrieben. Da sowohl MD5 als auch SHA-1 Weiterentwicklungen von MD4 sind, sind sich die beiden Verfahren sehr ähnlich. Beide Verfahren verwenden 512-Bit-Blöcke, Kettenvariablen der Länge 32 Bit und vier Runden in der Kompressionsfunktion. Die Funktionsweise von MD5 ist daher am einfachsten zu beschreiben, wenn wir uns die Unterschiede zu SHA-1 anschauen:

- MD5 liefert einen Hashwert der Länge 128 Bit (bei SHA-1 sind es 160 Bit). Deshalb gibt es auch nur vier Kettenvariablen.
- MD5 begnügt sich mit 16 Operationen pro Runde. Bei SHA-1 sind es 20.
- Die Bit-Operationen bei MD5 sind etwas anders aufgebaut als bei SHA-1.

MD5 ist zwar aus historischen Gründen immer noch weit verbreitet, doch in neueren Implementierungen wird das Verfahren kaum noch verwendet. Schuld daran ist zum einen die Länge des Hashwerts: 128 Bit gelten heutzutage als untere Grenze, wünschenswert sind mindestens 160 Bit. Darüber hinaus gibt es inzwischen mehrere beunruhigende Kryptoanalyse-Ergebnisse. So fand 1996 der deutsche Kryptograf Hans Dobbertin vom Bundesamt für Sicherheit in der Informationstechnik (BSI) eine Angriffsmethode gegen MD5 [Dobber]. Es gelang ihm, bei einer MD5-Variante Kollisionen zu finden, die sich nur durch geänderte Initialisierungswerte der Kettenvariablen vom Original unterschieden. In den Jahren nach 2004 ging es dann Schlag auf Schlag. Zunächst gelang es der chinesischen Kryptografin Xiaoyun Wang, die auch SHA-1 gebrochen hatte, zusammen mit einigen Kollegen, erstmals MD-5-Kollisionen zu generieren [WaFeLY]. Inzwischen ist es sogar möglich, Kollisionen auf einem gewöhnlichen PC innerhalb von Stunden zu berechnen [Klima]. MD5 ist damit endgültig tot.

14.2.5 RIPEMD-160

Die lange Zeit wichtigste Alternative zu SHA-1 war eine in Europa entwickelte Hashfunktion namens **RIPEMD-160** [BoDoPr]. Diese ist unter Mitwirkung des deutschen Kryptografen Hans Dobbertin entstanden, der auch den beschriebenen Angriff auf MD5 entdeckte. RIPEMD-160 ist die Weiterentwicklung einer Hashfunktion namens RIPEMD, die von einem gänzlich anderen Kryptografen-Team entwickelt wurde. RIPEMD wiederum ist wie SHA-1 und MD5 eine Weiterentwicklung von MD4.

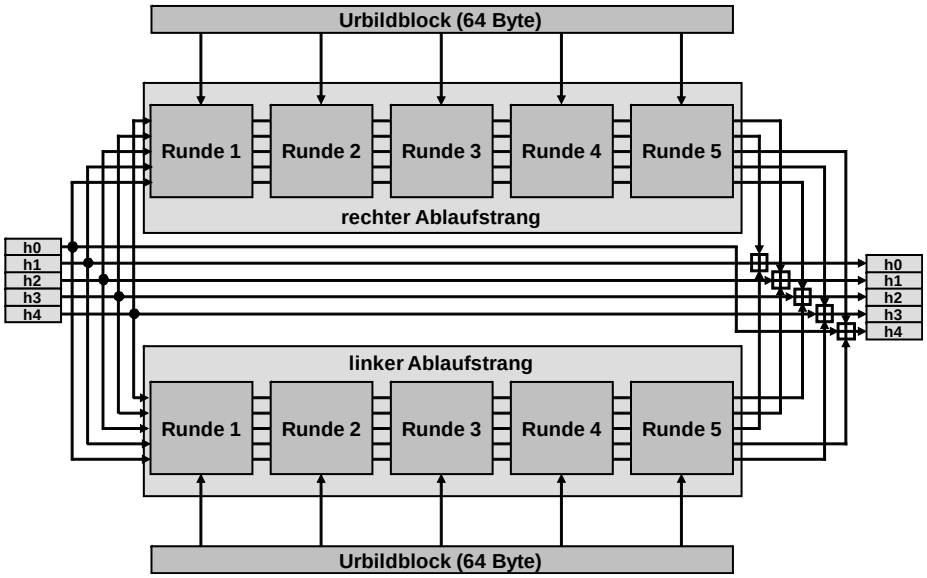


Abb. 14–4 *Die Kompressionsfunktion von RIPEMD-160 arbeitet in zwei parallelen Ablaufsträngen.*

Hans Dobbertin erhielt 1994 als damaliger Mitarbeiter des BSI die Aufgabe, die Sicherheit von RIPEMD zu überprüfen, und fand dabei einige Schwachstellen. Daraufhin machte er sich zusammen mit Antoon Bosselaers und Bart Preneel an die Arbeit, das Verfahren zu verbessern. Auf diese Weise entstand RIPEMD-160. Inzwischen hat sich dieses Verfahren etabliert und ist damit eines der wenigen Krypto-Verfahren mit Praxisrelevanz, an dessen Entwicklung ein Deutscher beteiligt war.

Funktionsweise von RIPEMD-160

Da RIPEMD-160 wie SHA-1 seine Wurzeln in MD4 hat, haben die beiden Verfahren einige Gemeinsamkeiten. Wie SHA-1 arbeitet RIPEMD-160 mit 512-Bit-Blöcken und generiert einen 160-Bit-Hashwert. Auch das Padding am Anfang ist dasselbe. Zudem arbeitet RIPEMD-160 genauso wie SHA-1 mit einer Kompressionsfunktion und mit fünf Kettenvariablen (*A*, *B*, *C*, *D* und *E*). Die Kompressionsfunktion von RIPEMD-160 besteht aus zwei voneinander unabhängigen Ablaufsträngen, die am Ende miteinander verknüpft werden (siehe Abbildung 14–4). Jeder Ablaufstrang ist in fünf Runden gegliedert. Jede der fünf Runden sieht jeweils 16 Teilrunden vor. Zur Erinnerung: Bei SHA-1 sind es vier Runden mit je 20 Teilrunden, wobei es nur einen Ablaufstrang gibt. Wie in Abbildung 14–5 ersichtlich, geht in jede RIPEMD-160-Teilrunde eine Konstante ein. Darüber hinaus gibt es eine Funktion *g*. Konstanten und Funktion im oberen Ablaufstrang sind wie folgt definiert:

	Funktion *g* für oberen Ablaufstrang	Konstante *K* für oberen Ablaufstrang
Runde 1	$B \oplus C \oplus D$	00000000
Runde 2	$(B \wedge C) \vee (\neg B \wedge D)$	5A827999
Runde 3	$(B \wedge \neg C) \oplus D$	6ED9EBA1
Runde 4	$(B \wedge D) \vee (B \wedge \neg D)$	8F1BBCDC
Runde 5	$B \oplus (C \vee \neg D)$	A953FD4E

Wie in der Abbildung zu sehen, gibt es in jeder Teilrunde eine Linksrotation um *r* Einheiten. Welche Werte *r* in der jeweiligen Runde nacheinander annimmt, zeigt die folgende Liste:

- *Runde 1*: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
- *Runde 2*: 7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8
- *Runde 3*: 3, 10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13, 11, 5, 12
- *Runde 4*: 1, 9, 11, 10, 0, 8, 12, 4, 13, 3, 7, 15, 14, 5, 6, 2
- *Runde 5*: 4, 0, 5, 9, 7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13

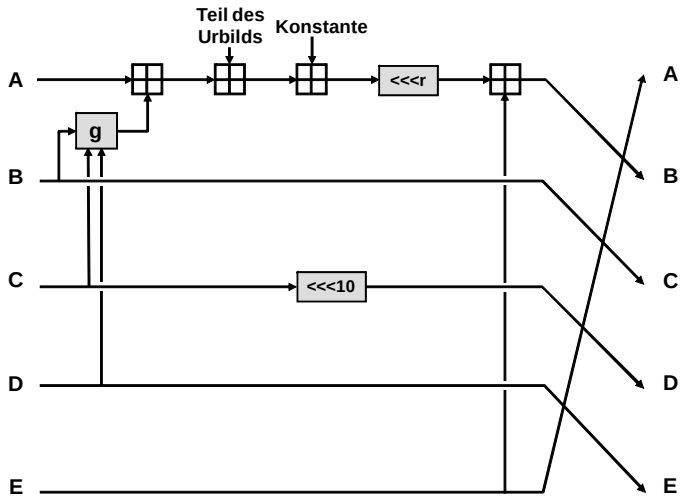


Abb. 14–5 *Jede RIPEMD-160-Runde sieht 16 dieser Operationen vor. Die Konstante und die Funktion g sind von der jeweiligen Runde abhängig.*

Ein Ablaufstrang der RIPEMD-160-Kompressionsfunktion verarbeitet insgesamt 80 32-Bit-Blöcke aus einem Urbildblock. Da ein Urbildblock nur aus 16 32-Bit-Blöcken besteht, ist eine Expansion notwendig (ähnlich wie bei SHA-1). Anders als bei SHA-1 gibt es jedoch keine Expansionsformel, sondern eine Expansionstabelle. Nummeriert man die Urbildblöcke von 0 bis 15, dann werden im oberen

Ablaufstrang die folgenden 80 Urbildblöcke in der genannten Reihenfolge eingesetzt:

- *Runde 1*: 11, 14, 15, 12, 5, 8, 7, 9, 11, 13, 14, 15, 6, 7, 9, 8
- *Runde 2*: 7, 6, 8, 13, 11, 9, 7, 15, 7, 12, 15, 9, 11, 7, 13, 12
- *Runde 3*: 11, 13, 6, 7, 14, 9, 13, 15, 14, 8, 13, 6, 5, 12, 7, 5
- *Runde 4*: 11, 12, 14, 15, 14, 15, 9, 8, 9, 14, 5, 6, 8, 6, 5, 12
- *Runde 5*: 9, 15, 5, 11, 6, 8, 13, 12, 5, 12, 13, 14, 11, 8, 5, 6

Der untere Ablaufstrang von RIPEMD-160 ist ähnlich wie der obere aufgebaut. Folgende Festlegungen gelten (beachten Sie, dass *g* gleich wie im oberen Ablaufstrang definiert ist, nur in umgekehrter Reihenfolge):

	Funktion *g* für unteren Ablaufstrang	**Konstante *K* für unteren Ablaufstrang**
Runde 1	$B \oplus (C \vee \neg D)$	50A28BE6
Runde 2	$(B \wedge \neg C) \oplus D$	5C4DD124
Runde 3	$(B \wedge C) \vee (\neg B \wedge D)$	6D703EF3
Runde 4	$(B \wedge D) \vee (B \wedge \neg D)$	7A6D76E9
Runde 5	$B \oplus C \oplus D$	00000000

Die Zahl *r* nimmt im unteren Ablaufstrang folgende Werte an:

- *Runde 1*: 5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12
- *Runde 2*: 6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2
- *Runde 3*: 15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13
- *Runde 4*: 8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14
- *Runde 5*: 12, 15, 10, 4, 1, 5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11

Die Datenexpansion im unteren Ablaufstrang sieht vor, dass die Urbildblöcke 0 bis 15 in folgender Reihenfolge insgesamt 80 Mal verwendet werden:

- *Runde 1*: 8, 9, 9, 11, 13, 15, 15, 5, 7, 7, 8, 11, 14, 14, 12, 6
- *Runde 2*: 9, 13, 15, 7, 12, 8, 9, 11, 7, 7, 12, 7, 6, 15, 13, 11
- *Runde 3*: 9, 7, 15, 11, 8, 6, 6, 14, 12, 13, 5, 14, 13, 13, 7, 5
- *Runde 4*: 15, 5, 8, 11, 14, 14, 6, 14, 6, 9, 12, 9, 12, 5, 15, 8
- *Runde 5*: 8, 5, 12, 9, 12, 5, 14, 6, 8, 13, 6, 5, 15, 13, 11, 11

Außer RIPEMD-160 gibt es noch RIPEMD-128, RIPEMD-256 und RIPEMD-320 (mit den naheliegenden Hashwert-Längen). Im Gegensatz zur SHA-Familie bieten die RIPEMD-Varianten mit längerem Hashwert keine zusätzlichen krypto-

grafischen Finessen und sind daher nur zu empfehlen, wenn eine Hashfunktion mit mehr Ausgabe-Bits benötigt wird.

Bewertung von RIPEMD-160

RIPEMD-160 ist zweifellos eine gute Alternative zu SHA-2. Das Verfahren ist vergleichsweise gut untersucht und hat bisher keine Schwächen offenbart. Auch die aus China stammenden Angriffe auf SHA-1 und MD5 lassen sich bisher nicht auf RIPEMD-160 ausdehnen. Darüber hinaus bietet das Verfahren einen Datendurchsatz, der nur etwa 15 Prozent unter dem von SHA-1 liegt, was in der Praxis hinnehmbar ist. Allerdings wird RIPEMD-160 zweifellos an Bedeutung verlieren, wenn sich SHA-3 (Keccak) in den kommenden Jahren durchsetzen wird.

14.3 SHA-3 (Keccak)

Als im Jahr 2004 der oben beschriebene Angriff auf SHA-1 bekannt wurde, zeigten sich viele IT-Sicherheitsexperten besorgt. Zwar gab es mit den SHA-2-Algorithmen und RIPEMD-160 einige als sicher geltende Alternativen, doch diese ähnelten SHA-1 in vielerlei Hinsicht, weshalb man damit rechnen musste, dass auch sie Schwächen offenbaren würden. Andere Verfahren, wie Tiger oder WHIRLPOOL (von diesen wird noch die Rede sein), waren noch zu wenig untersucht, um als sicher zu gelten. Weitere Alternativen gab es kaum. Viele forderten daher die US-Standardisierungsbehörde NIST dazu auf, einen Wettbewerb nach Vorbild des AES-Wettbewerbs zu veranstalten, um eine neue, vertrauenswürdige kryptografische Hashfunktion zu finden. Tatsächlich kam das NIST diesem Wunsch nach. Der Wettbewerb begann im Jahr 2008. Gefordert war ein Verfahren, das Hashwerte der Länge 224, 256, 384 und 512 Bit liefert – eine Hashwert-Länge von 160 Bit, wie sie von SHA-1 unterstützt wird, war optional. Das Siegerverfahren sollte unter dem Namen **SHA-3** standardisiert werden.

Das Interesse am SHA-3-Wettbewerb war groß. Bis zum Ende der Einreichungsfrist am 31. Oktober 2008 gingen beim NIST nicht weniger als 64 Algorithmen ein (beim AES-Wettbewerb waren es nur 15 gewesen). Im Vorfeld des SHA-3-Wettbewerbs wurden damit mehr kryptografische Hashfunktionen veröffentlicht als in der gesamten vorherigen Geschichte dieser Verfahren zusammen. Auch zwei Kandidaten aus dem deutschsprachigen Raum waren dabei: *Twister*, das von Studenten der Universität Weimar entwickelt wurde, und *MeshHash* des Gießeners Björn Fay. Zudem waren einige Kryptografen aus Deutschland, Österreich und der Schweiz als Mitglieder internationaler Teams mit dabei. Dazu gehörte insbesondere der Weimarer Professor Stefan Lucks, der zusammen mit Bruce Schneier und anderen das Verfahren *Skein* einreichte (siehe Abschnitt 14.4.5).

Von den 64 Einreichungen ließ das NIST nur 51 zur ersten Wettbewerbsrunde zu – die anderen hatten offensichtliche Mängel. Von diesen 51 Kandidaten

wählte das NIST Ende 2009 14 für die zweite Runde aus. 16 der Ausgeschiedenen wiesen Sicherheitsmängel auf, 11 weitere blieben aus anderen Gründen auf der Strecke – teilweise, weil es zu wenige Untersuchungen durch Experten gegeben hatte. Die restlichen zehn Ausgeschiedenen wurden von den Einreichern selbst für untauglich erklärt. Zu letzterer Gruppe gehörte auch das als Mitfavorit gestartete Verfahren *MD6* von Ron Rivest. Im Gegensatz zu MD2, MD4 und MD5 erlangte MD6 daher keine praktische Bedeutung.

Aus den 14 Verfahren der zweiten Runde wählte das NIST fünf Finalisten aus. Diese hießen **BLAKE**, **Grøstl**, **JH**, **Keccak** und **Skein**. Diese Verfahren hinterließen zwar einen guten Eindruck, doch keines davon konnte (insbesondere, was die Performanz anbelangte) die SHA-2-Algorithmen wirklich in den Schatten stellen. Dadurch kam die Frage auf, ob es überhaupt sinnvoll war, einen neuen Standard zu schaffen. Das NIST entschied sich trotz dieses Einwands dafür, einen Sieger auszuwählen. SHA-3, so die Argumentation, sollte die SHA-2-Familie nicht ersetzen, sondern eine Alternative darstellen. Die Überlegung lautete: Wenn SHA-2 geknackt wird, dann kann man auf SHA-3 umstellen – oder umgekehrt. Am 2. Oktober 2012, also auf den Tag genau 12 Jahre nach Verkündung des AES-Gewinners, wurde der Sieger verkündet. Er hieß Keccak.

Keccak wurde von einem Vierer-Team entwickelt, dem Guido Bertoni, Joan Daemen, Michaël Peeters und Gilles Van Assche angehörten. Alle vier sind für Industrieunternehmen in Europa aktiv. Joan Daemen hatte zusammen mit Vincent Rijmen bereits den AES-Sieger Rijndael entwickelt – er kann also für sich in Anspruch nehmen, die beiden wichtigsten Algorithmen-Wettbewerbe der Kryptografie-Geschichte gewonnen zu haben. Obwohl sowohl AES als auch SHA-3 US-Standards sind, stammen die Entwickler beider Sieger-Verfahren aus Europa.

Wie schon der AES-Wettbewerb entwickelte sich auch der SHA-2-Wettbewerb zu einer gelungenen Veranstaltung. Die Entscheidungen des NIST waren nachvollziehbar, es gab keine unnötige Geheimniskrämerei. Durch die große Zahl der Einreichungen erhielt jedoch nicht jeder Teilnehmer die Aufmerksamkeit, die eigentlich wünschenswert gewesen wäre. Einige Verfahren schieden daher nur deshalb aus, weil sich niemand mit einer Kryptoanalyse beschäftigen wollte oder weil bereits ein ähnlich aufgebauter Algorithmus gesetzt war. Die von namhaften Kryptografen entwickelten Verfahren hatten dadurch klare Vorteile. Etwas unglücklich ist meiner Meinung nach die Wahl des Namens »SHA-3«. Diese Bezeichnung legt nahe, dass das neue Verfahren den Vorgängern SHA-1 und SHA-2 ähnelt, was natürlich nicht der Fall ist. Vielleicht wäre »Advanced Hash Algorithm« (AHA) die bessere Wahl gewesen. Ich könnte mir vorstellen, dass SHA-3 aus diesem Grund auch zukünftig von vielen als »Keccak« bezeichnet werden wird – im Gegensatz zum AES, dessen ursprünglicher Name »Rijndael« heute kaum noch verwendet wird.

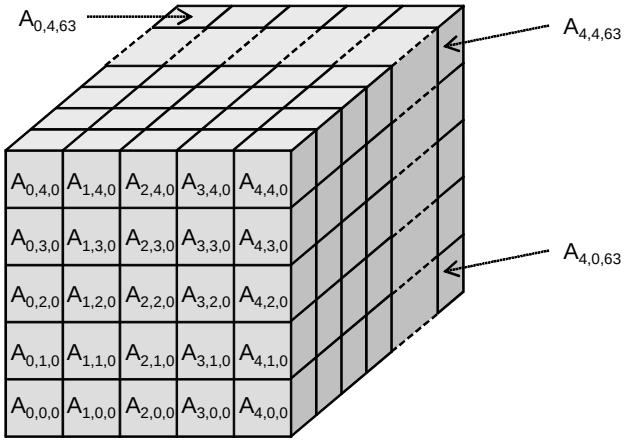


Abb. 14–6 *Der Status von Keccak ist ein dreidimensionales Array bestehend aus 5×5×64 Bits.*

14.3.1 Funktionsweise von Keccak

Keccak nutzt ein dreidimensionales Array von 5×5×64 Bits als Status, was eine Statusgröße von 1.600 Bit zur Folge hat. Ein senkrechter Fünf-Bit-Block im Status heißt Spalte, ein waagerechter (von links nach rechts) Reihe. Ein senkrechter 64-Bit-Block (von vorne nach hinten) wird als Wort bezeichnet. Die Differenz zwischen der Status- und der Blocklänge (erste ist immer größer als letztere) wird als *Kapazität* bezeichnet. Je größer die Kapazität, desto höher ist (mutmaßlich) die Sicherheit des Verfahrens. Die Hashwert-Länge beträgt per Definition die Hälfte der Kapazität (es ist jedoch einfach möglich, längere oder kürze Hashwerte zu generieren).

Wenn wir einen Hashwert der Länge 224 Bit haben wollen, dann müssen wir die Kapazität auf das Doppelte, also auf 448 festlegen. Das wiederum bedeutet, dass die Blocklänge 1.152 Bit betragen muss, da die Blocklänge und die Kapazität zusammen so groß wie die Statuslänge sein müssen. Das Urbild, das gehasht werden soll, muss dementsprechend in 1.152-Bit-Blöcke aufgeteilt werden. Das Padding funktioniert wie folgt: Man füllt den Leerteil des letzten Blocks mit einer Eins, so vielen Nullen wie notwendig und anschließend mit einer weiteren Eins auf. Wenn also im letzten Block sieben Bits übrig sind, dann wird mit 1000001 aufgefüllt. Das kürzeste mögliche Füllmuster ist 11. Sind weniger als zwei Bits übrig, dann muss ein weiterer Block angehängt werden.

Das Schwamm-Prinzip

Das Funktionsprinzip von Keccak wird von den Entwicklern als **Schwamm** (**Sponge**) bezeichnet. Dahinter steckt die Vorstellung, dass dieses Konstrukt ein beliebig langes Urbild »aufsaugt«, um anschließend zur Bildung des Hashwerts

»ausgewrungen« zu werden. Der Aufbau des Schwamms wird in Abbildung 14–7 beschrieben (wir gehen von einer Statusgröße von 1.600 Bit aus, obwohl das Schwammprinzip auch für andere Größen funktioniert). Der (mit Nullbits initialisierte) Status wird jeweils mit einem Urbildblock exklusiv-oder-verknüpft. Dabei bleibt eine Anzahl von Bits unberührt, die der Kapazität entsprechen (im von uns betrachteten Fall sind es 448 unberührte Bits). Nach der Exklusiv-oder-Verknüpfung wird der Status einer Kompressionsfunktion zugeführt. Nach dem letzten Block werden (oben beginnend) so viele Bits aus dem Status ausgelesen, wie es der Hashwert-Länge entspricht. In unserem Fall sind dies 224. Diese Bits werden als Hashwert verwendet.

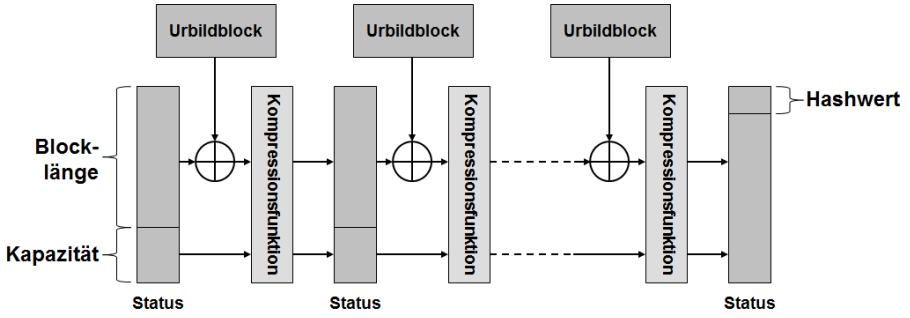


Abb. 14–7 *Das Funktionsprinzip von Keccak wird auch als Schwamm bezeichnet. Die Länge des Hashwerts entspricht der Hälfte der Kapazität.*

Es ist natürlich auch möglich, am Ende mehr als 224 Bits aus dem Status zu lesen und als Hashwert zu verwenden (das Maximum liegt bei 1.600 Bits). Außerdem kann man die Kompressionsfunktion auf den letzten Status ein weiteres Mal (dieses Mal ohne Einbringen eines Klartextblocks) anwenden, wodurch man 1.600 weitere Bits gewinnt. Durch weitere Anwendungen der Kompressionsfunktion lässt sich der Hashwert auf eine beliebige Länge ausdehnen. Allerdings gilt: Je mehr Material für den Hashwert verwendet wird, desto geringer die Sicherheit. In den SHA-3-Standard werden voraussichtlich nur die folgenden vier Kombinationen eingehen:

Hashwert-Länge	Blocklänge	Kapazität
224	1152	448
256	1088	512
384	832	768
512	576	1024

Die Kompressionsfunktion

Eine Funktion kann als Kompressionsfunktion für den Schwamm genutzt werden, wenn sie Bitfolgen in der Größe des Status (in unserem Fall 1.600 Bit) verarbeitet und ausgibt. Keccak ist durch das Schwamm-Prinzip sowie durch eine Kompressionsfunktion definiert, das dieses nutzt. Diese Kompressionsfunktion läuft in 24 Runden ab. Jede Runde besteht aus fünf Teilrunden, die mit den griechischen Buchstaben θ, ρ, π, χ und ι (theta, rho, pi, chi und iota) bezeichnet werden. Diese laufen wie folgt ab (A ist der Status):

1. In **Teilrunde θ** wird von allen 320 Spalten des Status das Paritätsbit berechnet und in regelmäßiger Form mit benachbarten Spalten exklusiv-oder-verknüpft. Die Formel lautet:

   $A_{i,j,k} = A_{i,j,k} \oplus \mathrm{par}(A_{0..4,j-1,k}) \oplus \mathrm{par}(A_{0..4,j+1,k-1})$
2. In **Teilrunde ρ** wird jedes der 25 Wörter des Status nach hinten rotiert. Um wie viele Einheiten rotiert wird, ergibt sich aus folgender Tabelle (die Zahlen sind modulo 64 zu nehmen).

	x=0	x=1	x=2	x=3	x=4
y=0	0	1	190	28	91
y=1	36	300	6	55	276
y=2	3	10	171	153	231
y=3	105	45	15	21	136
y=4	210	66	253	120	78

3. In **Teilrunde π** werden die 25 Wörter des Status nach folgender Formel permutiert:

   $A_{3i+2j,i,0..63} = A_{i,j,0..63}$
4. In **Teilrunde χ** werden jeweils die Bits einer Reihe wie folgt verknüpft:

   $A_{i,j,k} = A_{i,j,k} \oplus \neg A_{i,j+1,k} \wedge A_{i,j+2,k}$
5. In **Teilrunde ι** wird eine Konstante mit dem Status exklusiv-oder-verknüpft, die sich mit jeder Runde ändert. Die Konstante ist, wie der Status, ein $5 \times 5 \times 64$-Bit-Array. Nur sieben der 1.600 Bits der Konstante können einen von Null verschiedenen Wert haben. Diese sieben Bits liegen alle innerhalb des unteren linken Worts. Die Konstante wird mithilfe eines linearen Feedback-Shift-Registers generiert, was uns aber an dieser Stelle nicht interessieren soll. Der Wert der Konstante ist in folgender Liste in Hexadezimalschreibweise gegeben (man muss sich Status und Konstante hierbei als 1.600-Bit-Folge der Form $A_{0,0,0}$, $A_{1,0,0}$, $A_{2,0,0}$, ..., $A_{4,4,62}$, $A_{4,4,63}$ vorstellen):

Runde 1: 00 00 00 00 00 00 00 01
Runde 2: 00 00 00 00 00 00 80 82
Runde 3: 80 00 00 00 00 00 80 8A
Runde 4: 80 00 00 00 80 00 80 00
Runde 5: 00 00 00 00 00 00 80 8B
Runde 6: 00 00 00 00 80 00 00 01
Runde 7: 80 00 00 00 80 00 80 81
Runde 8: 80 00 00 00 00 00 80 09
Runde 9: 00 00 00 00 00 00 00 8A
Runde 10: 00 00 00 00 00 00 00 88
Runde 11: 00 00 00 00 80 00 80 09
Runde 12: 00 00 00 00 80 00 00 0A
Runde 13: 00 00 00 00 80 00 80 8B
Runde 14: 80 00 00 00 00 00 00 8B
Runde 15: 80 00 00 00 00 00 80 89
Runde 16: 80 00 00 00 00 00 80 03
Runde 17: 80 00 00 00 00 00 80 02
Runde 18: 80 00 00 00 00 00 00 80
Runde 19: 00 00 00 00 00 00 80 0A
Runde 20: 80 00 00 00 80 00 00 0A
Runde 21: 80 00 00 00 80 00 80 81
Runde 22: 80 00 00 00 00 00 80 80
Runde 23: 00 00 00 00 80 00 00 01
Runde 24: 80 00 00 00 80 00 80 08

Nur die vierte Teilrunde sieht eine nichtlineare Operation vor. Sie ist für die Konfusion zuständig. Alle anderen Runden sind linear und sorgen somit für Diffusion. Die fünfte Teilrunde ist die einzige, deren Ablauf sich (durch den unterschiedlichen Wert der Konstante) von Runde zu Runde unterscheidet.

Bewertung von Keccak

Bei Redaktionsschluss dieses Buchs war Keccak gerade einmal seit zwei Monaten zum SHA-3 gekürt (vermutlich ist dieses Buch das erste überhaupt, in dem Keccak beschrieben wird). Praktische Erfahrungen damit gab es zu diesem Zeitpunkt noch nicht.

14.4 Weitere Hashfunktionen

Vor dem SHA-3-Wettbewerb gab es nur wenige Alternativen zu den mit MD4 verwandten Verfahren (zu diesen gehört auch die SHA-Familie und RIPEMD-160). Zwei davon schauen wir uns im Folgenden an.

14.4.1 Tiger

Tiger ist eine kryptografische Hashfunktion, die 1995 von den beiden bekannten Kryptografen Ross Anderson und Eli Biham veröffentlicht wurde [AndBih]. Tiger entstand etwa zur gleichen Zeit wie RIPEMD-160, ist aber im Gegensatz zu diesem keine Weiterentwicklung eines bestehenden Verfahrens. Der Name wurde von den beiden Erfindern gewählt, weil sie eine besonders starke und schnelle kryptografische Hashfunktion entwickeln wollten. Als weiteres Designziel geben die Autoren an, dass Tiger auf 64-Bit-Prozessoren besonders performant laufen soll.

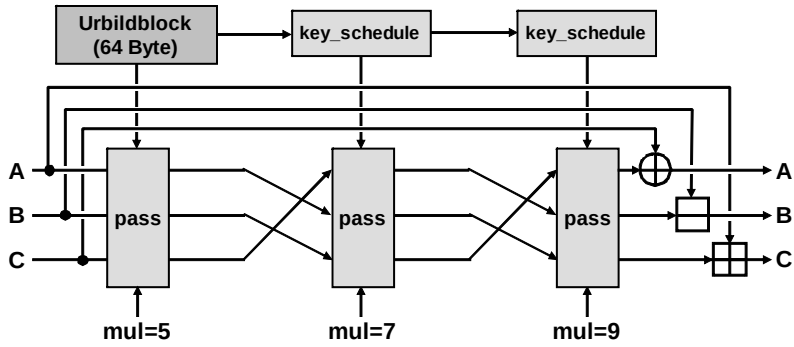


Abb. 14–8 *Die Kompressionsfunktion von Tiger sieht drei Runden vor, die als pass bezeichnet werden.*

Funktionsweise von Tiger

Tiger verarbeitet Blöcke der Länge 512 Bit und liefert in der Standardversion (auch als **Tiger/192** bezeichnet) einen 192-Bit-Hashwert. Daneben gibt es auch die beiden Varianten **Tiger/160** und **Tiger/128**, die bei einer ansonsten gleichen Funktionsweise den Hashwert abschneiden und dadurch auf 160 bzw. 128 Bit verkürzen. Tiger arbeitet mit einem Status bestehend aus drei Kettenvariablen (*A*, *B* und *C*) zu je 64 Bit. Diese Kettenvariablen werden mit den Werten (in Hexadezimalschreibweise) 0123456789ABCDEF, FEDCBA9876543210 und F096A5B4C3B2E187 initialisiert.

Bevor das Verfahren starten kann, ist ein Padding notwendig. Um dieses durchzuführen, benötigt man am Ende des letzten Urbildblocks mindestens 65 leere Bits. Sind diese nicht vorhanden, wird ein weiterer Block angehängt. Anschließend wird in die letzten 64 Bits die Länge des ungepaddeten Urbilds geschrieben, davor steht ein Bit mit dem Wert 1. Alle Bit-Werte zwischen dem Ende des ungepaddeten Urbilds und den 64 letzten Bit werden mit Null-Bits aufgefüllt. Das gepaddete Urbild wird in Blöcke der Länge 512 Bit aufgeteilt und anschließend blockweise einer Kompressionsfunktion zugeführt. Dabei wird ein Block in die acht Teilblöcke $x_0, x_1, ..., x_7$ aufgeteilt, die jeweils 64 Bit umfassen. Der Inhalt der Kettenvariablen nach dem Bearbeiten aller Blöcke bildet den Hashwert.

Das Design der Kompressionsfunktion sieht drei Runden vor, wobei eine Runde als *pass* bezeichnet wird (siehe Abbildung 14–8). In jede Runde geht der Urbildblock mit ein, wobei dieser vor der zweiten und dritten Runde jeweils mit der Funktion *key_schedule* bearbeitet wird. Der Aufbau von key_schedule ist in Abbildung 14–9 dargestellt. Jede Runde gliedert sich in acht Teilrunden (*round*), was in Abbildung 14–10 zu sehen ist. Der Aufbau einer Teilrunde ist aus Abbildung 14–11 ersichtlich. In den Teilrunden kommen vier S-Boxen (s_0, s_1, s_2 und s_3) zum Einsatz, die jeweils 8 Bit auf 8 Bit abbilden. Auf deren Beschreibung verzichte ich an dieser Stelle.

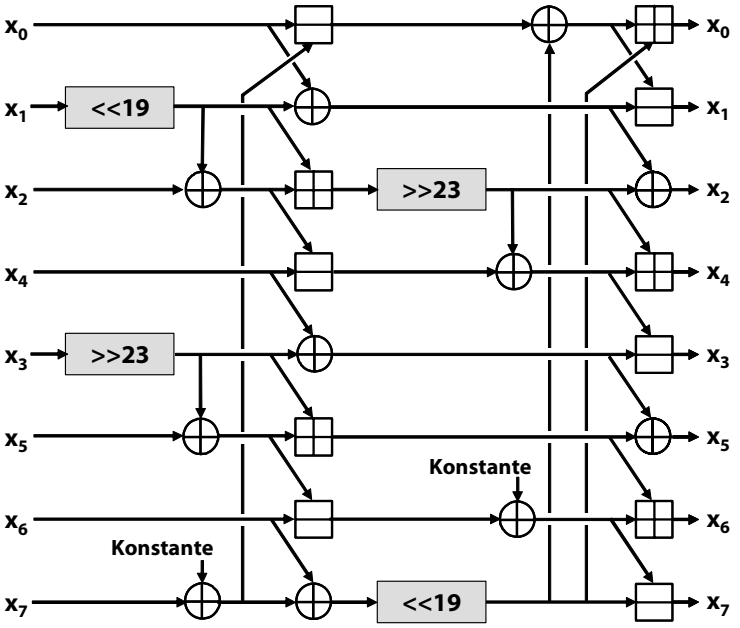


Abb. 14–9 *In der Funktion key_schedule wird der Urbildblock für die weitere Verwendung aufbereitet.*

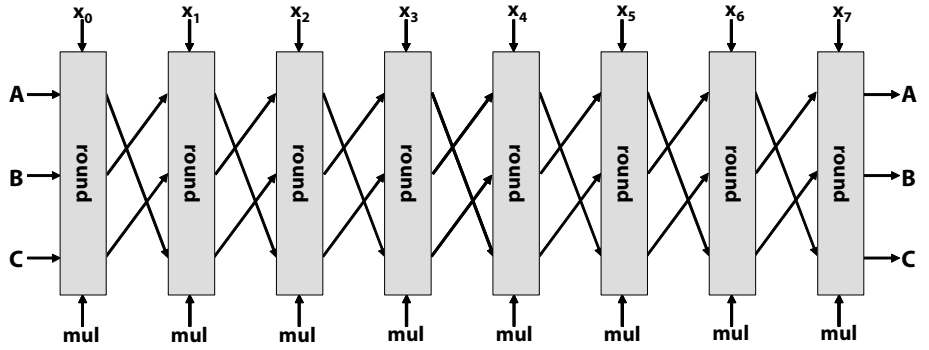


Abb. 14–10 *Jede Tiger-Runde besteht aus acht Teilrunden, die round genannt werden. $x_0, x_1, ..., x_7$ ist ein Urbildblock. Die Variable mul kann die Werte 5, 7 und 9 annehmen.*

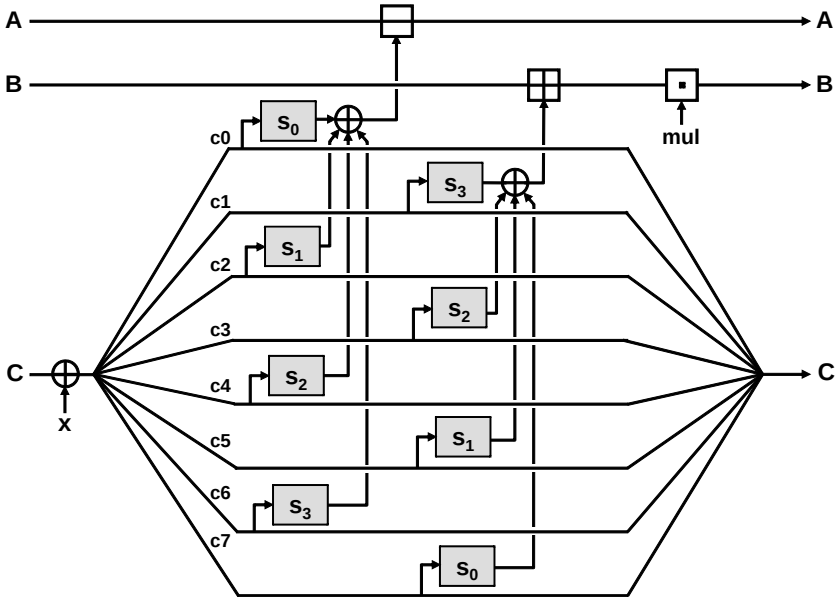


Abb. 14–11 *Eine Teilrunde (round) von Tiger. s_0, s_1, s_2 und s_3 sind S-Boxen.*

Bewertung von Tiger

Tiger ist nicht so weit verbreitet wie SHA-1 oder RIPEMD-160. Das Verfahren ist daher auch noch nicht so gut untersucht. John Kelsey und Stefan Lucks fanden Kollisionen bei Tiger mit reduzierter Rundenzahl, was jedoch keine praktischen Konsequenzen hat [KelLuc]. Tiger ist zweifellos eine interessante Alternative zu SHA-1 und RIPEMD-160, der jedoch der Makel anhaftet, bisher noch nicht gut genug analysiert worden zu sein.

14.4.2 WHIRLPOOL

WHIRLPOOL ist eine noch recht junge kryptografische Hashfunktion, die von AES-Miterfinder Vincent Rijmen zusammen mit dem brasilianischen Kryptografen Paulo S. L. M. Barreto entwickelt wurde [BarRij]. Die beiden Autoren veröffentlichten 2000 eine erste Beschreibung des Verfahrens, die sie in den Folgejahren zweimal überarbeiteten. Die 2003 erschienene dritte Version gilt als die endgültige und soll uns als einzige an dieser Stelle interessieren. Auch in der sonstigen Literatur ist stets die dritte Version gemeint, wenn von WHIRLPOOL die Rede ist.

Funktionsweise von WHIRLPOOL

WHIRLPOOL arbeitet mit 64 Kettenvariablen, die zusammen als *Hashstatus* bezeichnet werden. Jede Kettenvariable besteht aus einem Byte, was insgesamt 512 Bit ergibt. Der Hashwert ist der Inhalt des Hashstatus nach Abarbeitung des Algorithmus. Als Kompressionsfunktion verwendet WHIRLPOOL ein speziell für diesen Zweck konstruiertes symmetrisches Verschlüsselungsverfahren, das schlicht als *W* bezeichnet wird. W verschlüsselt Blöcke der Länge 512 Bit mit einem 512-Bit-Schlüssel.

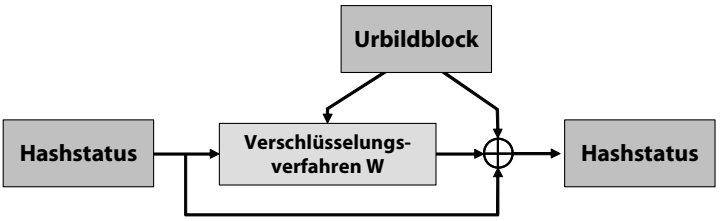


Abb. 14–12 *Die Kompressionsfunktion von WHIRLPOOL ist ein Verschlüsselungsverfahren namens W, das dem AES ähnelt.*

Der Aufbau von WHIRLPOOL ist in Abbildung 14–12 zu erkennen. Das Urbild wird in Blöcke der Länge 512 Bit aufgeteilt ($m1$, $m2$, $m3$, ...). Nacheinander wird jeweils der aktuelle Inhalt des Hashstatus mit W verschlüsselt, wobei der aktuelle Urbildblock als Schlüssel dient. Eine Exklusiv-oder-Verknüpfung des aktuellen Urbildblocks, der aktuellen Kettenvariablen und des Verschlüsselungsergebnisses liefern den neuen Inhalt der Kettenvariablen. Als mathematische Formel sieht dies wie folgt aus (H_i ist der Hashstatus):

$$H_i = (W_{mi-1}(H_{i-1})) \oplus H_{i-1} \oplus mi\text{-}1$$

Der Hashstatus wird mit einer Folge von 512 Null-Bits initialisiert. Es gilt also H_0=00...00. Bevor das Verfahren auf diese Weise zum Einsatz kommen kann, ist ein Padding notwendig. Um dieses durchzuführen, benötigt man am Ende des letzten Urbildblocks mindestens 257 leere Bit. Sind diese nicht vorhanden, wird ein weiterer Block angehängt. Anschließend wird in die letzten 256 Bit die Länge des ungepaddeten Urbilds geschrieben, davor steht ein Bit mit dem Wert 1. Alle Bit-Werte zwischen dem Ende des ungepaddeten Urbilds und den 257 letzten Bit werden mit Null-Bits aufgefüllt.

Das Verschlüsselungsverfahren W

Der wichtigste Bestandteil von WHIRLPOOL ist das als Kompressionsfunktion verwendete Verschlüsselungsverfahren W, dessen Schlüssel und Bit-Länge jeweils 512 Bit betragen. W wurde speziell für WHIRLPOOL geschaffen und wird mei-

nes Wissens nirgendwo zur Verschlüsselung eingesetzt. So gesehen kann man sich darüber streiten, ob es überhaupt sinnvoll ist, W als Verschlüsselungsverfahren zu bezeichnen. Da die WHIRLPOOL-Erfinder dies tun, möchte ich es an dieser Stelle übernehmen.

Die Funktionsweise von W kann eine gewisse Ähnlichkeit mit dem AES nicht verleugnen. Kein Wunder, denn WHIRLPOOL-Miterfinder Vincent Rijmen ist ja einer der beiden Väter des AES. Den W-Klartext sollte man sich als quadratische Matrix aus 64 Bytes vorstellen, Gleiches gilt für den W-Schlüssel. Das W-Design sieht zehn Runden vor, in die jeweils ein 512-Bit-Subschlüssel eingeht. Jede W-Runde besteht aus vier Schritten:

1. *Nichtlinearer Schritt* (entspricht der SubByte-Funktion des AES): Hier wird jedes der 64 Bytes einer S-Box zugeführt. Diese ist allerdings nicht mit der AES-S-Box identisch, sondern aus einer mehrfach eingesetzten kleineren S-Box sowie aus einer Permutationsfunktion zusammengesetzt.
2. *Permutationsschritt* (entspricht der ShiftRow-Funktion des AES): Hier wird jede Spalte der Matrix um eine bestimmte Einheit rotiert. Man beachte, dass der entsprechende Schritt des AES keine Spalten-, sondern eine Zeilenrotation vornimmt.
3. *Diffusionsschritt* (entspricht der MixColumn-Funktion des AES): Durch eine Matrix-Multiplikation nimmt dieser Schritt eine Durchmischung der Zeilen vor (beim AES werden auf ähnliche Weise die Spalten durchmischt).
4. *Additionsschritt* (entspricht der AddKey-Funktion des AES): In diesem Schritt wird der Schlüssel mit der Matrix exklusiv-oder-verknüpft.

Während beim AES die letzte Runde einen etwas anderen Ablauf hat (zusätzliches AddRoundKey statt MixColumns), sehen bei W alle Runden des Verfahrens dieselben vier Schritte vor. Da W in zehn Runden abläuft, benötigen wir zehn Subschlüssel K_1, ..., K_{10}. Die Schlüsselaufbereitung erfolgt mithilfe des W-Verfahrens. Ist R eine W-Runde, dann gilt: $K_1 = R_{c1}(K)$, $K_2 = R_{c2}(K_1)$, ..., $K_{10} = R_{c10}(K_9)$. Die als Schlüssel verwendeten Variablen $c1$, $c2$, ... sind jeweils 512-Bit-Werte (8×8-Byte-Matrizen), die aus der S-Box abgeleitet werden.

Bewertung von WHIRLPOOL

WHIRLPOOL ist neben Tiger die bedeutendste Neuentwicklung im Bereich der kryptografischen Hashfunktion in den Jahren vor dem SHA-3-Wettbewerb. Bisher sind keine Schwächen des Verfahrens bekannt. Allerdings ist es bisher noch zu wenig untersucht, um als sicher gelten zu können.

14.4.3 Weitere kryptografische Hashfunktionen

Es gibt noch weitere kryptografische Hashfunktionen, die in der Literatur vorgeschlagen wurden. Allerdings ist deren Zahl deutlich geringer als etwa die der symmetrischen Verschlüsselungsverfahren. Hier ein Überblick im Schnelldurchlauf:

- **MD2** stammt wie MD4 und MD5 aus der Werkstatt von Ron Rivest [RFC1319]. Trotz der Namensähnlichkeit ist MD2 anders aufgebaut als die beiden letztgenannten Verfahren. MD2 verarbeitet 128-Bit-Blöcke und erzeugt einen 128-Bit-Hashwert. Anfang der neunziger Jahre erreichte das Verfahren eine gewisse Verbreitung, wurde dann jedoch von anderen verdrängt. Daran waren keine Sicherheitsmängel, sondern die geringe Geschwindigkeit schuld. MD2 erreicht seine höchste Performanz auf Acht-Bit-Prozessoren, die in den Neunzigern aus der Mode kamen.
- **N-Hash**: N-Hash ist eine kryptografische Hashfunktion, die dem Verschlüsselungsverfahren FEAL ähnelt und vom gleichen Entwicklerteam stammt [Schn96]. N-Hash gilt wie FEAL als unsicher.
- **Haval**: Haval ist eine weitere MD4-Weiterentwicklung [Schn96]. Sie spielt in der Praxis keine Rolle.
- **Snefru**: Diese kryptografische Hashfunktion stammt von Krypto-Pionier Ralph Merkle [Schn96]. Sie sieht eine variable Rundenzahl vor. Je nach Rundenzahl ist Snefru entweder zu langsam oder unsicher.

Einige weitere kryptografische Hashfunktionen entstanden in den Jahren vor dem SHA-3-Wettbewerb. Dazu gehört **RadioGatún** [BeDaAP], das ich in der letzten Ausgabe dieses Buchs ausführlich vorgestellt habe. RadioGatún war der Vorläufer von SHA-3-Sieger Keccak und stammt vom gleichen Entwicklerteam. Weitere Verfahren, die etwa zur gleichen Zeit entstanden, sind **LASH** [BePSSS], **FORK-256** [HoSHLM], **DHA-256** [LCKLHS] und **VSH** [CoLeSt]. Im Übrigen gab es auch MD1 und MD3, doch diese Verfahren brachten es nicht zur Praxisreife.

14.4.4 Hashfunktionen aus Verschlüsselungsverfahren

Eine einfache Möglichkeit, eine kryptografische Hashfunktion zu konstruieren, besteht in der Verwendung einer Blockchiffre. Oder anders formuliert: Neben den in Abschnitt 19.1 beschriebenen Betriebsarten für Blockchiffren gibt es weitere, die nicht das Verschlüsseln, sondern das Hashen zum Ziel haben. Einige Beispiele schauen wir uns im Folgenden an. Für deren Erklärung verwende ich folgende Bezeichnungen:

- $(U_0, U_1, ..., U_{n-1})$ ist das Urbild, U bezeichnet einen beliebigen Urbildblock.
- A ist der Status bzw. die Kettenvariable. A wird mit einer Konstanten initialisiert, die vor der Verwendung des Verfahrens festgelegt werden muss. Der Wert, der am Ende in A steht, ist der Hashwert.
- Die verwendete Blockchiffre wird mit E bezeichnet. Eine Verschlüsselung der Nachricht m mit dem Schlüssel k wird als $E_k(m)$ notiert. K ist eine Konstante, die als Schlüssel verwendet wird.

Die folgende Tabelle nennt vier Varianten (beachten Sie, dass die Hashwert-Länge in allen Fällen der Blocklänge von E entspricht):

Variante	Kompressionsfunktion	Blocklänge der Hashfunktion
1	$A = A \oplus E_K(U)$	Blocklänge von E
2	$A = E_U(A)$	Schlüssellänge von E
3	$A = E_A(U) \oplus U$	Blocklänge von E
4	$A = E_A(U) \oplus A \oplus U$	Blocklänge von E

Variante 1

Variante 1 sieht vor, dass Alice das Urbild im CBC-Modus mit E verschlüsselt. Als Schlüssel verwendet sie eine öffentlich bekannte Konstante. Der Initialisierungswert des Status entspricht dem Initialisierungsvektor, der letzte Geheimtextblock ist der Hashwert. Diese Variante ist zwar äußerst naheliegend, dafür aber auch äußerst unsicher. Dies liegt daran, dass ein Verschlüsselungsverfahren mit bekanntem Schlüssel per Definition einfach umkehrbar ist. Mallory kann also zu jedem beliebigen Hashwert ohne große Mühe ein passendes Urbild berechnen. Beispielsweise kann er U_1 bis U_{n-1} nach seinen Vorstellungen wählen und U_0 entsprechend anpassen.

Variante 2

In Variante 2 wählt Alice eine Konstante und verschlüsselt diese nacheinander mit allen Urbildblöcken als Schlüssel. Diese Variante hat den Nachteil, dass sie gegenüber einer Meet-in-the-Middle-Attacke anfällig ist. Diese funktioniert so:

1. Mallory wählt ein Urbild aus, in dem der Teil $(U_1, ..., U_{n-2})$ seinen Vorstellungen entspricht. Er kann darin etwa eine Staubsauger-Bestellung unterbringen. Den Inhalt von U_0 und U_{n-1} lässt Mallory zunächst offen.
2. Mallory wählt ein beliebiges U_0. Anschließend berechnet er den Hashwert von $(U_0, ..., U_{n-1})$ und trägt diesen in eine Liste ein. Danach erhöht er den Wert von U_0 um 1 und wiederholt den Vorgang. Mallory fährt auf diese Weise fort, wodurch sich die Liste nach und nach füllt.
3. Gleichzeitig mit Schritt 2 nimmt Mallory den gewünschten Hashwert und ein beliebiges U_{n-1}. Dann entschlüsselt er den Hashwert mit U_{n-1} als Schlüssel

und trägt das Ergebnis in eine weitere Liste ein. Danach erhöht er U_{n-1} um 1 und wiederholt den Vorgang. Auch mit diesem Prozedere fährt Mallory fort und füllt auf diese Weise nach und nach die zweite Liste.

4. Mallory führt die Schritte 2 und 3 parallel so lange durch, bis sich irgendwann eine Übereinstimmung in den beiden Listen ergibt. Ist dies der Fall, dann hat Mallory ein passendes Urbild zum Hashwert gefunden. Der erste und der letzte Block dieses Urbilds enthalten irgendwelche bedeutungslosen Werte. Dazwischen steht die Staubsauger-Bestellung oder eine andere von Mallory gewählte Nachricht.

Beträgt die Blocklänge von *E* *n* Bit, dann benötigt Mallory gemäß dem Geburtstagsphänomen im Schnitt etwa $2^{n/2}$ Versuche, bis sich die gewünschte Übereinstimmung ergibt. Den DES (64 Bit Blocklänge) sollte Alice für diese Anwendung also nicht nehmen. Auch der AES (128 Bit Blocklänge) ist nur bedingt geeignet.

Variante 3 und 4

Variante 3 und 4 haben den Vorteil, dass eine Meet-in-the-Middle-Attacke nicht funktioniert. Auch ansonsten gelten sie als sicher. Beide haben die Eigenschaft, dass der Status einerseits die Länge eines Klartextblocks haben muss, andererseits aber auch als Schlüssel verwendet wird. Am besten funktioniert dies, wenn Schlüssellänge und Blocklänge von *E* gleich sind. Ist die Blocklänge größer, dann muss Alice jeweils einen Teil des Blocks abschneiden. Ist die Blocklänge kleiner, wird ein Auffüllen notwendig.

Fazit

Obwohl es also durchaus Möglichkeiten gibt, ein symmetrisches Verschlüsselungsverfahren als kryptografische Hashfunktion einzusetzen, gilt dies nicht als der ideale Weg. Störend ist beispielsweise, dass bei den sicheren Varianten für jeden Block ein anderer Schlüssel zum Einsatz kommt. Dies bedeutet, dass Alice ständig eine neue Schlüsselaufbereitung vornehmen muss. Verfahren mit einer aufwendigen Schlüsselaufbereitung (etwa Blowfish) sind daher gänzlich ungeeignet für diesen Anwendungszweck. Die meisten anderen Algorithmen erzeugen zumindest Hashfunktionen, die langsamer sind als etwa SHA-1.

Auch die Länge des Hashwerts spricht gegen Hashfunktionen aus symmetrischen Verfahren. 160 Bit gelten heute als das Minimum. Die meisten neueren Verschlüsselungsverfahren haben jedoch eine Blocklänge (und damit auch Hashwert-Länge) von 128 Bit. Darüber hinaus haben kryptografische Hashfunktionen andere Sicherheitsziele als Verschlüsselungsverfahren. So funktioniert beispielsweise die lineare Kryptoanalyse bei Hashfunktionen grundsätzlich nicht (die differenzielle dagegen schon). Eine gut designte kryptografische Hashfunktion ist daher stets schneller als eine aus einem Verschlüsselungsverfahren generierte.

14.4.5 Hashfunktionen aus Tweak-Verfahren

In ein Tweak-Verschlüsselungsverfahren geht neben dem Schlüssel noch eine weitere Funktion ein (siehe Abschnitt 10.4). Tweak-Verfahren lassen sich auch zur Konstruktion kryptografischer Hashfunktionen nutzen. Wie das geht, zeigt beispielsweise die kryptografische Hashfunktion **Skein**, die beim SHA-3-Wettbewerb unter die besten fünf kam [FLSWBK]. Zum Entwicklerteam gehörten unter anderem Bruce Schneier und der deutsche Informatik-Professor Stefan Lucks von der Universität Weimar. Skein verwendet das Tweak-Verfahren Threefish (siehe Abschnitt 10.4.2), das speziell für diesen Zweck entwickelt wurde. Die Betriebsart, in der Threefish verwendet wird, heißt *UBI Chaining Mode* (UBI steht für »Unique Block Iteration«). Die Kompressionsfunktion sieht wie folgt aus (*A* ist der Status, *U* ein Urbildblock, *E* das Verschlüsselungsverfahren und *t* der Tweak):

$$A = U \oplus E_{A,t}(U)$$

Der Tweak ändert sich mit jedem Aufruf. Daraus ergibt sich der Vorteil, dass die Kompressionsfunktion für jeden verarbeiteten Block eine andere ist. Man kann sich leicht vorstellen, dass dies Angriffe erschwert. Im Fall von Skein sind im Tweak die Anzahl der bereits verarbeiteten Bytes, die Information, ob es sich um den ersten und/oder den letzten Urbildblock handelt, sowie der sogenannte Aufruftyp enthalten. Der Aufruftyp erklärt sich daraus, dass die Kompressionsfunktion bei Skein nicht nur einmal pro Block aufgerufen wird. Stattdessen erfolgt der erste Aufruf zur Initialisierung, ohne dass Urbildmaterial verarbeitet wird (Aufruftyp *Cfg*). Sind alle Blöcke abgearbeitet (Aufruftyp *Msg*), dann wird die Kompressionsfunktion erneut aufgerufen (Aufruftyp *Out*) – bei längeren Hashwerten sogar mehrmals.

14.5 Schlüsselabhängige Hashfunktionen

Wie Sie in diesem Kapitel erfahren haben, arbeitet eine kryptografische Hashfunktion ohne Schlüssel. Dies ist auch nicht notwendig, da es nicht stört, wenn Abhörer Mallory einen Hashwert berechnen kann. Es gibt jedoch auch Anwendungsfälle, in denen Mallory diese Möglichkeit nicht haben soll. Ein Beispiel: Alice schickt an Bob eine Nachricht, die nicht geheim bleiben muss (Mallory darf sie also lesen); Alice will jedoch, dass die Nachricht von Mallory nicht unbemerkt verändert wird. Alice könnte nun ihre Nachricht (bzw. einen kryptografischen Hashwert davon) digital signieren. Es gibt jedoch noch eine andere Möglichkeit: Alice verwendet einen kryptografischen Hashwert, der nur mithilfe eines Schlüssels berechnet werden kann. Da sie mit Bob schon vor längerer Zeit einen geheimen Schlüssel ausgetauscht hat, kann Empfänger Bob den Hashwert ebenfalls berechnen und so überprüfen, dass die Nachricht unverändert ist. Mallory kann dies jedoch nicht, da er den Schlüssel nicht kennt. Eine kryptografische Hashfunktion, die für die Berechnung eines Hashwerts einen geheimen Schlüssel benö-

tigt, wird als **schlüsselabhängige kryptografische Hashfunktion** oder einfach als **schlüsselabhängige Hashfunktion** bezeichnet. Häufig wird hierfür auch der Begriff **Message Authentication Code** (**MAC**) verwendet.

14.5.1 Anwendungsbereiche

Nun fragen Sie sich vielleicht, wozu man schlüsselabhängige Hashfunktionen benötigt, wo es doch digitale Signaturen gibt (oder umgekehrt). Die erste Frage hat folgende Antwort: Eine schlüsselabhängige Hashfunktion benötigt viel weniger Rechenzeit als eine digitale Signatur und kommt mit kürzeren Schlüsseln aus. Aus wirtschaftlicher Sicht ist eine schlüsselabhängige Hashfunktion also zu bevorzugen. Eine digitale Signatur hat dagegen einen anderen Vorteil gegenüber einer schlüsselabhängigen Hashfunktion: Sie sorgt für Verbindlichkeit. Eine digitale Signatur mit Bobs privatem Schlüssel kann nur Bob anfertigen. Wenn Bob ein Buch bei Online-Händler Otto bestellt, dann kann Otto vor Gericht gehen, falls Bob die signierte Bestellung abstreitet. Wenn Bob dagegen einen geheimen Schlüssel mit Otto vereinbart und zur Bestellung eine schlüsselabhängige Hashfunktion verwendet, dann kann Otto ihm nichts beweisen. Bob kann jederzeit behaupten, Otto hätte den schlüsselabhängigen Hashwert selbst generiert. Da Otto den Schlüssel kennt, könnte es ja tatsächlich so gewesen sein.

Schlüsselabhängige Hashfunktionen haben andere Sicherheitsziele als gewöhnliche kryptografische Hashfunktionen. Bei Letzteren versucht Mallory, Kollisionen zu finden, bei Ersteren ist er dagegen am Schlüssel interessiert. Bei einer schlüsselabhängigen Hashfunktion ähnelt die Kryptoanalyse daher dem Brechen eines symmetrischen Verschlüsselungsverfahrens. Eine Ciphertext-Only-Attacke hat dabei jedoch keinen Sinn, da das Urbild nicht geheim ist. Stattdessen kann es Mallory stets mit einer **Known-Preimage-** und manchmal mit einer **Chosen-Preimage-Attacke** versuchen (Preimage ist der englische Ausdruck für Urbild), was einer Known- bzw. Chosen-Plaintext-Attacke entspricht.

14.5.2 Die wichtigsten schlüsselabhängigen Hashfunktionen

Beim Design schlüsselabhängiger Hashfunktionen lohnt es sich nicht, das Rad neu zu erfinden. Stattdessen sind alle gängigen schlüsselabhängigen Hashfunktionen Weiterentwicklungen bereits bekannter kryptografischer Verfahren. Dabei gibt es zwei gängige Möglichkeiten: Entweder Alice und Bob verwenden eine kryptografische Hashfunktion als Grundlage oder sie nutzen ein symmetrisches Verschlüsselungsverfahren. Im erstgenannten Fall heißt die bekannteste Variante HMAC, im zweitgenannten Fall CBC-MAC. Eine dritte, bisher nur selten genutzte Möglichkeit ist eine Stromchiffre, die zusätzlich einen Hashwert generiert (siehe Abschnitt 16.1.1).

HMAC

Es gibt mehrere einfache Methoden, um aus einer kryptografischen Hashfunktion eine schlüsselabhängige Hashfunktion zu konstruieren (k ist im Folgenden der Schlüssel, U das Urbild, H eine kryptografische Hashfunktion und h der Hashwert). Von diesen einfachen Methoden gilt folgende als die beste: $h = H(k,U,k)$. Der Schlüssel wird also sowohl vor dem Urbild als auch dahinter platziert. Allerdings hat diese Methode theoretische Schwächen [PreOor], weshalb sie oft gemieden wird.

Als sicher gilt dagegen eine Methode namens **HMAC** (»H« steht hierbei für Hashfunktion; »MAC« ist, wie erwähnt, ein anderes Wort für eine schlüsselabhängige Hashfunktion). Die HMAC-Methode ist mehrfach standardisiert [RFC2104, RFC2202, FIPS-198]. Sie lässt sich mit jeder beliebigen kryptografischen Hashfunktion realisieren, vorteilhaft ist jedoch eine Funktion, die auf die gängige Art mit einem Status arbeitet. Ist b die Blocklänge der Hashfunktion, dann muss die Schlüssellänge ebenfalls b betragen (falls notwendig wird ein kürzerer Schlüssel mit Null-Bits aufgefüllt). Zur Definition von HMAC benötigen wir außerdem die beiden Variablen *opad* und *ipad*. Beide sind Bit-Folgen der Länge b. *opad* besteht aus lauter Bytes mit dem Wert 01011100, *ipad* aus lauter Bytes mit dem Wert 00110110. Eine HMAC-Funktion ist nun wie folgt definiert (»||« steht für die Konkatenation):

$$h = H((k \oplus opad) || H((k \oplus ipad) || U))$$

Die verwendete kryptografische Hashfunktion kommt also doppelt zum Einsatz. Dieser Nachteil lässt sich kompensieren, wenn derselbe Schlüssel für unterschiedliche Urbilder verwendet wird und ein mit Status arbeitendes Verfahren wie SHA-1 verwendet wird. In diesem Fall hat $k \oplus opad$ immer denselben Wert, wodurch die Initialisierung des Status für eine der beiden Abarbeitungen der Hashfunktion vorberechnet werden kann. Durch diese Besonderheit sind HMAC-Funktionen relativ performant. Obendrein gelten sie als sicher. Kein Wunder, dass sie in der Praxis häufig eingesetzt werden.

CBC-MAC

Es gibt zahlreiche Möglichkeiten, aus einem symmetrischen Verschlüsselungsverfahren eine schlüsselabhängige Hashfunktion zu generieren. Die wichtigste davon hat den Namen **CBC-MAC**. Diese Methode ist mit jeder beliebigen Blockchiffre (z. B. AES) anwendbar. Sie sieht vor, dass Alice das Urbild im CBC-Modus verschlüsselt (mit k als Schlüssel). Der letzte Block des Geheimtexts ist der Hashwert. Als Initialisierungsvektor wird meist eine Folge von Null-Bits verwendet, zudem ist ein Padding des Urbilds notwendig.

Die Idee des CBC-MAC ist schon recht alt. Bereits 1985 wurde diese Methode in Form des (inzwischen nicht mehr gültigen) US-Standards FIPS-113 standardisiert. Als symmetrisches Verschlüsselungsverfahren ist dabei der DES

vorgesehen, das Padding besteht aus einem Auffüllen mit Null-Bits. Es gibt weitere Standards, in denen nach dem CBC-MAC-Prinzip aufgebaute schlüsselabhängige Hashfunktionen festgelegt werden (z. B. *ISO/IEC 9797-1:2011*).

Beachten Sie, dass sich das CBC-MAC-Prinzip nicht auf schlüsselunabhängige kryptografische Hashfunktionen übertragen lässt. Bei bekanntem Schlüssel ist es einfach, Kollisionen zu generieren. Dieser scheinbare Makel spielt jedoch bei der typischen Anwendung einer kryptografischen Hashfunktion keine Rolle. Schon etwas kritischer sind einige andere Fallen, die sich beim Einsatz von CBC-MAC-Funktionen ergeben. Wenn Sie eine CBC-MAC-Funktion in der Praxis einsetzen wollen, sollten Sie sich daher zuvor genauer mit diesem Thema beschäftigen. [BeKiRo] ist ein guter Einstieg, [NESSIE] liefert ebenfalls interessante Informationen.

Weitere schlüsselabhängige Hashfunktionen

Die folgenden weiteren schlüsselabhängigen Hashfunktionen sind von Interesse:

- **UMAC**: UMAC ist eine weitere Methode zur Nutzung eines symmetrischen Verschlüsselungsverfahrens als schlüsselabhängige Hashfunktion. Eine Variante davon – auf Basis des AES – ist in [RFC4418] standardisiert.
- **EMAC**: Auch dies ist eine Methode zur Nutzung eines symmetrischen Verschlüsselungsverfahrens als schlüsselabhängige Hashfunktion [NESSIE].
- **TTMAC**: Dies ist eine schlüsselabhängige Hashfunktion auf Basis von RIPEMD-160 [NESSIE].
- **COMP128**: Diese schlüsselabhängige Hashfunktion ist eine der wenigen, die weder auf einem symmetrischen Verschlüsselungsverfahren noch auf einer kryptografischen Hashfunktion basiert. Die Funktionsweise sieht Permutationen und Substitutionen vor, die an symmetrische Verschlüsselungsverfahren erinnern [Sin].

Fazit

Schlüsselabhängige Hashfunktionen sind zwar nur ein kleines Teilgebiet der Kryptografie. Dennoch wurde zu diesem Thema weit mehr veröffentlicht, als ich an dieser Stelle behandeln kann. Leider ist mir keine Literaturquelle bekannt, in der der aktuelle Wissensstand zum Thema schlüsselabhängige Hashfunktionen auf verständliche Weise zusammengefasst wird. Es lohnt sich aber, das entsprechende Kapitel in [FeScKo] zu lesen.

Die wichtigste Frage lautet nun: Welche schlüsselabhängige Hashfunktion ist die beste? Natürlich ist diese Frage nicht eindeutig und für alle Fälle zu beantworten. Wenn Sie die Überlegungen dieses Kapitels bis hierher verfolgt haben, dürfte allerdings klar sein, dass Sie mit einer HMAC-Funktion (beispielsweise auf Basis von SHA-256 oder RIPEMD-160) am wenigsten falsch machen können.

14.6 Weitere Anwendungen kryptografischer Hashfunktionen

Bisher haben wir nur zwei Anwendungen kryptografischer Hashfunktionen betrachtet: Zum einen ging es um Hashwerte für digitale Signaturen, zum anderen um schlüsselabhängige Hashwerte. Es gibt allerdings noch einige weitere Anwendungsgebiete, die wir uns nun anschauen wollen.

14.6.1 Hashbäume

Wenn Alice eine Nachricht hasht und den Hashwert anschließend signiert, dann benötigt Bob stets die gesamte Nachricht, um die Echtheit der Signatur überprüfen zu können. Dies kann durchaus ein Nachteil sein. Wenn Alice etwa eine ganze Datenbank signiert, Bob aber nur an einem einzelnen Eintrag daraus interessiert ist, entsteht unnötige Verifikationsarbeit. Es gibt jedoch eine recht einfache Methode, um diesen Mangel zu beheben. Um diese zu erklären, nehmen wir an, dass Alice eine Liste L_0, L_1, ..., L_{n-1} signieren soll und Bob einzelne Einträge davon verifizieren will. Die Methode funktioniert nun wie folgt (siehe auch Abbildung 14–13):

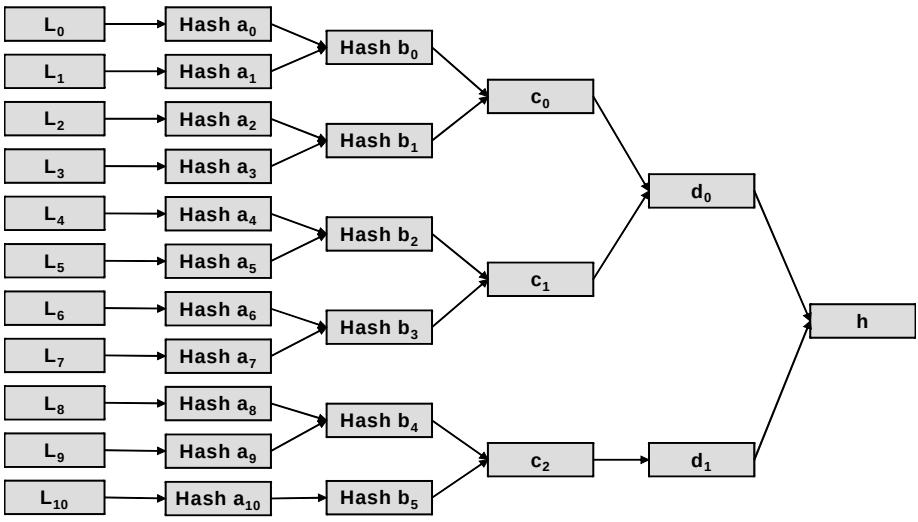


Abb. 14–13 *Durch den Einsatz von Hashbäumen benötigt man nicht den gesamten signierten Text, um eine digitale Signatur zu verifizieren.*

- Alice berechnet zu jedem Listeneintrag einen Hashwert. Die n Hashwerte nennen wir a_0, a_1, ..., a_{n-1}.
- Alice wendet auf a_0 und a_1 erneut eine Hashfunktion an und erhält dadurch den Hashwert b_0. Auf gleiche Weise erhält sie aus a_2 und a_3 den Hashwert b_1 usw. Bleibt am Ende a_{n-1} übrig (dies passiert, wenn n ungerade ist), dann gilt $b_{(n-2)/2}=a_{n-1}$.

- Alice wendet auf b_0 und b_1 erneut eine Hashfunktion an und erhält dadurch c_0. Auf analoge Weise erhält sie c_1, c_2 usw.
- Alice führt diese Vorgehensweise fort, bis nur noch ein Hashwert übrig bleibt. Diesen Hashwert h signiert sie.

Alice hat nun eine Datenstruktur, wie sie in der Abbildung zu sehen ist. Diese Datenstruktur wird **Hashbaum** genannt. Hashbäume wurden bereits 1979 von dem bekannten Kryptografen Ralph Merkle erfunden. Will Bob überprüfen, ob ein Listeneintrag (nehmen wir L_3) korrekt ist, dann benötigt er dazu nicht die gesamte Liste, sondern nur den Listeneintrag L_3 sowie die Hashwerte a_2, b_0, c_1 und d_1. Bob berechnet aus diesen Informationen a_3, b_1, c_0, d_0 und h. Mit h kann er die Signatur verifizieren.

Hashbäume wirken zwar auf den ersten Blick etwas umständlich. Ihr Nutzen ist jedoch bei langen Listen erheblich. Nehmen wir beispielsweise an, dass Alice eine Liste mit einer Million Einträgen mit einem Hashbaum signiert hat. Statt einer Million Einträge muss Bob in diesem Fall nur einen Eintrag sowie etwa 20 Hashwerte verarbeiten. Von Vorteil ist dabei auch, dass Alice nicht die gesamte Liste herausgeben muss. Eine Anwendung von Hashbäumen sind beispielsweise Sperrbäume in Public-Key-Infrastrukturen (siehe Abschnitt 28.3.3).

14.6.2 Weitere Anwendungen

Es gibt einige weitere Anwendungsfelder für kryptografische Hashfunktionen, auf die ich im Verlauf des Buchs noch näher eingehen werde. Hier ein kurzer Überblick:

- Kryptografische Hashfunktionen können als Pseudozufallsgenerator eingesetzt werden (mehr darüber in Abschnitt 15.1.3).
- Kryptografische Hashfunktionen werden zum Mischen verschiedener Zufallsquellen verwendet (mehr darüber in Abschnitt 15.1.5).
- Kryptografische Hashfunktionen werden verwendet, um aus einem Schlüssel (Masterschlüssel) einen weiteren (Sitzungsschlüssel) abzuleiten (siehe Abschnitt 31.3.2).
- Kryptografische Hashfunktionen lassen sich für das Generieren von Einmal-Passwörtern nutzen (siehe Abschnitt 21.3.1).
- Kryptografische Hashfunktionen liefern Verfahren, mit denen sich feststellen lässt, ob jemand derjenige ist, der er vorgibt zu sein (Authentifizierung, mehr dazu in Kapitel 21).

Wie Sie sehen, sind kryptografische Hashfunktionen ein wichtiges Instrument in der Hand eines Kryptografen. Sie werden Ihnen daher im Verlauf des Buchs noch einige Male begegnen.

15 Kryptografische Zufallsgeneratoren

Zufallszahlen spielen in der Kryptografie eine wichtige Rolle. Alice und Bob brauchen sie vor allem, um Schlüssel zu generieren. Dies kann ein AES-Schlüssel sein (128 bis 256 Bit) oder auch ein One-Time-Pad-Schlüssel, der so lang ist wie die zu verschlüsselnde Nachricht. Als RSA-Schlüssel kann Alice zwar keine beliebige Zufallszahl verwenden, da ein solcher Schlüssel aus zwei Primzahlen berechnet wird. Alle praktikablen Primzahlgeneratoren sind jedoch gewöhnliche Zufallsgeneratoren, die Nichtprimzahlen verwerfen. Public-Key-Verfahren, die wie Diffie-Hellman, ElGamal oder DSA auf dem diskreten Logarithmus basieren, sehen private Schlüssel vor, die in der Regel ebenfalls mit einem Zufallsgenerator gebildet werden. Zudem muss Alice für jede ElGamal- oder DSA-Signatur, die sie erstellt, eine weitere Zufallszahl generieren.

In allen genannten Fällen gilt: Mallory kann selbst das sicherste Verfahren knacken, wenn er die verwendete Zufallszahl kennt. Es leuchtet daher ein, dass ein Kryptografiebuch auf Zufallszahlen eingehen sollte. Weniger einleuchtend ist

dagegen, warum es gleich ein ganzes Kapitel zu diesem Thema sein muss – schließlich bietet so gut wie jede Programmiersprache einen Zufallsgenerator. Die Erklärung ist einfach: Viele Zufallsgeneratoren, die von Compilern oder Betriebssystemen zur Verfügung gestellt werden, sind für die Kryptografie nicht gut genug. In der Tat gibt es keinen Teilbereich der Kryptografie, der so oft und so gründlich unterschätzt wird wie das Generieren von Zufallszahlen. Deshalb werden wir die Sache etwas genauer betrachten.

Fangen wir mit einigen Definitionen an: Unter einer **Zufallsfolge** verstehen wir in diesem Zusammenhang eine längere, zufällige Folge von Nullen und Einsen (eine Definition des Begriffs »zufällig« erfolgt später). Wie diese Nullen und Einsen interpretiert werden – beispielsweise in Achtergruppen als ASCII-Code oder einfach als Binärzahlen – ist unerheblich. Eine **Zufallszahl** ist ein Ausschnitt aus einer Zufallsfolge mit einer begrenzten Länge (zum Beispiel 128 Bit, wenn die Zufallszahl als AES-Schlüssel verwendet werden soll). Ein **Zufallsgenerator** (auch Zufallszahlengenerator genannt) ist ein Verfahren, das eine Zufallsfolge als Ergebnis liefert.

Für Zufallszahlen gibt es auch außerhalb der Kryptografie viele Anwendungen. So bestimmen Computerspiele per Zufall, aus welcher Richtung ein Alien kommt, das abgeschossen werden muss; Simulationsprogramme bestimmen per Zufall, welche Länder im Meer versinken und wie viele Menschen an einer Krankheit sterben; probabilistische Algorithmen berechnen mithilfe von Zufallszahlen Näherungswerte, wenn eine genaue Berechnung zu komplex ist. Durch diese vielen Anwendungen gibt es längst Berge von Literatur zum Thema Zufallszahlen.

15.1 Zufallszahlen in der Kryptografie

Wer nun denkt, die gängigen Verfahren zur Generierung von Zufallszahlen ließen sich problemlos auch in der Kryptografie einsetzen, der irrt. Die Crux ist, dass nichtkryptografische Anwendungen weitaus geringere Anforderungen an Zufallszahlen stellen als die Kryptografie (eine ähnliche Situation gibt es bei kryptografischen Hashfunktionen und bei kryptografischen Protokollen). Probabilistische Algorithmen und die meisten Simulationen benötigen lediglich Zufallsfolgen, die bestimmte statistische Eigenschaften haben. In diesen Fällen genügt es also, »zufällig« als »statistisch gut verteilt« zu definieren. Bei Computerspielen kommt als Anforderung hinzu, dass es für den Spieler nicht auf einfache Weise möglich sein darf, die Richtung vorherzusagen, aus der das Alien kommt. Hier heißt zufällig also »statistisch gut verteilt und nicht einfach vorhersehbar«.

15.1.1 Anforderungen der Kryptografie

In der Kryptografie gelten dagegen deutlich schärfere Anforderungen als bei Computerspielen oder Simulationen. Es darf für Mallory auch mit größtem Aufwand und in vielen Versuchen nicht möglich sein, eine Zufallszahl vorherzusagen. Mallory muss einen RSA-, AES- oder One-Time-Pad-Schlüssel schließlich nicht beim ersten Versuch erraten, um Erfolg zu haben – es genügt, wenn er dies mithilfe seiner Supercomputer-Armada nach Milliarden von Versuchen schafft. »Zufällig« heißt in der Kryptografie also stets »mit realistischem Aufwand nicht vorhersagbar«.

Die meisten Zufallsgeneratoren, die in der nichtkryptografischen Literatur erwähnt werden oder in Compilern implementiert sind, liefern Ergebnisse, die für einen Abhörer vom Schlage Mallorys zu erraten sind – insbesondere dann, wenn wir wieder davon ausgehen, dass Mallory das verwendete Verfahren (sprich: den verwendeten Zufallsgenerator) kennt. Wir benötigen daher spezielle Zufallsgeneratoren, die den Anforderungen der Kryptografie Rechnung tragen. Möglicherweise denken Sie nun, es müsste trotz allem möglich sein, auch kryptografisch verwendbare Zufallsgeneratoren auf einfache Weise zu konstruieren. Dem ist aber nicht so – es gehört schon etwas Know-how dazu.

15.1.2 Echte Zufallsgeneratoren

Nehmen wir nun an, Alice wolle Zufallszahlen generieren, um einen Schlüssel für die Kommunikation mit Bob zu erhalten. Am sichersten ist es, wenn Alice einen physikalischen Vorgang verwendet, der nicht reproduzierbar ist. Ein Zufallsgenerator, der so etwas realisiert, wird **echter Zufallsgenerator** genannt. Gute Beispiele für echte Zufallsgeneratoren sind das Werfen einer Münze oder das Würfeln. Beide Vorgänge liefern ohne Zweifel unvorhersehbare Ergebnisse. Um einen echt zufälligen 128-Bit-Schlüssel zu erhalten, müsste Alice 128 Mal eine Münze werfen oder 50 Mal würfeln (warum gerade 50 Mal, müssen Sie jetzt selbst ausrechnen).

Aus naheliegenden Gründen werden Münzen und Würfel in der Praxis nur selten zur Schlüsselgenerierung eingesetzt. Besser ist es da schon, ein spezielles Hardwaremodul zu verwenden, das Zufallszahlen aus bestimmten Messwerten generiert. In der Praxis sind dies meist Spannungsschwankungen an Widerständen oder Dioden. Der Vorteil solcher Hardwaremodule ist, dass sie eine große Menge von Zufallszahlen pro Zeiteinheit liefern. Dafür ist eine derartige Luxuslösung recht umständlich, denn bisher zählt ein Hardwarezufallsgenerator leider noch nicht zur Standardausstattung eines Computers.

Anstatt ein spezielles Zufallsmodul zu verwenden, kann Alice auch auf bereits vorhandene Hardware zurückgreifen. Die Zeit zwischen zwei Laufwerkszugriffen, Tastatureingaben, Mausbewegungen und Ähnlichem birgt genügend Zufall, um ab und zu einen Schlüssel zu generieren. Einen großen Durchsatz wird

Alice damit sicherlich nicht erzielen, wenn die Sache nicht zu unsicher sein soll. Solange jedoch nur am Anfang einer Kommunikation ein 128-Bit-Schlüssel benötigt wird, dürfte diese Vorgehensweise ausreichen.

15.1.3 Pseudozufallsgeneratoren

Das Generieren von Zufallszahlen wird einfacher, wenn Alice dazu keine Hardware benötigt. Ein Zufallsgenerator, der ohne eine physikalische Messung auskommt und daher nur aus einem in Software implementierten Algorithmus besteht, wird als **Pseudozufallsgenerator** bezeichnet. Leider ist es nicht ganz einfach, allein durch Software unvorhersehbare Zufallszahlen zu generieren (wir setzen ja voraus, dass Mallory das Verfahren kennt). Dies liegt an der Tatsache, dass ein Computer dazu geschaffen ist, vorhersehbar zu agieren. Seine Aufgabe ist es, zu einer bestimmten Eingabe mithilfe eines bestimmten Algorithmus eine bestimmte Ausgabe zu liefern. Alles ist also bestimmt, und wenn Alice daher einen Zufallsgenerator allein mithilfe eines Softwareprogramms entwickelt, dann ist die resultierende Zufallsfolge für Mallory immer vorhersehbar und damit zunächst einmal nicht kryptografietauglich.

Abb. 15–1 *Bei einem Pseudozufallsgenerator wird ein Startwert sukzessiv fortgeschaltet.*

Da es kryptografietaugliche Pseudozufallsgeneratoren streng genommen nicht gibt, geht man in der Kryptografie immer davon aus, dass eine echte Zufallszahl als Eingabe vorliegt, die Mallory nicht bekannt ist. Ein Pseudozufallsgenerator ist damit in der Kryptografie stets ein Verfahren, das einen zufälligen Eingabewert (**Startwert** oder auch **Seed** genannt) zu einer beliebig langen Zufallsfolge verarbeitet. Dies geschieht, indem eine Funktion f (**Fortschaltfunktion**) auf den Startwert x beliebig oft angewandt wird. Mathematisch bedeutet dies, dass der Startwert x die erste Zufallszahl ist, $f(x)$ die zweite, $f(f(x))$ die dritte, $f(f(f(x)))$ die vierte und so weiter. Dabei ist jedoch Vorsicht geboten: Wenn Mallory eine Zufallszahl erfährt und die Funktion f kennt, dann kann er alle folgenden Zufallszahlen berechnen. Um dies zu vermeiden, gibt es zwei Möglichkeiten: Zum einen kann der Pseudozufallsgenerator immer nur einen Teil oder einen kryptografischen Hashwert des aktuellen Werts ausgeben; zum anderen kann Alice eine schlüsselabhängige Fortschaltfunktion verwenden.

Damit Alice eine Zufallsfolge, die mit einem Pseudozufallsgenerator erstellt worden ist, kryptografisch verwenden kann, müssen zwei Voraussetzungen erfüllt sein: Der Startwert darf von Mallory nicht zu erraten sein, und es muss eine Funktion f gewählt werden, die ohne Kenntnis des Startwerts keine Rück-

schlüsse auf die generierten Zufallszahlen ermöglicht. Dabei gehen wir von einem schlauen Mallory aus, der die Funktion f kennt (den Startwert x dagegen nicht). Dies bedeutet: Ein kryptografischer Pseudozufallsgenerator liefert zwar stets vorhersagbare Ergebnisse. Wenn der Startwert nicht bekannt ist, sind die Ergebnisse jedoch nicht vorhersagbar. Wie Sie sicher bemerkt haben, hat der Startwert die Funktion eines geheimen Schlüssels.

Wenn Sie nun denken, Alice könnte eine brauchbare Fortschaltfunktion programmieren, indem sie möglichst viele Rechenoperationen möglichst unübersichtlich kombiniert, dann liegen Sie falsch. Untersucht man die Ausgaben von Pseudozufallsgeneratoren, die auf diese chaotische Art programmiert sind, dann stellt man meist fest, dass einige Zufallszahlen recht häufig, die anderen dagegen selten oder gar nie vorkommen. Bei mehrfacher Anwendung wird man meist auch beobachten, dass sich die Zahlenfolgen schon bald wiederholen. Security by Intricacy ist also bei Pseudozufallsgeneratoren eindeutig fehl am Platze. Ein alte Kryptografenregel lautet daher: Pseudozufallsgeneratoren sollten niemals pseudozufällig generiert werden. Angesichts dieser Tatsache ist es kein Wunder, dass es mittlerweile stapelweise Literatur zum Thema kryptografische Pseudozufallsgeneratoren gibt. Die Erfolge sind nicht ausgeblieben: Gute Verfahren sind heute in ausreichender Zahl bekannt.

Pseudozufallsgeneratoren haben gegenüber echten Zufallsgeneratoren einen entscheidenden Vorteil: Sie sind sehr gut zu untersuchen, da sie von äußeren Einflüssen unabhängig sind und bei gleichem Startwert auch die gleiche Zufallsfolge erzeugen. Nicht zuletzt deshalb konzentriert sich ein großer Teil der Literatur auf Pseudozufallsgeneratoren. Dennoch bleiben zwei Makel: Zum einen liefern Pseudozufallsgeneratoren prinzipiell nur nachvollziehbare Zufallsfolgen. Zum anderen ist ein Pseudozufallsgenerator ohne echt zufälligen Startwert nur wenig wert.

15.1.4 Die Grauzone zwischen echt und pseudo

In die Grauzone zwischen echtem Zufall und Pseudozufall kommt Alice, wenn sie etwa den Inhalt bestimmter Speicherbereiche im Betriebssystem zur Zufallsgenerierung heranzieht. Ideal ist es, wenn sie hierbei eine Hashsumme aus Speicherbereichen bildet, deren Inhalt sich häufig ändert und schwer vorhersehbar ist. Wird ein Speicherbereich unmittelbar von der Hardware beeinflusst, dann hat diese Methode den Charakter eines echten Zufallsgenerators. Verwendet Alice dagegen solche Bereiche, die von der Software verändert werden, dann handelt es sich eher um einen Pseudozufallsgenerator. Bei richtiger Ausführung ist die Verwendung von Speicherbereichen äußerst sicher. Leider ist diese Methode schlecht mit mathematischen Mitteln zu untersuchen, und das mögen Kryptografen nicht.

15.1.5 Mischen von Zufallsquellen

Wenn Alice eine möglichst hohe Sicherheit erreichen will, dann bietet es sich für sie an, verschiedene Zufallsfolgen zu mischen (siehe dazu auch [RFC1750]). Dazu muss sie eine Funktion (**Mischfunktion**) verwenden, die eine beliebig große Anzahl an Eingabe-Bits auf eine kurze Bit-Folge abbildet, die sie dann als Zufallszahl verwendet. Die Ausgabe der Mischfunktion muss statistisch gut verteilt sein und darf insbesondere nur dann zu erraten sein, wenn alle Eingabe-Bits bekannt sind (wir setzen voraus, dass Mallory die Mischfunktion bekannt ist). Diese Anforderungen an eine Mischfunktion werden in fast perfekter Weise von einer (guten) kryptografischen Hashfunktion erfüllt.

Die Generierung sicherer Zufallszahlen mit einer Mischfunktion sieht vor, dass Alice einen Speicherbereich (**Zufallspool**) mit Zufallszahlen unterschiedlicher Herkunft füllt. Diese Zufallszahlen können beispielsweise zu einem Teil von einem Pseudozufallsgenerator stammen und zu einem anderen Teil von irgendwelchen Hardwaremessgrößen. Als weitere Quelle kann Alice Daten aus einem Speicherbereich verwenden, auf den das Betriebssystem häufig zugreift und der sich daher oft und unvorhersehbar ändert. Schließlich kann sie noch die Uhrzeit, eine Tastatureingabe und einen geheimen Schlüssel verwenden, um den Zufallspool aufzufüllen. Zuletzt bildet Alice den Zufallspool mit einer kryptografischen Hashfunktion als Mischfunktion auf eine kürzere Bit-Kette ab. Wenn Mallory diese Zufallszahl erraten will, dann muss er den gesamten Inhalt des Zufallspools erraten.

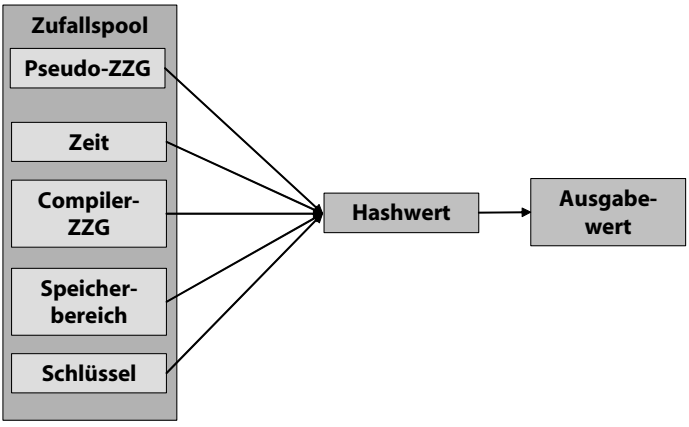


Abb. 15–2 *Das Mischen von Zufallszahlen unterschiedlicher Herkunft mit einer Hashfunktion gilt als gute Methode zur Generierung von Zufallsfolgen.*

15.2 Die wichtigsten Pseudozufallsgeneratoren

Schauen wir uns nun an, welche Pseudozufallsgeneratoren Alice und Bob in der Praxis verwenden können. Alle der im Folgenden beschriebenen Verfahren sind speziell für kryptografische Zwecke entwickelt worden und gelten damit (falls nicht anders gesagt) als sicher. Eine Besonderheit von Pseudozufallsgeneratoren gegenüber anderen kryptografischen Verfahren besteht darin, dass sie nicht interoperabel sein müssen (es spielt keine Rolle, wenn Alice ihre Zufallszahlen anders generiert als Bob). Aus diesem Grund sind nur wenige Pseudozufallsgeneratoren standardisiert, und es sind sehr viele unterschiedliche Verfahren in Gebrauch.

Einige gute Pseudozufallsgeneratoren findet man in den Veröffentlichungen von Standardisierungsgremien. Da es nicht auf Interoperabilität ankommt, haben diese Beschreibungen meist nicht den Status eines Standards und sind in der Regel auch nicht bis ins letzte Bit festgelegt. Der Implementierer hat dadurch verschiedene Freiheiten. So basieren viele der Pseudozufallsgeneratoren auf einem Verschlüsselungsverfahren oder einer Hashfunktion, ohne dass festgelegt ist, welcher Algorithmus verwendet wird. In viele Verfahren gehen neben dem Startwert zusätzliche Bit-Folgen ein, die aus einer anderen Zufallsquelle kommen können, wobei die Details ebenfalls offen gelassen werden.

Eine interessante Veröffentlichung von einem Standardisierungsgremium ist beispielsweise die *Recommendation for Random Number Generation Using Deterministic Random Bit Generators (NIST SP 800-90)* der US-Standardisierungsbehörde NIST [NIST800-90]. In dieser Empfehlung, die kostenlos im Internet erhältlich ist, findet sich eine recht vielfältige Auswahl geeigneter Pseudozufallsgeneratoren – leider ist die Beschreibung meist etwas umständlich. Der Data Signature Standard (DSS, siehe Abschnitt 12.3.2), der das digitale Signaturverfahren DSA spezifiziert, führt im Anhang ebenfalls exemplarisch einige Pseudozufallsgeneratoren auf [FIPS-186]. Gleiches gilt für den Anhang des Standards ANSI X9.42 [X9.42], in dessen Hauptteil die Verfahren Diffie-Hellman und MQV beschrieben werden. Zu erwähnen ist außerdem der japanische Krypto-Wettbewerb CRYPTREC (siehe Abschnitt 18.5.4), in dem es eine Kategorie Pseudozufallsgeneratoren gab [CRYPTR].

Auf den folgenden Seiten werden wir einige Pseudozufallsgeneratoren betrachten, die größtenteils aus den genannten Quellen stammen. Die beschriebenen Methoden lassen sich danach klassifizieren, ob eine kryptografische Hashfunktion, eine schlüsselabhängige Hashfunktion, ein symmetrisches Verschlüsselungsverfahren oder eine mathematische Problemstellung zur Fortschaltung verwendet wird. Leider haben die jeweiligen Verfahren in vielen Fällen keinen Namen. Stattdessen wird in der Literatur meist das Dokument, in dem das Verfahren beschrieben wird, mit der zugehörigen Kapitelnummer als Bezeichnung genannt (z.B. *ANSI X9.42-2001 Annex C.1*). Dies ist zwar umständlich und teilweise etwas verwirrend, da manche Methoden in mehreren Dokumenten beschrieben werden, doch es gibt bisher keine bessere Namensgebung.

15.2.1 Kryptografische Hashfunktionen als Fortschaltfunktion

Ein naheliegender Ansatz für die Konstruktion eines Pseudozufallsgenerators besteht darin, als Fortschaltfunktion eine kryptografische Hashfunktion zu verwenden. Alice kann etwa einen Startwert der Länge 224 Bit mit der kryptografischen Hashfunktion Keccak fortschalten. Wenn Keccak einen 224-Bit-Hashwert ausgibt, erhält Alice auf diese Weise eine Folge von zufälligen 224-Bit-Werten. Von diesen sollte sie jedoch nur einen Teil (beispielsweise 80 Bit pro Fortschaltung) verwenden, damit Mallory die Zufallsfolge nicht selbst fortschreiben kann. Noch besser ist es, wenn Alice aus dem jeweiligen 224-Bit-Wert einen Hashwert generiert (auf andere Weise als bei der Fortschaltung) und diesen als Zufallswert nutzt. Dies sieht dann etwa so aus (die Variable A hat 224 Bit und wird mit dem Startwert initialisiert):

1. A = Keccak(A)
2. Gib als Zufallszahl aus: Keccak(01010101||A)

Das Byte 01010101 stellt sicher, dass die ausgegebene Zufallszahl nicht mit dem Ergebnis der nächsten Fortschaltung identisch ist. Es gibt zahlreiche Abwandlungen dieses Vorgehens. Vier davon stelle ich in den nächsten Abschnitten vor.

NIST SP 800-90 Kapitel 10.1.1

Der folgende Pseudozufallsgenerator wird in Kapitel 10 der bereits erwähnten NIST-Empfehlung beschrieben [NIST800-90] und wird folgerichtig als **NIST SP 800-90 Kapitel 10.1.1** bezeichnet. Das Verfahren sieht vor, eine kryptografische Hashfunktion als Fortschaltfunktion zu verwenden. Dieselbe Hashfunktion kommt hierbei mehrfach zum Einsatz. Welche Hashfunktion verwendet wird, ist nicht festgelegt. Die Hashwert-Länge ist im Folgenden b. Zusätzlich spielt ein Zähler (*Reseed-Counter*) eine Rolle, an dem erkennbar ist, wie viele Fortschaltungen Alice bereits mit demselben Startwert durchgeführt hat (ist eine Obergrenze erreicht, dann kann Alice einen neuen Startwert verwenden). Der Reseed-Counter wird mit 1 initialisiert.

Der Startwert des Verfahrens wird nicht direkt verwendet. Stattdessen bearbeitet Alice ihn zur Initialisierung zusammen mit einer Konstanten und zusätzlichen Eingabewerten, deren Größe und Herkunft nicht festgelegt sind, mit einer kryptografischen Hashfunktion. Das Ergebnis wird zum neuen Startwert und geht als solcher in Form der Variablen A in die Fortschaltfunktion ein. Diese sieht wie folgt aus (X steht für einen beliebigen zusätzlichen Input, der beispielsweise von einem anderen Zufallsgenerator kommen kann):

1. $A = A$ + Hashfunktion(00000010||A||X) (mod 2^b)
2. Gib als Zufallswert aus: Hashfunktion(A)
3. $A = A$ + Hashfunktion(00000011||A) + C + *Reseed-Counter* (mod 2^b)
4. *Reseed-Counter* = *Reseed-Counter* + 1

Alice kann Schritt 1 auch weglassen, wenn sie keine sinnvolle Quelle für X hat. Der beschriebene Ablauf der Fortschaltung ist gegenüber der NIST-Empfehlung etwas vereinfacht, da er pro Aufruf jeweils b Zufalls-Bits ausgibt. In der NIST-Empfehlung sind einige Schritte etwas allgemeiner gehalten, damit Alice auch mehr oder weniger Zufalls-Bits pro Fortschaltung generieren kann.

ANSI X9.42-2001 Annex C.1

Der Pseudozufallsgenerator **ANSI X9.42-2001 Annex C.1** (er wird in [X9.42] beschrieben) verwendet die kryptografische Hashfunktion SHA-1. Allerdings mit einer leichten Änderung: Auf das von SHA-1 vorgesehene Padding wird verzichtet, stattdessen füllt Alice das Urbild (es ist maximal 512 Bit lang) mit Null-Bits zu einem 512-Bit-Block auf. Alice wählt einen Startwert, der wiederum in einer Variablen A gespeichert wird. Deren Länge b beträgt mindestens 160 und höchstens 512 Bit. Pro Fortschaltung werden 160 Zufalls-Bits generiert. Die Fortschaltung sieht wie folgt aus (X steht wiederum für einen beliebigen zusätzlichen Input, der beispielsweise von einem anderen Zufallsgenerator kommen kann):

1. $A = A+X \pmod{2^b}$
2. Gib als Zufallswert aus: SHA-1(A)
3. $A = A + \text{SHA-1}(A) + 1 \pmod{2^b}$

Es spricht nichts dagegen, dieses Verfahren auch mit einer anderen kryptografischen Hashfunktion anzuwenden. Gegenüber *NIST SP 800-90 Kapitel 10* hat es den Vorteil, dass die Generierung des Hashwerts durch das einfachere Padding etwas performanter ist. *ANSI X9.42-2001 Annex C.1* ist eines von drei Verfahren, die in der Kategorie Pseudozufallsgeneratoren des CRYPTREC-Wettbewerbs positiv bewertet wurden [CRYPTR]. Als Wettbewerbssieger kann man das Verfahren dennoch nicht bezeichnen, da die CRYPTREC-Regularien von Anfang an vorsahen, in der Kategorie Pseudozufallsgeneratoren nur ein paar Beispiele zu betrachten. Angesichts der Vielzahl von Verfahren in unzähligen Varianten und der fehlenden Notwendigkeit für eine Standardisierung verzichtete man auf einen umfassenden Wettbewerb in dieser Sparte.

FIPS 186-2 (+ change notice 1) Appendix 3.1

FIPS 186-2 ist der Data Signature Standard (DSS), also der Standard, in dem der DSA beschrieben wird [FIPS-186]. Im Anhang 3.1 dieses Dokuments wird ein Pseudozufallsgenerator beschrieben, den Alice und Bob verwenden können, um einen privaten DSA-Schlüssel zu berechnen. Dabei handelt es sich um eine Zahl, die kleiner als die vom Standard vorgegebene 160-Bit-Primzahl q sein muss. Mit der Change Notice 1 wurde das im Standard ursprünglich aufgeführte Verfahren durch ein anderes ersetzt. Die Bezeichnung für dieses Verfahren ist daher **FIPS 186-2 (+ change notice 1) Appendix 3.1**. Es entspricht dem oben beschriebenen

ANSI X9.42-2001 Annex C.1 mit nur einem minimalen Unterschied: Ist eine 160-Bit-Zufallszahl größer als q, dann wird q abgezogen. Dieser Schritt ist notwendig, damit die Zufallszahl als Schlüssel für den DSA nutzbar ist. Wenn Alice und Bob das Verfahren jedoch für andere Zwecke nutzen wollen, können sie diesen Schritt weglassen. *FIPS 186-2 (+ change notice 1) Appendix 3.1* wurde im Rahmen des CRYPTREC-Wettbewerbs als positiv bewertet.

FIPS 186-2 (+ change notice 1) revised Appendix 3.1

Eine Eigenschaft von *FIPS 186-2 (+ change notice 1) Appendix 3.1* (und damit auch von *ANSI X9.42-2001 Annex C.1*) ist folgende: Da A bis zu 512 Bit lang sein kann, der Hashwert jedoch immer nur 160 Bits hat, wirkt sich die Fortschaltung (diese besteht im Wesentlichen aus einer Addition der beiden Werte) nur auf einen Teil von A aus. Diese Eigenschaft lässt nach Angaben des NIST einen (allerdings nur theoretisch interessanten) Angriff zu. Das NIST reichte daher ein leicht geändertes Verfahren nach, das als **FIPS 186-2 (+ change notice 1) revised Appendix 3.1** bezeichnet wird [FIPS-186]. Dieses läuft wie folgt ab (SHA-1 wird wiederum mit vereinfachtem Padding verwendet):

1. $A = A+X \pmod{2^b}$
2. Gib als erste Hälfte des Zufallswerts aus: SHA-1(A)
3. $A = A + \text{SHA-1}(A) + 1 \pmod{2^b}$
4. $A = A+X \pmod{2^b}$
5. Gib als zweite Hälfte des Zufallswerts aus: SHA-1(A)
6. $A = A + \text{SHA-1}(A) + 1 \pmod{2^b}$

Auch hier sieht die Empfehlung vor, den Zufallswert modulo q zu verwenden, damit er für den DSA nutzbar ist. Wenn Alice und Bob jedoch einen anderen Verwendungszweck im Auge haben, können sie diesen Schritt weglassen. Wie man leicht erkennt, ist der Unterschied zwischen *FIPS 186-2 (+ change notice 1) Appendix 3.1* und *ANSI X9.42-2001 Annex C.1* auf der einen Seite sowie *FIPS 186-2 (+ change notice 1) revised Appendix 3.1* auf der anderen Seite minimal. Ich habe das letztgenannte Verfahren an dieser Stelle nur aufgeführt, weil es im CRYPTREC-Wettbewerb als dritter und letzter der hier beschriebenen Pseudozufallsgeneratoren positiv bewertet wurde.

15.2.2 Schlüsselabhängige Hashfunktionen als Fortschaltfunktion

Anstatt einer gewöhnlichen kryptografischen Hashfunktion können Alice und Bob zur Zufallsgenerierung auch eine schlüsselabhängige Hashfunktion verwenden. Dies funktioniert etwa mit der folgenden einfachen Methode (K ist der Schlüssel):

1. $A = \text{hash}_K(A)$
2. Gib als Zufallswert aus: A

Alice und Bob können auch ein dem CTR-Modus (siehe Abschnitt 19.1.5) nachempfundenes Verfahren nutzen:

1. $A = A + 1$
2. Gib als Zufallswert aus: $\text{hash}_K(A)$

In [NIST800-90] wird ein etwas erweitertes Verfahren beschrieben, das ich als **NIST SP 800-90 Kapitel 10.1.2** bezeichnen will. Dieses hat die Eigenschaft, dass der Schlüssel im Rahmen der Fortschaltung einen neuen Wert annimmt. Das Verfahren gibt es in zwei Varianten – je nachdem, ob eine zusätzliche Zufallsquelle X zur Verfügung steht oder nicht. Beide Varianten setzen voraus, dass die Schlüssellänge und die Hashwert-Länge der verwendeten schlüsselabhängigen Hashfunktion gleich sind. Der Startwert wird in beiden Fällen vorbearbeitet, was ich an dieser Stelle übergehen will. Wenn keine zusätzliche Zufallsquelle X einfließen soll, dann läuft die Fortschaltung wie folgt ab:

1. $A = \text{hash}_K(A)$
2. Gib als Zufallswert aus: A
3. $K = \text{hash}_K(A||00000000)$
4. $A = \text{hash}_K(A)$

Pro Fortschaltung werden so viele Bits ausgegeben, wie es der Hashwert-Länge entspricht. Schritt 1 und 2 können mehrfach ausgeführt werden, wenn pro Fortschaltung eine größere Zahl von Bits ausgegeben werden soll. Wenn ein Wert für X einfließt, dann läuft das Verfahren so ab:

1. $K = \text{hash}_K(A||00000000||X)$
2. $A = \text{hash}_K(A)$
3. $K = \text{hash}_K(A||00000001||X)$
4. $A = \text{hash}_K(A)$
5. $A = \text{hash}_K(A)$
6. Gib als Zufallswert aus: A
7. $K = \text{hash}_K(A||00000000||X)$
8. $A = \text{hash}_K(A)$
9. $K = \text{hash}_K(A||00000001||X)$
10. $A = \text{hash}_K(A)$

Pro Fortschaltung werden auch hier so viele Bits ausgegeben, wie es der Hashwert-Länge entspricht. Schritt 5 und 6 können mehrfach ausgeführt werden, wenn pro Fortschaltung eine größere Zahl von Bits ausgegeben werden soll.

15.2.3 Blockchiffren als Fortschaltfunktion

Es ist recht offensichtlich, dass Alice und Bob neben einer (gegebenenfalls schlüsselabhängigen) kryptografischen Hashfunktion auch ein symmetrisches Verschlüsselungsverfahren (also eine Blockchiffre) zur Konstruktion eines Pseudozu-

fallsgenerators verwenden können. Dies funktioniert beispielsweise wie folgt (*K* ist der Schlüssel):

1. $A = E_K(A)$
2. Gib als Zufallswert aus: A

Alice und Bob können die Blockchiffre auch im CTR-Modus (siehe Abschnitt 19.1.5) nutzen:

1. $A = A + 1$
2. Gib als Zufallswert aus: $E_K(A)$

Ein weiteres Verfahren steht in [NIST800-90] und wird als **NIST SP 800-90 Kapitel 10.2.1** bezeichnet. Es sieht vor, dass sich im Rahmen der Fortschaltung nicht nur *A*, sondern auch der Schlüssel *K* ändert. Der Startwert wird vorbearbeitet, was ich an dieser Stelle übergehen will. Der Ablauf ist wie folgt festgelegt (*b* ist die Blocklänge des Verfahrens):

1. $A = A + 1 \pmod{2^b}$
2. $A = E_K(A)$
3. $A = A \oplus X$
4. $K = A$ (falls *A* mehr Bits als *K* hat, werden die überschüssigen Bits ignoriert)
5. $A = A + 1 \pmod{2^b}$
6. Gib als Zufallswert aus: $E_K(A)$
7. $A = A + 1 \pmod{2^b}$
8. $A = E_K(A)$
9. $A = A \oplus X$
10. $K = A$ (falls *A* mehr Bits als *K* hat, werden die überschüssigen Bits ignoriert)

Das Verfahren ist gegenüber der angegebenen Literaturquelle etwas vereinfacht dargestellt. Der hier gezeigte Ablauf setzt voraus, dass die Länge von *A* der Blocklänge des Verschlüsselungsverfahrens (und damit *b*) entspricht, wobei die Blocklänge nicht kürzer als die Schlüssellänge sein darf. Pro Fortschaltung werden *b* Bits generiert. Das Original-Verfahren aus dem Standard ist bezüglich dieser Größen etwas variabler, dafür jedoch deutlich unübersichtlicher.

15.2.4 Linear rückgekoppelte Schieberegister

Eine Jahrzehnte alte Methode zum Bau von Pseudozufallsgeneratoren ist die Verwendung von linear rückgekoppelten Schieberegistern. Diese werden auch als **LFSR** (Linear Feedback Shift Register) bezeichnet. Ein LFSR besteht aus einer Folge von *n* Variablen, die ich als a_0, a_1, ..., a_{n-1} bezeichnen werde. Jede Variable kann nur die Werte 0 und 1 annehmen. Wenn Alice ein Zufalls-Bit benötigt, dann liest sie den Wert von a_0 aus. Anschließend wird der Wert von a_1 in a_0 geschoben, a_1 erhält den Wert von a_2, a_2 den Wert von a_3 und so weiter. Der neue Inhalt der Variablen a_{n-1} wird mit einer Fortschaltfunktion aus der bisherigen Belegung der

n Variablen berechnet. Diese Fortschaltfunktion (**Feedback-Funktion**) ist bei einem LFSR die Exklusiv-oder-Verknüpfung einiger der n Variablen. Ist beispielsweise n=5, dann können etwa a_2 und a_3 als die Variablen festgelegt werden, die in die Feedback-Funktion eingehen. Diese Variablen werden als **Tap-Variablen** bezeichnet. Mit anderen Worten lässt sich die Feedback-Funktion folgendermaßen beschreiben: Gibt es vor der Verschiebung in den Tap-Variablen eine ungerade Anzahl von Einsen, dann wird eine Eins in a_{n-1} geschoben. Wenn nicht, dann ist es eine Null, die in a_{n-1} geschoben wird.

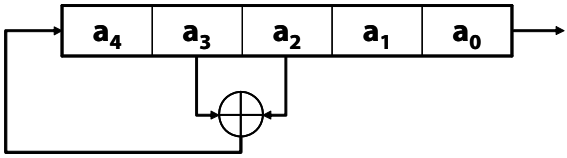


Abb. 15–3 *Beispiel für ein linear rückgekoppeltes Schieberegister (LFSR). a_2 und a_3 sind die Tap-Variablen.*

Die Werte, die von einem LFSR ausgegeben werden, bilden eine Zufallsfolge, die Alice für die Schlüsselgenerierung nutzen kann. Die Anfangsbelegung der Variablen ist der Startwert, die Feedback-Funktion übernimmt die Rolle der Fortschaltfunktion. Welche Variablen in die Fortschaltfunktion eingehen und wie viele Variablen es gibt, sollte Alice geheim halten und als einen Teil des Schlüssels betrachten (den Startwert natürlich auch).

Ein Beispiel

Betrachten wir nun ein Beispiel: Ein LFSR bestehe aus fünf Variablen a_0, a_1, a_2, a_3 und a_4. Die Tap-Variablen seien a_2 und a_3. Wählt Alice den Startwert 01100, dann ergeben sich folgende Belegungen.

Zufalls-Bit	a_0	a_1	a_2	a_3	a_4
1	0	1	1	0	0
2	1	1	0	0	1
3	1	0	0	1	0
4	0	0	1	0	1
5	0	1	0	1	1
6	1	0	1	1	1
7	0	1	1	1	0
8	1	1	1	0	0
9	1	1	0	0	1
10	1	0	0	1	0
11	0	0	1	0	1
12	0	1	0	1	1

Die resultierende Zufallsfolge ergibt sich aus den Belegungen von a_0. Sie lautet 011001011100. Die ersten fünf Bit der Zufallsfolge sind mit dem Startwert identisch.

Anwendungen von LFSRs

Mit einem LFSR lässt sich auf einfache Weise eine Zufallsfolge generieren. Wie schwer es für Mallory ist, einen Wert dieser Zufallsfolge vorherzusagen, hängt davon ab, wie viele Variablen verwendet werden (je mehr, desto besser) und welche davon als Tap-Variablen dienen. Insgesamt ist die Sicherheit eines LFSR-Pseudozufallsgenerators eher bescheiden. Dass LFSRs dennoch in keinem guten Kryptografie-Buch (auch nicht in diesem) fehlen, hat zwei Gründe: Zum einen sind LFSRs auf einfache Weise in Hardware realisierbar und erlauben dort obendrein eine sehr hohe Generierungsgeschwindigkeit; zum anderen lassen sich LFSRs mit anderen Techniken zu sichereren Krypto-Verfahren kombinieren. In den folgenden Kapiteln werden Sie noch einige Verschlüsselungsverfahren kennenlernen, die auf LFSRs basieren.

15.2.5 Nichtlinear rückgekoppelte Schieberegister

Tauscht man bei einem LFSR die Exklusiv-oder-Verknüpfung durch eine andere Funktion aus, dann entsteht ein nichtlinear rückgekoppeltes Schieberegister (**NFSR**). Da die Feedback-Funktion eines NFSR zahlreiche unterschiedliche Formen annehmen kann, sind NFSRs deutlich vielfältiger als LFSRs. Viele Varianten sind bisher kaum untersucht, andere sind kryptografisch unbrauchbar. In vielen Fällen sind keine theoretischen Grundlagen bekannt, die eine Untersuchung ermöglichen würden. Da Kryptografen eine solche Ungewissheit nicht mögen, sind NFSRs nicht so weit verbreitet wie LFSRs. Trotzdem werden Sie in diesem Buch einige Verschlüsselungsverfahren kennenlernen, die NFSRs verwenden. Manche Verfahren verwenden sogar beide Formen rückgekoppelter Schieberegister.

15.2.6 Zahlentheoretische Pseudozufallsgeneratoren

Wie ich in verschiedenen Kapiteln dieses Buchs beschrieben habe, basieren asymmetrische Krypto-Verfahren auf speziellen mathematischen Problemen. Bei RSA ist dies das Faktorisierungsproblem, bei Diffie-Hellman und dem DSA ist es der diskrete Logarithmus. Von Bedeutung sind außerdem die elliptischen Kurven, mit denen sich ein spezieller diskreter Logarithmus definieren lässt. Diese mathematischen Problemstellungen lassen sich auch für die Konstruktion von Pseudozufallsgeneratoren verwenden. Man spricht hierbei von einem **zahlentheoretischen Zufallsgenerator**.

Zahlentheoretische Zufallsgeneratoren sind vergleichsweise langsam, gelten jedoch durch ihre mathematische Fundierung als besonders sicher. Allerdings

zeigt die Erfahrung der vergangenen zwei Jahrzehnte, dass Alice und Bob auch ohne diese (in der Theorie) höhere Sicherheit auskommen und die anderen in diesem Kapitel beschriebenen Pseudozufallsgeneratoren völlig ausreichen. Ich will daher an dieser Stelle auf eine ausführliche Beschreibung verzichten und lediglich zwei Beispiele nennen:

- **Blum Blum Shub** ist ein Pseudozufallsgenerator, dessen Sicherheit wie beim RSA-Verfahren auf dem Faktorisierungsproblem beruht. Eine Beschreibung findet sich beispielsweise in den Standardwerken [Schn96] und [MeOeVa].
- **Dual Elliptic Curve Deterministic RBG:** Dieses Verfahren nutzt elliptische Kurven zur Zufallsgenerierung. Details dazu finden sich in [NIST800-90].

15.3 Primzahlgeneratoren

Wenn Alice ein RSA-Schlüsselpaar generiert, dann benötigt sie dafür zwei Primzahlen (bei Krypto-Verfahren, die auf dem diskreten Logarithmus beruhen, werden ebenfalls Primzahlen benötigt, die jedoch nicht geheim bleiben müssen und von einer Gruppe von Anwendern gleichzeitig verwendet werden können). Alice muss also in der Lage sein, zwei große Primzahlen zu generieren, die so zufällig sind, dass Mallory sie nicht erraten kann. Wie Sie vielleicht wissen, gibt es unendlich viele Primzahlen. Allerdings benötigt Alice eine Primzahl bestimmter Länge (etwa 512 Bit), und deren Anzahl ist naturgemäß begrenzt. Dennoch ist dies kein Grund zur Sorge: Es gibt etwa 10^{50} Primzahlen der Länge 512 Bit, und die Anzahl der Primzahlen steigt in etwa proportional zur Bit-Länge. Bei 1.024 Bit sind es damit schon etwa 10^{100} Primzahlen. Mallory hat somit nicht die geringste Chance, eine zufällig gewählte Primzahl zu erraten.

Da an Primzahlen kein Mangel herrscht, stellt sich nun die Frage, wie Alice eine zufällige Primzahl bestimmter Länge generieren kann. Zu diesem Thema gibt es genügend Veröffentlichungen, um damit ein ganzes Buch zu füllen. Wir müssen uns dagegen mit dem Existenzminimum begnügen. Zu diesem gehört die Tatsache, dass alle gängigen Primzahlgeneratoren nach dem gleichen Prinzip arbeiten: Erst wird eine Zufallszahl generiert, dann wird mit einem probabilistischen Algorithmus geprüft, ob es sich um eine Primzahl handelt. Falls nicht, wird eine neue Zufallszahl generiert und getestet. Dieses Prozedere wird wiederholt, bis schließlich eine Primzahl gefunden wird. »Probabilistisch« bedeutet dabei, dass der Algorithmus in Ausnahmefällen ein falsches Ergebnis liefert und eine Zahl für eine Primzahl hält, die in Wirklichkeit keine ist. Die Wahrscheinlichkeit für einen solchen Ausnahmefall kann jedoch durch die Wahl geeigneter Parameter beliebig klein gemacht werden (der Aufwand steigt mit sinkender Wahrscheinlichkeit). Ein guter Algorithmus, mit dem festgestellt werden kann, ob eine Zahl eine Primzahl ist, muss so verwendbar sein, dass sich bei einer geringen Wahrscheinlichkeit für eine Ausnahme dennoch der Aufwand im Rahmen hält. Zwei-

fellos wäre es besser, einen nichtprobabilistischen Algorithmus für den Primzahltest zu verwenden. Alle hierzu bekannten Methoden sind jedoch so aufwendig, dass ein probabilistisches Verfahren allemal die deutlich bessere Wahl ist.

Das mit Abstand bedeutendste Verfahren zur Primzahlgenerierung ist das **Miller-Rabin-Verfahren**. Dieses ist leicht zu implementieren und funktioniert nach dem oben beschriebenen Prinzip. Miller-Rabin sieht vor, dass zuerst eine Zufallszahl generiert wird, um anschließend mit einem sehr effektiven probabilistischen Test zu überprüfen, ob es sich um eine Primzahl handelt. Falls nicht, wird die nächste Zufallszahl auf diese Weise getestet. Dabei kommt für jeden Test eine zusätzliche Zufallszahl zum Einsatz, die kleiner als die getestete sein muss. Da der Primzahltest probabilistisch ist, besteht die (geringe) Gefahr, dass eine Zufallszahl den Test besteht, obwohl sie nicht prim ist. Viele Implementierungen führen deshalb mehrere Testrunden mit unterschiedlichen zusätzlichen Zufallszahlen durch. Die Wahrscheinlichkeit, dass eine Nichtprimzahl alle Runden übersteht, wird dadurch verschwindend gering. Wie das Miller-Rabin-Verfahren genau funktioniert, können Sie unter anderem in den Standardwerken [Schn96] und [MeOoVa] nachlesen.

In der Praxis wird Miller-Rabin meist mit einigen einfachen Vorabmaßnahmen kombiniert. So werden das erste und das letzte Bit einer zu testenden Zahl auf Eins gesetzt, damit die Zahl groß genug und ungerade ist. Außerdem wird vorab auf direktem Weg getestet, ob die betrachtete Zahl durch kleinere Primzahlen (beispielsweise alle Primzahlen kleiner als 2.000) teilbar ist. Erst wenn diese Vortests positiv verlaufen sind, wird das Miller-Rabin-Verfahren angewendet.

16 Stromchiffren

Wie Sie im vorhergehenden Kapitel erfahren haben, erzeugt ein Pseudozufallsgenerator aus einem kurzen Startwert eine beliebig lange Zufallsfolge. Alice und Bob können das ausnutzen, um auf einfache Weise Daten zu verschlüsseln. Dazu verwenden sie die besagte Zufallsfolge wie einen One-Time-Pad. Dies bedeutet, dass Absenderin Alice den Klartext mit der Zufallsfolge bitweise exklusiv-oder-verknüpft, wodurch der Geheimtext entsteht. Empfänger Bob macht die Verschlüsselung rückgängig, indem er eine Exklusiv-oder-Verknüpfung der Zufallsfolge mit dem Geheimtext durchführt. Voraussetzung ist, dass Alice und Bob hierbei die gleiche Zufallsfolge verwenden. Diese Bedingung ist erfüllt, wenn die beiden den gleichen Pseudozufallsgenerator (also die gleiche Fortschaltfunktion) sowie den gleichen Startwert verwenden. Falls die Fortschaltfunktion schlüsselabhängig ist, müssen Alice und Bob zudem den gleichen Schlüssel verwenden.

16.1 Aufbau und Eigenschaften von Stromchiffren

Ein Verfahren, das eine Zufallsfolge generiert, die für die Exklusiv-oder-Verknüpfung mit einem Klartext verwendet wird, wird **Stromchiffre** genannt. Eine Stromchiffre ist ein symmetrisches Verschlüsselungsverfahren, das Alice und Bob ähnlich wie den DES oder AES einsetzen können. Wie Sie sich erinnern, werden DES, AES und die anderen bisher in diesem Buch beschriebenen symmetrischen Verschlüsselungsverfahren als Blockchiffren bezeichnet. Stromchiffren sind eine gleichwertige Alternative dazu. Alle gängigen symmetrischen Verschlüsselungsverfahren gehören entweder zu den Block- oder zu den Stromchiffren.

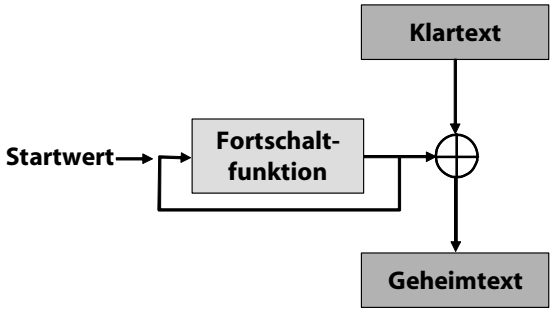


Abb. 16–1 *Bei einer Stromchiffre wird die Ausgabe eines Pseudozufallsgenerators mit dem Klartext verknüpft.*

Die Zufallsfolge, die eine Stromchiffre generiert, wird als **Keystream** bezeichnet. Die wichtigste Regel im Zusammenhang mit Stromchiffren lautet (wie beim One-Time-Pad): Ein Keystream darf auf keinen Fall mehrfach verwendet werden. Halten sich Alice und Bob nicht daran, dann kann Mallory eine Ciphertext-Only-Attacke starten, indem er zwei Geheimtexte, die mit demselben Keystream generiert wurden, miteinander exklusiv-oder-verknüpft. Als Ergebnis erhält er dann eine Exklusiv-oder-Verknüpfung der beiden Klartexte, was in vielen Fällen für eine erfolgreiche Kryptoanalyse ausreicht.

Wie Sie in Abschnitt 19.1 erfahren werden, kann man eine Blockchiffre in verschiedenen Betriebsarten nutzen. Die Betriebsarten OFB, CFB und CTR sehen vor, dass der Klartext mit dem Ergebnis einer kryptografischen Operation exklusiv-oder-verknüpft wird. Man kann eine Stromchiffre daher auch als Verfahren definieren, das für die Nutzung in einem dieser Modi (oder einem anderen, der einen Keystream vorsieht) geschaffen wurde. Fast alle bedeutenden Stromchiffren nutzen den OFB-Modus oder eine Verallgemeinerung davon, während CFB und CTR nur eine Nebenrolle spielen. Die folgenden Abschnitte beziehen sich daher ausschließlich auf die (gegebenenfalls verallgemeinerte) OFB-Variante.

16.1.1 Wie eine Stromchiffre funktioniert

Im Gegensatz zu einem Pseudozufallsgenerator muss eine Stromchiffre genau spezifiziert sein, damit es zwischen Alice und Bob keine Interoperabilitätsprobleme gibt. Anders als die im vorhergehenden Kapitel beschriebenen Pseudozufallsgeneratoren lassen die gängigen Stromchiffren daher keinen oder nur sehr wenig Spielraum für die Implementierung. Wie ein Pseudozufallsgenerator arbeitet auch eine Stromchiffre mit einer Fortschaltfunktion. Darüber hinaus gibt es einen **Status**. Dies ist eine Variable (sie kann in mehrere Teilvariablen aufgeteilt sein), in der das Ergebnis der letzten Fortschaltung gespeichert wird.

Bevor ein Keystream generiert wird, ist eine **Initialisierung** des Status notwendig. Der Startwert des Status kann geheim sein und als Schlüssel verwendet werden. Bei den meisten Verfahren ist dies jedoch nicht der Fall – stattdessen ist die Fortschaltfunktion schlüsselabhängig. Ein nichtgeheimer Startwert wird als **Initialisierungsvektor** bezeichnet. Durch diesen können Alice und Bob eine Wiederverwendung der gleichen Zufallsfolge vermeiden, indem sie für jeden Klartext einen anderen Wert wählen. Es kann sich dabei beispielsweise um eine Zufallszahl handeln. Möglich ist auch ein Zähler, dessen Wert nach jeder Verwendung erhöht wird. Da der Initialisierungsvektor nicht geheim sein muss, kann ihn Alice zusammen mit der verschlüsselten Nachricht unverschlüsselt an Bob schicken.

Moderne Stromchiffren sehen meist eine etwas komplexere Initialisierung vor. Oft werden im Rahmen dieser Initialisierung einige Iterationen ohne Keystream-Generierung durchgeführt, um die Sicherheit zu erhöhen. Der Inhalt des Status wird bei neueren Stromchiffren meist nicht direkt als Keystream verwendet. Stattdessen wird der Status einem Zwischenschritt zugeführt, der **Keystream-Extraktion** genannt wird. Erst das Ergebnis dieser Operation wird mit dem Klartext bzw. Geheimtext exklusiv-oder-verknüpft. Beachten Sie, dass die Keystream-Extraktion per Definition nicht den Status verändert, sondern lediglich den Ausgabewert aus diesem berechnet. Sowohl die Initialisierung als auch die Keystream-Extraktion können schlüsselabhängig sein.

Eine wichtige Kenngröße einer Stromchiffre ist die Zahl der Keystream-Bits, die pro Iteration generiert werden. Hierbei gibt es große Unterschiede – zwischen einem Bit und mehreren Hundert Bit pro Fortschaltung reicht das Spektrum. Bei manchen Verfahren ist die Bit-Zahl pro Iteration sogar variabel. Meistens ist die Zahl der jeweils generierten Keystream-Bits kleiner als die Blocklänge einer Blockchiffre (diese liegt meist bei 64 oder 128 Bit).

Die Funktionsweise einer Stromchiffre ähnelt in einigen Belangen der einer kryptografischen Hashfunktion. So arbeiten auch Letztere mit einem Status (oft als Kettenvariablen bezeichnet), und die zur Berechnung des Hashwerts eingesetzte Kompressionsfunktion hat eine ähnliche Aufgabe wie die Fortschaltfunktion einer Stromchiffre. Einige neuere Stromchiffren generieren daher nicht nur einen Keystream, sondern bilden auch gleich einen schlüsselabhängigen Hash-

wert. Allerdings konnte sich bisher kein Verfahren mit dieser Eigenschaft in der Praxis durchsetzen.

Wie Sie auf den folgenden Seiten mitbekommen werden, spielt bei Stromchiffren die Frage, ob eine solche in Hardware oder Software implementiert werden soll, eine noch wichtigere Rolle als bei Blockchiffren. Für Hardware geeignete Stromchiffren nutzen meist LFSRs, NFSRs oder ähnliche Konstrukte und generieren nur ein paar wenige Keystream-Bits pro Iteration. Der Status fällt hierbei meistens eher klein aus (beispielsweise 200 bis 300 Bit), um die benötigte Speichermenge auf dem jeweiligen Chip möglichst klein zu halten. Für Software entwickelte Stromchiffren generieren dagegen meist mehr Keystream-Bits pro Iteration (teilweise mehrere Hundert). Der Status kann ebenfalls etwas größer sein und mehrere KByte in Anspruch nehmen, da (beispielsweise auf einem PC) meist genügend Speicherplatz zur Verfügung steht.

16.1.2 Angriffe auf Stromchiffren

Wie bei jedem symmetrischen Verschlüsselungsverfahren stehen Bösewicht Mallory auch bei einer Stromchiffre drei Angriffswege zur Verfügung: die Ciphertext-Only-, die Known-Plaintext- und die Chosen-Plaintext-Attacke. Allerdings ist diese Einteilung bei Stromchiffren nicht besonders wichtig. So bringt eine Chosen-Plaintext-Attacke auf eine Stromchiffre Mallory gegenüber einer Known-Plaintext-Attacke keinen Vorteil – in beiden Fällen kann er nicht mehr tun, als mit dem Klartext aus dem Geheimtext den Keystream zu berechnen, um diesen anschließend zu untersuchen. Auch bei einer Ciphertext-Only-Attacke ist der Keystream Mallorys erstes Ziel – mit erratenen Klartextteilen oder statischen Eigenschaften kann er Informationen über diesen erlangen. Ein Angriff auf eine Stromchiffre sieht daher so gut wie immer wie folgt aus: Mallory kennt einen Teil des (unendlich langen) Keystreams und versucht aus diesem, den Schlüssel oder ihm unbekannte Keystream-Segmente zu ermitteln. Bei einer guten Stromchiffre darf dies naturgemäß nicht möglich sein, selbst wenn Mallory sehr große Teile des Keystreams zur Verfügung stehen. Wie bei Blockchiffren kann es Mallory auch bei Stromchiffren mit differenzieller und linearer Kryptoanalyse versuchen. Auch schwache Schlüssel und eine Related-Key-Attacke können Ansatzpunkte sein.

16.1.3 Stromchiffren und Blockchiffren im Vergleich

Stromchiffren sind weniger gut untersucht und weniger verbreitet als Blockchiffren. Dies hat vor allem historische Gründe. Der DES, die Mutter aller symmetrischen Verfahren, ist eine Blockchiffre. Deshalb waren die ersten DES-Alternativen ebenfalls Blockchiffren. Auch am AES-Wettbewerb durften nur Blockchiffren teilnehmen. Diese Entwicklung hat dazu geführt, dass Stromchiffren in der kryptografischen Forschung lange vernachlässigt wurden. Bis heute ist das Know-how der Kryptografen in diesem Bereich geringer als bei den Blockchiffren. Dies

schlägt sich auch darin nieder, dass in den letzten Jahren deutlich mehr Stromchiffren als Blockchiffren geknackt wurden. Allerdings gibt es aus den letzten Jahren einige äußerst interessante Neuentwicklungen, die eine hohe Performanz bieten und trotzdem bisher keine Sicherheitsmängel offenbart haben.

Stromchiffren hatten lange Zeit einen entscheidenden Vorteil gegenüber Blockchiffren: Sie verbrauchten weniger Ressourcen und konnten trotzdem hohe Verschlüsselungsgeschwindigkeiten erzielen. Doch auch hier gibt es neue Entwicklungen. Einige Verfahren, wie etwa PRESENT, haben in den letzten Jahren gezeigt, dass auch Blockchiffren extrem ressourcensparend arbeiten können. Man darf gespannt sein, wie die Entwicklung von Block- und Stromchiffren in den nächsten Jahren verlaufen wird. Davon abgesehen haben Stromchiffren folgende Vorteile:

- Alice und Bob können den Keystream einer Stromchiffre vorausberechnen und puffern, was in vielen Szenarien die Performanz erhöht.
- Tritt im Geheimtext ein Bit-Fehler auf, dann ist bei einer Stromchiffre nur ein Bit des Klartexts zerstört und nicht (wie bei einer Blockchiffre) ein ganzer Block.

Von Nachteil bei Stromchiffren sind folgende Aspekte:

- Die hohe Verschlüsselungsgeschwindigkeit wird oft durch eine relativ aufwendige Initialisierung erkauft. Ein etwaiger Performanzvorteil macht sich daher oft nur bei längeren Klartexten bemerkbar.
- Wenn Mallory ein Bit des Geheimtexts ändert, dann ändert sich auch das entsprehende Bit im Klartext (und umgekehrt). Dies ist ein klarer Verstoß gegen die Zufallsorakel-Eigenschaft. Wenn Mallory Teile des Klartexts kennt, dann kann er diesen gezielt manipulieren. Dies ist ein Grund dafür, dass Stromchiffren mit integrierter kryptografischer Hashfunktion eine sinnvolle Sache sind.
- Derselbe Keystream (und damit dasselbe Paar von Initialisierungsvektor und Schlüssel) einer Stromchiffre darf nicht mehrfach verwendet werden.
- Ein beliebiger Block eines Geheimtexts lässt sich nicht einzeln entschlüsseln.
- Die meisten Stromchiffren lassen sich nicht ohne Weiteres im CTR-Modus betreiben (siehe Abschnitt 19.1). Ausnahmen sind solche Stromchiffren, die speziell für den CTR-Modus entwickelt wurden. Mit Salsa20 werden Sie in diesem Kapitel eine solche kennenlernen.

16.2 RC4

Die über lange Zeit hinweg bekannteste und bedeutendste Stromchiffre hat den Namen **RC4**. Sie gehört zu den zahlreichen Entwicklungen von Ron Rivest, dem RSA-Miterfinder und gegenwärtig vielleicht bedeutendsten Kryptografen der Welt. RC4 steht für *Rivest Cipher 4* und gehört namenstechnisch in eine Reihe mit den Blockchiffren RC2, RC5 und RC6, die Sie bereits in Abschnitt 9.3 ken-

nengelernt haben. Die Funktionsweise ist jedoch eine völlig andere als bei den anderen drei Mitgliedern der RC-Familie.

Rivest entwickelte RC4 im Jahr 1987. Zum Ärger vieler Kollegen hielt er die Funktionsweise geheim und verhinderte so eine Analyse durch andere Experten. So war die Schadenfreude groß, als 1994 der Quellcode des Verfahrens im Internet auftauchte (mit unbekannter Herkunft, versteht sich). Bis heute ist keine offizielle Spezifikation von RC4 erschienen, doch längst kursieren genügend inoffizielle Beschreibungen, wobei manche das Verfahren ARCFOUR nennen, um möglichen Streitigkeiten aus dem Weg zu gehen.

Nach dem Bekanntwerden der Funktionsweise wunderte sich die gesamte Fachwelt über den simplen Aufbau von RC4. Es gibt Programmierer, die das Verfahren aus dem Gedächtnis implementieren können (versuchen Sie das einmal mit dem DES oder AES). Alice und Bob können RC4 mit etwas Übung sogar ohne Computerunterstützung von Hand einsetzen – auch das dürfte den beiden mit anderen Verfahren schwerfallen.

16.2.1 Funktionsweise von RC4

RC4 ist vor allem für Softwareimplementierungen geeignet. Die Schlüssellänge ist variabel und kann zwischen einem und 2.048 Bit betragen (vorteilhaft ist ein Vielfaches von acht). Am sinnvollsten ist eine Schlüssellänge von 128 oder 256 Bit. Der Schlüssel beeinflusst ausschließlich den Startwert (Seed), während die Fortschaltfunktion nicht schlüsselabhängig ist. Das Herzstück des Verfahrens ist ein aus 256 Byte-Variablen bestehender Status, in dem jede Zahl zwischen 0 und 255 genau einmal vorkommt. Die 256 Variablen werden im Folgenden als s_0, s_1, ..., s_{255} bezeichnet. Pro Fortschaltung erzeugt RC4 ein Byte Keystream. Es gibt keinen Initialisierungsvektor, Alice und Bob können jedoch einen Teil des potenziell 2.048 Bit langen Schlüssels als solchen verwenden. Zur Beschreibung der Funktionsweise benötigen wir zwei Variablen i und j, die am Anfang den Wert 0 besitzen, sowie eine weitere Variable k. Der Status s_0, s_1, ..., s_{255} wird nun mit folgender Fortschaltfunktion bearbeitet:

$i := i + 1 \pmod{256}$
$j := j + s_i \pmod{256}$
vertausche s_i und s_j
$k := s_i + s_j \pmod{256}$

Das Keystream-Byte, das nun ausgegeben wird, ist s_k (die Keystream-Extraktion besteht also lediglich aus der Auswahl eines Bytes). Die Initialisierung sieht vor, dass aus dem Schlüssel die Anfangsbelegung des Status berechnet wird. Dazu wird die Zahlenfolge mit s_0=0, s_1=1, s_2=2, ..., s_{255}=255 initialisiert. Zudem wird der Schlüssel byteweise in eine weitere Zahlenfolge k_0, k_1, k_2, ..., k_{255} geschrieben. Bei einer Schlüssellänge von 2.048 Bit geht dies genau auf. Ist der Schlüssel

kürzer als 2.048 Bit, dann wird er mehrfach hintereinander geschrieben. Bevor die Verschlüsselung losgehen kann, wird folgender Algorithmus zur Initialisierung abgearbeitet:

Für i von 0 bis 255
  $j := j + s_i + k_i \pmod{256}$
  vertausche s_i und s_j

Damit ist RC4 vollständig beschrieben – einfacher geht es kaum.

16.2.2 Bewertung von RC4

RC4 ist nicht nur besonders einfach, sondern auch besonders schnell. Die Verschlüsselungsgeschwindigkeit liegt bei Softwareimplementierungen etwa zehnmal höher als beim DES. Diese Eigenschaften haben RC4 zu einem weit verbreiteten Verfahren gemacht, das beispielsweise von den gängigen Webbrowsern genutzt wird und bei der Absicherung von Wireless LANs (siehe Abschnitt 33.2) zum Einsatz kommt. Bis vor ein paar Jahren war RC4 die Stromchiffre schlechthin und unter den für Softwareumsetzungen gedachten Stromchiffren nahezu konkurrenzlos.

RC4 galt lange als sicher. In den letzten Jahren haben jedoch verschiedene Kryptografen mehrere Schwächen aufgedeckt, die zumindest Anlass zur Vorsicht geben. Bereits 1997 zeigte sich, dass RC4 bei statistischen Tests geringfügig schlechter abschneidet als ein echter Zufallsgenerator [Golic]. 2001 fanden die beiden Kryptografen Scott Fluhrer und David McGrew statistische Auffälligkeiten in den von RC4 generierten Folgen, die ausreichten, um eine RC4-Zufallsfolge der Länge 1 Gigabyte von einer echten Zufallsfolge zu unterscheiden [FluMcG]. Ein Jahr später zeigten Itsik Mantin und Adi Shamir, dass das zweite Byte einer RC4-Zufallsfolge mit doppelt so hoher Wahrscheinlichkeit Null beträgt wie bei einer echten Zufallsfolge [ManSha]. Mit diesen Ergebnissen wurde klar, dass RC4 kein perfektes Zufallsorakel ist.

Als bisher größtes Problem hat sich erwiesen, dass kleine Teile eines RC4-Schlüssels einen überproportional großen Einfluss auf die ersten Bytes einer RC4-Zufallsfolge ausüben. Das erste Byte einer solchen Folge ist sogar nur von drei von insgesamt 256 Werten abhängig, die bei der Schlüsselaufbereitung entstehen. Aus Sicht eines Chiffren-Designers bedeutet dies: Die Konfusionseigenschaft ist bei RC4 nicht in ausreichendem Maße gegeben.

Die Tatsache, dass das erste RC4-Zufallsfolgen-Byte nur von einem kleinen Teil des Schlüssels abhängt, lieferte die Vorlage für den bisher wichtigsten Angriff auf RC4. Dieser stammt in seiner ursprünglichen Form von Fluhrer, Mantin und Shamir und wurde 2001 veröffentlicht [FlMaSh]. Der Angriff funktioniert für verschiedene Schlüssellängen, uns soll jedoch nur die Schlüssellänge 128 Bit interessieren. Die Vorgehensweise setzt zunächst voraus, dass Alice und Bob einen

gemeinsamen geheimen Schlüssel besitzen, der 104 Bit lang ist. Die fehlenden 24 Bit legen die beiden für jeden RC4-Verschlüsselungsvorgang neu fest, wobei keine Geheimhaltung dieses 24-Bit-Werts vorgesehen ist. Darüber hinaus wird vorausgesetzt, dass Mallory jeweils das erste Klartext-Byte einer verschlüsselten Nachricht kennt (es handelt sich also um eine Known-Plaintext-Attacke). Diese Voraussetzungen klingen zwar etwas konstruiert, entsprechen jedoch genau dem Einsatzszenario von RC4 im WLAN-Krypto-Standard WEP (siehe Abschnitt 33.2.1), auf den Fluhrer, Mantin und Shamir abzielten.

Der Fluhrer-Mantin-Shamir-Angriff nutzt die besagte RC4-Konfusionsschwäche im ersten Zufallsfolgen-Byte und ermöglicht es Mallory, den geheimen 104-Bit-Schlüssel zu ermitteln, den Alice und Bob verwenden. Für das Knacken eines solchen Schlüssels benötigt Mallory etwa 4 bis 6 Millionen RC4-Geheimtexte, die mit unterschiedlichen 24-Bit-Werten verschlüsselt wurden und von denen er jeweils das erste Klartext-Byte kennt. 2007 veröffentlichten die drei Kryptografen Andrei Pyshkin, Erik Tews und Ralf-Philipp Weinmann von der Universität Darmstadt eine Weiterentwicklung dieser Attacke, die deutlich weniger Klartext benötigt [PyTeWe]. Ihre Arbeit basiert auf einer Analyse ihres Kollegen Andreas Klein [Klein]. Welche Auswirkungen diese Resultate auf die Praxis haben, können Sie im Abschnitt über WEP nachlesen (Abschnitt 33.2.1).

Es ist nicht besonders schwierig, den Fluhrer-Mantin-Shamir-Angriff durch eine geeignete Implementierung zu verhindern. Zum einen sollte hierbei nach Möglichkeit der gesamte RC4-Schlüssel geheim bleiben – falls nicht, sollte der bekannte Teil mithilfe einer kryptografischen Hashfunktion mit dem unbekannten Teil kombiniert werden. Zum anderen können Alice und Bob vereinbaren, dass sie die ersten RC4-Zufallsfolgen-Bytes nicht verwenden, sondern erst beim zehnten oder hundertsten anfangen. Und schließlich ist ein Schlüsselwechsel spätestens nach einigen Hunderttausend Nachrichten angebracht. Leider wandten die WEP-Entwickler keine der drei genannten Maßnahmen an.

Wie ist also nun die Sicherheit von RC4 einzuschätzen? Klar ist, dass der gute Ruf des Verfahrens in den letzten Jahren einige Schrammen erhalten hat. Die bisher entdeckten Schwächen sind jedoch entweder nicht praxisrelevant oder leicht zu beheben. Am Ende bleibt also die Erkenntnis: Die Einfachheit und Schnelligkeit von RC4 erkauft man sich durch Fallstricke beim praktischen Einsatz und durch einige theoretische Schwächen. In neuen Implementierungen sollten Sie daher lieber ein anderes Verfahren verwenden (etwa den AES). Wer dagegen nicht auf RC4 verzichten will oder kann, sollte sich die Schwächen des Verfahrens genau ansehen und bei der Implementierung auf eine Vermeidung der entsprechenden Fehler achten.

16.3 A5

Eine weitere bedeutende Stromchiffre trägt den Namen **A5**. Dabei handelt es sich um das Verschlüsselungsverfahren, das weltweit von allen GSM-Handys eingesetzt wird, um die Übertragung zur nächsten Empfangsstation zu verschlüsseln – also die Teilstrecke, die per Funk übertragen wird. Dass es sinnvoll ist, im Mobilfunk Verschlüsselung einzusetzen, haben Sie bereits in Abschnitt 3.3.5 erfahren.

Der A5-Algorithmus entstand Mitte der 80er Jahre zusammen mit dem GSM-Standard innerhalb der europäischen Standardisierungsbehörde ETSI. Über die Leute, die das Verfahren entwickelt haben, gibt es in der Literatur keine Informationen. Ähnlich wie bei RC4 war auch die Funktionsweise von A5 zunächst geheim. Im Laufe der Zeit wurden jedoch einige Details der Funktionsweise öffentlich, und inzwischen ist das gesamte Verfahren bekannt. Im Internet kursieren mehrere Beschreibungen, beispielsweise [BrGoWa].

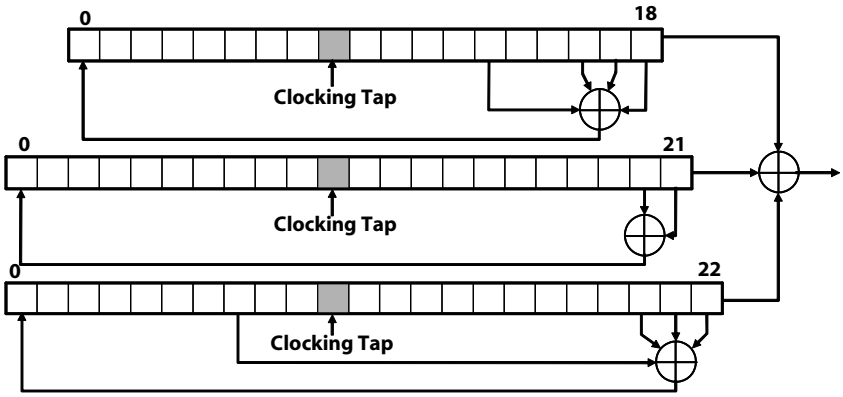


Abb. 16–2 *Funktionsweise des Verschlüsselungsverfahrens A5. Es sieht drei LFSRs vor, deren Ausgabe mit dem Klartext verknüpft wird. Inzwischen ist bekannt, dass es sich dabei um ein vergleichsweise schwaches Verfahren handelt.*

16.3.1 Funktionsweise von A5

A5 ist eine Stromchiffre, die auf LFSRs basiert. Es werden drei LFSRs verwendet, welche die Länge 19, 22 und 23 haben. Welche Variablen in die Fortschaltfunktion eingehen, ist in Abbildung 16–2 ersichtlich. Die Zufallsfolge, mit der Alice ihre Nachricht verschlüsseln kann, wird gewonnen, indem die Ausgabe-Bits der drei LFSRs miteinander exklusiv-oder-verknüpft werden. Wenn ein Zufalls-Bit generiert wird, werden dafür nicht notwendigerweise alle drei LFSRs fortgeschaltet. Stattdessen wird nur dann fortgeschaltet, wenn eine bestimmte Variable (**Clocking Tap**) des jeweiligen LFSR einen bestimmten Wert hat. Die Clocking Taps liegen etwa in der Mitte des jeweiligen LFSR (die Positionen sind genau fest-

gelegt, siehe Abbildung 16–2). Wenn es in den drei Clocking Taps mehr Einsen als Nullen gibt, dann werden diejenigen LFSRs fortgeschaltet, die den Wert 1 haben. Sind die Nullen in den Clocking Taps in der Mehrzahl, dann werden analog die LFSR mit dem Clocking-Tap-Wert 0 fortgeschaltet. Auf diese Weise werden immer mindestens zwei LFSR für ein Zufalls-Bit fortgeschaltet. Die Schlüssellänge von A5 beträgt 64 Bit. Mithilfe dieser 64 Bit wird nach einem bestimmten Ablauf eine Anfangsbelegung der drei LFSRs generiert. Bevor das Verfahren Zufalls-Bits ausgibt, werden noch einige »Leerrunden« gefahren, was eine zusätzliche Sicherheit bringen soll.

Die hier beschriebene Methode wird übrigens auch als A5/1 bezeichnet. Sie wird vor allem in Europa verwendet. Daneben gibt es noch die abgeschwächte Version A5/2, die beispielsweise in Australien eingesetzt wird. A5/2 sieht ein viertes LFSR vor, das die Fortschaltung der drei anderen LFSRs regelt.

16.3.2 Bewertung von A5

A5 ist schnell (vor allem in Hardware) und die Funktionsweise recht einfach. Ist A5 damit eine ernst zu nehmende Alternative zu AES, Twofish und RC6? Nein, denn A5 ist unsicher. Je mehr in den letzten Jahren über die Funktionsweise durchsickerte, desto klarer wurde, dass A5 nicht den Anforderungen an ein modernes kryptografisches Verfahren entspricht. Die Idee der drei LFSRs erwies sich dabei zwar nicht als grundsätzlich falsch. Die Entwickler von A5 haben aber offensichtlich die Länge und die Tap-Variablen nicht optimal gewählt. Die Art und Weise, wie beim Design von A5 mit guten Werkzeugen falsch umgegangen wurde, ließ sogar den Verdacht aufkommen, dass die Entwickler mit A5 bewusst einen knackbaren Algorithmus etablieren wollten.

Es gibt mehrere beunruhigende Kryptoanalyse-Resultate, die in den letzten Jahren über A5/1 veröffentlicht wurden [LuWeZe]. Das bisher spektakulärste Ergebnis stammt von Biryukov, Shamir und Wagner [BiShWa]. Sie beschrieben eine Methode, mit der es möglich ist, den Schlüssel durch eine Known-Plaintext-Attacke mit einem PC innerhalb einer Sekunde zu berechnen. Abgesehen von einer sehr umfangreichen Vorberechnung wird dazu der Klar- und Schlüsseltext benötigt, der in den ersten zwei Minuten eines Telefongesprächs produziert wird. Noch schlimmer sieht es mit der Sicherheit von A5/2 aus. Gemäß den Kryptoanalyse-Ergebnissen der letzten Jahre liegt die effektive Schlüssellänge von A5/2 bei etwa 17 Bit. Da wird die vollständige Schlüsselsuche zum Kinderspiel.

16.4 E_0

In Abschnitt 33.3 dieses Buchs wird das Thema Bluetooth im Mittelpunkt stehen. Es handelt sich dabei um eine Technik für die drahtlose Datenübertragung über kurze Distanzen. Speziell für Bluetooth haben verschiedene Experten eine Strom-

chiffre entwickelt, die den Namen E_0 trägt [Blueto]. E_0 ist kostengünstig in Hardware zu implementieren und arbeitet in dieser Form schnell und energiesparend. E_0 ist damit für PDAs, PC-Peripherie, Headsets und ähnliche Geräte geeignet, die eine Bluetooth-Schnittstelle unterstützen. Ob E_0 für diesen Anwendungsbereich auch sicher genug ist, ist dagegen eine andere Frage. Wie wir im Folgenden sehen werden, gibt es durchaus Zweifel daran.

16.4.1 Funktionsweise von E_0

Im Mittelpunkt von E_0 stehen vier LFSRs, die wir von LFSR1 bis LFSR4 durchnummerieren wollen. In dieser Reihenfolge haben die LFSRs die Längen 25, 31, 33 und 39 Bit, was zusammen 128 Variablen ergibt. Man kann E_0 daher eine Schlüssellänge von 128 Bit zuschreiben, auch wenn die weiter unten beschriebene Schlüsselaufbereitung diese Tatsache relativiert.

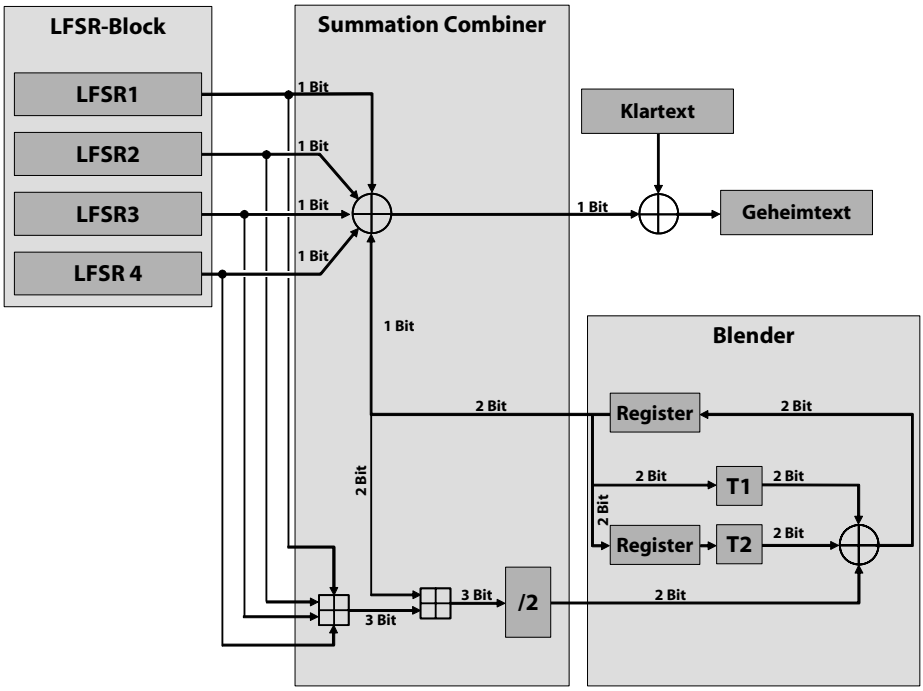


Abb. 16–3 *E_0 ist eine Stromchiffre, die auf vier LFSRs basiert. Außerdem geht die Ausgabe einer zusätzlichen Komponente (Blender) in die Verschlüsselung mit ein.*

Jedes der vier genannten LFSRs besitzt vier Tap-Variablen. Bei LFSR1 sind dies die Variablen Nummer 8, 12, 20 und 25, bei LFSR2 sind es 12, 16, 24, 31, bei LFSR3 sind es 4, 24, 28 und 33 sowie bei LFSR1 die Variablen 4, 28, 36 und 39. Die Größen der LFSRs und die Lage der Tap-Variablen sind so gewählt, dass alle

2^{128} möglichen Variablenbelegungen durchlaufen werden, bevor die LFSRs wieder in den Ausgangszustand zurückkommen (einzige Ausnahme ist die Belegung mit 128 Nullen). Die Ausgabe der LFSRs erfolgt nicht über die jeweils rechts gelegenen Variablen. Stattdessen liefern LFSR1 und LFSR2 den Inhalt von Variable 24 als Ausgabe, während LFSR3 und LFSR4 jeweils Variable 32 ausgeben.

Der Aufbau von E_0 ist in Abbildung 16–3 ersichtlich. Das Design sieht drei Bestandteile vor:

- *LFSR-Block*: Der LFSR-Block enthält die vier beschriebenen LFSRs.
- *Summation Combiner*: Dieser realisiert zwei logische Funktionen, die beide die vier Ausgabe-Bits der LFSRs entgegennehmen. Die erste logische Funktion nimmt noch ein fünftes Bit hinzu, das aus dem Blender stammt, und liefert als Ausgabe die Exklusiv-oder-Verknüpfung der fünf Eingabe-Bits. Die Ausgaben bilden zusammen den Zufallsstrom, mit dem der Klartext verknüpft wird. Die zweite logische Funktion liefert zwei Bit für den Blender und nimmt dazu zwei Bit aus dem Blender entgegen. Der Aufbau dieser Funktion ist aus Abbildung 16–3 ersichtlich.
- *Blender*: Der Blender ist ebenfalls eine einfache logische Funktion, jedoch nicht zustandslos. Zum Blender gehören zwei Zwei-Bit-Register, die in jedem Taktzyklus den aktuellen Eingabewert speichern und den vorhergehenden ausgeben. Die beiden Register können insgesamt 16 Zustände annehmen. Wie in Abbildung 16–3 zu sehen ist, nimmt der Blender zwei Zwei-Bit-Werte entgegen und gibt einen Zwei-Bit-Wert aus. Der Aufbau des Blenders sieht neben den beiden Registern noch eine Exklusiv-oder-Verknüpfung sowie die beiden Funktionen T_1 und T_2 vor. Letztere bilden jeweils einen Zwei-Bit-Wert nach einer festen Tabelle auf einen anderen Zwei-Bit-Wert ab (es handelt sich also um zwei Mini-S-Boxen).

Die Generierung des Zufallsstroms funktioniert also bitweise. Zum Verschlüsseln und Entschlüsseln wird der Zufallsstrom per Exklusiv-oder-Verknüpfung mit dem Klartext bzw. dem Schlüsseltext verbunden.

Schlüsselaufbereitung von E_0

Die vier LFSRs, die E_0 vorsieht, bestehen zusammen aus 128 Variablen. Am einfachsten wäre es, einen 128-Bit-Schlüssel zu nehmen, diesen auf die jeweiligen Variablen zu verteilen und loszulegen (für die Initialisierung der Register könnte man eine Konstante verwenden oder den Schlüssel um vier Bit verlängern). Diese simple Methode der Schlüsselaufbereitung erschien den E_0-Schöpfern jedoch zu unsicher. Sie sahen wohl das Problem, dass Alice und Bob zweimal denselben Schlüssel verwenden könnten, was bei einer Stromchiffre nicht passieren darf.

Stattdessen sieht E_0 eine relativ aufwendige Schlüsselaufbereitung vor. Diese setzt voraus, dass ein 128-Bit-Schlüssel K, eine 48-Bit-Bluetooth-Geräteadresse,

eine 26-Bit-Uhrzeit sowie eine 6-Bit-Konstante (111001) vorhanden sind. Wenn man so will, ist *K* der Schlüssel, während die drei anderen Werte zusammen einen 80-Bit-Initialisierungsvektor bilden. Die Uhrzeit soll sicherstellen, dass kein Initialisierungsvektor doppelt verwendet wird (ist dies der Fall, dann ist eine doppelte Verwendung von *K* unkritisch).

Schlüssel und Initialisierungsvektor haben zusammen die Länge 208 Bit. Als weiterer Input für die Schlüsselaufbereitung wird die gewünschte effektive Schlüssellänge *L* benötigt. *L* kann alle ganzzahligen Werte zwischen 1 und 16 annehmen. Die Schlüsselaufbereitung läuft nun wie folgt ab (wir gehen davon aus, dass Alice diesen Vorgang durchführt):

1. Zunächst bringt Alice den Schlüssel *K* auf eine effektive Schlüssellänge von *L* Byte. Dieser Schritt wurde in die Schlüsselaufbereitung aufgenommen, weil der Export von starker Kryptografie aus den USA bis 2000 verboten war (und weil es einige andere Länder gibt, in denen der Einsatz sicherer kryptografischer Verfahren Probleme bereiten kann). US-Hersteller von Bluetooth-Geräten benötigten daher eine Möglichkeit, die Schlüssellänge zu begrenzen (meist auf 40 Bit). Da sich die Rechtslage inzwischen entspannt hat, kann Alice diesen Schritt in der Regel übergehen. Ist jedoch eine Verkürzung der effektiven Schlüssellänge gefordert, dann muss Alice eine Rechenoperation durchführen, bei der *K* als Polynom betrachtet und mit zwei anderen Polynomen verknüpft wird (siehe dazu Abschnitt 13.1.1). Durch diese Operation wird *K* zwar nicht kürzer (die Länge bleibt bei 128 Bit), doch die Zahl der möglichen Schlüssel verkleinert sich auf 2^{8L}. Die effektive Schlüssellänge beträgt dadurch *L* Byte.
2. Nun muss Alice die 208 Bit des Schlüssels und des Initialisierungsvektors in die LFSRs einbringen. Dazu ordnet sie jedes der 208 Bit nach einer im Standard vorgegebenen Tabelle einem der vier LFSRs zu (die Tabelle ist so gestaltet, dass es eine gute Durchmischung gibt). LFSR1 und LFSR3 erhalten je 49 Bit zugeordnet, LFSR2 und LFSR4 erhalten je 55 Bit zugeordnet. Nun beginnt für alle vier LFSRs parallel folgender Ablauf (die Feedback-Funktion der LFSRs ist dabei zunächst deaktiviert):
   a) Nimm das nächste Bit aus den 49 bzw. 55 zugeordneten Bit und schiebe es von links in das LFSR. Wiederhole diesen Vorgang einmal pro Taktzyklus.
   b) Ist das LFSR voll, dann aktiviere die Feedback-Funktion. Schiebe jeweils das Ergebnis der Feedback-Funktion exklusiv-oder-verknüpft mit dem nächsten zugeordneten Bit von links in das LFSR. Wiederhole diesen Vorgang einmal pro Taktzyklus.
   c) Sind alle zugeordneten Bit eingeschoben, dann schiebe nur noch das Ergebnis der Feedback-Funktion ein (dies entspricht dem normalen Betrieb des LFSR). Wiederhole diesen Vorgang einmal pro Taktzyklus.

4. Dieser Ablauf endet mit dem 55. Taktzyklus, wenn alle 210 Bit aus Schlüssel und Initialisierungsvektor eingeschoben sind. Ab dem 39. Taktzyklus (wenn also LFSR4 voll ist) produziert das Verfahren Zufalls-Bits (die Register werden mit 0 initialisiert). Es werden also bis hierhin 16 Zufalls-Bits produziert.
5. Alice lässt das Verfahren weiter laufen (alle LFSRs sind im Normalbetrieb) und produziert weitere 184 Zufalls-Bits (also insgesamt 200). Von diesen nimmt sie die letzten 128 und kopiert sie in die vier LFSRs. Die Register werden dabei nicht verändert.

Mit der neuen Belegung der LFSRs ist die Schlüsselaufbereitung abgeschlossen. Der Verschlüsselungsbetrieb von E_0 kann beginnen.

16.4.2 Bewertung von E_0

E_0 wurde speziell für Bluetooth entwickelt. Was Performanz und eine einfache Realisierung in Hardware anbelangt, ist das Verfahren für diese Anwendung sicherlich bestens geeignet. Bezüglich der Sicherheit von E_0 gibt es inzwischen jedoch Bedenken. Bereits 1999 zeigten Hermelin und Nyberg, dass E_0 in 2^{64} Arbeitsschritten zu brechen ist, wenn Abhörer Mallory 2^{64} Bit des Zufallsstroms kennt. Dieser Angriff ist deutlich einfacher als eine vollständige Schlüsselsuche, die 2^{128} Schritte erfordert. Fluhrer und Lucks entdeckten, dass E_0 mit 132 Bit aus dem Zufallsstrom in 2^{84} Arbeitsschritten und mit 2^{43} Bit aus dem Zufallsstrom in 2^{73} Arbeitsschritten zu knacken ist. Mit anderen Worten heißt dies: Die effektive Schlüssellänge von E_0 beträgt nur zwischen 73 und 84 Bit, selbst wenn L auf 16 gesetzt wird. 2005 veröffentlichten Lu, Meier und Vaudenay einen weiteren Angriff. Sie benötigten die jeweils ersten 24 Bit von etwa 15 Millionen Bluetooth-Paketen, um in 2^{38} Berechnungen den Schlüssel zu ermitteln.

Diese Ergebnisse machen deutlich, dass E_0 nicht die Sicherheit bietet, die man von einem modernen kryptografischen Verfahren erwartet. Ich würde daher niemandem die Nutzung von E_0 empfehlen, sofern ein anderes Verfahren zur Verfügung steht. Diese Einschätzung ist offensichtlich weit verbreitet, und so hat E_0 außerhalb von Bluetooth bisher keine Bedeutung erlangt.

Innerhalb von Bluetooth ist E_0 allerdings gesetzt und daher nicht zu vermeiden. Zum Glück sind die bisher entdeckten Schwachstellen noch nicht praxisrelevant, da sie einen zu großen Aufwand oder unrealistisch große Mengen an Bit aus dem Zufallsstrom erfordern. Auch der Angriff von Lu, Meier und Vaudenay fällt in diese Kategorie, wenn Alice und Bob den Schlüssel K oft genug wechseln. Bleibt also zu hoffen, dass die bisher bekannten Angriffe in den nächsten Jahren nicht verbessert werden.

16.5 Crypto1

Crypto1 ist eine Stromchiffre, die in den weit verbreiteten Mifare-Chips der Firma NXP (früher Philips) genutzt wird. Die Schlüssellänge beträgt 48 Bit, was nicht gerade viel ist. Mifare-Chips werden in kontaktlosen Smartcards und ähnlichen Minigeräten verwendet, die beispielsweise im öffentlichen Nahverkehr, für den Gebäudezutritt oder für bargeldlose Bezahlsysteme zum Einsatz kommen. Die Anzahl der bisher hergestellten Mifare-Chips liegt bei über 500 Millionen, wobei jedoch nicht in allen Crypto1 implementiert ist (es gibt verschiedene Mifare-Varianten, die sich deutlich unterscheiden). Der Verwendungszweck von Crypto1 auf Mifare-Chips ist das Berechnen einer Response aus einer Challenge, die beispielsweise von einer Schließanlage kommt.

Crypto1 entstand Anfang der neunziger Jahre. Offensichtlich waren die Entwickler keine Kryptografen, denn das Design des Verfahrens entspricht bei weitem nicht dem damaligen Stand der Technik. Allerdings fiel dies zunächst nicht auf, da der Hersteller die Spezifikation von Crypto1 geheim hielt. Dies ging immerhin bis zum Jahr 2007 gut, doch dann gelang es den beiden Deutschen Karsten Nohl and Henryk Plötz durch eine Analyse der Mifare-Hardware, die Funktionsweise von Crypto1 zu rekonstruieren [NohPlö]. Die beiden verrieten zwar nicht alle Details, doch bereits im März 2008 präsentierte eine niederländische Forschergruppe das komplette Design [GGMRVW]. Die Forscher von der Universität Nimwegen waren bei ihren Untersuchungen (im Gegensatz zu Nohl und Plötz) ohne eine Betrachtung der Hardware ausgekommen. Es genügte ihnen, den Chip mit einigen Daten zu füttern, die nicht dem vorgesehenen Format entsprachen, und die Ergebnisse zu analysieren.

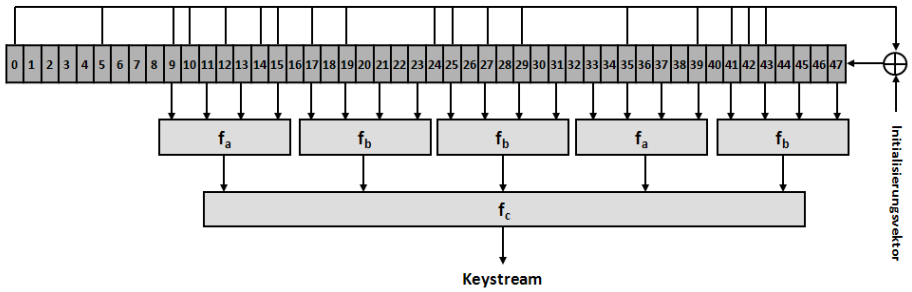


Abb. 16–4 *Crypto1 ist eine einfache und nicht besonders sichere Stromchiffre.*

16.5.1 Funktionsweise von Crypto1

Crypto1 verwendet einen Status, der aus 48 Bits besteht. Die Fortschaltfunktion ist in Abbildung 16–4 dargestellt. Sie besteht ausschließlich aus einem LFSR, das mit 18 Tap-Variablen arbeitet. Die Keystream-Extraktion (in derselben Abbildung ersichtlich) erfolgt mit den Funktionen f_a, f_b und f_c. Diese lassen sich als Ersetzungstabellen (also S-Boxen) implementieren. f_a und f_b bilden jeweils vier Bits auf ein Bit ab, während f_c fünf Bits auf ein Bit abbildet. Betrachtet man nacheinander die Eingaben 1111, 1110, 1101, 1100, ..., 0000 (für f_a und f_b) bzw. 11111, 11110, 11101, 11100, ..., 00000 (für f_c), dann ergeben sich folgende Ausgabewerte (in Klammern die zugehörige Hexadezimalzahl):

- f_a: 00100110 11000111 (26c7)
- f_b: 00001101 11010011 (0dd3)
- f_c: 01000100 01010111 11000011 10110011 (4457c3b3)

Für die Initialisierung spielt ein Initialisierungsvektor der Länge 32 Bit eine Rolle. In der Mifare-Implementierung wird dieser Initialisierungsvektor aus einer Kennnummer und einer Challenge bzw. Response gebildet. Die Initialisierung sieht vor, dass 32 Iterationen ohne Keystream-Generierung durchgearbeitet werden, wobei jeweils ein Bit des Initialisierungsvektors an der in der Abbildung gekennzeichneten Stelle eingeht.

16.5.2 Bewertung von Crypto1

Für einen Kryptografen ist schon auf den ersten Blick erkennbar, dass Crypto1 keine allzu große Sicherheit bieten kann. Sowohl die Initialisierung als auch die Fortschaltung als auch die Keystream-Extraktion enthalten zu wenig Komplexität, um sicher zu sein. Darüber hinaus ist die Schlüssellänge mit 48 Bit eindeutig zu kurz, um einer vollständigen Schlüsselsuche Gegenwehr zu leisten. Eine gewisse Sicherheit bot Crypto1 daher nur, solange die Funktionsweise geheim war.

Nachdem die Details über Crypto1 an die Öffentlichkeit gelangt waren, wurde das Verfahren zu einer leichten Beute für einige Kryptografen. Eine vollständige Schlüsselsuche mit einem PC dauerte gerade einmal neun Stunden. Die niederländische Forschergruppe, die das Design des Verfahrens veröffentlicht hatte, knackte gleich auch das Challenge-Response-Protokoll, in dessen Rahmen Crypto1 im Mifare-Chip eingesetzt wird [GGMRVW]. Und schließlich fanden Nicolas Courtois, Karsten Nohl und Sean O'Neil einen Angriff, der Crypto1 komplett demontierte [CoNoON]. Sie benötigten gerade einmal 50 Keystream-Bits und einen Initialisierungsvektor, um mit einem handelsüblichen PC innerhalb von 200 Sekunden den Schlüssel zu ermitteln. Ein so großes Maß an Unsicherheit ist selten in der Kryptografie. Es versteht sich daher von selbst, dass man von der Verwendung von Crypto1 nur abraten kann.

16.6 Die Verfahren des eSTREAM-Wettbewerbs

eSTREAM ist der Name eines Krypto-Wettbewerbs, der von 2004 bis 2008 lief und der den Zweck hatte, besonders empfehlenswerte Stromchiffren zu ermitteln. eSTREAM kam zustande, nachdem zuvor ein anderer Krypto-Wettbewerb namens NESSIE eine erstaunliche Lücke aufgezeigt hatte. Das Ziel von NESSIE war es gewesen, empfehlenswerte kryptografische Verfahren in unterschiedlichen Kategorien (z. B. Blockchiffren, Zufallsgeneratoren, kryptografische Hashfunktionen) zu prämieren. Details dazu finden sich in Abschnitt 18.5.4. In der NESSIE-Kategorie Stromchiffren gab es überraschenderweise keinen Sieger, da alle Kandidaten ernsthafte Schwächen gezeigt hatten. Dies gab den Anlass dazu, einen weiteren Wettbewerb zu starten, in dem es ausschließlich um Stromchiffren ging. Damit war eSTREAM geboren. Dessen Regeln sahen vier Kategorien vor. Gesucht wurden Stromchiffren für Hardwareimplementierungen sowie Stromchiffren für Softwareimplementierungen. In beiden Fällen wurde zwischen herkömmlichen Verfahren und solchen, die auch einen schlüsselabhängigen Hashwert generieren, unterschieden. Manche Stromchiffren gingen in mehreren Kategorien an den Start.

Es gab insgesamt 34 Einreichungen für eSTREAM. Erstaunlich viele davon wurden innerhalb kurzer Zeit geknackt, was belegt, dass das Stromchiffren-Design noch längst nicht so weit fortgeschritten ist wie das Blockchiffren-Design. Einige weitere Teilnehmer schieden wegen anderer Schwächen aus. Insbesondere mussten alle Verfahren mit Hashwert-Generierung die Segel streichen, weshalb erneut zwei Wettbewerbskategorien keinen Sieger hatten. Am Ende blieben acht als empfehlenswert eingeschätzte Stromchiffren übrig, die im April 2008 in das sogenannte eSTREAM-Portfolio aufgenommen wurden. Noch im gleichen Jahr musste die Jury jedoch ein Verfahren (F-FCSR-H) aufgrund einer neu entdeckten Schwäche aus dem Portfolio streichen.

Es gehören also sieben Stromchiffren zum elitären Kreis der eSTREAM-Gewinner. Diese sieben Verfahren schauen wir uns im Folgenden genauer an. Die ersten vier (HC-128, Rabbit, Salsa20/12 und SOSEMANUK) gehören zu den softwareoptimierten Algorithmen, die verbleibenden drei (Trivium, Grain und MICKEY) sind eher für Hardware geeignet. Die Ermittlung einer Rangliste oder eines einzelnen Siegers gehörte nicht zum eSTREAM-Reglement. Es gab jedoch bei der Fachkonferenz SASC im Jahr 2008 eine unverbindliche Abstimmung, bei der jeder Konferenzteilnehmer für jedes Verfahren eine Bewertung von geeignet (+5), neutral (0) oder ungeeignet (-5) abgeben konnte. Die SASC-Abstimmung ergab folgende Mittelwerte:

	Softwareverfahren		Hardwareverfahren	
1	Rabbit	2,80	Trivium	4,35
2	Salsa20	2,80	Grain	3,50
3	Sosemanuk	1,20	F-FCSR-H	0,52
4	HC-128	0,60	MICKEY	0,17

Weitere Angaben zum Ablauf des Wettbewerbs finden sich in der Veröffentlichung *The eSTREAM Portfolio* [eSTREA], die den Charakter eines Abschlussberichts hat.

16.6.1 HC-128

HC-128 ist eine Stromchiffre, die von Hongjun Wu von der Universität Leuven entwickelt wurde [Wu]. Es handelt sich um eine für den eSTREAM-Wettbewerb angepasste Version der (in diesem Buch nicht behandelten) Stromchiffre HC-256. Das Verfahren ist für eine Implementierung in Software gedacht. Es nutzt einen 128-Bit-Schlüssel sowie einen 128-Bit-Startvektor, um pro Iteration 32 Keystream-Bits zu generieren.

Funktionsweise von HC-128

HC-128 arbeitet mit einem Status der Größe 4.096 Byte. Dies ist mit Abstand der größte Status unter den sieben Verfahren im eSTREAM-Portfolio, wodurch HC-128 für Umgebungen mit knappem Speicher ungeeignet ist. Der Status ist in die beiden Teile P und Q aufgeteilt, die aus jeweils 512 Variablen zu je 32 Bit bestehen. Diese Variablen werden als $P[0]$, $P[1]$, ..., $P[511]$ sowie $Q[0]$, $Q[1]$, ..., $Q[511]$ bezeichnet. Für die Fortschaltung und die Initialisierung des Status spielen die sechs Funktionen f_1, f_2, g_1, g_2, h_1 und h_2 eine Rolle. Die Funktionsweisen von f_1, f_2, g_1 und g_2 sind in Abbildung 16–5 dargestellt (*rot* steht hierbei für eine Rechtsrotation um die genannte Zahl; ist diese Zahl negativ, dann handelt es sich um eine Linksrotation). Für die Funktionen h_1 und h_2 gilt folgende Formel (x ist hierbei eine 32-Bit-Variable, die sich aus den vier Bytes x_0, x_1, x_2 und x_3 zusammensetzt):

$$h_1(x) = Q[x_0] + Q[256 + x_2]$$
$$h_2(x) = P[x_0] + P[256 + x_2]$$

Die Fortschaltung des Status (nach erfolgter Initialisierung) hängt davon ab, um die wievielte Iteration es sich handelt. Für die Iterationen 0 bis 511, 1.024 bis 1.536, 2.048 bis 2.559, 3.072 bis 3.584 usw. gilt die folgende Formel (i steht für die Nummer der Iteration modulo 512, die Subtraktionen erfolgen ebenfalls modulo 512):

$$P[i] = P[i] + g_1(P[i\text{-}3], P[i\text{-}10], P[i\text{-}511])$$

Als Keystream wird ausgegeben: $h_1(P[i\text{-}12]) \oplus P[i])$. Für alle anderen Iterationen gilt folgende Fortschaltung:

$$Q[i] = Q[i] + g_2(Q[i\text{-}3], Q[i\text{-}10], Q[i\text{-}511])$$

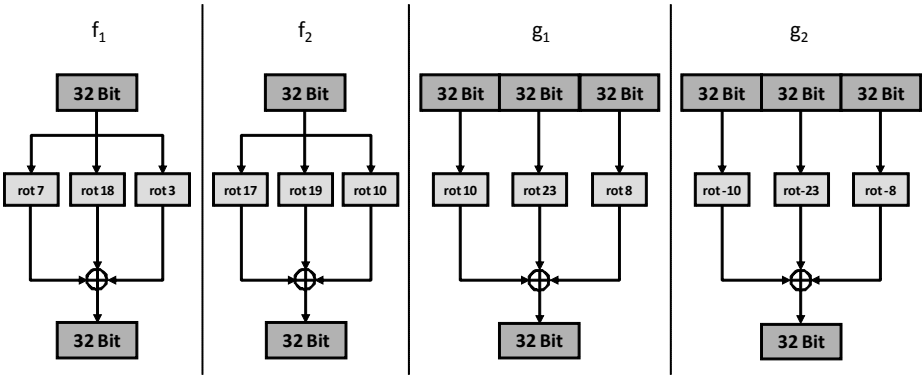


Abb. 16–5 *Für die Funktionsweise von HC-128 spielen die Funktionen f_1, f_2, g_1, und g_2 eine Rolle.*

Als Keystream wird ausgegeben: $h_2(Q[i\text{-}12]) \oplus Q[i]$. Die Initialisierung beginnt mit dem in Abbildung 16–6 beschriebenen Schema. Hierbei wird eine Variable W verwendet, die aus 1.280 Blöcken zu je 32 Bit besteht. Die ersten 16 Blöcke werden mit dem Schlüssel und dem Initialisierungsvektor (beide werden jeweils doppelt verwendet) aufgefüllt. Die restlichen Blöcke ergeben sich mithilfe der Funktion u, wobei gilt (i ist die Nummer des Blocks, also am Anfang 16, dann 17, dann 18 usw.):

$$u = W_{i\text{-}16} + f_1(W_{i\text{-}15}) + W_{i\text{-}7} + f_2(W_{i\text{-}2}) + i$$

Die Inhalte von W werden anschließend wie folgt in P und Q übertragen:

Für i von 0 bis 511: $P[i] = W_{i+256}$ und $Q[i] = W_{i+768}$

Anschließend folgen insgesamt 1.024 Iterationen ohne Keystream-Generierung mit leicht geänderter Fortschaltung. Die ersten 512 Iterationen haben folgenden Ablauf (i läuft von 0 bis 512, die Subtraktionen erfolgen modulo 512):

$$P[i] = P[i] + g_1(P[i]\text{-}3], P[i\text{-}10], P[i\text{-}511]) \oplus h_1(P[i\text{-}12])$$

Die verbleibenden 512 Leeriterationen haben folgenden Ablauf (i läuft wieder von 0 bis 512, die Subtraktionen erfolgen modulo 512):

$$Q[i] = Q[i] + g_2(Q[i]\text{-}3], Q[i\text{-}10], Q[i\text{-}511]) \oplus h_2(Q[i\text{-}12])$$

Damit ist die Initialisierung abgeschlossen, und die Generierung des Keystreams kann beginnen.

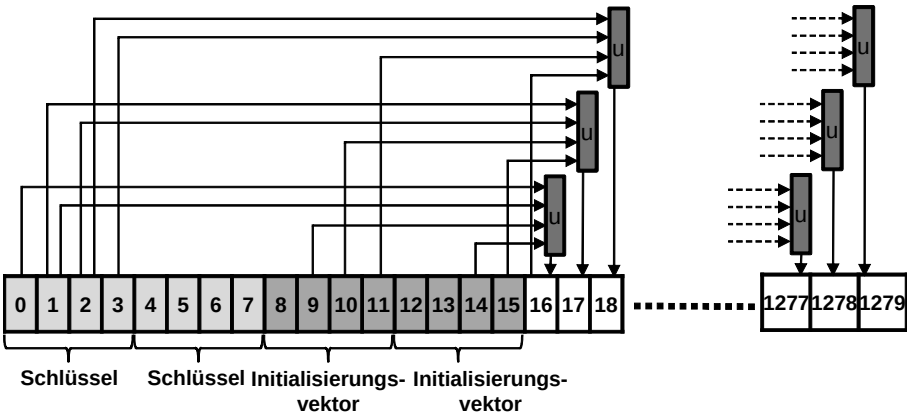


Abb. 16–6 *Die Initialisierung von HC-128 arbeitet mit einer 1.280-Bit-Variablen.*

Bewertung von HC-128

In *The eSTREAM Portfolio* wird HC-128 als das Verfahren bezeichnet, das die Anforderungen an eine Stromchiffre für Softwareimplementierungen am besten erfüllt [eSTREA]. In der Tat ist HC-128 nach aktuellem Stand der Forschung nicht nur sicher, sondern (nach erfolgter Initialisierung) auch schnell. Die hohe Performanz wird zwar durch einen recht umfangreichen Status der Länge 4.096 Byte erkauft, doch bei einer Umsetzung auf dem PC spielt dies keine wesentliche Rolle. Ein Pferdefuß ist allerdings die recht aufwendige Initialisierung des Status. HC-128 ist daher (noch mehr als andere Stromchiffren) für Anwendungen interessant, bei denen größere Datenmengen ohne Wechsel des Schlüssels oder des Initialisierungsvektors verschlüsselt werden. Dieser Nachteil sowie verschiedene Bedenken bezüglich der Sicherheit (die sich bisher nicht bestätigten) sorgten dafür, dass HC-128 bei der SASC-Abstimmung nur Platz 4 erreichte – mit deutlichem Abstand zu den drei Erstplatzierten.

16.6.2 Rabbit

Rabbit ist eine Stromchiffre, die von vier Kryptografen aus Kopenhagen entwickelt wurde [BoVeCZ]. Das Verfahren arbeitet mit einem Schlüssel der Länge 128 Bit sowie einem 64-Bit-Initialisierungsvektor. Es ist besonders für die Implementierung in Software geeignet. Nach einer Initialisierung gibt Rabbit pro Iteration 128 Bit Keystream aus.

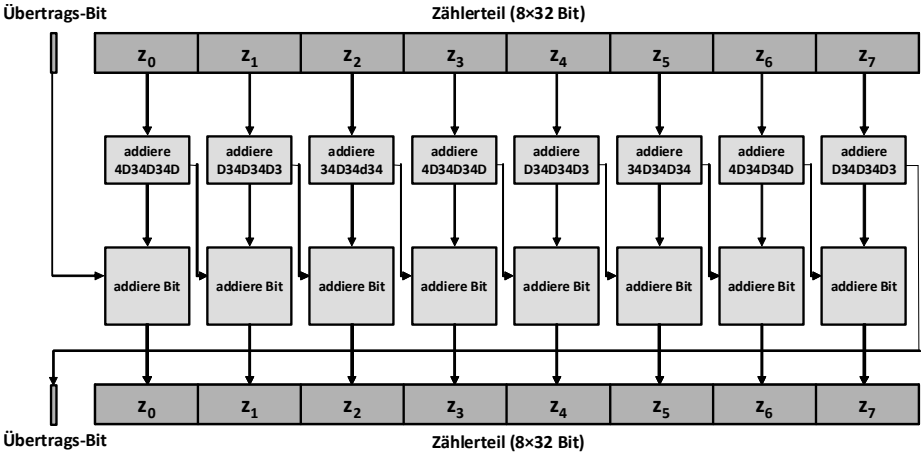


Abb. 16–7 *Der Zählerteil von Rabbit wird mit linearen Funktionen fortgeschaltet.*

Funktionsweise von Rabbit

Rabbit arbeitet mit einem Status der Länge 513 Bit. Dieser teilt sich in drei Teile auf:

- *Statusteil*: Dieser besteht aus acht 32-Bit-Variablen (s_0, s_1, ..., s_7).
- *Zählerteil*: Dieser besteht ebenfalls aus acht 32-Bit-Variablen (z_0, z_1, ..., z_7).
- *Übertrags-Bit*: Dies ist ein zusätzliches Bit.

Die Fortschaltfunktion von Rabbit bearbeitet alle 513 Bit, wobei der Statusteil in nichtlinearer Form und der Zählerteil in linearer Form fortgeschaltet wird. Die Fortschaltung des Zählerteils ist in Abbildung 16–7 dargestellt. Dieser Vorgang sieht vor, dass zu jeder der acht 32-Bit-Variablen des Zählerteils jeweils eine Konstante gezählt wird (alle Additionen erfolgen modulo 2^{32}). Diese Konstante hat für jede der acht 32-Bit-Variablen einen anderen Wert. Für z_0 lautet der Wert 4D34D34D (in hexadezimaler Schreibweise), für Zählervariable z_1 wird D34D34D3 verwendet. Die anderen Werte können Sie der Abbildung entnehmen.

Zusätzlich wird im Rahmen der Fortschaltung zu jeder Variablen des Zählerteils der Inhalt eines Bits (also die Zahl 0 oder 1) addiert. Welchen Wert dieses Bit annimmt, hängt bei den Variablen z_1 bis z_7 jeweils vom linken Nachbarn ab. Wenn bei den beiden Additionen von z_0 ein Ergebnis von größer 2^{32} entsteht (und daher eine Subtraktion von 2^{32} notwendig ist, da es sich um eine Modulo-Addition handelt), dann hat das Bit für z_1 den Wert 1, ansonsten 0. Wenn bei den beiden Additionen von z_1 ein Resultat größer 2^{32} entsteht, dann ist der Wert des Bits für z_3 1, ansonsten 0. Bei den verbleibenden Variablen verhält es sich analog. Für die Zählervariable z_0, die keinen linken Nachbarn hat, gilt eine andere Regelung. Hier ist der Wert 1, wenn die Additionen der Zählervariable z_7 in der vorhergehenden Iteration ein Ergebnis größer als 2^{32} ergeben haben. Um den Wert dieses Bits aus der vorherigen Iteration zu speichern, wird das Übertrags-Bit des

Status verwendet. Die Fortschaltung des Statusteils ist in Abbildung 16–8 dargestellt. Im Mittelpunkt des Ablaufs steht die Funktion g, in die auch der Zählerteil eingeht. Dabei wird vorausgesetzt, dass die Fortschaltung des Zählerteils schon erfolgt ist. Für $n=0, 1, ..., 7$ gilt (>> steht für eine bitweise Verschiebung nach rechts):

$$g := (s_n + z_n)^2 \oplus ((s_n + z_n)^2 >> 32) \bmod 2^{32}$$

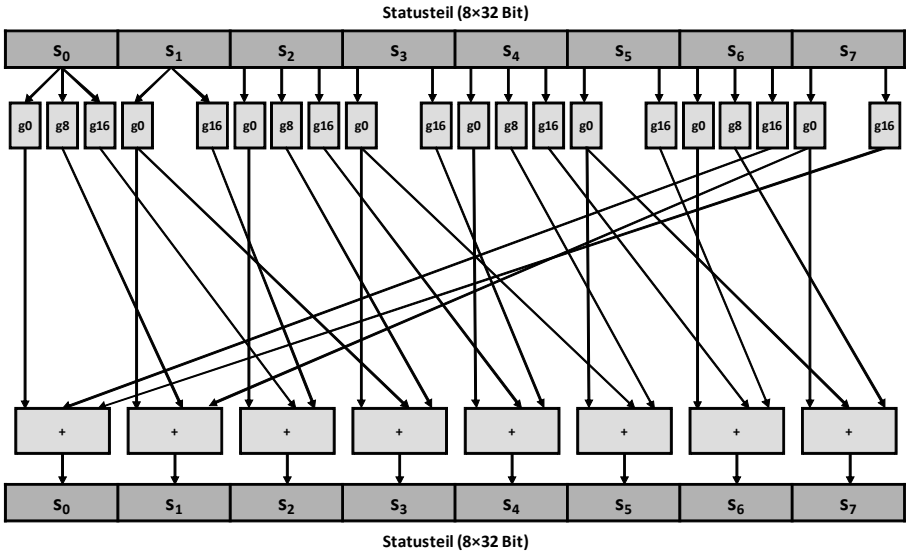


Abb. 16–8 *Die Fortschaltung des Statusteils von Rabbit erfolgt mit nichtlinearen Funktionen.*

Bei den beiden Quadrierungen in dieser Formel entsteht jeweils ein 64-Bit-Wert. Da jedoch am Ende der Formel ein »mod 2^{32}« steht, werden nur die letzten 32 Bit als Ergebnis verwendet. Der Zusatz $g0$, $g8$ bzw. $g16$ bedeutet, dass das Resultat der obigen Formel zusätzlich um 0, 8 bzw. 16 Bit nach links rotiert wird.

Nach der Fortschaltung werden aus dem Statusteil 128 Bit Keystream extrahiert. Wie dies funktioniert, ist in folgender Tabelle dargestellt. e_0, ..., e_7 sind hierbei 16-Bit-Wörter und bilden den Keystream. Mit $s_{0,0}$, $s_{1,0}$, ..., $s_{7,0}$ ist jeweils die linke Hälfte einer Statusvariablen gemeint, mit $s_{0,1}$, $s_{1,1}$, ..., $s_{7,1}$ entsprechend die jeweils rechte Hälfte.

e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7
$s_{0,0} \oplus s_{5,1}$	$s_{0,1} \oplus s_{3,0}$	$s_{2,0} \oplus s_{7,1}$	$s_{2,1} \oplus s_{5,0}$	$s_{4,0} \oplus s_{1,1}$	$s_{4,1} \oplus s_{7,0}$	$s_{6,0} \oplus s_{3,1}$	$s_{6,1} \oplus s_{1,0}$

Zur vollständigen Beschreibung des Verfahrens fehlt nun noch die Initialisierung des Status. Das Übertrags-Bit wird mit 0 initialisiert. Für den Status- und den Zählerteil gilt ein etwas komplexeres Vorgehen. Um dieses zu erklären, gehen wir

davon aus, dass der 128-Bit-Schlüssel in Form von acht 16-Bit-Wörtern gegeben ist, die wir als k_0, k_1, ..., k_7 bezeichnen. Die Initialisierung des Statusteils beginnt wie folgt ($k_1||k_0$, $k_6||k_5$ usw. steht für die Konkatenation der jeweiligen Variablen):

s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
$k_1\|\|k_0$	$k_6\|\|k_5$	$k_3\|\|k_2$	$k_0\|\|k_7$	$k_5\|\|k_4$	$k_2\|\|k_1$	$k_7\|\|k_6$	$k_4\|\|k_3$

Die Initialisierung des Zählerteils sieht wie folgt aus:

z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
$k_4\|\|k_5$	$k_1\|\|k_2$	$k_6\|\|k_7$	$k_3\|\|k_4$	$k_0\|\|k_1$	$k_5\|\|k_6$	$k_2\|\|k_3$	$k_7\|\|k_0$

Nach diesem ersten Initialisierungsschritt wird der Statusteil viermal fortgeschaltet, während der Zählerteil unverändert bleibt. Anschließend werden die Zählervariablen wie folgt verändert:

z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
$z_0 \oplus s_4$	$z_1 \oplus s_5$	$z_2 \oplus s_6$	$z_3 \oplus s_7$	$z_4 \oplus s_0$	$z_5 \oplus s_1$	$z_6 \oplus s_2$	$z_7 \oplus s_3$

Nun muss noch der 64-Bit-Initialisierungsvektor einfließen. Dieser wird in der Form i_0, i_1, ..., i_7 notiert und beeinflusst lediglich den Zählerteil:

z_0	z_1	z_2	z_3
$z_0 \oplus i_4\|\|i_5\|\|i_6\|\|i_7$	$z_1 \oplus i_0\|\|i_1\|\|i_4\|\|i_5$	$z_2 \oplus i_0\|\|i_1\|\|i_2\|\|i_3$	$z_3 \oplus \imath_2\|\|\imath_3\|\|\imath_6\|\|\imath_7$
z_4	z_5	z_6	z_7
$z_4 \oplus i_4\|\|i_5\|\|i_6\|\|i_7$	$z_5 \oplus i_0\|\|i_1\|\|i_4\|\|i_5$	$z_6 \oplus i_0\|\|i_1\|\|i_2\|\|i_3$	$z_7 \oplus \imath_2\|\|\imath_3\|\|\imath_6\|\|\imath_7$

Zum Schluss der Initialisierung wird der gesamte Status viermal fortgeschaltet. Anschließend kann der Status zum Produzieren von Keystream genutzt werden.

Bewertung von Rabbit

Rabbit stammt aus dem Jahr 2003 und gehört damit zu den ältesten eSTREAM-Kandidaten. Da trotz dieses vergleichsweise hohen Lebensalters bisher keine Schwachstellen bekannt geworden sind, zählt Rabbit zweifellos zur ersten Garde unter den Stromchiffren. Dies wird auch durch den geteilten ersten Platz bei der SASC-Abstimmung bekräftigt. Allerdings ist Rabbit das einzige Verfahren im eSTREAM-Portfolio, das patentiert ist. Zwar fallen nur für kommerzielle Anwendungen Patentgebühren an, doch schon allein diese Einschränkung hat sich in der Vergangenheit für mehrere Verfahren (etwa IDEA) als wenig förderlich erwiesen. Wer keine spezielle Vorliebe für Rabbit hat, sollte sich daher lieber nach etwas anderem umsehen.

16.6.3 Salsa20

Salsa20 ist eine Stromchiffre, die von dem US-Kryptologen Daniel Bernstein entwickelt wurde [Bernst]. Die Schlüssellänge beträgt 128 oder 256 Bit (wir wollen nur ersteren Fall betrachten). Pro Iteration generiert das Verfahren 512 Keystream-Bits (dies ist deutlich mehr als sonst bei Stromchiffren üblich). Der Initialisierungsvektor hat die Länge 64 Bit.

Funktionsweise von Salsa20

Der Aufbau von Salsa20 wirkt ungewohnt. Im Gegensatz zu den anderen in diesem Buch betrachteten Stromchiffren entspricht das Funktionsprinzip nicht dem OFB-Modus, sondern dem CTR-Modus (siehe Abschnitt 19.1.5). Das bedeutet: Der Status besteht nur aus einem Zähler (als n bezeichnet), und die Fortschaltfunktion sieht lediglich ein Hochzählen des Status vor. Die für die Sicherheit notwendige Komplexität des Verfahrens liegt dafür in der Keystream-Extraktion. Diese läuft in mehreren Runden ab. Wie der Name des Verfahrens andeutet, waren ursprünglich 20 Runden vorgesehen. Um die Performanz zu erhöhen, reichte der Erfinder Daniel Bernstein eine 12-Runden- und eine 8-Runden-Version nach, die als Salsa20/12 bzw. Salsa20/8 bezeichnet werden. In das eSTREAM-Portfolio ging Salsa20/12 ein.

Für die Keystream-Extraktion benötigt Salsa20 eine Variable der Länge 512 Bit. Diese ist in Form einer 4×4-Matrix gegeben, deren 16 Einträge jeweils 32 Bit umfassen. Diese Matrix wird als X bezeichnet und wie folgt notiert:

$$\begin{matrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{matrix}$$

Für die Keystream-Extraktion spielt zudem die in Abbildung 16–9 dargestellte Funktion (Viertelrundenfunktion) eine Rolle. In diese gehen vier 32-Bit-Variablen ein, und vier 32-Bit-Variablen werden ausgegeben. Die Viertelrundenfunktion lässt sich auf die Matrix X sowohl spaltenweise als auch zeilenweise anwenden. In ersterem Fall erfolgen vier Funktionsaufrufe, wobei (x_0, x_4, x_8, x_{12}), (x_1, x_5, x_9, x_{13}), (x_2, x_6, x_{10}, x_{14}) und (x_3, x_7, x_{11}, x_{15}) jeweils auf sich selbst abgebildet werden. Bei der zeilenweisen Anwendung werden entsprechend (x_0, ..., x_3), (x_4, ..., x_7), (x_8, ..., x_{11}) und (x_{12}, ..., x_{15}) durch insgesamt vier Funktionsaufrufe auf sich selbst abgebildet. Auf Basis der Viertelrundenfunktion wird eine weitere Funktion (als H bezeichnet) definiert, die die Matrix X auf sich selbst abbildet. H hat folgenden Ablauf:

1. Durchlaufe sechs Mal:
   - Wende Viertelrundenfunktion spaltenweise an
   - Wende Viertelrundenfunktion zeilenweise an
2. Addiere den ursprünglichen Wert der Matrix X

Anders ausgedrückt gilt: $X = X + (V_{zw}(V_{sw}(X)))^6$. V steht hierbei für die Viertelrundenfunktion und *zw* bzw. *sw* für zeilen- bzw. spaltenweise. Das spaltenweise oder zeilenweise Anwenden der Viertelrundenfunktion auf X bezeichnet eine Runde. Pro Runde wird also die Viertelrundenfunktion viermal aufgerufen, was deren Namen erklärt. Das sechsmalige Durchlaufen der oben genannten Schleife entspricht zwölf Runden und damit der Variante Salsa20/12. Für Salsa20/8 sind vier Durchläufe notwendig. In der ursprünglichen Version von Salsa20 gab es zehn Durchläufe und damit 20 Runden. Die Keystream-Extraktion läuft unabhängig von der Rundenzahl wie folgt ab (|| steht wie üblich für die Konkatenation):

$$keystream = H(c_0 \| k \| c_1 \| n \| c_2 \| k \| c_3)$$

c_0, c_2, c_2 und c_3 sind Konstanten, deren Wert Sie in der Originalarbeit nachlesen können. Jede Konstante hat eine Länge von 4 Byte. k ist der Schlüssel (16 Byte). n ist der Status (also der Zähler), der nach jeder Fortschaltung um eins hochzählt wird. Der Anfangswert von n entspricht dem Initialisierungsvektor. Die Länge von n beträgt 16 Byte. Wie Sie leicht nachrechnen, haben die eingegebenen Variablen zusammen die Länge 512 Bit (64 Byte) und lassen sich dadurch als Matrix darstellen und als Eingabe von H verwenden. Die Ausgabe ist erneut ein 512-Bit-Wert. Dieser wird als Keystream ausgegeben.

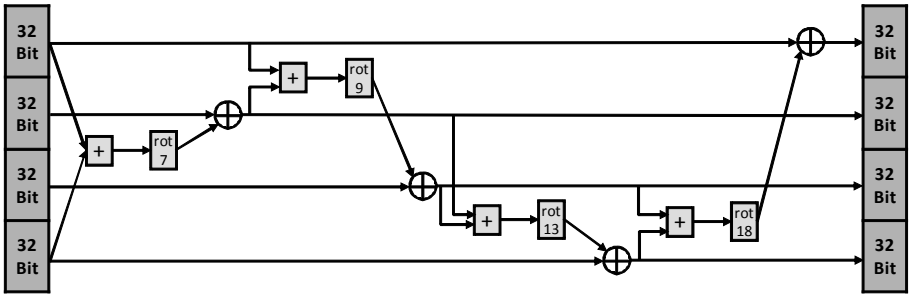


Abb. 16–9 *Die Viertelrundenfunktion ist der zentrale Bestandteil von Salsa20. Sie spielt bei der Keystream-Generierung eine Rolle.*

Bewertung von Salsa20

Salsa20 ist durch sein außergewöhnliches Design zweifellos eine interessante Stromchiffre. Das Verfahren hat alle Vorteile, die der in Abschnitt 19.1.5 beschriebene CTR-Modus bietet. Insbesondere lässt sich damit ein einzelner Nachrichtenblock entschlüsseln, ohne dass zuvor eine größere Zahl von Fortschaltun-

gen notwendig wäre. Sollten gegen Salsa20 weiterhin keine wirkungsvollen Angriffe entdeckt werden, dann kann man Salsa20 zu den besten Stromchiffren überhaupt zählen. Das sahen auch die SASC-Juroren so und setzten Salsa20 neben Rabbit auf Platz 1 unter den Softwareverfahren. Die Autoren von *The eSTREAM Portfolio* stimmen dieser Ansicht zu. Sie bezeichnen Salsa20 als »sehr erfolgreichen Vorschlag« [eSTREA].

16.6.4 Sosemanuk

Sosemanuk ist eine Stromchiffre, die von einem zwölfköpfigen französischen Kryptografenteam entwickelt wurde [BBCCGG]. Die Schlüssellänge beträgt zwischen 128 und 256 Bit, der Initialisierungsvektor hat eine Länge von 128 Bit. Sosemanuk erzeugt pro Iteration 32 Keystream-Bits, von denen jeweils vier auf einmal ausgegeben werden. Das Design von Sosemanuk ist eine Weiterentwicklung eines Verfahrens namens Snow und nimmt einige Anleihen beim symmetrischen Verschlüsselungsverfahren Serpent (siehe Abschnitt 9.2). So erklärt sich auch der Name: Sosemanuk ist ein unter kanadischen Indianern verbreitetes Spiel, das übersetzt »Schneeschlange« heißt – eine passende Bezeichnung für ein Verfahren, das mit Snow (Schnee) und Serpent (Schlange) verwandt ist.

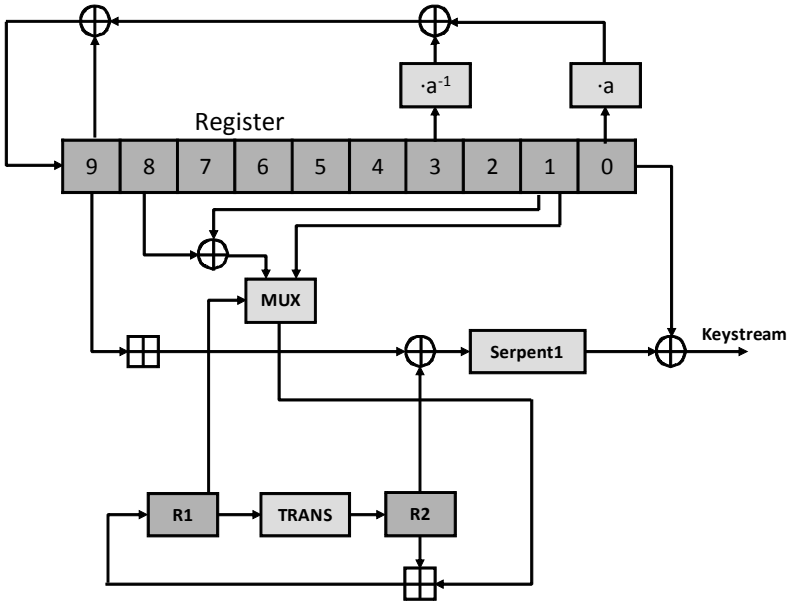


Abb. 16–10 *Die Stromchiffre Sosemanuk arbeitet mit einem Status, der aus insgesamt zwölf 32-Bit-Variablen besteht. Zehn davon bilden das Register, die zwei verbleibenden heißen R1 und R2.*

Funktionsweise von Sosemanuk

Sosemanuk verwendet einen Status der Länge 384 Bit. Davon sind zehnmal 32 Bit ähnlich wie ein LFSR angeordnet und werden als Register bezeichnet. Die verbleibenden zwei 32-Bit-Variablen werden *R1* und *R2* genannt. Die Funktionsweise von Sosemanuk ist in Abbildung 16–10 zu sehen. Die einzelnen Bestandteile haben folgende Bedeutung:

- Die Zahl *a* ist eine Konstante, deren Wert in der Spezifikation gegeben ist. Bei der Fortschaltung wird eine der zehn Registervariablen mit *a* und eine weitere mit dem Kehrwert von *a* multipliziert.
- *MUX* ist eine Funktion, in die zwei 32-Bit-Werte sowie ein weiteres Bit (es ist das niederwertigste Bit von *R1*) eingehen. Ist dieses einzelne Bit Null, dann wird der erste 32-Bit-Wert als Ergebnis ausgegeben, ansonsten der zweite.
- *TRANS* ist eine Funktion, die einen 32-Bit-Wert entgegennimmt, diesen mit der Konstante 54655307 (in Hexadezimalschreibweise) multipliziert (modulo 2^{32}) und anschließend eine Linksrotation um sieben Bit vornimmt.
- *Serpent1* nimmt 128 Bit (in vier 32-Bit-Blöcken) entgegen und gibt ein Ergebnis im gleichen Format aus. Die 128 eingegebenen Bits werden mit der S-Box S_2 des Verschlüsselungsverfahrens Serpent bearbeitet. Es werden 32 parallele S-Boxen benötigt, um die 128 Bit verarbeiten zu können (eine S-Box nimmt vier Bits entgegen). Vor der S-Box-Anwendung wird die Eingangspermutation von Serpent durchgeführt, danach die Ausgangspermutation.

Eine Besonderheit von Sosemanuk ist, dass stets vier Iterationen durchgeführt werden müssen, bevor anschließend vier Keystream-Blöcke (zu je 32 Bit) auf einmal ausgegeben werden. Dies liegt daran, dass die Funktion *Serpent1* vier 32-Bit-Blöcke als Input benötigt und deshalb jeweils drei Blöcke gepuffert werden müssen. Die Initialisierung von Sosemanuk ist in Abbildung 16–11 dargestellt. Auch hier haben die Entwickler des Verfahrens einige Funktionsblöcke von Serpent übernommen.

Bewertung von Sosemanuk

In der SASC-Bewertung belegte Sosemanuk bei den Softwareverfahren den dritten Platz. Damit lag das Verfahren deutlich hinter Rabbit und Salsa20, aber auch deutlich vor allen anderen. Wesentliche Schwachstellen sind bisher nicht bekannt.

16.6.5 Trivium

Trivium ist eine Stromchiffre, die von dem bekannten belgischen Kryptologen Bart Preneel zusammen mit seinem Landsmann Christophe De Cannière entwickelt wurde [PreCan]. Die beiden beschreiben das Design als einen »Vorgang zum Herausfinden, wie weit eine Stromchiffre vereinfacht werden kann, ohne die Sicherheit, die Geschwindigkeit oder die Flexibilität zu opfern«.

Funktionsweise von Trivium

Trivium arbeitet mit einer Statusvariablen der Länge 288 Bit. Die Bits sind von 1 bis 288 durchnummeriert. Die Schlüssellänge beträgt 80 Bit, zudem gibt es einen 80-Bit-Initialisierungsvektor. Die Ausgabe des Keystream erfolgt mit einem Bit pro Iteration. Die Fortschaltung folgt dem folgenden einfachen Ablauf (siehe Abbildung 16–12). Von den 288 Bits des Status gehen 15 in logische Operationen ein, aus denen die drei Werte t_1, t_2 und t_3 berechnet werden. t_3 wird im Rahmen der Fortschaltung an Position 1 des Registers geschrieben, t_1 an Position 94 und t_2 an Position 178. Zudem wandern die Inhalte der 288 Bits jeweils um eine Position nach rechts (dies ist in der Abbildung nicht dargestellt). Ausnahmen sind die Positionen 93, 177 und 288, die nicht verschoben werden. Für die Positionen 1, 94 und 178 gelten bei dieser Verschiebung die Werte vor dem Überschreiben mit t_1, t_2 und t_3.

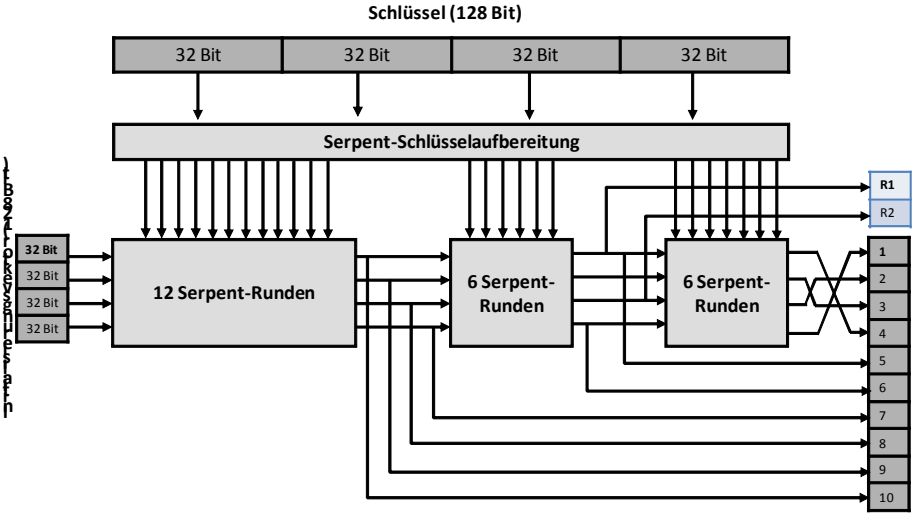


Abb. 16–11 *So werden die 12 Statusvariablen von Sosemanuk initialisiert.*

Das Keystream-Bit wird extrahiert, indem die Status-Bits Nummer 66, 93, 162, 177, 243 und 288 miteinander exklusiv-oder-verknüpft werden. Mit anderen Worten heißt dies: Wenn die Zahl der Einsen in diesen sechs Bits ungerade ist, dann wird eine Eins ausgegeben, ansonsten eine Null. Die Initialisierung läuft wie folgt ab:

1. Der Schlüssel (80 Bit) wird in die Bits 1 bis 80 des Status geschrieben.
2. Der Initialisierungsvektor (80 Bit) wird in die Bits 94 bis 173 des Status geschrieben.
3. Die Bits 286, 287 und 288 werden auf Eins gesetzt.

4. Alle verbleibenden Bits werden auf Null gesetzt.
5. Es werden 1.136 (also 4·288) Iterationen abgearbeitet, ohne dass Keystream ausgegeben wird.

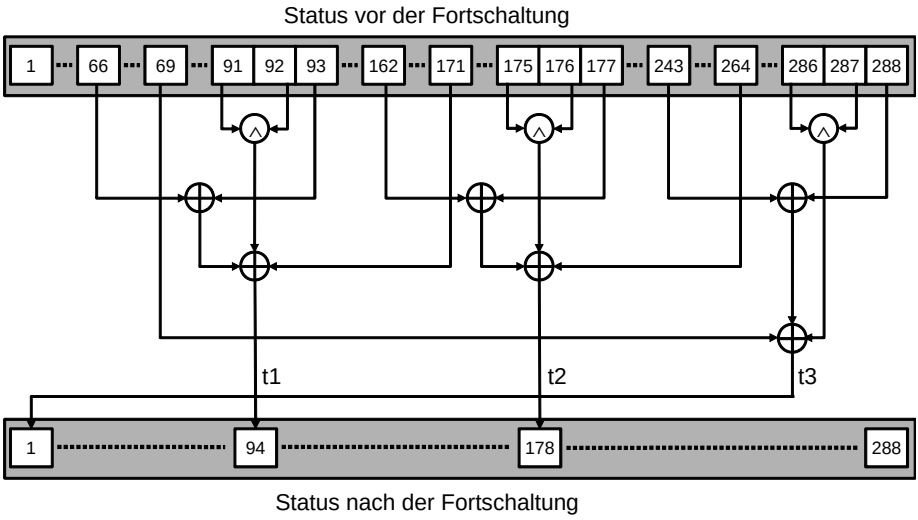


Abb. 16–12 *Trivium erhielt im Rahmen des eSTREAM-Wettbewerbs hervorragende Beurteilungen. Das für Hardwareumgebungen gedachte Verfahren ist einfach und gilt als sicher.*

Bewertung von Trivium

In der SASC-Rangliste belegt Trivium den ersten Platz unter den Stromchiffren für Hardwareimplementierungen. Mit einer beeindruckenden Bewertung von 4.35 (auf einer Skala von -5 bis +5) handelt es sich dabei eindeutig um das beste Verfahren des eSTREAM-Wettbewerbs.

16.6.6 Grain

Grain ist eine Stromchiffre, die von Martin Hell, Thomas Johansson und Willi Meier entwickelt wurde [HeJoMe]. Sie arbeitet mit einem Schlüssel der Länge 80 Bit und einem Initialisierungsvektor der Länge 64 Bit. Nach der Initialisierung gibt Grain ein Keystream-Bit pro Iteration aus.

Funktionsweise von Grain

Grain arbeitet mit einem LFSR und einem NFSR. Beide haben je 80 Bit und bilden zusammen den Status. Die Funktionsweise der Fortschaltung und der Keystream-Extraktion ist in Abbildung 16–13 zu sehen. Die Bits des LFSR werden im Folgenden mit l_0, l_1, ..., l_{79}, die Bits des NFSR mit n_0, n_1, n_2, ..., n_{79} bezeichnet.

Die in der Abbildung ersichtliche Funktion *f* ist die Feedback-Funktion des LFSR und hat folgende Definition:

$$f = l_{62} \oplus l_{51} \oplus l_{38} \oplus l_{23} \oplus l_{13} \oplus l_0$$

Die Funktion *g* ist die Feedback-Funktion des NFSR. Sie ist wie folgt definiert:

$$\begin{aligned} g = {} & n_{62} \oplus n_{60} \oplus n_{52} \oplus n_{45} \oplus n_{37} \oplus n_{33} \oplus n_{28} \oplus n_{21} \\ & \oplus n_{15} \oplus n_9 \oplus n_0 \oplus n_{63}n_{60} \oplus n_{37}n_{33} \oplus n_{15}n_9 \\ & \oplus n_{60}n_{52}n_{45} \oplus n_{33}n_{28}n_{21} \oplus n_{63}n_{45}n_{28}n_9 \\ & \oplus n_{60}n_{52}n_{37}n_{33} \oplus n_{63}n_{60}n_{21}n_{15} \\ & \oplus n_{63}n_{60}n_{52}n_{45}n_{37} \oplus n_{33}n_{28}n_{21}n_{15}n_9 \\ & \oplus n_{52}n_{45}n_{37}n_{33}n_{28}n_{21} \end{aligned}$$

Die Keystream-Extraktion erfolgt mithilfe der Funktion *h*, in die fünf Bits aus dem Status eingehen. *h* ist wie folgt definiert:

$$\begin{aligned} h = {} & n_{25} \oplus l_{63} \oplus n_3n_{64} \oplus n_{46}n_{64} \oplus n_{64}l_{63} \oplus n_3n_{25}n_{46} \oplus n_3n_{46}n_{64} \oplus n_3n_{46}l_{63} \\ & \oplus n_{25}n_{46}l_{63} \oplus n_{64}n_{64}l_{63} \end{aligned}$$

Die Ausgabe von *h* wird mit mehreren Bits des NFSR verknüpft. Das so entstehende Bit wird als Keystream ausgegeben. Die Initialisierung erfolgt, indem der Schlüssel (80 Bit) in das NFSR (80 Bit) und der Initialisierungsvektor (64 Bit) in das LFSR (80 Bit) geschrieben wird. Die verbleibenden 16 Bit im LFSR werden mit Einsen aufgefüllt. Anschließend werden zur Initialisierung 160 Fortschaltungen durchgeführt. Diese funktionieren wie die normalen Fortschaltungen, außer dass zum Ergebnis von *f* und *g* jeweils das Keystream-Bit (das während der Initialisierung noch nicht für den Keystream verwendet wird) der vorhergehenden Iteration per Exklusiv-Oder gezählt wird.

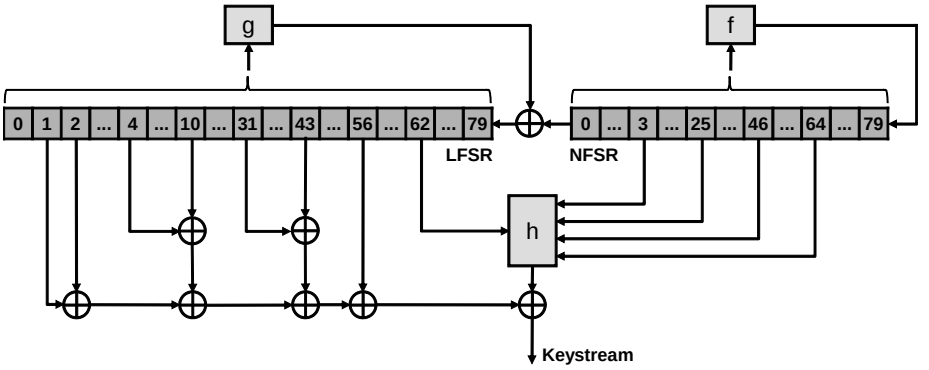


Abb. 16–13 *Grain ist eine Stromchiffre für Hardwareimplementierungen. Sie arbeitet mit einem LFSR (links) und einem NFSR (rechts).*

Bewertung von Grain

Grain überzeugt vor allem durch seine Einfachheit und den geringen Speicherbedarf bei gleichzeitig (nach aktuellem Stand der Forschung) hoher Sicherheit. Eine Besonderheit von Grain liegt darin, dass der Output an Keystream-Bits pro Iteration erhöht werden kann. In der SASC-Rangliste konnte Grain allerdings nicht mit dem ebenfalls einfachen und sicheren Trivium mithalten.

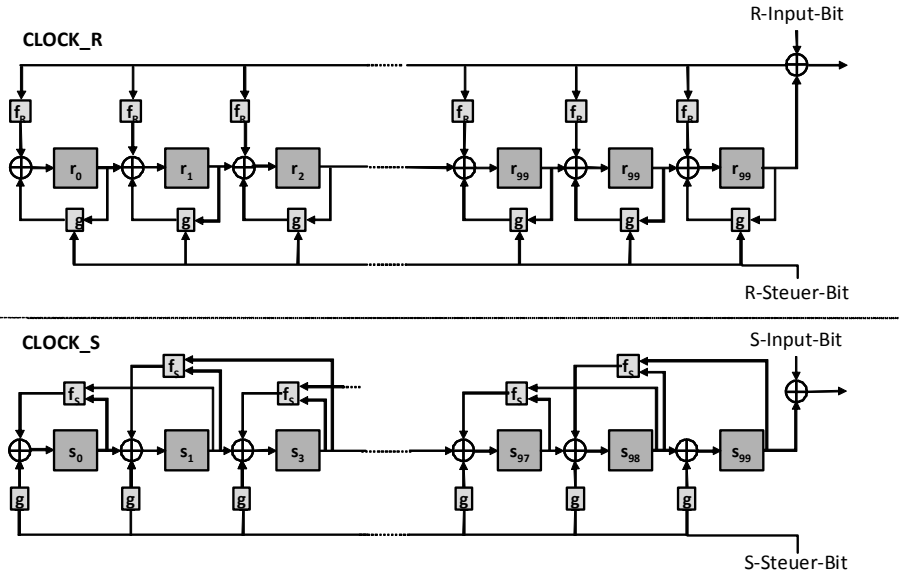


Abb. 16–14 *Die für Hardwareumsetzungen optimierte Stromchiffre MICKEY arbeitet mit den beiden Funktionen CLOCK_R und CLOCK_S.*

16.6.7 MICKEY

MICKEY ist eine Stromchiffre, die von Steve Babbage und Matthew Dodd entwickelt wurde [BabDod]. Der Name ist eine Abkürzung für *Mutual Irregular Clocking Keystream Generator*. Es gibt mehrere Versionen von MICKEY. Ins eSTREAM-Portfolio aufgenommen wurde die Version MICKEY v2, um die es im Folgenden geht. Deren Schlüssellänge beträgt 80 Bit, der Initialisierungsvektor hat eine Länge zwischen 0 und 80 Bit. Jede Fortschaltung erzeugt ein Keystream-Bit.

Funktionsweise von MICKEY

MICKEY arbeitet mit einem Status der Länge 200 Bit, der in zwei gleich große Teile *R* und *S* aufgeteilt ist. Die Fortschaltung von *R* ist linear und erfolgt mit der in Abbildung 16–14 oben gezeigten Funktion, die *CLOCK_R* genannt wird. In

diese gehen neben *R* zwei zusätzliche Bits ein, die als *R*-Input-Bit und *R*-Steuer-Bit bezeichnet werden. Jede Bit-Variable von *R* erhält bei einer Fortschaltung einen neuen Wert, der unter anderem von der links davon stehenden Bit-Variable abhängt. Die Funktion f_R, die hierbei eine Rolle spielt, reicht den Eingangswert unverändert durch, wenn die rechts darunter stehende Variable eine der folgenden Nummern hat: 0, 1, 3, 4, 5, 6, 9, 12, 13, 16, 19, 20, 21, 22, 25, 8, 37, 38, 41, 42, 45, 46, 50, 52, 54, 56, 58, 60, 61, 63, 64, 65, 66, 67, 71, 72, 79, 80, 81, 82, 87, 88, 89, 90, 91, 92, 94, 95, 96, 97. In allen anderen Fällen liefert f_R den Wert 0. Die Funktion g_R ist ebenfalls recht einfach. Sie liefert den Eingabewert als Ausgabe, wenn das *R*-Steuer-Bit 1 ist, ansonsten ist das Ergebnis 0.

Bei *S* erfolgt die Fortschaltung mithilfe der hochgradig nichtlinearen Funktion *CLOCK_S*. Auch in diese gehen zwei zusätzliche Bits ein (*S*-Input-Bit und *S*-Steuer-Bit). *CLOCK_S* läuft nach dem in Abbildung 16–14 unten ersichtlichen Schema ab. Auch hier erhält jede der 100 Bit-Variablen im Rahmen der Fortschaltung einen neuen Wert, der unter anderem von der Bit-Variable links davon abhängig ist. Die Funktion f_S hat folgenden Aufbau (*x* steht für den oberen, *y* für den unteren Wert):

$$f_S(x,y)=(x\oplus comp0)\cdot(y\oplus comp1)$$

comp0 und *comp1* sind Variablen, die bei jedem Aufruf einen anderen Wert annehmen. Die jeweiligen Werte sind in einer Tabelle festgelegt, die ich aus Platzgründen an dieser Stelle nicht aufführen will. Die Funktion g_S hat folgende Formel: $g(x)=x\cdot y$. Mithilfe von *CLOCK_R* und *CLOCK_S* wird nun die Funktion *CLOCK_KG* definiert, die neben *R* und *S* die beiden Ein-Bit-Variablen *Mixing-Bit* und *Input-Bit* entgegennimmt. *CLOCK_KG* hat folgenden Ablauf:

1. *R-Steuer-Bit*$=s_{34}\oplus r_{67}$
2. *S-Steuer-Bit*$=s_{67}\oplus r_{33}$
3. Falls *Mixing-Bit*$=1$ dann: *R-Input-Bit*$=$*Input-Bit*$\oplus s_{50}$
   sonst: *R-Input-Bit*$=$*Input-Bit*
4. *S-Input-Bit*$=$*Input-Bit*
5. CLOCK_R
6. CLOCK_S

Die Fortschaltung des Status erfolgt durch den Aufruf von *CLOCK_KG* mit den Argumenten *R*, *S*, *Mixing-Bit*$=0$ und *Input-Bit*$=0$. Vor jeder Fortschaltung wird das Bit $z=r_0\oplus s_0$ als nächstes Bit des Keystreams ausgegeben. Zur Initialisierung werden zunächst alle 200 Bits von *R* und *S* auf Null gesetzt. Anschließend erfolgen drei weitere Initialisierungsschritte:

1. Für jedes Bit des Initialisierungsvektors (von links nach rechts) wird *CLOCK_KG* einmal aufgerufen. Neben *R* und *S* gehen *Mixing-Bit*=1 und *Input-Bit* =Bit-Wert ein. Je nach Länge des Initialisierungsvektors erfolgen zwischen 0 und 80 Aufrufe.
2. Für jedes Bit des Schlüssels (von links nach rechts) wird *CLOCK_KG* einmal aufgerufen. Neben *R* und *S* gehen *Mixing-Bit*=1 und *Input-Bit*=Schlüssel-Bit in *CLOCK_KG* ein. Die Anzahl der Aufrufe beträgt 80.
3. Es folgen weitere 100 Aufrufe von *CLOCK_KG* mit *Mixing-Bit*=1 und *Input-Bit*=0.

Bewertung von MICKEY

In der SASC-Rangliste ist MICKEY mit einer mittelmäßigen Bewertung von 0,17 vertreten. Dies reichte immerhin für Platz 4. Da das drittplatzierte Verfahren wegen zwischenzeitlich festgestellter Schwachstellen inzwischen nicht mehr zum Portfolio gehört, kann man MICKEY auf Platz 3 einordnen. Die gegenüber Trivium und Grain schlechtere Bewertung kam vor allem durch die vergleichsweise geringe Performanz zustande. Diese ist jedoch auch auf einen hohen Sicherheitspuffer zurückzuführen, den die Entwickler eingebaut haben.

16.6.8 Erkenntnisse aus dem eSTREAM-Wettbewerb

eSTREAM hat die Entwicklung von Stromchiffren zweifellos vorangebracht. Nachdem zuvor keine einzige Stromchiffre den NESSIE-Wettbewerb überstanden hatte, gibt es nun immerhin ein Portfolio von sieben akzeptablen Verfahren. eSTREAM hat jedoch auch gezeigt, dass der Stand der Forschung zum Thema Stromchiffren noch längst nicht so weit fortgeschritten ist wie bei den Blockchiffren. Letztere hatten bereits vor zehn Jahren einen so hohen Entwicklungsstand erreicht, dass die Juroren des 1997 gestarteten AES-Wettbewerbs aus einer ganzen Reihe von hervorragenden Verfahren wählen konnten. Der eSTREAM-Wettbewerb, der 2004 begann, lieferte im Vergleich dazu ein eher enttäuschendes Bild. Zwar gab es mehr als doppelt so viele Einreichungen wie beim AES, doch am Ende schaffte es mit Trivium gerade einmal ein Verfahren, Bestnoten zu erhalten. Die sechs weiteren Verfahren des Portfolios fallen im Vergleich dazu schon deutlich ab. Alle anderen Wettbewerbsteilnehmer erwiesen sich sogar als unbrauchbar. Die Tatsache, dass von vier eSTREAM-Kategorien zwei ohne Sieger blieben, ist auch nicht gerade erfreulich. Es gibt also noch einiges zu tun im Bereich der Stromchiffren, und ich bin sicher, dass in den nächsten Jahren noch so manches Verfahren entstehen wird.

16.7 Welche Stromchiffre ist die beste?

Bevor ich auf die Beantwortung dieser Frage eingehe, möchte ich im Schnelldurchlauf noch ein paar weitere Stromchiffren vorstellen.

16.7.1 Weitere Stromchiffren

Die folgende Liste nennt ein paar Stromchiffren, auf die ich bisher nicht eingegangen bin und die auch nicht am eSTREAM-Wettbewerb teilgenommen haben:

- CSA: Dieses Verfahren ist eine Besonderheit, da es sowohl eine Block- als auch eine Stromchiffre enthält, die miteinander kombiniert werden. CSA (Common-Scrambling-Algorithmus) ist ein Teil des DVB-Standards für digitales Fernsehen und wurde für das Verschlüsseln von Fernsehbildern geschaffen. Das bereits 1994 entstandene Verfahren war zunächst geheim und gelangte erst 2002 durch die Analyse einer Softwareimplementierung an die Öffentlichkeit. Es zeigte sich sehr schnell, dass CSA nicht die Sicherheit bietet, die man von einem modernen Krypto-Verfahren gewohnt ist [CSA]. Ein praktisch verwertbarer Angriff ist jedoch bisher nicht bekannt.
- **MUGI**: MUGI ist eine Stromchiffre, die 2002 veröffentlicht wurde und bisher als sicher gilt [WaFuTP].
- **MULTI-S01**: Diese Stromchiffre wurde im Jahr 2000 von der japanischen Firma Hitachi veröffentlicht. Der Name steht für »MULTImedia encryption algorithm-S01«. Es sind keine Schwachstellen bekannt [MULTI].
- **SEAL**: SEAL (Software-optimized Encryption ALgorithm) wurde bereits 1994 veröffentlicht und zählt damit zu den ältesten Verfahren dieser Art (für eine Beschreibung siehe beispielsweise [Schn96]). Die Entwickler waren Phillip Rogaway und Don Coppersmith von IBM. Coppersmith ist einer der Väter des DES.
- **Pike**: Die Stromchiffre Pike stammt ebenfalls aus dem Jahr 1994 [Ande94]. Ihr Erfinder ist der in diesem Buch mehrfach erwähnte Ross Anderson. Das Verfahren gilt nach heutigem Stand der Forschung als sicher.
- **T-310**: Dies ist eine Stromchiffre, die in den achtziger Jahren in der DDR entstand und dort innerhalb einer weit verbreiteten Verschlüsselungsmaschine gleichen Namens zum Einsatz kam. Die Schlüssellänge beträgt 230 Bit. Auf Basis von Unterlagen aus der Birthler-Behörde, die mir Bernd Lippmann zur Verfügung stellte, konnte ich dieses historisch interessante Verfahren 2006 öffentlich bekannt machen [Schm06]. Leider gibt es bisher keine veröffentlichten Kryptoanalyse-Resultate. Wer diesbezüglich aktiv werden möchte, kann mich gerne kontaktieren.

Es wäre kein Problem, an dieser Stelle noch einige weitere Stromchiffren aufzuzählen. Aus Platzgründen will ich jedoch davon absehen.

16.7.2 Welche Stromchiffren sind empfehlenswert?

Bereits im Abschnitt über den eSTREAM-Wettbewerb dürfte Ihnen klar geworden sein, dass Alice und Bob im Bereich der Stromchiffren weit weniger Auswahl haben als bei den Blockchiffren. Für Hochsicherheitsanwendungen gibt es keine einzige Stromchiffre, die man uneingeschränkt empfehlen kann. Für Softwareimplementierungen ist sicherlich RC4 ein Kandidat – allerdings haben einige Kryptoanalyse-Resultate am Image dieses Verfahrens gekratzt. Die im eSTREAM-Wettbewerb in der Sparte Software erfolgreichen HC-128, Sosemanuk, Salsa20 und Rabbit sind immerhin vielversprechende Alternativen. Ansonsten bleiben für eine Softwareimplementierung noch einige Exoten wie SEAL oder Pike, die allerdings nie so sehr im Blickpunkt standen, dass Kryptografen sie ausführlich genug untersuchten.

Im Bereich der hardwarerelevanten Stromchiffren sieht die Lage ähnlich bescheiden aus. Hier gibt es mit Trivium ein Verfahren, das im eSTREAM-Wettbewerb als einziges Bestnoten erhielt. Die Alternativen sind jedoch dünn gesät. E_0, A5 und Crypto1 entsprechen nicht den Erwartungen, die man heute an ein symmetrisches Verfahren stellt. Die Entwicklung von Stromchiffren hat also noch längst nicht das Niveau erreicht, das Blockchiffren bereits vor einem Jahrzehnt hatten. Die Auswahl an überzeugenden Verfahren ist daher gering. Es gibt also noch mehr als genug Arbeit für Kryptografen.

Teil 3

Implementierung von Kryptografie

17 Real-World-Attacken

Wenn Mallory versucht, eine Verschlüsselung von Alice und Bob zu knacken, dann muss er sich in der Regel nicht alleine auf mathematische und statistische Werkzeuge verlassen. Auch über einen geklauten PC oder einen geschickt programmierten Computervirus kann Mallory an den Schlüssel kommen. Solche Angriffe, die nicht gegen ein Verfahren an sich, sondern gegen dessen Einsatz in der Praxis gerichtet sind, werden **Real-World-Attacken** genannt.

17.1 Seitenkanalangriffe

Im Jahr 1995 überraschte der US-Kryptograf Paul Kocher die Fachwelt. Mit einer neuartigen Methode gelang es ihm, einer RSA-Implementierung innerhalb einiger Stunden den privaten Schlüssel zu entlocken. Mit den bekannten mathematischen Verfahren (Faktorisierung) hätte ein solches Unterfangen bis zum Ende des Universums keinen Erfolg gehabt. Kocher hatte jedoch einen Ansatz gefunden, der

über die reine Mathematik hinausging: Er maß die Zeit, die das Modul zum RSA-Entschlüsseln benötigte [Koch95]. In einer Chosen-Ciphertext-Attacke lieferten ihm schon einige Tausend Entschlüsselungsvorgänge genug Informationen, um den Schlüssel mittels Zeitvergleich rekonstruieren zu können.

Natürlich war mit diesem Angriff, den Kocher **Zeitangriff** (Timing Attack) nannte, RSA nicht gebrochen. Der Angriff traf schließlich nur die Implementierung, nicht aber das Verfahren an sich. In der Folgezeit entwickelten auch andere Kryptografen Angriffsmethoden, die die Ausführungszeit oder andere Messwerte zu Hilfe nahmen, um eine Krypto-Implementierung auszuhebeln. Ein solcher Angriff, der kryptografisch scheinbar nicht relevante Zusatzinformationen verwendet, wird **Seitenkanalangriff** genannt. Mit dem Aufkommen von Seitenkanalangriffen standen Chiffren-Designer und die Hersteller von Krypto-Lösungen vor völlig neuen Herausforderungen.

17.1.1 Zeitangriffe

Für die von Paul Kocher erfundenen Zeitangriffe benötigt Angreifer Mallory nur eine vergleichsweise simple Hardwareausstattung. Dennoch sind Zeitangriffe erstaunlich leistungsfähig, sofern eine Implementierung nicht speziell dagegen geschützt ist. Besonders gut funktionieren sie gegen asymmetrische Verfahren, da diese vergleichsweise komplexe Rechenoperationen erfordern, was sich in größeren Rechenzeit-Differenzen niederschlägt. Bei symmetrischen Verfahren hat es Mallory dagegen deutlich schwerer. Der DES verwendet beispielsweise nur Exklusiv-oder-Verknüpfungen, Substitutionen und Permutationen – diese Operationen haben eine konstante Rechenzeit. Auch symmetrische Verfahren, die komplexere arithmetische Operationen vorsehen (etwas der AES), sind meist unkritisch, da die jeweiligen Implementierungen in der Regel mit vorberechneten Ersetzungstabellen arbeiten.

Beispiele für Zeitangriffe

Die erwähnte Veröffentlichung von Paul Kocher aus dem Jahr 1996 stellte Zeitangriffe nicht nur erstmals vor, sondern behandelte gleich auch das naheliegendste Beispiel für einen solchen. Ziel des Angriffs ist die Exponentialfunktion in einem endlichen Körper. Wenn eine Krypto-Implementierung $a^b \pmod{n}$ für irgendwelche größeren Zahlen a, b und n berechnet, dann geschieht dies fast immer mit dem sogenannten Square-and-Multiply-Algorithmus. Dieser sieht für jedes Bit von b eine Aktion vor. Hat das gerade bearbeitete Bit den Wert Eins, dann ist diese Aktion eine Multiplikation, ansonsten eine Quadrierung. Da eine Multiplikation in diesem Zusammenhang mehr Zeit benötigt als eine Quadrierung, kann Mallory (er kennt a, n und die Länge von b) aus einer Zeitmessung Rückschlüsse auf die Zahl der Einsen in b schließen. Variiert Mallory die Zahl a, dann reichen seine Informationen irgendwann aus, um b zu berechnen.

Die Exponentialfunktion in einem endlichen Körper spielt bei den Verfahren RSA, DSA, ElGamal und Diffie-Hellman eine zentrale Rolle. Genau diese sind anfällig gegenüber einem Zeitangriff auf den Square-and-Multiply-Algorithmus. Schon mit einigen Tausend Zeitmessungen lassen sich entsprechende Implementierungen mit 1.024-Bit-Schlüsseln knacken. Auf den ersten Blick scheint auch das symmetrische Verschlüsselungsverfahren SAFER von einem solchen Angriff bedroht, da dieses die Exponentialfunktion nutzt (mit a=45 und n=257). Allerdings verwenden fast alle Implementierungen eine Ersetzungstabelle.

Dennoch gibt es auch symmetrische Verschlüsselungsverfahren, die als anfällig gegenüber einem Zeitangriff gelten. Das bekannteste Beispiel ist IDEA, das in den neunziger Jahren als wichtigste DES-Alternative galt. IDEA verwendet Multiplikationen modulo $2^{16}+1$. Deren Ausführungsdauer ist stark von den beiden Eingabewerten abhängig. 1998 veröffentlichten Kelsey, Schneier, Wagner und Hall die Grundzüge eines Zeitangriffs, der dies ausnutzte [KeScWH]. Einige weitere Zeitangriffe beziehen sich auf sehr spezielle Implementierungen. Im Jahr 2003 gelang es beispielsweise Dan Boneh und David Brumley, einen Webserver, der mit der Open-Source-Lösung OpenSSL arbeitete, anzugreifen. Ihr Zeitangriff ermittelte den privaten RSA-Schlüssel innerhalb einiger Stunden [BonBru].

Maßnahmen gegen Zeitangriffe

Zum Glück ist es nicht allzu schwierig, Zeitangriffe zu verhindern. Der Hersteller einer Krypto-Implementierung muss lediglich künstliche Verzögerungen in den Ver- bzw. Entschlüsselungsprozess einbauen. Beispielsweise kann er einen solchen Prozess so gestalten, dass dieser unabhängig vom Eingabewert immer gleich lange dauert. Oder es wird an einer Stelle des Verfahrens eine Zufallszahl eingerechnet, die an anderer Stelle wieder herausgerechnet wird. Dies bezeichnet man auch als **Blinding**. Beim Design symmetrischer Chiffren gibt es eine weitere Möglichkeit: Man kann auf Bestandteile verzichten, die Unterschiede in der Rechenzeit aufweisen.

17.1.2 Stromangriffe

Eine weitere Form des Seitenkanalangriffs ist der **Stromangriff**. Mallory füttert hierbei ein kryptografisches Modul mit Nachrichten und misst den Stromverbrauch, der beim Verschlüsseln (bzw. Entschlüsseln) entsteht. Aus den Schwankungen der Stromstärke, die er an einem Oszilloskop ablesen kann, zieht Mallory Rückschlüsse auf den Schlüssel. Dass ein Stromangriff funktioniert, liegt daran, dass die verschiedenen Bauteile eines Hardwaremoduls einen unterschiedlichen Stromverbrauch haben und dieser obendrein davon abhängt, ob eine bestimmte Schaltung geschlossen (eins) oder offen (null) ist.

Stromangriffe wurden 1998 von Paul Kocher der Öffentlichkeit vorgestellt [JaJuKo]. Sie sind etwas aufwendiger in der Durchführung als Zeitangriffe, gelten

jedoch als wirkungsvoller. Man kann Stromangriffe sogar als Verallgemeinerung der Zeitangriffe betrachten, da auf einem Oszilloskop neben dem Stromverbrauch stets auch die Verarbeitungszeit abzulesen ist. Stromangriffe sind jedoch nur bei Hardwareimplementierungen sinnvoll möglich, da sich bei einer Software die Krypto-Operationen meist mit anderen Aktivitäten des Betriebssystems überlagern. Vor allem Smartcards sind gegenüber Stromangriffen anfällig, da diese auf eine externe Stromversorgung angewiesen sind, die Mallory gut messen kann.

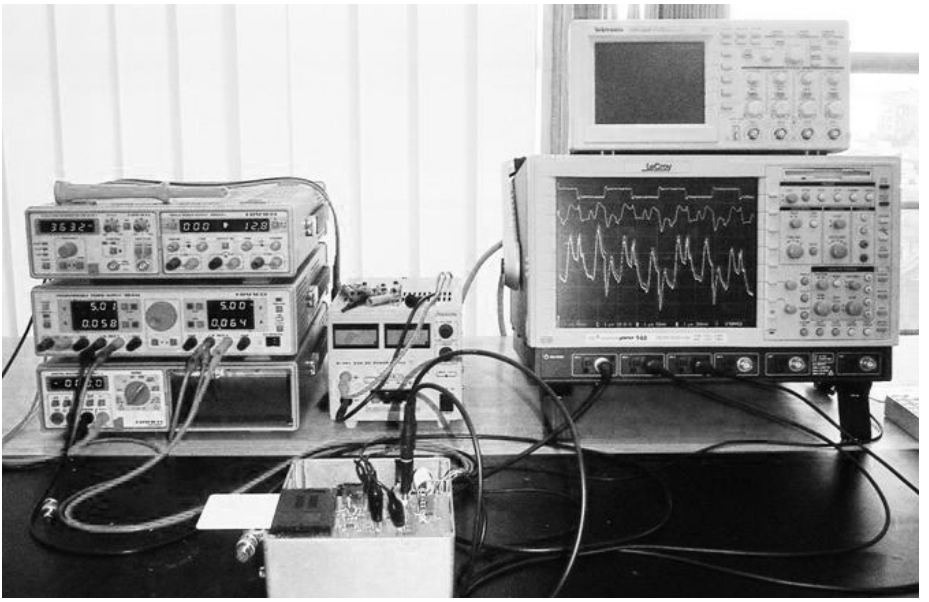


Abb. 17–1 *Für einen Stromangriff benötigt Angreifer Mallory eine geeignete Ausrüstung. Dazu gehört insbesondere ein Speicheroszilloskop. (Quelle: cryptovision)*

Simpler Stromangriff

Der einfachste Stromangriff ist der **simple Stromangriff**. Dieser nutzt lediglich die Messwerte aus einzelnen Ver- oder Entschlüsselungsvorgängen. Mallorys Ziel ist es zunächst, einzelne Bestandteile des Verfahrens bestimmten Stellen der Stromverbrauchskurve zuzuordnen. Anschließend versucht er, Rückschlüsse auf den Schlüssel zu ziehen. Abbildung 17–2 zeigt ein einfaches Beispiel. Hier wurden 32 Bytes innerhalb einer Smartcard aus dem EEPROM in das RAM geladen (derartiges passiert oft mit einem Schlüssel). Die ersten 16 Bytes sind mit Zufallswerten gefüllt, während die verbleibenden 16 Bytes jeweils denselben Wert enthalten. Der Unterschied ist auf dem Oszilloskop-Bild deutlich zu erkennen.

Ein naheliegendes Opfer des simplen Stromangriffs ist erneut der Square-and-Multiply-Algorithmus. Bei diesem lassen sich Multiplikationen und Quadrierungen am Stromverbrauch oft gut unterscheiden. Aber auch symmetrische Ver-

schlüsselungsverfahren wie der DES und der AES sind teilweise gegenüber simplen Stromangriffen anfällig, da ihre Bestandteile (u.a. Permutationen und Substitutionen) deutliche Spuren auf dem Oszilloskop hinterlassen.

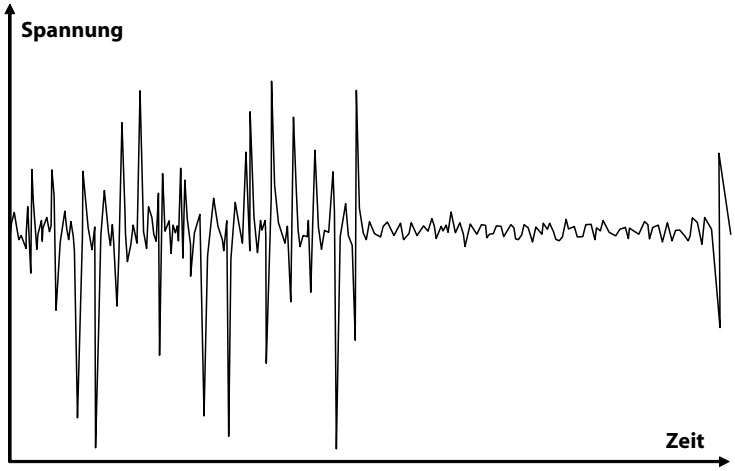


Abb. 17–2 *Der Unterschied zwischen den ersten 16 Bytes (Zufallswerte) und den restlichen 16 Bytes (jeweils mit demselben Wert) ist im Oszilloskop-Bild gut zu erkennen.*

Differenzieller Stromangriff

Beim **differenziellen Stromangriff** führt Mallory nicht nur eine, sondern mehrere Verschlüsselungen oder Entschlüsselungen durch. Idealerweise stehen ihm dabei mehrere Krypto-Module mit unterschiedlichen Schlüsseln zur Verfügung. Anschließend nutzt Mallory die Mittel der Statistik, um die zahlreichen Messkurven miteinander zu verrechnen. Das Ergebnis ist oft eine Messkurve, die besonders scharfe Konturen hat und dadurch Rückschlüsse auf den Arbeitsablauf zulässt. Daraus versucht Mallory, den Schlüssel zu ermitteln. Der differenzielle Stromangriff ist schwieriger, aber auch wirkungsvoller als der simple.

Bereits 1998 veröffentlichte Paul Kocher zusammen mit zwei Kollegen einen differenziellen Stromangriff auf den DES [JaJuKo]. Seinen Ausführungen zufolge gelang es ihm, mit etwa 100.000 Verschlüsselungsoperationen einen DES-Schlüssel zu rekonstruieren. Eine andere Gruppe von Kryptografen zeigte 2003, dass auch RC4 per differenziellem Stromangriff zu knacken ist. Sie ermittelten den Schlüssel mit einigen Tausend Versuchen [AACRR].

Maßnahmen gegen Stromangriffe

Auch gegen Stromangriffe gibt es wirksame Gegenmaßnahmen. So kann der Hersteller eines Krypto-Moduls den Stromverbrauch verschleiern, indem er Dummy-Operationen in einen Ver- bzw. Entschlüsselungsvorgang einbaut. Diese können

parallel zum Nutzbetrieb laufen und verzögern die Laufzeit daher nicht notwendigerweise. Darüber hinaus kann ein Krypto-Modul ein künstliches Rauschen erzeugen, das den eigentlichen Stromverbrauch überlagert.

17.1.3 Fehlerangriffe

Im September 1996 legten drei Mitarbeiter der US-Firma Bellcore den Grundstein für eine weitere Klasse von Seitenkanalangriffen, die als **Fehlerangriffe** bezeichnet werden [BoDeLi]. Bei einem Fehlerangriff beschädigt Angreifer Mallory eine Krypto-Implementierung oder bedient sie falsch, um aus den anschließend auftretenden Fehlern auf den Schlüssel zu schließen. Die drei genannten Bellcore-Forscher beschädigten Smartcards mit Hitze oder mechanisch. Durch einen Vergleich zwischen dem korrekten und dem falschen Resultat konnten sie teilweise den Schlüssel rekonstruieren. Adi Shamir und Eli Biham – die Erfinder der differenziellen Kryptoanalyse – übertrugen die Idee des Bellcore-Angriffs auf symmetrische Verfahren und führten den **differenziellen Fehlerangriff** ein [BiSh97] – so nennt man Fehlerangriffe, bei denen der Angreifer die Ergebnisse mehrerer Fehlerinduzierungen miteinander vergleicht. Ross Anderson und Markus Kuhn schlugen als zusätzliche Variante einen abrupten Taktwechsel vor, der zu aufschlussreichen Fehlern führen kann [AndKuh].

Auch Fehlerangriffe sind vor allem für Smartcards geeignet, da Mallory an diese am einfachsten herankommt. In der Tat haben Angriffe dieser Art in der Smartcard-Welt für einigen Wirbel gesorgt. In der Praxis spielen sie bisher jedoch kaum eine Rolle. Es ist offensichtlich recht schwierig, Hardwaremodule so zu beschädigen, dass ein gezielter Angriff möglich ist.

17.1.4 Weitere Seitenkanalangriffe

Ein weiterer Seitenkanal, der sich für entsprechende Angriffe nutzen lässt, ist die elektromagnetische Abstrahlung. Dementsprechend spricht man von **elektromagnetischen Angriffen**. Auch hier kann man wiederum zwischen dem simplen elektromagnetischen Angriff und dem differenziellen elektromagnetischen Angriff unterscheiden. Darüber hinaus gibt der innere Zustand eines Prozessors Informationen über den Schlüssel preis [BoDeLi]. Obwohl Seitenkanalangriffe vor allem gegenüber Hardwareimplementierungen eine wirksame Waffe sein können, sollte man sie nicht überschätzen. Dies belegen folgende Argumente:

- Wir gehen an dieser Stelle davon aus, dass Mallory direkten Zugriff auf das betreffende Hardwaremodul hat und dieses in Gang setzen kann (er wird also nicht von einer PIN aufgehalten). Diese Extremsituation ist in der Praxis häufig nicht gegeben.

- Die beschriebenen Angriffe können durch geeignete Konstruktionsmaßnahmen weitgehend verhindert werden. Man muss diese Seitenkanalangriffe daher als Teil eines Wettlaufs zwischen Hackern und Herstellern sehen.
- Viele der beschriebenen Angriffe haben eher theoretischen Charakter.

Unabhängig davon zeigen Seitenkanalangriffe (mehr dazu gibt es in [DriZeg]) eindrucksvoll, dass Sicherheit in der Kryptografie deutlich mehr bedeutet als sichere Krypto-Verfahren – auch auf eine sichere Umsetzung kommt es an.

17.2 Malware-Angriffe

Mallorys Arbeit als Spion und Codeknacker gestaltet sich deutlich einfacher, wenn er Zugang zu Alices Arbeitsumgebung hat. Kann sich Mallory beispielsweise auf Alices PC umsehen und auf dort gespeicherte Daten zugreifen, dann ist dies für ihn oft schon die halbe Miete. Zum Glück kann Alice Mallory in der Praxis meist davon abhalten, auf ihren Rechner zuzugreifen. Allerdings gibt es für Mallory Möglichkeiten, sich diesen Zugang auf indirekte Weise zu verschaffen, und zwar mithilfe eines Virus, eines Computerwurms oder eines Trojaners. Eine solche schadhafte Software (Malware) kann beispielsweise Alices geheimen Schlüssel auslesen und zu Mallory schicken. Gleiches kann die Malware mit dem Klartext, Alices Passwort oder sonstigen vertraulichen Informationen tun. Nutzt Mallory eine solche Vorgehensweise, dann bezeichnet man dies als **Malware-Angriff**.

Natürlich können Viren, Würmer und Trojaner weit mehr Schaden anrichten, als nur eine Krypto-Implementierung auszuhebeln. Malware ist daher ein eigenständiges Thema der IT-Sicherheit. Die Auswirkungen von Malware-Aktivitäten auf den Kryptografie-Einsatz können jedoch so entscheidend sein, dass sich ein Krypto-Experte auch mit bösartiger Software beschäftigen sollte. Bekannt ist, dass Malware vor allem für Windows-Betriebssysteme ein Problem ist. Durch die weite Verbreitung und die vergleichsweise offene Architektur ist Windows ein lohnendes Ziel für alle Viren- und Trojaner-Entwickler. Darüber hinaus gibt es boshafte Software in deutlich geringerem Umfang auch für andere Endanwender-Betriebssysteme wie Linux oder das Mac OS. Selbst Smartphone-Viren sind im Umlauf. Wenig anfällig gegenüber Malware sind dagegen Hardwaremodule und eingebettete Systeme, da diese nur sehr eingeschränkt mit der Außenwelt kommunizieren.

Viren, Würmer und Trojaner, die sich speziell gegen kryptografische Software richten, sind bisher noch recht selten. Dafür gelangten in den letzten Jahren unter anderem einige Trojaner in Umlauf, die Passwörter aus den Dateien des Anwenders ausspähten. Neben einigen Computerspielen (etwa World of Warcraft) waren auch mehrere Internet-Provider davon betroffen. Es ist klar, dass ein Trojaner, der Passwörter aufspürt, in ähnlicher Form auch geheime Schlüssel auslesen könnte.

17.2.1 Malware und digitale Signaturen

Das größte Aufsehen haben in den letzten Jahren Malware-Angriffe auf digitale Signaturen erregt. Um das Funktionieren eines solchen Angriffs zu veranschaulichen, nehmen wir an, dass Alice einen Webbrowser verwendet, der das digitale Signieren von HTML-Seiten unterstützt. Alices privater Signaturschlüssel sei in einer Smartcard sicher gespeichert, ihr öffentlicher Schlüssel sei dem Verifizierer bekannt. Mallory hat nun scheinbar keine Möglichkeit, eine Signatur zu fälschen. Mit folgendem Malware-Angriff geht es aber doch:

1. Mallory programmiert einen speziellen Trojaner und sorgt dafür, dass dieser auf Alices Betriebssystem zum Laufen kommt.
2. Will Alice über ihren Webbrowser eine digitale Signatur erstellen, dann greift der Trojaner zunächst nicht ein. Stattdessen wartet er, bis der Browser die zu signierende HTML-Seite darstellt und Alice zur Eingabe ihrer PIN für die Smartcard auffordert.
3. Wenn der Browser anschließend einen Hashwert der HTML-Seite zur Erstellung der Signatur an die Smartcard schickt, wird der Trojaner aktiv. Er fängt den Hashwert ab, ersetzt ihn durch einen anderen und leitet diesen dann an die Smartcard weiter. Der Trojaner führt also quasi eine Man-in-the-Middle-Attacke auf Alices PC durch. Der eingespielte Hashwert kann sich auf beliebige Daten beziehen, die einen für Mallory vorteilhaften Inhalt haben. Alice bemerkt von diesem Austausch nichts, da ihr Browser nur die originale HTML-Datei anzeigt.
4. Im letzten Schritt lässt der Trojaner Mallory die Signatur zukommen (beispielsweise per E-Mail).

Das Problem, das Mallory in diesem Fall ausnutzt, besteht darin, dass Alice etwas anderes sieht, als sie signiert (»What you see is not what you sign«). Man spricht in diesem Zusammenhang auch von einem **Darstellungsangriff**. Die Schwierigkeit im Zusammenhang mit Darstellungsangriffen besteht darin, dass Alice sie nur verhindern kann, indem sie Malware an sich verhindert. Genau das ist jedoch nicht einfach, denn digitale Signaturen kommen meist auf Client-Rechnern mit Netzanschluss zum Einsatz, die nur schwer gegen bösartige Programme abzusichern sind.

Theoretisch könnten die Hersteller von PCs Darstellungsangriffe verhindern, indem sie die Darstellung zu signierender Daten in ein Malware-sicheres Modul auslagern. Doch während es relativ einfach ist, den privaten Schlüssel in eine sichere Umgebung (z. B. Smartcard) zu verbannen, ist eine Auslagerung der kompletten Datendarstellung schon deutlich schwieriger. Trotzdem wurde auch Derartiges schon überlegt. Als 1997 das deutsche Signaturgesetz in Kraft trat, schlugen einige Verantwortliche allen Ernstes eine abgeschottete und versiegelte Darstellungshardware für PCs vor, die keine Einfallstür für Malware bieten sollte. Eine solche Darstellungskomponente sollte den gesamten Umgang mit der

Signatur übernehmen und die zu signierenden Daten auf einem eigenen Monitor anzeigen. Am Ende konnte sich dieser Vorschlag jedoch nicht durchsetzen – zum Glück, denn es ist kaum anzunehmen, dass der Markt eine Darstellungsspezialhardware zum Preis von über 1.000 Euro akzeptiert hätte.

Vier Jahre später zeigten die Malware-Spezialisten Adrian Spalka, Armin Cremers und Hanno Landweg von der Universität Bonn, dass Malware-Angriffe eine durchaus reale Bedrohung sind. 2001 veröffentlichten die drei eine Arbeit, in der sie Darstellungsangriffe auf verschiedene, unter Windows laufende Signaturprogramme beschrieben. Bei mehreren Produkten dieser Art war es ihnen gelungen, durch einen Trojaner die signierte Nachricht zu manipulieren [Roos]. Die Hersteller wiesen zwar zu Recht darauf hin, dass derartige Angriffe nie völlig zu verhindern sind, einige von ihnen mussten sich jedoch den Vorwurf gefallen lassen, nicht alle programmiertechnischen Möglichkeiten zur Verhinderung von Trojanern ausgeschöpft zu haben.

Einige IT-Sicherheitsexperten bezweifeln ohnehin, dass ein PC mit angeschlossener Smartcard die richtige Technologie für digitale Signaturen ist. Ross Anderson sieht beispielsweise einen PDA (heute würde man sagen: ein Smartphone) als bessere Alternative an [Ande08]. Diese Einschätzung ist zwar nicht von der Hand zu weisen, doch das Grundproblem eines Darstellungsangriffs bleibt auch auf einem Smartphone bestehen, sofern dieser ans Internet angeschlossen ist und die einfache Installation von Software zulässt. So bleibt Alice am Ende nur eine Möglichkeit: Sie muss ihren Computer – egal ob PC oder Smartphone – vor Malware schützen.

17.2.2 Vom Entwickler eingebaute Hintertüren

In den Bereich der Malware-Angriffe im weiteren Sinne fallen auch Schwachstellen, die der Entwickler einer Software selbst eingebaut hat (theoretisch könnte auch ein Chiffren-Designer eine Hintertür in sein Verfahren einbauen, doch das ergibt wenig Sinn, wie in Abschnitt 7.1.3 beschrieben). Ein bekanntes nichtkryptografisches Beispiel stammt aus dem Film War Games. In diesem nutzt ein Hacker eine vom Entwickler eingebaute Hintertür (in Form eines schwachen Passworts), um in den Computer des Verteidigungsministeriums einzudringen, wodurch beinahe ein Atomkrieg ausbricht. Es ist nie auszuschließen, dass auch der Entwickler einer Krypto-Software absichtlich eine Schwäche in sein Programm einbaut, und sei es nur zu Debug-Zwecken.

Wenn ein Programmierer die notwendige kriminelle Energie mitbringt, dann bieten sich teilweise recht subtile Möglichkeiten für eingebaute Hintertüren. 1993 machte der bekannte Kryptograf Gustavus Simmons auf eine solche Möglichkeit aufmerksam [Simmon]. Simmons zeigte, dass das Signaturverfahren DSA Alice es ermöglicht, einen Teil ihrer Signatur in nahezu beliebiger Form zu beeinflussen. Dadurch ist Alice in der Lage, eine kurze Nachricht in ihre Signatur zu

kodieren (man bezeichnet dies als Covert Channel). Ein böswilliger Entwickler kann dies nutzen, um in jeder Signatur einen Teil des privaten DSA-Schlüssels unterzubringen, ohne dass Alice etwas davon merkt.

Neben diesen bisher eher theoretischen Betrachtungen gab es auch schon einen konkreten Verdacht auf eine absichtlich eingebaute Hintertür. 1999 vermeldete der Chaos Computer Club in einer Pressemeldung, es gäbe eine solche für die NSA in Microsoft-Betriebssystemen [CCC]. Als Beweis präsentierte der Club einen öffentlichen RSA-Schlüssel, der in einer speziellen Windows-NT-Version den Namen NSAKEY trug. Der Verdacht auf eine NSA-Hintertür zerstreute sich jedoch schnell wieder. Den NSA-Schlüssel gab es zwar tatsächlich, er hatte allerdings eine vergleichsweise harmlose (allerdings auch undokumentierte) Funktion – er stellte sicher, dass das Windows-Betriebssystem von der NSA signierte Krypto-Module zuließ. Ansonsten waren nur von Microsoft signierte Module zulässig, was das Betriebssystem mit einem anderen RSA-Schlüssel überprüfte (eine solche Maßnahme war notwendig, um den damaligen Exportbestimmungen der USA zu genügen). Offensichtlich hatte ein Entwickler geschlampt, wodurch der seltsame Name des NSA-Schlüssels in einer Windows-NT-Version zu erkennen war.

Auch wenn sich der NSA-Schlüssel als heiße Luft erwies, kursieren seit den neunziger Jahren immer wieder Gerüchte, die etwas von versteckten Hintertüren in Krypto-Produkten aus den USA wissen wollen. Bisher wurde noch nie etwas derartiges bewiesen. Dennoch lehnen manche Unternehmen mit hohem Sicherheitsbedarf und insbesondere viele Behörden in aller Welt den Einsatz von US-Krypto-Produkten kategorisch ab – zur Freude der deutschen Krypto-Industrie.

17.2.3 Gegenmaßnahmen

Malware-Angriffen ist mit kryptografischen Mitteln kaum beizukommen. Dennoch gibt es wirksame Gegenmaßnahmen. In einem Unternehmen kann ein gutes Content-Security-Konzept die Gefahr von Malware deutlich senken. Heimanwender sollten sich (unabhängig von kryptografischen Überlegungen) einen guten Virenscanner besorgen. Sinnvoll ist darüber hinaus der Einsatz von Smartcards. Diese können zwar einen Darstellungsangriff nicht verhindern, sie sorgen jedoch dafür, dass die Malware nicht gleich Alices privaten Signaturschlüssel abgreift.

Zudem gilt die Offenlegung des Quellcodes einer Software als gute Maßnahme gegen Malware-Angriffe. Zwar lassen sich dadurch keine Viren verhindern, doch absichtlich (oder unabsichtlich) eingebaute Fehlfunktionen können auf diese Weise entdeckt werden. Ein offengelegter Quellcode verhindert sowohl Trojaner, die in scheinbar harmlosen Programmen versteckt sind, als auch absichtlich eingebaute Hintertüren in kryptografischer Software.

Wer seinen Quellcode nicht offenlegen will oder befürchtet, dass es zu wenige Experten geben wird, die sich den Quellcode anschauen, kann es mit einer Evalu-

ierung nach einem anerkannten Kriterienkatalog versuchen (siehe Kapitel 25). Eine solche Evaluierung verhindert zwar keine Viren, macht jedoch absichtlich eingebaute Hintertüren und Programmfehler deutlich unwahrscheinlicher. Allerdings ist eine Evaluierung ein teurer Spaß, der schnell sechsstellige Euro-Beträge kosten kann. Allzu eilig sollte man es damit auch nicht haben, da das gesamte Evaluierungsprozedere ein Jahr oder mehr in Anspruch nehmen kann.

Leider klingen diese Maßnahmen gegen Malware-Angriffe für einen Kryptografen reichlich unbefriedigend. In der Kryptografie ist man es gewohnt, dass ein Algorithmus bis zum Ende des Universums nicht zu knacken ist. Dies ist jedoch die Theorie, und Malware-Angriffe sind ein Teil der Praxis.

17.3 Physikalische Angriffe

Da wir Mallory als besonders durchtriebenen Zeitgenossen annehmen, müssen wir auch damit rechnen, dass er Alices PC oder deren Krypto-Hardware klaut. Sind darauf irgendwelche Schlüssel oder vertrauliche Daten im Klartext gespeichert, dann hat Mallory möglicherweise leichtes Spiel. Natürlich weiß Alice das. Deshalb wird sie versuchen, ihren PC oder ihr Krypto-Modul durch entsprechende Konstruktionsmaßnahmen zu schützen. Beispielsweise kann sie eine Smartcard einsetzen, die den darauf gespeicherten Schlüssel nicht auslesen lässt. Wenn Mallory versucht, mit den Mitteln der Physik an einen derart geschützten Schlüssel zu kommen, dann sprechen wir von einem **physikalischen Angriff**. Physikalische Angriffe sind vor allem für kryptografische Hardwareimplementierungen wie Smartcards, HSMs oder spezielle Krypto-Chips eine Gefahr.

17.3.1 Die wichtigsten physikalischen Angriffe

Wenn Mallory erst einmal eine Krypto-Hardware in seinen Händen hält, dann kann er eine ganze Reihe unterschiedlicher physikalischer Angriffe starten. Wie erfolgreich er dabei ist, hängt stark von seinem Budget ab, denn das Auslesen eines Schlüssels aus einer geschützten Hardware erfordert teilweise teure Hightech-Geräte. Leider würde eine ausführliche Betrachtung dieses Themas den Rahmen dieses Buchs bei weitem sprengen. Die folgenden Ausführungen liefern daher nur einen groben Überblick, für weitere Informationen empfehle ich [Ande08].

Wenn sich Mallory an einer Krypto-Hardware zu schaffen machen kann, dann kann er versuchen, deren Speicher auszulesen. Da wir an dieser Stelle nur physikalische Angriffe betrachten, nehmen wir an, dass es für Mallory keine Möglichkeit gibt, über die üblichen Funktionen des Moduls an den Speicherinhalt heranzukommen. Er muss also versuchen, direkt an den Speicher zu gelangen, was bei einem nicht speziell geschützten Modul mit gängiger Laborausrüstung möglich ist. Ein Problem hierbei besteht darin, dass flüchtiger Speicher seinen Inhalt verliert, wenn die Stromversorgung wegfällt. Ein Herausreißen des

Moduls kann daher dazu führen, dass die von Mallory begehrten Daten verschwinden. Er kann dies jedoch verhindern, indem er das Modul einer tiefen Temperatur aussetzt, bevor er die Stromversorgung unterbricht. Schon eine Temperatur von –20 Grad soll ausreichen, um den Speicherinhalt einige Minuten lang zu konservieren. Mit Röntgenstrahlen soll es ebenfalls möglich sein, flüchtige Speicherinhalte haltbar zu machen.

Wenn Mallory mit seiner elektrotechnischen Ausrüstung den Speicher nicht auslesen kann, dann kann er versuchen, diesen direkt in Augenschein zu nehmen und jedes Bit einzeln abzulesen. Dies kann er beispielsweise bewerkstelligen, indem er den Speicher in flüssiges Helium legt und anschließend unter dem Elektronenmikroskop untersucht. Technisch einfacher ist es meist, die Leitungen innerhalb eines Krypto-Moduls abzugreifen und deren Inhalt zu analysieren. Allerdings ist dies nur möglich, wenn Mallory in den laufenden Betrieb des Moduls eingreifen kann. Von Nachteil ist außerdem, dass ein Schlüssel nur selten über eine solche Leitung geschickt wird. Bei asymmetrischen Verfahren können die verschickten Informationen jedoch ausreichen, um den privaten Schlüssel zu rekonstruieren.

17.3.2 Gegenmaßnahmen

Die besten Gegenmaßnahmen gegen physikalische Angriffe sind selbst physikalischer Natur. Das Ziel hierbei muss es sein, dass sich Mallory nicht ungehindert an einem Stück Hardware zu schaffen machen kann.

Konstruktive Maßnahmen

Es bietet sich an, Maßnahmen gegen physikalische Angriffe in mehrere Schichten aufzuteilen. Die äußerste Schicht besteht aus gebäudetechnischen Maßnahmen, die verhindern sollen, dass Mallory überhaupt an das Modul herankommt. So sollte ein Raum in dem Krypto-Hardware aufbewahrt wird, abgeschlossen sein. Ein Krypto-Modul, das von einem Server genutzt wird, sollte samt Server in einem Rechenzentrum untergebracht sein. Dieses Rechenzentrum sollte durch dicke Wände, Alarmanlage, Sicherheitstüren und ähnliche Maßnahmen vor Mallorys Zugriff geschützt sein. Außerdem sollte Alice ihre Smartcard nicht unbeobachtet im Büro herumliegen lassen, sondern sie bei sich tragen oder sie wegschließen.

Hat Mallory die äußerste Sicherheitsschicht durchbrochen, dann sollten ihm in der nächsten Schicht die äußeren Schutzmechanismen des Moduls das Leben erschweren. In vielen Fällen reicht es aus, wenn eine Krypto-Hardware so beschaffen ist, dass man physikalische Angriffe erkennen kann (**Einbruchsevidenz**). Dies ist beispielsweise der Fall, wenn das Gehäuse robust ist und sich nicht ohne weiteres öffnen lässt. Siegellack, Plomben, Schaumstofffüllungen für die Hohlräume und ähnliche Hilfsmittel erfüllen oft denselben Zweck.

Wenn Einbruchsevidenz nicht ausreicht, dann ist ein **Einbruchsschutz** notwendig. Hier gibt es passive Maßnahmen wie die Verwendung robuster Gehäuse. Meist lohnt es sich, zusätzlich aktive Komponenten einzusetzen, die beispielsweise Alarm auslösen, wenn sich jemand daran zu schaffen macht. Es gibt auch Implementierungen, die für eine Löschung aller Schlüssel sorgen, sobald irgendwelche Sensoren verdächtige Werte melden. Dies kann etwa die Unterbrechung einer Stromleitung, eine extreme Temperatur oder eine mechanische Erschütterung sein.

Trotz allem kann es vorkommen, dass Mallory auch die zweite Schicht durchbricht und sich die Innereien eines Krypto-Moduls vornehmen kann. Es gibt jedoch Maßnahmen, mit denen ein solches selbst in diesem Fall noch Widerstand leisten kann – dies ist dann die innerste Sicherheitsschicht. So sind beispielsweise viele kryptografische Module (z.B. Smartcards) bewusst unübersichtlich konstruiert, damit Mallory die einzelnen Bestandteile und den Datenfluss nicht nachvollziehen kann. Zudem arbeiten solche Implementierungen oft mit Attrappen, die Mallory nicht vorhandene Bauteile vorgaukeln. Da die betreffenden Komponenten zudem meist mikroskopisch klein sind, ist es für Mallory oft schwer, den Überblick zu behalten. Zwar ist eine solche Vorgehensweise aus Sicht eines Kryptografen Security by Obscurity, doch für Hardwaredesign gelten nun einmal andere Gesetze als für das Chiffren-Design.

Sicheres Löschen

Ein wichtiger Beitrag zur Abwehr physikalischer Angriffe besteht darin, nicht mehr benötigte sensible Daten zu löschen. Dieses Löschen muss endgültig sein – Mallory darf keine Möglichkeit haben, es rückgängig zu machen. Allerdings ist ein solches **sicheres Löschen** einfacher gesagt als getan. Dies liegt zum Beispiel daran, dass Betriebssysteme Arbeitsspeicherbereiche oft auf die Festplatte auslagern. Das jeweilige Krypto-Programm bekommt davon möglicherweise gar nichts mit und weiß anschließend nicht, welche Daten auf der Festplatte es löschen müsste. Außerdem kann ein plötzlicher Ausfall der Stromversorgung dafür sorgen, dass das besagte Programm nicht mehr dazu kommt, etwas zu löschen. Wer eine Krypto-Software implementiert, sollte daher genau über das Speichermanagement des Betriebssystems Bescheid wissen, um Schlüsselleichen auf der Festplatte zuverlässig löschen zu können. Dieses Thema geht allerdings über den Inhalt dieses Buchs hinaus.

Doch selbst wenn Alices Software alle Schlüssel penibel löscht, reicht dies nicht unbedingt aus. Dies liegt daran, dass die meisten Betriebssysteme zu löschende Daten nicht wirklich aus dem Speicher entfernen, sondern sie lediglich zum Überschreiben freigeben – dabei bleiben die Daten zunächst erhalten. Alice sollte Schlüssel also löschen, indem sie die entsprechenden Speicherbereiche aktiv überschreibt. Allerdings reicht auch das nicht in allen Fällen aus. So haben magnetische Speichermedien in der Praxis die Eigenschaft, dass der Schreibkopf des

Laufwerks auf dem Medium Spuren hinterlässt, an denen man frühere Belegungen ablesen kann. Dazu braucht man zwar eine Hightech-Ausrüstung inklusive Elektronenmikroskop. Es gibt jedoch Unternehmen, die auf die Wiederherstellung verlorener Daten spezialisiert sind (**Datenrettung**) und dabei auch das Wiederherstellen gelöschter Daten als Dienstleistung anbieten.

Aus den genannten Gründen sollte Alice einen nicht mehr benötigten Schlüssel immer mehrfach überschreiben. Wie und wie oft, dazu gibt es unterschiedliche Angaben. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) empfiehlt in seinen *Richtlinien des BSI zum Geheimschutz von VS beim Einsatz von IT (VSITR)* ein sechsmaliges Überschreiben der Daten mit vorgegebenen Mustern [VSITR]. Ähnliche Methoden werden in den US-Standards *US NAVSO P-5239-26 (RLL)*, *US NAVSO P-5239-26 (MFM)* und *DOD 5220.22-M* beschrieben. Peter Gutman empfiehlt sogar eine 89-fache Überschreibung [Gutman]. Davon abgesehen gibt es eine weitere Maßnahme, die Alice beherzigen sollte: Nicht mehr benötigte Hardware, auf der noch Schlüsselleichen zu finden sein könnten, sollte sie komplett vernichten.

17.4 Schwachstellen durch Implementierungsfehler

Bisher sind wir davon ausgegangen, dass Alice und Bob auf Krypto-Produkte zurückgreifen können, die fehlerfrei funktionieren. In der Praxis ist eine solche Voraussetzung jedoch nie gegeben. Dies zeigen schon die zahlreichen Sicherheitslücken, die regelmäßig in Standardprogrammen auftauchen. Vom Webbrowser bis zum Betriebssystem gibt es kaum eine Anwendersoftware, die nicht ab und zu sicherheitskritische Fehler offenbart. Diese mit Patches und Work-arounds zu beheben, gehört zu den Routineaufgaben jedes Administrators. Doch selbst wenn der Hersteller nach Bekanntwerden eines solchen Fehlers die entsprechende Gegenmaßnahme bereitstellt, heißt dies noch lange nicht, dass alle Anwender und Administratoren diese umgehend nutzen. So ist es auch kein Zufall, dass der berüchtigte Internet-Worm, der 1988 buchstäblich das halbe Internet lahmlegte, zwei schon damals wohlbekannte und behebbare Schwächen von Unix-Betriebssystemen nutzte.

Dabei sind Implementierungsfehler keine Besonderheit der IT-Sicherheit oder gar der Kryptografie. Im Gegenteil: Die Entwicklung von Programmcode ist von jeher eine fehleranfällige Sache, und so gehören Systemabstürze, obskure Fehlermeldungen sowie andere informationstechnische Merkwürdigkeiten für den Computeranwender zum Alltag. Es scheint ein Naturgesetz zu sein, dass Software mehr Fehler enthält als beispielsweise Elektrogeräte oder Autos. Warum sollte die Kryptografie diesbezüglich eine Ausnahme machen?

17.4.1 Implementierungsfehler in der Praxis

Die meisten sicherheitsrelevanten Programmfehler fallen zwar nicht in den Bereich der Kryptografie. Dennoch gibt es mehr als genug davon. Hier ein paar Beispiele:

- 2000 entdeckte der Deutsche Ralf Senderek einen sicherheitskritischen Fehler in der Krypto-Software PGP [Sender]. Durch diesen Fehler war es auf einfache Weise möglich, der Software einen zusätzlichen Schlüssel unterzuschieben, der ein anschließendes Entschlüsseln des betreffenden Geheimtexts erlaubte.
- 2005 wurde ein Fehler in der Verschlüsselungsfunktion der Microsoft-Programme Word und Excel entdeckt [Bach05]. Dieser Fehler besteht darin, dass RC4-Initialisierungsvektoren mehrfach zum Einsatz kommen, was Mallory einen einfachen Angriff ermöglicht.
- 2006 zeigten die israelischen Wissenschaftler Omer Berkmann und Odelia Moshe Ostrovsky in einem Fachaufsatz, dass der Transport von Geheimnummern (PINs) zwischen Bankautomat und Bank-Rechenzentrum nicht sachgemäß verschlüsselt wird [BerOst]. Ein Insider kann dies nutzen, um ohne größeren Aufwand an PINs zu kommen. Da es sich um eine standardisierte Technologie handelt, sind praktisch alle Banken betroffen.

Alle genannten Beispiele betreffen Implementierungen, die weit verbreitet sind. Man kann daher vermuten, dass diese Fälle nur die Spitze des Eisbergs bilden. Aller Wahrscheinlichkeit nach produzieren weniger bekannte Anbieter sogar noch mehr Fehler als die Riesen der Branche, ohne dass dies auffällt oder gar von der Presse aufgegriffen wird. Alice und Bob müssen also jederzeit damit rechnen, dass die Krypto-Produkte, die sie einsetzen, Fehler enthalten. Nach den Gründen dafür muss man nicht lange suchen. Neben der allgemeinen Bug-Anfälligkeit von Software kommt oft Schlamperei dazu. Vor allem bei Produkten, die Kryptografie nur als eine von vielen Funktionen enthalten, hat sich schon so mancher Entwickler verschätzt. Das Problem hierbei ist, dass die Kryptografie nicht etwa weniger Aufmerksamkeit benötigt als andere Bereiche, sondern mehr. Dies liegt daran, dass kryptografische Software nicht nur korrekt laufen muss, sondern auch sicher. Wir werden noch sehen, wo der Unterschied liegt.

Die Hauptursache für die Bedrohung, die von Implementierungsfehlern ausgeht, liegt jedoch tiefer. Der Markt fordert nun einmal Softwareprodukte, die viele Features zu einem geringen Preis unterstützen und auf unterschiedlichen Plattformen verfügbar sind. Diese Lösungen müssen zudem auf den gängigen Betriebssystemen laufen, die wiederum selbst schon äußerst komplex und außerdem offen sind. Komplexität und Offenheit sind jedoch Feinde der Kryptografie.

17.4.2 Implementierungsfehler in vielen Variationen

Es ist schwierig, die zahlreichen Spielarten von kryptografierelevanten Programmfehlern in eine Übersicht zu bringen. Von Bedeutung sind für uns vor allem solche Fehler, die es Mallory ermöglichen, geheimes Schlüsselmaterial zu lesen. Andere Programmierfehler wiederum geben unter bestimmten Umständen den Klartext oder Teile davon preis. Oft ist der Übergang zwischen einem Programmfehler und einem unsicheren Protokoll fließend. Im Folgenden will ich einige Beispiele für Implementierungsfehler vorstellen, ohne das Thema vollständig zu behandeln.

Falsch implementierte Verfahren

Wenn ein Krypto-Verfahren selbst fehlerhaft implementiert ist, fällt dies meist schnell auf. Eine fehlerhafte Verschlüsselung wird oft dadurch entdeckt, dass die zugehörige Entschlüsselung nicht das gewünschte Ergebnis liefert. Die Sache ist jedoch nicht immer so einfach. Eine Feistel-Chiffre tut beispielsweise auch dann ihren Dienst, wenn eine Rundenfunktion nicht korrekt arbeitet. Da zum Ver- und Entschlüsseln dieselbe Rundenfunktion verwendet wird, hebt sich ein Fehler häufig selbst auf. Dabei kann ein Fehler in der Rundenfunktion die Sicherheit des Verfahrens zunichte machen.

Auch bei asymmetrischen Verfahren können sich Fehler einschleichen, die nicht sofort auffallen. Gerade bei RSA, dem bedeutendsten asymmetrischen Verfahren, ist eine sorgfältige Auswahl der Parameter notwendig. Dummerweise funktioniert das Verfahren jedoch auch bei schlecht gewählten Parametern, ohne dass etwas Auffälliges passiert. Ähnliches gilt auch für alle anderen gängigen asymmetrischen Verfahren.

Besonders anfällig gegenüber Implementierungsfehlern sind kryptografische Verfahren, bei denen es keine gegenläufigen Operationen wie Ver- und Entschlüsselung gibt. Zufallsgeneratoren sind ein Beispiel dafür. Einer Zufallsfolge ist es nicht auf den ersten Blick anzusehen, ob sie wirklich zufällig ist. Ähnlich verhält es sich mit einer kryptografischen Hashfunktion.

Um Implementierungsfehler bei kryptografischen Verfahren zu verhindern, muss die jeweilige Software auf geeignete Weise getestet werden. Die Beschreibung eines Verfahrens enthält aus diesem Grund meist ein paar Testvektoren, also Vergleichswerte (z.B. Klartext und zugehöriger Geheimtext), mit dem der Implementierer die Korrektheit der jeweiligen Funktion überprüfen kann. Schlecht gewählte RSA-Parameter oder falsch implementierte Zufallsgeneratoren wird man damit jedoch nicht ohne weiteres entdecken. Ähnlich schwierig zu finden sind Fehler in der Implementierung kryptografischer Protokolle. Eines ist daher nicht zu bestreiten: Schon der erste Schritt bei der Umsetzung kryptografischer Verfahren in die Praxis ist mit einigen Fallstricken versehen.

Fehler in der Zugangskontrolle

Viele sicherheitskritische Implementierungsfehler haben mit dem Thema Zugangskontrolle zu tun. In einem Client-Server-System mit vielen Benutzern muss genau geregelt sein, wer auf welche Daten zugreifen darf. Ein Fehler in der Zugangskontrolle kann schnell dazu führen, dass Alice auf einmal Daten lesen kann, die sie eigentlich nichts angehen, oder dass sie sogar den privaten Schlüssel anderer Leute nutzen kann.

Versäumtes Löschen

Wenn Mallory Zugriff auf Alices Rechner hat (etwa wenn er deren Laptop gestohlen hat), ergeben sich für ihn weitere Möglichkeiten, von Implementierungsfehlern zu profitieren. Wie bereits erwähnt, ist das sichere Löschen eine wichtige Funktion, die eine gute Krypto-Implementierung erfüllen muss. Tut sie das nicht, dann muss man dies als Fehler betrachten.

Integrationsfehler

Gute Bausteine sind keine Garantie für ein hochwertiges Haus. Diese Binsenweisheit gilt auch für die Kryptografie. So zeigt sich in der Praxis immer wieder, dass viele Implementierungsfehler aus der falschen Verwendung oder dem falschen Zusammenspiel korrekt implementierter Komponenten resultieren. Ein einfaches Beispiel: Eine Bibliotheksfunktion arbeitet mit einem geheimen Schlüssel und speichert diesen. Das Hauptprogramm hätte nun die Aufgabe, diesen Schlüssel nach der Verwendung zu löschen, tut dies aber nicht. Trotz einer fehlerfreien Bibliotheksfunktion hat sich also ein Fehler eingeschlichen. Ein weiteres Beispiel: Selbst die beste Implementierung einer Stromchiffre kann nicht verhindern, dass ein Programm sie mehrfach mit demselben Initialisierungsvektor und demselben Schlüssel aufruft. Eines ist daher klar: Obwohl es absolut sinnvoll ist, kryptografische Software in Form voneinander unabhängiger Module zu strukturieren, ist ein modularer Aufbau aus korrekt implementierten Komponenten alleine noch keine Garantie für ein fehlerfreies Programm.

17.4.3 Gegenmaßnahmen

Es versteht sich von selbst, dass kryptografische Maßnahmen gegen Implementierungsfehler wenig nützen. Wir müssen also auf einer anderen Ebene ansetzen, und zwar in der Softwareentwicklung. Das wichtigste Problem in diesem Zusammenhang ist, dass die Entwicklung kryptografischer Software andere Anforderungen stellt als die Entwicklung sicherheitsunkritischer Software. Software, die keine Sicherheitsfunktion erfüllt, gilt als korrekt, wenn sie alle Funktionen in der vorgesehenen Form ausführt. Bei kryptografischer Software gilt jedoch noch eine zusätzliche Anforderung: Eine solche ist erst dann korrekt, wenn Schlüssel und

Klartexte ausreichend geschützt sind. Dieser Grundsatz muss in jeder Phase der Softwareentwicklung beachtet werden.

Spezifikation

Da kryptografische Software zusätzliche Anforderungen erfüllen muss, muss schon die Spezifikation entsprechende Gesichtspunkte enthalten. Hier einige Beispiele:

- Schlüssel und Klartexte müssen schnellstmöglich und sicher gelöscht werden, wenn sie nicht mehr benötigt werden.
- Dauerhaft gespeicherte Schlüssel müssen verschlüsselt abgelegt werden.
- Ein Zufallsgenerator sollte einen geeigneten Selbsttest enthalten.
- Fehlermeldungen sollten so beschaffen sein, dass sie keine geheimen Informationen preisgeben.
- Insgesamt ist es von Vorteil, eine kryptografische Software modular zu gestalten. Wenn es beispielsweise für die symmetrische Verschlüsselung, für das kryptografische Hashen und für die Zufallsgenerierung jeweils ein eigenes, abgeschlossenes Modul gibt, dann hat dies aus Sicht der Sicherheit deutliche Vorteile. So können Alice und Bob jedes Modul einzeln analysieren und gegebenenfalls das verwendete Verfahren austauschen. Auch eine Sicherheitsanalyse des gesamten Systems wird deutlich einfacher, wenn es verschiedene Module mit klar definierten Aufgaben gibt.

Programmierung

Bei der Programmierung ist darauf zu achten, dass möglichst wenig Fehler entstehen. Dies ist eine Binsenweisheit, die für jede Software gilt und zudem nie vollständig erfüllbar ist. Die einzige praktische Empfehlung, die sich daraus ableitet, ist, dass kryptografische Software noch kritischer gegenüber Fehlern ist als andere Programme und dass Fehler oftmals schwerer entdeckt werden.

Tests

Es versteht sich von selbst, dass kryptografische Software einige Tests erfordert, die bei anderen Programmen nicht notwendig sind. Dabei muss die korrekte Ausgabe aller kryptografischen Funktionen (z.B. mit Testvektoren) überprüft werden. Darüber hinaus sind Tests eines gegebenenfalls vorhandenen Zufallsgenerators Pflicht. Außerdem sollte nach etwaigen Schlüssel-Speicherleichen gesucht werden.

Weitere Werkzeuge

Abgesehen von der Softwareentwicklung an sich gibt es einige weitere Werkzeuge, die Implementierungsfehler verhindern oder zumindest ihre Folgen abschwächen können. Hier die wichtigsten:

- *Offenlegung des Quellcodes*: Je mehr Experten sich ein Programm anschauen, desto größer die Wahrscheinlichkeit, dass etwaige Fehler entdeckt werden.
- *Auslagerung*: Geheime Schlüssel lassen sich in ein spezielles Speichermedium auslagern (z.B. in eine Smartcard), das keinen Zugriff auf den Schlüssel erlaubt. Sofern dieser Sicherheitsmechanismus fehlerfrei implementiert ist, kann auch eine fehlerhafte Software keinen Zugriff auf den Schlüssel gewähren.
- *Evaluierung*: Ein Weg zur Vermeidung von Softwarefehlern ist eine Evaluierung nach einem Kriterienwerk wie Common Criteria oder FIPS-140. Darum geht es in Kapitel 25.

Auch für den Anwender kryptografischer Software ergeben sich ein paar naheliegende Möglichkeiten, wie sie Implementierungsfehlern aus dem Weg gehen können. Dazu gehört einerseits das Setzen auf Bewährtes – Alice und Bob sollten im Zweifelsfall lieber eine bewährte Software einsetzen als die Betaversion des neuesten Krypto-Programms. Darüber hinaus sollten sie immer die neuesten sicherheitsrelevanten Updates installieren. Davon abgesehen gilt: Augen und Ohren offen halten, damit man rechtzeitig von etwaigen Fehlern in den verwendeten Krypto-Lösungen erfährt und entsprechend reagieren kann.

17.5 Insiderangriffe

Wer sich für Real-World-Attacken interessiert, sollte den Fachaufsatz *Why Cryptosystems Fail* von Ross Anderson lesen [Ande93]. Dieser ist zwar schon 20 Jahre alt, hat aber in der Zwischenzeit nichts von seiner Aktualität eingebüßt. In »Why Cryptosystems fail« beschreibt Anderson seine Erfahrungen im Zusammenhang mit Geldautomaten, die seinerzeit bereits in größerem Umfang mit kryptografischen Techniken abgesichert wurden. Wie aus heutiger Sicht nicht anders zu erwarten, stellte Anderson fest, dass die klassische Kryptoanalyse mit mathematischen Methoden bei den zahlreichen aufgetretenen Betrugsversuchen kaum eine Rolle spielte. Dafür gab es Real-World-Attacken unterschiedlicher Spielarten, wobei unter anderem Implementierungsfehler so manchen Angriff ermöglichten.

17.5.1 Unterschätzte Insider

Eine besonders interessante Erkenntnis ergab sich, als Anderson einen Blick auf die Herkunft der verzeichneten Angriffe warf: Auffällig viele Geldautomaten-Betrüger waren Insider – es handelte sich also um **Insiderangriffe**. So gab es Bankangestellte, die auf den Namen eines Kunden ohne dessen Wissen eine Karte ausstellten und damit Geld abhoben. Ein Wartungstechniker stattete einen Geldautomaten mit einem kleinen Computer aus, der alle Kontonummern und Geheimzahlen mitprotokollierte. Anderson berichtet, dass eine Bank mit 50.000 Angestellten mit etwa zwei Insider-Betrugsfällen pro Tag rechnen müsse.

Andersons Erkenntnisse decken sich mit der in IT-Sicherheitskreisen bekannten Tatsache, dass viele Angriffe (je nach Quelle zwischen 40 und 75 Prozent) einen internen Ursprung haben. Es ist schon erstaunlich, dass (wie Anderson berichtet) ein Bankangestellter für ein paar Tausend Pfund, die er einem Kunden abnahm, seinen Arbeitsplatz riskierte und in Kauf nahm, fortan keinen vergleichbaren mehr zu finden. Ein Grund für diese unerwartete Tatsache ist sicherlich, dass Geldautomaten-Betrüger mit Insiderstatus damals noch darauf hoffen konnten, dass man im Streitfall dem Kunden nicht glauben würde. Inzwischen dürfte jedoch klar sein, dass Banken und die Polizei bei jeder Unregelmäßigkeit von Anfang an einen Insiderangriff ins Kalkül ziehen.

17.5.2 Gegenmaßnahmen

Bei der Suche nach Maßnahmen zur Verhinderung von Insiderangriffen kommt man nicht umhin, den Mitarbeitern eines Unternehmens ein gewisses Misstrauen entgegenzubringen. Es bleibt also nichts anderes übrig, als Murphys Gesetz anzuwenden: Wenn ein Mitarbeiter die Möglichkeit zum Betrug hat, dann muss man davon ausgehen, dass er sie irgendwann nutzt. Diese Prämisse sollten Sie auf jeden Fall beachten, wenn Sie ein kryptografisch abgesichertes Computersystem planen.

Will man etwas genauer wissen, wie sich Insiderangriffe in der Praxis verhindern lassen, so stellt man leider fest, dass es dazu herzlich wenig Literatur gibt. Offenbar haben die IT-Sicherheitsexperten in diesem Bereich bisher vieles dem Zufall und dem gesunden Menschenverstand überlassen. Dennoch gibt es einige seit langem bekannte Maßnahmen, die Insiderangriffe deutlich erschweren. Eine wichtige Voraussetzung dafür ist stets ein durchdachtes Rollenkonzept. Eine Rolle ist eine Menge von Rechten und Pflichten, die einer Person zugeordnet ist. In einem komplexen Computersystem gibt es stets mehrere Rollen, angefangen vom Administrator über den Revisor bis zum Anwender.

Bei der Entwicklung eines Rollenkonzepts sollte man Insiderangriffe stets im Auge behalten, beispielsweise durch folgende Maßnahmen:

- **Need-to-know-Prinzip**: Jede Rolle sollte nur diejenigen Rechte erhalten, die sie benötigt.
- **Rollentrennung**: Es ist zwar nicht grundsätzlich verboten, dass verschiedene Rollen von derselben Person ausgefüllt werden. In einem Rollenkonzept sollte es jedoch Unverträglichkeitsregelungen geben, die sicherstellen, dass eine einzelne Person nicht zu viele Rechte erhält. Dies schließt ein, verschiedene Schritte eines sicherheitskritischen Prozesses von unterschiedlichen Personen erledigen zu lassen. Eine Regelung, die dieser Idee entspricht, ist vor allem in der Finanzbranche üblich. Sie lautet: Das operative Geschäft (z.B. Handel mit Aktien, Verkauf von Versicherungen) muss von der Geschäftsabwicklung

getrennt sein. Der Niedergang der britischen Barings-Bank im Jahr 1995, der durch die versteckten Spekulationen des Angestellten Nick Leeson verursacht wurde, war maßgeblich darauf zurückzuführen, dass es keine geeignete Rollentrennung gab.

- **Logdaten-Erhebung**: Alle Vorgänge innerhalb des Systems sollten automatisch protokolliert werden.
- **Vier-Augen-Prinzip**: Wichtige Vorgänge (etwa die Überweisung eines größeren Geldbetrags oder die Generierung eines Schlüssels) sollten nur durchgeführt werden, wenn zwei Personen daran beteiligt sind. Diese Regelung sollte nach Möglichkeit technisch erzwungen werden. Ross Anderson berichtet in der erwähnten Arbeit, dass die Zahl der Insiderangriffe auf das Zehnfache anstieg, nachdem einige Banken an einigen Stellen das Vier-Augen-Prinzip aus Kostengründen abgeschafft hatten.
- **Auditierung**: Ein wichtiges Instrument zur Aufdeckung von Insiderangriffen ist schließlich noch die Durchführung von Kontrollen. In diese müssen vor allem die Logdaten einbezogen werden.

Das große Problem bei den genannten Maßnahmen ist, dass sie einerseits einen großen Aufwand erfordern und lästig sind, andererseits aber weder den Umsatz erhöhen noch kurzfristig Kosten einsparen. Der Anlass für die Einführung eines angemessenen Rollenkonzepts mit entsprechenden Zusatzmaßnahmen ist daher oft genug ein Schadensfall.

17.6 Der Anwender als Schwachstelle

Mindestens genauso gefährlich wie ein krimineller Insider ist in vielen Fällen ein leichtsinniger Anwender. Wenn Sie also ein praxistaugliches Krypto-System zum Einsatz bringen wollen, sollten Sie sich auf jeden Fall Gedanken darüber machen, wer am Ende vor dem PC sitzt, um diesen zu bedienen. Dabei sollten Sie nicht unbedingt davon ausgehen, dass sich die Anwender freuen, Kryptografie einsetzen zu dürfen.

17.6.1 Schwachstellen durch Anwenderfehler

Anwender sind faul, vergesslich und interessieren sich nicht für Kryptografie. Welche Konsequenzen dies in der Praxis haben kann, wollen wir uns im Folgenden ansehen.

Verzicht auf Verschlüsselung

Selbst das beste Krypto-System hat für Alice und Bob keinen großen Nutzen, wenn sie es nicht verwenden. Diese Erkenntnis ist sicherlich nicht besonders überraschend. Sehr wohl überraschend ist dagegen, dass dieser Umstand zu den wichtigsten praktischen Problemen beim Einsatz von Kryptografie gehört. Mit anderen Worten: Viele Anwender sind schlichtweg zu bequem oder zu unwissend, um Kryptografie einzusetzen. Dieses Problem gab es schon zu Zeiten der Enigma, als so mancher Funker eine Nachricht im Klartext sendete – sei es, weil die Verschlüsselungsmaschine gerade defekt war oder weil der vereinbarte Schlüssel nicht funktionierte.

Bedienfehler

Schon im Zweiten Weltkrieg hielt sich nicht jeder Funker an die Vorschriften beim Umgang mit der jeweiligen Verschlüsselungsmaschine. Besonders verhängnisvoll endete dies, als im Jahr 1941 ein deutscher Soldat einen mit der Lorenz-Maschine verschlüsselten Spruch zweimal verschickte und dabei verbotenerweise den Schlüssel nicht wechselte. Dieser Fehler ermöglichte den Briten den Durchbruch beim Knacken der Lorenz-Maschine.

Auch im 21. Jahrhundert sind kryptografische Anwendungen vor Fehlbedienungen nicht gefeit. Wenn ein Anwender beispielsweise den Unterschied zwischen einer Verschlüsselung und einer digitalen Signatur nicht kennt, dann hat Mallory möglicherweise leichtes Spiel. Es kommt zudem auch vor, dass ein Anwender versehentlich auf den »Senden«-Button klickt, ohne vorher die vorgesehene Verschlüsselung durchzuführen. Wer kryptografische Software entwickelt, sollte solche Probleme stets im Auge behalten.

Leicht zu erratende Passwörter

Im Zweiten Weltkrieg verwendeten Funker immer wieder Enigma-Schlüssel wie AAA oder ABC. Die britischen Codeknacker mussten daher in vielen Fällen nicht lange nach den Schlüsseln suchen. Das gleiche Phänomen ist auch heute noch für viele Pannen bei der Verschlüsselung verantwortlich. Kryptografische Schlüssel werden oft aus einem Passwort generiert oder sind mit einem Passwort geschützt. Leider ist der Leichtsinn von Anwendern beim Umgang mit Passwörtern unter Sicherheitsexperten fast schon sprichwörtlich (siehe Abschnitt 21.1.1).

Abb. 17–3 *So mancher Anwender benötigt ein Dutzend oder mehr Passwörter. Kein Wunder, dass sich viele auf diese Weise behelfen.*

Social Engineering

Als **Social Engineering** bezeichnet man einen Angriff, bei dem Mallory versucht, durch Vorspiegelung falscher Tatsachen Sicherheitsvorkehrungen zu umgehen (etwa an Alices geheimen Schlüssel zu kommen). Eine einfache Form des Social Engineering besteht darin, dass Mallory einen Administrator der Firma Krypt & Co. anruft und diesen unter einem Vorwand nach einem gültigen Passwort fragt. Mallory kann sich auch als Reinigungskraft bei Krypt & Co. einstellen lassen oder einfach nur mit dem notwendigen Maß an Selbstsicherheit in die Firma spazieren und dort herumliegende CDs mitnehmen. Eine gute Quelle für interessante Informationen ist oft auch der Altpapier-Container eines Unternehmens. Weitere Methoden finden sich in [Eichle].

Social-Engineering-Angriffe gehören zum Standardrepertoire vieler Anbieter, die Sicherheitsanalysen durchführen. Meist versucht der Angreifer zunächst, ins Firmengebäude zu kommen, indem er sich eine vielfrequentierte Eingangstür vornimmt und sich dort von einem freundlichen Mitarbeiter die Klinke in die Hand geben lässt. Anschließend sucht der Angreifer nach ungesperrten PCs, schaut, was der Drucker gerade ausdruckt, oder bittet als angeblicher Mitarbeiter der IT-Abteilung vermeintliche Kollegen um ihre Mithilfe.


Stadtsparkasse München

Sehr geehrter Kunde,

Da gegenwärtig die Betrügereien mit den Bankkonten von unseren Kundschaften öfters zustande kommen, sind wir genötigt, nachträglich eine zusätzliche Autorisation von den Kunden der Stadtsparkasse München durchzuführen.
Der Sicherheitsdienst von der Stadtsparkasse München hat die Entscheidung getroffen, ein neues Datensicherheitssystem einzuführen. Im Zusammenhang damit wurden von unseren Fachleuten sowohl die Protokolle der Informationsübertragung, als auch die Methode der Kodierung der übertragenen Daten neu erstellt.

Infolgedessen bitten wir Sie, eine spezielle **Form der zusätzlichen Autorisation** auszufüllen.

FORM AUSFÜLLEN

Diese Sicherheitsregeln wurden nur zum Schutz der Interessen von unseren Kunden eingesetzt.

Danke für Ihre Zusammenarbeitarbeit,
Administration der Stadtsparkasse München

© 2005 Stadtsparkasse München

Abb. 17–4 *Gefälschte E-Mail, die den Empfänger dazu verleiten soll, auf einer fingierten Webseite sein Passwort einzugeben. Diese Vorgehensweise heißt Phishing.*

Solche Angriffe, bei denen sich Mallory direkt in die Höhle des Löwen wagt, haben zwar eine enorme pädagogische Wirkung, doch sie sind in der Praxis recht selten. Wesentlich häufiger sind dagegen E-Mails, mit denen der Angreifer an Passwörter gelangen will, oder entsprechende Telefonanrufe. Besonders beliebt sind seit einigen Jahren gefälschte E-Mails, die den Empfänger dazu verleiten sollen, auf eine bestimmte Webseite zu gehen und dort sein Online-Banking-Passwort einzutippen (**Phishing**). Hinter solchen E-Mails stecken oft kriminelle Organisationen, die sich auf diese Weise Zugang zu Online-Konten verschaffen wollen. Trotz zahlreicher Warnungen in den Medien fallen manche Menschen immer noch darauf herein.

17.6.2 Gegenmaßnahmen

Auch wenn der sprichwörtliche DAU (dümmster anzunehmender User) nie aussterben wird, gibt es einige wirksame Gegenmaßnahmen gegen Anwender-Fehlverhalten.

Sensibilisierung

Der erste Schritt zur Behebung der Schwachstelle Mensch besteht zweifellos darin, selbigen über die möglichen Gefahren aufzuklären. Gefordert ist also, das Bewusstsein (**Awareness**) des Anwenders für IT-Sicherheit zu schärfen. Dieses

Thema Awareness (genauer gesagt Security Awareness) hat in den Unternehmen in den letzten Jahren deutlich an Bedeutung gewonnen. Dies liegt zum einen an der Wichtigkeit des Ganzen, zum anderen aber auch daran, dass Awareness-Maßnahmen vergleichsweise kostengünstig sind und auf wenig Widerstand bei anderen Abteilungen stoßen. Gerade in den Jahren 2001 bis 2005, als die IT-Branche eine Krise durchmachte, mussten die IT-Sicherheitsverantwortlichen in den Unternehmen mit geringen Budgets auskommen und hatten es schwer, ihre Interessen gegenüber anderen Abteilungen durchzusetzen. Eine Awareness-Kampagne war in dieser Situation oft die einzige IT-Sicherheitsmaßnahme, die sich durchführen ließ.

Wie man sich leicht vorstellen kann, erfordern Awareness-Maßnahmen Fähigkeiten, die wenig mit Kryptografie zu tun haben. Stattdessen ist Know-how im Marketing und in der Personalführung notwendig. Viele Unternehmen entwickelten Plakate, Mauspads und andere Werbeträger, auf denen den Mitarbeitern Tipps zum Umgang mit der IT-Sicherheit gegeben werden. Andere laden ihre Mitarbeiter zu regelrechten Awareness-Shows ein, in denen der sicherheitsbewusste Umgang mit IT-Komponenten vermittelt wurde. Die Münchener Rückversicherung startete eine in der Szene viel beachtete Awareness-Kampagne mit unterschiedlichen Aktivitäten [Lardsc]. Der Energiekonzern RWE ließ sogar eine (an »Dilbert« erinnernde) Cartoon-Serie namens »Walt and Friends« entwickeln, um dadurch das Bewusstsein für Fragen der Security Awareness zu schärfen.

Eine weitere Methode, die in der Praxis zur Anwendung kommt, sind Bürobegehungen durch Sicherheitsexperten (in Abwesenheit der Mitarbeiter). Entdecken die Kontrolleure an einem Arbeitsplatz beispielsweise ein aufgeschriebenes Passwort oder eine liegen gelassene Smartcard, dann legen sie eine gelbe Karte (oder etwas Ähnliches) auf den Schreibtisch. Wer dagegen einen aus Sicht der IT-Sicherheit sauberen Arbeitsplatz hinterlassen hat, erhält ein Stück Schokolade oder eine andere symbolische Belohnung.

Solche Aktivitäten machen klar, dass so mancher IT-Sicherheitsexperte umdenken muss, wenn es um das Thema Awareness geht. Der Aufwand lohnt sich jedoch unbestritten. Schon allein, wenn es gelingt, den einen oder anderen Anwender dazu zu bewegen, eine vorhandene Krypto-Software auch tatsächlich einzusetzen, ist damit mehr erreicht, als das beste Verschlüsselungsverfahren erreichen könnte, wenn es nicht genutzt wird.

Benutzerfreundlichkeit

In [Ande08] berichtet der Autor Ross Anderson von einer Behörde, die einen sechsstelligen Betrag in eine Krypto-Lösung investierte, die einen verschlüsselten Datenaustausch ermöglichte. Leider war diese Vorrichtung äußerst umständlich zu bedienen. Die Folge war unvermeidlich: Die Mitarbeiter der Behörde umgingen die besagte Lösung und tauschten ihre Daten entgegen den Vorschriften unverschlüsselt aus.

Dieses Beispiel ist nur eines von vielen, das zeigt: Benutzerfreundlichkeit ist enorm wichtig in der Kryptografie. Anwender sind nun einmal von Natur aus faul. Ich habe selbst schon zahlreiche Projekte erlebt, in denen anfangs vereinbart wurde, alle E-Mails und alle wichtigen Dateien zu verschlüsseln, doch in vielen Fällen wurde dieser gute Vorsatz nicht durchgezogen. Sobald es auch nur einen Projektbeteiligten gibt, der keine Krypto-Software besitzt (vor allem in größeren Unternehmen ist es meist verboten, auf eigene Faust Programme auf dem PC zu installieren), wird das Verschlüsseln schon so umständlich, dass man es oft sein lässt. Einige Verschlüsselungsprogramme unterstützen zwar auch selbstentschlüsselnde Dateien (unter Windows sind dies exe-Dateien), doch diese werden von vielen Firewalls aussortiert. Angesichts solcher Widrigkeiten ist der Verzicht auf Verschlüsselung oft der Weg des geringsten Widerstands.

Aus den genannten Umständen lässt sich ein einfacher Schluss ziehen: Krypto-Software muss so benutzerfreundlich wie nur möglich sein. Jeder zusätzliche Button und jede unverständliche Fehlermeldung können den Ausschlag dafür geben, dass ein Anwender das Verschlüsseln zu umständlich findet und darauf verzichtet. Diese Erkenntnis hat sich bei Kryptografie-Herstellern natürlich herumgesprochen. Längst arbeiten bei solchen Anbietern mehr Anwendungsspezialisten als Kryptografen. Für kryptografische Entwicklungsprojekte, die eine Bedienung durch den Endanwender vorsehen, sollte man Zusatzaufwand einplanen, damit die Benutzeroberfläche auch wirklich DAU-kompatibel ist. Dass vor diesem Hintergrund auch eine gute Dokumentation notwendig ist, versteht sich von selbst.

Ein weiteres Problem, das oft selbst den verständnisvollsten Krypto-Anwender zur Weißglut bringt, ist mangelnde Interoperabilität. Wenn Alice und Bob verschlüsselt miteinander kommunizieren, müssen die beiden dasselbe Format und dieselben Verfahren unterstützen. Diese Voraussetzung ist jedoch in der Praxis oft genug nicht gegeben, was wieder einmal dazu führt, dass viele Anwender auf den Krypto-Einsatz verzichten. Das beste Beispiel liefert die E-Mail-Verschlüsselung, bei der aufgrund der zahlreichen Kommunikationsbeziehungen Interoperabilität besonders wichtig ist. Zwar hat sich in den letzten zehn Jahren S/MIME (Abschnitt 36.2) als Standard für diesen Zweck etabliert, doch das heißt noch lange nicht, dass zwei unterschiedliche S/MIME-Implementierungen problemlos miteinander reden können. Vor allem in den ersten Jahren hatten E-Mail-Anwender regelmäßig mit obskuren Fehlermeldungen, Abstürzen und ähnlichen Merkwürdigkeiten zu kämpfen. Oft war die Deinstallation der S/MIME-Software die einzige Möglichkeit, wieder ungestört arbeiten zu können. Unnötig zu erwähnen, dass diese Interoperabilitäts-Problematik der Akzeptanz von E-Mail-Verschlüsselung nicht gerade zuträglich war. Immerhin ist dieses Problem inzwischen deutlich kleiner geworden, da die verwendeten Programme verbessert wurden oder vom Markt verschwanden. Trotzdem ist mangelnde Interoperabilität noch immer ein Awareness-Killer.

Vorschriften

Da das Zuckerbrot (Awareness-Maßnahmen, Benutzerfreundlichkeit) allein nicht hilft, muss in Sachen Anwenderverhalten manchmal auch die Peitsche herhalten. Mit anderen Worten: Ein Unternehmen muss bestimmte Verhaltensweisen des Anwenders mit entsprechenden Vorschriften erzwingen. Schon im Arbeitsvertrag kann vom Arbeitnehmer ein angemessener Umgang mit sicherheitskritischen Daten verlangt werden. Details finden sich meist in speziellen Anweisungen. Solche Anweisungen sollten natürlich knapp und verständlich verfasst sein.

Vorschriften zum Thema IT-Sicherheit haben nur einen Sinn, wenn sie auch kontrolliert werden. Dies kann im Rahmen von Sicherheitsaudits geschehen. Außerdem muss es Strafbestimmungen geben (beispielsweise für das unverschlüsselte Verschicken einer sensiblen Mail), die im Extremfall bis zu einer Abmahnung oder Kündigung gehen können. Vorschriften zum sicheren Umgang mit Informationen sind ein wichtiger Bestandteil eines unternehmensweiten Sicherheitskonzepts.

Verhinderung

Am einfachsten lässt sich das Fehlverhalten von Anwendern verhindern, wenn dieses technisch gar nicht möglich oder zumindest umständlich ist. Dafür gibt es viele Beispiele:

- Passwörter lassen sich notieren und an Kollegen weitergeben, biometrische Merkmale (z.B. ein Fingerabdruck) dagegen nicht. Letztere zwingen den Anwender also, sicherheitskritische Dinge zu unterlassen.
- Ein E-Mail-Client oder ein Verschlüsselungs-Gateway lassen sich meist so konfigurieren, dass der Anwender zur Verschlüsselung gezwungen wird.
- Es gibt Verschlüsselungsprogramme, die in das Dateisystem integriert sind und automatisch alles verschlüsseln, was in einem bestimmten Speicherbereich abgelegt wird. Anwenderin Alice hat dadurch gar keine Möglichkeit, ihre Daten nicht zu verschlüsseln.

17.7 Fazit

Eine altbekannte Weisheit lautet: In einer Expertendiskussion nimmt nicht dasjenige Thema den meisten Raum ein, das am wichtigsten ist, sondern dasjenige, mit dem sich die Teilnehmer am besten auskennen. An diese Erkenntnis fühlt man sich fast zwangsläufig erinnert, wenn man an Real-World-Attacken denkt. Diese machen zwar in der Praxis einen Großteil aller Angriffe aus. In der wissenschaftlichen Forschung sind sie jedoch nur ein Randthema. Nur Seitenkanalangriffe und Implementierungsfehler spielen dort überhaupt eine Rolle, während Insiderangriffe und Anwenderfehlverhalten für viele Kryptografen Fremdwörter zu sein scheinen.

Der Grund für diesen Missstand ist klar: Fast alle Kryptografen sind von Hause aus Mathematiker, Informatiker oder Elektrotechniker. Da liegt es nun einmal nahe, sich mit Primzahlfaktorisierung, differenzieller Kryptoanalyse und Zeitangriffen zu beschäftigen. Psychologische und ergonomische Fragestellungen, die für die Entwicklung guter Krypto-Lösungen ebenso wichtig wären, kommen da fast zwangsläufig zu kurz.

Ich würde dies sogar als eine Grundlagenkrise der Kryptografie bezeichnen: Es werden immer mehr und immer bessere Algorithmen und Protokolle entwickelt, während es nur selten gelingt, diese auf ausreichend zuverlässige Weise einzusetzen. Real-World-Attacken sind daher meiner Meinung nach das derzeit größte Problem in der Kryptografie. Es genügt einfach nicht, Algorithmen zu verwenden, die Sicherheit bis zum Ende des Universums liefern, wenn Mallory durch einen Programmfehler oder eine Nachlässigkeit von Anwender Bob an den Schlüssel kommt.

Die Lösung dieses Grundlagenproblems ist zwar schwierig, doch es ist klar, wohin die Reise gehen muss. Anstatt die Kryptografie nur mathematisch und elektrotechnisch zu betrachten, müssen zukünftig auch die anderen in diesem Kapitel beschriebenen Fragestellungen eine Rolle spielen. Vor allem für Psychologen ergeben sich meiner Meinung nach spannende Aufgaben. Ich hoffe, dass derartige Arbeiten zukünftig gehäuft in Fachzeitschriften und -konferenzen auftauchen werden.

18 Standardisierung in der Kryptografie

Alice und Bob möchten verschlüsselt per E-Mail kommunizieren. Dummerweise haben sich die beiden zwar auf die zu verwendenden Krypto-Verfahren, jedoch nicht auf Details wie das Nachrichtenformat geeinigt. Nun rätselt Bob, was wohl in Alices E-Mail der verschlüsselte AES-Schlüssel und was die verschlüsselte Nachricht ist. Mit welcher Kodierung Alice den AES-Schlüssel vor dem Verschlüsseln mit RSA in eine Zahl umgewandelt hat, weiß Bob ebenfalls nicht.

18.1 Standards

Das beschriebene Problem stellt sich natürlich nicht nur für Alice und Bob, sondern auch und gerade, wenn es um einige Millionen Internetnutzer geht. Wenn Kryptografie in großem Umfang eingesetzt werden soll, dann müssen daher einheitliche **Standards** (ein anderes Wort für Standard ist Norm) festgelegt werden, an die sich alle halten sollten.

18.1.1 Standardisierungsgremien

Die Bedeutung von Standards geht weit über die Kryptografie hinaus. Deshalb gibt es unzählige Gruppierungen (Standardisierungsgremien), die Standards für alle Lebenslagen entwickeln. Viele davon sind staatliche Behörden oder gehören zu Industrieverbänden. Die wichtigste Standardisierungsorganisation weltweit ist die zur UNO gehörende International Organization for Standardization (**ISO**). Diese standardisiert schlichtweg alles, was standardisiert werden kann. Von der Reißfestigkeit von Kondomen über die Maße von Frachtcontainern bis hin zur Qualitätskontrolle im Management (ISO 9000) haben die Experten der ISO inzwischen weit über 10.000 Standards erarbeitet. Der ISO gehören die Standardisierungsbehörden aller wichtigen Staaten an, darunter auch das Deutsche Institut für Normung (DIN) und das US-amerikanische American National Standards Institute (ANSI). Viele von Industrieverbänden erstellte Standards werden von der ISO übernommen. Der für uns wichtigste Standard der ISO ist das ISO-OSI-Schichtenmodell, von dem in Kapitel 31 noch die Rede sein wird. Im Bereich der Computernetze sind vor allem folgende Standardisierungsorganisationen aktiv:

- Die ITU (International Telecommunication Union) ist die internationale Organisation der Telefongesellschaften. Für Telekommunikationsstandards ist die Abteilung ITU-T (früher CCITT) zuständig. Der für Kryptografen wichtige X.509-Standard stammt von diesem Gremium.
- Das ETSI (European Telecommunications Standards Institute) ist eine europäische Standardisierungsorganisation, die Standards für die Telekommunikation entwickelt. Vom ETSI kommen unter anderem die Mobilfunk-Standards GSM und UMTS.
- Das IEEE (Institute of Electrical and Electronics Engineers, sprich »Aitripeli«) ist ein weltweiter Zusammenschluss von Elektroingenieuren, der auch Standards veröffentlicht. Für uns am wichtigsten: IEEE 802.11 (WLAN), IEEE 802.1x (Authentifizierung in Rechnernetzen) und IEEE P1363 (Public-Key-Verfahren).
- Das American National Standards Institute (ANSI) veröffentlicht ebenfalls Standards, die für die Kryptografie relevant sind, beispielsweise die weiter unten in diesem Kapitel beschriebenen X.9-Standards.
- Das National Institute of Standards and Technology (NIST) gehört zum US-Handelsministerium und entspricht etwa der Physikalisch-Technischen Bundesanstalt (PTB) in Deutschland. Die Verfahren DES, DSA und die SHA-Familie stammen von dieser Organisation.
- Die IETF und das WWW-Konsortium entwickeln ebenfalls Standards. Das wird im folgenden Unterel beschrieben.

18.1.2 Standardisierung im Internet

Das Internet, in dessen Dunstkreis zahlreiche Krypto-Standards entstanden sind, hat seine eigenen Standardisierungsmechanismen. Im Mittelpunkt steht die Internet Engineering Task Force (IETF). Die IETF ist eine lose Gruppierung von Interessierten aus Wirtschaft und Forschung. Sie trifft sich in regelmäßigen Abständen und kommuniziert ansonsten über E-Mail und Mailinglisten. Wenn innerhalb der IETF der Wunsch für die Entwicklung eines Standards aufkommt, dann wird zunächst im zuständigen Arbeitsbereich eine Arbeitsgruppe gebildet, die einen **Internet Draft** (Standardvorschlag) entwickelt. Nun kommt die Internet Engineering Steering Group (IESG) ins Spiel, die ihr Einverständnis zu diesem Internet Draft geben muss. Ist dieses erfolgt, dann wird aus dem Internet Draft ein Request for Comment (**RFC**). Die erste Stufe eines RFC ist der Proposed Standard. Existieren nach sechs Monaten zwei unabhängige Implementierungen des RFC, so erhebt die IESG den RFC vom Proposed Standard zum Draft Standard. Nach weiteren Tests durch die IESG wird der RFC schließlich zum offiziellen Standard. Die Bezeichnung »Request for Comment« ist nicht besonders glücklich, da auch abgeschlossene Standards damit bezeichnet werden.

Nicht alle RFCs sind jedoch Standards oder angehende Standards. Es gibt auch RFCs, die bestimmte Verfahren spezifizieren oder Informationen zu einem bestimmten Gebiet liefern. RFCs sind durchnummeriert, inzwischen gibt es etwa 6.000. Viele davon sind jedoch von späteren RFCs abgelöst worden und daher nicht mehr gültig. Für uns sind vor allem die RFCs und Internet Drafts von Interesse, die mit Kryptografie zu tun haben. Viele davon – aber längst nicht alle – wurden im Arbeitsbereich Sicherheit der IETF entwickelt.

Für das World Wide Web entwickelt nicht nur die IETF, sondern auch das **World-Wide-Web-Konsortium** (**W3C**) Standards. Das W3C ist im Gegensatz zur IETF ein Industrieverband, seine Treffen sind daher nicht für alle Interessierten, sondern nur für Vertreter der Mitgliedsfirmen zugänglich. Die Arbeit des W3C ist aus diesem Grund etwas effektiver und schneller. W3C-Standards durchlaufen oft das IETF-Prozedere, wodurch sie zu offiziellen Internetstandards werden können.

18.2 Wissenswertes zum Thema Standards

In der Praxis hat sich immer wieder gezeigt, dass Standards auch nicht mehr sind als ein Stück Papier. Welcher Standard sich durchsetzt und welcher nicht, hängt von vielen Dingen ab, nicht immer jedoch vom Standardisierungsgremium. So mancher Standard erweist sich als fehlerhaft oder als zu kompliziert. Manche Standards kommen zu früh, manche zu spät, und manche setzen sich ganz einfach deshalb nicht durch, weil sich stattdessen ein anderer etabliert. Dabei spielen auch politische Aspekte eine Rolle: Ein marktbeherrschendes Unternehmen kann es sich oftmals leisten, seine eigenen Standards durchzudrücken, an die sich andere zähneknirschend halten müssen.

Ein Standard, der sich behauptet, wird oft zum Maß aller Dinge. Jeder, der ein Produkt verkaufen will, muss sich aufs Bit genau daran halten. Ein Standard, der sich nicht durchsetzt, ist dagegen kaum das Papier wert, auf dem er gedruckt ist. Die beschriebenen Standardisierungsgremien und Standardisierungsprozesse sind für uns daher gar nicht so wichtig. Wichtig ist, was sich durchsetzt. Fast alle internetrelevanten Kryptografie-Standards, die sich durchgesetzt haben, stelle ich Ihnen in diesem Buch vor.

Es kommt immer wieder vor, dass unterschiedliche Standards gleiche Verfahren oder Objekte verwenden. Zu diesem Zweck hat die ISO sogenannte **OIDs** (Object Identifier) eingeführt. Eine OID ist eine eindeutige Kennung eines Verfahrens oder Objekts, die im Idealfall in allen Standards weltweit einheitlich verwendet wird. OIDs sind hierarchisch aufgebaut. So beginnen etwa alle Internet-OIDs mit den Zahlen 1.3.6.1, da die ISO der zuständigen Internetorganisation IANA diesen Bereich zugewiesen hat. Alle Internet-OIDs, die in den Bereich Sicherheit fallen, beginnen mit 1.3.6.1.5.

OIDs spielen auch für die Kryptografie eine wichtige Rolle. So gibt es zahlreiche OIDs, die für ein Krypto-Verfahren stehen. Wenn ein Verfahren mit unterschiedlichen Schlüssellängen oder in unterschiedlichen Modi gebräuchlich ist, dann gibt es meist sogar mehrere OIDs. Die meisten Krypto-OIDs beginnen nicht mit den Internetzahlen 1.3.6.1.5, da sie auch in anderen Bereichen verwendet werden. RC4 hat etwa die OID 1.2.840.113549.3.4. Dabei ist 1.2.840.113549 die Zahl, die am Anfang aller OIDs steht, die von der Firma RSA Security beantragt wurden. Als Alternative zu einer OID bietet sich in manchen Fällen eine URL an (also eine Internetadresse wie http//:www.krypto.kl). Eine URL ist eindeutig und hat zusätzlich den Vorteil, dass man sie nutzen kann, um im Internet Informationen zum entsprechenden Objekt zu hinterlegen. Einige kryptorelevante Standards verwenden daher eine URL statt einer OID.

18.3 Wichtige Kryptografie-Standards

Kryptografie-Standards werden Sie in diesem Buch noch einige kennenlernen. Zum Warmwerden möchte ich Ihnen zunächst einige Vertreter vorstellen, die grundlegende Formate und Parameter festlegen und dadurch einen großen Einfluss auf die Nutzung von Kryptografie in der Praxis haben. Viele davon werden von anderen, komplexeren Standards verwendet.

18.3.1 PKCS

Die Public Key Cryptography Standards (**PKCS**) sind eine von der Firma RSA Security entwickelte Serie von Standards, die hauptsächlich Datenformate für verschiedene Bereiche der Kryptografie festlegen. PKCS entstand 1991, seitdem wird die Serie ständig weiterentwickelt. Den jeweils aktuellen Stand können Sie

auf der zugehörigen Webseite nachlesen [PKCS]. Die PKCS-Serie besteht momentan aus den Standards PKCS#1 bis PKCS#15, wobei die Nummern #2 und #4 inzwischen in #1 aufgegangen sind. Es gibt also derzeit 13 PKCS-Standards. Wie der Name sagt, geht es in PKCS vorwiegend um asymmetrische Verfahren. Da die unterschiedlichen PKCS-Standards voneinander unabhängig sind, kann man die PKCS-Serie als einen kryptografischen Werkzeugkasten betrachten, der für viele grundlegende Problemstellungen Formate und Verfahren bereitstellt. Die PKCS-Standards sind für die Kryptografie so bedeutend, dass ich auf jeden davon eingehen will. Hier sind zunächst einige der weniger wichtigen:

- **PKCS#3** beschreibt eine Methode zur Implementierung des Diffie-Hellman-Schlüsselaustauschs [PKCS#3]. Es handelt sich dabei um einen der einfachsten PKCS-Standards. In ihm werden vor allem die Datenformate und die Methoden für die Darstellung von Schlüsseln beschrieben. Zudem wird eine OID für den Diffie-Hellman-Schlüsselaustausch gemäß PKCS#3 festgelegt. Sie lautet 1.2.840.113549.3.
- **PKCS#9** beschreibt Erweiterungen für PKCS#6, PKCS#7 und PKCS#8 [PKCS#9].
- **PKCS#13** ist ein derzeit noch im Aufbau befindlicher Standard zum Thema ECC-Verfahren.
- **PKCS#14** ist ein ebenfalls noch im Aufbau befindlicher Standard zur Pseudozufallsgenerierung [PKCS#14].

Alle weiteren Mitglieder der PKCS-Familie werden Sie im Laufe dieses Buchs kennenlernen.

18.3.2 IEEE P1363

Neben den PKCS-Standards etabliert sich derzeit eine weitere Standardfamilie, die Formate und grundlegende Verfahren festlegt: IEEE P1363 oder kurz **P1363**. Wie der Name andeutet, steckt hinter dieser Serie von Standards die Organisation IEEE, die uns im Zusammenhang mit drahtlosen Netzen und deren Absicherung noch einmal begegnen wird. Wie PKCS ist auch P1363 eine kryptografische Werkzeugkiste für asymmetrische Kryptografie. P1363 ist jedoch deutlich umfangreicher als PKCS, da mehr Verfahren – darunter auch viele vergleichsweise neue Algorithmen – berücksichtigt werden. Dabei ist nicht zu übersehen, dass P1363 weniger RSA-lastig ist als PKCS (wen wundert's, schließlich stammt PKCS von der Firma RSA Security). Allerdings hat die Vielfalt des Standards und die Experimentierfreudigkeit der Entwickler dazu geführt, dass P1363 noch längst nicht die Verbreitung von PKCS hat. P1363 ist in fünf Teile aufgeteilt, von denen vier noch nicht abgeschlossen oder sogar erst im Entstehen sind.

Der erste Teil von P1363 ist der einzige, von dem bereits abgeschlossene Versionen vorliegen. Inhalt dieses Standards sind verschiedene Verfahren und For-

mate aus der asymmetrischen Kryptografie. Es handelt sich dabei um bewährte Verfahren, während neuere Entwicklungen in die anderen P1363-Teile geschoben wurden. Nachdem die Entwicklung von P1363 im Jahr 1994 begonnen hatte, konnte man 2000 den ersten abgeschlossenen Standard vermelden [P1363]. Dieser trägt den Namen P1363-2000 und behandelt asymmetrische Verfahren auf Basis des Faktorisierungsproblems (RSA und eine Variante davon), auf Basis des diskreten Logarithmus (unter anderem Diffie-Hellman und DSA) sowie einige ECC-Verfahren.

Da sich die Entwicklung des ersten P1363-Teils hinzog, gliederte man 1997 einige Bereiche aus und rief damit P1363a ins Leben. Im Jahr 2004 war auch dieser Standard abgeschlossen und wurde unter dem Namen P1363a-2004 veröffentlicht [P1363a]. Er enthält weitere Verfahren und Formate zu Verfahren auf Basis des Faktorisierungsproblems, des diskreten Logarithmus und elliptischer Kurven sowie einige weitere Ergänzungen zu P1363-2000.

Im Gegensatz zum ersten P1363-Teil, der in P1363a-2004 und P1363-2000 gegliedert ist, befinden sich die anderen drei noch in der Entwicklung. Man bezeichnet diese vier Teile als P1363.1, P1363.2 und P1363.3. Sie haben folgende Inhalte:

- *P1363.1 (Gitterbasierte Public-Key-Kryptografie)*: Dieser zweite Teil von P1363 behandelt gitterbasierte Kryptografie, genauer gesagt die Verfahren NTRU-Encrypt und NTRU-Sign [P1363.1]. Da sich diese beiden Verfahren bisher noch nicht in der Praxis durchsetzen konnten, muss sich erst noch zeigen, ob P1363.1 eine größere Bedeutung erlangen wird.
- *P1363.2 (Passwortbasierte Public-Key-Kryptografie)*: Hier werden mehrere Protokolle beschrieben, die eine Authentifizierung mit Schlüsselaustausch auf Basis von Passwörtern und asymmetrischer Kryptografie realisieren. Dazu gehören EKE, SPEKE und SRP [P1363.2].
- *P1363.3 (Identitätsbasierte Public-Key-Kryptografie mit Pairings)*: In diesem Teil werden identitätsbasierte Krypto-Systeme beschrieben [P1363.3]. Mehr zu diesem Thema gibt es in Abschnitt 26.6.

Weitere Standardisierungen werden derzeit diskutiert. P1363 ist zweifellos eine sehr interessante Standardserie. Lobenswert ist allemal, dass darin zahlreiche neue Verfahren vorkommen, die anderswo noch nicht standardisiert wurden. Allerdings haben große Teile der Standardserie eher experimentellen Charakter, und der große Umfang sowie die Komplexität machen P1363 unhandlich.

18.3.3 ANSI X.9

Vom American National Standards Institute (ANSI) stammen einige Krypto-Standards, die für das US-Finanzwesen entwickelt wurden. Diese Standards sind in der ANSI-Arbeitsgruppe X.9.F1 entstanden, weshalb sie »ANSI X.9« im

Namen tragen. Von den diversen ANSI-X.9-Standards sind vor allem diejenigen verbreitet, die die Verwendung asymmetrischer Krypto-Verfahren spezifizieren. Die folgende (unvollständige) Liste gibt einen Überblick:

- **ANSI X9.42**: In diesem Standard geht es um Schlüsselaustausch-Verfahren auf Basis des diskreten Logarithmus, genauer gesagt um Diffie-Hellman und MQV [X9.42]. Die aktuellste Version stammt aus dem Jahr 2003. Interessant ist auch der Anhang dieses Standards, in dem ein gängiger Pseudozufallsgenerator beschrieben wird (siehe Abschnitt 15.2.1).
- **ANSI X9.44**: Dieser Standard beschreibt die Verwendung des RSA-Verfahrens in verschiedenen Varianten [X9.44]. Die aktuellste Version von ANSI X9.44 stammt aus dem Jahr 2007.
- **ANSI X9.62**: Dieser Standard betrachtet Signaturverfahren auf Basis elliptischer Kurven [X9.62]. Die aktuelle Version stammt aus dem Jahr 2005.
- **ANSI X9.63**: Dieser X.9-Standard behandelt Verfahren auf Basis elliptischer Kurven, also ECDH und ECMQV [X9.63]. Die aktuellste Version stammt aus dem Jahr 2001.

18.3.4 NSA Suite B

NSA Suite B (verkürzt auch als **Suite B** bezeichnet) ist eine im Jahr 2005 von der NSA veröffentlichte Liste, die vier Krypto-Verfahren enthält. Details dazu finden sich in [Schm08/3]. Auf der Liste stehen ein symmetrischer Verschlüsselungsalgorithmus (AES), eine kryptografische Hashfunktion (SHA in den Varianten SHA-256 und SHA-384), ein Signaturverfahren (ECDSA) und ein Schlüsselaustauschverfahren (ECDH sowie als Alternative ECMQV). Auf Basis dieser vier Verfahren will die NSA in den nächsten Jahren die kryptografischen Lösungen modernisieren, die derzeit bei Behörden in den USA im Einsatz sind. Alle vier Verfahren sind nicht neu und werden bereits in anderen US-Standards beschrieben. Suite B enthält daher keine Spezifikationen, sondern verweist lediglich auf andere Standards.

Interessant an Suite B ist vor allem die Tatsache, dass die Algorithmenliste zwei Verfahren auf Basis elliptischer Kurven festschreibt, während das bisher deutlich weiter verbreitete RSA-Verfahren nicht aufgeführt ist. Die NSA kaufte vor der Veröffentlichung von Suite B sogar 26 ECC-Patente der kanadischen Firma Certicom auf, damit die Anwendung der asymmetrischen Verfahren auf der Liste ohne Lizenzzahlungen möglich ist (die NSA will von Unternehmen, die Suite-B-Produkte an US-Behörden liefern, keine Patentgebühren verlangen). Man kann Suite B daher als Bekenntnis der NSA zu elliptischen Kurven auffassen. Unternehmen, die Suite-B-Produkte an US-Behörden verkaufen wollen, müssen zunächst eines von mehreren Evaluationsprogrammen der NSA oder des NIST durchlaufen müssen. Diese Programme setzen auf den Common Criteria oder auf

FIPS 140-2 auf. Für ausländische Firmen ist es schwierig, zur Teilnahme an einem Evaluationsprogramm zugelassen zu werden.

Suite B gilt formal zwar nur für US-Behörden, doch die Signalwirkung geht weit über diesen Bereich hinaus. Insbesondere könnte Suite B dazu beitragen, dass sich ECC-Verfahren auch in anderen Ländern immer mehr durchsetzen. Bereits jetzt sind einige Produkte mit Suite-B-Unterstützung auf dem Markt. Dazu gehört vor allem das Betriebssystem Windows, dessen IPsec-Implementierung seit dem Service Pack 1von Vista Suite-B-Verfahren unterstützt. Zudem gibt es für die in Teil 5 dieses Buchs beschrieben Netzwerk-Protokolle IPsec [RFC4869], TLS [SaReHo] und S/MIME [RFC5008] Suite-B-Erweiterungen (die Literaturangaben beziehen sich auf die Erweiterung, nicht auf das Protokoll). Eine USA-Lastigkeit ist hierbei kaum zu übersehen. So stammen fast alle Suite-B-konformen Produkte aus den Vereinigten Staaten, und die genannten Protokollerweiterung sind alle unter Beteiligung der NSA entstanden. Es bleibt daher abzuwarten, ob und wie Suite B außerhalb der USA akzeptiert werden wird.

18.4 Standards für verschlüsselte und signierte Daten

Zu den wichtigsten Inhalten der kryptografischen Standardisierung gehören naturgemäß die verschlüsselten und signierten Daten an sich. Für diese muss es geeignete Formate geben, die es Alice und Bob beispielsweise erlauben, den verschlüsselten Teil vom Rest der Nachricht oder die Signatur vom signierten Text zu trennen. Im Folgenden schauen wir uns die wichtigsten Formate dieser Art an.

18.4.1 PKCS#7

PKCS#7 ist ein Standard, der ein Format für verschlüsselte oder signierte Nachrichten beschreibt [PKCS#7]. Dieses Format wird auch als **Cryptographic Message Syntax** (**CMS**) bezeichnet. Wie der Name andeutet, handelt es sich bei PKCS#7 um einen Standard aus der PKCS-Serie. Eine PKCS#7-Nachricht kann einen von sechs Inhaltstypen annehmen. Der Inhaltstyp ist abhängig davon, ob die Nachricht verschlüsselt, signiert oder gehasht wird. PKCS#7 ist rekursiv aufgebaut. Dies bedeutet, dass einige der Inhaltstypen ein Format vorsehen, das selbst eine PKCS#7-Nachricht enthält. Die sechs Inhaltstypen werden im Folgenden vorgestellt:

- *Data*: Dieser Inhaltstyp beschreibt ein einfaches Format für Daten, die weder verschlüsselt noch signiert noch gehasht sind. Durch diesen Inhaltstyp lässt PKCS#7 auch Nachrichten zu, auf die keine Kryptografie angewendet wurde.
- *Signed-data*: Dieser Inhaltstyp standardisiert ein Format für eine signierte Nachricht. Signiert wird (wie üblich) nicht die gesamte Nachricht, sondern nur deren Hashwert. Als Signaturverfahren wird RSA verwendet, das Format der Signatur entspricht dem PKCS#1-Standard. Die Nachricht, die signiert

wird, ist selbst wieder eine PKCS#7-Nachricht. Außer dieser Nachricht sieht das PKCS#7-Format eine Reihe weiterer Informationen vor, die mitsigniert werden. Dazu gehören etwa die OID der verwendeten kryptografischen Hashfunktion und des Signaturverfahrens sowie eine Versionsnummer (die aktuelle PKCS#7-Version ist 1.5) und einige Informationen über das digitale Zertifikat des Signierers. Durch diese Zusatzinformationen ist es für Empfänger Bob deutlich leichter, eine Signatur von Alice zu verifizieren – wie sollte er etwa eine Signatur verifizieren, ohne das verwendete Verfahren zu kennen.

- *Enveloped-data*: Dieser Inhaltstyp standardisiert ein Format für Daten, auf die ein Hybridverfahren (z.B. RSA+DES) angewendet wurde. Das Format sieht eine Nachricht vor, die mit einem symmetrischen Verfahren verschlüsselt worden ist. Der verwendete geheime Schlüssel wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt, der so entstandene verschlüsselte Schlüssel ist Bestandteil des Formats. Die verschlüsselte Nachricht ist selbst wieder eine PKCS#7-Nachricht. Eine Nachricht in diesem Format wird auch als **digitaler Briefumschlag** (digital envelope) bezeichnet.
- *Signed-and-enveloped-data*: Dieser Inhaltstyp sieht ein Format für eine signierte Nachricht vor, die zudem mit einem Hybridverfahren verschlüsselt worden ist. Die Nachricht ist selbst wieder eine PKCS#7-Nachricht.
- *Digested-data*: Dieser Inhaltstyp spezifiziert ein Format für eine Nachricht, auf die eine kryptografische Hashfunktion angewendet worden ist. Die Nachricht ist selbst wieder eine PKCS#7-Nachricht.
- *Encrypted-data*: Dieser Inhaltstyp beschreibt ein Format für eine Nachricht, die mit einem symmetrischen Verfahren verschlüsselt worden ist. Die Nachricht ist selbst wieder eine PKCS#7-Nachricht.

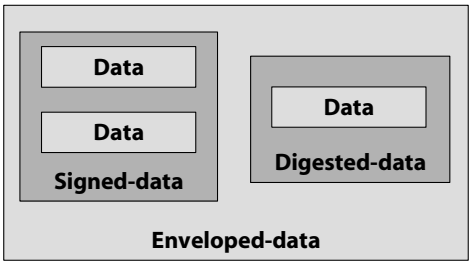


Abb. 18–1 *PKCS#7 beschreibt ein rekursives Format für verschlüsselte, signierte und gehashte Daten. In diesem Bild sind eine signierte und eine gehashte Einheit in einer verschlüsselten enthalten.*

Wie Sie erkennen können, sehen außer dem ersten alle Inhaltstypen ein Format vor, das selbst wieder eine PKCS#7-Nachricht enthält. Wenn Alice PKCS#7 anwenden will, dann benötigt sie zunächst einmal eine Nachricht, der sie den

Inhaltstyp Data zuweisen kann. Diese Nachricht kann sie dann für einen beliebigen anderen Inhaltstyp verwenden und erhält so eine neue PKCS#7-Nachricht. Durch den rekursiven Aufbau von PKCS#7 kann sie diese nun wieder für einen anderen Inhaltstyp verwenden.

18.4.2 XML Signature und XML Encryption

Die Beschreibungssprache XML (Extensible Markup Language) ist ein populäres Werkzeug, um Datenformate aller Art festzulegen. Das Grundprinzip hierbei ist recht einfach: Wie HTML verwendet auch XML sogenannte Tags, mit denen Daten strukturiert werden. Ein Tag hat immer die Form <xxx> oder </xxx>, wobei xxx für eine beliebige Buchstabenfolge steht. <xxx> markiert den Anfang, </xxx> das Ende einer Einheit. Eine solche Einheit wird als **XML-Element** (oder einfach nur Element) bezeichnet. Eine Eigenschaft eines Elements kann in einem Attribut festgelegt werden. Elemente können auch verschachtelt aufgebaut sein. Welche Elemente vorgesehen sind und welche Bedeutung diese haben, muss für die jeweilige Einsatzumgebung festgelegt werden.

Dementsprechend kann XML zur Festlegung von Datenformaten für jeden erdenklichen Zweck verwendet werden. Egal ob Texte, Konfigurationseinstellungen oder ausführbarer Programmcode – geeignet definierte XML-Elemente können die Grundlage dafür liefern. Die Idee von XML hat längst zahlreiche Anhänger gefunden, weshalb es inzwischen zahlreiche XML-basierte Formate für unterschiedliche Einsatzzwecke gibt. Da XML als Beschreibungssprache stark im Kommen ist, bietet es sich an, auch ein Format für verschlüsselte und signierte Daten auf XML-Basis festzulegen. Genau dies hat das WWW-Konsortium, das auch XML entwickelt hat, getan. Das Ergebnis sind zwei Standards namens XML Encryption und XML Digital Signature. Zusammen kann man diese als XML-basierte Alternative zu PKCS#7 bezeichnen.

XML Encryption

XML Encryption ist ein XML-basierter Standard, der ein Format für verschlüsselte Daten beschreibt [XMLE]. Er ermöglicht die Verschlüsselung eines gesamten XML-Dokuments, eines einzelnen XML-Elementes oder des Inhalts eines XML-Elements. Ähnlich wie PKCS#7 ist auch XML Encryption rekursiv aufgebaut. Innerhalb eines verschlüsselten XML-Bestandteils können also weitere verschlüsselte XML-Bestandteile vorkommen. Damit XML Encryption seine Funktion erfüllen kann, sieht es verschiedene Elemente vor, die diesem Zweck dienen:

- *EncryptedData*: Dieses Element steht für die verschlüsselten Daten. Ein Attribut informiert darüber, ob ein ganzes XML-Element oder nur der Inhalt des Elementes verschlüsselt wird.

- *EncryptionMethod*: Dieses XML-Element beschreibt den Algorithmus, der zur Verschlüsselung eingesetzt wird. Als Kennung für den Algorithmus wird keine OID, sondern eine URL verwendet. Verzichtet Absenderin Alice auf dieses Element, dann muss Empfänger Bob den Verschlüsselungsalgorithmus kennen.
- *KeyInfo*: Hier sind Informationen über den Schlüssel enthalten, mit dem die Daten verschlüsselt worden sind. Auch an dieser Stelle kommt eine URL als Kennung zum Einsatz.
- *CipherData*: Dies ist das verschlüsselte Element. Es beinhaltet entweder ein oder mehrere Elemente vom Typ CipherValue oder CipherReference.
- *CipherValue*: Dieses Element enthält – falls vorhanden – die verschlüsselten Daten.
- *CipherReference*: Dieses Element enthält einen Verweis auf die verschlüsselten Daten in Form einer URL. Ein XML-Encryption-Dokument muss also selbst keine verschlüsselten Daten enthalten, sondern kann diese an eine in diesem Element genannte Stelle auslagern.

Das folgende Beispiel zeigt eine Zahlungsinformation im XML-Format, bei der ein Element verschlüsselt ist (es handelt sich dabei um eine Kreditkartennummer):

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>Alice Onliner</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
                 xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <CipherData>
      <CipherValue>A23B45C56</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

XML Encryption wird von einigen anderen Standards verwendet, die in der Kryptografie eine Rolle spielen. Der bekannteste davon ist SAML (siehe Abschnitt 22.5).

XML Signature

Neben einem XML-Standard für die Verschlüsselung gibt es auch einen für signierte Daten. Dieser heißt **XML Signature** [XMLS]. Das Format, das dieser Standard beschreibt, sieht ein Signaturelement vor, das mit den Tags `<signature>` und `</signature>` markiert wird. Das Signaturelement enthält den zu signierenden XML-Text, verschiedene Elemente mit Zusatzinformationen (z. B. über die verwendeten Verfahren) und die Signatur selbst in Form eines dafür definierten XML-Elements. Zudem kann der öffentliche Schlüssel des Signierers enthalten sein, beispielsweise als digitales Zertifikat.

Auch XML Signature ist rekursiv aufgebaut, wodurch ein signiertes Element beliebig viele signierte Elemente enthalten kann. Die verwendeten Verfahren werden wiederum nicht mit einer OID, sondern mit einer URL identifiziert. Die Daten, auf die sich die digitale Signatur bezieht, müssen nicht im Signaturelement selbst vorhanden sein, sondern können auch ausgelagert und über eine URL referenziert werden. Natürlich gehen diese Daten dennoch in die Hashwert-Bildung ein. Hier ein einfaches Beispiel:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-c14n-
                                                                  20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
                                                                  20010315"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
</Signature>
```

Eine wichtige Erweiterung von XML Signature ist **XAdES** (XML Advanced Electronic Signatures) [XAdES]. XAdES beschreibt ein Format für digital signierte Daten, das sich für fortgeschrittene Signaturen gemäß der EU-Signaturrichtlinie nutzen lässt. Es gibt sechs Varianten des Formats, die jeweils ein unterschiedliches Sicherheitsniveau bieten.

18.4.3 Weitere Formate

Weitere Formate für verschlüsselte bzw. signierte Daten werden in den Standards OpenPGP und S/MIME beschrieben. Diese beiden Formate kommen vor allem für verschlüsselte bzw. signierte E-Mails zum Einsatz, obwohl sie auch für andere Zwecke geeignet sind. Um OpenPGP geht es in Abschnitt 36.3, um S/MIME in Abschnitt 36.2.

18.5 Standardisierungswettbewerbe

Viele Standards werden von Gremien entwickelt, deren Mitglieder häufig unterschiedliche Interessen haben. Dies führt dazu, dass so mancher Standard ziemlich kompliziert ist und viele Kompromisse enthält. Als Alternative haben sich in vielen Fällen **Standardisierungswettbewerbe** bewährt. Bei einem solchen haben Experten die Möglichkeit, Vorschläge einzureichen. Eine Kommission entscheidet, welche Einreichung gewinnt.

18.5.1 Der DES-Wettbewerb

Die Mutter aller Standardisierungswettbewerbe begann um das Jahr 1970. Zu diesem Zeitpunkt entwickelte sich der Computer von einem teuren Spezialgerät immer mehr zu einem Werkzeug, das von Industrie und Behörden für alltägliche Zwecke genutzt wurde. Die Notwendigkeit für eine zuverlässige Verschlüsselung war angesichts dieser Entwicklung offensichtlich. Doch obwohl es im Militär- und Geheimdienstbereich bereits damals ein umfangreiches Krypto-Know-how gab, war das öffentlich verfügbare Wissen zum Thema Kryptografie gleich Null. Niemand wusste also so recht, wie eine computergestützte Verschlüsselung aussehen sollte.

Um diesen Mangel zu beheben, hob die US-Behörde NBS (heute NIST) 1973 den ersten Krypto-Standardisierungswettbewerb der Geschichte aus der Taufe. Die Behörde rief dazu auf, Vorschläge für ein computertaugliches Verschlüsselungsverfahren einzureichen. Das beste davon sollte zu einem gesetzlich verankerten Standard erklärt werden. Der Aufruf des NBS verfehlte seine Wirkung nicht, und so gingen in der Folgezeit Algorithmen-Vorschläge in großer Zahl ein. Eine Sichtung der Einreichungen verlief jedoch enttäuschend – kein einziges Verfahren erschien den Standardisierern praxistauglich. Die erste Runde des ersten Wettbewerbs endete daher mit einem Abbruch. 1974 versuchte es das NBS mit einem erneuten Aufruf. Und dieses Mal gab es einen aussichtsreichen Kandidaten. IBM, die damals führende Computerfirma, hatte ein Verfahren namens *Lucifer* eingereicht. Lucifer war das Produkt eines IBM-Forschungsprojekts, in dessen Verlauf mehrere erfahrene Computerexperten ein sicheres und praxistaugliches Verschlüsselungsverfahren entwickelt hatten.

Als einziger ernst zu nehmender Kandidat gewann Lucifer den Wettbewerb. Die US-Geheimbehörde NSA nahm noch einige Änderungen vor, bevor das Ergebnis schließlich als Data Encryption Standard (DES) standardisiert wurde. Ungeteilte Zustimmung erhielt der DES jedoch nicht. Zunächst konnte sich kaum jemand einen Reim auf das komplizierte und scheinbar willkürlich aufgebaute Verschlüsselungsverfahren machen. Das NBS und die NSA trugen auch nicht gerade zur Erhellung bei, indem sie die Designkriterien für den DES geheim hielten. Erst nach und nach zeigte sich, dass das scheinbar wirre Design des DES bis in jede Einzelheit durchdacht und damit sehr wirkungsvoll war. So wurde der

DES zum wichtigsten aller Verschlüsselungsverfahren und zum Vorbild zahlreicher anderer Chiffren. Der Sieger des ersten Algorithmen-Wettbewerbs erwies sich damit allen anfänglichen Zweifeln zum Trotz als würdig.

18.5.2 Der AES-Wettbewerb

Angesichts der hohen Qualität des DES konnte sich das NBS (das zwischenzeitlich in NIST umbenannt wurde) mit dem nächsten Wettbewerb fast 25 Jahre Zeit lassen. Erst 1997 gab es ernsthafte Bestrebungen, einen neuen Verschlüsselungsstandard zu finden. Dieser sollte AES (Advanced Encryption Standard) genannt und im Rahmen eines Mitte 1998 gestarteten Wettbewerbs gefunden werden. Wer teilnehmen wollte, musste eine vollständig dokumentierte Blockchiffre mit Referenzimplementierung einreichen, die für Hard- und Software gleichermaßen geeignet und frei von Patentrechten war. Das Verfahren sollte drei unterschiedliche Schlüssellängen (128 Bit, 192 Bit und 256 Bit) erlauben. Die Länge eines Blocks wurde vom NIST auf 128 Bit festgelegt (also doppelt so groß wie beim DES). Zum Auftakt des AES-Wettbewerbs überprüfte das NIST lediglich formale Kriterien. Am 20. August 1998 veröffentlichte der Veranstalter eine Liste von 15 Verfahren, die diese erfüllten und damit für den eigentlichen Wettbewerb zugelassen wurden. Diese 15 Verfahren wurden fortan als AES-Kandidaten bezeichnet.

Die erste Runde

Bis Mitte 1999 wählte das NIST auf Basis von Expertenmeinungen die fünf besten Algorithmen aus den 15 AES-Kandidaten aus. Diese kamen in die zweite Runde und damit in die Endauswahl. Es handelte sich dabei um die Verfahren Twofish, MARS, Serpent, RC6 und Rijndael. Die restlichen zehn Verfahren mussten sich aus dem Wettbewerb verabschieden. Fünf davon traf die Höchststrafe: Sie wurden wegen ernst zu nehmenden Sicherheitslücken aussortiert. Zwar ist bis heute kein Angriff auf eines der Verfahren bekannt, der sich in der Praxis nutzen ließe, doch die Anforderungen in der Kryptografie sind nun einmal hart – alles, was besser ist als die vollständige Schlüsselsuche, gilt als Schwäche. Hier die Liste der Unsicheren:

- *LOKI97*: Diese Feistel-Chiffre mit 16 Runden zählte zu den größten Enttäuschungen des AES-Wettbewerbs [BroPip]. Obwohl mit Jennifer Seberry und Josef Pieprzyk zwei namhafte Kryptografen an der Entwicklung mitgewirkt hatten, zeigten sich bei der Sicherheit einige Schwächen. Mit linearer Kryptoanalyse gelang eine Known-Plaintext-Attacke mit 2^{56} Klartextblöcken.
- *FROG*: Auch dieses Verfahren kann per linearer Kryptoanalyse mit einer Known-Plaintext-Attacke mit 2^{56} Klartextblöcken gebrochen werden [GeLeCh]. Die Performanz erwies sich ebenfalls als nicht berauschend.

- *DEAL*: Diesen Algorithmus kann man als Anpassung des DES an die AES-Kriterien bezeichnen [Knudse]. Dennoch zeigten sich einige potenzielle Sicherheitslücken in der Schlüsselaufbereitung. Die Performanz entspricht etwa der von Triple-DES, was sich als zu langsam erwies.
- *HPC*: HPC steht für »Hasty Pudding Cipher« [Schroe]. Der kuriose Name konnte nicht verhindern, dass das Verfahren in der ersten Runde ausschied. Dies lag unter anderem daran, dass HPC-Erfinder Richard Schroeppel einige exotische Funktionselemente in sein Verfahren einbaute, deren kryptografische Eigenschaften noch nicht ausreichend analysiert sind. Zudem erwies sich die Schlüsselaufbereitung als unsicher, da für eine große Anzahl an Schlüsseln jeweils 2^{30} äquivalente Schlüssel existieren.
- *MAGENTA*: Das einzige deutsche Verfahren im AES-Wettbewerb. Es wurde von Mitarbeitern der Deutschen Telekom entwickelt. Leider unterlief diesen ein schwerer Fehler bei der Schlüsselaufbereitung, der schon bei der Vorstellung auf der ersten AES-Konferenz auffiel (Details dazu gibt es in Abschnitt 9.7).

Fünf weitere Algorithmen mussten die Segel streichen, obwohl sie keine größeren Sicherheitsmängel aufwiesen. Grund für das Ausscheiden war die geringe Performanz auf unterschiedlichen Plattformen oder der zu hohe Speicherverbrauch. Folgende Verfahren waren betroffen:

- *E2*: E2 ist eine Feistel-Chiffre und stammt aus Japan [KMAUOT]. E2 gilt als das beste Verfahren, das die zweite Runde nicht erreichte. Eine etwas zu geringe Performanz sowie ein knapp bemessener Sicherheitsspielraum gaben den Ausschlag.
- *CAST-256*: Dieses Verfahren ist eine Variante von CAST, um das es in Abschnitt 9.6 geht. Es erwies sich als zu langsam und verbrauchte zu viel Speicher.
- *SAFER+*: Eine Variante von SAFER, das Sie bereits aus Abschnitt 9.5 kennen. Das Verfahren erwies sich als zu langsam, um in die zweite Runde zu kommen. Die in Abschnitt 9.5 erwähnte Meet-in-the-Middle-Attacke wurde als weniger bedeutend eingeschätzt.
- *CRYPTON*: CRYPTON stammt von einem südkoreanischen Kryptologen [LimHwa]. Es hat einige Ähnlichkeit mit Rijndael, erreicht jedoch dessen Verschlüsselungsgeschwindigkeit nicht.
- *DFC*: Diese aus Frankreich stammende Feistel-Chiffre schied ebenfalls wegen einer nicht ausreichenden Performanz aus [GGHNPP]. Vor allem auf Nicht-64-Bit-Architekturen erschien das Verfahren zu langsam.

Die zweite Runde

Von den fünf Algorithmen, die es in die zweite Runde geschafft hatten, schieden die folgenden zwei vorzeitig aus:

- *RC6*: Dieses Verfahren von Ron Rivest kennen Sie aus Abschnitt 9.3. Es schnitt bezüglich der Performanz etwas schlechter ab als die drei Finalisten. Die schlüsselabhängigen Rotationen, die das Verfahren vorsieht, wurden in der Bewertungsphase viel diskutiert und wirkten sich negativ auf die Siegchancen aus.
- *MARS*: MARS stammt von der Firma IBM, die auch den DES entwickelte. Das Verfahren wird in Abschnitt 9.4 beschrieben. Nachdem MARS die erste Runde unbeschadet überstanden hatte, wurden in der zweiten einige Ungereimtheiten offenkundig.

Die Entscheidung

Die folgenden drei Algorithmen schnitten beim AES-Wettbewerb am besten ab:

- *Rijndael*: Dieses Verfahren wird in Kapitel 8 ausführlich beschrieben.
- Twofish: Bruce Schneiers Twofish kennen Sie bereits aus Abschnitt 9.2. Neben dem Sieger Rijndael und Serpent schnitt es im AES-Wettbewerb am besten ab. Eine geringere Performanz im Vergleich zu Rijndael und das etwas weniger elegante Design gaben den Ausschlag, dass Twofish nicht zum AES gekürt wurde.
- *Serpent*: Diesen Algorithmus habe ich in Abschnitt 9.1 vorgestellt. Sein konservatives Design macht Serpent vergleichsweise langsam. Deshalb konnte sich das Verfahren nicht gegen Rijndael durchsetzen.

Über diese drei Verfahren wurde heftig diskutiert, bevor das NIST am 2. Oktober 2000 schließlich den Gewinner bekannt gab: Rijndael hatte das Rennen gemacht. Dass sich Rijndael gegenüber Twofish durchsetzte, lag daran, dass Letzterer etwas langsamer und komplexer ist. Serpent hatte in Softwareimplementierungen die etwas geringere Performanz gezeigt. So gesehen ist Rijndael sicherlich ein würdiger Sieger, auch wenn sich einige Kryptografen aufgrund einer höheren Anzahl von Runden für Twofish oder Serpent ausgesprochen hatten (es wurde auch – allerdings erfolglos – vorgeschlagen, die Rundenzahl von Rijndael zu erhöhen).

Fazit

Die gesamte Fachwelt ist sich darüber einig, dass der AES-Wettbewerb in jeder Hinsicht ein voller Erfolg war. Vor allem die Idee, den AES in Form eines Wettbewerbs festzulegen (und nicht etwa durch ein Gremium), wurde immer wieder gelobt. Darüber hinaus waren die Entscheidungen des NIST allesamt nachvollziehbar, die vom DES bekannte Geheimniskrämerei blieb aus. Noch heute schwärmen viele Kryptologen vom AES-Wettbewerb. Angesichts dieser uneingeschränkt guten Erfahrungen muss man sich fast zwingend folgende Frage stellen: Lässt sich das Prinzip eines Algorithmen-Wettbewerbs auch auf andere Bereiche der Kryptografie oder gar auf völlig andere Gebiete übertragen? Ein Kandidat

wäre sicherlich das Protokoll IPsec gewesen (Kapitel 34), das so ziemlich alle Nachteile hat, die von einer Gremiumsentwicklung zu erwarten sind: IPsec ist komplex, es fehlt eine einheitliche Designlinie, und es hat vermeidbare Schwächen. Nicht nur Krypto-Papst Bruce Schneier vertritt daher die Ansicht, dass ein IPsec-Wettbewerb eine sinnvolle Sache gewesen wäre [FeSc00].

Ein anderes Beispiel, das nichts mit Kryptografie zu tun hat: die deutsche Rechtschreibreform. Auch diese wurde von einem vielköpfigen Gremium in jahrelanger Arbeit hinter verschlossenen Türen ausgetüftelt. Das Ergebnis war eine Ansammlung von Kompromissen, entbehrte einer klaren Linie und wies zahlreiche innere Widersprüche auf. Vielleicht hätte ein Wettbewerb nach AES-Vorbild zu einem besseren Ergebnis geführt. Man sollte jedoch nicht vergessen, dass der AES-Wettbewerb ideale Voraussetzungen hatte, die in dieser Form nur selten anzutreffen sind. So waren sich alle einig, dass ein neuer Standard notwendig war – bei der Rechtschreibung hätten dagegen viele am liebsten ganz auf eine Reform verzichtet. Zudem musste die AES-Jury kaum Rücksicht auf Lobbygruppen nehmen – bei der Entwicklung von IPsec saßen dagegen Industrievertreter im Boot, die auf eine Wahrung ihrer Interessen achteten. Ein IPsec-Wettbewerb wäre dadurch deutlich komplizierter geworden, als es der AES-Wettbewerb war. Doch allen Einwänden zum Trotz: Als Alternative zur Gremienarbeit ist ein Wettbewerb immer eine Überlegung wert.

18.5.3 Der SHA-3-Wettbewerb

Die in den USA standardisierte Hashfunktion SHA-1 galt lange als sicher (siehe Abschnitt 14.2.1). 2004 traten jedoch erstmals Schwächen in diesem Verfahren zu Tage, und in den Folgejahren kamen weitere dazu. Zwar gab es mit den SHA-2-Verfahren eine ebenfalls standardisierte Alternative, die von den neu entdeckten Angriffen nicht betroffen war, doch durch die Ähnlichkeit von SHA-1 und SHA-2 musste man damit rechnen, dass auch SHA-2 irgendwann unter die Räder kommen würde. Die US-Standardisierungsbehörde NIST beschloss daher, im Rahmen eines Wettbewerbs ein neues Verfahren zu suchen. 2007 wurde die »SHA-3 Challenge« gestartet.

Bis zum 31. Oktober 2008 konnten Kryptografen ihre Verfahren beim NIST einreichen. Bezüglich des Designs machte das NIST kaum Einschränkungen. Gefordert war lediglich, dass Hashwerte der Länge 224, 256, 384 und 512 Bit möglich waren. Eine Hashwert-Länge von 160 Bit, wie sie von SHA-1 unterstützt wurde, war bei SHA-3 dagegen optional. Die Zahl der Teilnehmer lag schließlich bei 64 und war damit deutlich größer als beim AES-Wettbewerb. Auch zwei Produktionen aus Deutschland waren dabei: das Verfahren **Twister**, das von Studenten der Universität Weimar entwickelt wurde, und **MeshHash** von Björn Fay. Weitere Kryptografen aus Deutschland, Österreich und der Schweiz mischten als Mitglieder internationaler Teams im Wettbewerb mit.

Nur 51 der 64 SHA-3-Einreichungen ließ das NIST zur ersten Wettbewerbsrunde zu, nachdem die anderen offensichtliche Mängel aufgewiesen hatten (beim AES wurden seinerzeit alle Einreichungen akzeptiert). Von den zugelassenen Verfahren kamen 14 in die zweite Runde. 16 davon waren wegen Sicherheitsmängeln ausgeschlossen worden. Zehn der Ausgeschiedenen wurden von den Einreichern selbst für untauglich erklärt. Prominentestes Opfer in dieser Phase war das Verfahren **MD6** des in diesem Buch mehrfach erwähnten Ron Rivest. Besser schnitt dagegen die kryptografische Hashfunktion **Skein** ab (siehe Abschnitt 14.4.5). Diese erzielte eine hohe Aufmerksamkeit, weil der bekannte Kryptologe Bruce Schneier zum Entwicklerteam gehörte. Mit Stefan Lucks von der Universität Weimar war auch ein Deutscher an der Entwicklung beteiligt. Neben Schneier und Lucks zählten sechs weitere Kryptografen zum Skein-Team.

Nach zahlreichen Analysen und Diskussionen ließ das NIST Ende 2010 schließlich fünf Verfahren für die Endrunde zu: Neben Skein waren dies **JH**, **Grøstl**, **BLAKE** und **Keccak**. Nun zeichnete sich jedoch ab, dass keiner der Finalisten wesentliche Vorteile gegenüber SHA-2 zu bieten hatte. Trotz gegenteiliger Befürchtungen war es niemandem gelungen, die erfolgreichen Angriffe gegen SHA-1 auf SHA-2 auszudehnen – ein wichtiger Grund für die Durchführung des SHA-3-Wettbewerbs fiel damit weg. Darüber hinaus konnte keiner der fünf SHA-3-Finalisten nennenswerte Vorteile in der Ausführungsgeschwindigkeit gegenüber SHA-2 vorweisen. Es wurde daher ernsthaft diskutiert, den Wettbewerb ohne Sieger enden zu lassen.

Doch das NIST entschied anders. Am 2. Oktober 2012 (also genau 12 Jahre nach Verkündung des AES-Siegers) erklärte es Keccak zum Sieger. Vor allem das elegante Design, der klare Aufbau und die Eignung für viele Plattformen hatten die Behörde von Keccak überzeugt. Die Frage, warum das NIST überhaupt einen Sieger festlegte, obwohl keiner der Kandidaten wesentlich besser als SHA-2 war, wurde wie folgt beantwortet: Es kann nicht schaden, zwei standardisierte Verfahren zu haben. Dies bietet nicht nur Flexibilität, sondern auch Sicherheit – sollte eines der beiden Verfahren gebrochen werden, dann ist das andere eine naheliegende Alternative. Wie Keccak funktioniert, können Sie in Abschnitt 14.3 nachlesen.

18.5.4 Weitere Wettbewerbe

Die positiven Erfahrungen mit dem AES-Wettbewerb führten dazu, dass in den Folgejahren weitere Veranstaltungen mit ähnlichem Ablauf gestartet wurden – auch wenn es dabei meist nicht um einen Standard, sondern nur um eine unverbindliche Empfehlung ging. Die folgenden Abschnitte geben einen Überblick.

NESSIE

NESSIE ist ein inzwischen abgeschlossenes europäisches Forschungsprojekt, dessen Inhalt ein Wettbewerb zur Findung starker kryptografischer Verfahren war [NESSIE]. Die Abkürzung NESSIE steht für *New European Schemes for Signatures, Integrity and Encryption*. Im Gegensatz zum AES-Wettbewerb ging es bei NESSIE nicht nur um symmetrische Verschlüsselungsverfahren. Vielmehr wurden in sieben Kategorien Sieger gekürt, darunter teilweise sogar mehrere pro Kategorie, um unterschiedliche Block- und Schlüssellängen zu unterstützen. NESSIE startete 1999. Es gab insgesamt 42 Einreichungen, was bei sieben Kategorien nicht unbedingt viel ist. 2003 wurden die Sieger verkündet:

- *Blockchiffre*: In dieser Kategorie gab es drei Unterkategorien: 64, 128 und 256 Bit Blocklänge. Die Sieger hießen MISTY1 (64 Bit), Camellia und AES (128 Bit) sowie SHACAL-2 (256 Bit).
- *Stromchiffre*: Diese Kategorie blieb ohne Sieger. Alle drei Kandidaten fielen durch.
- *Kryptografische Hashfunktion*: Hier gewannen WHIRLPOOL, SHA-256, SHA-384 und SHA-512.
- *Schlüsselabhängige Hashfunktion*: Auch hier gab es mit UMAC, TTMAC, EMAC und HMAC mehrere Sieger.
- *Asymmetrische Verschlüsselung und Schlüsselaustausch*: Hier lauteten die Sieger PSE-KEM (eine Variante von Diffie-Hellman), RSA-KEM (eine RSA-Variante) und ACE-KEM (ein ECC-Verfahren).
- *Signaturverfahren*: Hier gewannen RSA-PSS (eine RSA-Variante), ECDSA und SFLASH.
- *Authentifikation mit asymmetrischen Verfahren*: In dieser Kategorie gab es nur einen Teilnehmer. Dieser wurde zum Sieger erklärt. Es handelt sich dabei um ein Verfahren auf Basis des diskreten Logarithmus mit dem Namen GPS (nicht zu verwechseln mit dem gleichnamigen Global Positioning System).

Der NESSIE-Wettbewerb sollte ein europäisches Gegengewicht zum amerikanischen AES-Wettbewerb setzen. Im Gegensatz zum Vorbild war ein Sieg jedoch nicht mit einer Deklarierung als Standard verbunden. Dies war wohl der Hauptgrund, warum NESSIE ein deutlich geringeres Interesse erfuhr als der AES-Wettbewerb.

eSTREAM

Nachdem die Kategorie Stromchiffren des NESSIE-Wettbewerbs ohne Sieger geblieben war, startete die von der EU geförderte Organisation ECRYPT im Jahr 2004 einen neuen Wettbewerb namens **eSTREAM**, in dem nur Stromchiffren betrachtet wurden. Es gab 34 Einreichungen. Das Auswahlverfahren endete 2008

mit zunächst acht Empfehlungen. Ein Verfahren musste aufgrund nachträglich entdeckter Schwächen von der Liste gestrichen werden, weshalb man nun von sieben eSTREAM-Siegern sprechen kann. Details finden sich in Abschnitt 16.6.

CRYPTREC

CRYPTREC (Cryptography Research and Evaluation Committee) ist der Name eines japanischen Komitees, das etwa zeitgleich zu NESSIE empfehlenswerte Krypto-Verfahren in verschiedenen Kategorien auswählte. Der Auswahlprozess endete 2002 [CRYPTR]. Das Komitee besteht immer noch, hat jedoch seitdem keine neue Auswahl mehr durchgeführt. Die folgende Liste nennt die Kategorien und die jeweils empfohlenen Verfahren:

- *Authentifizierung*: Hier gab es keine Empfehlung.
- *Digitale Signatur*: Hier stehen DSA, ECDSA sowie die RSA-Varianten RSA-PSS und RSASSA-PKCS1 v1.5 auf der Liste der CRYPTREC-Empfehlungen.
- *Asymmetrische Verschlüsselung*: Die beiden empfohlenen Verfahren sind die RSA-Varianten RSA-OAEP und RSAES-PKCS1 v1.5.
- *Asymmetrischer Schlüsselaustausch*: Auf der Liste stehen Diffie-Hellman, ECDH und PSEC-KEM. Letzteres ist ein Verfahren auf Basis elliptischer Kurven.
- *Blockchiffren mit 64 Bit Blocklänge*: Hier werden CIPHERUNICORN-E, Hierocrypt-L1, MISTY1 und Triple-DES (mit drei unterschiedlichen Schlüsseln) empfohlen. Abgesehen von Triple-DES stammen diese Verfahren aus Japan.
- *Blockchiffren mit 128 Bit Blocklänge*: Die empfohlenen Verfahren sind AES, Camellia, CIPHERUNICORN-A, Hierocrypt-3 und SC2000. Die drei letztgenannten Algorithmen haben nur in Japan eine größere Verbreitung.
- *Stromchiffren*: Auf der Empfehlungsliste stehen MUGI, MULTI-S01 und RC4 (mit 128-Bit-Schlüssel). Die beiden erstgenannten Verfahren stammen aus Japan und sind in anderen Ländern kaum verbreitet.
- *Kryptografische Hashfunktionen*: RIPEMD-160, SHA-1, SHA-256, SHA-384 und SHA-512 sind die empfohlenen Verfahren.
- *Kryptografische Pseudozufallsgeneratoren*: Hier gab es keine expliziten Empfehlungen, sondern nur eine exemplarische Auflistung einiger als sicher geltender Verfahren. Die Liste enthält *ANSI X9.42-2001 Annex C.1*, *FIPS 186-2 (+ change notice 1) Appendix 3.1* und *FIPS 186-2 (+ change notice 1) revised Appendix 3.1*. Alle drei Verfahren werden in Abschnitt 15.2 beschrieben.

19 Betriebsarten und Datenformatierung

Mit den Krypto-Verfahren, die ich im zweiten Teil dieses Buchs vorgestellt habe, stehen Alice und Bob wirkungsvolle Werkzeuge zur Verfügung. Allerdings kommt es auch in der Kryptografie – wie überall im Leben – darauf an, das jeweilige Werkzeug richtig einzusetzen. In diesem Kapitel werden wir uns daher die Frage stellen, welche Sicherheitslücken und sonstige Nachteile sich ergeben, wenn Alice und Bob kryptografische Verfahren auf die jeweils naheliegendste Weise einsetzen. Und wir werden die weniger naheliegenden Einsatzalternativen betrachten. Dabei werden Sie sehen, dass sich bei symmetrischen Verfahren ganz andere Probleme und Lösungen ergeben als bei asymmetrischen.

19.1 Betriebsarten von Blockchiffren

Das Verschlüsseln einer Nachricht mit einem symmetrischen Verfahren (beispielsweise dem AES) gehört für Alice und Bob zu den einfachsten Übungen. Doch selbst diesen simplen Vorgang gibt es in mehreren Varianten. Eine solche Variante wird als **Betriebsart** bezeichnet. Die fünf wichtigsten Betriebsarten für symmetrische Verfahren schauen wir uns im Folgenden an. Diese Betrachtungen gelten allerdings zunächst nur für Blockchiffren. Für Stromchiffren gibt es deutlich weniger Variationsmöglichkeiten, worauf ich am Ende dieses Unterkapitels kurz eingehen werde. Wir gehen in den folgenden Betrachtungen immer von einer Blockchiffre aus, die Blöcke der Länge 128 Bit verarbeitet (der AES ist ein Beispiel dafür). Alle Überlegungen lassen sich problemlos auf andere Blocklängen übertragen.

19.1.1 Electronic-Codebook-Modus

Bei der Verwendung symmetrischer Verschlüsselungsverfahren sind wir bisher von folgender Verwendungsweise ausgegangen: Wenn Alice eine Nachricht an Bob verschlüsselt, dann teilt sie diese in Blöcke der Länge 128 Bit auf und wendet das Verfahren auf jeden Block einzeln an. Diese Methode ist die einfachste aller Betriebsarten und wird **Electronic-Codebook-Modus** (ECB) genannt. Der Name rührt daher, dass Alice theoretisch für jeden Schlüssel ein Codebuch (siehe Abschnitt 4.2.5) anfertigen könnte, in dem für jeden möglichen Klartextblock (davon gibt es 2^{128}) der zugehörige Geheimtextblock aufgeführt ist. Dies ist (in der Theorie) möglich, weil ein Verschlüsselungsverfahren im ECB-Modus monoalphabetisch ist. Als mathematische Formel geschrieben sieht der ECB-Modus wie folgt aus (i ist die Blocknummer, m_i der Klartext, c_i der Geheimtext, k der Schlüssel und E die Verschlüsselungsfunktion):

$$c_i = E_k(m_i)$$

Der ECB-Modus ist zwar die einfachste, aber sicherlich nicht die beste Betriebsart. Die Tatsache, dass der ECB-Modus theoretisch eine Häufigkeitsanalyse erlaubt (wie jede monoalphabetische Chiffre), ist einer davon. Allerdings gibt es bei 128 Bit Blocklänge schlichtweg zu viele Geheimtexte, um eine praktisch verwertbare Statistik anzufertigen. Sehr wohl von praktischer Relevanz sind dagegen folgende Nachteile:

- Mallory kann bei einer ECB-verschlüsselten Nachricht Blöcke herausnehmen oder die Reihenfolge verändern, ohne dass Alice es merkt. Er kann auch die Blöcke verschiedener Nachrichten zusammenwürfeln. Je nach Anwendung können sich dadurch merkwürdige Effekte ergeben.
- Gleiche Klartextblöcke ergeben im ECB-Modus bei gleichem Schlüssel stets gleiche Geheimtextblöcke. Dies ist beispielsweise ein Nachteil, wenn (wie bei den Betriebssystemen Unix oder Windows) Passwortlisten verschlüsselt wer-

den. Wenn Alices verschlüsseltes Passwort gleich aussieht wie das eines anderen Anwenders, dann weiß sie, dass dieser das gleiche Passwort verwendet.
- Der ECB-Modus kennt nur eine Blocklänge (in diesem Fall 128 Bit). Wenn die Daten, die Alice verschlüsseln will, in kleineren Einheiten eintreffen (etwa byteweise), muss Alice entweder warten, bis sich ein 128-Bit-Puffer gefüllt hat, oder sie muss den Puffer mit bedeutungslosem Material auffüllen.
- Ein falsches Bit im Geheimtext (ein solches kann in der Praxis durchaus vorkommen) führt im ECB-Modus dazu, dass Empfänger Bob den gesamten Geheimtextblock nicht korrekt entschlüsseln kann. Wir werden sehen, dass andere Betriebsarten weniger anfällig gegenüber Geheimtextfehlern sind.

Aufgrund dieser Nachteile kommt der ECB-Modus in der Praxis meist nur für das Verschlüsseln von Zufallszahlen (die als Schlüssel verwendet werden) zum Einsatz.

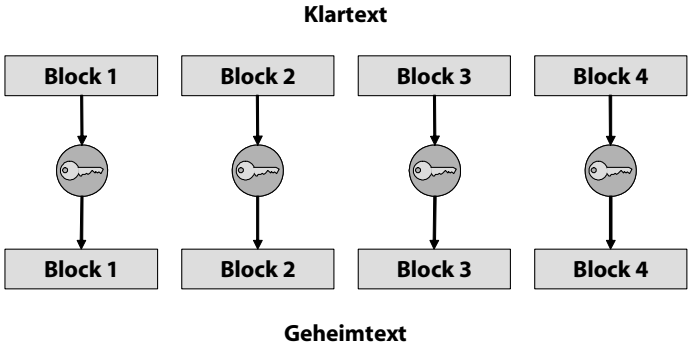


Abb. 19–1 *Im ECB-Modus wird jeder Klartextblock unabhängig von den anderen verschlüsselt. ECB ist damit die einfachste Betriebsart.*

19.1.2 Cipher-Block-Chaining-Modus

Ein Problem beim ECB-Modus besteht darin, dass der gleiche Klartextblock mit dem gleichen Schlüssel stets den gleichen Geheimtextblock ergibt. Dies lässt sich dadurch verhindern, dass Alice den Klartextblock vor dem Verschlüsseln mit dem letzten Geheimtextblock exklusiv-oder-verknüpft. Diese Betriebsart wird **Cipher-Block-Chaining-Modus** (CBC) genannt. Bei der Entschlüsselung muss Bob entsprechend nach dem Entschlüsseln eines Blocks den vorhergehenden Geheimtextblock abziehen. Ein Verschlüsselungsverfahren im CBC-Modus ist polyalphabetisch und hat folgende mathematische Formel:

$$c_i = E_k(m_i \oplus c_{i-1})$$

Verschlüsselt Alice ihre Daten im CBC-Modus, dann können in ihrer Nachricht beliebig viele inhaltsgleiche Blöcke auftreten, ohne dass dies Mallory bei Betrachtung des Geheimtexts auffällt. Damit Alice auch den ersten Klartextblock einer

Nachricht mit einem Geheimtextblock verknüpfen kann, sieht der CBC-Modus zusätzlich einen Dummyblock vor, der als Erstes verschlüsselt wird. Dieser Dummyblock wird als **Initialisierungsvektor** (**IV**) bezeichnet. Alice und Bob müssen ihn nicht geheim halten, sie sollten einen solchen jedoch nicht mehrfach verwenden.

Die Anfälligkeit gegenüber Fehlern im Geheimtext ist im CBC-Modus noch etwas höher als im ECB-Modus. Bei einem falschen Geheimtext-Bit erhält Bob nach dem EBC-Entschlüsseln einen zerstörten Klartextblock und zusätzlich ein falsches Klartext-Bit. Der CBC-Modus wird vor allem zum Verschlüsseln von Dateien verwendet.

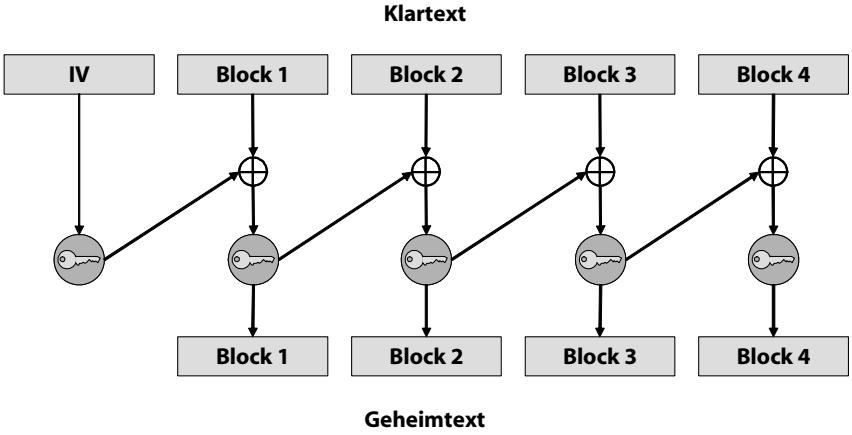


Abb. 19–2 *Im CBC-Modus wird jeder Klartextblock mit dem letzten Geheimtextblock verknüpft.*

19.1.3 Output-Feedback-Modus

Ein anderer Ansatz zur Nutzung einer Blockchiffre besteht darin, sie als Stromchiffre zu verwenden. Diese Betriebsart wird als **Output-Feedback-Modus** (**OFB**) bezeichnet. Der Verschlüsselungsalgorithmus dient beim OFB als Fortschaltfunktion der Stromchiffre. Alice benötigt für eine OFB-Verschlüsselung zunächst einen Startwert (Initialisierungsvektor) s_0 in Form eines 128-Bit-Blocks. Sie verschlüsselt s_0 und exklusiv-oder-verknüpft das Ergebnis s_1 anschließend mit dem ersten Klartextblock. Dann verschlüsselt sie s_1 und verknüpft das Ergebnis s_2 mit dem zweiten Block und so weiter. Der Klartext beeinflusst hierbei die Verschlüsselung mit dem jeweiligen Algorithmus nicht, sondern wird immer nur mit dem Ergebnis dieser Verschlüsselung verknüpft. Mathematisch ausgedrückt sieht dies wie folgt aus:

$$s_i = E_k(s_{i-1})$$
$$c_i = s_i \oplus m_i$$

Die CBC-Entschlüsselung entsteht durch eine erneute Anwendung derselben Formel. Der Algorithmus wird dabei nie im Entschlüsselungsmodus betrieben. Der OFB-Modus hat einige Vorteile:

- OFB ist sehr schnell, wenn Alice die Verschlüsselung von s_i im Voraus berechnet. In diesem Fall besteht die Verschlüsselung des Klartexts nur noch aus einer Exklusiv-oder-Verknüpfung.
- Alice und Bob können im OFB-Modus auch kurze Blöcke verschlüsseln, ohne diese auf 128 Bit auffüllen zu müssen. Insbesondere können sie ihre Nachrichten byteweise verarbeiten, indem sie stets nur das erste Byte der Ausgabe des Verschlüsselungsverfahrens verwenden. Im ECB- oder CBC-Modus müssten die beiden dagegen für jedes übertragene Byte 15 Bytes an Füllmaterial generieren. Die übertragene Datenmenge würde sich dadurch versechzehnfachen.

Der OFB-Modus eignet sich vor allem für Anwendungsfälle, in denen der Klartext in kleinen Teilen (etwa byteweise) eintrifft und Fehler zu erwarten sind. Dies ist bei vielen Kommunikationsverbindungen der Fall.

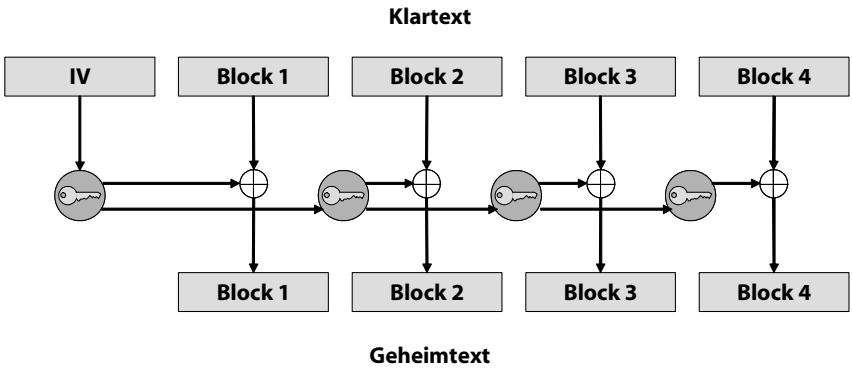


Abb. 19–3 *Im OFB-Modus wird eine Blockchiffre als Stromchiffre betrieben.*

19.1.4 Cipher-Feedback-Modus

Als nächste Betriebsart betrachten wir den **Cipher-Feedback-Modus** (**CFB**). Dieser sieht vor, dass Alice zur Verschlüsselung eines Klartextblocks zunächst den letzten Geheimtextblock mit dem Algorithmus verschlüsselt und das Ergebnis mit dem Klartextblock exklusiv-oder-verknüpft. Mathematisch sieht dies wie folgt aus:

$$c_i = E_k(c_{i-1}) \oplus m_i$$

Der CFB-Modus ähnelt in einigen Punkten dem OFB-Modus und benötigt wie dieser einen Initialisierungsvektor s_0. Es gibt weitere Parallelen zum OFB-Modus: So ist ein Verschlüsselungsverfahren im CFB-Modus polyalphabetisch. Außerdem ermöglicht CFB (wie OFB) eine Vorausberechnung des kryptografisch rele-

vanten Teils einer Verschlüsselung (danach fehlt nur noch eine Exklusiv-oder-Verknüpfung mit dem Klartext).

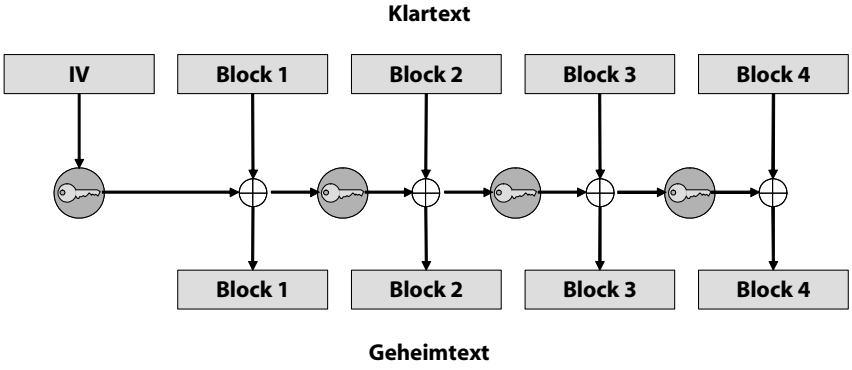


Abb. 19–4 *Im CFB-Modus wird die Verknüpfung aus Klartextblock und einem Zwischenergebnis zum Geheimtextblock und zum Zwischenergebnis für den neuen Block.*

Der wesentliche Unterschied zwischen OFB und CFB besteht darin, dass beim CFB-Modus jeder Geheimtextblock von den vorhergehenden abhängig ist, beim OFB-Modus dagegen nicht. Aus diesem Unterschied ergibt sich sowohl der wichtigste Vorteil als auch der wichtigste Nachteil von CFB gegenüber OFB.

Der wichtigste Vorteil: Beim CFB-Modus kann Bob jeden beliebigen Geheimtextblock einer Nachricht entschlüsseln, sofern er den vorhergehenden Geheimtextblock kennt. Dies bedeutet, dass Bob bei der Entschlüsselung wahlfreien Zugriff auf jeden Block hat. Beim OFB-Modus muss Bob zur Entschlüsselung eines Blocks dagegen für jeden vorhergehenden Block der Nachricht eine Verschlüsselung der Zustandsvariablen durchführen.

Der wichtigste Nachteil: Ein falsches Bit im Geheimtext richtet im CFB-Modus mehr Schaden an als im OFB-Modus. Neben einem falschen Bit im Klartext ergibt sich in diesem Fall ein kompletter Block, der zerstört wird. CFB kommt aus diesem Grund vor allem in Kommunikationsverbindungen zum Einsatz, in denen Übertragungsfehler selten sind. Ansonsten ist der Einsatzbereich ähnlich wie beim OFB-Modus.

Der CFB-Modus lässt sich nach folgendem Schema abwandeln (n sei die Länge eines Nachrichtenblocks m_i; n sei kleiner als die Blocklänge; s_i sei ein 128-Bit-Wert): Alice verschlüsselt zunächst den Initialisierungsvektor s_0 und exklusiv-oder-verknüpft das Ergebnis mit dem n Bit langen Klartext. Dann generiert sie s_1, indem sie den Inhalt von s_0 um n Bit nach rechts schiebt (wobei die n rechtesten Bits gelöscht werden) und die freigewordenen n linkesten Bit mit dem ersten Geheimtextblock auffüllt. Der zweite Geheimtextblock entsteht durch die Verschlüsselung von s_1 und einer anschließenden Exklusiv-oder-Verknüpfung mit dem zweiten Klartextblock. s_2 berechnet sich aus s_1 wie s_1 aus s_0.

19.1.5 Counter-Modus

Der **Counter-Modus** (**CTR**) ist der Shooting-Star unter den Betriebsarten. Es gibt ihn zwar schon seit den siebziger Jahren, doch erst in den letzten Jahren ist er richtig populär geworden und hat Einzug in verschiedene Standards gehalten. Viele Kryptografen geben dem Counter-Modus heute den Vorzug gegenüber den anderen vier beschriebenen Betriebsarten. Der Counter-Modus funktioniert in der einfachsten Form so (die Variable z_i wird Zähler genannt):

1. Alice nimmt einen Initialisierungsvektor z_0 und verschlüsselt ihn. Das Ergebnis der Verschlüsselung addiert sie zum ersten Klartextblock (Exklusiv-oder-Verknüpfung) und erhält so den ersten Geheimtextblock.
2. Alice verschlüsselt $z_1 = z_0 + 1$, addiert das Ergebnis zum zweiten Klartextblock (Exklusiv-oder-Verknüpfung) und erhält so den zweiten Geheimtextblock.
3. Alice verschlüsselt $z_2 = z_1 + 1$, addiert das Ergebnis zum dritten Klartextblock (Exklusiv-oder-Verknüpfung) und erhält so den dritten Geheimtextblock.
4. usw.

Mit anderen Worten: Alice zählt den Zähler hoch, verschlüsselt ihn jeweils und verknüpft das Ergebnis mit dem Klartext. Anstatt den Zähler jeweils um 1 zu erhöhen, kann Alice auch eine andere Schrittweite wählen (oder ein anderes Rechenverfahren, beispielsweise eine Multiplikation). Alice und Bob müssen den Zähler nicht geheim halten. Wichtig ist dagegen, dass die beiden bei gleichem Schlüssel nie einen Zähler doppelt verwenden, da Mallory ansonsten durch eine Exklusiv-oder-Verknüpfung der beiden Geheimtextblöcke eine Exklusiv-oder-Verknüpfung der Klartextblöcke erhält (ähnlich wie bei einer Stromchiffre, bei der kein Schlüssel doppelt zum Einsatz kommen darf).

In der Praxis wird meist eine Variation des Counter-Modus eingesetzt, die einen dreiteiligen Initialisierungsvektor vorsieht. Ist der Zähler eine 128-Bit-Zahl, dann besteht der Initialisierungsvektor beispielsweise aus einer Nachrichtennummer (56 Bit), einem Zufallswert (16 Bit) und 56 Null-Bits. Die Schrittweite beim Hochzählen beträgt 1. In dieser Konfiguration können Alice und Bob 2^{56} Nachrichten verschlüsseln, die jeweils bis zu 2^{56} Blöcke lang sind, ohne dass sich ein Zähler wiederholt. Dies sollte für die Praxis (und bis zum nächsten Schlüsselwechsel) ausreichen. Der CTR-Modus hat viele Vorteile: Er ist auch für kleine Blocklängen geeignet und benötigt nur die Verschlüsselungsfunktion (nicht aber die Entschlüsselungsfunktion) des verwendeten Algorithmus. Im Gegensatz zu CBC, OFB und CFB lässt sich CTR parallelisieren, und jeder Block lässt sich unabhängig von den anderen ver- bzw. entschlüsseln. So gesehen ist es offensichtlich, warum der CTR-Modus bei Experten so beliebt ist. Der einzige Nachteil: Alice und Bob müssen besonders aufpassen, dass sie keinen Zähler doppelt verwenden.

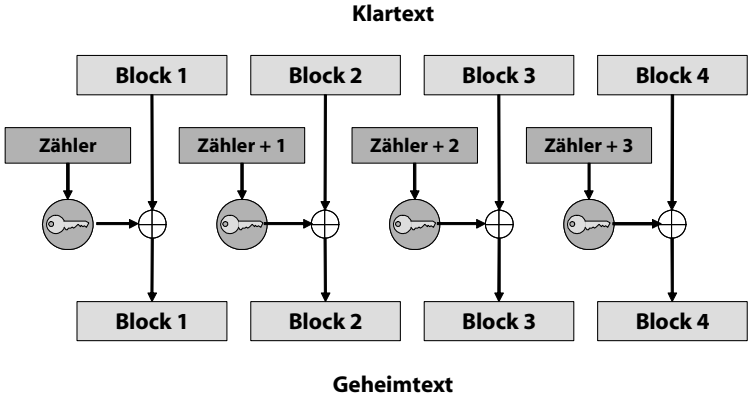


Abb. 19–5 *Der Counter-Modus ist der Shooting-Star unter den Betriebsarten.*

19.1.6 Fazit

Es gibt noch weitere Betriebsarten für Blockchiffren. Insbesondere ist auch die Nutzung einer Blockchiffre als kryptografische Hashfunktion eine Betriebsart (siehe Abschnitt 14.4.4). Oder genauer ausgedrückt: Es gibt mehrere Blockchiffren-Betriebsarten, die nicht dem Verschlüsseln, sondern dem Generieren eines Hashwerts dienen. Wenn wir uns allerdings auf die Verschlüsselung beschränken, dann sind ECB, CBC, OFB, CFB und CTR mit Abstand die wichtigsten. Die folgende Tabelle fasst deren Eigenschaften noch einmal zusammen:

	ECB	CBC	OFB	CFB	CTR
Formel	$c_i=E(k,m_i)$	$c_i=E(k,m_i)\oplus c_{i-1}$	$s_i=E(k,s_{i-1})$ $c_i=s_i\oplus m_i$	$c_i=E(k,c_{i-1})\oplus m_i$	$c_i=E(z_i)\oplus m_i$ $z_{i+i}=z_i+1$
mono- / polyalphabetisch	monoalphabetisch	polyalphabetisch	polyalphabetisch	polyalphabetisch	polyalphabetisch
Entschlüsselungsfunktion	notwendig	notwendig	nicht notwendig	nicht notwendig	nicht notwendig
Wahlfreier Zugriff auf einzelne Blöcke	möglich	nicht möglich	nicht möglich	bei Entschlüsselung möglich	möglich
Vorberechnung	nicht möglich	nicht möglich	möglich	möglich	möglich
Folge eines falschen Bits im Geheimtext	ein Block zerstört	ein Block + ein Bit zerstört	ein Bit zerstört	ein Block + ein Bit zerstört	ein Block zerstört

→

	ECB	CBC	OFB	CFB	CTR
Blocklänge	Blocklänge des Algorithmus	Blocklänge des Algorithmus	Blocklänge des Algorithmus oder kürzer	Blocklänge des Algorithmus oder kürzer	Blocklänge des Algorithmus oder kürzer
Spezielle Gefahren			Gefahr bei Mehrfachverwendung eines Initialisierungsvektors		Gefahr bei Mehrfachverwendung eines Zählers

Für die Entwickler von Netzwerkprotokollen haben die unterschiedlichen Betriebsarten zur Folge, dass sie nicht nur das verwendete Verfahren, sondern auch die Betriebsart festlegen müssen. Aus diesem Grund gibt es für alle gängigen Blockchiffren mehrere OIDs – jeweils eine für eine bestimmte Betriebsart. Bei CBC, CFB, OFB und CTR ist mit der OID jeweils auch die Generierung des Startwerts festgelegt. Weitere Betrachtungen zum Thema Betriebsarten finden Sie in [FeScKo]. Für Stromchiffren spielen Betriebsarten übrigens keine wesentliche Rolle. Stromchiffren arbeiten in der Regel per Design im OFB-Modus, manche könnte man auch im CFB-Modus einsetzen. Für CTR gibt es einige speziell entwickelte Stromchiffren. ECB und CBC haben dagegen für eine Stromchiffre keinen Sinn.

19.2 Betriebsarten von Tweak-Verfahren

In ein Tweak-Verschlüsselungsverfahren geht neben dem Schlüssel noch eine weitere Information in den Verschlüsselungsvorgang ein, die Tweak genannt wird (siehe Abschnitt 10.4). Alice und Bob müssen den Tweak nicht geheim halten. Welche Vorteile Tweak-Verfahren haben, sieht man unter anderem, wenn man sie in den Betriebsarten einsetzt, die Sie in Abschnitt 19.1 kennengelernt haben. Dies sieht wie folgt aus:

- *ECB*: Ein Verschlüsselungsverfahren ohne Tweak liefert im ECB-Modus bei gleichem Klartext und gleichem Schlüssel immer den gleichen Geheimtext. Bei einem Tweak-Verfahren ist das nicht der Fall, sofern der Tweak sich ändert.
- *CBC*: Der CBC-Modus lässt sich bei einem Tweak-Verfahren realisieren, indem Alice und Bob den vorhergehenden Geheimtextblock jeweils als Tweak verwenden, anstatt ihn per Exklusiv-oder-Verknüpfung einzubringen. Die unerwünschte Eigenschaft des CBC, dass sich bei einer Manipulation einzelner Bits die folgenden Blöcke nur geringfügig und in vorhersehbarer Weise ändern, fällt dadurch weg.
- *OFB*: Im OFB-Modus ist ein Tweak meist unnötig, da es bereits einen Initialisierungsvektor gibt.

- *CFB*: Auch im CFB-Modus bringt ein Tweak-Verfahren keine besonderen Vorteile.
- *CTR*: Wenn Alice und Bob den Zähler als Tweak verwenden, ist dies eine elegante Möglichkeit, einen CTR-Modus zu realisieren.

Tweak-Verfahren sind noch nicht allzu verbreitet, daher werden auch die entsprechenden Betriebsarten kaum eingesetzt. Eine wichtigere Rolle spielen Tweak-Verfahren, wenn es um die Nutzung als kryptografische Hashfunktion geht (Abschnitt 14.4.5).

19.3 Formaterhaltende Verschlüsselung

Alice möchte eine vierstellige Geheimnummer (zum Beispiel 0815) verschlüsselt an Bob schicken. Dazu kann sie die Geheimnumer mit dem AES verschlüsseln und erhält dann einen 128-Bit-Block als Geheimtext. Die Software, die Alice und Bob verwenden, lässt jedoch nur eine vierstellige Nummer als Eingabe zu. Die beiden benötigen daher eine Möglichkeit, um einen Vier-Ziffern-Block in einen Vier-Ziffern-Block zu verschlüsseln. Ein solches Szenario, in dem der Geheimtext den gleichen Aufbau haben muss wie der Klartext, bezeichnet man als **formaterhaltende Verschlüsselung**.

Anstatt ein spezielles Verfahren für die formaterhaltende Verschlüsselung zu entwickeln, können Alice und Bob auch einen vorhandenen Algorithmus (wir gehen vom AES aus) verwenden. Im eingangs beschriebenen Beispiel könnte das so aussehen (wir gehen davon aus, dass Alice und Bob einen gemeinsamen AES-Schlüssel besitzen):

1. Alice verschlüsselt die Zahlen von 0000 bis 9999 jeweils mit dem AES (sie verwendet dazu den Schlüssel, der auch Bob bekannt ist). Dadurch erhält sie 10.000 Geheimtexte, die jeweils 128 Bit lang sind.
2. Alice sortiert die 10.000 Geheimtexte der Größe nach (jeder Geheimtext wird dabei als Binärzahl interpretiert). Wenn nun beispielsweise der Geheimtext von 0815 an 4711. Stelle steht, dann heißt dies: 0815 wird in 4711 verschlüsselt.
3. Alice schickt die Zahl 4711 an Bob.
4. Bob verschlüsselt seinerseits die Zahlen von 0000 bis 9999 mit dem AES und sortiert die Ergebnisse. Dadurch erkennt er, dass an 4711. Stelle der Geheimtext von 0815 steht. Dies ist die vierstellige Zahl, die Alice übermittelt hat.

Neben dieser einfachen (aber auch recht aufwendigen) Methode gibt es zahlreiche weitere. Als weiterführende Literatur empfehle ich [BeRiRT].

19.4 Datenformatierung für das RSA-Verfahren

Auch für die asymmetrische Kryptografie gilt: Es gibt stets verschiedene Möglichkeiten, ein Verfahren einzusetzen, und die naheliegendste Methode muss nicht immer die beste sein. Betriebsarten, wie wir sie gerade bei den Blockchiffren betrachtet haben, spielen bei asymmetrischen Verfahren allerdings keine Rolle. Die unterschiedlichen Einsatzmöglichkeiten von RSA und DLSSs ergeben sich stattdessen aus der Art, wie Absenderin Alice die jeweiligen Daten vor der Verschlüsselung oder Signierung bearbeitet. Man nennt dies auch **Datenformatierung**.

Wie Sie gleich sehen werden, spielt die Datenformatierung vor allem für das RSA-Verfahren eine wichtige Rolle. Verfahren auf Basis des diskreten Logarithmus benötigen dagegen so gut wie keine vorhergehende Bearbeitung. Die in der Praxis am häufigsten eingesetzte RSA-Datenformatierung ist im **PKCS#1-Standard** festgelegt [PKCS#1]. Auch der P1363-Standard behandelt das Thema Datenformatierung für asymmetrische Verfahren [P1363].

19.4.1 Der PKCS#1-Standard

PKCS#1 gehört zu den PKCS-Standards, von denen bereits in Abschnitt 18.3.1 die Rede war. PKCS#1 ist eine Spezifikation zur Implementierung des RSA-Verfahrens. Darin wird zunächst festgelegt, wie eine Folge von Bytes als natürliche Zahl interpretiert wird. Diese Festlegung wird benötigt, da Klartext und Geheimtext beim RSA-Verfahren stets als natürliche Zahlen vorliegen müssen. Anschließend beschreibt PKCS#1 den bekannten RSA-Ablauf inklusive möglicher Fehlermeldungen (beispielsweise »message representative out of range«, wenn der Klartext zu lang ist). Darüber hinaus enthält PKCS#1 noch Formate für öffentliche und private RSA-Schlüssel sowie verschiedene OIDs.

Der interessanteste Bestandteil von PKCS#1 sind zweifellos die darin beschriebenen Methoden für die Datenformatierung. Diese wollen wir im Folgenden näher betrachten. Insgesamt gibt es vier Datenformatierungsverfahren in PKCS#1: In Version 1 des Standards wird jeweils ein Verfahren für digitale Signaturen und eines für die Verschlüsselung beschrieben. In Version 2 kommt für beide Anwendungsfälle jeweils ein neues hinzu, welches das in Version 1 vorhandene Verfahren ersetzen soll. Aus Kompatibilitätsgründen sind jedoch auch die alten Formatierungsverfahren noch im Standard enthalten.

19.4.2 Datenformatierung für die RSA-Verschlüsselung

Wenn Alice einen Klartext (zum Beispiel den Schlüssel für ein symmetrisches Verfahren) RSA-verschlüsseln will, dann kann sie sich die Sache einfach machen und den Klartext ohne Änderungen verschlüsseln. Dabei tauchen allerdings einige Probleme auf:

- Wenn der Klartext ein Zufallswert ist (bei einem geheimen Schlüssel ist das ja der Fall), dann kann Bob nicht feststellen, ob das Ergebnis seiner Entschlüsselung auch wirklich dem ursprünglichen Klartext entspricht. Wenn Bob etwa versehentlich einen falschen privaten RSA-Schlüssel zum Entschlüsseln verwendet, dann erhält er ein falsches Ergebnis, ohne es zu merken. Gleiches passiert, wenn der Geheimtext auf dem Transportweg verändert wurde. Es lohnt sich für Alice also, den RSA-Klartext im Rahmen der Datenformatierung so zu kennzeichnen, dass Bob ihn von einer zufälligen Bit-Folge unterscheiden kann. Wenn wir von einem Hardwaremodul (beispielsweise einer Smartcard) ausgehen, das RSA-Entschlüsselungen durchführen kann, dann bringt eine solche Vorgehensweise einen zusätzlichen Vorteil mit sich. Der Hersteller kann dieses Modul so programmieren, dass es ungültige RSA-Klartexte anhand der fehlenden Formatierung als solche erkennt und nicht ausgibt. Dies erschwert Mallory eine Chosen-Ciphertext-Attacke erheblich.
- Wie Sie aus Abschnitt 11.3.1 wissen, geht das RSA-Verschlüsseln besonders schnell, wenn Alice einen kleinen Exponenten verwendet (die Zahlen 3 und 17 sind besonders gut geeignet). Verschlüsselt Alice jedoch denselben Text mit einem kleinen Exponenten mehrfach (mit mehreren unterschiedlichen öffentlichen Schlüsseln), dann kann Mallory eine Low-Exponent-Attacke starten. Dies wird verhindert, wenn Alice im Rahmen der Datenformatierung dafür sorgt, dass jeder RSA-Klartext anders aussieht (dies ist etwa der Fall, wenn Alice eine Zufallszahl hinzufügt).
- Aus Abschnitt 12.2 wissen Sie sicherlich noch, dass eine RSA-Entschlüsselung und eine RSA-Signierung mathematisch gesehen das Gleiche sind. Dies kann Mallory ausnutzen, indem er Alice beispielsweise einen RSA-Geheimtext zum Signieren vorlegt und so den Klartext erhält (Unterschiebe-Angriff). Solche Angriffe können Alice und Bob verhindern, indem sie RSA-Klartexte so formatieren, dass sie von einem zu signierenden Hashwert zu unterscheiden sind.
- In manchen Einsatzszenarien ist es für Mallory von Vorteil, wenn ein RSA-Klartext bestimmte Bit-Muster enthält oder eine bestimmte Länge hat. So kann etwa ein Seitenkanalangriff unter solchen Voraussetzungen einfacher sein. Eine Datenformatierung sollte daher für eine einheitliche Länge des Klartexts sorgen und möglichst wenig Regelmäßigkeiten darin zulassen.

Es lohnt sich also für Alice, die Daten, die sie mit dem RSA-Verfahren verschlüsselt, vorher zu formatieren.

Formatierung für die Verschlüsselung nach PKCS#1 Version 1

Die erste Version des PKCS#1-Standards (PKCS#1v1) sieht eine Datenformatierung für die RSA-Verschlüsselung vor, die die genannten Probleme wenigstens teilweise löst. Der formatierte Klartext hat gemäß PKCS#1v1 folgende vier Bestandteile:

1. Das erste Byte des formatierten Klartexts hat immer den Wert 00000010. Dies ist ein Erkennungszeichen, an dem Bob nach dem Entschlüsseln feststellen kann, ob es sich um einen gültigen Klartext handelt.
2. Nach dem ersten Byte folgen beliebig viele Zufallsbytes, die ungleich Null sind. Diese Zufallsbytes sollten von Alice für jeden Verschlüsselungsvorgang anders gewählt werden. Dadurch wird sichergestellt, dass sich die formatierten Klartexte auch dann unterscheiden, wenn die ursprünglichen Klartexte identisch sind. Außerdem bringen die Zufallsbytes den Klartext auf eine einheitliche Länge und enthalten keine Muster.
3. Nach den Zufallsbytes folgt ein Byte mit dem Wert 00000000. Dieses Byte ist ein weiteres Erkennungszeichen für einen gültigen Klartext und trennt die Zufallsbytes vom folgenden Teil.
4. Als Letztes kommt der ursprüngliche Klartext.

Hat Empfänger Bob die Nachricht von Alice entschlüsselt, dann prüft er zunächst nach, ob ein Byte mit dem Wert 00000010 am Anfang steht. Ist dies der Fall, dann weiß er, dass es sich mit hoher Wahrscheinlichkeit um einen formatierten Klartext handelt. Anschließend sucht er nach dem ersten Byte mit dem Wert 00000000. Alles, was danach kommt, ist der ursprüngliche Klartext.

1998 entdeckte der Kryptograf Daniel Bleichenbacher eine Schwachstelle in der Datenformatierung gemäß PKCS#1 Version 1 [Bleich]. Mallory kann diese Schwachstelle zu einem Angriff nutzen, den man zur Familie der Seitenkanalangriffe zählt (siehe Abschnitt 17.1). Unter bestimmten Umständen kann Mallory damit eine Nachricht entschlüsseln, die Bob mit Alices öffentlichem Schlüssel verschlüsselt hat. Der genannte Seitenkanalangriff sieht vor, dass Mallory die von Bob verschlüsselte Nachricht nach einer bestimmten Methode verändert und damit insgesamt etwa eine Million veränderter Nachrichten erzeugt. Diese Nachrichten sendet er an Alice. Wenn Alice jede der Nachrichten entschlüsselt und Mallory in jedem Fall erfährt, ob das Format der entschlüsselten Nachricht korrekt ist, dann kann dieser mit dem von Bleichenbacher beschriebenen Verfahren den echten Klartext rekonstruieren. Natürlich ist dieser Angriff nicht immer realistisch. Außerdem erhält Mallory dabei nur den Klartext, nicht aber den Schlüssel. Dennoch war diese Schwachstelle Grund genug, den PKCS#1-Standard um eine verbesserte Datenformatierung zu erweitern.

Formatierung für die Verschlüsselung nach PKCS#1 Version 2

Die Datenformatierung nach PKCS#1v1 hat zwei Nachteile:

- Jeder PKCS#1v1-formatierte Klartext, der irgendwo auf der Welt verschlüsselt wird, beginnt mit einem Byte des Inhalts 00000010. Außerdem kommt in jedem PKCS#1v1-Klartext ein Nullbyte vor. Dies sind zwei klare Verstöße gegen das Paradigma, Muster im Klartext zu vermeiden. Bisher hat zwar noch niemand einen Angriff gefunden, der diese Muster ausnutzt, doch unschön ist diese Sache allemal.

- Die Zufallsbytes und die ursprüngliche Nachricht sind frei wählbar. In manchen Einsatzszenarien kann Mallory dies dazu nutzen, für ihn vorteilhafte Inhalte in den RSA-Klartext zu bringen. Auch das widerspricht dem Ziel, Muster im Klartext zu vermeiden.

Um auch diese Schwächen zu beheben, führte Version 2 von PKCS#1 eine neue Formatierung für die RSA-Verschlüsselung ein. Diese wird als **OAEP-Kodierung** bezeichnet (die Abkürzung steht für »Optimal Asymmetric Encryption Padding«) und spielt auch im P1363-Standard eine Rolle [P1363]. Im PKCS#1v2-Standard wird empfohlen, zukünftig die OAEP-Kodierung zu implementieren, wobei eine PKCS#1-konforme Implementierung aber nach wie vor auch die herkömmliche Methode gemäß PKCS#1v1 unterstützen muss.

Die OAEP-Kodierung sieht vor, dass Alice ihre Zufallsbytes nicht einfach zwischen das Erkennungszeichen und den Klartext schreibt. Stattdessen nimmt sie die Zufallsbytes (im Normalfall 32 Bit) als Startwert für einen Pseudozufallsgenerator, der in der Regel mithilfe der ohnehin verwendeten kryptografischen Hashfunktion realisiert wird. Auf diese Weise generiert sie eine Zufallsfolge, die mit dem Klartext exklusiv-oder-verknüpft wird. Das Resultat wird maskierter Klartext genannt, der Vorgang wird als **Maskierung** bezeichnet. Der Startwert für den Zufallsgenerator wird dem maskierten Klartext vorangestellt. Dieser wird jedoch selbst auch maskiert, wobei der maskierte Klartext den Startwert für den Pseudozufallsgenerator bildet. Da Bob die Länge des maskierten Startwerts kennt, kann er auf einfache Weise erst den Startwert und dann den Klartext wiederherstellen. An einem Erkennungszeichen erkennt er, dass es sich um einen gültigen Klartext handelt.

Der Vorteil bei der OAEP-Kodierung liegt darin, dass Mallory (vorausgesetzt, die verwendete kryptografische Hashfunktion ist sicher) kein einziges Bit des formatierten Klartexts direkt beeinflussen kann – die Datenformatierung ist in diesem Fall also ein Zufallsorakel. Daher enthält ein formatierter PKCS#1v2-Klartext auch keinerlei Muster, die Mallory weiterhelfen könnten. Trotzdem kann Bob nach der Demaskierung anhand der Kodierung feststellen, ob es sich um einen gültigen Klartext handelt.

19.4.3 Datenformatierung für RSA-Signaturen

Auch bei RSA-Signaturen kann sich Alice die Sache einfach machen, indem sie den zu signierenden Text nimmt, darauf eine kryptografische Hashfunktion anwendet und den Hashwert unverändert signiert. Natürlich ist es dabei sinnvoll, wenn Alice an die Nachricht einige Zusatzinformationen (wie etwa die OID des verwendeten Verfahrens) hängt, die mit in den Hashwert eingehen. Derartige Zusätze sind beispielsweise in PKCS#7 standardisiert und sollen an dieser Stelle nicht interessieren. Hier interessiert uns vielmehr, was Alice mit dem Hashwert

macht. Wenn sie diesen unverändert signiert, dann ergeben sich wiederum verschiedene Probleme:

- Wenn der Hashwert der Nachricht nicht mit dem Ergebnis der Berechnung mit dem öffentlichen RSA-Schlüssel von Alice übereinstimmt, weiß Bob zwar, dass die digitale Signatur falsch ist. Er weiß jedoch nicht, ob es sich um eine echte Signatur zum falschen Hashwert oder um eine komplett falsche Signatur handelt. Es ist daher sinnvoll, einen Hashwert vor dem Signieren so zu formatieren, dass das Resultat eine erkennbare Struktur hat.
- Wenn Mallory die Möglichkeit hat, Hashwerte von Alice signieren zu lassen (das heißt, wenn Alice den Hashwert nicht selbst aus der Nachricht generiert), kann er eine Chosen-Ciphertext-Attacke starten (RSA bietet ja ein paar Möglichkeiten dafür). Diese Voraussetzung ist beispielsweise gegeben, wenn ein Hardwarechip Signaturen zu beliebigen Eingabewerten erstellt. Diese Angriffsmöglichkeit fällt weg, wenn Alice Hashwerte nur nach einer Datenformatierung signiert, die Mallory möglichst wenig Raum für das Einbringen von für ihn vorteilhaften Mustern lässt.

Um die genannten Probleme zu lösen, spezifiziert PKCS#1 zwei Methoden zur Datenformatierung für RSA-Signaturen. Dabei ist zu beachten, dass die typische Länge eines Hashwerts 160 Bit beträgt, RSA aber meist Datenblöcke der Länge 1.024 oder 2.048 Bit verschlüsselt (die Länge eines Datenblocks entspricht der Schlüssellänge). Aus diesem Grund schadet es nicht, wenn der formatierte Wert länger ist als der Hashwert.

Formatierung für Signierung nach PKCS#1 Version 1

Die Formatierung eines Hashwerts gemäß der ersten Version von PKCS#1 ist recht einfach. Das Format besteht aus vier Teilen:

- Das erste Byte des formatierten Hashwerts hat immer den Wert 00000001. Dadurch wird sichergestellt, dass der formatierte Hashwert kleiner ist als der Modulus.
- Nun folgen so viele Bytes mit dem Inhalt 255, dass die Gesamtlänge des formatierten Hashwerts der des Modulus entspricht.
- Nun folgt ein Byte mit dem Inhalt Null.
- Der vierte Teil besteht aus dem Hashwert und einer OID des Hashverfahrens.

Erhält Bob nach Anwendung von Alices öffentlichem Schlüssel auf eine Signatur einen Wert, der nicht der beschriebenen Syntax entspricht, dann weiß er sofort, dass es sich um keine echte Signatur handeln kann.

Formatierung für Signierung nach PKCS#1 Version 2

Der Nachteil am beschriebenen Verfahren ist natürlich, dass wiederum eine unerwünschte Regelmäßigkeit in die formatierten Daten kommt. Um dies zu vermeiden, sieht die zweite Version von PKCS#1 eine Maskierung vor, wie Sie sie bereits von der PKCS#1v2-Formatierung für die Verschlüsselung kennen. Dabei geht in den Hashwert zusätzlich zur Nachricht noch ein Zufallswert (Salt) ein. Das Salt ist zudem Teil eines Datenblocks, der außerdem noch einen Block von Nullbytes enthält, der die formatierte Nachricht auf die Länge des Schlüssels ausdehnt. Der Hashwert wird als Startwert für die Maskierung verwendet. Die formatierte Nachricht besteht aus dem Hashwert und dem maskierten Datenblock. Die Maskierung hat wiederum zur Folge, dass der resultierende Wert keine Regelmäßigkeiten aufweist, wodurch ein Zufallsorakel gegeben ist. Nach einer Demaskierung sieht Bob dennoch, ob der Hashwert gültig ist.

19.5 Datenformatierung für DLSSs

Bei Verfahren, die auf dem diskreten Logarithmus beruhen, spielt die Datenformatierung eine weit geringere Rolle als beim RSA-Verfahren. Bei Diffie-Hellman wird gar keine Formatierung benötigt, da bei diesem Verfahren nicht verschlüsselt oder signiert, sondern ein gemeinsamer Schlüssel generiert wird. Bei Signaturverfahren auf Basis des diskreten Logarithmus (DLSSs) – etwa ElGamal und DSA – ist eine Formatierung des Hashwerts ebenfalls nicht besonders interessant. Dies liegt zum einen daran, dass in jede Signatur ein Zufallswert eingeht (siehe Abschnitt 12.3). Dadurch ist von vornherein jede Signatur anders, auch wenn es sich um denselben Hashwert handelt. Zum anderen gibt es bei den gängigen DLSSs kein Verifikationsergebnis, das Bob auf eine bestimmte Struktur untersuchen kann (die Verifikation besteht stattdessen aus der Überprüfung einer mathematischen Gleichung). Deshalb bringt es auch nichts, den Hashwert um irgendwelche Erkennungszeichen zu erweitern. Ähnliche Überlegungen ergeben, dass auch eine Maskierung des Hashwerts bei einem DLSS keinen großen Sinn hätte.

20 Kryptografische Protokolle

Kryptografische Verfahren können ihre Wirkung meist nur dann entfalten, wenn sie in einen Ablauf eingebunden sind, an dem zwei oder mehrere Personen beteiligt sind. Beispielsweise kann Alice irgendwelche Daten verschlüsseln, die Bob anschließend entschlüsselt. Damit ein solcher Ablauf funktioniert, ist ein Ablaufplan notwendig, an den sich alle Beteiligten halten müssen. Ein solcher Ablaufplan wird **Kommunikationsprotokoll** (oder einfach nur **Protokoll**) genannt.

20.1 Protokolle

Protokolle haben zunächst einmal nichts mit Kryptografie zu tun. Vielmehr wird ein Protokoll immer dann benötigt, wenn Alice und Bob über ein Computernetz miteinander kommunizieren. Dann nämlich muss ein Ablaufplan regeln, wer wann eine bestimmte Nachricht abschickt und welche Bedeutung diese im jeweiligen Zusammenhang hat. Alice und Bob stehen dabei exemplarisch für zwei

beliebige Menschen oder Computer, und es können auch mehr als zwei Partner an der Kommunikation beteiligt sein.

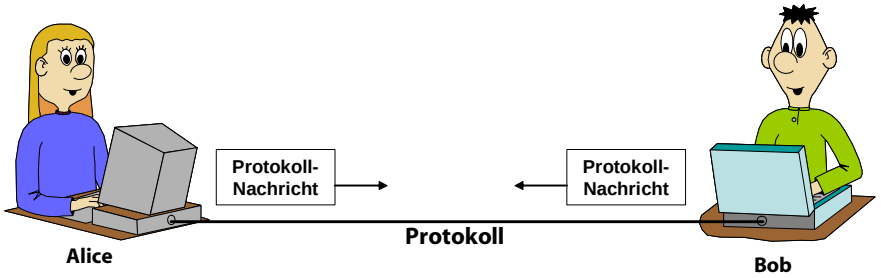


Abb. 20–1 *Durch ein geeignetes Protokoll können sich die Rechner von Alice und Bob miteinander verständigen.*

20.1.1 Konzeptprotokolle

Betrachten wir nun ein Beispielprotokoll, das es Alice ermöglicht, am Telefon bestimmte Sachen von Bob zu erfragen. Es könnte etwa so aussehen:

1. Alice wählt Bobs Nummer.
2. Bob nimmt den Hörer ab und meldet sich mit »Bob«.
3. Alice meldet sich mit »Alice«.
4. Alice stellt eine der folgenden Fragen:
   - »Spielst du heute um <Uhrzeit> mit mir Tennis?«
   - »Gehen wir heute um <Uhrzeit> ins Kino?«
   - »Treffen wir uns um <Uhrzeit> in der Pizzeria?«
5. Bob antwortet mit »ja« oder mit »nein«.
6. Wenn Alice eine weitere Frage hat, geht sie zurück zu Punkt 4, ansonsten verabschiedet sie sich mit »Tschüs Bob«.
7. Bob verabschiedet sich mit »Tschüs Alice«. Beide legen auf.

Wenn wir uns Alice und Bob als Computer vorstellen, dann ist es wichtig, dass beide das gleiche Protokoll verwenden und dass sich beide daran halten. Da Computer bekanntermaßen recht dumm sind, gäbe es beispielsweise Probleme, wenn Alice fragte »Spielen wir um <Uhrzeit> Squash?«. Diese Frage ist nicht im Protokoll vorgesehen, Bob kann sie daher nicht beantworten.

In vielen Fällen reicht es, wenn ein Protokoll auf dem Niveau beschrieben wird wie unser Beispielprotokoll. Etwas mehr als zehn Zeilen reichten für die Beschreibung aus. Allerdings kann ein solches Protokoll nicht ohne weiteres implementiert werden, denn dazu sind die einzelnen Schritte nicht genau genug festgelegt. Ist ein Protokoll nur auf derart vage Weise beschrieben, dann sprechen wir von einem **Konzeptprotokoll**. Konzeptprotokolle sind sehr nützlich, wenn es

darum geht, einen Ablauf zu veranschaulichen oder grobe Mängel darin zu entdecken. Als Grundlage für die Implementierung einer Software sind sie dagegen nicht geeignet.

20.1.2 Netzwerkprotokolle

Soll ein Protokoll implementiert werden, dann benötigt man eine sehr genaue Protokollspezifikation. Eine solche muss Bit für Bit festlegen, welche Daten unter welchen Umständen übertragen werden und welche Bedeutung diese jeweils haben. Dabei müssen auch Ausnahmesituationen und Fehlerfälle berücksichtigt werden, damit Alice beispielsweise nicht stundenlang auf eine Antwort von Bob wartet, während Bob längst die Kommunikation abgebrochen hat. Darüber hinaus muss eine gute Protokollspezifikation auch Dinge wie Kompatibilität zu älteren Versionen, Unterstützung von Standardformaten und vieles mehr vorsehen. Ein Protokoll, das so genau spezifiziert ist, dass es implementiert werden kann, heißt **Netzwerkprotokoll**. Durch die genannten Anforderungen kann die Beschreibung eines Netzwerkprotokolls schnell mehrere Dutzend Seiten füllen. Netzwerkprotokolle werden Ihnen in diesem Buch noch viele begegnen. Beispiele dafür sind etwa die im Internet gebräuchlichen Protokolle HTTP, IP, TCP und FTP.

Bei der Entwicklung eines Netzwerkprotokolls muss erst einmal eine Reihe von erlaubten Nachrichtentypen (sogenannte **Protokollnachrichten**) festgelegt werden. Eine Protokollnachricht ähnelt einem Befehl in einer Programmiersprache und kann gegebenenfalls mit einem Argument versehen werden. Beim obigen Protokoll könnte man etwa die Protokollnachrichten *Name(<Vorname>,<Nachname>)*, *Begrüßung(<Vorname>)*, *Frage(<Fragennummer>,<Uhrzeit>)*, *Antwort (<ja/nein>)* und *Abschied(<Vorname>)* festlegen. Protokollnachrichten haben in der Praxis stets noch einen Bestandteil, in dem Zusatzinformationen wie die Adresse des Absenders und des Empfängers, ein Hinweis auf die Länge der Nachricht oder Ähnliches gespeichert sind. Dieser Bestandteil wird als **Header** bezeichnet. Der für die Abarbeitung des Protokolls relevante Inhalt (etwa *Name(<Vorname>,<Nachname>)* wird dagegen **Nutzlast** (oder auch **Payload**) genannt. In der Regel steht der Header vor der Nutzlast.

20.1.3 Eigenschaften von Netzwerkprotokollen

Da Protokolle ein weites Feld sind, fragen wir uns erst einmal, welche Eigenschaften diese haben können bzw. müssen. Dabei beschränken wir uns auf Netzwerkprotokolle, da diese im Vergleich zu Konzeptprotokollen anspruchsvoller sind. Die vielleicht wichtigste Methode zur Klassifizierung von Netzwerkprotokollen ist das sogenannte OSI-Modell, das in Abschnitt 31.1 erläutert wird. In diesem Kapitel will ich Ihnen jedoch zunächst einmal einige Eigenschaften vorstellen, die von diesem Modell unabhängig sind.

Fehlertoleranz

Bei der Übertragung von Protokollnachrichten tauchen Fehler auf. So kommt es häufig genug vor, dass etwa eine Protokollnachricht von Alice bei Bob gar nicht ankommt oder dass ein Teil davon fehlt. Eine wichtige Eigenschaft, die ein Netzwerkprotokoll deshalb haben muss, ist **Fehlertoleranz**. Um Fehlertoleranz zu gewährleisten, muss eine Protokollspezifikation für alle Eventualitäten passende Aktionen und gegebenenfalls Protokollnachrichten mit Fehlermeldungen vorsehen. Im Notfall müssen sowohl Alice als auch Bob in der Lage sein, die Kommunikation nach einem Fehler abzubrechen.

Verhandlungsfähigkeit

Es gibt viele Details in einem Protokoll, die auf verschiedene Weise realisiert werden können. Dazu gehören etwa die Fragen, in welchem Format die Nutzdaten übertragen werden, welche Protokollversion verwendet wird, ob ein Komprimierungsverfahren eingesetzt wird und vieles mehr. Diese Belange müssen genau geklärt sein, damit eine Kommunikation stattfinden kann. Um trotzdem ein gewisses Maß an Flexibilität zuzulassen, sehen viele Netzwerkprotokolle die Möglichkeit vor, dass sich Alice und Bob auf bestimmte Parameter einigen können. Dies kann etwa so aussehen, dass Alice in einer dafür vorgesehenen Protokollnachricht eine Auswahl von Parametern vorgibt (etwa *Ich unterstütze die Versionen 1.0, 1.1, 1.2 und 2.0 des Protokolls*) und Bob sich aus dieser Vorgabe den ihm passenden Parameter aussucht. Unterstützt ein Protokoll ein derartiges Aushandeln von Parametern, dann spricht man von **Verhandlungsfähigkeit**. Die meisten Netzwerkprotokolle sind verhandlungsfähig.

Zustandslosigkeit

Wenn Alice in unserem anfangs beschriebenen Protokoll Bob eine der drei Fragen stellt, dann muss sie sich merken, welche Frage sie gestellt hat. Falls sie das nicht tut, kann sie mit Bobs Antwort (»ja« oder »nein«) nichts anfangen. Für einen Computer bedeutet dies, dass er einen Bereich seines Speichers dazu verwenden muss, um sich bestimmte Informationen über den bisherigen Ablauf des Protokolls zu merken. Dieser Speicherbereich wird **Kontext** genannt. Die Gesamtheit der Kontexte von Alice und Bob sowie der dazwischen liegenden Kommunikationsverbindung wird als **Assoziation** bezeichnet. Am besten ist es, wenn sich Alice und Bob in ihrem Kontext jeweils merken, in welcher Phase der Kommunikation sie gerade stehen, denn dadurch wird es einfacher, bei fehlerhaften oder ausbleibenden Protokollnachrichten geeignet zu reagieren. Ein Kontext kann verschiedene Werte annehmen. Alices Kontext könnte bei unserem Protokoll etwa folgende Werte haben (die Uhrzeit lassen wir dabei erst einmal außen vor):

1. Anfangswert
2. Name von Bob eingegangen und eigener Name abgeschickt
3. Frage 1 gestellt
4. Frage 2 gestellt
5. Frage 3 gestellt
6. Frage wird von Bob mit »ja« beantwortet
7. Frage wird von Bob mit »nein« beantwortet
8. Ende 1: Alice und Bob spielen Tennis
9. Ende 2: Alice und Bob gehen ins Kino
10. Ende 3: Alice und Bob essen Pizza
11. Ende 4: Alice und Bob treffen sich überhaupt nicht

Jeder dieser Werte, die der Kontext annehmen kann, wird als **Zustand** bezeichnet. Der Kontext kann nicht von einem beliebigen Zustand in jeden anderen übergehen (Ende 1 folgt beispielsweise nur dann, wenn Bob Frage 1 mit »ja« beantwortet). Es muss also genau definiert werden, welche anderen Zustände von einem bestimmten Zustand erreicht werden dürfen. Bei einem fehlertoleranten Protokoll müssen Alice und Bob zudem in der Lage sein, zu einem früheren Zustand zurückzugehen, um einen Teil der Kommunikation zu wiederholen. Dazu können eventuell zusätzliche Zustände notwendig sein. Nimmt man noch variable Größen wie die Uhrzeit dazu, dann gibt es noch sehr viel mehr Zustände.

Wichtig ist nun zu wissen, dass Alice bei manchen Protokollen ohne Kontext auskommt. In diesem Fall trifft sie ihre Entscheidungen nur auf Basis allgemeiner Regeln und mithilfe der letzten erhaltenen Nachricht. Eine Komponente ohne Kontext wird **zustandslos** genannt. Im anderen Fall nennt man sie **zustandsbehaftet**. Wenn alle an einer Kommunikation beteiligten Parteien zustandslos sind, dann spricht man von einem **zustandslosen Protokoll**. Zustandslose Protokolle sind einfacher und leichter zu implementieren als zustandsbehaftete. Letztere sind dagegen leistungsfähiger. Ein typisches Beispiel für ein zustandsloses Protokoll ist das Verschicken einer E-Mail. Da eine Mail normalerweise anderen E-Mails unabhängig ist, benötigen Alice und Bob keinen Kontext.

20.2 Protokolle in der Kryptografie

Auch in der Kryptografie spielen Protokolle eine wichtige Rolle. Meist genügt es nämlich nicht, irgendwelche Daten einfach nur zu verschlüsseln oder zu signieren. Vielmehr muss in der Praxis ein Ablauf festgelegt werden, mit dem zwei oder mehr Personen zu einem bestimmten Ziel kommen. Von einem **kryptografischen Protokoll** (oder auch **Krypto-Protokoll**) sprechen wir, wenn Alice und Bob in einem Protokoll kryptografische Techniken (insbesondere Verschlüsselung, digitale Signatur oder kryptografische Hashfunktionen) verwenden, um sich damit vor Angriffen von Mallory zu schützen. Ein kryptografisches Protokoll kann als Konzept- oder Netzwerkprotokoll vorliegen. Einige Beispiele für kryptografische

Protokolle haben Sie bereits kennengelernt. Der Diffie-Hellman-Schlüsselaustausch ist ein solches, MQV ebenfalls. Wenden Alice und Bob ein Hybridverfahren zum Schlüsselaustausch und zur verschlüsselten Kommunikation an, dann handelt es sich ebenfalls um ein kryptografisches Protokoll.

Bei einem kryptografischen Netzwerkprotokoll müssen die eingesetzten Krypto-Verfahren mit allen Formaten genau festgelegt sein. Die PKCS-Standards werden beispielsweise häufig als Grundlage für kryptografische Netzwerkprotokolle verwendet. Natürlich können auch an einem kryptografischen Protokoll mehr als zwei Kommunikationspartner beteiligt sein. In der Regel sind es jedoch zwei Beteiligte, in unserem Fall also meist Alice und Bob.

20.2.1 Eigenschaften kryptografischer Netzwerkprotokolle

Auch bei kryptografischen Protokollen spielen die in Abschnitt 20.1.3 beschriebenen Eigenschaften eine Rolle. Wie Sie sehen werden, gehen die Ansprüche an kryptografische Protokolle jedoch weit über die bereits behandelten hinaus.

Fehlertoleranz bei kryptografischen Protokollen

Wie bei jedem Netzwerkprotokoll ist auch bei einem kryptografischen Netzwerkprotokoll die Fehlertoleranz eine wichtige Eigenschaft. Wie schon bei Hashfunktionen und bei Zufallsgeneratoren gilt jedoch auch an dieser Stelle: Die Anforderungen, welche die Kryptografie stellt, sind deutlich höher als die anderer Anwendungen. Bei einem kryptografischen Protokoll rechnen wir nämlich nicht nur damit, dass auf dem Übertragungsweg Nachrichten verloren gehen oder dass versehentlich ein Bit umkippt. Vielmehr gehen wir wiederum davon aus, dass Abhörer Mallory durch gezielte Änderungen oder Unterschlagungen versucht, sich einen Vorteil zu verschaffen. Zur Fehlertoleranz, die ein Teilbereich der Safety ist, kommt bei kryptografischen Protokollen daher Manipulationstoleranz als Anforderung. Welche Möglichkeiten Mallory zur Manipulation hat und welche Gegenmaßnahmen es gibt, erfahren Sie im Verlauf dieses Kapitels.

Verhandlungsfähige kryptografische Protokolle

Wie Sie sich denken können, ist Verhandlungsfähigkeit auch und gerade bei kryptografischen Protokollen ein wichtiger Aspekt. Es geht vor allem darum, dass Alice und Bob die Verwendung von Krypto-Verfahren und zugehörigen Parametern aushandeln können. Möglichkeiten zur Verhandlung in kryptografischen Protokollen gibt es viele: Alice kann Bob in einer Protokollnachricht am Anfang der Kommunikation etwa mitteilen, welche Verfahren für den Schlüsselaustausch, welche Schlüssellängen für den öffentlichen Schlüssel, welche symmetrischen Verschlüsselungsverfahren, welche symmetrischen Schlüssellängen und welche Betriebsarten sie unterstützt. Bob kann sich anschließend die ihm pas-

sende Kombination aus dieser Vorauswahl aussuchen. Die einzelnen Verfahren und Parameter werden dabei meist mithilfe von OIDs identifiziert.

Ist ein kryptografisches Netzwerkprotokoll verhandlungsfähig, dann hat dies nicht nur den Vorteil, dass Alice und Bob ihre aktuellen Lieblingsverfahren einsetzen können. Vorteilhaft ist zusätzlich, dass in ein bestehendes Krypto-Protokoll im Nachhinein neue Verfahren eingebracht werden können. Nicht zuletzt deshalb legen sich die meisten Netzwerkprotokolle nicht auf ein bestimmtes Verfahren fest. Vielmehr wird meist eine Liste von unterstützten Verfahren genannt, aus denen sich der Implementierer und später auch der Anwender seine Präferenz aussuchen kann. Diese Liste ist in der Regel erweiterbar, ohne dass die Protokollspezifikation geändert werden muss.

Zustandslosigkeit bei kryptografischen Protokollen

Zustände spielen auch bei kryptografischen Protokollen eine wichtige Rolle. Stellen Sie sich vor, Alice und Bob tauschen mit dem RSA-Verfahren einen Schlüssel aus und verschlüsseln danach ihre Kommunikation mit einem symmetrischen Verfahren. In diesem Fall brauchen Alice und Bob zumindest einen Kontext, in dem sie den geheimen Schlüssel speichern können. Schon ein so einfaches Protokoll kann daher kaum zustandslos realisiert werden. Eine Assoziation wird bei kryptografischen Protokollen **Sicherheitsassoziation** genannt.

Minimum Disclosure

Eine Protokollnachricht ist im Normalfall nicht vollständig verschlüsselt – zumindest ein Teil des Headers muss unverschlüsselt bleiben, damit der Empfänger und die Zwischenstationen auf dem Transport etwas mit der Nachricht anfangen können. Ziel muss es jedoch sein, so wenig wie möglich unverschlüsselt zu übertragen. Dieses Ziel wird als **Minimum Disclosure** bezeichnet. Was genau darunter zu verstehen ist, hängt vom jeweiligen Protokoll ab.

20.3 Angriffe auf kryptografische Protokolle

Wenn Alice und Bob sichere Krypto-Verfahren einsetzen, dann heißt dies noch lange nicht, dass sie Mallory damit überlistet haben. Es gibt nämlich durchaus Angriffe auf Protokolle, die auch bei sicheren Krypto-Verfahren funktionieren. Um diese geht es in den folgenden Unterkapiteln. Dabei gehen wir davon aus, dass Mallory das Protokoll kennt, das Alice und Bob verwenden (es gilt also das Kerckhoffs'sche Prinzip). Wenn es um Angriffe auf Protokolle geht, dann dürfen Sie sich Mallory nicht nur als Abhörer vorstellen, der die Kommunikation zwischen Alice und Bob belauscht, sondern auch als jemanden, der aktiv in die Kommunikation eingreift.

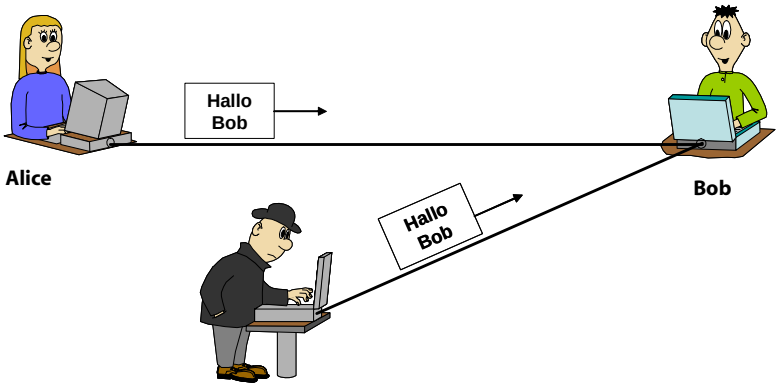


Abb. 20–2 *Bei einer Replay-Attacke spielt Mallory eine abgefangene Nachricht erneut ein.*

20.3.1 Replay-Attacke

Alice kennt sich an der Börse aus und möchte Bob daran teilhaben lassen. Deshalb haben die beiden ein einfaches (zustandsloses) Protokoll vereinbart, das nur aus einer einzigen Protokollnachricht besteht. Diese lautet *Kaufe Aktien der Firma <Firmenname>*, wobei jeweils der Name einer Firma eingesetzt wird. Immer wenn Alice einen heißen Tipp hat, schickt sie eine solche Nachricht an Bob. Damit Mallory nicht mitlesen kann, verschlüsselt Alice den Nutzteil der Protokollnachricht mit dem AES, der bekanntlich sehr sicher ist. Da Alice und Bob einen gemeinsamen Schlüssel besitzen, brauchen sie sich um den Schlüsselaustausch keine Sorgen zu machen.

Muss Mallory nun klein beigeben? Nein, denn er kann das Protokoll angreifen, ohne die AES-Verschlüsselung knacken zu müssen. Dieser Angriff ist denkbar einfach: Schickt Alice eine Nachricht an Bob, in der sie zum Kauf der Aktien von Krypt & Co. rät, dann hört Mallory diese ab und sendet sie ein paar Tage später erneut. Bob denkt dann, Alice würde nun erneut zum Kauf der Krypt & Co.-Aktien raten und kauft diese. Tags darauf sinkt der Aktienkurs, und Bob verliert sein Erspartes. Einen solchen Angriff auf ein Protokoll, bei dem Mallory eine ältere Nachricht noch einmal sendet, wird **Replay-Attacke** genannt. Das Besondere an einer Replay-Attacke ist, dass Mallory die betreffende Nachricht in der Regel selbst nicht lesen kann, da sie verschlüsselt ist. Er kann damit jedoch einen bereits ausgelösten Vorgang noch einmal auslösen.

Die beste Gegenmaßnahme gegen eine Replay-Attacke liegt auf der Hand: Alice und Bob müssen eine Zeitangabe (**Timestamp**) in die verschlüsselte Protokollnachricht aufnehmen. Ist der Timestamp zu alt, dann akzeptiert Bob die Nachricht nicht. Es gibt jedoch auch andere Methoden zur Abwehr von Replay-Attacken: So können sich Alice und Bob auch darauf einigen, dass bei jeder Nachricht eine Nummer mitgeschickt wird, die hochgezählt wird. Ziel ist es in

jedem Fall, dass keine zwei Nachrichten von Alice gleich aussehen, denn dann fällt eine Wiederholung sofort auf.

20.3.2 Spoofing-Attacke

Da Mallory das Protokoll kennt, mit dem Alice und Bob kommunizieren, kann er versuchen, eine Kommunikation mit Bob in Gang zu bringen und sich dabei für Alice auszugeben. Ein solcher Angriff wird **Spoofing-Attacke** genannt. Bei nichtkryptografischen Protokollen ist Spoofing oft recht einfach. So ist es etwa für Mallory eine leichte Übung, eine falsche IP-Adresse in seinen Nachrichten anzugeben (IP-Spoofing). Spoofing-Attacken können mit einer guten Authentifizierung verhindert werden (siehe Abschnitt 21.3).

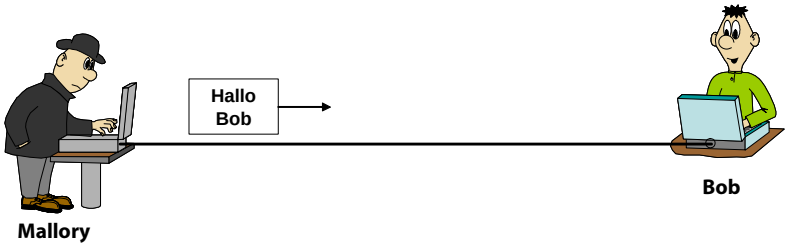


Abb. 20–3 *Bei einer Spoofing-Attacke gibt sich Mallory als Alice aus.*

20.3.3 Man-in-the-Middle-Attacke

Alice und Bob verwenden das Diffie-Hellman-Verfahren zum Schlüsselaustausch. Obwohl das Verfahren selbst sehr sicher ist, kann Mallory das Protokoll angreifen, wenn er die Protokollnachrichten unbemerkt verändern kann. Der Angriff ist recht einfach: Wenn Alice ihren öffentlichen Schlüssel g^x an Bob schickt, dann fängt Mallory ihn ab, ersetzt ihn durch seinen eigenen öffentlichen Schlüssel und sendet diesen an Bob weiter. Wenn Bob anschließend seinen öffentlichen Schlüssel g^y an Alice schickt, ersetzt Mallory auch diesen durch seinen eigenen öffentlichen Schlüssel. Anschließend berechnen Alice und Bob aus dem empfangenen öffentlichen Schlüssel jeweils einen geheimen Schlüssel. Die beiden denken nun, sie hätten einen gemeinsamen geheimen Schlüssel $k=g^{xy} \pmod p$. In Wirklichkeit hat jedoch Alice nun einen geheimen Schlüssel k_1 gemeinsam mit Mallory und Bob einen Schlüssel k_2 ebenfalls gemeinsam mit Mallory. Wenn Alice anschließend eine mit k_1 verschlüsselte Nachricht an Bob schickt, dann fängt Mallory diese ab, entschlüsselt sie, verschlüsselt sie erneut mit k_2 und sendet sie an Bob. Schickt Bob eine Nachricht an Alice, dann läuft die Sache umgekehrt. Alice und Bob denken also, sie kommunizieren miteinander, doch in Wirklichkeit kommunizieren beide mit Mallory.

Abb. 20–4 *Bei einer Man-in-the-Middle-Attacke greift Mallory aktiv in die Kommunikation zwischen Alice und Bob ein.*

Einen solchen Angriff, bei dem Mallory aktiv in die Kommunikation zwischen Alice und Bob eingreift, nennt man **Man-in-the-Middle-Attacke** (nicht zu verwechseln mit der Meet-in-the-Middle-Attacke). Bei einer Man-in-the-Middle-Attacke verändert Mallory Protokollnachrichten in einer Form, die ihm Vorteile bringt. Dies ist etwa bei verhandlungsfähigen kryptografischen Protokollen eine Gefahr: Mallory kann hier die Krypto-Verfahren, die Alice Bob anbietet, verändern. So kann er erreichen, dass die beiden ein unsicheres oder gar kein Verfahren einsetzen. Eine Man-in-the-Middle-Attacke funktioniert meist nur in der Anfangsphase eines Protokolls. Sobald Alice und Bob einen gemeinsamen geheimen Schlüssel haben und diesen richtig einsetzen (zur Verschlüsselung oder für eine schlüsselabhängige Hashfunktion), ist Mallory machtlos.

Abb. 20–5 *Bei einer Hijacking-Attacke übernimmt Mallory eine Verbindung von Alice.*

20.3.4 Hijacking-Attacke

Bei einer Man-in-the-Middle-Attacke beschränkt sich Mallory darauf, Protokollnachrichten gezielt zu verändern. Mallory kann jedoch noch weiter gehen: Er kann die Protokollnachrichten von Alice ab einem bestimmten Zeitpunkt ganz abblocken und von da an die Kommunikation mit Bob selbst führen. In einem solchen Fall spricht man von einer **Hijacking-Attacke**. Eine Hijacking-Attacke ist eine Kombination aus Man-in-the-Middle- und Spoofing-Attacke. Die Gegenmaßnahmen sind daher klar: Mit einer Authentifizierung und der Verwendung von Verschlüsselung oder einer kryptografischen Hashfunktion können Alice und Bob Mallory das Handwerk legen.

20.3.5 Known-Key-Attacken

In der Kryptografie setzt man normalerweise voraus, dass Mallory einen von Alice und Bob verwendeten Schlüssel nicht kennt (öffentliche Schlüssel sind natürlich eine Ausnahme). Wenn es jedoch um Protokolle geht, müssen wir diese Voraussetzung etwas aufweichen. Alice und Bob verwenden eine Protokollimplementierung möglicherweise über Jahre hinweg, ohne dabei ständig den Schlüssel zu wechseln. Da kann es schon einmal vorkommen, dass Mallory mit seiner kriminellen Energie in den Besitz eines Schlüssels gelangt. Selbstverständlich ist ein Schlüssel in Mallorys Händen aus kryptografischer Sicht immer eine Katastrophe. Es stellt sich jedoch die Frage, ob sich der Schaden begrenzen lässt, den Mallory mit einem gestohlenen Schlüssel anrichten kann. Die Antwort ist ja, denn man kann ein Protokoll durchaus so gestalten, dass Mallory mit einem bekannten Schlüssel (**Known-Key-Attacke**) keinen Flurschaden anrichten kann.

Masterschlüssel und Sitzungsschlüssel

Wenn wir davon ausgehen, dass Alice und Bob einen gemeinsamen geheimen Schlüssel besitzen (z.B. einen Preshared Key), dann gibt es eine einfache Maßnahme zur Erschwerung einer Known-Key-Attacke. Diese sieht vor, dass der besagte geheime Schlüssel als **Masterschlüssel** genutzt wird. Ein Masterschlüssel kommt nie zum Verschlüsseln oder für eine schlüsselabhängige Hashfunktion zum Einsatz. Stattdessen nutzen ihn Alice und Bob nur zur Generierung anderer Schlüssel (beispielsweise, indem sie eine kryptografische Hashfunktion darauf anwenden). Nur mit den neu generierten Schlüsseln, die auch als **Sitzungsschlüssel** bezeichnet werden, wird verschlüsselt oder gehasht. Wenn Alice und Bob für jede Kommunikation einen neuen Sitzungsschlüssel vom Masterschlüssel ableiten, dann erhöht dies die Sicherheit eines Protokolls gegenüber einer Known-Key-Attacke deutlich. Voraussetzung ist, dass Alice und Bob ihren Masterschlüssel besonders schützen. Idealerweise speichern sie ihn unauslesbar auf einer Smartcard. Diese Smartcard übernimmt dann das Generieren der Sitzungsschlüssel, mit denen anschließend in einem weniger geschützten Umfeld das Protokoll abgearbeitet wird.

Forward Security und Backward Security

Nehmen wir an, Alice und Bob schicken sich jeden Tag ein paar Nachrichten zu. Sie verschlüsseln ihre Kommunikation mit einem Hybridverfahren bestehend aus dem AES und RSA, wobei sie den AES-Schlüssel täglich wechseln und neu per RSA übertragen. Mallory zeichnet die verschlüsselte Kommunikation zwischen Alice und Bob über längere Zeit hinweg auf. Irgendwann – die CDs stapeln sich schon in seinem Keller – gelingt es Mallory, in Besitz eines Schlüssels zu kommen. Kann Mallory damit nun im Rahmen einer Known-Key-Attacke die gesamte aufgezeichnete Kommunikation nachträglich entschlüsseln?

Die Anwort hängt davon ab, welchen Schlüssel Mallory besitzt. Handelt es sich um den RSA-Schlüssel, den Alice und Bob seit Jahren unverändert nutzen, dann haben die beiden Pech: Mallory kann damit alle AES-Sitzungsschlüssel entschlüsseln, die er abgefangen hat, und dadurch alle gesammelten Nachrichten lesen. Ist es dagegen ein AES-Schlüssel, den Mallory in seinen Besitz gebracht hat, dann hält sich der Schaden in Grenzen. Denn da Alice und Bob täglich einen neuen AES-Schlüssel einsetzen, beschränken sich Mallorys Entschlüsselungsmöglichkeiten auf einen Tag.

Natürlich lassen sich solche Überlegungen auch bei anderen kryptografischen Protokollen anstellen. Wenn wir dabei weiterhin von einem täglichen Schlüsselwechsel ausgehen, stellt sich jeweils folgende Frage: Was passiert, wenn Mallory das Schlüsselmaterial eines Tages kennt? Klar ist, dass er an diesem Tag alles entschlüsseln kann. Das verwendete Protokoll kann jedoch so gestaltet sein, dass Mallory mit den Schlüsseln eines Tages keine Chance hat, die Nachrichten früherer Tage zu entschlüsseln. In diesem Fall spricht man von **Backward Security**. Hat Mallory keine Möglichkeit, die Nachrichtenübertragungen späterer Tage zu lesen, dann heißt die entsprechende Protokolleigenschaft **Forward Security**. Die meisten Protokolle, die Backward Security bieten, bieten auch Forward Security.

Wie wir sehen werden, gibt es Protokolle, die zwar ein hohes Maß an Forward Security oder Backward Security bieten, die aber – selbst wenn die eingesetzten Krypto-Verfahren sicher sind – unter bestimmten Bedingungen anfällig gegenüber einer Known-Key-Attacke sind. Es ist daher manchmal sinnvoll, von perfekter und nichtperfekter Forward Security bzw. Backward Security zu reden. Sind beide Eigenschaften in perfekter Form erfüllt, dann bezeichnet man dies auch als **Perfect Forward Secrecy** (**PFS**). Leider ist diese weit verbreitete Bezeichnung nicht ganz glücklich, da intuitiv nicht klar ist, dass »forward« in diesem Fall »backward« miteinschließt. Auch die Unterscheidung zwischen »Security« und »Secrecy«, die hier nicht von Belang ist, kann zu Verwirrungen führen. Ich werde den Begriff Perfect Forward Secrecy daher nicht weiter verwenden. Außerdem ist im Folgenden, falls nicht anders gesagt wird, mit Forward Security bzw. Backward Security immer der perfekte Fall gemeint.

Sowohl (perfekte) Forward Security als auch (perfekte) Backward Security sind zum Beispiel gegeben, wenn Alice und Bob den Diffie-Hellman-Schlüsselaustausch einsetzen und dabei für jede Kommunikation neue Schlüsselpaare verwenden (dabei müssen sie eine Man-in-the-Middle-Attacke verhindern, beispielsweise durch einen zusätzlichen Preshared Key). Noch besser ist es, statt Diffie-Hellman MQV einzusetzen, das für Alice und Bob je zwei Schlüsselpaare vorsieht – diese haben den Zweck, Forward Secutrity und Backward Security zu gewährleisten. Auch wenn Alice und Bob das RSA-Verfahren zum Schlüsselaustausch verwenden, können sie die beiden gewünschten Eigenschaften erreichen. Dazu müssen sie für jede Kommunikation ein neues RSA-Schlüsselpaar einsetzen, wobei sie wiederum eine Man-in-the-Middle-Attacke verhindern sollten.

Ein Beispiel für nichtperfekte Forward Security liefert das sogenannte **Baby-Duck-Verfahren**. Dieses sieht vor, dass jeder Sitzungsschlüssel, den Alice und Bob an einem Tag einsetzen, von den Sitzungsschlüsseln früherer Tage abhängt. Wenn Mallory also das Schlüsselmaterial eines Tages kennt, dann kann er damit nicht die Kommunikation des nächsten entschlüsseln – es sei denn, er kennt auch sämtliche Sitzungsschlüssel der Tage und Wochen zuvor. Die Forward Security ist in diesem Fall nicht perfekt, weil Mallory aus vorhandenen Schlüsseln auf zukünftige schließen kann. Doch je mehr Tage in das Baby-Duck-Verfahren einbezogen werden, desto unwahrscheinlicher wird dies.

Key-Compromise Impersonation Security

Wenn Mallory Alices privaten RSA-Schlüssel kennt, dann kann er sich gegenüber Bob als Alice ausgeben (Gleiches gilt natürlich, wenn es sich um einen privaten Diffie-Hellman-, DSA- oder ECC-Schlüssel handelt). Verwenden Alice und Bob ein symmetrisches Verfahren für die Authentifizierung oder Verschlüsselung, dann kann sich Mallory gegenüber Alice als Bob ausgeben, sofern er den gemeinsamen geheimen Schlüssel der beiden kennt. Die genannten Überlegungen sind nicht besonders erstaunlich und zeigen, dass eine Known-Key-Attacke schon per Definition eine große Gefahr darstellt. Zusätzlich stellt sich folgende Frage: Kann sich Mallory mit einem von Alice gestohlenen Schlüssel gegenüber Alice als eine andere Person ausgeben? Hat ein kryptografisches Protokoll die Eigenschaft, dass Mallory einen solchen Angriff (es handelt sich um eine Spoofing-Attacke) nicht starten kann, dann spricht man von **Key-Compromise Impersonation Security**.

Verwenden Alice und Bob ein symmetrisches Verfahren, dann kann sich Mallory gegenüber Alice als Bob ausgeben, nachdem er Alice den geheimen Schlüssel gestohlen hat (dies liegt daran, dass Alice und Bob denselben Schlüssel verwenden). Key-Compromise Impersonation Security ist diesbezüglich also nicht gegeben. Andererseits kann sich Mallory nicht gegenüber Alice als Carol ausgeben, da Alice und Carol einen anderen Schlüssel verwenden (wir nehmen an, dass Mallory diesen nicht kennt). Eine symmetrische Authentifizierung bietet also immerhin teilweise Key-Compromise Impersonation Security.

Verwenden Alice und Bob RSA für Schlüsselaustausch und Authentifizierung, ist Key-Compromise Impersonation Security gegeben. Anders liegt der Fall bei Diffie-Hellman. Den gemeinsamen Schlüssel $g^{xy} \pmod p$ kann Mallory berechnen, sobald er Alices privaten Schlüssel x und Bobs öffentlichen Schlüssel $g^y \pmod p$ kennt. Bobs privater Schlüssel y muss ihm dazu nicht bekannt sein. Um Diffie-Hellman durch eine Erweiterung mit Key-Compromise Impersonation Security auszustatten, wurden Protokolle wie MQV und HMQV entwickelt.

20.3.6 Verkehrsflussanalyse

Auch wenn Alice und Bob alles sicher verschlüsseln, was sie sich zuschicken, und selbst wenn sie dabei ein sicheres Protokoll verwenden, so bleibt Mallory doch eine letzte Möglichkeit, um etwas über die Kommunikation zu erfahren: Er kann eine Statistik darüber führen, wann und wie Alice und Bob miteinander kommunizieren. Dieser Angriff wird **Verkehrsflussanalyse** genannt. Verkehrsflussanalysen sind gefährlicher, als Sie vielleicht denken. Wenn Alice etwa als Mitarbeiterin von Krypt & Co. mit einem Börsianer kommuniziert, der kurz darauf Aktien des Unternehmens kauft, dann kann dies ein Hinweis auf Insiderhandel sein. Wenn der Chef der Firma Krypt & Co. auf einmal rege mit dem Chef eines Konkurrenzunternehmens korrespondiert, dann steht vielleicht eine Fusion bevor.

Eine Verkehrsflussanalyse wurde übrigens den »raffiniertesten Verbrechern Deutschlands« [Spitra] zum Verhängnis. Es handelt sich dabei um zwei Kriminelle, die mit der Entführung der Industriellenerben Lars und Meike Schlecker sowie mit mehreren weiteren Kapitalverbrechen Millionen erbeuteten. Ihr Weg war nach 23 Jahren zu Ende, als einer der beiden aus einer Telefonzelle zunächst mit einem Opfer telefonierte und anschließend zu Hause anrief. Die Polizei konnte den fraglichen Anschluss ermitteln und die Täter festnehmen.

Leider kann man eine Verkehrsflussanalyse mit kryptografischen Mitteln nicht verhindern. Es gibt jedoch einige Möglichkeiten, Mallory seine Arbeit zu erschweren: So sollten Alice und Bob möglichst alles verschlüsseln, was über das Netz geht (nicht nur die wirklich geheimen Dinge). Dadurch wird es für Mallory erheblich schwieriger, das Interessante vom Uninteressanten zu trennen. Zum anderen sollten die verwendeten Protokolle das Minimum-Disclosure-Prinzip befolgen. Dies bedeutet, dass nicht nur die Nutzdaten einer Protokollnachricht verschlüsselt werden, sondern möglichst alles außer der Zieladresse. Darüber hinaus können verschlüsselte Dummy-Mails ohne relevanten Inhalt Mallory das Leben schwer machen.

Abb. 20–6 *Bei einer Verkehrsflussanalyse kennt Mallory nur Sender und Empfänger, nicht aber den Inhalt der Nachrichten.*

20.3.7 Denial-of-Service-Attacke

Wenn Mallory bei seinen Angriffsversuchen auf Protokolle überhaupt nichts mehr in seiner Trickkiste hat, dann kann er es immer noch mit einer **Denial-of-Service-Attacke** versuchen. Eine solche hat ausschließlich das Ziel, die Kommunikation zwischen Alice und Bob zu verhindern. Weit verbreitet sind dafür zwei Möglichkeiten:

- Mallory blockt eine Protokollnachricht von Alice ab oder verändert sie so, dass Bob damit nichts mehr anfangen kann.
- Mallory überflutet Bob mit falschen Protokollnachrichten, damit dieser die echte Nachricht von Alice nicht rechtzeitig bearbeiten kann. Diese Variante funktioniert oft besonders gut, wenn asymmetrische Verfahren im Spiel sind. So kann beispielsweise eine Signaturverifikation recht aufwendig sein. Wenn Mallory Bob in großen Mengen mit signierten Nachrichten eindeckt, kann dessen Rechner möglicherweise schnell in die Knie gehen.

Oft gibt es noch andere Möglichkeiten, wie Mallory durch falsche Protokollnachrichten eine Kommunikation verhindern kann. Dies kann etwa bei einer fehlerhaften Implementierung der Fall sein, die sich durch eine falsche Protokollnachricht zum Absturz bringen lässt. Übrigens sind auch Denial-of-Service-Attacken häufig wirkungsvoller, als es scheint. Wenn Mallory beispielsweise jede verschlüsselte Protokollnachricht von Alice an Bob so verändert, dass Bob sie nicht mehr entschlüsseln kann, dann werden die beiden früher oder später an der Qualität ihrer Verschlüsselungssoftware zweifeln und ihre Nachrichten der Einfachheit halber unverschlüsselt austauschen – davon profitiert Abhörer Mallory natürlich ungemein.

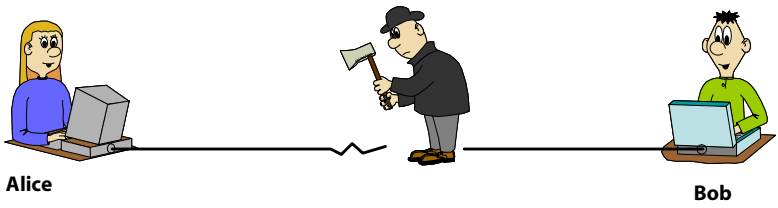


Abb. 20–7 *Bei einer Denial-of-Service-Attacke verhindert Mallory, dass überhaupt eine Kommunikation stattfindet. Leider lässt sich ein solcher Angriff mithilfe der Kryptografie nur schwer bekämpfen.*

20.3.8 Sonstige Angriffe

Es gibt auch Angriffe auf Protokolle, die nicht zu den bisher beschriebenen gehören. Wichtig ist in jedem Fall die Tatsache, dass sichere Krypto-Verfahren noch lange kein sicheres Protokoll machen. Vielmehr ist auch die Entwicklung von Protokollen eine eigenständige Disziplin der Kryptografie, deren Erkenntnisse bei der Entwicklung von Krypto-Produkten berücksichtigt werden müssen.

20.4 Beispielprotokolle

In Abschnitt 7.1 haben Sie erfahren, dass das Design eines Verschlüsselungsverfahrens eine schwierige Sache ist. Das Design eines Protokolls steht diesem in nichts nach. Wenn es um ein Protokoll mit komplexen Anforderungen geht (z.B. mit mehr als zwei Beteiligten, die unterschiedliche Interessen haben), kann das Thema Protokolldesign sogar zu einer übermenschlichen Aufgabe werden. Damit Sie von der Herausforderung des Protokolldesigns wenigstens eine schwache Ahnung bekommen, wollen wir uns im Folgenden ein (einfaches) Beispiel anschauen.

20.4.1 Beispielprotokoll: Messgerät sendet an PC

Um ein beispielhaftes kryptografisches Protokoll zu entwickeln, nehmen wir an, in einer Industrieanlage befinde sich ein Messgerät, das bei Bedarf einen bestimmten Wert misst und an den PC von Mitarbeiterin Alice sendet. Den Messwert nehmen wir als vertraulich an. Wie üblich gehen wir davon aus, dass Bösewicht Mallory, der ein Interesse an den Messwerten hat, die gesamte Kommunikation nach Belieben abhören, verändern und blockieren kann. Ohne Kryptografie sieht unser Protokoll wie folgt aus:

1. Alice sendet eine Anfrage an das Messgerät.
2. Das Messgerät führt eine Messung durch und sendet das Ergebnis an Alice.

Da keine Kryptografie im Einsatz ist, hat Mallory leichtes Spiel. Er kann insbesondere die Antwort des Messgeräts lesen, was nicht erwünscht ist. Wenn wir nun Kryptografie einsetzen, gelangen wir zu folgendem Protokoll:

1. Alice sendet eine Anfrage an das Messgerät.
2. Das Messgerät führt eine Messung durch und sendet das Ergebnis verschlüsselt an Alice. Zur Verschlüsselung verwendet das Gerät einen vorab konfigurierten symmetrischen Schlüssel, der nur ihm und Alice (bzw. ihrem PC) bekannt ist (Preshared Key).

Mallory kann nun die Anwort nicht mehr lesen, was schon ein wichtiger Schritt ist. Er kann jedoch eine Anfrage fälschen. Dies ist unerwünscht, da ein Messvorgang zum falschen Zeitpunkt das Gerät zerstören kann. Wir müssen also eine Authentifizierung einbauen. Etwa so:

1. Alice sendet eine Anfrage an das Messgerät, die mit einem schlüsselabhängigen Hashwert gesichert ist. Dazu verwendet sie denselben Schlüssel wie das Messgerät zur Verschlüsselung.
2. Das Messgerät überprüft den Hashwert, führt im positiven Fall eine Messung durch und sendet das Ergebnis verschlüsselt an Alice.

Nun kann Mallory zwar nicht mehr nach Belieben Anfragen fälschen. Er kann jedoch eine einmal von Alice verwendete Anfrage wiederverwenden (Replay-Attacke). Wir müssen daher die Anfragen nummerieren:

1. Alice sendet eine gehashte Anfrage an das Messgerät. Die Anfrage enthält eine Nummer, die jeweils hochgezählt wird.
2. Das Messgerät überprüft den Hashwert sowie die Nummer, führt im positiven Fall eine Messung durch und sendet das Ergebnis verschlüsselt an Alice.

Da Mallory Nachrichten abblocken kann, funktioniert folgender Angriff: Wenn Alice eine Anfrage stellt, blockt Mallory diese ab. Alice erhält daher keine Antwort und denkt möglicherweise, es liege ein Defekt vor. Mallory wartet bis zu einem für ihn günstigen Zeitpunkt und sendet dann die Anfrage an das Messgerät. Um dies zu vermeiden, können wir Zeitstempel einsetzen:

1. Alice sendet eine gehashte Anfrage an das Messgerät. Die Anfrage enthält die aktuelle Uhrzeit.
2. Das Messgerät überprüft den Hashwert sowie die Uhrzeit, führt im positiven Fall eine Messung durch und sendet das Ergebnis zusammen mit der Anfrage-Uhrzeit verschlüsselt an Alice.

Eine Replay-Attacke ist nun verhindert. Mallory kann jedoch eine alte Antwort des Messgeräts wiedereinspielen, nachdem Alice eine Anfrage gestellt hat (die echte Antwort blockt er ab). Dies liefert Alice einen falschen Messwert. Das Messgerät muss daher die Anfrage-Uhrzeit in der Antwort zurückschicken:

1. Alice sendet eine gehashte Anfrage an das Messgerät. Die Anfrage enthält die aktuelle Uhrzeit.
2. Das Messgerät überprüft den Hashwert sowie die Uhrzeit, führt im positiven Fall eine Messung durch und sendet das Ergebnis zusammen mit der Anfrage-Uhrzeit verschlüsselt an Alice.

Nun nehmen wir an, Mallory habe durch eine Real-World-Attacke den Schlüssel herausbekommen, den Alice und das Messgerät verwenden. Unter dieser Voraussetzung kann Mallory nicht nur alle früheren Messwerte lesen (sofern er diese aufbewahrt hat), sondern auch alle zukünftigen (Known-Key-Attacke). Um einen solchen Angriff zu verhindern, können Alice und das Messgerät ein Schlüsselaustausch-Verfahren wie MQV verwenden. Statt eines Preshared Key setzen sie dabei dauerhafte Diffie-Hellman-Schlüsselpaare ein. Unser Protokoll müssen wir hierzu wie folgt ändern:

1. Alice sendet einen temporären öffentlichen MQV-Schlüssel an das Messgerät.
2. Das Messgerät berechnet den gemeinsamen Sitzungsschlüssel und sendet seinen temporären öffentlichen MQV-Schlüssel an Alice.
3. Alice berechnet ebenfalls den gemeinsamen Sitzungsschlüssel und sendet eine damit gehashte Anfrage an das Messgerät. Die Anfrage enthält die aktuelle Uhrzeit.
4. Das Messgerät überprüft den Hashwert sowie die Uhrzeit, führt im positiven Fall eine Messung durch und sendet das Ergebnis zusammen mit der Anfrage-Uhrzeit verschlüsselt an Alice.

Bei dieser Protokollversion wollen wir es bewenden lassen. In der Praxis kommen jedoch oft weitere Anforderungen dazu, wie etwa folgende:

- Das Messgerät enthält keine oder keine zuverlässige Uhr. Dadurch ist die Nutzung von Zeitstempeln nicht möglich. Vor allem bei kleineren Hardwarekomponenten ist dies ein gängiges Problem. Mir sind zudem bereits mehrere Fälle untergekommen, in denen eine falsch gehende Uhr einen Protokollablauf zum Abbruch gebracht hat.
- Die Messwerte, die das Gerät schickt, müssen verbindlich sein und im Zweifelsfall auch als Beweis vor Gericht eingesetzt werden können. Dies lässt sich durch den Einsatz digitaler Signaturen erreichen.
- Das Messgerät enthält keinen zuverlässigen Zufallsgenerator für die Generierung eines temporären MQV-Schlüssels. Auch dies ist in der Praxis eine keineswegs ungewöhnliche Rahmenbedingung.
- Die Rechenkapazität des Messgeräts ist zu schwach für asymmetrische Kryptografie. Das MQV-Verfahren ist daher nicht nutzbar (RSA und Diffie-Hellman ebenfalls nicht).
- Verkehrsflussanalysen sind zu verhindern. Schon allein die Tatsache, dass Mallory mitbekommt, wann Alice eine Anfrage startet, kann ihm wertvolle Informationen liefern.
- Es gibt neben Alice noch andere Personen, die eine Messung veranlassen können. Hierbei muss nachvollziehbar sein, wer eine entsprechende Nachricht an das Gerät geschickt hat.

Wie ein Protokoll aussehen könnte, das auch diese Zusatzanforderungen erfüllt, können Sie sich selbst überlegen. Unabhängig davon sollte Ihnen nun klar geworden sein, dass das Design eines kryptografischen Protokolls ein hartes Brot ist, bei dem auch Spezialisten oft ins Schwitzen geraten. Dabei ist die eben beschriebene Aufgabenstellung noch vergleichsweise einfach. Kompliziert wird es etwa, wenn mehr als zwei Parteien am Protokoll beteiligt sind oder wenn Alice selbst ein Interesse am Betrügen hat.

20.4.2 Weitere Beispielprotokolle

Eine Sammlung von mehreren Dutzend Konzeptprotokollen beschreibt Bruce Schneier in seinem Werk [Schn96]. Dabei beginnt Schneier mit einfachen Anwendungsfällen wie Schlüsselaustausch oder Public-Key-Verschlüsselung mit mehreren Schlüsseln. Darüber hinaus gibt es zahlreiche Signaturprotokolle wie etwa Proxy-Signaturen, Gruppensignaturen, blinde Signaturen oder simultanes Vertragssignieren. Den Höhepunkt bilden zweifellos Protokolle wie mentales Pokern, Online-Wahlen oder virtuelles Münzwerfen. So interessant diese Protokolle auch sind, für die Praxis haben sie meist keine große Relevanz, weshalb ich sie hier nicht weiter behandle. Interessant sind für uns dagegen Protokolle für das Digital Rights Management und für das Online-Bezahlen. Diese werden in diesem Buch noch eine Rolle spielen.

21 Authentifizierung

In unserem täglichen Leben kommt es immer wieder vor, dass wir überprüfen, ob eine bestimmte Person diejenige ist, die sie vorgibt zu sein. Alice prüft beispielsweise unbewusst die Echtheit von Bob, wenn sie ihn auf der Straße trifft oder mit ihm telefoniert. Alice selbst beweist ihre Echtheit mit einer Geheimnummer am Geldautomaten oder mit ihrem Ausweis bei einer Grenzkontrolle. Wenn Alice einen Brief erhält, dann überprüft sie die Echtheit des Absenders, indem sie die Unterschrift darauf betrachtet. Eine solche Überprüfung der Echtheit (Authentizität) wird **Authentifizierung** genannt.

21.1 Authentifizierung im Überblick

Leider gibt es im Bereich der Authentifizierung viele Begriffe, die in der Literatur unterschiedlich verwendet werden. In diesem Buch bedeutet Authentifizierung – wie bereits gesagt – die Überprüfung, ob jemand derjenige ist, der er vorgibt zu

sein. Der Begriff **Authentifikation** hat die gleiche Bedeutung. Mit **Authentisierung** (oder **Authentikation**) bezeichne ich denselben Vorgang, nur aus Sicht des Überprüften. Wenn Alice also am Bankautomaten Geld abhebt, dann authentisiert sie sich mit ihrer Geheimnummer. Gleichzeitig authentifiziert der Geldautomat Alice. Für besondere Verwirrung sorgt außerdem immer wieder der Begriff **Identifizierung** (beziehungsweise **Identifikation**). Damit ist in diesem Buch ein Vorgang gemeint, bei dem Alice ihren Namen (oder sonst eine Kennung, beispielsweise in Form ihrer Scheckkarte) angibt. Im Normalfall erfolgt also zuerst eine Identifizierung (Alice gibt ihren Namen an) und dann eine Authentifizierung (es wird überprüft, ob es sich wirklich um Alice handelt). Leider hat der Begriff Identifizierung im Zusammenhang in der Biometrie noch eine andere Bedeutung (siehe Abschnitt 21.2.1).

Authentifizierung hat im Allgemeinen nichts mit Computern und schon gar nicht mit Kryptografie zu tun. Vielmehr ist die Authentifizierung ein alltäglicher Vorgang, der uns auch außerhalb der Computerwelt ständig begegnet. Dennoch gibt es sehr wohl Berührungspunkte zwischen der Authentifizierung und der Kryptografie. Die Kryptografie benötigt einerseits die Authentifizierung und stellt ihr andererseits Werkzeuge zur Verfügung, die vor allem in Computernetzen unentbehrlich sind. Schauen wir uns zunächst an, welche Möglichkeiten im Alltag zur Authentifizierung zu Verfügung stehen. Wir können hierbei auch von Authentifizierungsprotokollen sprechen, da ein solcher Vorgang immer mehrere Schritte und mindestens zwei Beteiligte erfordert. Wenn wir annehmen, dass Bob die Echtheit von Alice überprüfen will, dann hat er grundsätzlich drei Möglichkeiten:

- Bob kontrolliert, ob Alice eine bestimmte Information kennt (*etwas, was man weiß*). Beispiele hierfür sind Passwörter, Geheimnummern, geheime Schlüssel und persönliche Informationen. Man spricht in diesem Zusammenhang von einer **Authentifizierung durch Wissen**.
- Bob überprüft, ob Alice einen bestimmten, schwer zu fälschenden Gegenstand besitzt (*etwas, was man hat*). Das wichtigste Beispiel dafür ist ein Ausweis. In diesem Fall nennt man den Vorgang **Authentifizierung durch Besitz**.
- Bob überprüft ein unverwechselbares, schwer fälschbares persönliches Merkmal von Alice (*etwas, was man ist*). Beispiele dafür sind das Aussehen, der Fingerabdruck oder die Unterschrift. Der Fachbegriff ist **Authentifizierung durch Eigenschaft**.

In einem Satz heißt dies: Man authentisiert sich durch etwas, was man weiß, was man hat oder was man ist. Diese drei Varianten der Authentifizierung werden wir in den nächsten Kapiteln näher untersuchen.

21.1.1 Etwas, was man weiß

Beginnen wir unsere Betrachtung mit der Authentifizierung durch Wissen (*etwas, was man weiß*). Diese hat den Vorteil, dass sie besonders einfach ist, denn Alice und Bob benötigen keinen Gegenstand und keine Messgeräte.

Passwörter

Dass man durch die Kenntnis einer Information seine Authentizität beweisen kann, weiß die Menschheit spätestens seit Ali Baba und den 40 Räubern. Die Parole »Sesam, öffne dich« ist nichts anderes als ein **Passwort**. Heute werden Passwörter zwar nicht mehr für Höhleneingänge verwendet, ihre Verbreitung ist dennoch nach wie vor groß. Ob Bankkarten, Computerzugang oder Online-Banking – Passwörter sind allgegenwärtig. Im Folgenden werde ich im Übrigen nicht zwischen Passwort, Passphrase, Geheimnummer und PIN (Personal Identification Number) unterscheiden. Diese Begriffe sind aus Sicht der Kryptografie jeweils das Gleiche. Ein geheimer Schlüssel unterscheidet sich von einem Passwort nur dadurch, dass er meist länger und schwerer zu merken ist.

Ein Nachteil von Passwörtern war auch schon Ali Baba bekannt: Erfährt jemand anderer davon, dann stehen auch diesem die entsprechenden Türen offen. Außerdem werden Passwörter häufig vergessen – zumal es heutzutage eine wahre Schwemme davon gibt. Man denke nur an EC-Karte, Internetzugang, Internetbankkonto, Anrufbeantworter, Autoradio, Handy, Safe, Büroeingangstür, PC-Zugang, Intranet, Datenbank und die Bestellung im Online-Buchladen. Dass sich ein Mensch 20 Passwörter merken muss, ist keine Seltenheit mehr.

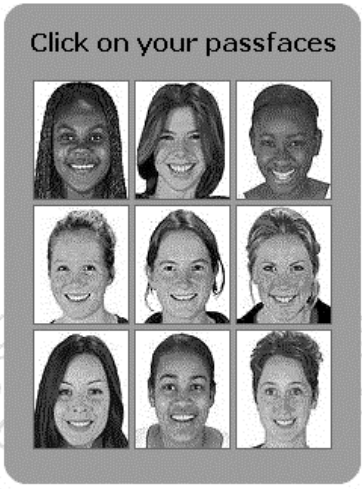


Abb. 21–1 *Passfaces ist ein Authentifizierungssystem , bei dem sich Alice aus mehreren Gesichtern ein vereinbartes aussuchen muss. Der Vorteil hierbei ist, dass Gesichter einfacher zu merken sind als Passwörter.*

Passfaces

Da Passwörter häufig vergessen werden, liegt die Idee nahe, statt diesen Informationen zu verwenden, die sich das menschliche Gehirn leichter merken kann. Eine Möglichkeit hierzu sind **Passfaces**. Diese sehen vor, dass Alice eines von mehreren zur Wahl stehenden Portraitfotos auswählt. Wenn das Verfahren sicher sein soll, dann muss dieser Vorgang mehrfach wiederholt werden. Wählt Alice beispielsweise dreimal eines aus neun dargestellten Gesichtern aus, dann liegt die Wahrscheinlichkeit für einen Zufallstreffer bei 1/729. Der Vorteil von Passfaces – es könnte beispielsweise an Bankautomaten zum Einsatz kommen – liegt darin, dass Menschen sich Gesichter besser merken können als Wörter und Zahlen. Passfaces sind damit ergonomischer als ein Passwort, und ergonomische Aspekte sind in der Kryptografie besonders wichtig, da sie eine höhere Akzeptanz durch die Anwender zur Folge haben. Zudem kann man Gesichter nicht aufschreiben, insbesondere wenn Alice die Fotos bei jeder Abfrage in unterschiedlicher Reihenfolge präsentiert bekommt.

Passgesten

Passgesten sind eine weitere Alternative zu Passwörtern. Als Geste gilt hierbei die Bewegung eines Fingers (oder mehrerer Finger) auf einem Touchscreen oder die Bewegung der Hand vor einer Kamera. So können beispielsweise drei Kreuze und ein Strich die Geheiminformation sein, die Alice zur Authentifizierung eingibt. Für Passgesten spricht, dass sie auf Tablet-PCs, Smartphones und ähnlichen Geräten ohne eine PC-Tastatur benutzerfreundlicher sind als Passwörter. Möglicherweise können sich viele Anwender Passgesten zudem besser merken als Passwörter. Mehr dazu gibt es in [GuSaBS].

Persönliche Informationen

Statt nach einem Passwort kann der Authentifizierende Alice nach einer persönlichen Information fragen, die Mallory nicht bekannt sein kann. Diese kann etwa Alices Lieblingsfilm, die Farbe ihrer Tapete im Wohnzimmer oder der Geburtsname ihrer Mutter sein. Es ist anzunehmen, dass Alice solche Informationen – im Gegensatz zu einem Passwort – nicht vergisst. Da Mallory solche Fragen nicht ohne weiteres beantworten kann, ist diese Form der Authentifizierung bei richtiger Anwendung recht sicher.

21.1.2 Was man hat

Der Besitz eines Gegenstands als Authentizitätsnachweis ist in Form von Ausweisen weit verbreitet. Mit einem Ausweis kann Alice auch die Echtheit des ihr unbekannten Zacharias überprüfen, sofern sie der Ausweisausgabestelle vertraut.

21.1.3 Was man ist

Persönliche Merkmale werden im Alltag am häufigsten zur Authentifizierung verwendet. Im Normalfall ist es schlicht das Aussehen, die Stimme und das Verhalten von Alice, das Bob verwendet, um festzustellen, ob es tatsächlich Alice ist, die er auf der Straße trifft. Interessanter wird die Sache, wenn ein Computer die Authentifizierung durch ein persönliches Merkmal übernehmen soll. Dies wird im folgenden Kapitel ausführlich behandelt.

21.2 Biometrische Authentifizierung

Als **Biometrie** bezeichnet man die Anwendung von Messverfahren auf organische Objekte. Dazu gehört etwa das Messen des Blutdrucks eines Menschen oder die Berechnung der Erntemenge auf einem Getreidefeld. An dieser Stelle interessiert uns, dass die Biometrie zur Authentifizierung eingesetzt werden kann. Es gibt nämlich messbare Größen des menschlichen Körpers (z. B. Fingerabdruck, Musterung der Augenrückwand), die so individuell und unveränderlich sind, dass sie eine Authentifizierung erlauben. Man spricht hierbei von **biometrischer Authentifizierung**. Ein Gerät, das eine biometrische Authentifizierung durchführt, wird als **biometrisches System** (oder **Biometriesystem**) bezeichnet. Die biometrische Authentifizierung gehört in den Bereich der Authentifizierung durch Eigenschaft (was man ist).

21.2.1 Grundsätzliches zur biometrischen Authentifizierung

Im Folgenden gehen wir davon aus, dass die Kryptobank (das führende Geldinstitut in Kryptoland) an ihren Geldautomaten biometrische Authentifizierung einsetzen will. Kundin Alice will diesen Service nutzen.

Referenzwerte

Um Biometrie einzusetzen, muss die Kryptobank ihre Geldautomaten mit dem entsprechenden biometrischen System (zum Beispiel mit einem Fingerabdruckleser und dazugehörender Software) ausstatten. Außerdem muss die Kryptobank für jeden Kunden einen geeigneten Messwert als Vergleichsgröße speichern – dabei kann es sich etwa um die relevanten Parameter von Alices Fingerabdruck handeln. Als Speicherort für den Messwert (sogenannter **Referenzwert**) kommt eine Datenbank infrage. Möglich ist auch, dass jeder Kunde seinen Referenzwert auf der eigenen Smartcard mit sich trägt. Wenn Alice Geld abheben will, wird der Referenzwert mit dem neu gemessenen Wert verglichen. Bevor Alice die neu ausgerüsteten Geldautomaten nutzen kann, muss sie bei der Kryptobank das entsprechende Körpermerkmal (also etwa den Fingerabdruck) vermessen und speichern lassen (**Referenzwertbildung**).

Bei der Authentifizierung selbst gibt es zwei Möglichkeiten: Zum einen kann die Kryptobank vorsehen, dass Alice vor dem Messvorgang ihren Namen angeben muss. Die Software muss dann nach der Messung feststellen, ob der Messwert tatsächlich zu Alice gehört. Diese Variante wird **Verifizierung** genannt (nicht zu verwechseln mit der Verifikation einer digitalen Signatur). Zum anderen kann die Kryptobank ihre Automaten auch so einrichten, dass Alice keine zusätzliche Eingabe machen muss. Die Software findet anschließend allein auf Basis des Messwerts die richtige Person in der Datenbank. Diese Möglichkeit wird als **Identifizierung** bezeichnet (nicht zu verwechseln mit der Identifizierung, die wir in Abschnitt 21.1 definiert haben). Die Identifizierung ist benutzerfreundlicher als die Verifikation, dafür aber weniger performant und weniger sicher.

Fehlerkennungsrate und Fehlabweisungsrate

Mallory kann versuchen, am Geldautomaten sein biometrisches Merkmal einzugeben und hoffen, dass der Geldautomat ihn für Alice hält. Gelingt ihm das, dann spricht man von einer **Fehlerkennung**. Wenn mehrere Menschen versuchen, unberechtigterweise auf Alices Konto zuzugreifen, dann wird der Anteil der Fehlerkennungen als **Fehlerkennungsrate** bezeichnet. Auch der Begriff **False Acceptance Rate** (**FAR**) ist gebräuchlich. Andererseits kann es Alice auch passieren, dass der Automat ihren Messwert nicht anerkennt (dies kann etwa durch einen Messfehler passieren). Ein solcher Vorfall wird **Fehlabweisung** genannt. Der Anteil an Fehlabweisungen an einer größeren Menge von Authentifizierungen durch berechtigte Anwender wird als **Fehlabweisungsrate** bezeichnet. Auch hierfür gibt es englische Bezeichnungen: **False Rejection Rate** oder **FRR**.

Logischerweise sollte bei einem biometrischen System sowohl die Fehlabweisungsrate als auch die Fehlerkennungsrate möglichst gering sein. Dass beide bei Null liegen, ist in der Praxis leider unmöglich. Werte unter einem Prozent sind jedoch realistisch. Wie Abbildung 21–2 zeigt, steigt die Fehlerkennungsrate und sinkt die Fehlabweisungsrate, wenn es beim Vergleich der Messwerte mehr Spiel-

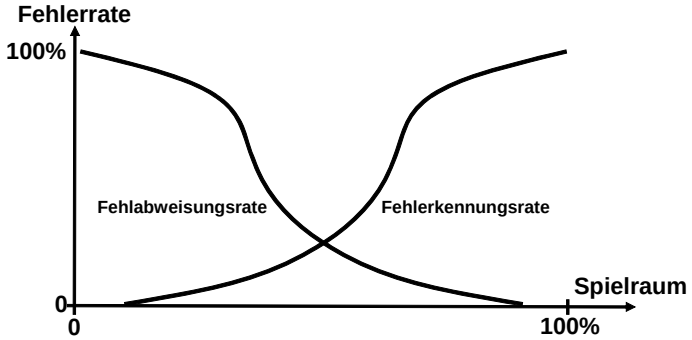


Abb. 21–2 *Verlauf der Fehlerkennungs- und der Fehlabweisungsrate bei einem Biometriesystem*

raum gibt (eine genaue Übereinstimmung gibt es so gut wie nie). Der Spielraum für den Vergleich ist bei den meisten Biometriesystemen konfigurierbar. Die Kryptobank muss also einen Kompromiss finden: Alice sollte möglichst selten ohne Geld nach Hause gehen müssen, und Mallory sollte möglichst selten von Alices Konto Geld abheben können.

21.2.2 Biometrische Merkmale

Die messbaren Größen des menschlichen Körpers, die zur biometrischen Authentifizierung verwendet werden können, heißen **biometrische Merkmale**. Im Folgenden schauen wir uns ein paar Beispiele an.

Fingerabdruckerkennung

Der Fingerabdruck ist aus der Kriminalistik bekannt. Verwendet man einen Fingerabdruckscanner als Messgerät, dann kann der Fingerabdruck sehr gut auch als biometrisches Merkmal eingesetzt werden. Diese Tatsache hat sich wohl inzwischen herumgesprochen, denn die Zahl der auf dem Markt erhältlichen Fingerabdruck-Erkennungssysteme ist nicht mehr zu überblicken (es dürften über 100 sein). Vorteilhaft bei Fingerabdruck-Erkennungssystemen ist die langjährige Erfahrung, die in diesem Bereich vorliegt. Darüber hinaus ist die Technik für den Anwender einfach, und die Kosten bleiben im Rahmen (ein Fingerabdruckscanner ist schon für etwa 100 Euro erhältlich). Von Nachteil bei der Fingerabdruckerkennung sind oftmals hygienische Aspekte.

Abb. 21–3 *Fingerabdruckerkennung an einem Gebäude*

Gesichtserkennung

Bei der **Gesichtserkennung** dient eine Kamera als Messgerät. Mithilfe von Bildverarbeitungstechniken stellt das biometrische System fest, ob das aufgenommene Gesicht mit einem in der Datenbank gespeicherten übereinstimmt. Ohne Zweifel hat die Gesichtserkennung einen gewissen Charme, denn der Ansatz ist sehr naheliegend. Wir Menschen erkennen uns schließlich auch am Gesicht, warum sollte es der Computer anders machen. Zudem gehören geeignete Kameras zu den billigeren Messgeräten. Die Akzeptanz durch die Anwender ist erfahrungsgemäß recht hoch, denn heutzutage ist man es gewohnt, von einer Kamera aufgenommen zu werden. Die Fehlerkennungsrate der Gesichtserkennung liegt bei guten Implementierungen verglichen mit anderen Biometriesystemen im Mittelfeld. Störend ist, dass sich das Gesicht eines Menschen mit der Zeit verändert. Bärte und Brillen wirken ebenfalls oft hinderlich. So ist mit einer hohen Fehlabweisungsrate zu rechnen, wenn das Gesichtserkennungssystem nicht lernfähig (adaptiv) ist.

Unterschriftenerkennung

Bei der **Unterschriftenerkennung** werden die Bewegungen eines Stifts bei der Anfertigung einer Unterschrift gemessen. Als Messgerät kann eine mit Sensoren bestückte Unterlage oder ein spezieller Sensorstift (oder auch beides zusammen) verwendet werden. Eine Unterschrift ist schwer zu fälschen, insbesondere wenn auch die Schreibgeschwindigkeit und die Aufdruckstärke in die Messung einbezogen wird. Da Alices Unterschrift jedoch auch erheblich variieren kann, muss der Spielraum für die Erkennung relativ groß ausfallen. Die hohe Akzeptanz bei den Anwendern und die kostengünstige Realisierung machen die Unterschriftenerkennung zu einer interessanten Biometrievariante.

Stimmerkennung

Bei der **Stimmerkennung** wird die menschliche Stimme aufgezeichnet, wobei ein Mikrofon als Messgerät dient. Mit geeigneten Algorithmen können aus einer solchen Aufzeichnung genügend charakteristische Merkmale gezogen werden, um die Stimme einem Menschen zuzuordnen. Damit ein Stimmerkennungssystem nicht mit einem Tonband überlistet werden kann, wird oftmals ein Wort vorgegeben, das Alice aussprechen muss. Die Stimmerkennung ist billig und einfach für den Anwender. Prädestiniert ist diese Methode für die Anwendung am Telefon. Der größte Nachteil ist die recht hohe Fehlabweisungsrate, die beispielsweise bei Heiserkeit erheblich steigt. Außerdem ergeben sich in der Praxis oft Akzeptanzprobleme: Wer spricht schon gerne mit einer Maschine?

Iriserkennung

Die **Iriserkennung** macht sich zunutze, dass der farbige Kranz um die Pupille des Auges (Iris oder Regenbogenhaut genannt) eine sehr individuelle Musterung aufweist. Als Messgerät wird bei der Iriserkennung eine hoch auflösende Digitalkamera benötigt, die eine Aufnahme der Augen anfertigt. Die Iriserkennung gehört zweifellos zu den interessantesten Biometrievarianten. Die Fehlerkennungsrate ist bei guten Implementierungen sehr niedrig und wird nur noch von der Retinaerkennung unterboten. Darüber hinaus ist das Verfahren für den Anwender unkompliziert. Erfahrungsgemäß gibt es jedoch Akzeptanzprobleme, die meist darauf zurückzuführen sind, dass die Iriserkennung immer wieder mit der Retinaerkennung verwechselt wird.

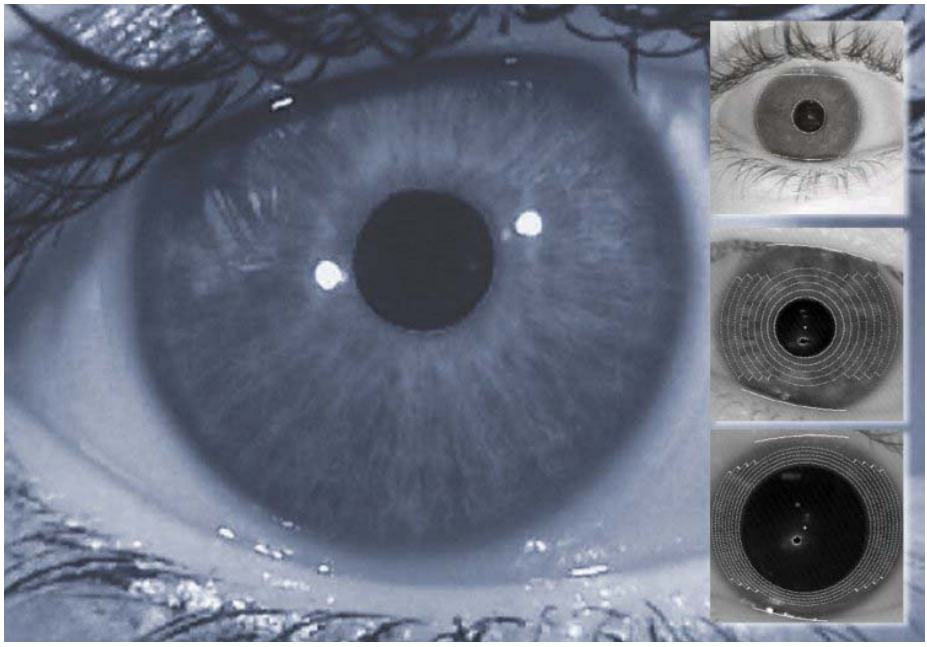


Abb. 21–4 *Die Iris eines Menschen ist individuell verschieden. Sie kann daher als biometrisches Merkmal verwendet werden.*

Die Herstellerfirmen von Iriserkennungssystemen behaupten, ein Überlisten mit einem Foto sei nicht möglich, weil bei der Messung Bewegungen der Iris registriert werden, die auf einem Foto nicht gegeben sind. Da solche Größenveränderungen durch einen Lichtimpuls angestoßen werden, dürfte auch ein Videofilm nichts nützen.

Weitere biometrische Merkmale

Weitere Biometrievarianten sind die Handerkennung, die **Erkennung des Tippverhaltens**, die Geruchserkennung, die Ohrerkennung, die Retinaerkennung, die Analyse der Körpermotorik, die Vermessung von Gehirnströmen oder die EKG-Erkennung. Auch die Kombination mehrerer Merkmale ist möglich. Noch in den Bereich der Fantasie fällt dagegen die DNA-Analyse. Diese hätte zwar eine Fehlerkennungsrate von praktisch Null zu bieten. Das mögliche Messgerät – ein mit Sensoren ausgestatteter Spucknapf – wäre allerdings etwas unappetitlich.

21.2.3 Fazit

Zu den Nachteilen der Biometrie gehört die Tatsache, dass diese letztendlich immer auf Messungen basiert, die prinzipiell auch gefälscht werden können. Egal ob Gesicht, Fingerabdruck oder Stimme – mit entsprechend großem Aufwand (Foto, Monitor, Wachsmodell, Tonband) kann Mallory das benötigte biometrische Merkmal so gut wie immer imitieren. Die Produkthersteller können lediglich versuchen, die praktische Umsetzung eines solchen Angriffs zu erschweren. Dabei fällt besonders ins Gewicht, dass sich ein biometrisches Merkmal nicht wie ein Passwort austauschen lässt. Alices Fingerabdruck bleibt unverändert, auch wenn Mallory es geschafft hat, ihn nachzumachen.

Abb. 21–5 *Iriserkennungssystem der Firma Byometric*

21.3 Authentifizierung in Computernetzen

Nach unserem Ausflug in die Authentifizierung im Allgemeinen betrachten wir nun, wie eine Authentifizierung in einem Computernetz (speziell im Internet) ablaufen kann. Wie Sie sich vorstellen können, spielt das Thema Authentifizierung gerade in diesem Zusammenhang eine wichtige Rolle, weil sich in einem anonymen Netzwerk oft jeder für jeden ausgeben kann. Wir gehen im Folgenden davon aus, dass Alice über das Internet mit der Kryptobank in Kontakt steht. Bevor es einen Nachrichtenaustausch gibt, will die Bank sicher sein, dass sich am anderen Ende der Leitung tatsächlich Alice (und nicht Bösewicht Mallory) befindet. Die Bank will also Alice authentifizieren. Möglicherweise will aber auch Alice wissen, ob sie es tatsächlich mit der Bank zu tun hat oder ob etwa Mallory sich als Kryptobank ausgibt. In manchen Fällen will also auch Alice die Bank authentifizieren. Ist dies der Fall, dann sprechen wir von **gegenseitiger Authentifizierung.** Zwar gelten im Internet ganz andere Regeln als im richtigen Leben, doch die Authentifizierung können wir dafür nicht neu erfinden. Daher gilt auch im Netz: Entscheidend ist etwas, was man ist, was man weiß oder was man hat.

21.3.1 Passwörter im Internet

Das Authentifizierungsmittel schlechthin im Internet ist das Passwort. Vom Online-Banking über den Online-Einkauf bis zum E-Mail-Abruf geht nichts ohne Passworteingabe.

Einfache Passwort-Authentifizierung

Wenn sich Alice und die Kryptobank keine größere Mühe machen, ist der Protokollablauf einer Passwort-Authentifizierung denkbar einfach: Alice sendet ihr Passwort über das Netz an die Bank. Diese überprüft die Korrektheit des Passworts. Ist es echt, dann ist Alice erfolgreich authentifiziert. Es dürfte klar sein, dass diese einfachste Form der Passwortabfrage für Mallory ein gefundenes Fressen ist. Da er das Netzwerk abhören kann, gelangt er ohne Mühe an Alices Passwort und kann es nach Belieben missbrauchen. Wir sollten uns daher nach besseren Methoden umsehen.

Einmal-Passwörter

Eine naheliegende Möglichkeit, dem Abhören von Passwörtern zu begegnen, sind **Einmal-Passwörter.** Wie der Name schon sagt, müssen sich Alice und die Kryptobank in diesem Fall gleich auf mehrere Passwörter einigen, von denen jedes nur einmal verwendet wird. Mallorys Hoffnung, ein erspähtes Passwort ein zweites Mal zu verwenden, ist damit von vornherein zunichte gemacht. Einmal-Passwörter kennen Sie vielleicht vom Online-Banking. Will Alice einen solchen Service bei der Kryptobank nutzen, so erhält sie von dieser eine Liste mit Passwörtern (meist

Transaktionsnummern (**TANs**) genannt). Jedesmal, wenn sie sich mit der Kryptobank verbindet und eine Transaktion startet, muss sie eine TAN aus ihrer Liste verwenden. Danach wird die TAN ungültig. Sind alle TANs aufgebraucht, muss Alice sich bei der Kryptobank eine neue Liste holen.

Ein Einmal-Passwort-Verfahren lässt sich sehr effektiv mit einer kryptografischen Hashfunktion realisieren. Dazu generiert die Kryptobank eine Zufallszahl p_0. Darauf wendet sie eine kryptografische Hashfunktion an und erhält so die TAN p_1, aus der sie durch erneutes Hashen p_2 erzeugt. Dies führt die Kryptobank mehrfach durch, bis sie beispielsweise p_{10} erhält. p_0 bis p_9 übergibt die Kryptobank als TAN-Liste an Alice und merkt sich lediglich p_{10}. p_9 ist Alices erste TAN. Die Bank kann deren Richtigkeit überprüfen, indem sie die kryptografische Hashfunktion auf p_9 anwendet und die Ergebnisse vergleicht. Die nächste TAN ist p_8, die wiederum p_9 ergeben muss, wenn sie durch die sichere Hashfunktion geschickt wird. Diese Vorgehensweise zur Generierung von Einmal-Passwörtern ist in einem RFC unter dem Namen **One-Time Password** (**OTP**) standardisiert [RFC2289]. Die US-Firma Bellcore hat das Verfahren für alle gängigen Betriebssysteme implementiert, die Implementierung trägt den Namen **S/Key**. Oft wird dieser Name auch für das Verfahren an sich verwendet. Die Bellcore-Implementierung benutzt wahlweise eine der (inzwischen veralteten) kryptografischen Hashfunktionen MD4 oder MD5.

Der OTP-Ansatz hat im Wesentlichen zwei Vorteile: Die Kryptobank muss keine große TAN-Liste speichern, und Mallory findet keine verwertbaren Passwörter, wenn er in den Bankrechner eindringt. Ein Nachteil von Einmal-Passwörtern ist jedoch offensichtlich: Alice muss ständig eine Passwortliste mit sich herumtragen. Wird diese von Mallory geklaut oder verliert Alice sie, dann können ihre Ersparnisse schnell zusammenschrumpfen.

Challenge-Response-Verfahren

Eine Alternative zu Einmal-Passwörtern besteht darin, dass Alice ihr Passwort (sie benötigt in diesem Fall nur eines) nicht übers Netz schickt. Stattdessen übergibt sie stets nur einen kryptografischen Hashwert des Passworts an die Kryptobank. Damit der kryptografische Hashwert nicht jedesmal gleich ist (und damit von Mallory wiederverwendet werden kann), muss jedoch noch eine weitere Information in den Hashwert einfließen. Diese Zusatzinformation kann etwa die aktuelle Zeit oder einfach eine Zufallszahl sein. Die Zusatzinformation muss nicht geheim bleiben, Alice kann sie übers Netz schicken (siehe Abbildung 21–6).

Am sichersten ist die beschriebene Vorgehensweise, wenn die Zusatzinformation direkt von der Bank kommt. Dazu sendet die Kryptobank in einer Protokollnachricht einen beliebigen Wert (**Challenge**) an Alice. Aus diesem Wert und dem Passwort bildet sie dann mit einer kryptografischen Hashfunktion die Antwort (**Response**), die sie zurück an die Kryptobank sendet. Ein Verfahren, das nach diesem Prinzip abläuft, wird **Challenge-Response-Verfahren** genannt.

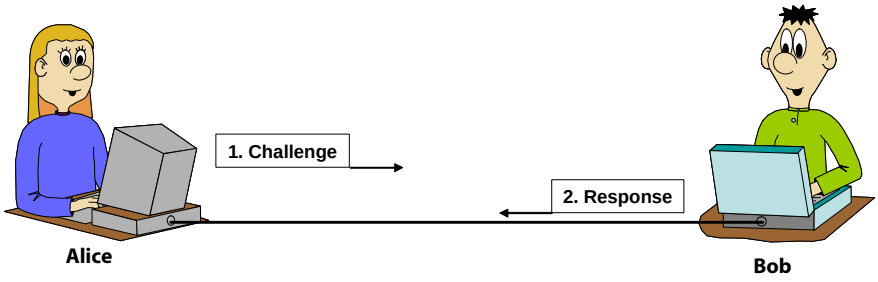


Abb. 21–6 *Bei einem Challenge-Response-Verfahren hat Mallory keine Möglichkeit, das Passwort abzuhören.*

Challenge-Response-Verfahren gibt es in unzähligen Variationen. Eine davon sieht vor, dass Alice einen kleinen Zettel mit einer Tabelle ausgehändigt bekommt (**Passworttabelle**). Will die Kryptobank sie authentifizieren, dann fordert diese Alice auf, beispielsweise den Inhalt der fünften Spalte in der dritten Reihe einzugeben (Challenge). Die Response liest Alice aus der Tabelle – sie muss keinen Hashwert berechnen. Eine Variante dieses Vorgehens besteht darin, dass Alice einen Text bei sich hat und die Bank sie auffordert, beispielsweise das zehnte Wort auf Seite 5 zu nennen. Der Vorteil dieser Methoden liegt darin, dass Alice ihr Passwort nicht ohne weiteres aufschreiben oder weitergeben kann (mit einem Fotokopierer geht es aber meistens doch). Dafür ist ein solches Vorgehen etwas unhandlich, und Alice muss stets die entsprechende Tabelle oder den entsprechenden Text mit sich tragen.

OTP-Tokens

Ein Challenge-Response-Verfahren setzt häufig voraus, dass Alice eine kryptografische Hashfunktion anwendet. Diese Anforderung stößt in der Praxis oft auf Probleme. Wenn Alice nämlich eine einfache Client-Software einsetzt, dann kann diese möglicherweise keine Hashwerte berechnen, da eine solche Funktion nicht vorgesehen ist. Auch das Auslagern einer solchen Berechnung an eine Smartcard ist meist nicht möglich, da einfache Clients in den seltensten Fällen die dazu notwendige Schnittstelle bereitstellen. Die Eingabe eines Passworts (bei dem es sich um eine Response handeln kann) wird dagegen von vielen Clients unterstützt, eventuell kann der Client sogar eine Challenge einblenden.

Ein eleganter Ausweg aus diesem Dilemma besteht darin, dass Alice eine spezielle Hardware nutzt, um die Response zu berechnen. Eine solche Hardware wird oft als *Token* bezeichnet. In diesem Buch will ich den Begriff **OTP-Token** verwenden (OTP steht hierbei für One-Time Password), um Verwechslungen mit anderen Objekten zu vermeiden, die ebenfalls Token genannt werden. Ein OTP-Token sieht meist aus wie ein kleiner Taschenrechner (mit Zehnertastatur und Display) und wird oft in Form eines Schlüsselanhängers oder im Scheckkartenformat angeboten. Auf dem Token ist ein Passwort (in diesem Fall spricht man

eher von einem Schlüssel) gespeichert. Eine Authentifizierung mit einem OTP-Token kann wie folgt aussehen (wir gehen davon aus, dass Alice auf ihr Online-Konto bei der Kryptobank zugreifen will):

1. Die Kryptobank schickt eine Challenge an Alice. Die Challenge ist in diesem Fall meist eine Zahl, beispielsweise 475099. Sie wird am Bildschirm angezeigt.
2. Alice liest die Challenge vom Bildschirm ab und tippt sie in ihr OTP-Token. Zusätzlich gibt sie eine Geheimnummer ein, die verhindern soll, dass Mallory ein gestohlenes OTP-Token missbrauchen kann.
3. Das OTP-Token berechnet mit der eingebauten Hashfunktion aus Challenge und Schlüssel die Response und zeigt diese im Anzeigefeld an.
4. Alice liest die Response ab und tippt sie in ihren PC ein. Die Response geht nun an die Kryptobank.
5. Die Kryptobank (ihr ist der Schlüssel auf dem Token bekannt) überprüft die Response. Im positiven Fall kann Alice auf ihr Konto zugreifen.

Abb. 21–7 *Die SecurID-Karte (links als Schlüsselanhänger, rechts im Scheckkarten-Format) ist ein OTP-Token.*

Das bekannteste und erfolgreichste OTP-Token-Produkt ist die **SecurID-Karte** der Firma RSA Security. In seiner ursprünglichen Version funktioniert dieses etwas anders als gerade beschrieben. So hat die (ursprüngliche) SecurID-Karte keine Tastatur und nimmt dementsprechend auch keine Challenge entegegen. Stattdessen blendet die SecurID-Karte jede Minute eine neue Response ein. Man kann daher sagen: Die aktuelle Uhrzeit (auf die Minute genau) ist in diesem Fall die Challenge. Der Chip berechnet die Response jedoch nicht direkt aus der Uhrzeit, sondern enthält einen Pseudozufallsgenerator, der von einem Startwert ausgehend jede Minute fortgeschaltet wird. Aus dem aktuellen Wert ergibt sich jeweils die eingeblendete Response. Es versteht sich von selbst, dass der Startwert für jede SecurID-Karte unterschiedlich sein muss. Wie die Fortschaltfunktion aussieht, wird von RSA Security geheim gehalten.

Neben der Variante ohne Tastatur gibt es inzwischen auch SecurID-Karten mit einer solchen. Neben RSA Security mischen noch einige andere Anbieter auf dem lukrativen OTP-Token-Markt mit, beispielsweise Kobil, Secure Computing und Vasco. Längst werden OTP-Tokens in zahlreichen Variationen angeboten:

- Wie bereits erwähnt, nehmen manche OTP-Tokens eine Challenge entgegen, andere nutzen dagegen die Uhrzeit als Challenge.
- Manche OTP-Tokens enthalten zusätzlich oder anstatt der Tastatur einen Scanner, der die Challenge vom Bildschirm ablesen kann. Alice kann sich dadurch das Eintippen der Challenge sparen. Auf dem Bildschirm wir die Challenge hierbei meist nicht mithilfe von Ziffern dargestellt, sondern über ein blinkendes Feld mithilfe eines Blink-Codes.
- Manche Tokens sehen vor, dass Anwenderin Alice eine Smartcard (z.B. ihre Bankkarte) einstecken muss. Die Karte übernimmt dann die Berechnung der Challenge. In diesem Fall ist die Karte das Authentifizierungsmittel, während das OTP-Token austauschbar ist.

Obwohl (oder gerade weil) OTP-Tokens vergleichsweise einfache Produkte sind, sind sie am Markt schon seit über 15 Jahren äußerst erfolgreich. Nicht nur die Firma RSA Security hat damit ein Vermögen verdient. Mit OTP-Tokens hat sich – wie so oft in der Kryptografie – eine pragmatische Lösung durchgesetzt. OTP-Tokens sind zwar nicht so sicher und so leistungsfähig wie Smartcards, aber sie erfüllen ihren Zweck, sie sind einfach zu handhaben, und ihr Funktionsprinzip ist auch für einen Laien zu verstehen. Die Nachfrage wird daher sicherlich auch in den kommenden Jahren groß sein.

Abb. 21–8 *Eine Auswahl von OTP-Tokens der Firma Secure Computing*

21.3.2 Authentifizierung mit asymmetrischen Verfahren

Die bisher vorgestellten Authentifizierungsverfahren setzen alle voraus, dass sich Alice und die Kryptobank auf ein Passwort oder einen Schlüssel geeinigt haben. Besser ist es natürlich, wenn dieser Schritt nicht notwendig ist. Für diese Anforde-

rung gibt es bekanntlich die asymmetrische Kryptografie, und diese lässt sich auch für Challenge-Response-Verfahren nutzen.

Challenge-Response mit digitaler Signatur

Eine einfache Möglichkeit zur Challenge-Response-Authentifizierung besteht darin, digitale Signaturen einzusetzen. Das entsprechende Protokoll sieht etwa so aus (Alice authentisiere sich gegenüber einem Server, der ihren öffentlichen Schlüssel kennt):

1. Der Server schickt eine Challenge an Alice.
2. Alice signiert die Challenge und sendet sie an den Server zurück.
3. Der Server verifiziert die Signatur. Ist sie echt, dann ist die Authentifizierung positiv verlaufen.

Da Alice einen privaten Schlüssel (also quasi ein Passwort) verwendet, haben wir hier einen weiteren Fall einer Authentifizierung durch Wissen. Speichert Alice ihren privaten Schlüssel auf einer Smartcard, dann wird daraus eine Authentifizierung durch Besitz.

Challenge-Response mit asymmetrischer Verschlüsselung

Auch mit einem asymmetrischen Verschlüsselungsverfahren können Alice und ein Server ein Challenge-Response-Protokoll abarbeiten. Wenn wir wieder davon ausgehen, dass der Server Alices öffentlichen Schlüssel kennt, dann sieht dies wie folgt aus:

1. Der Server generiert einen Zufallswert und verschlüsselt ihn mit Alices öffentlichem Schlüssel. Das Ergebnis sendet er als Challenge an Alice.
2. Alice entschlüsselt die Challenge mit ihrem privaten Schlüssel. Das Ergebnis sendet sie als Response an den Server.
3. Der Server prüft die Korrektheit der Response. Im positiven Fall ist die Authentifizierung erfolgreich.

Diese Variante hat gegenüber der Challenge-Response-Authentifizierung mit digitaler Signatur zwei Nachteile: Zum einen muss die Challenge in diesem Fall geheim bleiben; zum anderen muss der Server gleich am Anfang eine Public-Key-Operation durchführen, was Mallory eine Denial-of-Service-Attacke erleichtert. Es gibt jedoch auch einen interessanten Vorteil: Alice und Bob können die Response als geheimen Schlüssel für ein symmetrisches Verfahren verwenden. Die Challenge-Response-Authentifizierung mit asymmetrischer Verschlüsselung schließt also einen Schlüsselaustausch mit ein.

Passwortbasierte Authentifizierung mit asymmetrischer Kryptografie

Alice und Bob haben ein Passwort vereinbart. Mit diesem wollen sie eine gegenseitige Authentifizierung durchführen und dabei gleich einen geheimen Schlüssel austauschen. Kein Problem, möchte man meinen, denn mithilfe des Passworts können die beiden ein Challenge-Response-Verfahren zur Authentifizierung abarbeiten, und den benötigten Schlüssel leiten sie mit einer kryptografischen Hashfunktion vom Passwort ab. Diese Vorgehensweise hat jedoch einen Nachteil: Sie ist anfällig gegenüber einem Wörterbuch-Angriff. Wenn Mallory den Datenaustausch zwischen Alice und Bob abhört, dann kann er sich die verschlüsselte oder gehashte Challenge herauspicken und per Durchprobieren ermitteln, welches Passwort zur Bildung dieser Challenge verwendet wurde. Wenn Alice und Bob ein Passwort gewählt haben, das in einem Wörterbuch zu finden ist, dann hat dieser Angriff eine gute Erfolgschance.

Wir stellen also einmal mehr fest: Wenn die Sicherheit eines Krypto-Systems allein an einem Passwort hängt, dann kann es gefährlich werden. Alice und Bob können einen Wörterbuch-Angriff vermeiden, indem sie einen zufälligen 128-Bit-Schlüssel anstatt eines Passworts verwenden. Doch welcher Anwender will oder kann sich schon einen 128-Bit-Schlüssel merken?

Zum Glück gibt es eine zuverlässige Möglichkeit, Wörterbuch-Angriffe in passwortbasierten Authentifizierungsprotokollen zu vermeiden. Der Trick dabei ist, dass zusätzlich zum Passwort ein asymmetrisches Krypto-Verfahren verwendet wird. Dies klingt zunächst widersinnig, denn asymmetrische Verfahren kommen ansonsten gerade dann zum Einsatz, wenn Alice und Bob keine gemeinsame Geheiminformation (in diesem Fall ein Passwort) besitzen. Der vorliegende Fall ist jedoch eine Ausnahme. Hier wird die asymmetrische Kryptografie eingesetzt, um ein bereits vereinbartes Passwort sicherer zu machen. Oder aus anderer Perspektive betrachtet: Das Passwort dient in diesem Fall dazu, einen Schlüsselaustausch mit einem asymmetrischen Verfahren durch eine zusätzliche Authentifizierung gegen eine Man-in-the-Middle-Attacke abzusichern. Man spricht hierbei auch vom **passwortbasierten Schlüsselaustausch**.

Der passwortbasierte Schlüsselaustausch ist ein vergleichsweise junges Thema in der Kryptografie. Es handelt sich dabei wieder einmal um einen pragmatischen Ansatz, den Kryptografen lange vernachlässigt haben, da er in der Theorie unnötig ist (in der Theorie setzen Alice und Bob keine erratbaren Passwörter ein). Die Praxis sieht jedoch anders aus, und deshalb hat der passwortbasierte Schlüsselaustausch in den letzten Jahren schnell eine erhebliche Bedeutung erlangt.

Das einfachste gängige Protokoll für den passwortbasierten Schlüsselaustausch hat den Namen **SPEKE** (**Simple Password Exponential Key Exchange**). Bei diesem Protokoll handelt es sich um eine leicht abgewandelte Variante des Diffie-Hellman-Schlüsselaustauschs. Um es anzuwenden, einigen sich Alice und Bob auf ein Passwort w und eine Primzahl p. Mit einer kryptografischen Hashfunktion h

generieren sie die Zahl $g = h(w)^2$. g und p haben dieselbe Funktion wie beim Diffie-Hellman-Schlüsselaustausch. Das Quadrieren des Hashwerts bewirkt, dass g ein Generator der Gruppe $Z(p,\cdot)$ ist (man kann zeigen, dass dies für jeden Wert von $h(w)$ gilt, außer für 1 und p-1; diese beiden Werte dürfen nicht verwendet werden). Außerdem wählt Alice eine natürliche Zahl $x<p$, Bob wählt eine natürliche Zahl $y<p$. Auch diese Zahlen haben dieselbe Aufgabe wie bei Diffie-Hellman. Sie werden jedoch nicht dauerhaft verwendet, sondern für jede Protokollabarbeitung neu generiert. Nun beginnt folgender Ablauf, den Sie von Diffie-Hellman kennen:

1. Alice berechnet die Zahl $a=g^x \pmod p$. Sie schickt a an Bob.
2. Bob berechnet die Zahl $b=g^y \pmod p$. Er schickt b an Alice.
3. Alice berechnet aus dem erhaltenen b die Zahl $k_1:=b^x \pmod p$.
4. Bob berechnet aus dem erhaltenen a die Zahl $k_2:=a^y \pmod p$.

Jetzt gilt $k_1 = k_2$, weshalb wir dafür k schreiben. k ist ein geheimer Schlüssel, den Alice und Bob für ein Challenge-Response-Verfahren oder eine sonstige Authentifizierung nutzen können. Mallory kann k nur erraten, wenn er neben dem Passwort auch x oder y kennt. Ein Wörterbuch-Angriff zur Ermittlung des Passworts ist ohne Kenntnis von x oder y ebenfalls nicht möglich.

Neben SPEKE gibt es noch zahlreiche weitere Protokolle für den passwortbasierten Schlüsselaustausch. Dazu gehören Konzeptprotokolle mit Namen wie **EKE**, **DH-EKE**, **B-SPEKE** und **PAK**. Ein entsprechendes Netzwerkprotokoll ist unter dem Namen **Secure Remote Password Protocol** (**SRP**) spezifiziert [RFC2945]. SRP führt wie SPEKE einen Diffie-Hellman-Schlüsselaustausch durch, wobei jedoch nicht die Basis g, sondern der Exponent von einem Passwort abhängt. Ausführliche Informationen zu SRP gibt es in [Wu98]. Ein auf EKE basierendes Netzwerkprotokoll findet sich außerdem in [RFC6124]. Ein weiteres Netzwerkprotokoll mit ähnlichem Zweck ist **PACE**, das im Zusammenhang mit elektronischen Ausweisen eingesetzt wird [BSI-03110, Schm09/2].

21.3.3 Biometrie in Computernetzen

Der Einsatz von Biometrie im Netz bringt ähnliche Probleme mit sich wie ein Passwort. Wenn Alice einen biometrischen Messwert über das Netz zur Kryptobank schickt, dann kann Mallory diesen abfangen und für eine Replay-Attacke nutzen. Da ein biometrischer Messwert meist nicht genau mit dem Referenzwert übereinstimmt, bringt es wenig, eine kryptografische Hashfunktion einzusetzen. Biometrie wird daher nahezu nie in direkter zur Authentifizierung in Computernetzen verwendet. Indirekt spielt die Biometrie in diesem Zusammenhang aber dennoch eine Rolle, denn sie kann dazu beitragen, den Zugang zu einem geheimen Schlüssel zu kontrollieren, den Alice auf ihrer Festplatte oder auf einer Smartcard gespeichert hat.

22 Verteilte Authentifizierung

Alices Arbeitgeber Krypt & Co. betreibt drei Server-Anwendungen: ein Webportal, eine Datenbank und ein Zeiterfassungssystem. Auf alle drei kann Alice von ihrem PC aus über ein lokales Netz zugreifen. Selbstverständlich ist die Kommunikation zwischen PC und Server stets verschlüsselt, und Alice muss sich vor der Nutzung jeweils authentisieren. Leider ist das etwas umständlich: Das Webportal fordert ein mindestens achtstelliges Passwort mit Sonderzeichen, die Datenbank ist durch eine Smartcard gesichert, und die Zeiterfassung liest ein vierstelliges Passwort ein. Alice muss sich also mit drei unterschiedlichen Authentifizierungsmethoden herumschlagen und sich dabei zwei verschiedene Passwörter merken. Da IT-Anwender grundsätzlich faul und desinteressiert sind, kann es schnell passieren, dass Alice ihre Passwörter aufschreibt oder ab und zu eines vergisst. Die Folge sind unnötige Sicherheitsmängel und zusätzliche Arbeit für die IT-Abteilung. Man kann also folgende Forderungen formulieren: Wenn mehrere Server jeweils ihre eigene Authentifizierung durchführen, dann sollte nicht jeder sein

eigenes Süppchen kochen. Stattdessen ist eine geeignete Abstimmung sinnvoll. Welche Möglichkeiten es hierfür gibt, wollen wir uns im Folgenden anschauen.

22.1 Credential-Synchronisation

Um das genannte Problem zu lösen, kann Krypt & Co. dafür sorgen, dass die Authentifizierung an allen drei Server-Anwendungen gleich abläuft. Dies ist der Fall, wenn Alice an allen drei Servern dasselbe Passwort verwenden oder dieselbe Smartcard einsetzen kann. Da in der Praxis noch immer Passwörter die gängigste Authentifizierungtechnik sind, wird eine solche Maßnahme meist als **Passwort-Synchronisation** bezeichnet. Sachgemäßer ist allerdings der Begriff **Credential-Synchronisation**, da dieser auch Smartcards, Smart Tokens, Biometrie und andere Techniken einschließt.

Die Credential-Synchronisation ist ein Teilgebiet des Identity Management (siehe Abschnitt 28.1.1). Der IT-Administrator von Krypt & Co. kann eine solche manuell vornehmen, indem er für Alice auf jedem Server dieselbe Authentifizierungsmethode und gegebenenfalls dasselbe Passwort konfiguriert. Es gibt auch spezielle Software für die Credential-Synchronisation. Zudem gibt es die Möglichkeit, Credential-Synchronisation an einen zentralen Server auszulagern. Diesem Ansatz folgen Standards wie RADIUS und DIAMETER (siehe Abschnitt 22.4).

Es versteht sich von selbst, dass die Credential-Synchronisation ein zweischneidiges Schwert ist. Wenn Alice für mehrere Server die gleiche Art der Authentifzierung einsetzen kann, dann muss auch Mallory nur eine Hürde überwinden, um die Sicherheit mehrerer Server auszuhebeln. Andererseits ist ein dreifach ausgeführter Authentifizierungsvorgang, der sicher und benutzerfreundlich abläuft, allemal besser als drei unterschiedliche Vorgänge, die Alice nerven und sie beispielsweise zum Aufschreiben ihrer Passwörter verleiten.

22.2 Single Sign-On

Während sich Alice bei der Credential-Synchronisation gegenüber jedem Server einzeln authentifizieren muss (wenn auch auf gleiche Weise), geht das **Single Sign-On** (**SSO**) einen Schritt weiter. Single Sign-on sieht vor, dass sich Alice nur einmal authentifiziert und danach ohne weitere Aktion auf alle Server zugreifen kann. Single Sign-On macht Alice die Arbeit also noch etwas leichter als die Credential-Synchronisation.

Beim Single Sign-On gelten ähnliche Sicherheitsüberlegungen wie bei der Credential-Synchronisation: Einerseits profitiert Mallory davon, wenn er mit einer falschen Authentifizierung gleich mehrere Server angreifen kann; andererseits sind mehrere unterschiedliche Authentifizierungen unpraktisch und schon allein deshalb ein Sicherheitsrisiko. Single Sign-On und Credential-Synchronisation sind daher Zugeständnisse an die Erfahrung, dass Benutzerfreundlichkeit in

der IT-Sicherheit eine wichtige Rolle spielt. Viele SSO-Projekte werden ohnehin nicht in erster Linie aus Sicherheitsgründen durgeführt – stattdessen lautet die Überlegung, dass durch Single Sign-On vergessene Passwörter und verlorene Smartcards seltener werden, was Kosten spart. Unabhängig davon gilt: Wenn SSO eingeführt wird, dann sollte die eine Hürde, die Mallory in den Weg gestellt wird, möglichst hoch sein. Es lohnt sich also, Smartcards anstatt den deutlich weniger sicheren Passwörtern zu verwenden.

Es gibt mehrere unterschiedliche Ansätze zur Realisierung von Single Sign-On. Wenn wir diese im Folgenden anschauen, gehen wir weiterhin davon aus, dass Alice als Mitarbeiterin von Krypt & Co. Zugriff auf drei Server hat, wobei vor der SSO-Einführung unterschiedliche Authentifizierungsmethoden zum Einsatz kommen.

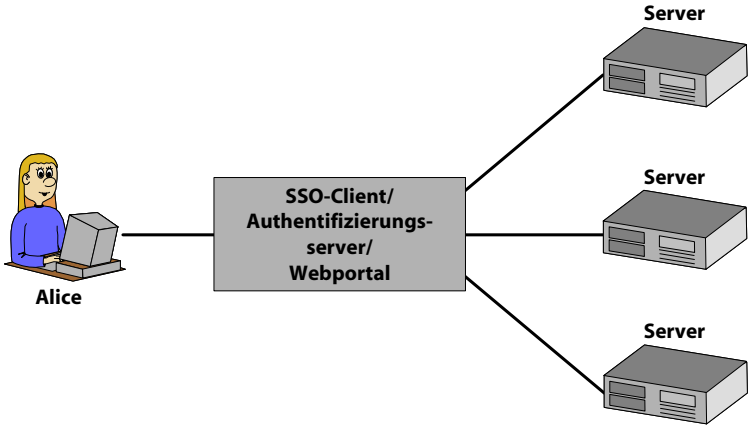


Abb. 22–1 *Single Sign-On ermöglicht es Alice, sich mit einem Login-Vorgang an mehreren Servern anzumelden.*

22.2.1 Lokales SSO

Die derzeit bedeutendste Form des Single Sign-On ist das **lokale SSO**. Dieses erfordert in seiner gängigen Form, dass alle von Alice genutzten Server ein Passwort zur Authentifizierung vorsehen. Alice hat auf ihrem PC einen speziellen Client (SSO-Client) installiert, der alle erscheinenden Login-Fenster automatisch abgreift und mit Alices Benutzernamen sowie dem Passwort ausfüllt. Um dies leisten zu können, muss der SSO-Client den Benutzernamen und alle Passwörter gespeichert haben. Auch die Einbeziehung eines zusätzlichen Servers, auf dem die Passwörter gespeichert sind und der vom SSO-Client angesprochen wird, ist möglich.

Bevor der SSO-Client eine Anmeldung an einem Server vornimmt, muss sich Alice ihm gegenüber authentisieren. Wenn der SSO-Client einen Server für das Speichern der Passwörter nutzt, dann erfolgt die Authentisierung gegenüber diesem. Das Authentisieren kann mit einer beliebigen Methode geschehen, also bei-

spielsweise mit einem Passwort oder einer Smartcard. Lokales SSO bedeutet also, dass sich Alice einmal (gegenüber dem SSO-Client) authentisiert und dass ihr daraufhin alle Server-Anwendungen offen stehen.

Mit dem lokalen SSO hat sich – wie so oft in der IT-Sicherheit – ein besonders pragmatischer Lösungsansatz durchgesetzt. Die großen Vorteile des lokalen SSO liegen darin, dass es nicht auf irgendwelche Standards angewiesen ist und bei allen wichtigen Client-Server-Lösungen funktioniert. Dafür wirkt es allerdings reichlich hausbacken, über interne Mechanismen des Betriebssystems irgendwelche Fenster anzusprechen und mit Passwörtern zu füttern. Doch solche Überlegungen sind für die meisten IT-Abteilungen zweitrangig, solange die Sache funktioniert.

22.2.2 Ticket-SSO

Deutlich eleganter als das lokale SSO ist das **Ticket-SSO**. Dieses sieht vor, dass es neben den Servern, die Alice nutzen will, einen weiteren Server gibt (**Authentifizierungsserver**), der nur für Authentifizierungszwecke benötigt wird. Alice authentisiert sich einmal gegenüber dem Authentifizierungsserver, und anschließend sorgt dieser dafür, dass Alice direkten Zugang zu den anderen Servern erhält. Der Authentifizierungsserver ersetzt also den SSO-Client.

Die Methode, mit der der Authentifizierungsserver Alice Zugang zu den anderen Servern verschafft, basiert auf sogenannten **Tickets**. Ein Ticket ist ein Datensatz, der Alices Name und die Zugangsberechtigung zum jeweiligen Server enthält. Ein Ticket wird vom Authentifizierungsserver entweder digital signiert oder so verschlüsselt, dass nur der jeweilige Server es lesen kann. Im Gegensatz zum lokalen SSO kann Ticket-SSO nur funktionieren, wenn die beteiligten Parteien (Alice, Server, Authentifizierungsserver) ein standardisiertes Protokoll verwenden. Da Standards immer ihre Zeit brauchen, hat sich lokales SSO viel schneller verbreitet als Ticket-SSO. Dank Standards wie SAML und Kerberos ist jedoch auch Ticket-SSO inzwischen durchaus populär.

22.2.3 Web-SSO

Viele Server-Anwendungen sind heutzutage als webbasierte Lösung realisiert. Dies bedeutet, dass Alice über ihren Webbrowser und eine geeignete WWW-Adresse auf den entsprechenden Dienst zugreifen kann. Wenn die drei Server der Firma Krypt & Co., auf die Alice zugreifen will, ebenfalls WWW-Lösungen sind, dann bietet es sich an, **Web-SSO** zu nutzen. Hierbei loggt sich Alice auf einem Portal ein, wobei sie zur Authentifizierung ihr Passwort eingibt oder ihre Smartcard einsteckt. Auf dem Portal erhält Alice dann Zugang zu mehreren Webanwendungen, bei denen sie sich nicht mehr separat einloggen muss. Natürlich müssen sich auch Web-SSO-fähige Server untereinander verständigen, damit Alice sich nicht mehrfach einloggen muss. Deshalb arbeiten auch viele Webanwendungen mit

SSO-Tickets. Mit SAML (Abschnitt 22.5) gibt es einen geeigneten Standard dafür.

22.3 Kerberos

Kerberos ist der Name eines kryptografischen Netzwerkprotokolls, das zum einen ein Ticket-SSO realisiert und zum anderen einen Schlüsselaustausch durchführt. Letztere Eigenschaft macht es zu einer Alternative zu Diffie-Hellman bzw. zu einem Schlüsselaustausch mit RSA. Charakteristisch an Kerberos ist, dass es ohne asymmetrische Kryptografie auskommt. Der Name des Protokolls ist an den dreiköpfigen Hund Kerberos bzw. Cerberus angelehnt, der in der griechischen Sage den Eingang zur Unterwelt bewacht. Entwickelt wurde Kerberos am Massachusetts Institute of Technology (MIT) in den USA, wo bis heute eine kostenlos einsetzbare Implementierung gepflegt wird (es gibt zahlreiche weitere Implementierungen, darunter mehrere kostenlose). Als Vorlage für Kerberos diente ein Konzeptprotokoll, das nach seinen Entwicklern Needham-Schroeder-Protkoll genannt wird [NeeSch]. Die aktuelle Version 5 von Kerberos wurde von einer Arbeitsgruppe der IETF standardisiert und ist in [RFC4120] beschrieben.

Der Zweck von Kerberos ist es, eine Authentifizierung inklusive Schlüsselaustausch zwischen zwei Beteiligten (Alice und Server) zu realisieren, die keinen gemeinsamen geheimen Schlüssel besitzen. Dieses Ziel gewährleistet das Kerberos-Protokoll nicht etwa mithilfe der asymmetrischen Kryptografie, sondern durch die Einführung eines dritten (und zusätzlich eines vierten) Protokollbeteiligten. Diese zusätzlichen Beteiligten können Alice auch gegenüber weiteren Servern authentisieren, wodurch ein Single Sign-on erreicht wird. Dieses Single Sign-on ist ticketbasiert, wobei die verwendeten Tickets nicht signiert, sondern verschlüsselt sind.

Die kryptografische Sicherheit von Kerberos wird durch ein symmetrisches Verschlüsselungsverfahren gewährleistet. Gemäß dem Standard muss eine Kerberos-Implementierung den AES mit 256 Bit Schlüssellänge unterstützen. Die zusätzliche Unterstützung des AES mit 128 Bit Schlüssellänge sowie eine DES- und Triple-DES-Unterstützung werden empfohlen. Eine symmetrische Verschlüsselung des Klartexts m mit dem Schlüssel K wird im Folgenden als $E_K(m)$ notiert.

22.3.1 Vereinfachtes Kerberos-Protokoll

Welche Idee hinter Kerberos steckt, sieht man am besten, wenn man zunächst eine vereinfachte Version des Protokolls betrachtet. Dieses vereinfachte Protokoll sieht neben Alice und dem Server (Zielserver) einen dritten Beteiligten vor, den wir Kerberos-Server nennen wollen. Das Protokoll setzt voraus, dass Alice und der Kerberos-Server einen gemeinsamen geheimen Schlüssel KAK vereinbart haben. Der Zielserver und der Kerberos-Server besitzen ebenfalls einen gemeinsamen geheimen Schlüssel (KZK). Einen Sinn hat das Protokoll nur, wenn es neben

Alice weitere Anwender und neben dem Zielserver weitere Server gibt, die jeweils einen gemeinsamen Schlüssel mit dem Kerberos-Server besitzen. Der Kerberos-Server dient nun als ein Vermittler, der einen beliebigen Anwender (in unserem Fall Alice) und einen beliebigen Server (in unserem Fall der Zielserver) mit einem gemeinsamen Schlüssel versorgt. Dieser Ablauf erfolgt nach folgendem Protokoll:

1. *Alice → Kerberos-Server: <Alice, Zielserver>*
   Alice sendet eine Nachricht an den Kerberos-Server, die ihren Namen und den Namen des Zielservers enthält. Der Kerberos-Server weiß dadurch, welche zwei Kommunikationspartner er mit einem gemeinsamen Schlüssel versorgen muss.
2. *Kerberos-Server → Alice: <$E_{KAK}(KAZ)$,$E_{KZK}(KAZ)$>*
   Der Kerberos-Server generiert einen geheimen Schlüssel *KAZ* und sendet diesen zweimal an Alice: einmal mit *KAK* und einmal mit *KZK* verschlüsselt.
3. *Alice → Zielserver*: *<$E_{KZK}(KAZ)$>*
   Alice sendet den mit *KZK* verschlüsselten Schlüssel *KAZ* an den Zielserver. Da dieser *KZK* kennt, kann er *KAZ* entschlüsseln.

Alice und der Zielserver haben nun einen gemeinsamen geheimen Schlüssel *KAZ*, womit der Schlüsselaustausch vollzogen ist. Mithilfe von *KAZ* können sie sich gegenseitig authentifizieren und ihre Nachrichten verschlüsseln. Wenn sich Alice anschließend an einem weiteren Server anmelden will (Zielserver 2), wird das beschriebene Protokoll noch einmal abgearbeitet. Alice benötigt dazu stets nur den Schlüssel *KAK*. Wenn ihre Software diesen Schlüssel gespeichert hat, kann diese die Anmeldung am Zielserver 2 völlig transparent durchführen, wodurch ein Single Sign-On gegeben ist. Es handelt sich dabei um ein ticketbasiertes SSO, da der Wert $E_{KZK}(KAZ)$ die Rolle eines Tickets einnimmt. Dieses Ticket erhält Alice vom Kerberos-Server und reicht es an den Zielserver weiter. Es ist nicht signiert, aber durch die symmetrische Verschlüsselung vor einer Fälschung durch Mallory geschützt.

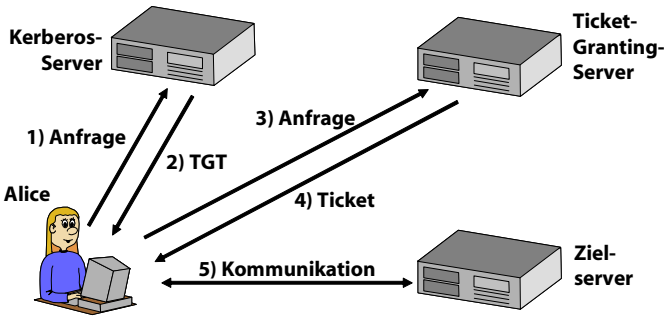


Abb. 22–2 *Kerberos ermöglicht ein ticketbasiertes Single Sign-On ohne asymmetrische Kryptografie.*

22.3.2 Vollständiges Kerberos-Protokoll

In der beschriebenen Form bietet Kerberos keine Trennung zwischen Masterschlüssel und Sitzungsschlüssel. Dadurch wird eine Known-Key-Attacke recht einfach: Wenn Mallory in den Besitz des Schlüssels *KAK* gelangt, dann kann er jede vergangene und zukünftige Kommunikation zwischen Alice und dem Zielserver entschlüsseln, da er mithilfe von *KAK* jeden Schlüssel *KAZ* ermitteln kann. Es ist also keine Perfect Forward Secrecy gegeben. Um diesen Mangel zu beheben, sieht das vollständige Kerberos-Protokoll vor, dass es neben dem Kerberos-Server einen weiteren Authentifizierungsserver gibt. Die Idee ist, dass der Schlüssel *KAK*, den Alice mit dem Kerberos-Server gemeinsam hat, nur selten zum Einsatz kommt und besonders geschützt ist (etwa durch eine Smartcard), wodurch er für Mallory besonders schwer zu ermitteln ist. Zusätzlich gibt es einen weiteren Schlüssel, der zur Übertragung von *KAZ* dient und der regelmäßig gewechselt wird. Wenn Mallory in den Besitz dieses zusätzlichen Schlüssels gelangt, richtet dies keinen Flurschaden an.

Der zusätzliche Authentifizierungsserver wird als **Ticket-Granting Server** (**TGS**) bezeichnet. Innerhalb eines Netzwerks kann es sogar mehrere Ticket-Granting Server geben, an dieser Stelle begnügen wir uns jedoch mit einem. Weiterhin gehen wir davon aus, dass Alice und der Kerberos-Server den gemeinsamen Schlüssel *KAK* besitzen. Der Kerberos-Server hat mit dem TGS den Schlüssel *KTK* gemeinsam, und der Zielserver besitzt mit dem TGS den gemeinsamen Schlüssel *KZT*. Weitere Schlüssel (*KAT* und *KAZ*) werden während des Protokollablaufs per Zufallsgenerator generiert.

Das Kerberos-Protokoll hat nur einen Sinn, wenn es außer Alice weitere Anwender und außer dem Zielserver weitere Anwendungsserver gibt. Ziel des Protokolls ist es, Alice und den Zielserver mit einem gemeinsamen Schlüssel *KAZ* zu versorgen. Im Vergleich zum vereinfachten Kerberos-Protokoll sieht die im Folgenden beschriebene vollständige Variante zum einen den zweiten Authentifizierungsserver TGS sowie einige weitere Feinheiten (z.B. Zeitstempel, Nachrichtennummern, Tickets mit beschränkter Gültigkeitsdauer) vor. Eine Kommunikation hat folgenden Ablauf:

1. *Alice* → *Kerberos-Server*: <*Alice, TGS,* t_1, n_1>
   Alice sendet eine Initialisierungsnachricht an den Kerberos-Server, in der sie ihren Namen und die Kennung eines TGS angibt. t_1 ist ein Zeitstempel, n_1 ist eine fortlaufende Nummer.
2. *Kerberos-Server* → *Alice*: <*Alice*, E_{KTK}(*Alice*, *TGS*, *KAT*, *Gültigkeitszeitraum*), E_{KAK}(*TGS*, *KAT, Gültigkeitszeitraum*)>
   Der Kerberos-Server generiert den Schlüssel *KAT*, den Alice und der TGS gemeinsam haben sollen. E_{KTK}(...) ist ein SSO-Ticket (Ticket-Granting Ticket). Es ist für den TGS lesbar, da dieser den Schlüssel *KTK* kennt. Das Ticket-Granting Ticket enthält einen Gültigkeitszeitraum, um zu verhindern, dass Alice es vor der Verwendung längere Zeit aufbewahrt.

3. *Alice* → *TGS*: <*Zielserver*, t_2, n_2, E_{KTK}(*Alice, TGT, KAT, Gültigkeitszeitraum*), E_{KAT}(*Alice*, t_1)>
   Alice sendet das Ticket-Granting Ticket an den TGS.
4. *TGS* → *Alice*: <*Alice*, E_{KTZ}(*Alice, Zielserver, KAZ*, *Gültigkeitszeitraum*), E_{KAT}(*Zielserver, KAZ, Gültigkeitsdauer*, n_2)>
   Der TGS entnimmt dem Ticket-Granting Ticket den Schlüssel *KAT*. Der Wert E_{KTZ}(...) ist ein weiteres SSO-Ticket. Dieses kann nur vom Zielserver entschlüsselt werden.
5. *Alice* → *Zielserver*: <E_{KTS}(*Alice*, *Zielserver*, *KAZ*, *Gültigkeitszeitraum*), E_{KAS}(*Alice*, t_2)>
   Alice sendet das SSO-Ticket an den Zielserver.
6. *Zielserver* → *Alice*: <E_{KAZ}(t_2)>
   Diese Nachricht ist eine Authentifizierung des Zielservers gegenüber Alice.

Alice und der Zielserver haben nach Abarbeiten dieses Protokolls den Schlüssel *KAZ* gemeinsam. Damit können sie von Mallory unbehelligt miteinander kommunizieren. Mit dem Schlüssel *KAT* kann sich Alice beim TGS Schlüssel für weitere Zielserver besorgen. Wenn dies von ihrer Software transparent erledigt wird, dann ist dies eine Form von Single Sign-On. Allerdings ist *KAT* nur eine beschränkte Zeit gültig (beispielsweise einen Tag lang). Anschließend muss Alice das beschriebene Protokoll erneut abarbeiten, um sich einen neuen Schlüssel *KAT* zu holen. Sollte Mallory in Besitz von *KAT* gelangen, dann hat er nur kurz etwas davon. Er kann damit auch keine alten Nachrichten entschlüsseln. Dadurch ist ein hohes Maß an Forward Security und Backward Security gegeben.

22.3.3 Vor- und Nachteile von Kerberos

Das Kerberos-Protokoll mutet auf den ersten Blick etwas umständlich an. Ein Protokoll ähnlicher Funktionalität könnte man mithilfe der asymmetrischen Kryptografie auch ohne Kerberos-Server bzw. Ticket-Granting-Server realisieren. In der Tat gilt es als größter Nachteil von Kerberos, dass das Protokoll stets auf die Verfügbarkeit der beiden Authentifizierungsserver angewiesen ist. Diese stellen zudem einen interessanten Angriffspunkt für Mallory dar. Hat dieser sich Zugang zum Kerberos-Server oder zum Ticket-Granting-Server verschafft, dann kann er nach Belieben die Kommunikation zwischen Alice und Bob entschlüsseln.

Die beiden Nachteile haben zur Folge, dass Kerberos fast nur innerhalb von Unternehmens- und Behördennetzen eingesetzt wird, in denen die Anwender dem Betreiber ein gewisses Vertrauen entgegenbringen. Im weltumspannenden Internet wird dagegen in aller Regel der Ansatz der Public-Key-Infrastrukturen bevorzugt. Ein Vorteil des Kerberos-Protokolls ist, dass es recht performant ist, da es auf asymmetrische Verfahren – die bekanntlich vergleichsweise aufwendig sind – verzichtet. Die Frage, warum die Entwickler von Kerberos auf asymmetrische

Verfahren verzichteten, ist einfach zu beantworten: Das Protokoll entstand Mitte der achtziger Jahre, als RSA und Diffie-Hellman noch nicht ausreichend untersucht waren, um als praxistauglich zu gelten. Seitdem sich Public-Key-Infrastrukturen durchgesetzt haben (siehe Kapitel 26), hat Kerberos deutlich an Bedeutung verloren.

22.4 RADIUS und andere Triple-A-Server

Alices Arbeitgeber Krypt & Co. betreibt auf dem Firmengelände sieben WLAN-Hotspots. Will Laptop-Anwenderin Alice eine Verbindung zum Firmennetz herstellen, dann verbindet sie sich jeweils mit dem nächstgelegenen davon. Da Krypt & Co. auf Sicherheit bedacht ist, muss sich Alice gegenüber dem jeweiligen Hotspot authentisieren, und die Kommunikation mit diesem wird verschlüsselt. An dieser Stelle taucht ein bereits angesprochenes Problem auf: Alice möchte sich nicht für sieben unterschiedliche Hotspots sieben unterschiedliche Passwörter merken oder sich gar mit sieben unterschiedlichen Smartcards herumschlagen.

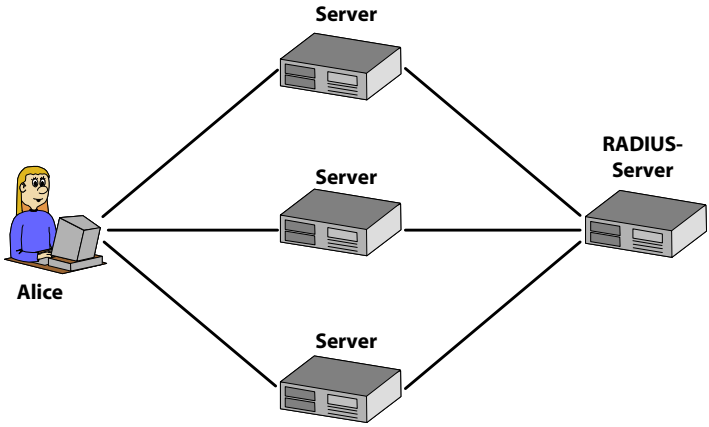


Abb. 22–3 *Mit RADIUS können verschiedene Server Alice auf gleiche Weise (z. B. mit demselben Passwort) authentifizieren.*

22.4.1 Triple-A-Server

Die Lösung des soeben beschriebenen Problems kennen Sie aus Abschnitt 22.1: Credential-Synchronisation. Wir gehen zunächst davon aus, dass sich Alice mit einem Passwort authentisiert und wir daher von Passwort-Synchronisation sprechen können. Eine naheliegende Methode zur Synchronisation von Passwörtern besteht darin, einen speziellen Server einzurichten, der für jeden Anwender ein Passwort speichert. Die WLAN-Hotspots merken sich in diesem Fall die Passwörter nicht, sondern fragen sie immer bei Bedarf beim Server ab. Wie man sich

leicht vorstellen kann, lässt sich ein derartiger Passwort-Prüf-Server zusätzlich auch für nichtkryptografische Zwecke nutzen. So kann dieser feststellen, welche Zugriffsrechte Alice hat (Autorisierung). Außerdem kann der Betreiber über einen solchen Server protokollieren, wer sich wann am Hotspot eingewählt hat. Bei kostenpflichtigen Diensten ist dies wichtig für die Abrechnung (Accounting).

Ein Server, der sich auf die beschriebene Weise um Authentifizierung (genauer gesagt um eine Credential-Synchronisation), Autorisierung und Accounting kümmert, wird **Triple-A-Server** genannt. Entsprechend heißt der Hotspot auch **Triple-A-Client**. Der dritte Beteiligte ist der Anwender, der sich einloggen will (in unserem Fall Alice). Der typische Triple-A-Ablauf sieht wie folgt aus:

1. Alice sendet eine Nachricht zum Verbindungsaufbau an den Triple-A-Client. In unserem Fall ist das der Hotspot.
2. Der Triple-A-Client fordert Alice auf, sich zu authentisieren (beispielsweise durch Eingabe von User-ID und Passwort).
3. Alice sendet User-ID und Passwort an den Triple-A-Client.
4. Der Triple-A-Client sendet eine Nachricht an den Triple-A-Server (*Access Request*), die diesem mitteilt, dass Alice authentifiziert werden möchte. Zu dieser Nachricht gehören Alices User-ID und ihr Passwort (oder ein Hashwert davon) sowie eine Kennung des Triple-A-Clients und dessen Passwort.
5. Der Triple-A-Server authentifiziert den Triple-A-Client mit dessen Passwort. Er überprüft dann Alices Passwort und stellt fest, welche Zugangsrechte sie hat. Hat der Triple-A-Server die notwendigen Informationen nicht gespeichert, so kann er selbst einen Access Request an einen anderen Triple-A-Server senden. Das Ergebnis des Passwort-Checks und die Zugangsrechte werden anschließend zurück an den Triple-A-Client gesendet.
6. Der Triple-A-Client gewährt Alice Zugang oder weist sie ab.

Ob in diesem Vorgang Einmal-Passwörter, gewöhnliche Passwörter oder ein Challenge-Response-Verfahren verwendet werden, hat auf den prinzipiellen Ablauf keinen Einfluss. Bei einem Challenge-Response-Verfahren ist jedoch zusätzlich noch ein Nachrichtenaustausch zwischen Triple-A-Server und Triple-A-Client notwendig. Auch eine zertifikatsbasierte Authentifizierung oder die Nutzung von Biometrie ist möglich.

Charakteristisch am beschriebenen Triple-A-Protokoll ist die doppelte Authentifizierung: Alice authentisiert sich gegenüber dem Triple-A-Client, der sich wiederum gegenüber dem Triple-A-Server authentisiert. Passwörter, sonstige Authentifizierungsdaten, Informationen über Zugriffsrechte und Accounting-Daten werden alle nur auf dem Triple-A-Server festgehalten. Welche Zugangspunkte Alice nutzen darf, kann daher zentral konfiguriert werden. Ihr Passwort muss Alice nur beim Triple-A-Server ändern und nicht für jeden Rechner einzeln – Crendential-Synchronisation ist also gegeben. Einziger Nachteil: Der Triple-A-Server muss besonders geschützt werden. Gelingt es Mallory nämlich, dort einzudringen, dann liegt ihm das System zu Füßen.

22.4.2 Beispiele für Triple-A-Server

Es gibt drei bedeutende Standards für Triple-A-Server: RADIUS, TACACS und DIAMETER. Alle drei funktionieren nach dem beschriebenen Prinzip.

RADIUS

Der bedeutendste Triple-A-Standard ist der **Remote Authentication Dial-In User Service** (**RADIUS**). RADIUS wurde von der Firma Livingston entwickelt und wird in [RFC2865] beschrieben. Über ein Dutzend weitere RFCs behandeln diverse RADIUS-Erweiterungen oder sonstige RADIUS-Themen. RADIUS ist ein weit verbreiteter De-facto-Standard, der in zahlreichen Produkten umgesetzt ist. Es gibt mindestens fünf Open-Source-Implementierungen, die bekanntesten davon heißen freeRADIUS und OpenRADIUS. Der Ablauf des RADIUS-Protokolls entspricht recht genau dem zuvor beschriebenen, wobei jedoch teilweise andere Bezeichnungen verwendet werden (RADIUS-Server und RADIUS-Client statt Triple-A-Server und Triple-A-Client).

TACACS

In [RFC1492] wird das unter Mitwirkung der Firma CISCO entstandene Protokoll **TACACS** (*Terminal Access Controller Access Control System*) beschrieben, das den Betrieb eines Triple-A-Servers ermöglicht. Durch die Hinzunahme von Zugriffsrechten und Accounting wurde daraus **TACACS+**. Dieses ist jedoch nicht über den Status eines Internet Drafts hinausgekommen. TACACS+ unterscheidet sich bezüglich der Funktionalität kaum von RADIUS, ist jedoch nicht mit diesem kompatibel.

DIAMETER

DIAMETER ist der Nachfolger von RADIUS. Der Name ist ein Wortspiel: »Diameter« heißt Durchmesser, und dieser beträgt bekanntlich das Doppelte des Radius. DIAMETER ist nicht vollständig abwärtskompatibel zu RADIUS, ermöglicht aber ein Upgrade. Das DIAMETER-Protokoll wird in [RFC3588] beschrieben. Aus Kryptografensicht hat es den Vorteil, dass es im Gegensatz zu RADIUS verhandlungsfähig ist. Alle weiteren Unterschiede zu RADIUS sind nichtkryptografischer Natur. Bisher steht DIAMETER noch im Schatten seines Vorgängers.

22.5 SAML

Mit Kerberos, RADIUS und einigen anderen gibt es bewährte Standards für das Single Sign-On und die Credential-Synchronisation. Die genannten Technologien sind allerdings schon recht alt und haben jeweils nur einen begrenzten Einsatzbe-

reich. In dem Vakuum, das dadurch in den letzten Jahren entstanden ist, hat sich inzwischen ein Standard namens **SAML** (Security Assertion Markup Language) etabliert [Hughes]. SAML ermöglicht sowohl Single Sign-On als auch Credential-Synchronisation. Im Gegensatz zu Kerberos werden auch asymmetrische Verfahren unterstützt.

SAML ist ein recht komplexer Standard mit unterschiedlichen Anwendungsmöglichkeiten. Der Einsatz von SAML ist insbesondere nicht auf die verteilte Authentifizierung begrenzt, auch wenn dieses derzeit die wichtigste Anwendung darstellt. Aus Sicht des Kryptografen lässt sich die Funktionsweise wie folgt zusammenfassen: SAML sieht digital signierte Tickets (**Security Assertions**) vor; diese ermöglichen es Servern, Authentifizierungsinformationen untereinander auszutauschen, um damit die gewünschten Ziele Single Sign-On und Credential-Synchronisation zu erreichen.

SAML basiert auf XML (Extensible Markup Language). XML ist ein weit verbreiteter Standard zur Definition maschinenlesbarer Beschreibungssprachen, der vom World-Wide-Web-Konsortium (W3C) gepflegt wird. Die Entwicklung von SAML begann im Jahr 2001 und erfolgt innerhalb der Standardisierungsorganisation OASIS, der unter anderem Firmen wie Sun Microsystems, IBM und SAP angehören. Die aktuelle SAML-Versionsnummer ist 2.0.

22.5.1 Funktionsweise von SAML

Dem Namen nach ist SAML eine Beschreibungssprache, die das Kodieren von Security Assertions erlaubt. Tatsächlich enthält der SAML-Standard jedoch nicht nur diese Beschreibungssprache, sondern zusätzlich ein Netzwerkprotokoll zur Übertragung von Security Assertions sowie einige weitere Inhalte. Formal betrachtet ist eine Security Assertion ein Datensatz, der sicherheitsrelevante Informationen über ein Objekt (beispielsweise über Anwenderin Alice) enthält. In der Regel ist eine Security Assertion signiert. So gesehen hat eine solche eine gewisse Ähnlichkeit mit einem digitalen Zertifikat (Abschnitt 26.2). Es ist zudem möglich, Security Assertions über XML Encryption oder SSL zu verschlüsseln. Die Entwickler von SAML hatten vor allem drei Anwendungsbereiche im Blickfeld:

- *Single Sign-On*: Hierbei übernimmt eine Security Assertion die Funktion eines SSO-Tickets.
- *Autorisierungsdienste*: Hierbei läuft die Kommunikation zwischen Alice und dem Server über einen Authentifizierungsserver, der für eine einheitliche Authentifizierung an unterschiedlichen Systemem sorgt. Es handelt sich also um eine Form der Credential-Synchronisation.
- *Verteilte Transaktionen*: Hierbei arbeiten mehrere Anwender gemeinsam an einer Aufgabe und verwenden Security Assertions für den sicheren Datenaustausch. Diese Anwendung spielt an dieser Stelle keine Rolle.

Das SAML-Netzwerkprotokoll ist ein Zwei-Wege-Protokoll – ähnlich wie HTTP, FTP und OCSP. Es sieht vor, dass eine Partei (Requester) eine Security Assertion bei einer anderen Partei (Responder) anfordert und dass der Responder mit einer Security Assertion antwortet. Der folgende Ablauf ist typisch:

1. Alice meldet sich an einem Anwendungsserver an. Dieser leitet die Anmeldung an einen Authentifizierungsserver weiter.
2. Der Authentifizierungsserver authentifiziert Alice auf eine festgelegte Weise (Smartcard, Passwort, ...).
3. Der Authentifizierungsserver schickt eine verschlüsselte und signierte Security Assertion an den Anwendungsserver. Darin teilt er mit, dass Alice korrekt authentifiziert ist.
4. Der Anwendungsserver überprüft die Security Assertion und gewährt Alice im positiven Fall Zugang.

Will sich Alice an einem weiteren Anwendungsserver anmelden, dann erfolgt dies erneut über den Authentifizierungsserver. Dieser sorgt beim beschriebenen Ablauf für eine Credential-Synchronisation, da Alice für jeden Anwendungsserver dieselbe Karte oder dasselbe Passwort nutzen kann. Der Authentifizierungsserver sorgt außerdem für ein Single Sign-On, sofern das Verschicken und die Verarbeitung einer Security Assertion für Alice transparent verlaufen.

22.5.2 SAML in der Praxis

SAML wurde ursprünglich vor allem für Webservices (also für Webdienste, die nicht von Menschen, sondern von Computern genutzt werden) entwickelt. Es wird inzwischen jedoch auch für den Zugang zu menschenlesbaren Webseiten genutzt. Besonders populär ist SAML als Lösung für eine Kombination aus Web-SSO und Ticket-SSO. In diesen Bereich fallen auch die beiden derzeit wichtigsten Projekte, die SAML nutzen:

- **Shibboleth**: Shibboleth ist eine Open-Source-Software für das Single Sign-On, die SAML verwendet [Shibbo].
- **Kantara-Initiative**: Hierbei handelt es sich um eine Initiative, an der sich zahlreiche Unternehmen, Organisationen und Privatpersonen beteiligen. Sie hat das Ziel, Rahmenbedingungen und Standards für das Identity Management zu schaffen. SAML spielt dabei eine zentrale Rolle.

Mit Microsoft account hat die Firma Microsoft eine Konkurrenzlösung geschaffen, die nicht auf SAML basiert.

23 Krypto-Hardware und Krypto-Software

Bisher haben wir uns mit Algorithmen, Protokollen und Standards beschäftigt, die Alice und Bob für ihre Kommunikation einsetzen können. Damit allein ist jedoch noch nie auch nur ein verschlüsseltes Bit über eine Leitung transportiert worden. Vielmehr benötigen Alice und Bob zu ihrem Glück noch eine Implementierung, welche die theoretischen Erkenntnisse in die Praxis umsetzt.

23.1 Krypto-Hardware oder Krypto-Software?

An dieser Stelle müssen wir uns eine grundsätzliche Frage stellen: Sollen Alice und Bob kryptografische Verfahren in Hardware oder in Software nutzen? Sowohl das eine als auch das andere hat seine Vor- und Nachteile.

23.1.1 Pro Software

Wenn ein Hersteller Kryptografie in Software realisieren will, dann bedeutet dies, dass er ein Programm für einen Computer erstellt, der für unterschiedliche Zwecke genutzt werden kann – beispielsweise für einen PC, einen Server oder ein Smartphone. Dabei gibt es unterschiedliche Möglichkeiten, wie die Realisierung einer Krypto-Software aussehen kann. Am einfachsten ist es für Alice und Bob, wenn Programme, die sie ohnehin verwenden, kryptografische Funktionen unterstützen (**native Krypto-Funktionen**). Diese Voraussetzung ist heutzutage in sehr vielen Fällen gegeben. Betriebssysteme wie Linux und Windows haben genauso kryptografische Mechanismen eingebaut wie die gängigen E-Mail-Clients, Webbrowser und Textverarbeitungsprogramme. Auch Server-Applikationen wie Verzeichnisdienste, Datenbankserver und Webserver bieten oft native Krypto-Funktionen.

Native Krypto-Funktionen sind zwar eine schöne Sache, in vielen Fällen aber nicht ausreichend. Kryptografie hat für die meisten Softwarehersteller nun einmal nicht den höchsten Stellenwert, und das sieht man vielen Produkten an. Für Unternehmen und Behörden mit einem besonders hohen Sicherheitsbedarf kommt noch ein weiteres Argument ins Spiel: Die meisten populären Softwarelösungen kommen aus den USA, wo die Politik (oft in Form der NSA) schon so manches Mal in die Kryptografie eingegriffen hat. Manche Organisationen (beispielsweise im Militärwesen) vermeiden daher die nativen Krypto-Funktionen von US-Herstellern.

Angesichts der erwähnten Nachteile nativer Krypto-Funktionen können sich die Hersteller kryptografischer Zusatzprodukte (**additive Krypto-Funktionen**) nach wie vor gut am Markt behaupten. Das Angebot ist reichhaltig. Für alle gängigen Betriebssysteme gibt es zahlreiche Dateiverschlüsselungsprogramme. Für Webbrowser und E-Mail-Clients werden Plugins und clientbasierte Proxy-Lösungen angeboten. Ähnlich sieht es auf Serverseite aus. Dort reicht das Angebot vom alleinstehenden E-Mail-Krypto-Gateway bis zum Krypto-Add-on für Server-Programme. Auch wenn sich die nativen Krypto-Funktionen der bekannten Softwareprogramme ständig verbessern, wird es auch zukünftig genug Raum für additive Lösungen geben. Allerdings müssen sich deren Anbieter vermehrt auf Nischenmärkte (z. B. Organisationen mit Hochsicherheitsanforderungen) spezialisieren.

Die folgende Liste nennt die wichtigsten Vorteile von Krypto-Software gegenüber Krypto-Hardware:

- Das Schreiben eines Computerprogramms ist einfacher und damit billiger als das Design von Hardware.
- Besonders wenn der Quellcode einer Software bekannt ist, lassen sich Hintertüren und unbeabsichtigte Schwächen leichter erkennen als bei Hardware.
- Einige symmetrische Verfahren (unter anderem zahlreiche Stromchiffren) sind in erster Linie für Softwareimplementierungen entwickelt worden.

- Nachträgliche Änderungen lassen sich bei Software leichter durchführen.
- Materialkosten fallen bei Hardware ins Gewicht, bei Software dagegen nicht.
- Software hat einen höheren Grad an Portabilität. Ein einmal implementierter Algorithmus lässt sich in der Regel auch auf andere Systeme übertragen.

23.1.2 Pro Hardware

Kryptografie lässt sich auch in Hardware realisieren. Dies bedeutet, dass ein Hersteller ein Hardwaremodul entwickelt, das kryptografische Verfahren ausführt. Die zu bearbeitende Nachricht und gegebenenfalls der Schlüssel werden auf das Hardwaremodul geleitet, das Resultat (etwa der Geheimtext) kommt zurück. In Hardware lassen sich die gleichen Funktionen realisieren wie in Software. Im ersteren Fall werden die Krypto-Algorithmen jedoch an ein externes Modul ausgelagert, während bei Krypto-Software alle Operationen auf demselben Rechner stattfinden. Auch Krypto-Hardware hat durchaus ihre Vorteile:

- Hardware ist in vielen Realisierungsformen schneller als Software.
- Einige symmetrische Verfahren (etwa der DES) sind für Hardware optimiert.
- Hardware ist deutlich weniger anfällig gegenüber Malware-Attacken. Auch sonst ist die gezielte Manipulation von Hardware deutlich schwieriger als bei Software.
- Mallory kann Software meist einfacher analysieren als Hardware.
- Bei einer Hardwareimplementierung können geheime Schlüssel so gespeichert werden, dass sie das Modul nie verlassen. Wenn dieses Modul zusätzlich durch ein Passwort geschützt ist, dann kann Mallory nicht einmal dann Schaden anrichten, wenn er das Modul stiehlt.
- Ein weiterer Vorteil von Hardware besteht darin, dass Alice ihr eigenes Hardwaremodul ständig bei sich tragen kann. Egal wo sie hinkommt, sie kann überall mit ihrem eigenen Schlüssel verschlüsseln, entschlüsseln und signieren.

23.1.3 Ist Hardware oder Software besser?

Den Vergleich zwischen Krypto-Hardware und Krypto-Software kann man leicht auf einen Nenner bringen: Krypto-Hardware ist die aufwendigere Lösung, dafür ist sie sicherer und meistens schneller. Bei hohen Anforderungen an Sicherheit und Performanz ist der Einsatz von Hardware daher oft unverzichtbar. Krypto-Software ist dagegen weniger aufwendig, weniger sicher und in vielen Fällen langsamer. Als einer der Hauptvorteile von Krypto-Hardware hat sich jedoch erwiesen, dass Alice ihr eigenes Krypto-Modul (beispielsweise in Form einer Smartcard) mit sich herumtragen kann, auf dem ihr geheimer Schlüssel gespeichert ist. Da der geheime Schlüssel das Modul nie verlässt und auch nicht auslesbar ist, bietet ein Hardwaremodul einen besonderen Schutz vor Mallory.

23.2 Smartcards

Sicherlich tragen auch Sie in Ihrer Geldbörse einige Karten aus Plastik mit sich herum – etwa die Bahn Card, eine Kreditkarte oder eine Bankkarte. All diese Plastikkarten haben zwar die gleiche Größe, unterscheiden sich in ihrer Funktionsweise aber teilweise erheblich. Befindet sich auf der Rückseite einer Plastikkarte ein dunkler Streifen von etwa einem Zentimeter Breite (ein Magnetstreifen), dann handelt es sich um eine **Magnetstreifenkarte** (Abbildung 23–1). Der Magnetstreifen besteht aus drei nicht sichtbaren Rillen, in denen Daten gespeichert werden können. 226 Byte sind es, die nach dem einschlägigen Standard ISO 7811 auf einen Streifen und damit auf eine Karte passen. Eine Magnetstreifenkarte ist damit ein Speichermedium wie ein Memory-Stick oder eine Festplatte, wenn auch mit sehr viel geringerer Speicherkapazität. Für die Kryptografie sind Magnetstreifenkarten nicht besonders interessant.

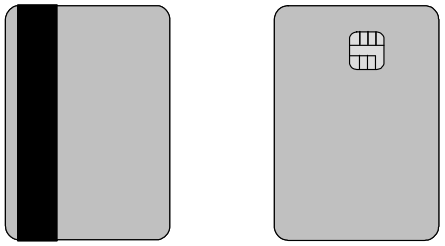


Abb. 23–1 *Magnetstreifenkarten haben einen schwarzen Streifen auf der Rückseite. Kontaktbehaftete Chipkarten sind an einem Metallkontakt zu erkennen.*

23.2.1 Smartcards und andere Chipkarten

Wesentlich leistungsfähiger als eine Magnetstreifenkarte ist die zweite wichtige Variante der Plastikkarte: die **Chipkarte** (Abbildung 23–1). Auf einer solchen ist ein Hardwarechip angebracht, der deutlich leistungsfähiger ist als ein Magnetstreifen. Es gibt einerseits **kontaktbehaftete Chipkarten**, die über einen Metallkontakt (in der Größe eines Fingernagels auf der Vorderseite) mit der Außenwelt kommunizieren. Davon zu unterscheiden sind **kontaktlose Chipkarten**, die per Funk angesprochen werden. Kontaktlose Chipkarten sind technisch aufwendiger und durch die kompliziertere Stromversorgung im Allgemeinen weniger leistungsfähig als ihre kontaktbehafteten Verwandten. Wenn Sie sich im Detail mit Chipkarten beschäftigen wollen, dann empfehle ich Ihnen das Standardwerk *Handbuch der Chipkarten* von Wolfgang Effing und Wolfgang Rankl [EffRan].

Damit ein Computer eine Chipkarte ansprechen kann, benötigt man ein **Karten-Terminal** (der ebenfalls verbreitete Begriff **Kartenleser** ist nicht ganz passend, da ein solcher auch zum Beschreiben einer Karte verwendet werden kann). Kon-

taktbehaftete Chipkarten benötigen andere Terminals als kontaktlose. Erstere müssen in das Terminal hineingesteckt werden, während für kontaktlose Karten nur eine räumliche Nähe erforderlich. ist. Abbildung 23–2 zeigt ein Karten-Terminal für kontaktbehaftete Karten mit eingebauter Tastatur. Ein solches hat den Vorteil, dass Alice eine etwaige Karten-Geheimnummer (PIN) nicht über die PC-Tastatur eingeben muss, wodurch ein Malware-Angriff (durch eine Software, die alle Tastatureingaben mitliest und so die PIN ermittelt) verhindert wird.

Chipkarten gibt es als **Speicherkarten** und als **Smartcards** (der Unterschied ist von außen nicht sichtbar). Bei einer Speicherkarte ist der Chip lediglich als Datenspeicher realisiert und daher nicht in der Lage, Berechnungen durchzuführen. Da Speicherkarten im Zusammenhang mit der Kryptografie keine große Rolle spielen, werde ich nicht näher auf sie eingehen. Bei einer Smartcard ist der Chip ein Computer im Kleinformat. Er besitzt einen Prozessor, einen Lesespeicher (ROM), einen Arbeitsspeicher (RAM) und einen elektrisch löschbaren programmierbaren Speicher (EEPROM). Ein spezielles Smartcard-Betriebssystem koordiniert diese auf dem Chip vorhandenen Ressourcen. Ein- und Ausgaben erfolgen über die Kontaktfläche auf der Vorderseite oder per Funk – je nachdem, ob es sich um eine kontaktlose oder kontaktbehaftete Smartcard handelt. Typische Smartcards neueren Datums haben etwa 256 KByte ROM, 128 KByte EEPROM und 32 KByte RAM zur Verfügung und werden von einer 20-MHz-CPU betrieben.

Abb. 23–2 *Ein Karten-Terminal mit eingebauter Tastatur erschwert Malware-Angriffe.*

Die Smartcard ist keine neue Erfindung. Erste Patente gab es schon Ende der sechziger und Anfang der siebziger Jahre. Damals war Hardware jedoch noch sperrig und teuer, weshalb an ein Massenprodukt nicht zu denken war. Erst Mitte der achtziger Jahre entwickelte sich die Smartcard zu einem Erfolgsmodell der Computertechnik. Bisher werden die schlauen Karten vor allem im Bank- und Zahlungsbereich eingesetzt. Pay-TV und Mobiltelefone sind weitere Anwendungsgebiete, die heute schon eine Rolle spielen. Groß im Kommen ist die Smartcard gegenwärtig im Gesundheitswesen und als Ausweis. Auch als Ersatz für Bargeld

gewinnen Smartcards immer mehr an Bedeutung. Erhebliche Steigerungsraten in der Smartcard-Branche sind daher für die kommenden Jahre vorprogrammiert.

23.2.2 Smartcard-Formfaktoren

Ein Chip, der auf eine Smartcard aufgebracht werden kann, lässt sich in aller Regel auch in andere Gegenstände integrieren. In den letzten Jahren haben immer mehr Hersteller von dieser Möglichkeit Gebrauch gemacht. Dadurch gibt es inzwischen eine ganze Reihe von Karten und anderen Gegenständen, die zwar wie eine herkömmliche Smartcard funktionieren, jedoch anders aussehen. Man spricht in diesem Zusammenhang auch vom **Formfaktor** einer Smartcard. Neben der bereits erwähnten Plastikkarte im Kreditkartenformat gibt es unter anderem folgende weiteren Smartcard-Formfaktoren (in einigen Fällen passt der Begriff »Smartcard« kaum noch, da es sich nicht um eine Karte handelt, ich will ihn der Einfachheit halber dennoch beibehalten):

- **USB-Token**: Ein solches sieht aus wie ein USB-Stick. Der wesentliche Unterschied zu einer herkömmlichen Smartcard besteht darin, ein USB-Token direkt in den USB-Port gesteckt wird, wodurch die Notwendigkeit für ein Terminal entfällt.
- **MicroSD-Smartcard**: MicroSD ist eine Hardwareschnittstelle, die vor allem in Smartphones anzutreffen ist. MicroSD wird normalerweise verwendet, um zusätzlichen Speicher an ein solches Gerät anzuschließen. Inzwischen gibt es jedoch auch fingernagelgroße Mini-Smartcards, die an die Micro-SD-Schnittstelle andocken. Diese MicroSD-Smartcards machen die Smartcard-Technologie auch für Handys und PDAs interessant.
- **SIM-Karte**: SIM-Karten sind spezielle kleine Smartcards, die in Handys eingesetzt werden. Inzwischen bieten einige Hersteller auch SIM-Karten an, die zusätzlich andere Smartcard-Funktionen übernehmen können. Dies ist eine weitere Möglichkeit, eine Smartcard mit dem Handy zu nutzen.
- **Smart Token**: Kontaktlose Smartcard-Chips lassen sich in nahezu jeden Gegenstand integrieren. Sofern es sich dabei um einen kleinen, handlichen Gegenstand handelt (z. B. Schlüsselanhänger, Autoschlüssel oder Armband), spricht man von einem Smart Token.

23.2.3 Smartcards und Kryptografie

Smartcards sind wie geschaffen für die Kryptografie. Sie können einen geheimen Schlüssel sicher speichern und mit diesem rechnen, ohne dass der Schlüssel den Kartenchip verlassen muss. Das typische Nutzungsszenario sieht vor, dass Alice Daten in den Chip hineinleitet, wo sie mit dem geheimen Schlüssel verschlüsselt, entschlüsselt oder signiert werden. Anschließend fließen die derart bearbeiteten Daten vom Chip zurück zu Alice. Selbst wenn Mallory den von Alice verwendeten

PC samt Terminal unter Kontrolle hat, hat er keine Möglichkeit, an den Schlüssel in der Karte zu kommen. Wenn Alice ihre Karte verliert oder diese gestohlen wird, kann sie dies meist verschmerzen, da Mallory den Schlüssel nicht aus der Karte lesen kann. Wenn der Hersteller eine Smartcard mit einem Passwort (PIN) ausstattet, ohne dessen Eingabe die Karte nicht arbeitet, hat Mallory überhaupt keine Möglichkeit zum Missbrauch einer gestohlenen Karte. Einen Nachteil haben Smartcards jedoch: Durch ihre spartanische Hardwareausstattung brauchen sie vergleichsweise lange für einen Verschlüsselungs- oder Signiervorgang.

Abb. 23–3 *Ein USB-Token funktioniert wie eine Smartcard, benötigt jedoch keinen Kartenleser.*

Symmetrische Verschlüsselung mit Smartcard

Smartcards eignen sich hervorragend für die symmetrische Verschlüsselung (und Entschlüsselung) – sofern Alice und Bob keine zu hohen Ansprüche an die Geschwindigkeit stellen. Alice leitet hierbei die zu verschlüsselnden Daten auf ihre Smartcard, wo sie mit dem AES oder einem anderen symmetrischen Verfahren verschlüsselt werden. Empfänger Bob kann analog verfahren, um die Entschlüsselung durchzuführen. Wenn größere Datenmengen im Spiel sind, wird auf dem Chip meist nur ein Masterschlüssel gespeichert, den Alice und Bob ausschließlich zum Verschlüsseln eines Sitzungsschlüssels nutzen. Die Verschlüsselung der eigentlichen Nachricht erfolgt in diesem Fall mit dem Sitzungsschlüssel außerhalb der Karte. Eine solche Methode ist beispielsweise im Pay-TV und im Mobilfunk verbreitet.

Symmetrische Authentifizierung mit Smartcard

Gut geeignet sind Smartcards auch für das Berechnen einer Response aus einer Challenge mit einem symmetrischen Verfahren. Diese Möglichkeit ist in vielen Zutrittskontrollanlagen implementiert. Wenn Alice eine Tür öffnen will, sendet das System zur Authentifizierung eine Challenge an Alices Karte. Diese berechnet (durch eine Verschlüsselung oder mit einer schlüsselabhängigen Hashfunktion) die Response und gibt diese zurück. Für diesen Zweck werden meist kontaktlose

Chipkarten oder Smart Tokens eingesetzt. Das Prinzip ist Ihnen vermutlich vom Türschloss Ihres Autos bekannt (der Autoschlüssel dient hierbei als Smart Token). Auch zahlreiche Gebäudetüren und Schließfächer sind auf diese Art geschützt. Oft werden für die symmetrische Authentifizierung Chipkarten verwendet, die außer dem Berechnen einer Response mit einer festverdrahteten Logik nichts können, dafür aber sehr kostengünstig sind – es handelt sich dabei nicht um Smartcards im eigentlichen Sinne. Gängige Chips dieser Art sind Mifare, Hitag (beide von NXP Semiconductors) sowie die Legic-Chips von LEGIC Identsystems.

Abb. 23–4 *Ein Autoschlüssel ist heute meist als Smart Token realisiert, das ein Challenge-Response-Verfahren ausführt.*

Asymmetrische Verschlüsselung mit Smartcard

Auch der private Schlüssel eines asymmetrischen Verschlüsselungsverfahrens (z.B. RSA) lässt sich auf einer Smartcard speichern. Der Kartenchip übernimmt hierbei das Entschlüsseln einer Nachricht auf Empfängerseite, wodurch der Schlüssel Bobs Karte nicht verlassen muss. Zum Verschlüsseln benötigt Absenderin Alice dagegen keine Smartcard, da dieser Vorgang mit dem öffentlichen Schlüssel erfolgt. Das wichtigste Problem in diesem Zusammenhang ist die Performanz: Asymmetrische Verfahren sind nun einmal äußerst rechenintensiv, was einen Smartcard-Chip schnell überfordert. Manche Hersteller nutzen daher kryptografische Ko-Prozessoren – dies erhöht jedoch den Preis für einen Kartenchip. Eine Alternative ist der Einsatz von Krypto-Systemen auf Basis elliptischer Kurven. Diese benötigen bei vergleichbarer Sicherheit nur ein Zehntel an Rechenzeit im Vergleich zu RSA oder DSA. Zwar ist dies immer noch deutlich mehr, als etwa der AES erfordert. Die Einsparung ist dennoch so groß, dass auf einen Ko-Prozessor verzichtet werden kann, sofern das EEPROM des Chips etwas größer ist. Krypto-Systeme auf Basis elliptischer Kurven erfreuen sich aus diesem Grund im Smartcard-Bereich steigender Beliebtheit. Dennoch gilt: Ein Smartcardchip, der nur ein symmetrisches Verschlüsselungsverfahren ausführen kann, ist deutlich billiger als ein solcher, der ein asymmetrisches Verfahren beherrscht.

Für asymmetrische Verfahren kommen bisher meist kontaktbehaftete Smartcards zum Einsatz. Dies liegt daran, dass die gängigen kontaktlosen Chips nicht besonders rechenstark sind (dies liegt wiederum an der kontaktlosen Stromversorgung, die nicht besonders leistungsfähig ist). Allerdings hat die kontaktlose Technik in den letzten Jahren deutliche Fortschritte gemacht. Kontaktlose Smartcards für asymmetrische Verfahren sind daher stark im Kommen.

Bekanntlich verwenden Alice und Bob asymmetrische Verfahren meist nur zum Verschlüsseln kleiner Datenmengen (z.B. eines Sitzungsschlüssels im Rahmen eines Hybridverfahrens). Das Verschlüsseln der eigentlichen Nachricht mit dem Sitzungsschlüssel könnte die Smartcard prinzipiell zwar ebenfalls übernehmen (dann müsste der Sitzungsschlüssel die Karte nicht verlassen). Dies ist jedoch wegen der geringen Leistungsfähigkeit eines Smartcard-Chips unüblich. Stattdessen sehen alle gängigen Implementierungen vor, dass nur die Berechnungen mit dem privaten Schlüssel des asymmetrischen Verfahrens auf der Karte stattfinden, während sich das angeschlossene Gerät (z.B. der PC) um die Ver- und Entschlüsselung mit dem Sitzungsschlüssel kümmert. Dieses Vorgehen bietet naturgemäß nicht die optimale Sicherheit: Wenn Bob einen von Mallory manipulierten PC verwendet, dann kann Mallory den Sitzungsschlüssel abgreifen. An Bobs privaten RSA-Schlüssel kommt er jedoch nicht heran, da dieser unauslesbar auf der Karte gespeichert ist. Dadurch ist immerhin Perfect Forward Secrecy gegeben.

Digitale Signaturen mit Smartcard

Besonders interessant sind Smartcards zweifellos für digitale Signaturen. Wenn Bob beispielsweise einen Text mit seinem Textverarbeitungsprogramm geschrieben hat und ihn signieren möchte, dann steckt er seine Karte in das Terminal am Computer und klickt auf den Menüpunkt »Signieren«. Die Software fragt nun nach Bobs PIN und bildet anschließend einen Hashwert mit einer kryptografischen Hashfunktion, den sie an die Smartcard sendet (alternativ kann auch die Smartcard selbst die Hashwert-Generierung übernehmen). Anschließend signiert die Smartcard den Hashwert und gibt die Signatur zurück.

Asymmetrische Authentifizierung mit Smartcard

Eine Smartcard lässt sich auch für die Authentifizierung mit einem asymmetrischenVerfahren nutzen. Alice verwendet den privaten Schlüssel auf ihrer Karte hierbei zur Berechnung einer Response aus einer Challenge. Wie Sie aus Abschnitt 21.3 wissen, kann Alice hierfür sowohl ein asymmetrisches Verschlüsselungsverfahren als auch ein digitales Signaturverfahren einsetzen (erstere Methode schließt einen Schlüsselaustausch mit ein). Smartcards werden in dieser Form oft zur Authentifizierung genutzt, wenn eine Passwortabfrage zu unsicher ist. Die asymmetrische Variante wird vor allem am PC eingesetzt, beispielsweise bei der Systemanmeldung, bei der Anmeldung am Webportal oder beim Aufbau einer VPN-Verbindung. Die weiter oben beschriebene symmetrische Challenge-Response-

Authentifizierung mit Smartcard ist dagegen an Türschlössern, Autoschlössern und Schließfächern weiter verbreitet. Warum das so ist, ist leicht einzusehen: Am PC kann Alice die Smartcard zusätzlich zum Signieren und für Verschlüsselungsanwendungen nutzen, wodurch es sinnvoll ist, asymmetrische Verfahren einzusetzen; beim Öffnen eines Schlosses gibt es diese Zusatzanwendungen nicht, weshalb die meisten Betreiber die billigere, symmetrische Variante bevorzugen.

Will ein Unternehmen die Passwort-Authentifizierung am PC durch eine sicherere Methode ersetzen, dann gibt es meist zwei Möglichkeiten: entweder eine Smartcard mit asymmetrischem Verfahren oder ein OTP-Token. Eine Abwägung zwischen Smartcard und OTP-Token gehört daher zu den Standardaufgaben eines IT-Sicherheitsbeauftragten. Für OTP-Tokens sprechen folgende Argumente:

- Ein OTP-Token ist meist etwas billiger als eine Smartcard – auch wenn die Preise natürlich variieren können.
- Ein OTP-Token kommt ohne Kartenleser, USB-Port und Treiber (Smartcard-Middleware) aus.

Für Smartcards spricht:

- Smartcards sind nicht nur zur Authentifizierung geeignet, sondern auch für das Signieren und Verschlüsseln.
- Smartcards sind sicherer als OTP-Tokens. So kann Mallory beispielsweise Alice über die Schulter schauen, während diese die Response von ihrem Token abliest. Wenn er schnell genug ist, kann er die Response vor Alice an den Server schicken. Auch eine Man-in-the-Middle-Attacke ist bei einem OTP-Token prinzipiell möglich. Eine Smartcard-Authentifizierung kann Mallory dagegen kaum angreifen.

OTP-Tokens sind seit etwa 15 Jahren weit verbreitet, doch Smartcards holen inzwischen auf.

Fazit

Kein Zweifel, Smartcards und Kryptografie passen hervorragend zusammen. Es gibt für Alice und Bob kaum eine bessere Möglichkeit, ihre Schlüssel zu schützen. Theoretisch jedenfalls, denn natürlich kann Mallory versuchen, mit Seitenkanalangriffen oder physikalischen Angriffen die Schutzmechanismen einer Smartcardszu umgehen und den geheimen Inhalt auszulesen. Um derartiges zu verhindern, haben sich die Smartcard-Hersteller einiges einfallen lassen. So sind die Speicherzellen von Smartcards meist bewusst unübersichtlich platziert. Es gibt Attrappen, die von echten Zellen nicht zu unterscheiden sind. Die Leiterbahnen nehmen Umwege, um sie schwerer verfolgbar zu machen. Weitere Sicherheitsvorkehrungen sehen vor, dass der Speicherinhalt automatisch gelöscht wird, wenn der Speicher mit Elektronen bestrahlt wird, was ebenfalls ein Auslesen verhindern soll. Einen Betrieb mit niederer Taktfrequenz (bei dem Mallory Veränderungen im

Speicher beobachten kann) lassen zahlreiche Chips ebenfalls nicht zu. Wir können also davon ausgehen, dass ein Schlüssel in einer Smartcard gut aufgehoben ist.

23.3 Hardware-Security-Module

Der größte Nachteil der Smartcard ist ihre geringe Performanz. Darüber hinaus bieten Smartcards meist keine Schnittstelle für die Erstellung einer Sicherheitskopie, und der Schutz vor physikalischen Angriffen ist trotz des beschriebenen Designs nicht optimal. Smartcards müssen nun einmal klein und billig sein, da ist eine großzügige Ausstattung mit speziellen Funktionen und Merkmalen schwierig bis unmöglich. Für solche Fälle, in denen eine Smartcard zu schwach ausgelegt ist, bietet sich ein **Hardware-Security-Modul** (**HSM**) an. Dabei handelt es sich um ein Hardwaremodul, das meist als PC-Einsteckkarte oder als eigenständige Box realisiert ist.

Die Funktion eines HSM kann man sich zunächst wie die einer Smartcard vorstellen. Auf dem HSM ist ein geheimer Schlüssel unauslesbar gespeichert. Signaturen und Verschlüsselungsoperationen kann das HSM selbst ausführen, wenn die zu bearbeitenden Daten über eine dafür vorgesehene Schnittstelle zu ihm geleitet werden. Im Gegensatz zu einer Smartcard ist ein HSM jedoch deutlich performanter und funktionsreicher. Die meisten HSMs bieten beispielsweise die Möglichkeit, eine Sicherungskopie des geheimen Schlüssels anzufertigen, was meist mithilfe des Secret-Sharings nach Shamir erfolgt (siehe Abschnitt 24.1.5). Darüber hinaus lässt sich mit einem guten HSM ein Vier-Augen-Prinzip (etwa für die Schlüsselgenerierung) realisieren. Die physikalischen Sicherheitsmaßnahmen sind aufwendiger als bei einer Smartcard. Ein HSM ist zudem teurer als eine Smartcard – die Preise beginnen bei etwa 5.000 Euro. Genutzt werden HSMs meist von Servern, die eine hohe Performanz sowie eine hohe Sicherheit gewährleisten müssen. Die Zertifizierungsinstanz (CA) einer Public-Key-Infrastruktur ist ein Beispiel dafür.

23.4 Kryptografie in eingebetteten Systemen

Zu den derzeit interessantesten Anwendungsgebieten der Kryptografie zählen zweifellos sogenannte **eingebettete Systeme**. Als eingebettetes System (Embedded System) bezeichnet man ein Hardwaremodul, das in ein technisches Gerät integriert ist, das vom Benutzer nicht als Computer wahrgenommen wird. Dieses Gerät kann beispielsweise ein Auto oder ein Staubsauger sein, aber auch ein Messgerät oder ein Mobiltelefon. Ein eingebettetes System besitzt keine PC-Tastatur und oft auch keinen Monitor. Der Benutzer kann das eingebettete System nicht frei programmieren. Eingebettete Systeme sind heutzutage allgegenwärtig (Ubiquitous Computing). Von der Spülmaschine über den Drucker bis zum MP3-Player kommt kaum ein anspruchsvolles technisches Gerät ohne ein integriertes Computermodul aus. Schätzungen gehen davon aus, dass auf jeden produzierten

PC etwa 50 eingebettete Computer kommen. In einem Auto sind heute bis zu 80 eingebettete Systeme im Einsatz.

Die Art, wie ein eingebettetes System realisiert wird, kann sehr unterschiedlich sein. In einem Geldautomaten oder Fahrscheindrucker kommt beispielsweise oft ein gewöhnlicher PC zum Einsatz. Die meisten eingebetteten Systeme sind dagegen weit weniger leistungsfähig als ein durchschnittlicher Bürorechner. Vor allem in Massenprodukten ist ein eingebettetes System oft ein wichtiger Kostenfaktor. Jeder Cent, der sich an der Hardware einsparen lässt, erhöht die Gewinnmarge des Herstellers.

Zu den wichtigsten Herausforderungen bei der Entwicklung eingebetteter Systeme zählt die Tatsache, dass der Hersteller nur schwer ein Softwareupdate vornehmen kann. Bugfixes, wie sie bei PC-Software weit verbreitet sind, sind bei eingebetteten Systemen kaum möglich. Bei einem Auto kann ein fehlerhaftes eingebettetes System sogar eine teure Rückrufaktion notwendig machen.

23.4.1 Eingebettete Systeme und Kryptografie

Es gehört nicht viel Fantasie dazu, sich die Notwendigkeit von Kryptografie in eingebetteten Systemen auszumalen. Das Auto ist ein typisches Beispiel dafür, an dem wir einige Szenarien durchspielen wollen:

- Die zahlreichen eingebetteten Systeme im Auto kommunizieren miteinander. Nicht auszudenken, was passiert, wenn Mallory falsche Daten einspielen kann. Authentifizierung ist hier also angebracht.
- Einige eingebettete Systeme im Auto zeichnen Diagnosedaten auf, die der Werkstatt bei der Reparatur und dem Hersteller beim Aufspüren von Schwachstellen helfen sollen. Diese Diagnosedaten sollen nicht in die Hände der Konkurrenz gelangen. Auch hier ist Authentifizierung notwendig, Verschlüsselung ebenfalls.
- Eingebettete Systeme kommen in der Motorsteuerung zur Anwendung und sorgen beispielsweise dafür, dass die Drehzahl im unkritischen Bereich bleibt. Viele Autohersteller verwenden sogar baugleiche Motoren für unterschiedliche Leistungsklassen (dies ermöglicht eine günstigere Produktion) und lassen ein eingebettetes System kontrollieren, dass der billigere Motor auch tatsächlich weniger Leistung bringt. Für gewiefte Bastler ist es reizvoll, die Funktionsweise eines Motorsteuerungssystems zu manipulieren (Chip-Tuning). Für den Hersteller kann dies sehr negative Folgen haben. So können getunte Chips für Umsatzverlust sorgen (weil der Kunde den billigeren Motor kauft). Außerdem sorgt Chip-Tuning unweigerlich für Defekte und Unfälle. Die Kryptografie kann dazu beitragen, Chip-Tuning zu verhindern.

Die genannten Aufgabenstellungen lassen sich leicht auf andere eingebettete Systeme übertragen. Angesichts der Vielzahl der eingebetteten Systeme dürfte die Bedeutung der Kryptografie in diesem Bereich einleuchten.

23.4.2 Kryptografische Herausforderungen in eingebetteten Systemen

Für den Einsatz in eingebetteten Systemen muss die Kryptografie zwar nicht neu erfunden werden. Dies heißt jedoch noch lange nicht, dass wir alle bisher betrachteten Verfahren und Protokolle unbesehen auf diesen Bereich übertragen können. Stattdessen gilt es, einige spezielle Rahmenbedingungen zu beachten. Diese wollen wir uns im Folgenden näher anschauen.

Beschränkte Ressourcen

Da die meisten eingebetteten Systeme nur wenig Rechenleistung und Speicherplatz zur Verfügung stellen, müssen die verwendeten kryptografischen Verfahren besonders ressourcenschonend realisiert sein. Um dieser Rahmenbedingung Rechnung zu tragen, gibt es zwei Ansätze:

- Manche eingebetteten Systeme verwenden einen speziellen **Krypto-Ko-Prozessor**. [SikStr] stellt beispielsweise einen Krypto-Chip vor, der Daten mit dem DES oder dem Triple-DES verschlüsselt. Durch die Hinzunahme dieses Krypto-Ko-Prozessors stieg der Durchsatz bei einer DES-Verschlüsselung von 1,3 KByte pro Sekunde auf 94 KByte pro Sekunde – dies entspricht einer Steigerung um den Faktor 72. Krypto-Ko-Prozessoren sind in den letzten Jahren zwar stark im Preis gesunken, ein wichtiger Kostenfaktor sind sie in vielen Anwendungen aber nach wie vor.
- Die begrenzten Ressourcen machen Verfahren und Protokolle attraktiv, die eine hohe Performanz und einen niedrigen Speicherverbrauch haben.

Da dies ein Kryptografiebuch und kein Hardwarebuch ist, interessiert uns vor allem der zweite Punkt dieser Aufzählung. Nahezu alle neueren symmetrischen Verfahren verbrauchen vergleichsweise wenig Ressourcen. Beim AES-Wettbewerb zählte eine entsprechende Anforderung zu den Auswahlkriterien. Kein Wunder, dass sich der AES bei den Herstellern eingebetteter Systeme großer Beliebtheit erfreut. Es gibt jedoch zahlreiche noch performantere Alternativen, beispielsweise die Algorithmen PRESENT und SEA sowie auf Hardware ausgelegte Stromchiffren.

Betrachtet man asymmetrische Verfahren, dann dürfte klar sein, dass die ansonsten so beliebten RSA- und DLSS-Verfahren nicht erste Wahl für eingebettete Systeme sind. Der derzeitige Eingebettete-Systeme-Boom ist daher auch eine wesentliche Triebkraft für den Einsatz von ECC-Krypto-Systemen. Auch HEC-Verfahren, XTR und NTRU könnten in den nächsten Jahren von dieser Entwicklung profitieren.

Fehlende Hardwareunterstützung

Es gibt zwei wichtige Hilfsmittel der Kryptografie, die in eingebetteten Systemen oft schmerzlich vermisst werden:

- *Uhr*: Viele eingebettete Systeme können ihren Zweck ohne Systemuhr erfüllen. Die Tatsache, dass in vielen kryptografischen Protokollen Zeitstempel eine wichtige Rolle spielen und diese nur mit einer Uhr zu generieren sind, ist für den Hersteller meist kein ausreichend starkes Argument, an diesem Zustand etwas zu ändern. Für den Kryptografen bleibt daher nur eine Möglichkeit: Er muss auf Protokolle setzen, die ohne Zeitstempel auskommen. Der Preis dafür ist meist ein zusätzlicher Protokollschritt.
- *Zufallsgenerator*: Ein echter Zufallsgenerator ist für ein eingebettetes System oft ein Luxus. Wenn ein eingebettetes System mit einem leistungsfähigeren Rechner kommuniziert, muss daher Letzterer das Generieren von Schlüsseln übernehmen. Sind dagegen beide Kommunikationspartner eingebettete Systeme, dann bleibt oft nichts anderes übrig, als einen einmal konfigurierten Schlüssel über längere Zeit zu verwenden.

Interaktion

Da ein eingebettetes System nicht wie ein PC über Maus, Tastatur und Monitor ansprechbar ist, müssen sich die Hersteller andere Möglichkeiten überlegen, wie der Nutzer (falls notwendig) mit dem Modul kommunizieren kann. Dies ist oft keine triviale Aufgabe, zumal Benutzerfreundlichkeit in der Kryptografie Pflicht ist. So ist es für die Entwickler eines MP3-Players zweifellos eine Herausforderung, eine Passworteingabe zu konzipieren.

Zugriff auf Sender und Empfänger

Entgegen dem klassischen Alice-Bob-Mallory-Modell hat Mallory bei eingebetteten Systemen meist nicht nur Zugriff auf abgehörte Daten, sondern auch auf die Endgeräte. Physikalische Angriffe und Seitenkanalangriffe sind bei eingebetteten Systemen daher fast immer eine realistische Bedrohung. Was man dagegen tun kann, steht in Kapitel 17.

Beschränkte Wartungsmöglichkeiten

Natürlich wirkt es sich auch auf die verwendete Kryptografie aus, dass die Firmware eines eingebetteten Systems nicht oder nur unter großem Aufwand ausgetauscht werden kann. Eine Folge besteht darin, dass die Entwicklung des Krypto-Programmcodes mit derselben Sorgfalt erfolgen muss wie bei allen anderen Codebestandteilen des eingebetteten Systems. Welche Entwicklungs- und Qualitätssicherungstechniken an dieser Stelle einzusetzen sind, ist nicht Thema dieses Buchs. Interessant ist dagegen die Frage, welche Verfahren und Protokolle für einen Langzeiteinsatz ohne Wartungsmöglichkeit geeignet sind. Leider ist mir keine Veröffentlichung bekannt, die diese Frage untersucht. Sie ist auch nicht leicht zu beantworten, da nicht einmal die Kriterien einfach zu formulieren sind.

23.5 RFID und Kryptografie

Zu den derzeit wichtigsten Zukunftsthemen der gesamten Informationstechnologie gehört zweifellos die **RFID**-Technik. RFID steht für Radio Frequency Identification und wird im Deutschen auch als Funkerkennung bezeichnet. Dahinter verbirgt sich eine Technik, bei der ein kleiner Computerchip (**RFID-Chip**, auch RFID-Tag genannt) per Funk mit einer Basisstation (RFID-Leser) verbunden ist. Die maximale Distanz, über die Chip und Leser miteinander kommunizieren können, beträgt meist nur ein paar Meter. Man unterscheidet zwischen aktiven und passiven RFID-Chips. Die aktive Variante enthält eine Batterie und versorgt sich dadurch selbst mit Strom. Passive RFID-Chips enthalten dagegen keine Batterie, sondern gewinnen die benötigte Energie per Induktion aus dem Funksignal der Basisstation. Für eine ausführliche Einführung zum Thema RFID empfehle ich [Finken] und [GilHan].

Die meiste Aufmerksamkeit innerhalb der RFID-Technik erhalten derzeit sogenannte **EPC**-Chips. EPC steht für Electronic Product Code und bezeichnet eine Bit-Folge, die je nach Standard zwischen 64 und 204 Bit lang ist. EPC-Chips zeichnen sich durch ihre Einfachheit aus. Ihre Funktionalität beschränkt sich im Wesentlichen darauf, einen EPC zu speichern und diesen auf Anfrage an die Basisstation zu übermitteln. Ein EPC-Chip ist oft kleiner als ein Quadratmillimeter und so dünn wie Papier. In sehr großen Stückzahlen liegt der Stückpreis bei kaum mehr als 5 Cent.

In den folgenden Ausführungen werde ich mich auf EPC-Chips beschränken. Andere (leistungsfähigere) RFID-Chips können wir dagegen zu den eingebetteten Systemen zählen. Das erklärte Ziel der RFID-Anbieter ist es, mit EPC-Chips die bisher üblichen Strichcodes zu ersetzen. Man kann die kleinen Chips an Gegenständen aller Art befestigen und kontaktlos auslesen. Durch die geringen Produktionskosten sind EPC-Chips auch und gerade für Massenanwendungen interessant. Ein wichtiger Aspekt dabei ist, dass der EPC Informationen wie die Art des Gegenstands und eine eindeutige Seriennummer enthalten kann.

Anwendungsmöglichkeiten für EPC-Chips gibt es zuhauf. So kann beispielsweise ein Supermarkt seine Artikel mit EPC-Chips ausstatten. Wenn Alice einkauft, erfasst eine Basisstation an der Kasse in Sekundenschnelle alle Waren in ihrem Einkaufswagen und berechnet den Preis. Wenn Alice anschließend irgendwelche Produkte in den Kühlschrank legt, kann dieser per RFID-Abfrage kontrollieren, ob das Mindesthaltbarkeitsdatum der darin befindlichen Waren abgelaufen ist. In ähnlicher Form kann der Mikrowellenherd automatisch ermitteln, wie lange er die eingestellte Suppe erhitzen muss. Selbst die Müllsortierung kann durch eine RFID-Unterstützung effektiver ablaufen.

Weitere EPC-Anwendungen ergeben sich im Unternehmen, wo die kleinen Chips beispielsweise die Inventarisierung erleichtern. Auch im Zahlungsverkehr – denkbar ist unter anderem eine Brieftasche, die anzeigt, wie viel Geld sie enthält – sind EPC-Chips nutzbar. Möglich wäre zudem ein EPC-Einsatz zur Authentifi-

zierung (als Schlüsselersatz oder auf Ausweisen), doch dafür wird man in absehbarer Zeit noch stärkere RFID-Varianten benötigen.

23.5.1 Sicherheitsprobleme beim Einsatz von EPC-Chips

Es ist offensichtlich, dass sich beim Einsatz von EPC-Chips ein wesentliches Sicherheitsproblem stellt. Dieses besteht darin, dass der Inhalt eines Chips auslesbar ist, ohne dass der Besitzer etwas davon bemerkt. Daher kann auch Bösewicht Mallory nach Herzenslust alle EPC-Chips auslesen, die in Reichweite seiner Basisstation gelangen. Gelingt ihm das Auslesen nicht – beispielsweise weil der Chip vom Benutzer zum Auslesen freigeschaltet werden muss –, dann kann Mallory immer noch versuchen, die Kommunikation zwischen einem Chip und einer Basisstation abzuhören.

Unbefugtes Auslesen eines EPC-Chips

Wenn es Mallory gelingt, EPCs routinemäßig auszulesen, dann kann dies unangenehme Folgen haben. Dies zeigen die folgenden Beispiele:

- Geht Mallory mit einer Basisstation in der Tasche durch einen Laden, dann kann er innerhalb von Minuten sämtliche Artikel mit EPC-Chip erfassen, die in den Regalen lagern. Wirtschaftsspionage könnte kaum einfacher sein.
- Platziert sich Mallory unauffällig in der Nähe der Ladenkasse, dann kann er den Inhalt einer Einkaufstüte auslesen. Dies ist ein Albtraum für jeden Datenschützer.
- Werden EPC-Chips auf Geldscheinen eingesetzt, dann kann Mallory unbemerkt feststellen, wie viel Geld sein Gegenüber in der Tasche hat.
- Trägt eine Person einen EPC-Chip mit sich (etwa am Handy oder auf einer Eintrittskarte), dann lässt sich mit geeignet positionierten Basisstationen ein detailliertes Bewegungsprofil erstellen. Im Extremfall führt dies zum gläsernen Bürger.

Es bedarf keiner allzu großen Fantasie, um diese Liste mit weiteren Beispielen fortzusetzen. Das Auslesen einer EPC sollte daher für Unbefugte nicht ohne Einschränkungen möglich sein.

Klonen eines EPC-Chips

Weitere Missbrauchsmöglichkeiten ergeben sich, wenn Mallory einen EPC-Chip fälscht. Sobald er den EPC kennt, ist dies technisch nicht allzu schwierig. So kann Mallory etwa das Kassensystem eines Supermarkts sabotieren, indem er zusätzliche EPC-Chips an Verkaufsartikeln anbringt. Auf ähnliche Weise lässt sich auch jedes Inventarisierungssystem im Unternehmen zum Zusammenbruch führen. Gefährlich wird das Klonen vor allem dann, wenn ein EPC-Chip zur Autenthifizierung eingesetzt wird. Dies ist aber (wie erwähnt) nicht der Fall.

Nichtkryptografische Gegenmaßnahmen

Es gibt verschiedene nichtkryptografische Gegenmaßnahmen, die Mallory das Handwerk beim Auslesen und Klonen von EPC-Chips erschweren:

- Alle gängigen EPC-Chips unterstützen einen sogenannten Kill-Befehl. Empfängt der Chip diesen Befehl (inklusive einem Passwort, das Missbrauch vermeiden soll), dann stellt er seinen Betrieb ein und ist nicht mehr nutzbar. Dies kann beispielsweise der Betreiber eines Supermarkts nutzen, indem er jeden EPC-Chip nach dem Auslesen an der Kasse per Kill-Befehl ausschaltet. Dadurch wird zwar vermieden, dass Mallory den Inhalt von Alices Einkaufstasche ausliest, doch dafür weiß nun auch Alices Kühlschrank nicht mehr, welche Waren abgelaufen sind.
- Es gibt zudem die Möglichkeit, mit sogenannten **Blockern** zu arbeiten. Ein Blocker funktioniert ähnlich wie ein Störsender. Er sorgt durch eigene Signale dafür, dass eine Basisstation die Nachricht eines EPC-Chips nicht mehr empfangen kann. Ein Blocker kann beispielsweise in Alices Einkaufstasche oder in ihrer Geldbörse zum Einsatz kommen. Es liegt allerdings auf der Hand, dass auch der Einsatz von Blockern nicht alle EPC-Sicherheitsprobleme löst.

Für weitere Informationen zum Thema RFID, EPC und die damit verbundenen Sicherheitsrisiken empfehle ich die Lektüre von [GaJuPa]. Bücher zu diesem Thema sind [LinTho] und [GarRos]. An dieser Stelle bleibt festzuhalten, dass ein EPC-Chip ohne Kryptografie keine ausreichende Sicherheit bietet.

23.5.2 RFID und Kryptografie

Da die nichtkryptografischen Sicherheitsmaßnahmen im Zusammenhang mit EPC-Chips nicht so recht funktionieren, müssen wir es mit kryptografischen versuchen. Bevor wir in dieses Thema einsteigen, betrachten wir jedoch die speziellen Rahmenbedingungen, die sich beim Krypto-Einsatz auf EPC-Chips stellen. Eines ist dabei offensichtlich: Auf einem EPC-Chip stehen extrem wenig Ressourcen zur Verfügung. Da ein Exemplar maximal ein paar Cent kosten darf (und selbst das ist für einen Strichcode-Ersatz im Supermarkt noch zu viel), ist die Hardwareausstattung eines EPC-Chips um ein bis zwei Größenordnungen kleiner als beispielsweise bei einer Smartcard.

Dass die Sicherheitsprobleme von EPC-Chips trotz der spartanischen Hardwareausstattung lösbar sind, liegt am vergleichsweise niedrigen Bedrohungspotenzial. Wir können davon ausgehen, dass Mallory keine teure Rechner-Armada wochenlang beschäftigen wird, nur um den EPC-Chip auf einer Packung Nudeln klonen zu können. Realistischerweise können wir eher von folgendem Szenario ausgehen:

- Mallory investiert höchstens ein paar Minuten pro EPC-Chip.
- Mallory hat keinen physikalischen Zugang zum EPC-Chip. Seitenkanalangriffe sind dadurch kaum möglich.
- Die Zeit, in der ein EPC-Chip für Mallory zugänglich ist, ist auf ein paar Stunden begrenzt.

Die große Herausforderung der EPC-Kryptografie besteht also darin, mit minimalistischen Mitteln eine moderate Sicherheit zu gewährleisten. Hierbei gibt es noch eine weitere spezielle Rahmenbedingung: Ein Authentifizierungsprotokoll sieht normalerweise vor, dass der zu Authentifizierende im ersten Schritt seinen Namen (bzw. eine Kennung) an den Authentifizierer schickt (ansonsten weiß Letzterer nicht, mit wem er es zu tun hat, und kann dadurch beispielsweise schlecht die Korrektheit des Passworts prüfen). Im Falle von EPC-Chips ist dies jedoch problematisch, da eine Kennung Mallory bereits reicht, um beispielsweise ein Bewegungsprofil erstellen zu können (die im EPC enthaltenen Nutzinformationen erhält er auf diese Weise allerdings nicht). Soll dies vermieden werden, dann muss sich die Identität des EPC-Chips implizit aus den Authentifizierungsdaten ergeben.

Und noch eine Schwierigkeit beim Einsatz von EPC-Chip-Kryptografie ist zu beachten: Das Schlüsselmanagement stellt nahezu unlösbare Probleme. Mehr dazu gibt es im Fazit dieses Kapitels.

EPC-Verschlüsselung

Ein einfaches kryptografisches Hilfsmittel, das einige EPC-Sicherheitsprobleme löst, sind sogenannte **EPC-Pseudonyme** (meist einfach nur als **Pseudonyme** bezeichnet). Wird ein solches verwendet, dann speichert der EPC-Chip statt dem EPC eine andere Bit-Folge (Pseudonym) und sendet diese bei Bedarf an die Basisstation. Das Pseudonym kann der symmetrisch verschlüsselte EPC sein (er wird bereits in verschlüsselter Form auf den Chip gebracht). Hat die Basisstation das Pseudonym ausgelesen, dann ordnet sie dieses durch eine Entschlüsselung (die Basisstation kennt den Schlüssel) oder mithilfe einer geheimen Zuordnungstabelle dem richtigen EPC zu. Beachten Sie, dass der EPC-Chip bei diesem Prozedere keine kryptografische Operation durchführen muss.

Durch ein Pseudonym wird zunächst einmal verhindert, dass Mallory den Inhalt des EPC lesen kann. Ein Bewegungsprofil kann er jedoch trotzdem erstellen. Deshalb verwenden viele Implementierungen mehrere Pseudonyme pro EPC, wobei der Chip diese im Wechsel abgibt. Allerdings machen es die beschränkten Ressourcen auf dem EPC-Chip notwendig, dass ein Satz Pseudonyme immer wieder von neuem zum Einsatz kommt. Damit Mallory nicht in Sekundenschnelle alle Pseudonyme nacheinander auslesen kann, findet ein Wechsel nicht ständig, sondern nur alle paar Minuten statt.

Pseudonyme auf einem EPC-Chip sind zwar eine schöne Sache, aber kein Allheilmittel. Insbesondere können sie ein EPC-Chip-Klonen nicht vollständig ver-

hindern. Mallory kann nämlich jedes ausgelesene Pseudonym auf einen eigenen Chip aufbringen und damit zumindest einen Teilklon herstellen, mit dem er einigen Schaden anrichten kann.

EPC-Authentifizierung

Wenn wir Mallorys EPC-Angriffe wirksam bekämpfen wollen, müssen wir etwas tiefer als bisher in die kryptografische Trickkiste greifen. Genauer gesagt benötigen wir eine geeignete Authentifizierung. Wenn wir ein (teilweises) Klonen des Chips verhindern wollen, dann muss die Basisstation den Chip authentifizieren. Um zu erreichen, dass auch eine Überwachung (Erstellen eines Bewegungsprofils) nicht funktioniert, muss zudem der Chip die Basisstation authentifizieren. Die einfachste Form der Authentifizierung ist bekanntlich die unverschlüsselte Passwortabfrage. Da wir jedoch davon ausgehen, dass Mallory die Kommunikation zwischen Chip und Basisstation abhört, kommt diese Methode nicht infrage. Wir benötigen stattdessen ein wechselseitiges Challenge-Response-Verfahren. Dieses könnte etwa so aussehen:

1. Der Chip sendet eine Kennung (z.B. ein EPC-Pseudonym oder einen Teil davon) und eine Challenge C_1 an die Basisstation. Es können auch mehrere EPC-Pseudonyme pro Chip vorhanden sein, um Mallory das Erstellen eines Bewegungsprofils zu erschweren.
2. Die Basisstation berechnet mit einem symmetrischen Verschlüsselungsverfahren oder einer schlüsselabhängigen Hashfunktion aus C_1 die Response R_1 und sendet diese mit einer weiteren Challenge C_2 an den Chip.
3. Der Chip wendet eine schlüsselabhängige Hashfunktion oder ein symmetrisches Verschlüsselungsverfahren auf die Challenge C_2 an und berechnet die Response R_2. Diese geht mit dem Pseudonym zurück an die Basisstation.
4. Die Basisstation überprüft die Response R_2 und ordnet das erhaltene Pseudonym dem passenden EPC zu (durch eine Entschlüsselung oder mithilfe einer Zuordnungstabelle).

Dieses Protokoll ist natürlich nur ein Prototyp, der noch die eine oder andere Schwäche enthält. Bevor Sie ein Protokoll zur EPC-Authentifizierung implementieren, sollten Sie sich daher in der einschlägigen Literatur über die Details erkundigen [ChLeBu, Juel05/1, Juel05/2, Juel05/3].

EPC-taugliche Krypto-Verfahren

Ein brauchbares Protokoll zur EPC-Authentifizierung setzt voraus, dass ein EPC-Chip ein Verschlüsselungsverfahren oder eine schlüsselabhängige Hashfunktion abarbeiten kann. Dies ist keine triviale Anforderung, denn für die Ausführung eines AES oder SHA-1 sind die gängigen EPC-Chips schlichtweg zu schwach. Daran wird sich so schnell auch nichts ändern. Erst wenn der Preis für einen EPC-

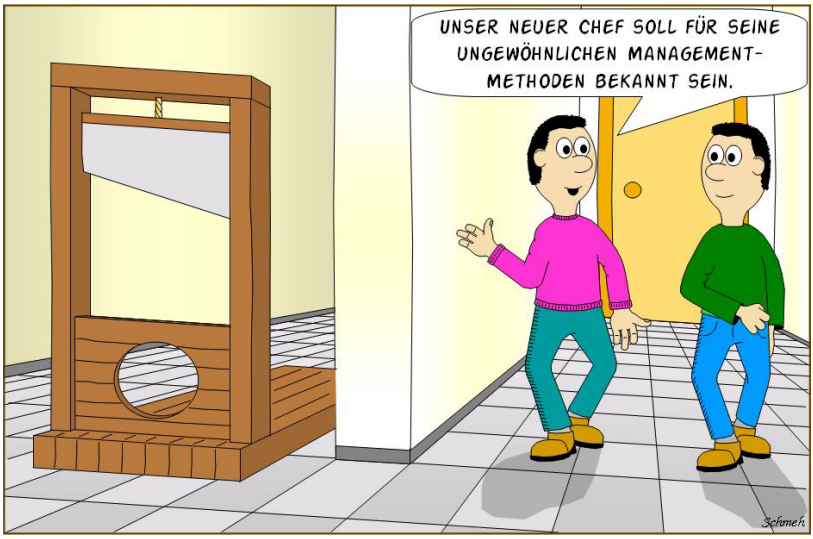
Chip deutlich unter einen Cent fällt, wird beispielsweise ein Ladenbetreiber einen Chip verwenden wollen, dessen Ressourcen zu einem Großteil von kryptografischen Verfahren verbraucht werden. Um dieses Problem zu lösen, haben verschiedene Autoren unterschiedliche Lösungswege vorgeschlagen:

- Juels schlägt ein Protokoll vor, bei dem die Verschlüsselung durch eine Tabelle realisiert wird, die für jede mögliche Challenge die passende Response enthält [Juel05/3]. Diese Lösung spart zwar Rechenkapazität, doch dafür ist die Anzahl von Authentifizierungen durch die Größe der Tabelle begrenzt.
- Die asiatischen Kryptografen Duc, Kim, Lee und Park beschreiben ein weiteres Protokoll zur EPC-Authentifizierung [DuKiLP]. Dieses verwendet eine Hashfunktion, die auf einer Prüfsumme basiert und sehr simpel ist. Es handelt sich dabei jedoch nicht um eine kryptografische Hashfunktion, weshalb das Protokoll wohl mit überschaubarem Aufwand zu knacken ist. Den Autoren der besagten Arbeit ist dies bewusst. Sie sehen die eingesetzte Hashfunktion lediglich als Vorschlag an, der jederzeit durch ein besseres Verfahren ersetzt werden kann.
- Inzwischen gibt es auch Verschlüsselungsverfahren, die speziell für EPC-Chips und andere Low-End-Umgebungen entwickelt worden sind. Beispiele dafür sind Present und SEA (siehe Abschnitt 10.3).

Fazit

Kryptografie für EPC-Chips ist ein noch junges und sehr aktives Forschungsgebiet. Wir dürfen gespannt sein, welche neuen Protokolle und Verfahren wir in den nächsten Jahren noch sehen werden. Dabei ist jedoch eines zu beachten: Das Schlüsselmanagement im Zusammenhang mit EPC-Chips ist eine ausgesprochen schwierige Angelegenheit. Der Grund dafür liegt auf der Hand: Kommen EPC-Chips innerhalb eines Supermarkts zum Einsatz, so ist es kein Problem, auf den Chips Schlüssel unterzubringen, die nur die Basisstation an der Kasse kennt. Schwierig wird es, wenn Kundin Alice ihre Ware mit nach Hause nimmt und nun auf einmal auch ihr Kühlschrank die jeweiligen Schlüssel kennen soll. Im Grunde müssen sogar alle Kühlschränke der Stadt die entsprechenden Schlüssel kennen, wenn der Supermarkt alle seine Kunden zufriedenstellen will. In der Praxis ist so etwas natürlich nicht möglich. Man wird also noch einiges an Forschungsaufwand investieren müssen, um einen sicheren Umgang mit EPC-Chips zu gewährleisten. Die Kryptografie kann nur einen Teil zur Lösung der zahlreichen Fragestellungen beitragen – allerdings einen sehr spannenden Teil.

24 Weitere kryptografische Werkzeuge



In diesem Kapitel werden wir uns mit drei weiteren Themen beschäftigen, die es bei der Implementierung von Kryptografie zu beachten gilt: Schlüsselmanagement, Trusted Computing und Krypto-APIs. Diese Themen haben zwar wenig miteinander zu tun, aber doch zwei Dinge gemeinsam: Sie sind wichtig und passen in kein anderes Kapitel dieses Buchs.

24.1 Management geheimer Schlüssel

In unserem Alice-Bob-Mallory-Modell sind wir bisher immer davon ausgegangen, dass Mallory zwar die Verfahren, aber nicht die geheimen Schlüssel kennt. Diese Voraussetzung ist zwar eine sinnvolle Arbeitshypothese, allerdings keine Selbstverständlichkeit. Im Gegenteil: Alice und Bob müssen äußerst vorsichtig sein, wenn sie ihre Schlüssel vor Mallory verbergen wollen. In diesem Unterkapi-

tel schauen wir uns die wichtigsten Maßnahmen an, die Alice und Bob für diesen Zweck zur Verfügung stehen. Es geht also um **Schlüsselmanagement**.

Im Folgenden gehen wir davon aus, dass Alice und Bob ein symmetrisches Verschlüsselungsverfahren verwenden, um miteinander zu kommunizieren – etwa den AES. Die meisten Überlegungen in diesem Kapitel gelten jedoch auch für die privaten Schlüssel asymmetrischer Verfahren, da Alice und Bob diese ebenfalls geheim halten müssen. Für das asymmetrische Schlüsselmanagement ist zudem das Thema Public-Key-Infrastrukturen zu beachten, um das es in Kapitel 26 geht. Der Aufbau der folgenden Unterkapitel folgt der Reihenfolge, in der die einzelnen Aufgaben des Schlüsselmanagements anfallen (Key Lifecycle).

24.1.1 Schlüsselgenerierung

Die erste Frage, die Alice und Bob sich stellen müssen, lautet: Wie sollen sie die verwendeten Schlüssel generieren? Bei einem asymmetrischen Verfahren wie RSA gibt der Algorithmus die Schlüsselgenerierung vor. Bei symmetrischen Verfahren gibt es dagegen verschiedene Möglichkeiten:

- *Zufallsgenerator*: Die einfachste und offensichtlichste Methode besteht darin, dass Alice und Bob ihren Schlüssel per Zufallsgenerator generieren. Dabei sollten sie die in Kapitel 15 erwähnten Aspekte beachten.
- *Diffie-Hellman*: Der Diffie-Hellman-Schlüsselaustausch sieht die implizite Generierung eines geheimen Schlüssels vor.
- *Passwort*: Alice und Bob können ihren Schlüssel mithilfe einer kryptografischen Hashfunktion von einem Passwort ableiten. Eine solche Methode wird im Standard PKCS#5 beschrieben (siehe unten). Der Vorteil dieser Methode liegt darin, dass Alice und Bob sich ein Passwort merken können, einen 128-Bit-Schlüssel aber nicht. Da wir davon ausgehen, dass Mallory die Methode kennt, mit der die beiden ihr Passwort in einen Schlüssel umwandeln, müssen Alice und Bob unbedingt ein sicheres Passwort wählen.
- *Biometrische Verschlüsselung*: Eine besonders pfiffige Methode zur Schlüsselgenerierung besteht darin, den Schlüssel aus einem biometrischen Merkmal (etwa dem Fingerabdruck) abzuleiten. Dies ist eine recht komplexe Aufgabe, da verschiedene biometrische Messungen bei derselben Person nie genau das gleiche Ergebnis liefern. Dennoch muss bei jeder Messung dieselbe Bit-Folge abgeleitet werden, die andererseits bei unterschiedlichen Personen nicht identisch sein darf. In [HaAnDa] gibt es mehr zu diesem Thema, das bisher in der Praxis kaum angewandt wird.

Wie erwähnt, beschreibt der Standard **PKCS#5** eine Methode zur Generierung eines Schlüssels aus einem Passwort [PKCS#5]. Diese Methode wurde von der IETF übernommen und als RFC veröffentlicht [RFC2898]. Genau genommen handelt es sich um zwei unterschiedliche Methoden, die als **PBKDF1** und

PBKDF2 bezeichnet werden (die sperrige Abkürzung steht für *Password Based Key Derivation Function*). PBKDF1 ist die ältere Methode. Sie stammt aus der ersten PKCS#5-Version und ist in der aktuellen Version 2.0 des Standards nur noch aus Kompatibilitätsgründen enthalten. PBKDF2 ist das in PKCS#5v2.0 empfohlene Verfahren. Beide Varianten lassen einige Details offen und sind damit verglichen mit anderen Standards recht unpräzise. Die Entwickler des Standards gingen davon aus, dass PBKDF1 und PBKDF2 weniger zur Kommunikation zwischen Alice und Bob, sondern eher zur Dateiverschlüsselung durch eine einzelne Person verwendet wird – eine bis aufs Bit genaue Standardisierung ist in diesem Fall nicht ganz so wichtig.

PBKDF1 ist recht simpel aufgebaut. Das Verfahren sieht vor, eine kryptografische Hashfunktion auf ein Passwort anzuwenden, um so einen Schlüssel zu generieren. Um Wörterbuch-Angriffe zu erschweren, wird die Hashfunktion mehrfach durchlaufen (der Standard empfiehlt mindestens 1.000 Durchläufe), und es kommt ein Salt zum Einsatz. Die Länge des Salt-Werts ist nicht vorgegeben. Als Hashfunktion gibt der Standard wahlweise MD2, MD5 oder SHA-1 vor. Die maximale Länge des abgeleiteten Schlüssels entspricht der Hashwert-Länge, also 128 Bit (bei MD2 und MD5) oder 160 Bit (bei SHA-1).

PBKDF2 ist etwas komplexer, erlaubt dafür jedoch die Generierung beliebig langer Schlüssel. Das Verfahren benötigt eine Funktion, die sich als Pseudozufallsgenerator nutzen lässt. Der PKCS#5-Standard nennt SHA-1-HMAC als einzigen Vertreter. PBKDF2 lässt sich am besten an einem konkreten Beispiel erklären. Wir nehmen an, Alice benötige einen 512-Bit-Schlüssel, der sich aus einem Passwort (etwa 10 Byte) ableitet. Da der benötigte Schlüssel länger ist als ein SHA-1-HMAC-Hashwert (160 Bit), wäre PBKDF1 an dieser Stelle ungeeignet. Außerdem verwendet Alice ein Salt zur Erschwerung von Wörterbuch-Angriffen. Dies kann etwa ein 8-Byte-Wert sein, der nicht geheim ist. Wir gehen zudem davon aus, dass Alice 1.000 SHA-1-HMAC-Durchläufe vorsieht, was ebenfalls Wörterbuch-Angriffe verlangsamen soll.

PBKDF2 sieht vor, dass Alice ihren Schlüssel nötigenfalls aus mehreren Hashwerten zusammensetzt. Zu einem 512-Bit-Schlüssel kommt Alice, indem sie vier Hashwerte verwendet (insgesamt 640 Bit) und die überschüssigen 128 Bit am Ende ignoriert. Die ersten 160 Bit des Schlüssels generiert Alice mit folgenden Formeln (»||« steht für die Konkatenation):

$$U_1 = HMAC_{\text{Passwort}}(Salt \| 1)$$
$$U_2 = HMAC_{\text{Passwort}}(U_1)$$
$$U_3 = HMAC_{\text{Passwort}}(U_2)$$
$$\ldots$$
$$U_{1000} = HMAC_{\text{Passwort}}(U_{999})$$

Die ersten 160 Bit des Schlüssels sind: $U_1 \oplus U_2 \oplus U_3 \oplus \ldots \oplus U_{1000}$. Die drei weiteren Teile des Schlüssels berechnet Alice auf dieselbe Weise, nur dass beim Berechnen von U_1 nicht der Wert 1, sondern entsprechend 2, 3 oder 4 eingeht.

Vom vierten Schlüsselteil werden nur die ersten 32 Bit verwendet, da Alice insgesamt nur 512 Bit benötigt. Wie man leicht sieht, lässt sich der beschriebene Ablauf auf eine beliebige Schlüssellänge und eine beliebige Zahl von HMAC-Durchläufen verallgemeinern. Auch die Hashwert-Länge lässt sich ändern. Der PKCS#5-Standard lässt Schlüssellänge, Passwortlänge, Salt-Länge und die Zahl der Durchläufe offen. Statt SHA-1-HMAC kann auch ein anderes Verfahren zum Einsatz kommen.

24.1.2 Schlüsselspeicherung

Ein weiteres kritisches Thema, das Alice und Bob beachten müssen, ist die Speicherung ihrer geheimen Schlüssel. In einer Hardwareimplementierung oder einem eingebetteten System ist dies meist eine Aufgabe des Chip-Designs. Auf einem PC stellen sich dagegen auch softwaretechnische Fragen. Natürlich muss ein geheimer Schlüssel für die Software zugänglich sein, die ihn nutzt. Andererseits sollte ein geheimer Schlüssel nicht auslesbar sein, wenn der Computer, auf dem die Software installiert ist, Mallory in die Hände fällt. Wiederum gibt es mehrere Möglichkeiten, dies zu bewerkstelligen:

- *Smartcard*: Smartcards, USB-Tokens und HSMs gelten als beste Form der Schlüsselspeicherung. Trusted Computing erfüllt einen ähnlichen Zweck.
- *Datei*: Nicht ideal, aber in der Praxis die häufigste Methode ist das Speichern des Schlüssels in einer verschlüsselten Datei auf der Festplatte. Dazu gibt es den Standard PKCS#8, der weiter unten in diesem Unterkapitel erklärt wird.
- *Passwort*: Wenn Alice und Bob ihren Schlüssel von einem Passwort ableiten, dann entfällt das Speichern des Schlüssels, da dieser immer bei Bedarf berechnet werden kann.
- *Biometrische Verschlüsselung*: Hierfür gelten dieselben Überlegungen wie für Passwörter.

Ein Format für das verschlüsselte Ablegen eines Schlüssels wird im Standard **PKCS#8** beschrieben [PKCS#8]. Es handelt sich dabei um ein Format, das für die privaten Schlüssel asymmetrischer Verfahren gedacht ist, jedoch auch für symmetrische Verfahren verwendet werden kann. PKCS#8 sieht eine Datenstruktur vor, die neben dem Schlüssel noch einige weitere Angaben enthält – beispielsweise eine Versionsnummer und der OID des Algorithmus, für den der Schlüssel verwendet werden kann. Für die Verschlüsselung der Datenstruktur kann ein nach PKCS#5 generierter Schlüssel genutzt werden.

Bekommt Mallory Alices PKCS#8-Datei in die Finger (etwa, nachdem er ihren Laptop gestohlen hat), dann kann er versuchen, diese zu entschlüsseln. Gelingt ihm das, dann gelangt er an Alices privaten Schlüssel. Ein solches Vorgehen funktioniert in der Praxis oft mit einem Wörterbuch-Angriff, da der verwendete Schlüssel in der Regel von einem Passwort abgeleitet ist (z. B. mit PKCS#5).

Eine Methode zur Erschwerung von Wörterbuch-Angriffen in diesem Zusammenhang ist die sogenannte **Cryptographic Camouflage**. Diese wurde von der Firma Arcot entwickelt und patentiert [Gatz]. Es gibt bisher keinen Standard dafür. Die Cryptographic Camouflage sieht vor, dass der private Schlüssel in einem Format verwendet wird, in dem er nicht als solcher erkennbar ist. Für das RSA-Verfahren bedeutet dies, dass nur die Zahl d gespeichert wird – ohne jegliche Formatierung. Da d in der Praxis immer ungerade ist, wird zusätzlich das niederwertigste Bit bei der Speicherung weggelassen, um auch dieses Erkennungsmerkmal zu entfernen. In dieser Form legt Alice den privaten Schlüssel verschlüsselt ab, wobei sie den verwendeten Schlüssel wiederum von einem Passwort ableitet.

Durch die Cryptographic Camouflage kann Mallory nicht erkennen, ob ein Entschlüsselungsversuch mit geratenem Schlüssel erfolgreich war. Ein Wörterbuch-Angriff funktioniert dadurch nicht ohne Weiteres. Wirklich wirksam ist diese Technik jedoch nur, wenn Mallory den zugehörigen öffentlichen Schlüssel nicht kennt (ansonsten kann er mit diesem überprüfen, ob eine Bit-Folge der gesuchte private Schlüssel ist). Die Cryptographic Camouflage lohnt sich daher vor allem, wenn Alice und Bob auch ihre öffentlichen Schlüssel geheim halten. Dies ist dann möglich, wenn die Zahl der potenziellen Kommunikationspartner begrenzt ist – etwa in einer Client-Server-Umgebung. Für die organisationsübergreifende E-Mail-Verschlüsselung oder das digitale Signieren von gerichtsverwertbaren Dokumenten bringt dieser Ansatz jedoch nur wenige Vorteile.

24.1.3 Schlüsselauthentifizierung

Egal, ob Alice und Bob ihren Schlüssel auf der Festplatte, auf einer Smartcard oder in anderer Form speichern – vor der Schlüsselnutzung sollte auf jeden Fall eine Authentifizierung gegenüber dem Schlüsselspeicher stehen. Im Falle einer Smartcard ist dies in der Regel eine PIN, die der Nutzer eingeben muss. Wenn die Smartcard öfter benötigt wird, dann kann eine Implementierung die PIN auch speichern und automatisch eingeben (PIN-Caching). Setzen Alice und Bob dagegen eine verschlüsselte Datei zur Schlüsselspeicherung ein, dann dient ein Passwort, das per PKCS#5 zu einem Schlüssel umgerechnet wird, zur Authentisierung. Entsprechendes gilt auch für die biometrische Verschlüsselung.

24.1.4 Schlüsseltransport und Schlüssel-Backup

Es gibt zahlreiche Fälle, in denen Alice und Bob einen geheimen Schlüssel verschlüsselt übertragen müssen. Ein solcher Vorgang ist beispielsweise notwendig, wenn Alice Bob einen Schlüssel auf sichere Weise mitteilen will. Auch wenn Alice eine verschlüsselte Kopie ihres Schlüssels ablegen will, ist ein solcher Fall gegeben. Für den Schlüsseltransport gibt es wiederum mehrere Möglichkeiten:

- *Asymmetrische Verschlüsselung*: Ein asymmetrisches Verfahren wie RSA lässt sich für den Schlüsseltransport nutzen.
- *Key-Wrapping*: Oftmals ist es auch sinnvoll, einen geheimen Schlüssel zum Transport oder zur Ablage symmetrisch zu verschlüsseln (**Key Wrapping**). In [RFC3217] werden zwei Methoden dazu beschrieben: Das Verschlüsseln eines Triple-DES-Schlüssels mit einem Triple-DES-Schlüssel sowie das Verschlüsseln eines RC5-Schlüssels mit einem RC5-Schlüssel. [RFC3394] beschreibt eine Key-Wrapping-Technik auf Basis des AES, die für beliebige Schlüssel geeignet ist.

24.1.5 Schlüsselaufteilung

Manchmal ist es sinnvoll, wenn Alice und Bob nur gemeinsam über einen bestimmten Schlüssel verfügen können. Dies ist eine Form des Vier-Augen-Prinzips, das Sie aus Abschnitt 17.5 kennen. Es gibt unterschiedliche Möglichkeiten, um eine solche Anforderung umzusetzen. So kann Alice die ersten 64 Bit eines 128-Bit-Schlüssels erhalten, während Bob die zweiten 64 Bit besitzt – nur wenn die beiden zusammenlegen, haben sie den kompletten Schlüssel. Noch besser ist es, wenn Alice eine 128-Bit-Folge A und Bob eine 128-Bit-Folge B erhält, wobei sich der Schlüssel K mit folgender Gleichung berechnen lässt: $A \oplus B = K$. In ähnlicher Form können Alice, Bob und Carol einen Schlüssel auch auf drei Personen verteilen. Hier gilt folgende Formel: $A \oplus B \oplus C = K$. Eine solche Vorgehensweise nenn man *Key-Splitting*. In der Literatur wird das Thema meist etwas allgemeiner abgehandelt (es muss ja nicht unbedingt ein Schlüssel sein, der aufgeteilt wird) und als **Secret-Splitting** bezeichnet.

Mathematisch deutlich anspruchsvoller wird die Sache, wenn Alice, Bob und Carol den Schlüssel so aufteilen wollen, dass nur zwei der drei anwesend sein müssen, um den Schlüssel zu ermitteln. Man spricht hierbei von einem 2-aus-3-Prinzip oder im allgemeinen Fall von einem m-aus-n-Prinzip. Die Vorgehensweise bezeichnet man als *Key-Sharing* oder (allgemeiner) als **Secret-Sharing**. Wie Sie sich denken können, ist Secret-Sharing in der Praxis deutlich wichtiger als Secret-Splitting, da bei Letzterem der Verlust eines Schlüsselteils den Verlust des ganzen Schlüssels zur Folge hat. Eine gängige Anwendung des Secret-Sharing ist beispielsweise die Sicherung bzw. Wiederherstellung eines Schlüssels, der in einem HSM gespeichert ist.

Das am weitesten verbreitete Secret-Sharing-Verfahren stammt von dem israelischen Kryptografen Adi Shamir und wird daher als **Secret-Sharing nach Shamir** bezeichnet [Sham79]. Das Verfahren nutzt folgenden mathematischen Sachverhalt aus:

- Zwei Punkte legen eine Gerade eindeutig fest.
  Die Gleichung einer Geraden lautet $y=ax+b$.
- Drei Punkte legen eine Parabel eindeutig fest.
  Die Gleichung einer Parabel lautet $y=ax^2+bx+c$.
- Vier Punkte legen eine Kubik eindeutig fest.
  Die Gleichung einer Kubik lautet $y=ax^3+bx^2+cx+d$.
- m Punkte legen eine Polynom-Kurve der Ordnung m-1 eindeutig fest.
  Die Gleichung einer Polynom-Kurve der Ordnung m-1 lautet
  $y=a_{m-1}x^{m-1}+...+a_2x^2+a_1x+a_0$.

Das Secret-Sharing nach Shamir sieht vor, dass von n Beteiligten jeder einen Punkt auf einer Polynom-Kurve der Ordnung m-1 erhält. Wenn m der n Personen ihre Punkte zur Verfügung stellen, dann lässt sich die Kurve rekonstruieren. Dadurch ist ein m-aus-n-Prinzip gegeben. Das Shamir-Verfahren legt fest, dass der Schlüssel als Wert a_0 in die Gleichung eingeht. Die Werte für a_1, a_2 usw. sind beliebig. Zu den Punkten, die auf die n Personen verteilt werden, darf nicht der Punkt $(0,a_0)$ gehören, da dieser den Schlüssel preisgibt.

Betrachten wir ein Beispiel. Der zu verteilende Schlüssel sei die Zahl 10, und es sei ein 3-aus-5-Prinzip gewünscht. Wir benötigen also eine Polynom-Kurve zweiten Grades mit a_0=10. Folgende Gleichung beschreibt eine solche:

$$y=2x^2-5x+10$$

Nun brauchen wir fünf Punkte auf dieser Kurve, beispielsweise (1,7), (2,8), (3,13), (4,22) und (5,35). Diese fünf Punkte werden auf fünf Personen verteilt. Wenn nun die erste, die dritte und die vierte Person ihre Punkte zusammenlegen, dann können sie folgende Gleichungen bilden:

$$a_2 + a_1 + a_0 = 7$$
$$9a_2 + 3a_1 + a_0 = 13$$
$$16a_2 + 4a_1 + a_0 = 22$$

Es ist nicht schwierig, dieses Gleichungssystem zu lösen und den Wert von a_0 zu ermitteln. Dieser beträgt 10 und ist der gesuchte Schlüssel. In der Praxis werden diese Berechnungen modulo einer Primzahl durchgeführt, da die Zahlen sonst zu groß werden (a_0 kann beispielsweise 128 Bit lang sein). Die verwendete Primzahl muss größer als der Wert von a_0 (also als der Schlüssel) sein.

24.1.6 Schlüsselwechsel

Es versteht sich von selbst, dass Alice und Bob nicht für unbegrenzte Zeit denselben geheimen Schlüssel verwenden sollten. Die Frage ist jedoch, wie oft ein Schlüsselwechsel sinnvoll ist. Viele kryptografische Protokolle sehen vor, dass Alice und Bob für jede Kommunikation einen neuen Sitzungsschlüssel verwen-

den, der mithilfe eines dauerhaften Schlüssels (Masterschlüssel) generiert wird. Allerdings sollte auch der Masterschlüssel nicht bis ans Ende aller Tage derselbe bleiben. Leider gibt es keine verbindliche Regel über die ideale Lebensdauer eines Schlüssels. Stattdessen sollte der Betreiber einer Krypto-Implementierung nach einem gangbaren Kompromiss suchen: Wird der Schlüssel zu oft gewechselt, dann kann dies die Akzeptanz bei den Anwendern beeinträchtigen. Ein zu seltener Schlüsselwechsel ist dagegen eine potenzielle Sicherheitslücke.

Davon abgesehen kann ein häufiger Schlüsselwechsel die Sicherheit eines an sich schwachen Krypto-Verfahrens deutlich erhöhen. Diese Tatsache wird beispielsweise im analogen Bezahl-Fernsehen genutzt. Da analoge Verschlüsselungsverfahren leicht zu knacken sind, verwenden die gängigen Decoder notgedrungen ein schwaches Verfahren, wechseln jedoch den Sitzungsschlüssel sehr oft (etwa alle zehn Sekunden) – in der Hoffnung, dass Mallory mit dem unbefugten Entschlüsseln nicht nachkommt.

24.1.7 Löschen eines Schlüssels

Wenn Bob seinen geheimen Schlüssel in einer verschlüsselten Datei auf seinem PC ablegt, ist er damit noch nicht auf der sicheren Seite. Denn immer dann, wenn verschlüsselt, entschlüsselt oder signiert wird, muss der Schlüssel logischerweise für kurze Zeit im Klartext vorliegen. Solange der Schlüssel im flüchtigen Arbeitsspeicher bleibt, ist dies nicht weiter tragisch. Gelangt der Schlüssel dagegen auf die Festplatte, dann könnte Mallory davon profitieren, wenn er Alices Rechner klaut. Für die Implementierung von Krypto-Verfahren gilt daher eine eherne Regel: Ein Schlüssel, der auf ein Speichermedium gelangt, muss von dort umgehend wieder gelöscht werden. Gleiches gilt natürlich auch für den Klartext, mit dem die Implementierung arbeitet. Weitere Infos dazu gibt es in Abschnitt 17.3.

24.1.8 Key Recovery

Verschlüsselung hat den Zweck, Daten für diejenigen unlesbar zu machen, die den passenden Schlüssel nicht kennen. Was aber ist, wenn Alice den Schlüssel verliert, mit dem sie ihre Daten verschlüsselt hat? Dann wird die Kryptografie zur Waffe, mit der sie sich selbst geschlagen hat. Es gibt jedoch eine Möglichkeit vorzubeugen, indem Alice eine Kopie ihres Schlüssels anfertigt und diese sicher verwahrt (Key Recovery). Im Falle eines Schlüsselverlusts nimmt sie die Kopie als Ersatz.

Es ist wohl unnötig zu erwähnen, dass Key Recovery ein zweischneidiges Schwert ist, denn eine Schlüsselkopie ist immer ein mögliches Einfallstor für Mallory. Andererseits ist Key Recovery (oder eine brauchbare Alternative dazu) in vielen Anwendungsszenarien ein absolutes Muss. Stellen Sie sich einmal vor, ein Unternehmen verliert die Codeentwicklung mehrerer Personenjahre, nur weil ein Mitarbeiter seine Smartcard verloren hat. Allerdings ist die Frage, wann und wie

Key Recovery einen Sinn hat, nicht ganz einfach zu beantworten und hängt zudem eng mit dem Thema Public-Key-Infrastrukturen zusammen. Deshalb werde ich Key Recovery in einem anderen Zusammenhang in Abschnitt 28.2 ausführlich vorstellen.

Technisch gesehen ist Key Recovery im Übrigen meist kein großes Problem. Mithilfe eines Key-Wrapping-Verfahrens kann Alice jederzeit eine Kopie ihres Schlüssels verschlüsselt abspeichern, sofern der Schlüssel für sie zugänglich ist. HSMs bieten in der Regel ebenfalls eine Key-Wrapping- oder sogar eine Klon-Funktion an. Etwas schwieriger wird es dagegen bei Smartcards, die ja per Design den Schlüssel nicht preisgeben sollen. Es ist jedoch möglich, einen Schlüssel außerhalb der Smartcard zu generieren und ihn vor dem Aufspielen auf den Chip zu kopieren.

Das eigentliche Problem liegt dagegen im Key-Recovery-Prozess. Der Betreiber eines Krypto-Systems muss sich genau darüber im Klaren sein, ob und unter welchen Umständen er Key Recovery zulässt. Es bietet sich an, für den Zugriff auf Schlüsselkopien ein Vier-Augen-Prinzip vorzusehen. Trotz aller Sicherheitsvorkehrungen sollten die Beteiligten ab und zu einen Key-Recovery-Vorgang testweise durchspielen. Ansonsten besteht die Gefahr, dass die Sache nicht funktioniert, wenn einmal der Ernstfall eintritt.

24.2 Trusted Computing und Kryptografie

Alice kann mit den Daten auf ihrem eigenen PC tun, was Sie möchte. Sie kann nach Belieben Software installieren, Daten kopieren, Daten verändern, Daten löschen und Daten verschicken. Normalerweise ist das auch gut so – schließlich ist es ja Alices Computer, auf dem sie tun und lassen darf, was sie will. Allerdings hat es auch eindeutig seine Nachteile, wenn ein PC-Anwender völlig frei über seine Daten verfügen kann. Dies zeigt sich im Wesentlichen an folgenden zwei Problemen:

- **Sicherheitsproblem**: Alle Möglichkeiten, die Alice auf ihrem PC hat, hat auch ein Hacker, Trojaner, Computerwurm oder Computervirus, der in das Betriebssystem ihres PCs eindringt (der Einfachheit halber wollen wir im Folgenden nur den Computervirus betrachten). Dies bedeutet beispielsweise, dass ein Virus alle Daten auf Alices Festplatte löschen kann oder ein auf Alices Festplatte gespeichertes Passwort per E-Mail an eine bestimmte Adresse schicken kann.
- **Rechteproblem**: Es gibt Fälle, in denen jemand Alice bestimmte Daten zur Verfügung stellt und nicht möchte, dass Alice diese kopiert oder verschickt oder für längere Zeit speichert. Ein Beispiel: Bob möchte möglicherweise nicht, dass Alice eine vom ihm geschickte Mail an andere Personen weiterleitet. Ein weiteres Beispiel: Ein Anbieter von Musik-Downloads will möglicher-

weise verhindern, dass Alice eine heruntergeladene MP3-Datei kopiert und an Bob weitergibt. Man spricht in diesem Zusammenhang auch von Digital Rights Management. Um dieses Thema geht es in Abschnitt 38.7.

Will man diese beiden Probleme in den Griff bekommen, dann bleibt nichts anderes übrig, als den bisher fast unbeschränkten Datenzugriff des Anwenders auf seinem PC zu beschränken. Es muss also einen Teilbereich des Computers geben, in dem Alice nicht nach Belieben Daten löschen, kopieren und speichern kann. Wenn es um das Sicherheitsproblem geht, genügt es, wenn dieser Teilbereich für übliche Softwareprogramme nicht zugänglich ist. Beim Rechteproblem darf Alice dagegen überhaupt keine Möglichkeit haben, den besagten nichtmanipulierbaren Teilbereich auszuhebeln.

In der Praxis bedeutet die Einführung eines geschützten Computerteilbereichs Folgendes: Der Hersteller baut in einen Computer ein Hardwaremodul ein, das verschiedene Aufgaben innerhalb des Betriebssystems übernimmt. Selbst Alice als Besitzerin des Computers darf nicht die Möglichkeit haben, den Inhalt des Moduls oder den Programmablauf darin zu manipulieren. Ist dies der Fall, dann hat auch ein Virus keine Chance, das Modul zu beeinträchtigen. Auch das Rechteproblem lässt sich mit einem solchen Modul lösen, da Alice Daten darin nicht nach Belieben kopieren oder verändern kann.

Kommt ein auf diese Weise vom Anwender geschütztes Hardwaremodule (speziell das weiter unten vorgestellte TPM) zum Einsatz, dann bezeichnet man dies als **Trusted Computing**. »Trusted« heißt bekanntlich »vertrauenswürdig«, was in diesem Fall einen doppelten Sinn hat. Zum einen kann Anwenderin Alice darauf vertrauen, dass ihr PC bei einem Virenbefall keine gefährlichen Aktionen zulässt. Zum anderen kann ein externer Beteiligter darauf vertrauen, dass Alices PC bestimmte Aktivitäten unterlässt, die nicht in seinem Sinne sind.

Schon diese kurze Einführung dürfte eines deutlich machen: Das Thema Trusted Computing birgt einigen Zündstoff. Es erfordert Eingriffe in die Privatsphäre des einzelnen Anwenders und nimmt ihm Rechte, die bis dahin selbstverständlich waren. Nicht umsonst ist in den letzten Jahren eine heftige Diskussion über den Sinn und Unsinn des Trusted Computing in Gang gekommen, an der sich auch Datenschützer und Bürgerrechtler beteiligen. An dieser Stelle soll uns jedoch nicht die gesellschaftliche Komponente des Trusted Computing, sondern nur die Technik interessieren.

24.2.1 Trusted Computing und Kryptografie

Trusted Computing hat zwar nicht notwendigerweise etwas mit Kryptografie zu tun, doch die Berührungspunkte sind offensichtlich. Man kann es so formulieren: Trusted Computing ist ein Hilfsmittel der Kryptografie, und die Kryptografie ist ein Hilfsmittel des Trusted Computing. Ersteres zeigt sich daran, dass ein für das Trusted Computing vorgesehenes Hardwaremodul Schlüssel speichern und kryp-

tografische Operationen durchführen kann. Das Modul übernimmt dadurch die Rolle, die sonst eine Smartcard oder ein HSM spielt. Malware-Attacken sind für Mallory schwierig bis unmöglich, da eine Malware nicht auf die Schlüssel zugreifen kann.

Dass umgekehrt die Kryptografie ein Hilfsmittel des Trusted Computing ist, ist ebenfalls nicht schwer zu erkennen. Die kryptografischen Operationen, die das Trusted-Computing-Modul durchführen kann, lassen sich hervorragend nutzen, um die eigentlichen Ziele des Trusted Computing (Lösung des Sicherheits- und des Rechteproblems) zu verfolgen. Hier ein Beispiel: Wenn Bob Alice eine Datei schickt (nehmen wir an, es handle sich um ein kleines Computerprogramm) und nicht möchte, dass Alice diese kopiert, dann könnte folgendes Protokoll zur Anwendung kommen:

1. Bob verschlüsselt die Datei mit dem öffentlichen Schlüssel, dessen Gegenstück auf Alices Trusted-Computing-Modul gespeichert ist.
2. Alice erhält die Datei. Da sie verschlüsselt ist, kann sie den Dateiinhalt nicht kopieren (genau das will Bob ja erreichen). Alice übergibt die Datei an das Trusted-Computing-Modul.
3. Das Trusted-Computing-Modul entschlüsselt die Datei mit dem darauf gespeicherten privaten Schlüssel.
4. Das Trusted-Computing-Modul führt das Programm aus.

Sofern das Trusted-Computing-Modul korrekt arbeitet, hat Alice keine Möglichkeit, die Datei zu kopieren. Natürlich ist dies in der Praxis meist nicht ganz so einfach, da ein halbwegs anspruchsvolles Programm nicht vollständig innerhalb des Trusted-Computing-Moduls ausgeführt werden kann. Darüber hinaus lässt sich das Kopieren einer Grafik- oder Musikdatei nie wirklich verhindern (Alice kann etwa einen Screenshot erstellen). Nähere Informationen zu diesem Thema gibt es im Abschnitt über Digital Rights Management (Abschnitt 38.7).

24.2.2 Das Trusted Platform Module

Die große Bedeutung des Trusted Computing zeigt sich nicht zuletzt an der Tatsache, dass sich über 100 Unternehmen zu einer weltweiten Organisation zusammengeschlossen haben, die sich dieses Themas annimmt. Deren Name lautet **Trusted Computing Group** (**TCG**). Praktisch alles, was in der IT-Welt Rang und Namen hat, ist in der TCG versammelt: Microsoft, IBM, Intel, Sun und viele andere. Das wichtigste Ziel der TCG ist die Entwicklung einer standardisierten Architektur für das Trusted Computing, die von möglichst vielen PCs, PDAs, Handys und sonstigen Geräten unterstützt werden soll.

Den Kern der von der TCG entwickelten Architektur bildet ein Hardwaremodul namens **TPM** (**Trusted Platform Module**). Das TPM ähnelt vom Aufbau und von der Leistungsfähigkeit her einem Smartcard-Chip. Im Unterschied

zu einem solchen ist das TPM jedoch dazu gedacht, als fester Bestandteil in die Hauptplatine eines PCs oder eines anderen Geräts eingebaut zu werden. Es darf nicht ohne weiteres möglich sein, das TPM vom zugehörigen Gerät zu trennen. Zudem muss das TPM mit den Methoden konzipiert und hergestellt werden, die sonst beim Bau sicherer Krypto-Chips zum Einsatz kommen.

Die Funktionalität des TPM besteht fast ausschließlich aus kryptografischen Funktionen. Das Gerät, in das das TPM eingebaut ist, kann diese Funktionen sowohl zu Zwecken des Trusted Computing als auch für beliebige andere kryptografische Zwecke nutzen. Wie diese Nutzung aussehen kann, ist Gegenstand einer Reihe weiterer Spezifikationen der TCG. Darüber hinaus haben Hardware- und Betriebssystemhersteller große Freiheiten bei der Entwicklung eigener TPM-Anwendungen. Die aktuelle Version des TPM ist 1.2. Die nächste Version soll die Nummer 2.0 haben und einige Neuerungen mit sich bringen.

Bestandteile des TPM

Ein TPM-Chip enthält folgende für uns relevante Komponenten:

- *SHA-1-Engine*: Dies ist eine Komponente, die zu einer beliebigen Eingabe den SHA-1-Hashwert berechnet.
- *HMAC-Engine*: Diese Komponente berechnet zu einer beliebigen Eingabe und einem zusätzlich einzugebenden Schlüssel den SHA-1-HMAC-Hashwert.
- *RNG*: Dies ist ein Zufallsgenerator. Es handelt sich dabei um einen echten Zufallsgenerator, der auf der Messung physikalischer Größen beruht.
- *Kryptografischer Ko-Prozessor*: Dies ist ein Prozessor, der RSA-Berechnungen durchführt. Zudem unterstützt der Ko-Prozessor symmetrische Verschlüsselung, indem der Klartext mit dem Schlüssel exklusiv-oder-verknüpft wird (One-Time-Pad), was nur für kurze Klartexte (etwa Schlüssel) geeignet ist. Diese Funktion ist jedoch nur innerhalb des Chips verfügbar.
- *Speicher*: Das TPM verfügt über volatilen und nichtvolatilen Speicher.
- *PCR* (Platform Configuration Register): Hierbei handelt es sich um Register, die zum Speichern von SHA-1-Hashwerten verwendet werden.

Beachten Sie, dass die TPM-Architektur (abgesehen vom nur intern verfügbaren One-Time-Pad) keine symmetrische Verschlüsselung vorsieht. Der Hersteller kann jedoch eine solche Funktion zusätzlich in den Chip integrieren. Darüber hinaus ist es möglich, eine TPM-Implementierung mit zusätzlichen asymmetrischen Verfahren (z. B. ECC-Verfahren) oder sonstigen kryptografischen Funktionen auszustatten.

Schlüssel

Auf einem TPM können (wie auf einer Smartcard) symmetrische und asymmetrische Schlüssel für beliebige Anwendungen gespeichert werden. Darüber hinaus sieht die TCG-Spezifikation drei spezielle Schlüsseltypen vor, die das TPM für bestimmte Aufgaben benötigt. Bei allen drei Typen handelt es sich um RSA-Schlüssel (genauer gesagt geht es immer um RSA-Schlüsselpaare):

- **Endorsement Key**: Jedes TPM-Exemplar hat ein RSA-Schlüsselpaar gespeichert, das als Endorsement Key (Bestätigungsschlüssel) bezeichnet wird. Der Endorsement Key wird im Chip erzeugt oder bereits vom Hersteller auf den Chip gebracht und ist nicht auslesbar. Seine Länge beträgt 2.048 Bit. Der Endorsement Key ist für jeden Chip anders (dies ist zumindest mit großer Wahrscheinlichkeit zu erwarten) und identifiziert einen solchen eindeutig. Auch der öffentliche Teil des Endorsement Key ist (aus Datenschutzgründen) nicht öffentlich zugänglich.
- **Attestation Identity Key**: Auch hierbei handelt es sich um ein RSA-Schlüsselpaar. Es wird ausschließlich für digitale Signaturen verwendet. Seine Nutzung erfordert die Eingabe eines 160-Bit-Passworts. Es kann mehrere Attestation Identity Keys auf einem TPM-Chip geben. Sie werden jeweils vom Endorsement Key abgeleitet. Die Schlüssellänge beträgt 2.048 Bit. Attestation Identity Keys dienen ausschließlich dem Signieren TPM-interner Daten wie PCRs und Schlüsseln.
- **Storage Root Key**: Einen solchen generiert der Besitzer eines Computers selbst. Es gibt nur einen pro TPM-Chip, bei einem Besitzerwechsel ist jedoch eine Löschung mit anschließender Neugenerierung möglich. Der Storage Root Key wird nur zum Verschlüsseln verwendet.

Da ein TPM dem Nutzer laut Spezifikation kein symmetrisches Verschlüsselungsverfahren anbieten muss, gibt es auch keine vorgegebenen symmetrischen Schlüssel.

24.2.3 Funktionen und Anwendungen des TPM

Wozu das TPM die beschriebenen Schlüssel benötigt, wird klar, wenn wir uns die typischen Anwendungsszenarien ansehen.

Verschlüsselung und digitale Signatur mit dem TPM

Das TPM lässt sich ähnlich nutzen wie eine Smartcard – allerdings mit dem Unterschied, dass das TPM fest an den jeweiligen Computer gebunden ist. Das TPM kann zum Beispiel einen privaten RSA-Schlüssel speichern, mit dem Alice die von Bob kommenden E-Mails entschlüsselt (Bob muss natürlich den dazu passenden öffentlichen RSA-Schlüssel kennen). Wenn Alice eine solche Mail erhält, leitet sie (bzw. ihre Mail-Software) den verschlüsselten symmetrischen

Schlüssel an das TPM weiter und erhält den Schlüssel im Klartext zurück. Anschließend kann sie die Mail selbst entschlüsseln. Auf ähnliche Weise kann Alice auch beliebige Daten mit einem privaten RSA-Schlüssel auf dem TPM signieren. Falls das TPM außer RSA noch andere asymmetrische Verfahren unterstützt, kann sie auch diese nutzen. Beachten Sie, dass Alice für solche Signier- und Entschlüsselungsaktivitäten keinen der drei genannten Schlüsseltypen verwendet. Eine weitere Funktion, die das TPM bietet, ist ein Hardwarezufallsgenerator.

Da der Speicherplatz auf dem TPM begrenzt ist, kann Alice nicht beliebig viele Schlüssel darauf speichern. Das TPM unterstützt jedoch das sichere Auslagern von Schlüsseln und anderen TPM-Inhalten. Dazu wird der bereits erwähnte Storage Root Key verwendet. Wenn es eine Bit-Folge auszulagern gilt, dann generiert das TPM zunächst einen symmetrischen Schlüssel und verwendet diesen, um die Bit-Folge zu verschlüsseln. Anschließend wird der symmetrische Schlüssel mit dem Storage Root Key verschlüsselt (es handelt sich also um ein Hybridverfahren). Die verschlüsselte Bit-Folge sowie der verschlüsselte Schlüssel werden außerhalb des TPM im ungeschützten Speicher des Computers abgelegt.

Das TPM und das Sicherheitsproblem

Eine Standardanwendung des TPM ist der Virenschutz durch sicheres Hochfahren eines Computersystems. Hierbei generiert der nach dem Starten ausgeführte Bootcode mithilfe des TPM einen Hashwert der wichtigsten Computergrundeinstellungen und speichert diesen in einem PCR. Anschließend startet der Betriebssystemlader und generiert einen Hashwert der für ihn wichtigen Daten und speichert ihn in einem weiteren PCR. Ähnlich verfahren anschließend das Betriebssystem selbst und die verschiedenen Betriebssystemkomponenten. Wenn das Betriebssystem schließlich läuft, ist eine Reihe von Hashwerten in den PCRs gespeichert, die sich nicht mehr ändern lassen – auch nicht von einem Virus.

Mithilfe dieser Hashwerte lässt sich jederzeit feststellen, ob sich an den Werten, die als Urbild in die Hashfunktion eingegangen sind, etwas geändert hat. Ist dies der Fall, dann bedeutet dies, dass ein Hacker, ein Virus oder auch ein Defekt Konfigurationseinstellungen oder ausführbaren Code verändert hat. Diese veränderte Stelle kann man anschließend isolieren oder mithilfe archivierter Daten in den ursprünglichen Zustand ersetzen.

Damit eine sichere Überprüfung der PCR-Inhalte möglich ist, kann das TPM diese mithilfe eines Attestation Identity Key signieren. Dies ist technisch gesehen kein größeres Problem. Schon eher problematisch ist dagegen die Frage, woher der Empfänger den öffentlichen Schlüssel kennt und weiß, dass es tatsächlich derjenige ist, der zum TPM gehört. Dieses Problem kann beispielsweise mithilfe einer Public-Key-Infrastruktur gelöst werden (siehe Teil 4 dieses Buchs), indem eine CA ein digitales Zertifikat für einen Attestation Identity Key ausstellt.

Das TPM und das Rechteproblem

Prinzipiell lässt sich das TPM auch zur Lösung des Rechteproblems (also für Digital Rights Management) nutzen. In der TPM-Dokumentation ist von einer solchen Anwendung jedoch nicht die Rede. Die Trusted Computing Group hat bisher bewusst darauf verzichtet, sich mit diesem umstrittenen Thema die Finger zu verbrennen. Offiziell gilt die Prämisse: Das TPM hat mit Digital Rights Management nichts zu tun.

24.2.4 Fazit

Inzwischen dürften einige Hundert Millionen TPMs produziert worden sein. Alle größeren Hersteller von PCs und Laptops bieten Geräte mit eingebauten TPM an. Auch einige Server gibt es in TPM-Auführung. Zahlreiche Produkte, etwa die Microsoft-Festplattenverschlüsselung BitLocker, sind in der Lage, ein TPM zu nutzen. Die ganz große Trusted-Computing-Euphorie ist bisher jedoch noch nicht ausgebrochen.

24.3 Krypto-APIs

Da in der Kryptografie unterschiedliche Programme häufig dieselben Verfahren einsetzen, bietet es sich an, die Schnittstelle zum Aufruf kryptografischer Funktionen zu vereinheitlichen. Mit anderen Worten: Ein standardisiertes **kryptografisches API** (API steht für Application Programming Interface) ist eine sinnvolle Sache. Man bezeichnet ein solches auch als **Krypto-API**. Weil die Notwendigkeit eines Krypto-API seit langem bekannt ist, können sich Softwareentwickler inzwischen aus einer ganzen Reihe davon bedienen. Die wichtigsten davon werden wir im Folgenden betrachten.

24.3.1 PKCS#11

Die wichtigste kryptografische API ist in PKCS#11 standardisiert und hat den Namen **Cryptoki** (**Cryptographical Token Interface**) [PKCS#11]. PKCS#11 ist ein weiterer Standard aus der PKCS-Reihe. Wie der Name andeutet, ermöglicht Cryptoki den Zugriff auf ein Objekt, das kryptografisches Token genannt wird. Ein kryptografisches Token ist in der Praxis häufig eine Smartcard, es kann sich jedoch auch um ein anderes Hardwaremodul (z.B. ein USB-Token oder ein HSM) oder auch um eine Softwarekomponente handeln.

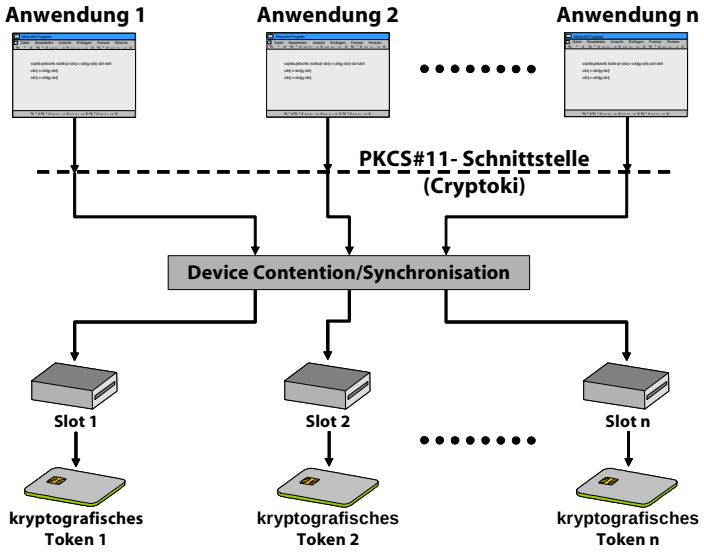


Abb. 24–1 *Über die PKCS#11-Schnittstelle können beliebig viele Anwendungen auf beliebig viele Smartcards (Token) zugreifen.*

Aufbau

PKCS#11 liegt ein Modell zugrunde, das folgende Komponenten vorsieht (siehe Abbildung 24–1):

- *Anwendung*: Dies ist typischerweise eine Software, die auf einem PC läuft.
- *PKCS#11-Schnittstelle* (Cryptoki)
- *Device Contention/Synchronisation*: Diese Komponente steuert die Kommunikation zwischen den Anwendungen sowie den Slots und den kryptografischen Token.
- *Slot*: Ein Slot entspricht einem Kartenleser.
- *Kryptografisches Token*: Dabei kann es sich beispielsweise um eine Smartcard handeln.

Jede Komponente kann auch mehrfach vorkommen. PKCS#11 verfolgt einen objektorientierten Ansatz. In jedem kryptografischen Token können drei Arten von Objekten vorhanden sein:

- *Daten* (data): Ein solches Objekt enthält nicht näher benannte Daten.
- *Schlüssel* (key): In einen solchen Objekt wird ein Schlüssel gespeichert. Dabei werden die drei Typen öffentlicher Schlüssel (public key), privater Schlüssel (private key) und geheimer Schlüssel (secret key) unterschieden.
- *Digitales Zertifikat* (certificate): Was ein digitales Zertifikat ist, wird in Abschnitt 26.1 erklärt.

Jedes Objekt hat Attribute, die beispielsweise die Zugriffsberechtigung oder kryptografische Parameter wie die Schlüssellänge festlegen.

Rollenmodell

PKCS#11 unterstützt ein einfaches Rollenkonzept, das zwei Rollen vorsieht: den Anwender und den Administrator (dieser wird in PKCS#11 *Security Officer* genannt). Beide Rollen gibt es nur in einer Ausprägung pro Token. PKCS#11 kann also beispielsweise nicht zwei Anwender desselben Tokens unterscheiden. Für beide Rollen gibt es je ein Passwort, das in diesem Zusammenhang PIN (Personal Identification Number) genannt wird. Die PIN ist eine beliebige Zeichenkette, die neben Zahlen auch Buchstaben und Sonderzeichen enthalten kann. Die PIN des Administrators heißt SO-PIN (Security Officer PIN), die PIN des Anwenders wird User-PIN genannt. Der Administrator ist für die Initialisierung des Tokens zuständig. Er kann außerdem die User-PIN ändern.

Prozesse

Den Zweck von PKCS#11 kann man auf Basis der beschriebenen Komponenten und Rollen wie folgt beschreiben: Der Anwender nutzt eine oder mehrere Anwendungen, um über die PKCS#11-Schnittstelle per Funktionsaufruf kryptografische Operationen zu starten. Diese Operationen finden innerhalb der Tokens statt und nutzen die dort vorhandenen Objekte. Das Ergebnis der jeweiligen Operation wird als Funktionsergebnis zurückgegeben. Neben den kryptografischen Funktionsaufrufen gibt es natürlich auch verschiedene verwaltungstechnische Operationen, die PKCS#11 unterstützt, wie beispielsweise das Eingeben der PIN oder das Initialisieren eines Tokens.

Betrachtet man nur die kryptografischen Funktionen, die PKCS#11 unterstützt, so kann man diese in folgende Bereiche einteilen: Verschlüsselung, Entschlüsselung, Hashwert-Generierung, Signieren und Generieren eines schlüsselabhängigen Hashwerts (MAC), Zweifach-Operationen, Schlüsselmanagement und Zufallszahlen-Generierung. Welches Verfahren für die jeweilige Operation verwendet wird, teilt die Applikation dem Token über Funktionsparameter mit. Der PKCS#11-Standard spezifiziert die Verwendung nahezu aller praxisrelevanten Verfahren:

- *Asymmetrische Verschlüsselung*: RSA in verschiedenen Varianten
- *Digitale Signatur*: RSA, DSA und ECDSA
- *Schlüsselaustausch*: Diffie-Hellman, ECDH, MQV und ECMQV
- *Symmetrische Verschlüsselung*: AES, DES, Triple-DES, RC2, RC4, RC5, IDEA, CAST, JUNIPER, BATON, Skipjack
- *Kryptografische Hashfunktionen*: MD2, MD5, SHA-1 und RIPEMD-160

Der PKCS#11-Standard macht jedoch keine Vorschriften, welche dieser Funktionen implementiert werden müssen und welche man weglassen kann. Der Entwickler einer Software muss also selbst entscheiden, welche Verfahren er sich herauspickt (in der Praxis wird dies schon allein aufgrund des beschränkten Smartcard-Speichers nur eine kleine Untermenge sein). Natürlich haben sich die Entwickler von PKCS#11 auch einige Gedanken über die Sicherheit einer Implementierung gemacht. So ist es möglich, einen Schlüssel unauslesbar in einem Token-Objekt abzulegen. Das Ausführen kryptografischer Operationen ist nur nach Eingabe der User-PIN möglich. Darüber hinaus kann ein Objekt nach Ende einer Session gelöscht werden.

Um kryptografische Tokens über PKCS#11 ansprechen zu können, benötigt man in der Regel einen Treiber, der die Tokens anspricht und die Schnittstelle der Anwendung zu Verfügung stellt. Viele Hersteller erweitern diesen Treiber um eine Benutzeroberfläche, über die der Administrator und der Anwender einige Verwaltungsoperationen (Ändern der PIN, Löschen der Tokenobjekte usw.) durchführen können. Eine solche Treiber-Software mit Benutzeroberfläche ist besonders im Zusammenhang mit Smartcards weit verbreitet und wird in dieser Ausprägung als **Smartcard-Middleware** bezeichnet.

Bewertung von PKCS#11

PKCS#11 ist eine äußerst wichtige API, das von vielen Softwareprodukten unterstützt wird (etwa von Mozilla Firefox und Linux). Man sollte sich jedoch darüber im Klaren sein, dass PKCS#11 nicht den Anspruch hat, alle wichtigen kryptografischen Aufgaben abzudecken. Insbesondere hat PKCS#11 folgende Einschränkungen:

- Es gibt nur einen Anwender und nur einen Administrator. Es ist nicht möglich, verschiedene Anwender mit unterschiedlichen Rechten über eine PKCS#11-Schnittstelle zu verwalten. Auch das Festlegen mehrerer Administratoren wird nicht unterstützt.
- PKCS#11 unterstützt nur kryptografische Funktionen auf niedriger Ebene. Dazu gehören Befehle zum Verschlüsseln, Signieren, Verifizieren und Ähnliches. Es gibt dagegen beispielsweise keinen Befehl zur Generierung einer Protokollnachricht oder zur Generierung eines digitalen Zertifikats.
- PKCS#11 ist nicht geeignet, den Kontext eines zustandsbehafteten kryptografischen Netzwerkprotokolls zu verwalten. Stattdessen muss eine Anwendung dies selbst tun und kann dabei lediglich die kryptografischen Operationen an das kryptografische Token auslagern.

Diese Einschränkungen fallen jedoch nicht ins Gewicht, wenn man PKCS#11 nur als API für den Zugriff auf Smartcards und ähnliche Krypto-Module nutzt. Ein deutlich größeres Problem sind dagegen die großen Freiheiten, die der Standard zulässt. Durch die Vielzahl der kryptografischen Operationen, die eine PKCS#11-

Implementierung unterstützen kann (aber nicht muss), ist längst nicht jedes PKCS#11-fähige Programm mit jedem PKCS#11-Modul interoperabel. Aus diesem Grund sind in der Zwischenzeit zahlreiche PKCS#11-Profile entstanden, also Standards, die den PKCS#11-Standard einengen. In jedem Fall empfiehlt sich im Zusammenhang mit PKCS#11 stets eines: Testen.

24.3.2 MS-CAPI

Die **Microsoft Cryptographic API** (**MS-CAPI**) ist eine von Microsoft entwickelte kryptografische API, die in den Windows-Betriebssystemen eingesetzt wird [Coleri]. Die MS-CAPI gilt als Microsoft-Gegenstück zu PKCS#11 und ermöglicht wie diese die Anbindung kryptografischer Module an ein Anwendungsprogramm. Ein kryptografisches Modul wird in diesem Zusammenhang als **Cryptographic Service Provider** (**CSP**) bezeichnet. Ein CSP kann vollständig als Software realisiert sein und dabei den Windows-Schlüsselspeicher nutzen, es kann sich jedoch auch um ein Programm handeln, das alle Aktivitäten an eine Smartcard oder eine andere Krypto-Hardware auslagert. Der Zugriff auf die Krypto-Funktionen erfolgt objektorientiert, wobei jeder Schlüssel als Objekt interpretiert wird.

Aufbau

Durch ihre Windows-Integration ist die MS-CAPI etwas anders aufgebaut als PKCS#11 (siehe Abbildung 24–2). Genau genommen sieht die MS-CAPI-Architektur nicht eine, sondern zwei Schnittstellen vor. Die eine davon ist die *CryptoAPI*. Diese nimmt Befehle von der Anwendung entgegen und gibt die Ergebnisse zurück. Die zweite Schnittstelle ist die *CryptoSPI* (SPI steht für System Programming Interface). Die CryptoSPI spricht die CSPs an. Zwischen der CryptoAPI und der CryptoSPI befindet sich eine Systemschicht, die vom Windows-Betriebssystem betrieben wird. Die Systemschicht hat die Aufgabe, die über die CryptoAPI eingehenden Befehle aufzunehmen und sie in geeigneter Form über die CryptoSPI an den jeweiligen CSP weiterzuleiten. Die Tatsache, dass es bei PKCS#11 keine Systemschicht gibt, zeigt die unterschiedlichen Philosophien, die hinter den beiden Standards stecken: PKCS#11 wurde hauptsächlich von Krypto-Herstellern entwickelt, die die Intelligenz lieber im Krypto-Modul selbst sehen. Die MS-CAPI stammt dagegen von Microsoft, das aus nachvollziehbaren Gründen eine starke Involvierung des Betriebssystems erreichen wollte.

Rollen

Das Rollenmodell der MS-CAPI ist an die Zugriffsrechte im Betriebssystem gekoppelt. Wenn ein CSP Schlüsselobjekte anlegt, dann sind diese nur für den jeweiligen Anwender zugänglich. Dabei ist es möglich, beliebig viele Schlüsselobjekte für unterschiedliche Anwender zu generieren.

Prozesse

Es gibt verschiedene Typen von CSPs. Jeder Typ bietet eine Reihe kryptografischer Funktionen an, die der Hersteller nach seinen Vorstellungen implementieren kann. Die Anwendung muss wissen, um welchen CSP-Typ es sich handelt, um durch einen Funktionsaufruf die entsprechende Funktion starten zu können. Da es im Allgemeinen mehrere CSPs eines Typs gibt, kann derselbe Funktionsaufruf unterschiedliche Folgen haben. Microsoft hat derzeit acht CSPs definiert. Jeder davon bietet Verschlüsselung, Signatur und Hashfunktion an, wobei Verfahren und Schlüssellängen variieren.

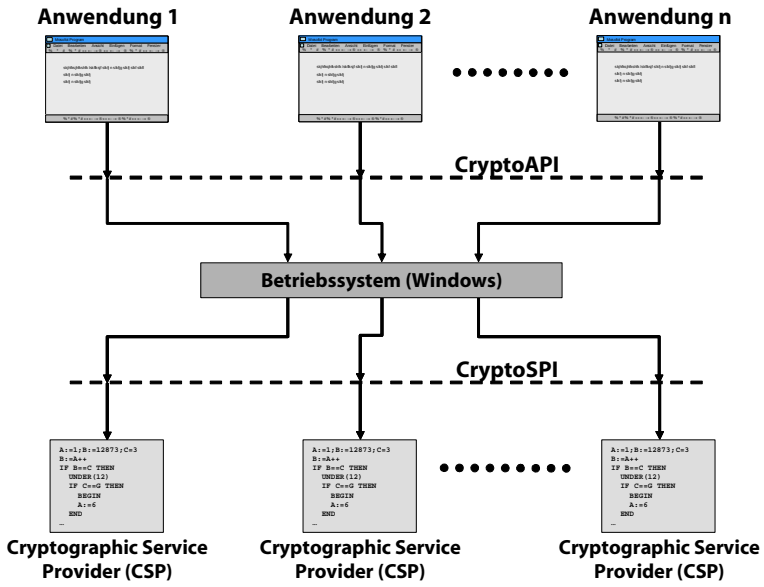


Abb. 24–2 *Die MS-CAPI ist in das Windows-Betriebssystem integriert.*

Bewertung der MS-CAPI

Da die CryptoAPI von Microsoft kommt und von einigen Produkten dieses Unternehmens unterstützt wird (etwa von Windows und dem Internet Explorer), ist diese Schnittstelle weit verbreitet. Außerhalb der Microsoft-Welt werden jedoch andere APIs verwendet. Ähnlich wie PKCS#11 ist die MS-CAPI nicht für die Generierung kryptografischer Protokollnachrichten und ähnliche komplexe Operationen geeignet. Vielmehr handelt es sich dabei um eine Schnittstelle, die lediglich kryptografische Basisoperationen wie das Verschlüsseln oder Signieren unterstützt.

24.3.3 Cryptography API Next Generation (CNG)

Die **Cryptography API Next Generation** (**CNG**) ist der Nachfolger der MS-CAPI [CNG]. Microsoft hat die CNG zusammen mit dem Betriebssystem Windows Vista eingeführt. Auch Windows Server 2008 unterstützt die neue Schnittstelle. Vermutlich wird uns die CNG auch in den kommenden Windows-Versionen begegnen und die MS-CAPI nach und nach verdrängen. Die CNG ist im Grundsatz ähnlich aufgebaut wie die MS-CAPI, hat jedoch in vielen Details ein verbessertes Design. Die beiden Krypto-APIs sind nicht miteinander kompatibel. Nach über zehn Jahren der Kontinuität müssen sich die Krypto-Hersteller damit auf eine komplett neue Schnittstelle für die Microsoft-Welt einstellen.

Die CNG unterstützt alle kryptografischen Funktionen, die bereits in der MS-CAPI vorhanden waren. Zusätzlich wurde die Unterstützung von ECC-Verfahren erweitert. Unter anderem lassen sich die in NSA Suite B (siehe Abschnitt 18.3.4) aufgeführten Verfahren mit der CNG aufrufen. Eine wichtige Neuerung ist zudem die geänderte Behandlung von Smartcards. Die MS-CAPI unterstützt eine Smartcard nur, wenn es einen speziell dafür entwickelten CSP gibt. Die CNG stellt dagegen einen generischen Smartcard-CSP (**Base CSP**) zur Verfügung, der nach unten über eine definierte Schnittstelle mit dem Treiber der Karte spricht (dieser Treiber wird als **Smart Card Minidriver** bezeichnet). Ein Smartcard-Hersteller muss daher für seine Karte keinen kompletten CSP mehr entwickeln. Stattdessen kann er einen (deutlich einfacheren) Smart Card Minidriver zur Verfügung stellen.

24.3.4 TokenD

TokenD (sprich: »tauken-di«) ist eine kryptografische API, die von der Firma Apple für das Betriebssystem Mac OS entwickelt wurde. TokenD hat einen ähnlichen Zweck wie die zuvor vorgestellten PKCS#11, MS-CAPI und CNG: Sie dient als Schnittstelle für eine Anwendung (z. B. Webbrowser oder E-Mail-Programm) beim Zugriff auf ein Krypto-Modul (z. B. Smartcard). Eine genaue Beschreibung von TokenD hat Apple bisher nicht veröffentlicht. Auch sonst haben die Themen Smartcards und Kryptografie bei Apple derzeit offensichtlich nicht die höchste Priorität. Dementsprechend ist die Anzahl der Programme, die TokenD unterstützen, bisher gering. Die Apple-Produkte iPad und iPhone, die mit dem Betriebssystem iOS arbeiten, boten bei Redaktionsschluss dieses Buchs noch gar keine TokenD-Unterstützung.

24.3.5 ISO/IEC 24727

Der Standard **ISO/IEC 24727** beschreibt eine weitere Krypto-API. Es handelt sich dabei um einen Standard, dessen vier Teile in den Jahren 2007 und 2008 erschienen sind [ISO24727-1, ISO24727-2, ISO24727-3, ISO24727-4]. Um sich eine Übersicht zu verschaffen, sollten Sie den ersten Teil lesen. ISO/IEC 24727

bietet die Möglichkeit, kryptografische Operationen auf einer Smartcard oder einem ähnlichen Hardwaremodul (z.B. USB-Token) aufzurufen. Der geplante Einsatzbereich sind vor allem elektronische Ausweise (siehe Abschnitt 38.6). Hierzu gehören insbesondere elektronische Personalausweise, wie sie derzeit in vielen Ländern eingeführt oder bereits genutzt werden. Allerdings lässt sich die Schnittstelle auch für andere Krypto-Module verwenden, die der Authentifizierung, Verschlüsselung oder digitalen Signierung dienen.

Im Mittelpunkt von ISO/IEC 24727 stehen zwei übereinanderliegende Schichten (siehe Abbildung 24–3). Die *Service Access Layer* spricht nach oben mit einem Anwendungsprogramm und nach unten mit der zweiten Schicht (*Generic Card Access Layer*). Letztere kommuniziert nach unten mit einer Anwendung auf einer Smartcard. Der Befehlssatz ist auf die Verwendung in Zusammenhang mit einem elektronischen Ausweis zugeschnitten. Im gängigsten Fall übernimmt eine Smartcard-Middleware die Umsetzung der beiden Schichten. Es ist jedoch auch möglich, die Generic Card Access Layer oder sogar beide Schichten auf der Smartcard zu implementieren.

Im Gegensatz zu PKCS#11, MS-CAPI und CNG, die die Unterscheidung mehrerer Module erlauben, ist ISO/IEC 24727 für das Ansprechen nur einer Smartcard gedacht. Die Verwaltung mehrerer Chips muss bei ISO/IEC 24727 daher auf höherer Ebene erfolgen. Da ISO/IEC 24727 noch recht neu ist, gibt es bisher nur wenige Implementierungen. Die Eignung für elektronische Ausweise könnte jedoch dafür sorgen, dass sich dieser Standard gegenüber PKCS#11 und CNG behaupten kann.

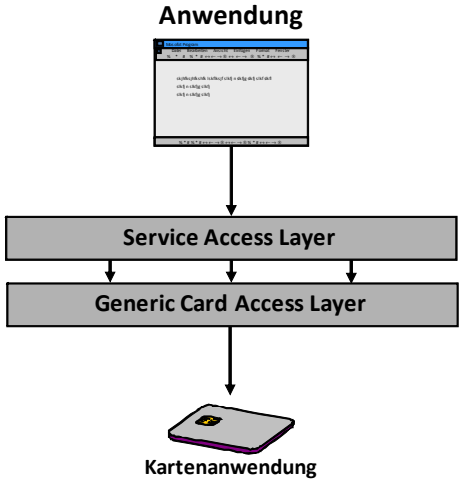


Abb. 24–3 *ISO/IEC 24727 ist eine Krypto-API, die vor allem für elektronische Ausweise geeignet ist.*

24.3.6 Universelle Krypto-APIs

Die bisher betrachteten Krypto-APIs PKCS#11, MS-CAPI, CNG und ISO/IEC 24727 sind für die Ansprache von Smartcards und Krypto-Modulen gedacht. Im Folgenden schauen wir uns einige Krypto-APIs an, die einen allgemeineren Ansatz verfolgen.

GSS-API und SSPI

Das **Generic Security Service Application Program Interface** (**GSS-API**) ist eine Krypto-API, die von der IETF standardisiert wurde. Ihre aktuelle Version 2.0 wird in [RFC2743] beschrieben, mehrere weitere RFCs spezifizieren Erweiterungen und spezielle Anwendungsbereiche. Unter dem Projektnamen **Kitten** entsteht derzeit die dritte Version der GSS-API. Im Gegensatz zu PKCS#11 und MS-CAPI setzt die GSS-API eine Stufe höher an. Sie dient nicht dazu, bestimmte Daten zu verschlüsseln oder zu signieren, sondern dazu, Protokollnachrichten für kryptografische Protokolle zu generieren. Unter anderem lassen sich SASL, die Secure Shell und Kerberos über die GSS-API nutzen.

Das **Security Service Provider Interface** (**SSPI**) ist eine von Microsoft entwickelte Krypto-API, die man als Adaptierung der GSS-API für Windows-Umgebungen bezeichnen kann. SSPI und GSS-API haben eine große Ähnlichkeit, sind jedoch nicht kompatibel.

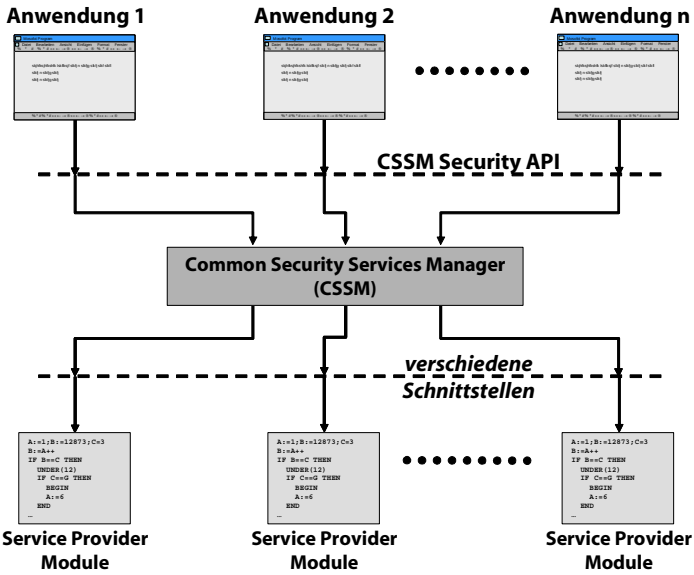


Abb. 24–4 *Die Common Data Security Architecture (CDSA) sieht eine leistungsfähige Krypto-API vor. Diese wird CSSM Security API genannt.*

CDSA

Die **Common Data Security Architecture** (**CDSA**) ist eine von Intel entwickelte Architektur, die kryptografische Funktionen über eine Krypto-API zur Verfügung stellt. Ziel ist es, dass von der Nutzung einer Smartcard bis zum Generieren einer Protokollnachricht in einem Krypto-Protokoll jede kryptografische Operation über CDSA abwickelbar ist. CDSA könnte daher theoretisch alle anderen hier behandelten Krypto-APIs ersetzen. Andererseits ist es jedoch auch möglich, andere Krypto-Schnittstellen über CDSA anzusprechen.

Ähnlich wie die MS-CAPI sieht CDSA unterhalb des API eine Zwischenschicht vor, die selbst CSPs anspricht. Diese Zwischenschicht wird als **Common Security Services Manager** (**CSSM**) bezeichnet. In der Theorie ist CDSA zweifellos die beste Krypto-API. Leider ist CDSA bei weitem nicht so verbreitet wie die meisten anderen an dieser Stelle beschriebenen Lösungen. Immerhin gibt es eine Open-Source-Implementierung von Intel. Weitere Informationen zu CDSA finden sich auf der Intel-Webseite.

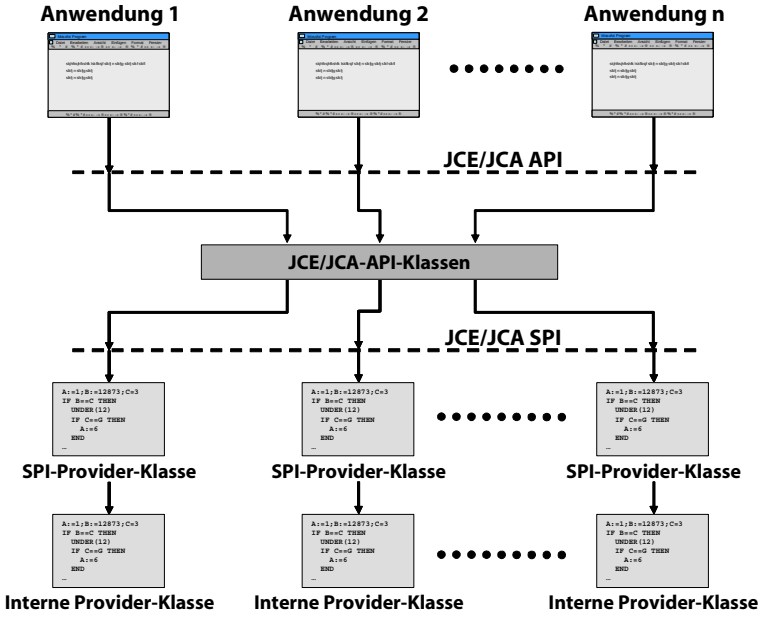


Abb. 24–5 *JCA/JCE ist eine Architektur, die eine Krypto-API auf Basis von Java vorsieht.*

Krypto-APIs in Java

Der *Java Development Kit* (JDK) von Sun, der inzwischen als Open Source verfügbar ist, ist die mit Abstand populärste Entwicklungsumgebung für die Programmiersprache Java. Der JDK unterstützt zahlreiche Krypto-Funktionen, die über ein Modell namens **JCA** (**Java Cryptography Architecture**) sowie dessen Erweiterung **JCE** (**Java Cryptography Extensions**) ansprechbar sind. JCA und JCE kann man inzwischen als eine Einheit betrachten, wobei JCA Authentifizierungs- und Signaturfunktionen unterstützt, während JCE den Aufruf von Verschlüsselungsverfahren und schlüsselabhängigen Hashfunktionen realisiert. Diese Trennung rührt daher, dass der Export von Verschlüsselungstechnik in den USA bis zum Jahr 2000 stark eingeschränkt war. Der JDK wurde deshalb ursprünglich nur ohne JCE exportiert. Seitdem die Exportbeschränkungen aufgehoben sind, gehört auch JCE zum Lieferumfang.

JCA/JCE sieht eine Architektur in mehreren Schichten vor (siehe Abbildung 24–5). Die Anwendung greift über eine Krypto-API auf eine Zwischenschicht zu, die wiederum selbst über eine SPI auf CSP-Klassen zugreift. Dabei wird zwischen den *SPI-Klassen* und den *internen Klassen* unterschieden. Weitere Informationen zu diesem Thema bietet beispielsweise [Hook].

Weitere Krypto-APIs

Die meisten Krypto-APIs, die nicht in diesem Kapitel beschrieben wurden, sind proprietär. Zahlreiche Krypto-Anbieter haben Krypto-Bibliotheken für unterschiedliche Plattformen im Angebot, die über eine nichtstandardisierte Schnittstelle nutzbar sind.

25 Evaluierung und Zertifizierung

Kann man Sicherheit messen? Die Antwort auf diese Frage lautet eindeutig nein. Egal, welche Krypto-Lösung Alice zur Kommunikation mit Bob einsetzt – sie weiß nie genau, wie gut die Chancen für Bösewicht Mallory stehen, einen erfolgreichen Angriff zu starten. Insbesondere kann sie folgende Sicherheitsfragen nie mit letzter Gewissheit beantworten:

- Sind die Verfahren und Protokolle, die Alice einsetzt, auch wirklich sicher?
- Ist die Krypto-Lösung, mit der Alice arbeitet, korrekt implementiert? Oder sorgt vielleicht ein Programmfehler dafür, dass Mallory ihre scheinbar sichere Verschlüsselung knacken kann?
- Ist der Hersteller von Alices Krypto-Lösung seriös? Oder hat er eine Hintertür in sein Produkt eingebaut, die Mallory die Arbeit erleichtert?
- Ist Alices Krypto-Lösung vor Diebstahl und vor dem Auslesen geheimer Schlüssel geschützt?

Eine zuverlässige Anwort auf diese Fragen gibt es nicht. Sicherheit ist nun einmal etwas anderes als die Rechengeschwindigkeit eines Prozessors oder die Größe einer Festplatte. Sicherheit ist nicht messbar.

Doch obwohl (oder gerade weil) es kein Messverfahren für die Sicherheit eines Computersystems gibt, haben Experten immer wieder nach Methoden gesucht, die einer objektiven Sicherheitseinstufung wenigstens nahe kommen. Vergleichsweise einfach ist dies, wenn es um kryptografische Verfahren oder Protokolle geht. Ein Verfahren oder Protokoll gilt dann als sicher, wenn es über mehrere Jahre hinweg von Krypto-Experten untersucht worden ist, ohne dass dabei Schwachstellen ans Licht gekommen sind. Je mehr Aufwand und Sachverstand in solche Untersuchungen geflossen sind, desto höher ist die vermutete Sicherheitsstufe.

Etwas schwieriger wird es, wenn es um die Implementierung kryptografischer Verfahren geht. Programmfehler, Hintertüren und unzureichende Schutzmaßnahmen fallen zwar in der Regel auch irgendwann auf – doch dann ist es meist schon zu spät. Will Alice die Sicherheit der von ihr verwendeten Krypto-Implementierung abschätzen, dann benötigt sie daher eine andere Methode. Die beste bekannte Vorgehensweise besteht darin, eine Implementierung nach einer standardisierten Liste von Kriterien zu untersuchen. Je nachdem, welche Kriterien erfüllt sind, erhält die Implementierung anschließend eine bestimmte Sicherheitsstufe zugeordnet.

Auf eine Untersuchung nach standardisierten Kriterien sollte sich Alice natürlich nur verlassen, wenn eine staatliche Behörde (etwa das BSI) oder ein anerkanntermaßen unabhängiges Unternehmen (etwa der TÜV) diese durchgeführt hat. Die Überprüfung eines Produkts nach festgelegten Kriterien durch eine derartige Institution wird **Evaluierung** genannt (sie wird von einem **Evaluator** durchgeführt). Hat ein Hersteller seine Krypto-Implementierung erfolgreich evaluieren lassen, dann bekommt er dies in Form einer Urkunde (einem Zertifikat) dokumentiert. Dies wird als **Zertifizierung** bezeichnet. Um Missverständnisse zu vermeiden: Diese Zertifizierung hat nichts mit digitalen Zertifikaten (Abschnitt 26.1) zu tun.

Im Zusammenhang mit der Kryptografie spielen derzeit drei Evaluierungsstandards eine Rolle: das inzwischen veraltete ITSEC, dessen Nachfolger Common Criteria sowie der US-Standard FIPS 140. Die beiden erstgenannten Standards beziehen sich auf das Thema IT-Sicherheit allgemein, während FIPS 140 ausschließlich der Evaluierung von Krypto-Implementierungen dient. Weitere Evaluierungsstandards, die ich nicht näher behandeln werde, sind folgende:

- **TCSEC** (Trusted Computer Security Evaluation Criteria): Diese einst in den USA bedeutsamen Kriterien sind auch als **Orange Book** bekannt, da sie in einem dicken Wälzer mit orangefarbenem Einband in Umlauf gebracht wurden. Das Orange Book erlaubt die Einordnung vollständiger Computersysteme (vor allem Betriebssysteme sind damit gemeint) in die Sicherheitsklassen A1, B1, B2, B3, C1, C2 und D (D ist die niedrigste). Inzwischen ist TCSEC von den Common Criteria abgelöst worden.

- **BSI-Grundschutzhandbuch**: Dieses sieht nicht die Evaluierung einzelner Produkte, sondern ganzer Organisationen vor. Vereinfacht ausgedrückt ist das BSI-Grundschutzhandbuch das Sicherheitsgegenstück zu ISO 9000. Kryptografie spielt im BSI-Grundschutzhandbuch zwar auch eine Rolle, jedoch nur als eines von vielen IT-Sicherheitswerkzeugen.
- **BS 7799** (**ISO 17799**): Dieser Standard ist das internationale Gegenstück zum BSI-Grundschutzhandbuch.
- **Webtrust**: Dieser Standard dient der Sicherheitsevaluierung von Webangeboten. Auch hier ist die Kryptografie nur eines von vielen Werkzeugen.

25.1 ITSEC

Die **Information Technology Security Evaluation Criteria** (**ITSEC**) wurden 1991 gemeinsam von Deutschland, Großbritannien, Frankreich und den Niederlanden eingeführt (eine Übersicht gibt die Webseite des BSI, weitere Infos bietet [Verste]). Zweck des ITSEC-Standards ist es, ein Maß für die Sicherheit einer IT-Komponente (diese wird **Evaluationsgegenstand** genannt) zu geben. Der Evaluationsgegenstand kann etwa ein Hardwaremodul oder ein Softwareprogramm sein. Es muss sich dabei nicht unbedingt um eine Kryptografie-Lösung handeln, vielmehr lässt sich auch eine Firewall, ein Betriebssystem oder eine Datenbank ITSEC-evaluieren.

Genau genommen definiert ITSEC zwei unterschiedliche Maße für die Sicherheit des Evaluationsgegenstands: Das eine Maß bewertet die Korrektheit der Implementierung, das andere die Wirksamkeit im Bezug auf die angestrebten Sicherheitsziele.

Über die Korrektheit und die Wirksamkeit eines Evaluationsgegenstands lässt sich natürlich erst dann eine Aussage machen, wenn geklärt ist, welchen Zweck dieser erfüllen soll. Will die Firma Krypt & Co. ein Produkt ITSEC-evaluieren lassen, dann muss sie daher zunächst die Funktionalität, die Sicherheitsmechanismen und die Sicherheitsziele des Evaluationsgegenstands definieren. Der ITSEC-Standard nennt dazu insgesamt zehn vordefinierte Möglichkeiten (Funktionalitätsklassen), die teilweise an den erwähnten TCSEC-Standard (Orange Book) angelehnt sind. Diese vorgegebenen Funktionalitätsklassen sind allerdings nur als Vorschläge zu verstehen und daher nicht bindend.

Korrektheit

Bekanntlich können Programmfehler und absichtlich eingebaute Hintertüren ein Problem für Krypto-Anwenderin Alice sein. Aus diesem Grund sieht der ITSEC-Standard eine Methode vor, mit der sich die Fehlerfreiheit eines Evaluationsgegenstands bewerten lässt. Dabei baut ITSEC auf folgende Erfahrungsregel: Je umfangreicher der Evaluationsgegenstand geprüft und getestet wurde, desto höher ist die Wahrscheinlichkeit, dass er keinen Fehler enthält. Dieser Grundsatz gilt natürlich

nicht nur in der IT-Sicherheit, sondern auch in jedem anderen Teilgebiet der Informationstechnik. Die ITSEC zugrunde liegenden Methoden zur Fehlerüberprüfung gibt es daher in ähnlicher Form auch anderswo. Entstanden sind diese Techniken vor allem im Zusammenhang mit Atomwaffen und in der Raumfahrt, wo Programmfehler eine besonders verheerende Wirkung haben können. Für die Korrektheit eines Evaluationsgegenstands gibt es gemäß ITSEC folgende Stufen:

- **E0**: Diese Stufe hat jedes Produkt automatisch.
- **E1**: Diese Stufe erreicht ein Evaluationsgegenstand, wenn seine Sicherheitsziele definiert sind und es eine informelle Beschreibung der Systemarchitektur gibt.
- **E2**: Diese Stufe setzt eine informelle, aber detaillierte Spezifikation der Systemarchitektur und der durchgeführten Tests voraus. Der Evaluationsgegenstand muss zudem eine Trennung zwischen Sicherheitskomponenten und anderen Bestandteilen vorsehen.
- **E3**: Für diese Stufe ist zusätzlich ein Test des Quellcodes notwendig. Zudem muss Hersteller Krypt & Co. geeignete Verteilungs-, Wartungs- und Konfigurationsverfahren festlegen.
- **E4**: Für diese Stufe muss Hersteller Krypt & Co. ein formales Modell für die Sicherheitsrichtlinien des Evaluationsgegenstands und eine strukturierte Beschreibung der Architektur vorlegen. Zudem muss der Hersteller ausführliche Tests über die Verwundbarkeit des zertifizierten Produkts durchführen und dokumentieren.
- **E5**: In dieser Stufe müssen ein detaillierter Entwurf und der Quellcode übereinstimmen. Tests über die Verwundbarkeit des Produkts müssen auf Quellcode-Ebene durchgeführt werden.
- **E6**: Für diese Höchststufe muss eine formale Beschreibung der Systemarchitektur vorliegen.

Wirksamkeit

Neben der Korrektheit bewertet ITSEC auch die Wirksamkeit des Evaluationsgegenstands. Hier gibt es drei Stufen:

- **Niedrig**: Diese Stufe erreicht der Evaluationsgegenstand, wenn erkennbar ist, dass dieser einen Schutz gegen zufällige, unbeabsichtigte Sicherheitsverletzungen bietet.
- **Mittel**: Diese Stufe ist erreicht, wenn erkennbar ist, dass der Evaluationsgegenstand Schutz gegen Angreifer mit beschränkten Gelegenheiten oder Betriebsmitteln bietet.
- **Hoch**: Diese Stufe ist erreicht, wenn erkennbar ist, dass der Evaluationsgegenstand nur von Angreifern überwunden werden kann, die über sehr gute Fachkenntnisse, Gelegenheiten und Betriebsmittel verfügen, wobei ein solcher erfolgreicher Angriff als normalerweise nicht durchführbar beurteilt wird.

Fazit

Erhält das Produkt der Firma Krypt & Co. beispielsweise ein E2/hoch-Zertifikat, so handelt es sich um eine Lösung mit hohen Sicherheitsvorkehrungen, die jedoch vergleichsweise oberflächlich auf Implementierungsfehler getestet wurde. »E5/niedrig« wäre im Gegensatz dazu mit großer Wahrscheinlichkeit fehlerfrei implementiert, würde jedoch nur moderate Mittel zur Abwehr von Angriffen zur Verfügung stellen.

Die diversen Kombinationsmöglichkeiten kommen in der Praxis unterschiedlich oft vor. Die Evaluationsstufe E1 trifft man nur selten an, die Stufen E5 und E6 existieren im kommerziellen Bereich praktisch nicht (über den militärischen Bereich, wo ITSEC ebenfalls eine Rolle spielt, ist öffentlich nichts bekannt). Auch die Stufe niedrig wird selten gewählt. Sie ist nebenbei auch unglücklich benannt, denn wer will schon ein Produkt mit dem offiziellen Prädikat »niedrig« haben?

ITSEC hat vor allem bei Behörden in Europa eine gewisse Bedeutung erlangt. In der freien Wirtschaft hielt sich das Interesse an ITSEC-evaluierten Produkten dagegen lange Zeit in Grenzen. Ende der neunziger Jahre sorgten dann gesetzliche Bestimmungen (vor allem das deutsche Signaturgesetz) dafür, dass sich eine ITSEC-Evaluierung für viele Hersteller lohnte. Vereinzelt ließen Anbieter ihre Produkte auch unabhängig von Gesetzen aus reinen Marketinggründen ITSEC-evaluieren. Nach dem zwischenzeitlichen Aufschwung ist die Zahl der ITSEC-Evaluierungen in den letzten Jahren wieder stark zurückgegangen, da es mit den Common Criteria einen anerkannten Nachfolger gibt. Heute wird ITSEC fast nur noch auf neue Versionen bereits evaluierter Produkte angewandt.

25.2 Common Criteria

Die **Common Criteria** (**CC**) sind ein Evaluierungsstandard für IT-Sicherheitslösungen (nicht nur für Kryptografie), auf den sich die USA und Kanada auf der einen Seite sowie Deutschland, die Niederlande, Großbritannien und Frankreich auf der anderen geeinigt haben (auch hierzu bietet die BSI-Webseite gute Einstiegsinformationen). Inzwischen erkennen auch zahlreiche andere Staaten dieses Kriterienwerk an. Die Common Criteria wurden als Nachfolger von ITSEC und TCSEC (Orange Book) geschaffen. Ein wichtiges Ziel des Standards ist es, weltweit einheitliche Evaluierungsvorgehensweisen zu schaffen, um zu verhindern, dass ein Hersteller sein Produkt für unterschiedliche Länder mehrfach evaluieren lassen muss. Dieses Vorhaben ist gelungen, und so haben die Common Criteria die beiden Vorgänger inzwischen größtenteils verdrängt. Inhaltlich hat der Common-Criteria-Standard mehr Ähnlichkeit mit ITSEC als mit TCSEC.

Wie ITSEC fordert auch CC zunächst eine genaue Festlegung der Funktionalität des Evaluationsgegenstands. Dazu gibt es wie bei ITSEC vordefinierte Funktionalitätsklassen (sogenannte Funktionsanforderungen). In der aktuellen Common-Criteria-Version 3.1 werden elf davon genannt:

- *FAU*: Sicherheitsprotokollierung
- *FCO*: Kommunikation
- *FCS*: Kryptografische Unterstützung
- *FDP*: Schutz der Benutzerdaten
- *FIA*: Identifikation und Authentifikation
- *FMT*: Sicherheitsmanagement
- *FPR*: Vertraulichkeit
- *FPT*: Schutz der Sicherheitsfunktionen
- *FRU*: Ressourcenverwendung
- *FTA*: Zugang
- *FTP*: Vertrauenswürdiger Kanal, vertrauenswürdige Verbindung

Alle genannten Funktionsanforderungen sind in mehrere Teile (Klassen, Familien und Komponenten) gegliedert. Durch das Zusammenfassen mehrerer Funktionalitätsanforderungen bzw. deren Komponenten lassen sich sogenannte **Schutzprofile** (Protection Profiles) definieren, die beispielsweise den typischen Funktionsumfang eines Verschlüsselungsmoduls oder einer Firewall repräsentieren. Auf Basis eines Schutzprofils – oder bei Bedarf auch unabhängig davon – werden **Sicherheitsziel**e (Security Targets) festgelegt. Die Sicherheitsziele sind derjenige Bestandteil einer Evaluierung, der die Funktionalität des Evaluationsgegenstands definieren. Wie bei ITSEC wird auch bei Common Criteria nach zwei Kriterien geprüft: Korrektheit und Wirksamkeit.

Korrektheit

Die Prüfung der Korrektheit ist bei Common Criteria ähnlich wie bei ITSEC gelöst. Während es bei ITSEC die Stufen E0 bis E6 gibt, existieren bei den Common Criteria die Stufen EAL1 bis EAL7 (EAL steht für **Evaluation Assurance Level**). Folgende Entsprechungen sind gegeben:

Common Criteria	ITSEC
–	E0
EAL1	–
EAL2	E1
EAL3	E2
EAL4	E3
EAL5	E4
EAL6	E5
EAL7	E6

Bei den untersten zwei Stufen (EAL1, E0) gibt es keine direkte Entsprechung. Während die ITSEC-Stufe E0 für keine Evaluierung steht, fordern die Common Criteria für EAL1 eine funktionale Prüfung. Funktional bedeutet, dass dem Evaluator nur das fertige Produkt, nicht aber der Quellcode oder eine Spezifikation zur Verfügung steht. Eine Common-Criteria-Stufe EAL0, die E0 entsprechen könnte, gibt es nicht.

Die Common Criteria erlauben es, dass der Evaluator auf Wunsch des Herstellers auch Korrektheitsprüfungen durchführen kann, die in der jeweiligen Stufe nicht vorgesehen sind. Wird beispielsweise ein Evaluationsgegenstand EAL4-evaluiert und gibt es zusätzliche Prüfungen, dann wird das Ergebnis als »EAL4 augmented« oder »EAL4+« bezeichnet. Dabei sollte man beachten, dass die Bezeichnung »augmented« bzw. »+« für verschiedene Produkte eine unterschiedliche Bedeutung haben kann. Halbwegs aussagekräftig ist ein Ausdruck wie »EAL4+« daher nur, wenn man die Namen der zusätzlichen Prüfungen dazu nennt.

Wirksamkeit

Im Gegensatz zu ITSEC sehen die Common Criteria keine Skala für die Wirksamkeit vor (lediglich für einige wenige Sicherheitsmechanismen gibt es spezielle Wirksamkeitsbewertungen, die in eine Evaluierung einfließen können). Dies widerspricht zwar dem Ziel, ein Maß für die Sicherheit zu finden, erlaubt aber mehr Flexibilität. Statt den drei Stufen hoch, mittel und niedrig gibt es bei den Common Criteria die bereits erwähnten Sicherheitsziele (Security Targets).

Während eine ITSEC-Zertifizierung mit einem Ausdruck wie »E4/hoch« oder »E2/mittel« angegeben wird, nennt man bei den Common Criteria den EAL und den Namen des Schutzprofils. Dies kann etwa so aussehen: »EAL4, BSI-PP-0815-2007«. Natürlich ist die Nennung des Schutzprofils etwas unhandlich, zumal Schutzprofile im Vergleich zu den ITSEC-Stufen »niedrig«, »mittel« und »hoch« nicht miteinander vergleichbar sind. Da IT-Sicherheitslösungen an sich jedoch auch vielfältig und schlecht vergleichbar sind, liegt dies in der Natur der Sache.

Ein Beispiel: ACOS

Da das Thema Common-Criteria-Evaluierung recht abstrakt ist, wollen wir uns nun ein reales Beispiel anschauen. Der betrachtete Evaluationsgegenstand ist die Smartcard-Lösung *ACOS* (Austria Card Operating System), die zur Erstellung digitaler Signaturen dient. ACOS wird von der österreichischen Firma *Austria Card* angeboten. Austria Card war zunächst eine Tochter der Österreichischen Nationalbank, also dem Gegenstück zur Deutschen Bundesbank. Inzwischen befindet sich das Unternehmen in Privatbesitz.

Die genaue Bezeichnung des Evaluationsgegenstands lautet »ACOS EMV-A03V1 Configuration A«. Nur für diese spezielle Version des Produkts ist die Zertifizierung gültig. Es handelt sich um eine EAL4+-Zertifizierung, die vom

Bundesamt für Sicherheit in der Informationstechnik (BSI) durchgeführt wurde (also von einer deutschen Behörde, nicht von einer österreichischen). Das »+« in der Evaluationsstufe kommt durch zwei zusätzliche Prüfaspekte zustande, denen das Produkt unterzogen wurde. Im Rahmen dieser zusätzlichen Tests wurden mögliche unsichere Zustände und bestimmte Schwachstellen untersucht.

Die wesentlichen Fakten zur Common-Criteria-Zertifizierung von ACOS stehen im 50-seitigen Zertifizierungsreport des BSI [Aust06]. Dieser Report ist öffentlich zugänglich (was nicht bei allen evaluierten Produkten der Fall ist). Im Zertifizierungsreport erfahren wir zunächst einmal, dass sich der Evaluationsgegenstand aus drei Bestandteilen zusammensetzt:

1. *Smartcard-Chip*: Dies ist die verwendete Hardware. Es handelt sich um ein Produkt der Firma Philips.
2. *ACOS Smartcard-Betriebssystem*: Dies ist das Betriebssystem, das die Hardware nutzbar macht.
3. *Software zur Signaturgenerierung*: Diese Software läuft auf dem ACOS-Betriebssystem. Sie unterstützt sowohl RSA- als auch ECDSA-Signaturen. Erstere entsprechen dem Standard PKCS#1 v1.5 (siehe Abschnitt 19.4.1) und nutzen eine Schlüssellänge bis 2.048 Bit. Letztere werden mit einer Schlüssellänge von maximal 256 Bit angefertigt. Zudem kann die Software Hashwerte mit SHA-1 generieren.

Eine PC-Software, die die Smartcard anspricht (also ein Treiber) ist nicht Bestandteil des Evaluationsgegenstands. Auf der Karte laufen neben der Signatur-Software einige weitere Programme, die ebenfalls nicht zum Evaluationsgegenstand gehören. Der Hersteller von ACOS hat also – wie allgemein üblich – eine enge Abgrenzung des Evaluationsgegenstands gewählt. Der Evaluationsgegenstand erfüllt folgende zwei Aufgaben:

1. Generierung von Signaturerstellungsdaten und von Signaturverifikationsdaten. Mit anderen Worten heißt dies: Die Smartcard kann ein asymmetrisches Schlüsselpaar selbst generieren.
2. Generierung digitaler Signaturen. Dies ist der eigentliche Zweck der Lösung: Sie soll zum digitalen Signieren eingesetzt werden.

Das verwendete Schutzprofil hat den Namen *BSI-PP-0006-2002* [Aust02]. Von den elf in den Common Criteria definierten Funktionsanforderungen nutzt der Evaluationsgegenstand sechs: FCS (kryptografische Unterstützung), FDP (Schutz der Benutzerdaten), FIA (Identifikation und Authentifizierung), FMT (Sicherheitsmanagement), FPT (Schutz der Sicherheitsfunktionen) und FTP (Vertrauenswürdiger Kanal, vertrauenswürdige Verbindung).

Der Evaluationsgegenstand nutzt jeweils mehrere Komponenten einer Funktionsanforderung. Insgesamt kommen 28 Komponenten zum Einsatz. 27 davon sind dem Common-Criteria-Standard entnommen, eine wurde neu definiert. Die

neu definierte Komponente hat den Namen *FTP_EMSEC.1* und gehört zur Funktionsanforderung FTP (vertrauenswürdiger Kanal, vertrauenswürdige Verbindung). Sie bezieht sich auf das Thema kompromittierende Abstrahlung.

Das verwendete Sicherheitsziel hat den Namen *BSI-DSZ-CC-0346* [Aust05]. In ihm sind die 28 Funktionsanforderungskomponenten jeweils in Teilkomponenten gegliedert. Jede Teilkomponente ist eine elementare Aussage wie: »Der Evaluationsgegenstand soll einen Signaturschlüssel löschen, sobald ein neuer erzeugt wird.« Insgesamt kommen über 100 derartiger Aussagen zusammen, wodurch das Sicherheitsziel im Detail beschrieben ist.

Die Stückzahl der ACOS-Karten mit CC-Evaluierung liegt im zweistelligen Millionenbereich. Die meisten Exemplare sind in Form österreichischer Bankkarten im Einsatz, weitere werden als Behördenausweise verwendet. Allerdings gibt es bisher nur etwa 50.000 Anwender, die die Signaturfunktionen dieser Karten nutzen. Abgesehen davon versucht die Firma Austria Card, ACOS auch als Plattform für digitale Personalausweise zu vermarkten, woran derzeit mehrere Staaten interessiert sind. Diese Entwicklung steht jedoch noch am Anfang.

Vor- und Nachteile der Common Criteria

Die Common Criteria haben ITSEC und TCSEC inzwischen verdrängt. Für die Hersteller von Krypto-Lösungen ist dies ein großer Vorteil, da ihnen nun ein weltweit anerkannter Standard zur Verfügung steht. Der Aufwand, der für eine Common-Criteria-Evaluierung anfällt, ist etwa gleich groß wie bei ITSEC. Hersteller und Evaluatoren mussten sich allerdings umstellen, da sich die beiden Standards in der Umsetzung durchaus unterscheiden. Auch die Umstellung auf die aktuelle Common-Criteria-Version 3.1 wird von Experten nicht gerade als trivial bezeichnet.

Die folgenden Betrachtungen der Vor- und Nachteile der Common Criteria gelten trotz der Unterschiede größtenteils auch für ITSEC. Wie man sich leicht vorstellen kann, sind Common-Criteria-Evaluierungen teuer. Schon für einen einfachen Evaluationsgegenstand und eine niedrige Evaluationsstufe muss der Hersteller einige Zehntausend Euro auf den Tisch blättern. Bei komplexeren Produkten gehen die Kosten schnell in die Hunderttausende. Als weiteres Problem kommt hinzu, dass ein Zertifikat stets nur für die evaluierte Version gilt. Schon ein kleiner Bugfix oder eine sonstige Änderung des Codes führt zu einer neuen Version, für die die Evaluierung nicht mehr gilt. Da eine Evaluierung ohne weiteres ein Jahr in Anspruch nehmen kann, ergibt sich nicht nur ein finanzielles, sondern auch ein zeitliches Problem.

Common-Criteria-evaluierte Produkte sind vor allem im Behördenbereich gefragt. Darüber hinaus gibt es gesetzliche Bestimmungen (etwa die Signaturgesetze in diversen Staaten), die eine Evaluierung fordern. Die meisten Gesetze lassen sowohl Common Criteria als auch ITSEC zu. Darüber hinaus hat eine CC-Evaluierung einen gewissen Marketingeffekt, der in den letzten Jahren von eini-

gen Unternehmen genutzt wird. Dennoch ist der Anteil der evaluierten Krypto-Produkte verglichen mit dem gesamten Markt recht klein.

Zu den Kritikpunkten an den Common Criteria gehört, dass die Kriterien sehr technisch sind. Benutzerfreundlichkeit ist dagegen kein Kriterium, obwohl dieser Aspekt die Sicherheit eines Computersystems maßgeblich beeinflusst. Viele Experten bemängeln zudem, dass die Common Criteria zu sehr an einzelnen Komponenten und zu wenig an Prozessen orientiert sind. Dieser Vorwurf ist aus Sicht der Kryptografie insofern berechtigt, als eine sichere Komponente noch kein sicheres Protokoll ausmacht.

Zusätzlich ist die Neutralität des Evaluators ein kritischer Punkt. Dieser hat ein wirtschaftliches Interesse daran, dass eine Evaluierung erfolgreich verläuft, da er vom Hersteller für seine Arbeit bezahlt wird. Dabei ist es nicht auszuschließen, dass ein Evaluator ein Auge zudrückt, um dem zahlenden Kunden einen Gefallen zu tun. Evaluierungsexperten verweisen jedoch zu Recht auf eine beträchtliche Zahl begonnener, aber nicht abgeschlossener Evaluierungen. Diese Fehlversuche belegen, dass ITSEC- und CC-Zertifikate nicht allein aus Gefälligkeit vergeben werden.

25.3 FIPS 140

FIPS 140 ist der dritte wichtige Evaluierungsstandard im Bereich der Kryptografie. Der vollständige Name dieses Standards lautet *Federal Information Processing Standards Publication 140*, die korrekte Abkürzung ist FIPS PUB 140 [FIPS-140]. In der Branche wird jedoch meist die handlichere Bezeichnung FIPS 140 verwendet, was ich im Folgenden auch tun werde. Die aktuelle Version des Standards hat die Nummer 2 und ist 2001 erschienen. Man spricht daher auch von FIPS 140-2.

Im Gegensatz zu ITSEC und Common Criteria ist FIPS 140 ausschließlich für die Anwendung in der Kryptografie vorgesehen. Nichtkryptografische Sicherheitsprodukte wie Firewalls oder sichere Datenbanken lassen sich damit also nicht evaluieren. Dafür lässt sich FIPS 140 sowohl auf Hardware- als auch auf Softwarelösungen anwenden. Der Evaluationsgegenstand wird in diesem Zusammenhang »kryptografisches Modul« genannt.

FIPS 140 ist vor allem in den USA weit verbreitet. Dort ist für Krypto-Lösungen, die von Behörden eingesetzt werden, eine FIPS-140-Evaluierung sogar vorgeschrieben. Auch in anderen Ländern gibt es Behörden, die FIPS-140-evaluierte Produkte nutzen, genauso wie viele Privatunternehmen. Der große Vorteil von FIPS 140 ist sein pragmatischer Ansatz. Der gesamte Standard ist simpler aufgebaut und verständlicher als ITSEC und Common Criteria. FIPS 140 enthält einen sehr konkreten Katalog von Anforderungen, die ein kryptografisches Modul erfüllen muss, um eine bestimmte Sicherheitseinstufung zu erhalten. Dabei gibt es lediglich vier Sicherheitsstufen, die schlicht mit Stufe 1 bis Stufe 4 (auf Englisch Level 1 bis Level 4) bezeichnet werden.

25.3.1 Die vier Stufen von FIPS 140

Stufe 1 ist die niedrigste Stufe, die FIPS 140 vorsieht. Stufe 4 ist die höchste. Bevor wir ins Detail gehen, schauen wir uns die vier Stufen im Überblick an. Dabei nehmen wir an, dass die Firma Krypt & Co. der Hersteller des zu evaluierenden Moduls ist.

Stufe 1

Die Anforderungen in Stufe 1 sind vergleichsweise gering. Wenn Krypt & Co. eine einfache Hardware- oder Softwarelösung professionell entwickelt und produziert, dann kann sie diese meist mit geringem Aufwand nach FIPS-140-Stufe 1 evaluieren lassen. Voraussetzungen sind eine ausführliche Dokumentation sowie einige technische Anforderungen, die kostengünstig erfüllbar sind.

Stufe 2

Um die FIPS-140-Stufe 2 zu erreichen, muss ein kryptografisches Modul einige zusätzliche Anforderungen erfüllen. Zum Beispiel müssen bei einer Hardwareimplementierung Manipulationen am Chip erkennbar sein (Einbruchsevidenz, siehe Abschnitt 17.3.2). Das Modul darf erst nutzbar sein, nachdem der Anwender sich authentisiert hat (etwa durch ein Passwort). Falls Krypt & Co. die kryptografischen Funktionen des Moduls auf einem der üblichen Betriebssysteme realisiert (z.B. Linux, Windows oder Smartcard-Betriebssystem), dann muss dieses nach einem vorgegebenen Schutzprofil nach den Common Criteria EAL2-evaluiert sein. Auch eine äquivalente Evaluierung (etwa E1 nach ITSEC) ist zulässig.

Stufe 3

Die FIPS-140-Stufe 3 sieht im Falle einer Hardwarerealisierung vor, dass das evaluierte Modul noch stärker als bei Level 2 gegen einen gewaltsamen Angriff auf die Hardware geschützt ist (inklusive Einbruchsschutz, siehe Abschnitt 17.3.2). Ein gegebenenfalls verwendetes Betriebssystem muss nach Common Criteria EAL3-zertifiziert sein (oder gleichwertig nach anderen Kriterien, z.B. ITSEC). Darüber hinaus gibt es zahlreiche technische Anforderungen, die Krypt & Co. erfüllen muss.

Stufe 4

Stufe 4 ist die höchste Stufe, die FIPS 140 vorsieht. Hier sind für eine Hardwarerealisierung noch weitergehende physikalische Schutzmaßnahmen (einschließlich eines starken Einbruchsschutzes) gefordert als bei Stufe 3. Umwelteinflüsse wie Feuer oder mechanische Einwirkungen dürfen kein Fehlverhalten verursachen (gegebenenfalls muss das Modul seinen Betrieb selbstständig einstel-

len). Das Betriebssystem (falls vorhanden) muss nach Common Criteria EAL4 oder nach anderen Kriterien gleichwertig zertifiziert sein.

25.3.2 Die Sicherheitsbereiche von FIPS 140

Die Kriterien, nach denen ein Evaluator entscheidet, in welche der vier FIPS-140-Stufen ein Evaluationsgegenstand einzuordnen ist, sind in elf Kapitel (Sicherheitsbereiche) gegliedert. Ein Blick auf die FIPS-140-Sicherheitsbereiche lohnt sich nicht nur für jemanden, der an einer Evaluierung interessiert ist. Vielmehr findet jeder, der ein sicheres Krypto-Modul entwickeln will, in den gut lesbar geschriebenen FIPS-140-Kriterien interessante Anregungen. Insbesondere liefert die Lektüre einen guten Überblick zum Thema Real-World-Attacken. In den nun folgenden Betrachtungen nennen die Überschriften der Unterkapitel den Namen des jeweiligen FIPS-140-Sicherheitsbereichs auf Deutsch, darunter ist jeweils in kursiver Schrift der englische Originaltitel aufgeführt.

Spezifikation

Cryptographic module specification (what must be documented)

Natürlich muss Krypt & Co. den FIPS-140-Evaluationsgegenstand sauber dokumentieren. Der erste der elf FIPS-140-Sicherheitsbereiche nennt daher Kriterien zur Beschreibung der Komponenten, Schnittstellen, Sicherheitsfunktionen und Sicherheitsrichtlinien des kryptografischen Moduls. Diese Dokumentationsanforderungen sind für alle vier FIPS-140-Stufen gleich.

Anschlüsse und Schnittstellen

Cryptographic module ports and interfaces (what information flows in and out, and how it must be segregated)

In diesem zweiten Sicherheitsbereich geht es um die Kommunikation des Evaluationsgegenstands mit der Außenwelt. Unabhängig von der FIPS-140-Stufe muss das kryptografische Modul über die folgenden vier logischen Schnittstellen ansprechbar sein: Dateneingang, Datenausgang, Steuerungseingang und Statusausgang. Für die Stufen 3 und 4 wird eine klare Trennung zwischen den Schnittstellen gefordert.

Rollen, Dienste und Authentifizierung

Roles, services and authentication (who can do what with the module, and how this is checked)

Ein FIPS-140-Evaluationsgegenstand bietet verschiedene Dienste an (z.B. Verschlüsselung, digitale Signatur, Initialisierung), um seinen Zweck zu erfüllen. Die

FIPS-140-Kriterien schreiben vor, dass für die Nutzung der diversen Dienste mindestens zwei Rollen unterschieden werden müssen. Zum einen gibt es den Anwender, der das Modul nutzen darf (z.B. zum Verschlüsseln); und zum anderen ist ein Administrator (*Crypto Officer*) vorgesehen, der Administrationsaufgaben durchführen darf. Diese Anforderung an das Rollenkonzept gilt für alle vier FIPS-140-Stufen.

Unterschiedliche Anforderungen gibt es dagegen an die Authentifizierung. Anwender, Administrator (und weitere Rollen, falls sie existieren), müssen sich vor einem Zugriff auf das Modul wie folgt authentisieren:

- In Stufe 1 fordern die FIPS-140-Kriterien gar keine Authentifizierung.
- In Stufe 2 ist eine Authentifizierung (in der Regel durch ein Passwort) vorgesehen. Der Evaluationsgegenstand muss in der Lage sein, Anwender und Administratoren jeweils als solche zu erkennen. Es ist jedoch nicht notwendig, dass das Modul verschiedene Administratoren voneinander unterscheiden kann. Auch verschiedene Anwender muss das Modul nicht unterscheiden können.
- In Stufe 3 und 4 muss der Evaluationsgegenstand verschiedene Anwender voneinander unterscheiden können. Gleiches gilt für Administratoren.

Zustandsmaschinen-Modell

Finite state model (documentation of the high-level states the module can be in, and how transitions occur)

Dieser FIPS-140-Sicherheitsbereich fordert, dass Krypt & Co. dem Evaluator eine Beschreibung des Evaluationsgegenstands in Form eines *Zustandsmaschinen-Modells* vorlegt. Als Zustandsmaschinen-Modell bezeichnet man in der Informatik ein Diagramm, das alle Zustände eines Objekts sowie die Übergänge zwischen diesen beschreibt.

Physikalische Sicherheit

Physical security (tamper evidence and resistance, and robustness against extreme environmental conditions)

Die Anforderungen an die physikalische Sicherheit des Evaluationsgegenstands sind die komplexesten im FIPS-140-Standard. Der entsprechende Sicherheitsbereich sieht eine lange Liste von Sicherheitskriterien vor, die das Modul vor physikalischen Angriffen schützen sollen. Diese Anforderungen gelten jedoch nur, wenn der Evaluationsgegenstand als Hardwarekomponente realisiert ist. Handelt es sich dagegen um eine reine Softwarelösung, dann spielen alle im Folgenden genannten Kriterien keine Rolle.

Wenn wir also von einer Hardwarelösung ausgehen, dann sieht FIPS 140 eine Einteilung in drei Klassen vor:

- *Ein-Chip-Module*: Hierzu gehören beispielsweise Smartcards.
- *Eingebettete Mehr-Chip-Module*: Dies sind beispielsweise Adapter.
- *Nichteingebettete Mehr-Chip-Module*: Router, E-Mail-Verschlüsselungs-Gateways oder VPN-Server sind Beispiele hierfür.

Für die FIPS-140-Stufe 1 gelten die folgenden Anforderungen:

- Alle Hardwaremodule dieser Stufe müssen produktionsreife Lösungen sein (also keine Prototypen).
- *Ein-Chip-Module*: Hierfür sieht Stufe 1 keine weiteren Anforderungen vor.
- *Eingebettete Mehr-Chip-Module*: Hier muss, falls anwendbar, Krypt & Co. ein handelsübliches Gehäuse oder eine Abdeckung vorsehen.
- *Nichteingebettete Mehr-Chip-Module*: In diesem Fall ist ein handelsübliches Gehäuse gefordert.

Für FIPS-140-Stufe 2 gelten die folgenden Anforderungen:

- Alle Hardwaremodule der Stufe 2 müssen so beschaffen sein, dass Manipulationen daran erkennbar sind (Einbruchsevidenz).
- *Ein-Chip-Module*: Diese muss Krypt & Co. so beschichten, dass ein gewaltsamer Zugriff auf das Modul erkennbar ist.
- *Eingebettete Mehr-Chip-Module*: Bei diesen kann entweder eine Beschichtung oder der Einbau in ein stabiles Gehäuse sicherstellen, dass Manipulationen auffallen.
- *Nichteingebettete Mehr-Chip-Module*: Solche Module muss Krypt & Co. in ein blickdichtes, verschließbares Gehäuse einbauen.

In Stufe 3 gelten die folgenden Anforderungen:

- Alle Hardwaremodule dieser Stufe benötigen einen Löschmechanismus, der beim Verdacht auf Manipulationen alle Schlüssel und sonstigen Geheiminformationen automatisch löscht (Einbruchsschutz).
- *Ein-Chip-Module*: Diese müssen entweder in ein besonders stabiles Gehäuse eingebaut werden, oder das Modul muss so beschaffen sein, dass es bei einer physikalischen Beeinträchtigung nicht mehr funktioniert.
- *Eingebettetes Mehr-Chip-Modul*: Hier gelten dieselben Anforderungen wie bei Ein-Chip-Modulen.
- *Nichteingebettetes Mehr-Chip-Modul*: Hier gelten ebenfalls dieselben Anforderungen wie bei Ein-Chip-Modulen.

In Stufe 4 gelten die folgenden Anforderungen:

- Alle Hardwaremodule dieser Stufe benötigen zusätzlich einen Schutz gegen Umwelteinflüsse wie Hitze oder mechanische Beinträchtigungen. Entweder

muss das Modul gegen solche Einflüsse reistent sein oder es muss gegebenenfalls seinen Betrieb einstellen.

- *Ein-Chip-Module*: Krypt & Co. muss ein solches in Stufe 4 in eine Schutzhülle eingießen. Ein Entfernen der Schutzhülle muss dazu führen, dass das Modul nicht mehr nutzbar ist.
- *Eingebettete Mehr-Chip-Module*: Ein eingebettetes Mehr-Chip-Modul muss einen Mechanismus enthalten, der bei einer physikalischen Beeinträchtigung alle vertraulichen Daten auf dem Modul löscht.
- *Nichteingebettetes Mehr-Chip-Modul*: Hier gelten dieselben Anforderungen wie bei einem eingebetteten Mehr-Chip-Modul.

Einsatzumgebung

Operational environment (what sort of operating system the module uses and is used by)

Die Einsatzumgebung eines Evaluationsgegenstands kann laut FIPS 140 zwei Formen annehmen:

- *Modifizierbar (modifiable operational environment)*: Als modifizierbare Einsatzumgebung gelten beispielsweise die gängigen PC-, PDA- und Smartcard-Betriebssysteme. Sie alle erlauben Schreibzugriffe auf große Teile des Speichers sowie das Installieren unterschiedlicher Software. Eine modifizierbare Einsatzumgebung kann man daher mit einem Betriebssystem gleichsetzen. Ist der Evaluationsgegenstand ein Stück Software, dann ist die modifizierbare Einsatzumgebung das Betriebssystem, auf dem dieser läuft. Im Falle einer Hardware ist die modifizierbare Einsatzumgebung Teil des Evaluationsgegenstands.
- *Eingeschränkt (limited operational environment)*: Eine eingeschränkte Einsatzumgebung liegt beispielsweise bei einer Hardwareimplementierung vor, die keine Codeänderungen oder -erweiterungen zulässt.

Für eingeschränkte Einsatzumgebungen (also für Betriebssysteme) sieht FIPS 140 eine ganze Reihe von Kriterien vor, die für die jeweilige Stufe erfüllt sein müssen:

- *Stufe 1*: Für die niedrigste Stufe muss das Betriebssystem sicherstellen, dass die vom Evaluationsgegenstand ausgeführten Prozesse unter der Kontrolle des Anwenders liegen. Darüber hinaus muss das Modul den eigenen Code mithilfe einer digitalen Signatur oder einer schlüsselabhängigen Hashfunktion überprüfen (dies wird auch im Sicherheitsbereich Selbsttests gefordert).
- *Stufe 2*: In dieser Stufe muss das verwendete Betriebssystem nach Common Criteria evaluiert sein (EAL2 und ein spezielles Schutzprofil). Auch eine äquivalente Evaluierung, z.B. nach ITSEC, ist zulässig. Darüber hinaus ist eine sichere Trennung zwischen den unterschiedlichen Rollen und Prozessen sowie eine Logdaten-Erhebung vorgeschrieben.

- *Stufe 3*: Hier ist eine EAL3-Zertifizierung des Betriebssystems vorgeschrieben (wiederum mit einem speziellen Schutzprofil, und wiederum ist alternativ auch eine äquivalente Evaluierung nach anderen Kriterien zulässig). Die Trennung von Prozessen muss sich auch auf die Schnittstellen beziehen. Die Logdaten-Erhebung muss detaillierter sein als bei Stufe 2.
- *Stufe 4*: Hier ist es eine EAL4-Zertifizierung mit speziellem Schutzprofil (oder äquivalent), die gefordert ist. Ansonsten gelten dieselben Anforderungen wie bei Stufe 3.

Für eingeschränkte Einsatzumgebungen gibt es in diesem Sicherheitsbereich keine Anforderungen.

Schlüsselmanagement

Cryptographic key management (generation, entry, output, storage and destruction of keys)

Der FIPS-140-Standard nennt sechs Teilbereiche, die den Umgang mit Schlüsseln regeln. Die Unterschiede zwischen den vier FIPS-140-Stufen sind in den meisten Fällen gering. Hier die Liste der sechs Teilbereiche:

1. *Zufallsgeneratoren*: Für Zufallsgeneratoren, die Schlüssel generieren, muss das Modul Selbsttests vorsehen (siehe Sicherheitsbereich Selbsttests). Der FIPS-140-Standard unterscheidet zwischen echten Zufallsgeneratoren und Pseudozufallsgeneratoren (erstere werden nichtdeterministisch, letztere deterministisch genannt). Der Standard nennt einige Pseudozufallsgeneratoren, deren Einsatz zulässig ist. Es handelt sich dabei um verschiedene Verfahren, die den DES oder SHA-1 in der Fortschaltfunktion verwenden. Einen als sicher eingestuften echten Zufallsgenerator gibt es in FIPS 140 bisher nicht, was sich jedoch ändern könnte. Vorläufig kann sich der Hersteller daher einen beliebigen echten Zufallsgenerator aussuchen, sofern dieser dem Stand der Technik entspricht.
2. *Schlüsselgeneratoren*: Wenn ein Schlüssel direkt aus einer Zufallszahl generiert wird, dann muss ein Zufallsgenerator zum Einsatz kommen, der den Anforderungen aus dem vorhergehenden Aufzählungspunkt entspricht. Wird dagegen zur Schlüsselgenerierung ein importierter Wert verwendet, dann muss dieser den Vorschriften für den Schlüsselimport entsprechen (siehe übernächsten Aufzählungspunkt).
3. *Schlüsselaustausch*: Sowohl ein manueller Schlüsselaustausch als auch die Verwendung eines asymmetrischen Verfahrens sind zulässig.
4. *Schlüsselimport und -export*: Der Evaluationsgegenstand darf in allen vier FIPS-140-Stufen das Importieren und Exportieren von Schlüsseln unterstützen (die Schlüsselgenerierung muss also nicht innerhalb des Moduls stattfin-

den). Für Stufe 1 und 2 gilt: Wird ein Schlüssel von oder in ein anderes kryptografisches Modul importiert oder exportiert, dann muss der Schlüsseltransport verschlüsselt erfolgen. In Stufe 3 und 4 muss zudem auch ein manuell generierter Schlüssel entweder verschlüsselt transportiert oder auf mehrere Administratoren verteilt werden.

5. *Schlüsselspeicherung*: In allen vier Stufen müssen private und geheime Schlüssel so gespeichert werden, dass sie nicht auslesbar sind.
6. *Löschen eines Schlüssels*: Der Evaluationsgegenstand muss die Möglichkeit bieten, Schlüssel und andere geheime Informationen sicher zu löschen.

Elektromagnetische Störung und elektromagnetische Verträglichkeit

Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

Dieser Sicherheitsbereich bezieht sich auf die elektromagnetische Strahlung, die jedes Computermodul abgibt. Diese Strahlung kann Rückschlüsse auf die verarbeiteten Daten erlauben (kompromittierende Abstrahlung), was sich für Seitenkanalangriffe nutzen lässt. Kompromittierende Abstrahlung ist in diesem FIPS-140-Sicherheitsbereich jedoch nicht das Thema. Stattdessen geht es darum, dass die Strahlung des Evaluationsgegenstands andere Geräte nicht stören darf und dass der Evaluationsgegenstand selbst gegen Störungen durch Strahlung abgesichert sein muss. Um diese Ziele zu gewährleisten, bezieht sich der FIPS-140-Standard an dieser Stelle auf ein US-Gesetz, das verschiedene Stufen des Strahlungsschutzes und der Strahlungsempfindlichkeit vorsieht.

Selbsttests

Self-tests (what must be tested and when, and what must be done if a test fails)

Um eine FIPS-140-Zertifizierung zu erhalten, muss der Evaluationsgegenstand verschiedene Selbsttests unterstützen. Der Standard unterscheidet in diesem Zusammenhang zwischen Selbsttests, die nach dem Systemstart durchgeführt werden (initialer Test), und solchen, die in bestimmten Situationen stattfinden (konditionaler Test). Folgende initiale Tests sind vorgeschrieben:

- *Algorithmentest*: Dies erfolgt durch die Eingabe eines Werts und die Kontrolle, ob die Ausgabe (z.B. Geheimtext) korrekt ist.
- *Integritätstest*: Ein solcher muss mithilfe von Prüfsummen oder einer kryptografischen Hashfunktion überprüfen, ob der Code des Moduls korrekt ist.
- *Funktionstest*: Alle sicherheitskritischen Funktionen müssen initial getestet werden.

Folgende konditionale Tests gibt es:

- *Konsistenztest*: Ein solcher muss nach dem Generieren eines Schlüsselpaars durchgeführt werden. Er stellt sicher, dass die Generierung korrekt verlaufen ist.
- *Ladetest*: Wenn neuer Code geladen wird, dann muss durch eine schlüsselabhängige Hashfunktion oder eine digitale Signatur sichergestellt werden, dass dieser nicht verändert wurde.
- *Schlüsseleingabetest*: Schlüssel, die eingegeben werden, müssen durch eine Prüfsumme vor Fehlern geschützt sein.
- *Zufallsgeneratorentest*: Kommt ein Zufallsgenerator zum Einsatz, dann müssen dessen Ausgaben regelmäßig getestet werden.
- *Umgehungstest*: Falls das Modul einen Leerlauf-Modus anbietet, muss durch Tests geprüft werden, dass dieser nicht versehentlich zum Einsatz kommt.

Zuverlässigkeit des Designs

Design assurance (what documentation must be provided to demonstrate that the module has been well designed and implemented)

Dieser Sicherheitsbereich geht auf folgende Gesichtspunkte ein

- *Konfigurationsmanagement*: Ein solches muss innerhalb des Moduls vorhanden sein (Konfigurationsdaten dürfen also nicht extern verwaltet werden).
- *Auslieferung und Betrieb*: In Stufe 1 muss nur die Installation, in den drei anderen auch die Auslieferung definiert und dokumentiert werden.
- *Entwicklung*: In Stufe 1 muss Krypt & Co. den Quellcode gegenüber dem Evaluator offenlegen. In Stufe 2 wird zusätzlich eine Spezifikation gefordert. Stufe 3 fordert zusätzlich die Verwendung einer höheren Programmiersprache. Stufe 4 setzt eine ganze Liste zusätzlicher Anforderungen hinzu: formales Modell, kommentierter Quelltext, Beweis der Übereinstimmung zwischen Design und Spezifikation.
- *Gebrauchsanweisung*: Sowohl für den Administrator als auch für den Anwender muss es eine Gebrauchsanweisung geben.

Verhinderung weiterer Angriffe

Mitigation of other attacks (if a module is designed to mitigate against, say, TEMPEST attacks, then its documentation must say how)

Der letzte FIPS-140-Sicherheitsbereich bezieht sich auf einige weitere Sicherheitsbedrohungen, die an anderer Stelle des Standards nicht enthalten sind. Vier davon werden genannt: Stromanalyse, Zeitanalyse, Fehleranalyse und kompromittierende Abstrahlung. Alle vier gehören in den Bereich der Seitenkanalangriffe und wurden wohl deshalb nicht an anderer Stelle in den Standard aufgenommen,

weil sie noch recht neu sind (Seitenkanalangriffe kamen erst Ende der neunziger Jahre auf). Man kann davon ausgehen, dass es in der nächsten Version von FIPS 140 einen eigenen Sicherheitsbereich für derartige Angriffe mit genau definierten Kriterien geben wird. Bisher heißt es nur lapidar: »Wenn ein kryptografisches Modul so entwickelt wurde, dass es einen oder mehrere spezielle Angriffe abwehren soll, dann sollen die Sicherheitsrichtlinien des Moduls die verwendeten Sicherheitsmechanismen zur Abwehr dieser Angriffe beschreiben.«

25.3.3 Bewertung von FIPS-140

FIPS 140 ist ein Standard, in dem die typische Mentalität der USA erkennbar ist. Der gesamte Inhalt ist pragmatisch und auf die Praxis ausgerichtet. Als Richtlinien dienen nicht irgendwelche theoretischen Sicherheitsüberlegungen, sondern Erfahrungen aus der realen Welt. Eine FIPS-140-Evaluierung ist daher deutlich kostengünstiger als eine Evaluierung nach Common Criteria. Leider bevorzugen deutsche und europäische Gesetze eindeutig ITSEC und Common Criteria, obwohl FIPS 140 in vielen Fällen eine interessante Alternative wäre. Offenbar scheuen sich die Gesetzgeber davor, einen US-Standard als Grundlage für ein europäisches Gesetz zu verwenden.

Dennoch wäre es falsch, FIPS 140 und Common Criteria gegeneinander auszuspielen. Es handelt sich schlichtweg um zwei unterschiedliche Ansätze. Die Common Criteria ermöglichen die Evaluierung einer beliebigen Sicherheitslösung und sind daher viel abstrakter aufgebaut. FIPS 140 beschränkt sich dagegen auf kryptografische Module und nennt deshalb viele Anforderungen ganz konkret. Auffällig ist, dass die Fehlerfreiheit bei den Common Criteria eine deutlich wichtigere Rolle spielt als bei FIPS 140. Bei Letzterem kommt dieses Thema nur als Bestandteil der Zuverlässigkeit des Designs zur Sprache, während die Common Criteria einen großen Teil der Evaluierung von der Strategie zur Fehlervermeidung abhängig machen.

Ohne Zweifel hat FIPS 140 auch seine Nachteile. Ein Kritikpunkt ist, dass die Beschränkung auf vier Sicherheitsstufen zu viele unterschiedliche Implementierungen in einen Topf wirft. Es gibt nun einmal einen wesentlichen Unterschied zwischen einem Hardwaremodul, das physikalisch geschützt ist, und einer Software, die auf einem PC läuft. Dennoch können beide Realisierungen dieselbe FIPS-140-Stufe besitzen.

Darüber hinaus fehlen bei FIPS 140 einige wichtige Gesichtspunkte. Neben den bereits erwähnten Seitenkanalangriffen glänzt vor allem das Thema Benutzerfreundlichkeit mit Abwesenheit. Dies ist zweifellos ein Mangel, denn ein kryptografisches Modul ist nicht einmal die Hälfte wert, wenn es falsch bedient wird. In der nächsten FIPS-140-Version ist ein Sicherheitsbereich Benutzerfreundlichkeit daher Pflicht.

25.4 Fazit und Alternativen

Keine Frage, weder eine ITSEC- noch eine Common-Criteria- noch eine FIPS-140-Evaluierung ist ein Wundermittel. Da man Sicherheit jedoch prinzipiell nicht messen kann, wird es wohl nie eine perfekte Lösung geben. Ein ITSEC-, CC- oder FIPS-140-Zertifikat ist jedoch allemal besser als gar kein objektives Bewertungskriterium für eine kryptografische Lösung und signalisiert immerhin, dass die Entwickler alles Erdenkliche unternommen haben, um sicherheitsrelevante Schwachstellen zu vermeiden.

25.4.1 Open Source als Alternative

Wer ein von unabhängigen Experten geprüftes Krypto-Produkt einsetzen will, aber keine evaluierte Lösung zur Verfügung hat, kann auf eine interessante Alternative zurückgreifen: Software mit offenem Quellcode. Der Vorteil hierbei ist klar: Ein Krypto-Modul, dessen Quellcode öffentlich verfügbar ist, kann von jedem untersucht werden. Gibt es genug Fachleute, die von dieser Möglichkeit Gebrauch machen, dann fallen Programmierfehler und absichtlich eingebaute Schwachstellen früher oder später auf und können behoben werden. Auch sonstige Sicherheitsprobleme lassen sich auf diese Weise abstellen.

Open Source

Die bedeutendste Variante von Software mit offenem Quellcode ist Open-Source-Software. Dabei handelt es sich um Programme, die für alle frei verwendbar sind und beliebig geändert werden dürfen. Das Betriebssystem Linux und der Browser Mozilla Firefox, die beide mehrere Krypto-Funktionen bieten, sind die bekanntesten Beispiele dafür. Die wichtigste Open-Source-Software, die ausschließlich der Kryptografie gewidmet ist, ist das Verschlüsselungs- und Signaturprogramm GnuPG. Die früheren Versionen von PGP waren zwar nicht Open Source, hatten jedoch einen offengelegten Quellcode. Bei den genannten Programmen funktionierte das Aufspüren von Schwachstellen durch die Community recht gut. Man kann daher tatsächlich davon ausgehen, dass etwa GnuPG eine mit einer evaluierten Software vergleichbare Sicherheit bietet.

Ist Open Source (oder zumindest offengelegter Quellcode) somit der Königsweg zu sicherer Software? Sicherlich nicht, denn dazu gibt es zu wenig Auswahl. Lösungen wie PGP oder GnuPG, die über mehrere Versionen hinweg von aufmerksamen Fachleuten begutachtet und ständig verbessert wurden, sind leider die Ausnahme. Es gibt eben nur begrenzt viele Menschen, die sich an der Entwicklung von Open-Source-Software beteiligen und vermutlich noch weniger, die sich die Mühe machen, Quellcode zu analysieren. Dass es für spezielle Anforderungen oft überhaupt keine Open-Source-Lösung gibt, dürfte ohnehin klar sein. Für einen kommerziellen Hersteller ist die Veröffentlichung des Quellcodes auch

nur von kleinem Nutzen, da sich meist nicht genügend Leute finden, die eine Analyse vornehmen.

Beispiele

Die folgende Liste nennt die wichtigsten Krypto-Programme mit offenem Quellcode:

- *Angewandte Kryptographie*: In Bruce Schneiers Buch *Angewandte Kryptographie* sind zahlreiche kryptografische Verfahren als Quelltext abgedruckt [Schn96]. Der US-Ausgabe liegt zudem eine CD bei.
- *AxCrypt*: Dies ist eine Software zur manuellen Dateiverschlüsselung für Windows-Betriebssysteme.
- *Bouncy Castle*: Hierbei handelt es sich um eine C++-Krypto-Bibliothek.
- *CrossCrypt*: Dies ist eine Software zur transparenten Dateiverschlüsselung für Windows-Betriebssysteme.
- *Crypto++*: Diese C++-Bibliothek unterstützt nahezu alle gängigen Krypto-Verfahren.
- *CrypTool*: Um die Krypto-Lernsoftware CrypTool geht es in Abschnitt 39.4.4.
- *Flexiprovider*: Dies ist eine Java-Krypto-Bibliothek.
- *FreeOTFE*: Dies ist eine weitere Software zur transparenten Dateiverschlüsselung auf Windows-Betriebssystemen.
- *GMP*: Dies ist eine Bibliothek, die Funktionen für die Langzahl-Arithmetik zur Verfügung stellt.
- *GnuPG*: Populäre Krypto-Software, die insbesondere zur Verschlüsselung von E-Mails verwendet wird. GnuPG ist die Open-Source-Alternative zu PGP.
- *OpenSSH*: Dies ist eine offene Implementierung der Secure Shell (Abschnitt 37.3).
- *OpenSSL*: Hierbei handelt es sich um eine Implemetierung des SSL-Protokolls (Kapitel 35).
- *OpenXPKI*: Diese Software zum Betrieb einer Public-Key-Infrastruktur ist der Nachfolger von OpenCA.
- *TrueCrypt*: Dies ist eine weitere Software zur transparenten Dateiverschlüsselung auf Windows-Betriebssystemen.

25.4.2 Theorie und Praxis

Bei genauerer Betrachtung bleibt die Erkenntnis, dass das schwierige Thema Krypto-Evaluierung und -Zertifizierung eine weitere Facette der in Kapitel 17 beschriebenen Problematik ist: Es ist äußerst schwierig, die in der Theorie hervorragenden Sicherheitseigenschaften der Kryptografie in die Praxis umzusetzen. Selbst ein bombensicheres Krypto-Verfahren ist nutzlos, wenn Alice und Bob es

falsch einsetzen oder ihre Schlüssel nicht geheim halten. Man kann es auch so formulieren: Es ist nicht nur schwierig, Kryptografie in die Praxis umzusetzen, sondern es ist auch schwierig, die Qualität der Umsetzung zu bewerten. Sicherheit ist nun einmal nicht messbar.

Teil 4

Public-Key-Infrastrukturen

26 Public-Key-Infrastrukturen

In diesem Kapitel werden Sie erfahren, dass es beim praktischen Einsatz von RSA, Diffie-Hellman und anderer asymmetrischer Verfahren zahlreiche Fallstricke gibt, die nur durch den Aufbau einer geeigneten Infrastruktur gelöst werden können. Eine solche Infrastruktur wird als **Public-Key-Infrastruktur** (**PKI**) bezeichnet. Der Aufbau einer Public-Key-Infrastrukturen ist in vielen Fällen eine Voraussetzung für den sinnvollen Einsatz asymmetrischer Kryptografie.

26.1 Warum brauchen wir eine PKI?

Beim Einsatz asymmetrischer Verfahren ohne zusätzliche Infrastruktur ergeben sich einige Probleme. Diese lassen sich in vier Bereiche aufteilen.

26.1.1 Authentizität der Schlüssel

Alice möchte Bob eine verschlüsselte E-Mail zuschicken. Dazu verwendet sie Bobs öffentlichen Schlüssel. Wenn Bösewicht Mallory es jedoch schafft, Alice seinen eigenen öffentlichen Schlüssel als den von Bob unterzujubeln, dann kann er selbst die Mail entschlüsseln. Mallory hat für einen solchen Angriff mehrere Möglichkeiten. Wenn Bob Alice seinen öffentlichen Schlüssel übers Netz zuschickt, dann kann er diesen abfangen und durch seinen eigenen ersetzen (Man-in-the-Middle-Attacke). Dasselbe kann Mallory tun, wenn sich Alice Bobs Schlüssel von einem Server herunterlädt. Außerdem kann Mallory versuchen, seinen eigenen Schlüssel im Netz als den von Bob zu verbreiten. Das Problem hierbei ist, dass man einem öffentlichen Schlüssel nicht ansieht, wem er gehört.

26.1.2 Sperrung von Schlüsseln

Mallory hat Alices privaten Schlüssel von deren Festplatte gestohlen. Damit kann er verschlüsselte Nachrichten lesen und digitale Signaturen fälschen. Zum Glück hat Alice den Diebstahl bemerkt. Sie verwendet den alten privaten Schlüssel daher nicht mehr (man spricht von einer **Sperrung**) und generiert sich stattdessen ein neues Schlüsselpaar. Doch woher sollen die Kommunikationspartner von Alice wissen, dass Alices alter Schlüssel gesperrt ist? Das Problem hierbei ist, dass man einem öffentlichen Schlüssel nicht ansieht, ob er gesperrt ist.

26.1.3 Verbindlichkeit

Der Sinn einer digitalen Signatur ist es unter anderem, für Verbindlichkeit zu sorgen. Dies bedeutet, dass Alice im Nachhinein keine Möglichkeit haben darf, eine angefertigte Signatur abzustreiten. Ein solches Abstreiten ist aber recht einfach: Alice behauptet, der Schlüssel, mit dem sie eine Signatur angefertigt hat, sei gar nicht ihrer. Das Problem hierbei ist, dass man einem öffentlichen Schlüssel nicht ansieht, wem er gehört.

26.1.4 Durchsetzen einer Policy

Die Firma Krypt & Co. will jeden Mitarbeiter mit einem Schlüsselpaar ausstatten. Dabei hat sie folgende Anforderungen:

- Jeder Mitarbeiter soll nur ein Schlüsselpaar bekommen, nicht mehrere.
- Schlüsselpaare sollen nur zum Verschlüsseln eingesetzt werden, nicht aber zur digitalen Signatur.
- Alle öffentlichen Schlüssel sollen zentral registriert werden.
- Es soll sichergestellt werden, dass jeder Mitarbeiter ausreichend große Schlüssellängen verwendet.

- Jedes Schlüsselpaar soll spätestens nach zwei Jahren durch ein neues ersetzt werden.
- Wenn ein Mitarbeiter das Unternehmen verlässt, dann soll dessen öffentlicher Schlüssel gesperrt werden.

All diese Probleme können von Krypt & Co. nur dann gelöst werden, wenn es eine Möglichkeit gibt, bestimmte Vorgehensweisen durchzusetzen. Diese Vorgehensweisen fasst man unter dem Begriff **Policy** zusammen. Das Problem hierbei ist, dass man einem öffentlichen Schlüssel nicht ansieht, welche Policy für ihn gültig ist.

26.2 Digitale Zertifikate

Die vier genannten Probleme haben alle damit zu tun, dass Bobs öffentlicher Schlüssel nur eine Bit-Folge ist, die keine zusätzlichen Informationen enthält. Um diesem Mangel abzuhelfen, kann man eine Datenstruktur einführen, die Bobs Schlüssel sowie Bobs Name beinhaltet und die außerdem digital signiert ist. Einen signierten Datensatz mit diesen Eigenschaften wird **digitales Zertifikat** (oder auch nur **Zertifikat**) genannt. Das Signieren eines Zertifikats wird als **Zertifizieren** bezeichnet. Neben Name und Schlüssel kann ein Zertifikat noch weitere Informationen enthalten: etwa eine Gültigkeitsdauer, eine Seriennummer und Angaben über den Verwendungszweck des öffentlichen Schlüssels. Die Signatur in einem digitalen Zertifikat bezieht sich immer auf alle darin enthaltenen Angaben.

Eine wichtige Frage bei einem digitalen Zertifikat lautet, wer ein solches signiert (wer also die Zertifizierung durchführt). Die naheliegendste Möglichkeit besteht darin, dass eine unabhängige Instanz eingerichtet wird, die diese Aufgabe übernimmt. Eine solche Instanz wird **Zertifizierungsstelle** oder **Certification Authority** (**CA**) genannt. Wenn in einem Unternehmen Public-Key-Verfahren eingesetzt werden sollen, dann bietet es sich an, die CA durch die IT-Abteilung betreiben zu lassen. Es ist jedoch auch möglich, dass etwa eine Behörde als CA-Betreiber auftritt oder dass ein Unternehmen die Dienste einer CA am Markt anbietet.

Einen Sinn hat eine CA nur dann, wenn Alice und Bob in der Lage sind, deren Signaturen zu verifizieren (auf jedem Zertifikat befindet sich schließlich eine solche). Zu diesem Zweck müssen Alice und Bob den öffentlichen Schlüssel der CA kennen. Nun stellen sich jedoch genau dieselben Fragen, die wir eingangs schon behandelt haben: Woher wissen Alice und Bob, ob es der richtige CA-Schlüssel ist? Woher wissen sie, ob er nicht gesperrt wurde? Ist die Gültigkeitsdauer des CA-Schlüssels schon abgelaufen? Diese Fragen lassen sich beantworten, wenn auch der öffentliche Schlüssel der CA in ein digitales Zertifikat gefasst wird. Dies ist auch eine durchaus übliche Vorgehensweise, wie wir noch sehen werden (man spricht dann von einem **CA-Zertifikat**). Allerdings verschieben sich die genannten Probleme dadurch nur auf diejenige CA, die das CA-Zertifikat signiert hat.

Es bleibt daher nichts anderes übrig, als den öffentlichen Schlüssel einer CA Alice und Bob persönlich oder mit geeigneten Sicherheitsmaßnahmen über das

Netz zu überreichen. So ist es beispielsweise üblich, dass ein Unternehmen bei der Verteilung von Software auf die PCs der Anwender den öffentlichen CA-Schlüssel gleich mitliefert. Falls Alice befürchtet, dass Mallory diesen Schlüssel ausgetauscht hat, kann sie einen Hashwert davon (theoretisch auch den ganzen Schlüssel) am Telefon mit einem CA-Mitarbeiter abgleichen. Manche Unternehmen veröffentlichen einen Hashwert des CA-Schlüssels auch im Intranet oder am Schwarzen Brett. Gäbe es keine CA, dann müsste Alice den Schlüssel-Check mit allen ihren Kommunikationspartnern durchführen. Ist dagegen eine CA im Einsatz, dann ist dies nur einmal notwendig.

Nachdem Sie nun wissen, was ein digitales Zertifikat und was eine CA ist, kann ich Ihnen nun auch eine genauere Definition des Begriffs Public-Key-Infrastruktur (PKI) geben. Als PKI bezeichnet man die Gesamtheit der Hardware und Software, die man benötigt, um digitale Zertifikate einzusetzen und eine CA zu betreiben. Welche Bestandteile eine PKI außer der CA im Einzelnen hat, werde ich in Abschnitt 26.5.1 beschreiben.

Name des Inhabers: Alice Onliner
CA: Zertifizierungsinstanz des Staates Kryptoland
Öffentlicher Schlüssel der CA: FA 4E 9E F0 DD 9E ED F2 DD 23 A2 8A 4E 23 E3 D4
Öffentlicher Schlüssel des Inhabers: F2 D2 0E ED FA 4E 9E 0A F2 DD 23 8A 32 44 F3 E9
Seriennummer der Zertifikats: 8549285792859
Gültigkeitszeitraum: 2007-07-01 – 2011-06-30
Signatur der CA: DD 9E EF 2D D2 38 A3 2D FA 4E 9E DA 49 47 F0 22

Abb. 26–1 *Beispiel für ein digitales Zertifikat. Darin sind unter anderem der Name des Inhabers und dessen öffentlicher Schlüssel enthalten.*

26.3 Vertrauensmodelle

Bisher sind wir davon ausgegangen, dass Alices und Bobs digitale Zertifikate von einer CA ausgestellt werden. Dies ist zwar der Normalfall, jedoch keineswegs selbstverständlich. Wenn wir uns fragen, wer ein Zertifikat signiert, dann hat dies etwas damit zu tun, wem Alice und Bob vertrauen. Man spricht deshalb von **Vertrauensmodellen**. Um uns das Thema Vertrauensmodelle näher anzuschauen, gehen wir im Folgenden davon aus, dass Alice eine Mail an Bob (diesen kennt sie

bekanntlich recht gut und vertraut ihm) oder an Zacharias (diesen kennt sie nicht und vertraut ihm daher auch nicht) schickt. Dabei verwendet Alice Bobs bzw. Zacharias' öffentlichen Schlüssel. Sie weiß jedoch nicht, ob der jeweilige Schlüssel echt ist. Wenn es um Verbindlichkeit geht, nehmen wir an, Bob signiert eine Nachricht an Alice mit seinem privaten Schlüssel.

An dieser Stellle stellt sich die Frage, ob man im Zusammenhang mit digitalen Zertifikaten ohne CA überhaupt noch von einer PKI reden kann. Dazu gibt es unterschiedliche Ansichten. In diesem Buch habe ich mich dafür entschieden, nur dann von einer PKI zu sprechen, wenn eine CA als Zertifikatsaussteller dient. Von den folgenden drei Untereln betrifft daher nur eines das Thema PKI direkt, während die anderen beiden Alternativen zu einer PKI aufzeigen.

26.3.1 Direct Trust

Das einfachste Vertrauensmodell (es wird **Direct Trust** genannt) sieht vor, dass Bob selbst die Authentizität seines öffentlichen Schlüssels gegenüber Alice bestätigt. Er kann sich dazu selbst ein Zertifikat ausstellen (viele Anwendungsprogramme unterstützen dies); er kann dies jedoch auch lassen, da ein selbst signiertes Zertifikat keinen großen Wert hat. In jedem Fall muss Bob den öffentlichen Schlüssel (gegebenenfalls inklusive Zertifikat) Alice zukommen lassen, beispielsweise per E-Mail oder auf CD-ROM. Besteht die Gefahr einer Man-in-the-Middle-Attacke durch Mallory, dann kann er anschließend zur Kontrolle mit Alice einen kryptografischen Hashwert des Schlüssels am Telefon vergleichen. Die Infrastruktur, die für Direct Trust benötigt wird, ist vernachlässigbar. Direct Trust ist so gesehen die Nulllösung in Sachen digitale Zertifikate. Schauen wir uns nun einmal an, wie sich die vier eingangs beschriebenen Probleme mit Direct Trust lösen lassen:

- *Authentizität der Schlüssel*: Diese lässt sich mit Direct Trust im kleinen Kreis durchaus gewährleisten. Bei größeren Anwendergruppen wird es dagegen schnell unübersichtlich.
- *Sperrung von Schlüsseln*: Die Sperrung eines Schlüssels lässt sich mit Direct Trust wie folgt realisieren: Wenn Bob nicht mehr will, dass Alice seinen Schlüssel einsetzt, dann teilt er ihr das persönlich mit. Wenn Bob viele Kommunikationspartner hat, dann wird das Sperren auf diese Weise etwas mühsam.
- *Verbindlichkeit*: Diese ist bei Direct Trust nur sehr schwer zu erreichen. Bob kann gegenüber Alice jederzeit abstreiten, dass es sich bei einem bestimmten öffentlichen Schlüssel um seinen eigenen handelt.
- *Durchsetzen einer Policy*: Das Durchsetzen einer Policy ist bei Direct Trust schwierig bis unmöglich.

Direct Trust ist das einfachste Vertrauensmodell. Da Alice und Bob sich kennen, funktioniert es zwischen den beiden recht gut. Wenn Alice allerdings eine E-Mail

an Zacharias (diesen kennt sie nur vom Namen her) senden will, dann ist die Sache etwas umständlicher. Wenn Alice nicht nur mit Zacharias, sondern auch mit vielen anderen Leuten verschlüsselte E-Mails austauschen will, dann wird die Variante Direct Trust endgültig zu einer mühsamen Angelegenheit. Direct Trust ist somit nur für weniger anspruchsvolle Anwendungen geeignet.

26.3.2 Web of Trust

Alice will eine Mail an Zacharias senden, ohne das umständliche Direct Trust zu verwenden. Zufälligerweise kennt Bob Zacharias' öffentlichen Schlüssel und hat ihn per Direct Trust überprüft. Da Alice Bobs öffentlichen Schlüssel kennt (ebenfalls durch Direct Trust), bietet sich folgende Lösung an: Bob signiert Zacharias' öffentlichen Schlüssel (stellt also ein digitales Zertifikat aus), bevor er ihn an Alice schickt. Alice prüft dann Bobs Signatur und kann somit auf die Echtheit von Zacharias' Schlüssel schließen. Wenn nun Zacharias selbst den Schlüssel seines Freundes Zeus zertifiziert, dann kann Alice daraus auf die Echtheit von Zeus' Schlüssel schließen. Auf diese Weise lassen sich beliebige Vertrauensketten bilden, die zusammengenommen ein Netz von Zertifikaten ergeben (Abbildung 26–2). Dieser Ansatz wird **Web of Trust** genannt.

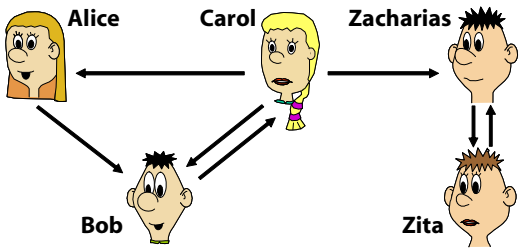


Abb. 26–2 *In einem Web of Trust kann jeder für jeden Zertifikate ausstellen.*

Auch bei einem Web of Trust ist die benötigte zusätzliche Infrastruktur nicht besonders umfangreich. Es bietet sich jedoch an, einen Server einzurichten, auf dem digitale Zertifikate abrufbar sind. Wie lassen sich nun die vier erwähnten Probleme mit einem Web of Trust lösen?

- *Authentizität der Schlüssel*: Diese lässt sich gut überprüfen, wenn Alice einen oder mehrere Bekannte hat, die ein bestimmtes Zertifikat signiert haben. In größeren Nutzergruppen funktioniert dies jedoch häufig nicht.
- *Sperrung von Schlüsseln*: Diese ist in einem Web of Trust noch mühsamer als beim Direct Trust. Wenn Alice Ihren öffentlichen Schlüssel sperren lassen will, dann muss sie es allen mitteilen, die ihren Schlüssel haben könnten. Dies können jedoch sehr viele sein, und so werden viele Anwender Alices gesperrten öffentlichen Schlüssel weiterhin verwenden.

- *Verbindlichkeit*: Diese ist bei einem Web of Trust etwas besser zu bewerkstelligen als beim Direct Trust, da es durch die Zertifikate ja jeweils Bürgen gibt, die die Zugehörigkeit eines Schlüssels zu einer Person belegen. Juristisch gesehen ist dies jedoch oft nicht ausreichend.
- *Durchsetzung einer Policy*: Dieses Ziel ist in einem Web of Trust schwer zu erreichen.

Ein Web of Trust ist somit etwas leistungsfähiger als Direct Trust. Leider funktioniert es in der Praxis meist nicht so recht. Jedenfalls nicht im Internet, für das es eigentlich gedacht war. Das Internet mit seinen mehreren Hundert Millionen Anwendern ist schlichtweg zu groß, um ein Web of Trust aufzuspannen, in dem jeder das Zertifikat eines jeden anderen über eine Vertrauenskette überprüfen kann. Will Alice eine Nachricht an den ihr unbekannten Zacharias schicken, dann muss sie womöglich eine Kette über zig Stationen finden, um dessen Zertifikat zu überprüfen. Wenn nur ein Glied der Kette nicht vertrauenswürdig ist, dann ist sie wertlos. Alice muss also nach Möglichkeit mehrere Vertrauensketten suchen, um durch eine mehrfache Kontrolle wirklich sicher sein zu können. Dass das Web of Trust trotz der Nachteile recht populär ist, liegt wie so oft am Sieg der Pragmatik: Es ist einfacher als eine PKI.

26.3.3 Hierarchical Trust

Kommen wir nun zurück zur Ausgangssituation und nehmen an, dass alle digitalen Zertifikate von einer CA ausgestellt werden. Wir haben es also mit einer PKI zu tun. Das Vertrauensmodell, das diese Situation bezeichnet, heißt **Hierarchical Trust**. Damit Alice die Signatur der CA auf Bobs digitalem Zertifikat verifizieren kann, muss diese ihren öffentlichen Schlüssel bekannt machen. Alice muss sich wiederum den öffentlichen Schlüssel der CA besorgen und sich vergewissern, dass es der echte ist. Dies verursacht einen gewissen Aufwand, was jedoch keine große Rolle spielt, denn Alice muss dies nur einmal tun. Sobald sie den Schlüssel der CA hat, kann sie alle digitalen Zertifikate verifizieren, die von dieser signiert wurden.

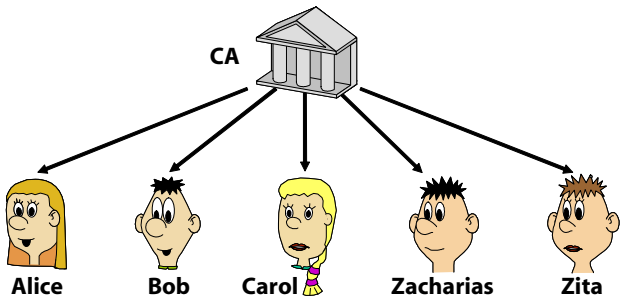


Abb. 26–3 *Beim Hierarchical Trust werden Zertifikate von einer Zertifizierungsinstanz (CA) ausgestellt.*

Zum Betrieb einer CA ist einiges an Aufwand notwendig. So muss die CA von einer vertrauenswürdigen Stelle betrieben werden. Alice muss die Gelegenheit haben, sich bei der CA anzumelden. Die CA muss die erstellten Zertifikate zugänglich machen. Der Begiff »Public-Key-Infrastruktur« kommt also nicht von ungefähr, denn Hierarchical Trust erfordert tatsächlich eine Infrastruktur. Betrachten wir nun noch die Frage, wie sich die vier beschriebenen Probleme der Public-Key-Verfahren mit dem Vertrauensmodell Hierarchical Trust lösen lassen:

- *Authentizität der Schlüssel*: Die Zuordnung zwischen Schlüssel und Besitzer funktioniert in einer PKI sehr gut.
- *Sperrung von Schlüsseln*: Die Sperrung eines öffentlichen Schlüssels lässt sich mit einer CA sehr gut realisieren. Es gibt verschiedene Ansätze dafür, auf die ich in Abschnitt 28.3 näher eingehen werde.
- *Verbindlichkeit*: Von einer CA ausgestellte digitale Zertifikate bieten ein hohes Maß an Verbindlichkeit.
- *Durchsetzen einer Policy*: Auch das Durchsetzen einer Policy ist bei Hierarchical Trust erheblich einfacher als beim Direct Trust oder in einem Web of Trust. Schlüsselwechsel, Schlüssellängen, Sperrungen und vieles mehr können von der CA gesteuert und überwacht werden.

Hierarchical Trust ist damit das komplexeste, aber auch das leistungsfähigste Vertrauensmodell. Wie bereits erwähnt, ist es das Vertrauensmodell, das jeder PKI zugrunde liegt. Fast alle folgenden Überlegungen beziehen sich daher auf Hierarchical Trust.

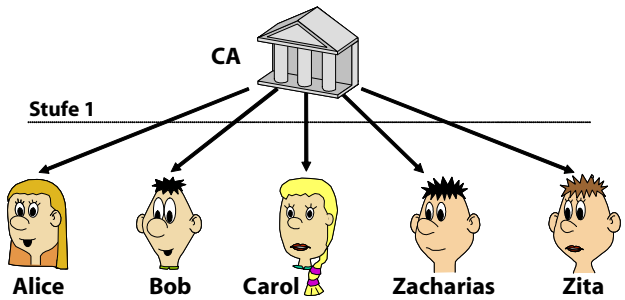


Abb. 26–4 *Eine einstufige Hierarchie ist der einfachste Fall einer Hierarchie, dem das Vertrauensmodell Hierarchical Trust zugrunde liegt.*

26.3.4 PKI-Varianten

Das Vertrauensmodell Hierarchical Trust, das jeder PKI zugrunde liegt, lässt mehrere Varianten zu. Die fünf wichtigsten davon schauen wir uns nun an.

Ein-Stufen-Hierarchie

Die einfachste Realisierungsform einer PKI besteht darin, dass es eine CA gibt, die für alle Anwender digitale Zertifikate ausstellt. Man spricht in diesem Fall von einer **Ein-Stufen-Hierarchie**. Die einzige vorhandene CA bildet die einzige Hierarchiestufe. Anwenderin Alice benötigt in diesem Fall nur einen öffentlichen CA-Schlüssel. Mit diesem kann sie alle Zertifikate verifizieren.

Webmodell

In der Praxis gibt es oftmals mehrere konkurrierende Betreiber von CAs, die um die Gunst der Benutzer buhlen. Dies hat zur Folge, dass Zertifikate von unterschiedlichen CAs im Umlauf sind. Alice benötigt daher eine ganze Liste von CA-Schlüsseln. Wenn sie ein Zertifikat von Bob verifizieren will, dann stellt sie zunächst fest, welche Zertifizierungsstelle dieses ausgestellt hat. Danach kann sie mit dem jeweiligen Schlüssel die Signatur auf dem Zertifikat überprüfen (Abbildung 26–5).

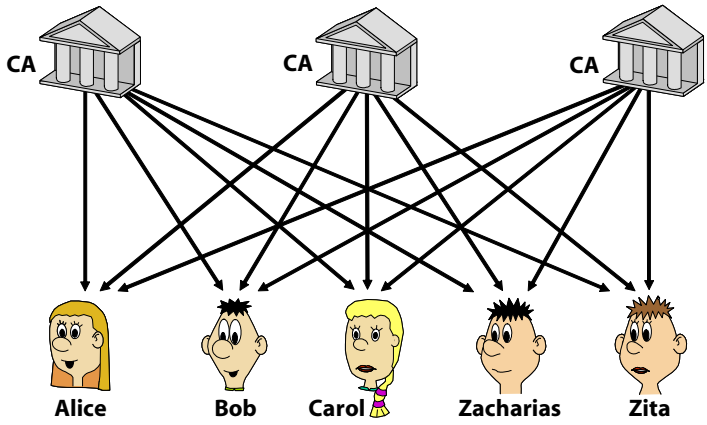


Abb. 26–5 *Im Webmodell gibt es mehrere CAs, die Zertifikate ausstellen. Ein Benutzer kann dabei mehrere Zertifikate von unterschiedlichen CAs besitzen.*

Diese Form des Hierarchical Trust, bei dem es viele voneinander unabhängige CAs gibt, wird meist als **Webmodell** bezeichnet. Der Name rührt daher, dass die beschriebene Situation im World Wide Web musterhaft gegeben ist: Es gibt sehr viele voneinander unabhängige CAs, die Zertifikate für das World Wide Web anbieten. Die gängigen Webbrowser haben daher bereits bei Auslieferung mehrere Hundert CA-Zertifikate gespeichert. Der Nachteil beim Webmodell ist, dass sich hier eine Policy nur schlecht durchsetzen lässt. Wenn Bob die Policy einer CA nicht akzeptieren will, dann holt er sich sein Zertifikat eben bei einer anderen. Da Alice mehrere CAs anerkennt, macht es für sie keinen Unterschied, welche davon Bobs Zertifikat ausgestellt hat.

Cross-Zertifizierung

Das Webmodell wird vor allem dann verwendet, wenn unterschiedliche CAs miteinander konkurrieren. Häufig gibt es jedoch auch die Situation, dass mehrere CAs vorhanden sind, deren Benutzerkreis sich nicht überschneidet. In diesem Fall bietet sich eine **Cross-Zertifizierung** an. Eine solche sieht vor, dass zwei CAs sich gegenseitig je ein Zertifikat ausstellen. CA_1 stellt also ein Zertifikat für CA_2 aus und beglaubigt damit den öffentlichen Schlüssel von CA_2. CA_2 macht dasselbe mit dem Schlüssel von CA_1.

Alice, die Kundin bei CA_1 ist, kennt den öffentlichen Schlüssel von CA_1 und sonst keinen öffentlichen CA-Schlüssel. Carol, Kundin bei CA_2, kennt den öffentlichen Schlüssel von CA_2 und sonst keinen. Wenn sich Alice Carols Zertifikat besorgt, dann holt sie sich das Zertifikat von CA_2 gleich mit. Anschließend verifiziert sie zunächst das Zertifikat von CA_2 mit dem öffentlichen Schlüssel von CA_1. Danach verifiziert sie Carols Zertifikat mit dem öffentlichen Schlüssel von CA_2. Durch eine Cross-Zertifizierung wird somit der Einzugsbereich einer CA auf den einer anderen ausgedehnt. Von Nachteil ist, dass mit einem Zertifikat immer gleich das Zertifikat der entsprechenden CA mitgeliefert werden muss.

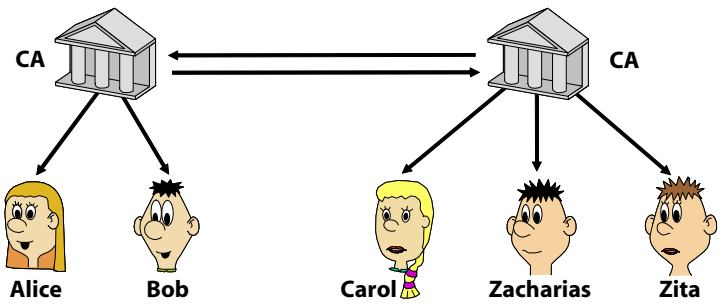


Abb. 26–6 *Bei einer Cross-Zertifizierung stellen sich zwei CAs gegenseitig je ein Zertifikat aus.*

Mehrstufige Hierarchie

Wenn neben CA_1 und CA_2 noch weitere CAs aktiv sind, die unterschiedliche Benutzer bedienen, dann wird eine gegenseitige Cross-Zertifizierung oft zu aufwendig. Stattdessen sollte in einem solchen Fall eine übergeordnete CA eingerichtet werden, die Zertifikate für die anderen ausstellt. Wenn es mehrere übergeordnete CAs gibt, dann können auch diese wiederum Zertifikate von einer weiteren übergeordneten CA erhalten. Man spricht in diesem Fall von einer **mehrstufigen Hierarchie**. Die CA an der Spitze der Hierarchie wird als **Wurzel-CA** bezeichnet.

Für Anwenderin Alice genügt es in diesem Fall, den öffentlichen Schlüssel der Wurzel-CA zu kennen. Mit diesem kann sie die Zertifikate der darunter liegenden CAs verifizieren und sich so die Hierarchie bis zu Benutzer Bob hinunterhangeln. Auch in diesem Fall müssen die Zertifikate der CAs, die in der Hierarchie zwi-

schen dem Benutzer und der Wurzel-CA stehen, mit einem Zertifikat gleich mitgeliefert werden.

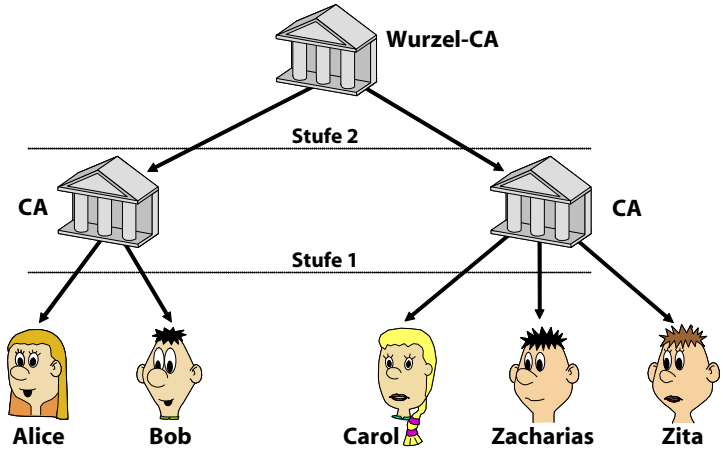


Abb. 26–7 *Beispiel für eine zweistufige CA-Hierarchie*

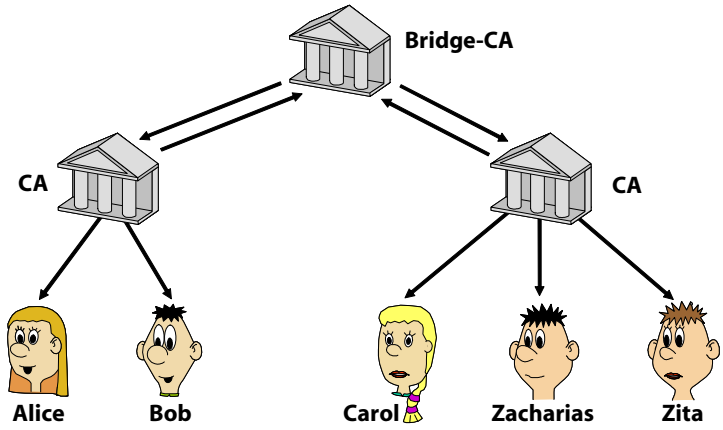


Abb. 26–8 *Eine Bridge-CA verbindet mehrere CAs, indem sie Cross-Zertifizierungen mit ihnen eingeht.*

Bridge-CA

Im richtigen Leben kommt es oft vor, dass sich eine CA nicht von einer übergeordneten CA zertifizieren lassen will. Dies hat meist Prestigegründe. Vor allem namhafte Unternehmen fürchten oft einen Imageschaden, wenn sich ihre firmeninterne CA – wenn auch nur formal – einer anderen CA unterordnen muss. Viele CA-Betreiber vermeiden daher eine Eingliederung in eine mehrstufige Hierarchie und bevorzugen stattdessen eine Cross-Zertifizierung mit einer **Bridge-CA**. Eine solche hat die Aufgabe, mit möglichst vielen CAs eine Cross-Zertifizie-

rung einzugehen.Ist eine Bridge-CA involviert, dann genügt es Alice, wenn sie den öffentlichen Schlüssel ihrer CA kennt. Mit diesem kann sie das Zertifikat der Bridge-CA überprüfen, damit dann ein anderes CA-Zertifikat und damit schließlich Bobs Zertifikat. Diese Vorgehensweise erscheint etwas umständlich, zumal eine Bridge-CA gegenüber einer Hierarchie keine unmittelbaren Vorteile hat. Dennoch ist das Bridge-CA-Modell in der Praxis recht beliebt – aus den besagten Prestigegründen.

26.4 PKI-Standards

Es versteht sich von selbst, dass es innerhalb einer PKI einen großen Bedarf für Standards gibt. Schließlich müssen Schlüssel, Zertifikate und andere Daten von unterschiedlichen PKI-Beteiligten verarbeitet werden. Eine wichtige Rolle spielen hierbei die bereits erwähnten PKCS-Standards, die zahlreiche PKI-relevante Formate beschreiben. Die bedeutendsten weiteren PKI-Standards stelle ich Ihnen nun vor.

26.4.1 X.509

Der älteste relevante Standard der PKI-Welt ist unter dem Namen **X.509** bekannt [X.509]. Der vollständige Name lautet *ITU-T X.509*. So umständlich wie die Namensgebung ist auch der Standard selbst. Vieles, was darin enthalten ist, ist nur noch historisch interessant. Die heute noch relevanten Teile sind nicht immer optimal gelöst. Diese Mängel sind vor allem darauf zurückzuführen, dass X.509 zu einer Zeit entstand, als der Aufbau von Public-Key-Infrastrukturen noch Zukunftsmusik war (die erste X.509-Version erschien 1988). Auch Überarbeitungen, die 1993 und 1997 erschienen, konnten die vorhandenen Probleme nur teilweise beheben.

X.509 ist ein Bestandteil des X.500-Standards für Verzeichnisdienste. Es handelt sich dabei eigentlich um einen Authentifizierungsstandard für Kommunikationsnetze, der aus einer Zusammenarbeit der ISO und der ITU-T (damals noch CCITT) entstanden ist [X.509]. Der Standard hat drei Bestandteile. Teil 1 und 2 – darin werden Authentifizierungsprotokolle spezifiziert – haben keine praktische Relevanz. Interessant ist dagegen der dritte Teil von X.509, der Formate für digitale Zertifikate und Sperrlisten beschreibt. Diese Formate spielen trotz einiger Mängel eine wichtige Rolle im PKI-Bereich. Wenn im Folgenden von X.509 die Rede ist, dann ist immer der dritte X.509-Teil gemeint.

26.4.2 PKIX

Der wohl wichtigste PKI-Standard trägt den Namen *Public Key Infrastructure, X.509*. Besser bekannt ist er unter der Abkürzung **PKIX**. PKIX ist ein Werk des Internet-Standardisierungsgremiums IETF und standardisiert so ziemlich alles, was beim Aufbau einer PKI von Belang ist. Bisher sind etwa 60 RFCs aus der PKIX-Arbeitsgruppe der IETF hervorgegangen, weitere werden folgen. Das PKIX zugrunde liegende Vertrauensmodell ist Hierarchical Trust. Wie das »X« im Namen andeutet, verwendet PKIX das Zertifikatsformat und die Sperrlisten des X.509-Standards. Darüber hinaus enthält PKIX zahlreiche weitere Formate, Protokolle und Vorgehensweisen.

26.4.3 Common PKI

Vom deutschen Signaturgesetz wird in Abschnitt 30.1 noch die Rede sein. Zunächst interessiert uns nur die Tatsache, dass dieses Signaturgesetz und die dazu gehörenden Bestimmungen zahlreiche Anforderungen an eine PKI enthalten. Wenn eine PKI die jeweiligen Bestimmungen erfüllt, dann hat eine digitale Signatur eine höhere juristische Beweiskraft. Da die Anforderungen aus dem Signaturgesetz sehr hoch sind, reichen zahlreiche Formate, Abläufe und Protokolle aus dem PKIX-Standard dafür nicht aus. Um diesem Missstand zu begegnen, entwickelten verschiedene deutsche Unternehmen zusammen mit dem BSI und dem Branchenverband Teletrust einen eigenen Standard, der als **Common PKI** bezeichnet wird (der alte Name lautete ISIS-MTT) [Common]. Common PKI standardisiert alle Teilbereiche einer PKI, die für das Signaturgesetz von Bedeutung sind. Der Standard verwendet zahlreiche PKIX-Bestandteile, weicht jedoch an einigen Stellen davon ab.

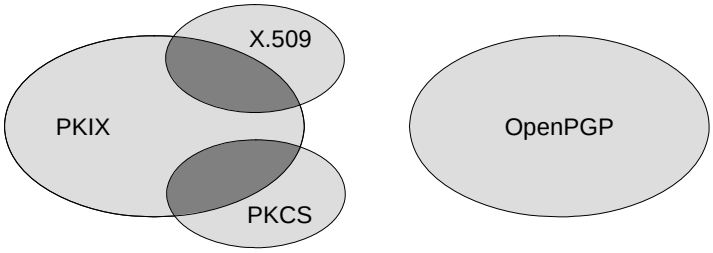


Abb. 26–9 *X.509 sowie die damit verwandten Standards PKIX und PKCS sind nicht mit OpenPGP kompatibel.*

26.4.4 OpenPGP

Die bisher betrachteten PKI-Standards X.509, PKIX und ISIS-MTT hängen eng miteinander zusammen. Insbesondere verwenden alle drei digitale Zertifikate nach dem X.509-Format. Es gibt jedoch auch einen PKI-Gesamtstandard, der eine komplett eigene Welt bildet und keine X.509-Zertifikate vorsieht: **OpenPGP** [RFC2440]. Dabei handelt es sich um einen Standard, der aus der Software PGP (Pretty Good Privacy) entstanden ist, von der in diesem Buch noch die Rede sein wird. PGP war ursprünglich eine proprietäre Krypto-Software, die hauptsächlich für E-Mails gedacht war. PGP sah digitale Zertifikate vor und unterstützte das Vertrauensmodell Web of Trust (dieses ist im Grunde eine Erfindung von PGP-Entwickler Phil Zimmermann). OpenPGP entstand 1997 mit dem Ziel, aus den von PGP definierten Formaten einen Standard für die Nutzung digitaler Zertifikate aufzubauen. In bester PGP-Tradition unterstützt OpenPGP das Vertrauensmodell Web of Trust, ist jedoch auch für Hierarchical Trust und damit zum Aufbau einer PKI nutzbar. OpenPGP hat eine große Anhängerschaft und wird daher unter Privatanwendern weiterhin populär bleiben. Im kommerziellen Bereich hat sich allerdings PKIX durchgesetzt.

26.5 Aufbau und Funktionsweise einer PKI

Wenn man den Aufbau und die Funktionsweise eines Computersystems betrachtet, dann bietet es sich meist an, dies in drei Schritten zu tun: erst die Komponenten, dann die Rollen und schließlich die Prozesse. Auf diese Weise werden wir uns in diesem Kapitel auch den Aufbau und die Funktionsweise einer PKI näher anschauen.

26.5.1 Komponenten einer PKI

Beim Blick auf die verschiedenen Komponenten einer PKI lassen wir der Einfachheit halber mehrstufige CA-Hierarchien außen vor – es gibt also nur eine CA. Unter dieser Voraussetzung können wir eine PKI als Client-Server-System betrachten und deren Komponenten in zwei Bereiche aufteilen: Server-Komponenten, die beim PKI-Betreiber zentral betrieben werden, und dezentrale Client-Komponenten. Die zentralen Komponenten einer PKI werden manchmal unter dem Begriff **Trust Center** zusammengefasst. Diese Bezeichnung ist vor allem dann üblich, wenn die Server-Komponenten der PKI durch bauliche Maßnahmen (dicke Wände, Zutrittskontrolle, Alarmanlage usw.) geschützt sind. Diese Voraussetzungen sind in der Praxis vor allem bei CA-Betreibern gegeben, die ihre Dienste am Markt anbieten. Mit einem Trust Center ist daher oft auch ein Anbieter von CA-Dienstleistungen gemeint. Übrigens ist »Trust Center« ein Pseudoanglizismus (also ein englisch klingendes Wort, das im Englischen ungebräuchlich

ist). Statt Trust Center verwenden Briten und Amerikaner meist nur den Ausdruck *Certification Authority* (CA), wobei oft nicht klar ist, ob damit nun ein ganzes Trust Center oder nur eine CA gemeint ist. So gesehen ist die deutsche Verwendungsweise durchaus sinnvoll.

Zertifizierungsstelle (CA)

Eine CA hat (wie bereits erwähnt) die Aufgabe, digitale Zertifikate zu generieren. Darüber hinaus kann eine CA auch Sperrlisten erstellen (siehe Abschnitt 28.3.1). Zu den Hilfsmitteln, die eine CA hierbei benötigt, gehört ein privater Schlüssel zum Signieren der Zertifikate und Sperrlisten. Darüber hinaus benötigt eine CA eine Datenbank zum Speichern der Anwenderdaten. Einige CA-Lösungen verwenden als Datenbank den Verzeichnisdienst, der auch als Zertifikateserver genutzt wird, in den meisten Fällen nimmt der PKI-Betreiber jedoch aus Sicherheitsgründen eine Trennung vor.

Es versteht sich von selbst, dass eine CA eine äußerst sicherheitskritische Komponente ist. Wenn Mallory es schafft, an den privaten Schlüssel der CA heranzukommen und damit Zertifikate zu fälschen, dann ist dies das Schlimmste, was innerhalb einer PKI passieren kann. Die CA sollte daher in einer sicheren Umgebung betrieben werden, wobei insbesondere der private CA-Schlüssel geschützt werden muss.

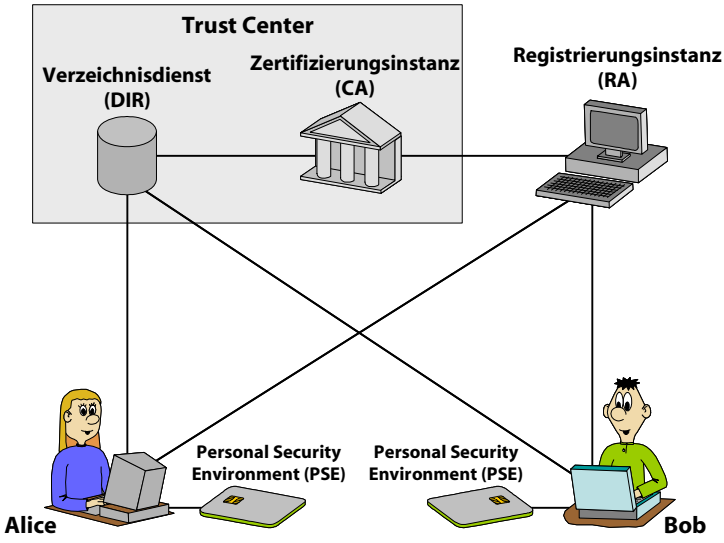


Abb. 26–10 *Aufbau einer PKI: Die zentralen Komponenten werden im Trust Center zusammengefasst. Die PCs von Alice und Bob sind Beispiele für Endeinheiten. Die Speicherung der privaten Schlüssel erfolgt im PSE, das etwa als Smartcard realisiert sein kann.*

Hardware-Security-Modul (HSM)

Was ein Hardware-Security-Modul (HSM) ist, habe ich bereits in Abschnitt 23.3 erklärt. Es handelt sich dabei um ein Hardwaremodul, in dem ein privater Schlüssel gespeichert ist und das eigenständig kryptografische Operationen durchführen kann. Die Speicherung eines CA-Schlüssels ist ein typischer und sinnvoller Anwendungsbereich für ein HSM. Die Vorteile des HSM-Einsatzes liegen auf der Hand: Der private CA-Schlüssel ist in einer sicheren Umgebung gespeichert und gelangt nicht in den Arbeitsspeicher oder auf die Festplatte des CA-Rechners. Insbesondere gelangt der private CA-Schlüssel nicht (wenn auch nur verschlüsselt) auf Backup-Bänder. Dies ist wichtig, da Backup-Dateien oftmals einen Angriffspunkt liefern – etwa dann, wenn Mallory an das entsprechende Band herankommt und den geheimen Schlüssel, mit dem der CA-Schlüssel verschlüsselt wurde, per Wörterbuch-Angriff ermitteln kann. Die Vorteile eines HSM gegenüber einer Softwarelösung sind damit klar. Warum aber sollte eine CA nicht wie die Anwender eine Smartcard für ihren Schlüssel einsetzen? Auch dafür gibt es einige Gründe:

- Ein HSM ist deutlich performanter als eine Smartcard.
- Gängige HSMs bieten ein leistungsfähigeres Rollenmodell. So kann der PKI-Betreiber für wichtige Aktionen (etwa das Generieren des privaten CA-Schlüssels) eine entsprechende Rolle einrichten, während Routineprozesse von einer weniger privilegierten Person erledigt werden. Auch ein Vier-Augen-Prinzip gehört zur Standardfunktionalität bei HSM-Produkten.
- Gängige HSMs bieten sichere Backup-Möglichkeiten. So lassen sich viele HSMs bei Eingabe des richtigen Passworts klonen (man benötigt dazu ein zweites HSM). Den Klon kann der Betrieber dann in den Tresor legen und bei Bedarf zum Einsatz bringen. Eine Smartcard bietet dagegen meist keine sichere Backup-Funktion – der Schlüssel muss also von Hand aufgebracht und zusätzlich gesichert werden.
- Ein gutes HSM ist durch seine Bauweise deutlich besser vor physikalischen Angriffen geschützt als eine Smartcard.

Es lohnt sich also, den CA-Schlüssel in ein HSM auszulagern. Wer seine CA in eine mehrstufige PKI-Hierarchie integrieren will (zum Beispiel in die Verwaltungs-PKI oder in eine Signaturgesetz-PKI) muss dies meist sogar tun, weil der Hierarchiebetreiber dies fordert.

Zertifikateserver

Wenn eine CA ein digitales Zertifikat für Alice generiert hat, dann muss sie dieses anderen PKI-Anwendern zugänglich machen. Aus diesem Grund gehört ein **Zertifikateserver** (Certificate Repository) zu den zentralen PKI-Komponenten. Der Zertifikateserver hält die von der CA ausgestellten Zertifikate und Sperrlisten

zum Abruf bereit und sollte für alle PKI-Anwender über eine Netzwerkverbindung zugänglich sein. Der Zertifikateserver ist in der Praxis meist ein Verzeichnisdienst, der nicht nur für die PKI genutzt wird. Wenn Bob Alices Zertifikat vom Zertifikateserver herunterladen will, dann verwendet er dazu in der Regel das *Lightweight Directory Access Protocol* (*LDAP*). Baut ein Unternehmen eine PKI auf, dann ist der zu nutzende Verzeichnisdienst meist schon vorhanden (beispielsweise als Anmeldeserver oder als Meta-Directory). Die zusätzliche Verwendung als Zertifikateserver ist meist unkritisch. Allerdings kommt es vor, dass ein bisher nur intern genutzter Verzeichnisdienst Zertifikate auch für Anwender außerhalb des Unternehmens zur Verfügung stellen muss. Eine dadurch notwendige Öffnung des Verzeichnisdiensts sollte in der Planungsphase möglichst früh berücksichtigt werden.

Zeitstempeldienst (TSA)

Eine weitere zentrale Komponente mancher PKIs ist ein **Zeitstempeldienst**. Ein solcher wird auch als **TSA** (Time Stamp Authority) oder **TSS** (Time Stamp Service) bezeichnet. Wozu ein Zeitstempeldienst benötigt wird, sehen wir am folgenden Betrugsszenario, das auch dann funktioniert, wenn Alices öffentlicher Signaturschlüssel von einer absolut vertrauenswürdigen CA zertifiziert ist:

1. Alice signiert ein Dokument.
2. Alice lässt ihr digitales Zertifikat sperren. Als Grund dafür gibt sie an, dass ihr privater Schlüssel gestohlen wurde.
3. Alice behauptet, das besagte Dokument sei nicht von ihr signiert worden. Stattdessen habe Mallory mit dem gestohlenen Schlüssel die Signatur angefertigt.

Auf ähnliche Weise kann Alice auch Signaturen abstreiten, wenn Mallory den CA-Schlüssel gestohlen hat oder ein neuer Angriff auf das verwendete Signaturverfahren bekannt geworden ist. In allen Fällen lautet die Frage, wann das Dokument signiert worden ist – vor oder nach der Sperrung, vor oder nach dem Diebstahl, vor oder nach Bekanntwerden des Angriffs. Das Problem ist, dass man einer Signatur nicht ansieht, wann sie angefertigt wurde.

Als Maßnahme gegen solche Angriffe kann man eine vertrauenswürdige Instanz einrichten, die eingereichte Dokumente zusammen mit dem aktuellen Datum und der Uhrzeit (einem sogenannten **Zeitstempel**) signiert. Eine solche Instanz wird Zeitstempeldienst genannt. Alice kann den Zeitstempeldienst nutzen, um ihre Signatur mit einer zuverlässigen Zeitangabe versehen zu lassen. Alices Geschäftspartner kann veranlassen, dass eine von ihr geleistete Signatur zeitgestempelt wird. Sollte Alices Zertifikat irgendwann gesperrt werden, dann kann Alice ihre Signatur nicht Mallory in die Schuhe schieben.

Ein Zeitstempeldienst wird teilweise von einem Trust Center als zusätzlicher Dienst angeboten. Neben der genannten Anwendung im Zusammenhang mit

digitalen Signaturen sind Zeitstempeldienste auch dann nützlich, wenn es um die Urheberschaft von Dokumenten geht. Wenn Bob beispielsweise befürchtet, dass Mallory sich als Urheber seines neuen Romans ausgeben könnte, dann versieht er ihn vor der Veröffentlichung mit einem Zeitstempel. Bob kann damit beweisen, dass der Roman entstanden ist, bevor Mallory sich als Urheber ausgegeben hat.

Ein Zeitstempeldienst benötigt ein Schlüsselpaar und ein digitales Zertifikat. Wird ein Zeitstempeldienst als zusätzliche Komponente eines Trust Centers betrieben, dann wird dieses Zertifikat meist von der CA dieses Trust Centers ausgestellt. Zur Kommunikation zwischen Alice und dem Zeitstempeldienst gibt es ein Netzwerkprotokoll namens **TSP** (Time Stamp Protocol), das in [RFC3161] beschrieben wird. Es wurde von der PKIX-Arbeitsgruppe entwickelt. TSP ist ein recht einfaches Protokoll, das eine Anfrage von Alice und eine Antwort des Zeitstempeldiensts vorsieht. Die jeweiligen Protokollnachrichten können per E-Mail, HTTP, FTP oder direkt über TCP übertragen werden.

Registrierungsstelle (RA)

Die **Registrierungsstelle** (**Registration Authority**, **RA**) ist die Anmeldestelle, über die Alice ihr Zertifikat beantragt. In einem Unternehmen ist die RA meist im Büro eines Administrators lokalisiert, während die Betreiber kommerzieller Trust Center oft die Post oder ein eigenes Filialnetz nutzen. Meist kommen auf eine CA mehrere RAs. Allerdings muss eine CA nicht zwingend eine RA betreiben, da Alice zur Zertifikate-Generierung auch direkt mit der CA kommunizieren kann. Ob es in einer PKI eine RA gibt und wie diese realisiert ist, hängt stark vom Enrollment-Prozess ab (Abschnitt 28.1).

Sperrstelle

Die **Sperrstelle** ist eine weitere mögliche Komponente innerhalb einer PKI. Die Sperrstelle ist die Anlaufstelle für Alice, wenn sie ihr Zertifikat sperren lassen will (etwa weil sie ihre Smartcard verloren hat). Viele PKI-Betreiber realisieren die Sperrstelle mithilfe einer bereits bestehenden Telefon-Hotline, deren Telefonnummer den Benutzern ohnehin bekannt ist. Der Hotline-Mitarbeiter, der Alices Anruf entgegennimmt, arbeitet typischerweise mit einer RA-Software, deren Funktionalität so eingeschränkt ist, dass er damit nur Sperrungen durchführen kann.

PKI-Anwendung

Interessant wird eine PKI erst, wenn Alice und Bob ihre Zertifikate und Schlüssel zum Verschlüsseln und Signieren nutzen können. Diese Aussage ist eigentlich eine Binsenweisheit, doch sie hat einen wichtigen Hintergrund. Als Ende der neunziger Jahre die ersten Unternehmen einen PKI-Aufbau angingen, machten sich die Beteiligten häufig zu wenig Gedanken über die genaue Nutzung der PKI. In vielen Unternehmen versuchte die Abteilung für IT-Sicherheit, den Aufbau einer PKI als

Infrastrukturmaßnahme durchzudrücken, um später nach und nach PKI-Anwendungen wie E-Mail-Verschlüsselung oder Virtuelle Private Netze einführen zu können. Diese Strategie scheiterte jedoch allzu oft an den Entscheidern im Unternehmen, die keine große Geldsumme für eine Infrastruktur in die Hand nehmen wollten, deren Zweck noch nicht klar erkennbar war. Diese Entwicklung ist ein Grund dafür, dass das Thema PKI um die Jahrtausendwende immer mehr zum Sorgenkind wurde.

Die Erfahrungen aus der Vergangenheit haben dazu geführt, dass IT-Sicherheitsabteilungen und Security-Berater PKI-Projekte heute anders angehen. Im Vordergrund steht dabei stets die Frage, wo das Unternehmen der Schuh drückt. Dies kann der Wunsch nach sicheren E-Mails sein oder auch die geplante Einführung eines Virtuellen Privaten Netzes. Kann eine Kryptografie-Anwendung mit PKI-Unterstützung das Problem lösen, dann ist ein PKI-Aufbau die logische Konsequenz. Eine PKI wird in diesem Fall nicht als Infrastrukturmaßnahme betrachtet, sondern als ein notwendiges Anhängsel einer konkreten Maßnahme. Diese neue Betrachtungsweise hat dazu geführt, dass die PKI ihren Sorgenkindstatus inzwischen wieder verloren hat.

Wo aber liegen nun die interessantesten Anwendungsmöglichkeiten einer PKI? Beispiele sind E-Mail-Verschlüsselung, Verschlüsselung von Dateien, Virtuelle Private Netze (VPNs) sowie die Signatur digitaler Dokumente. Wie diese Anwendungen funktionieren und was es sonst noch gibt, erfahren Sie in Teil 5 dieses Buchs.

Personal Security Environment (PSE)

Die Umgebung, in der Alices privater Schlüssel gespeichert ist, wird als **Personal Security Environment** (**PSE**) bezeichnet. Im Idealfall ist ein PSE eine Smartcard (oder ein USB-Token mit Smartcard-Funktionalität), die Alice ständig bei sich trägt und die durch eine PIN geschützt ist. Dies nennt man auch **Hardware-PSE**. Die wichtigste Alternative besteht darin, den Schlüssel in einer verschlüsselten Datei auf Alices Festplatte abzulegen – man spricht in diesem Fall von einem **Software-PSE**. Der symmetrische Schlüssel zur Verschlüsselung des privaten Schlüssels im PSE ist hierbei von einem Passwort abgeleitet. Alice muss dieses Passwort eingeben, wenn sie den privaten Schlüssel nutzen will.

Neben Hardware-PSEs und Software-PSEs haben sich in den letzten Jahren auch **Roaming-PSEs** etabliert. Ein Roaming-PSE sieht vor, dass Alices privater Schlüssel auf einem Server liegt. Wenn Alice den Schlüssel nutzen will, muss sie eine Online-Verbindung zum Roaming-PSE aufbauen und den Schlüssel herunter- oder die Nachricht hochladen. Da dies automatisiert erfolgen sollte, benötigt Alice eine Client-Software (**Roaming Client**), die die Kommunikation mit dem Roaming-PSE erledigt. Falls der Roaming Client den Schlüssel herunterlädt, sollte er diesen anschließend wieder aus dem Speicher löschen.

Es gibt einige Standards, die im Zusammenhang mit PSEs von Interesse sind:

- Die Schnittstelle zwischen Smartcard und Anwendung wird in der Regel mit PKCS#11 oder MS-CAPI realisiert (Abschnitt 24.3).
- Für die Speicherung des privaten Schlüssels, der PIN und der Zertifikate auf der Smartcard wird eine Dateistruktur benötigt. Eine solche wird im Standard **ISO/IEC 7816-15** beschrieben [ISO7816-15]. Dieser basiert auf einem inzwischen eingestellten Teil des PKCS-Standards PKCS#15, der einen ähnlichen Inhalt hat [PKCS#15].
- Soll der Zugang zu einer Smartcard mit Biometrie geschützt werden, dann ist **ISO/IEC 7816-11** relevant [ISO7816-11].
- Zur Realisierung eines Software-PSEs kann der bereits erwähnte PKCS#8-Standard genutzt werden (siehe Abschnitt 24.1.3).
- Die in ISO/IEC 7816-15 (bzw. PKCS#15) beschriebene Dateistruktur kann auch in einem Software-PSE eingesetzt werden. Wird dies gemacht, dann wird das resultierende Software-PSE **virtuelle Smartcard** genannt. Eine solche emuliert ein ISO/IEC-7816-15-konformes Hardware-PSE. Die in PKCS#15 beschriebene Virtuelle Smartcard wird mithilfe des PKCS#7-Formats und RFC 2630 (siehe Abschnitt 18.4.1) realisiert.
- Es gibt Fälle, in denen der Inhalt eines PSE (also vor allem der private Schlüssel) exportiert oder importiert werden muss. Bei Hardware-PSEs kommt dieser Fall kaum vor, da ein privater Schlüssel die Karte nie verlassen sollte. Bei Software-PSEs und HSMs kann ein Import oder Export jedoch sinnvoll sein. Formate für das verschlüsselte Speichern von PSE-Informationen und ähnlichen persönlichen Daten werden in **PKCS#12** beschrieben [PKCS#12]. PKCS#12 kennt verschiedene Modi, wobei meist eine Passwort-basierte Verschlüsselung gemäß einem PKCS#5-ähnlichen Verfahren eingesetzt wird. PKCS#12 wird insbesondere verwendet, um einen von der CA generierten privaten Schlüssel verschlüsselt zum Benutzer zu transportieren.

Eine wichtige Frage beim Betrieb einer PKI lautet: Wie viele Schlüsselpaare soll ein Anwender erhalten? Theoretisch reicht eines aus. So könnte Alice mit demselben privaten RSA-Schlüssel entschlüsseln, signieren und sich authentifizieren. In der Praxis ist dies jedoch meist nicht erwünscht. Da die Erstellung einer RSA-Signatur mathematisch gesehen derselbe Vorgang ist wie die Entschlüsselung einer RSA-verschlüsselten Nachricht, ermöglicht es Mallory verschiedene Angriffe, wenn Alice jeweils denselben Schlüssel verwendet (dies kann man durch eine geeignete Datenformatierung zwar verhindern, doch wer weiß, ob eine solche in allen Einsatzszenarien angewendet wird?). Darüber hinaus gelten für Signatur, Verschlüsselung und Authentifizierung oft unterschiedliche Policys. So kann der Betreiber einer PKI beispielsweise unterschiedliche Zertifikats-Gültigkeitsdauern oder unterschiedliche Schlüssellängen vorsehen. Recovery (Abschnitt 28.2) ist für Signatur und Authentifizierung in der Regel überflüssig, dafür aber für die Verschlüsselung umso wichtiger.

Aus den genannten Gründen vergeben die meisten CAs gleich mehrere Zertifikate an einen Anwender. In den häufigsten Fällen sind es zwei (zum Beispiel eines für Verschlüsselung und eines für digitale Signaturen) oder drei (Verschlüsselung, Signatur, Authentifizierung). Wenn viele PKI-Anwendungen vorgesehen sind, kann es durchaus auch vorkommen, dass Alice vier oder mehr Zertifikate auf einmal erhält. Wenn Alice ein Software-PSE oder ein Roaming-PSE erhält, dann ist dies technisch meist kein Problem, da genügend Speicherplatz vorhanden ist. Bei einer Smartcard können jedoch Speicherprobleme entstehen. Fünf Schlüsselpaare und einige weitere Zusatzinformationen kann ein gängiger Karten-Chip jedoch normalerweise verkraften.

Neben dem Speichern der aktuellen privaten Schlüssel sollte ein PSE auch das Speichern veralteter Schlüssel unterstützen. Dies nennt man **Key History**. Die Notwendigkeit einer Key History ist leicht einzusehen. Es kann in der Praxis oft vorkommen, dass Alice Daten entschlüsseln will, die mit einem alten öffentlichen Schlüssel verschlüsselt sind. Eine Key History spielt übrigens nur im Zusammenhang mit Verschlüsselung eine Rolle. Wenn es um digitale Signaturen und Authentifizierung geht, gibt es keine sinnvolle Verwendung für einen veralteten Schlüssel.

26.5.2 Rollen in einer PKI

Neben Komponenten sind in einer PKI auch Menschen von Bedeutung, die bestimmte Rollen ausfüllen. Die folgende Liste nennt die typischen Rollen eines PKI-Rollenmodells:

- **PKI-Leiter**: Der PKI-Leiter ist derjenige, der die Gesamtverantwortung für eine PKI trägt. Diese Rolle entspricht etwa der eines Projektleiters. Alle wichtigen Entscheidungen innerhalb der PKI müssen vom PKI-Leiter getragen werden.
- **CA-Operator**: Der CA-Operator ist für die Durchführung aller strategisch wichtigen Vorgänge innerhalb der PKI zuständig. Er hat alle Rechte, um die CA aufzusetzen, CA-Schlüssel zu generieren, die CA außer Betrieb zu nehmen und um beliebige Konfigurationseinstellungen zu ändern. Der CA-Operator hat also die volle administrative Kontrolle über die CA, ist jedoch nicht für das Tagesgeschäft zuständig. In der Praxis kann es auch mehrere CA-Operatoren geben. In der Regel übernimmt der PKI-Leiter einen der Operatoren-Posten, um dadurch aktiv in die PKI eingreifen zu können. Wenn innerhalb einer PKI Key Recovery eingesetzt werden soll (siehe Abschnitt 28.2), dann bietet es sich an, das Auslesen eines privaten Anwenderschlüssels aus der Recovery-Datenbank vom CA-Operator vornehmen zu lassen. Auf diese Weise wird sichergestellt, dass ein Recovery-Vorgang eine Ausnahme bleibt, die nur in begründeten Fällen zur Ausführung kommt.

- **PKI-Administrator**: Der PKI-Administrator ist für das Tagesgeschäft in der PKI-Administration zuständig. Dazu gehört beispielsweise die routinemäßige Überwachung der Trust-Center-Komponenten, der Betrieb des Backup-Systems und die Wartung der Logdaten-Erhebung. Es kann innerhalb einer PKI mehrere PKI-Administratoren geben, die sich die Arbeit teilen.
- **Registrator**: Der Registrator ist für den Betrieb einer RA zuständig. Er nimmt Zertifizierungsanträge entgegen und startet die Generierung eines Zertifikats. Darüber hinaus ist ein Registrator meist auch in anderen Fällen der Ansprechpartner des Anwenders, beispielsweise wenn es um das Sperren von Zertifikaten oder die Entgegennahme von Fragen geht.
- **Sperragent**: Der Sperragent ist für den Betrieb der Sperrstelle zuständig und hat dabei im Wesentlichen nur eine Aufgabe: Sperranträge von Anwendern entgegenzunehmen und sie an die CA weiterzuleiten. Der Sperragent ist meist ein Mitarbeiter einer Telefon-Hotline, der abgesehen vom Sperrstellenbetrieb keine weiteren Aufgaben innerhalb der PKI hat.
- **PKI-Anwender**: Eine wichtige Rolle spielen in einer PKI auch deren Anwender. In unserem Fall sind dies Alice und Bob. In der Praxis bilden meist die Mitarbeiter eines Unternehmens oder die Kunden eines öffentlichen Trust Centers die Anwender.

Um Insiderangriffe zu vermeiden (Abschnitt 17.5), sollte in einem PKI-Rollenmodell das Need-to-know-Prinzip (jede Rolle erhält nur diejenigen Rechte, die sie benötigt) beachtet werden. Dies bedeutet zum Beispiel, dass ein Sperragent nicht die Möglichkeit zur Generierung von Zertifikaten erhalten sollte. Eine strikte Rollentrennung ist innerhalb einer PKI dagegen meist nicht notwendig. So kann der PKI-Leiter durchaus auch CA-Operator sein, und ein CA-Operator kann auch als Registrator aktiv werden. Eine Rollentrennung hat allenfalls dann einen Sinn, wenn beispielsweise ein PKI-Auditor – dieser überprüft, ob der PKI-Betrieb ordnungsgemäß abläuft – als weitere Rolle eingerichtet wird. Der PKI-Auditor sollte aus naheliegenden Gründen keine Person sein, die bereits in anderer Funktion in den PKI-Betrieb involviert ist.

26.5.3 Prozesse in einer PKI

Der Sinn einer PKI ist das Erstellen, Sperren und Verwalten digitaler Zertifikate. Dies bezeichnet man auch als **Zertifikatemanagement**. Damit eine PKI diesen Zweck erfüllen kann, müssen diverse Prozesse abgearbeitet werden können. Um diese Prozesse geht es im Folgenden. Für die Kommunikation, die im Rahmen der typischen PKI-Prozesse zwischen den Komponenten einer PKI stattfindet, stellt der PKIX-Standard eine Familie von Protokollen zur Verfügung, die unter dem Namen **Certificate Management Protocols** (**CMP**) zusammengefasst werden. Sie werden in [RFC4210] beschrieben. Alle CMP-Protokolle verwenden einheitlich

aufgebaute Protokollnachrichten. Übertragen werden diese mithilfe eines anderen Protokolls – möglich sind etwa HTTP oder FTP. Auch die Übertragung direkt über TCP ist möglich. Zudem können CMP-Protokollnachrichten per E-Mail verschickt werden. Bei der folgenden Betrachtung von PKI-Prozessen spielt jeweils am Rande auch CMP eine Rolle.

CA-Initialisierung

Bevor eine CA den Betrieb aufnehmen kann, müssen einige Schritte zur Initialisierung abgearbeitet werden. So muss der Betreiber die CA-Software unter kontrollierten Umständen in Gang setzen, wobei er vor allem auf den privaten Schlüssel der CA achten muss. Dieser muss mit einem guten Zufallsgenerator generiert werden und darf beim Aufbau der CA nicht unbemerkt von einem Administrator kopiert werden können. In einer unternehmensinternen PKI wird die CA-Initialisierung meist recht pragmatisch gehandhabt. Der Prozess läuft im Wesentlichen ab wie die Einführung eines anderern Serversystems. Anders verhält es sich, wenn eine CA dem deutschen Signaturgesetz entsprechen muss oder aus anderen Gründen ein hohes Sicherheitsniveau gefordert wird. In einem solchen Fall ist ein aufwendiger Prozess notwendig, der unter anderem ein Vier-Augen-Prinzip, eine genaue Protokollierung aller Vorgänge sowie organisatorische Sicherheitsmaßnahmen erfordert.

CMP spielt im Zusammenhang mit der CA-Initialisierung keine große Rolle. Es werden lediglich einige Protokollnachrichten beschrieben, etwa für die Verbreitung einer initialen Sperrliste und für den Export des öffentlichen Schlüssels.

Initialisierung einer Endeinheit

Auch die diversen PKI-Endeinheiten müssen initialisiert werden. Dies ist zwar kein so aufwendiger Vorgang wie bei der CA, ganz trivial ist er jedoch trotzdem nicht. CMP sieht an dieser Stelle vor allem eine Protokollnachricht für den Import des öffentlichen Schlüssels der CA in die Endeinheit vor. Die Registrierung von Anwendern bei der CA und die Generierung von Zertifikaten gehören nicht in diesen Bereich.

Anwender-Enrollment

Der wichtigste und komplexeste Prozess in einer PKI ist das Ausstellen eines Zertifikats für einen Anwender. Dieser Vorgang, zu dem unter anderem die Registrierung des Anwenders, die Generierung eines Schlüsselpaars, das Signieren eines Zertifikats sowie die Übergabe des PSE gehören, wird auch als Enrollment (bzw. Anwender-Enrollment) bezeichnet. Das Thema Enrollment wird in Abschnitt 28.1 ausführlich behandelt.

Erneuerung eines Zertifikats

Nachdem Alice ein Zertifikat von der CA erhalten hat und dieses einige Zeit genutzt hat, kann es vorkommen, dass sie ein neues Zertifikat benötigt. Dies kann folgende Gründe haben:

- Der Gültigkeitszeitraum des Zertifikats neigt sich dem Ende entgegen.
- Der Inhalt des Zertifikats hat sich geändert. Dies kann etwa Alices Nachname sein (nachdem sie geheiratet hat) oder ihre E-Mail-Adresse.
- Alices Zertifikat ist gesperrt worden.

Damit eine Zertifikatserneuerung reibungslos vonstatten gehen kann, muss der Betreiber einer PKI einen Prozess dafür festlegen. In vielen Fällen läuft dieser Prozess ähnlich ab wie das Enrollment. Manche CA-Softwarelösungen unterstützen eine automatische E-Mail-Benachrichtigung des Anwenders für den Fall, dass dieser ein neues Zertifikat erhalten soll. In einer solchen E-Mail kann Alice beispielsweise dazu aufgefordert werden, die RA oder eine bestimmte Webseite aufzusuchen. Zudem bieten manche PKI-Softwarehersteller spezielle Clients an, die auf Alices Rechner installiert eine Zertifikatserneuerung automatisch durchführen können. Hierbei sendet der Client einen Erneuerungsantrag an die CA, wobei der zum alten Zertifikat gehörende private Schlüssel zur Authentifizierung verwendet wird.

Key Recovery

Ein besonders kritisches Thema innerhalb einer PKI ist **Key Recovery**. Darunter versteht man den Vorgang, dass eine CA einen privaten Schlüssel, der ja im Normalfall nur für den Besitzer zugänglich ist, aus einer Datenbank holt und dem Administrator zur Verfügung stellt. Um Key Recovery zu ermöglichen, muss eine CA eine Datenbank betreiben, in der alle privaten Schlüssel abgelegt werden. Dies ist nur dann sinnvoll möglich, wenn die Schlüsselpaare in der CA generiert werden.

Key Recovery ist oftmals eine sinnvolle Sache. Wenn Alice wichtige Daten verschlüsselt gespeichert hat und ihre Smartcard verliert, dann sind alle Daten verloren – es sei denn, die CA hat Alices privaten Schlüssel gespeichert und führt ein Key Recovery durch. Auch für Alices Arbeitgeber Krypt & Co. kann Key Recovery wichtig sein. Wenn Alice nämlich eines Tages gegen einen Baum fährt oder das Unternehmen im Zorn verlässt, dann können durch ein Key Recovery wenigstens ihre verschlüsselten Daten gerettet werden. Nicht zuletzt kann Key Recovery auch vom Staat genutzt werden. Wenn etwa die Polizei die Möglichkeit hat, bei Bedarf den privaten Schlüssel eines Kriminellen wiederherzustellen, dann erleichtert dies ihre Arbeit.

Trotz aller Vorteile ist Key Recovery mit Vorsicht zu genießen. Es kann nämlich sehr leicht missbraucht werden. Gelingt es Mallory, sich Zugang zur Datenbank zu verschaffen, in der die privaten Schlüssel abgelegt sind, dann bricht die gesamte Sicherheit einer PKI in sich zusammen. Außerdem dürfte klar sein, dass Alice nicht glücklich ist, wenn ihr Arbeitgeber ihre verschlüsselten Dateien und Nachrichten lesen kann. Key Recovery durch den Staat ist erst recht ein Albtraum, der an George Orwells Überwachungsstaat erinnert.

Über das Für und Wider von Key Recovery kann man lange diskutieren. Unbestritten ist, dass folgende Sicherheitsvorkehrungen getroffen werden sollten, wenn es praktiziert wird:

- Die zum Speichern der Schlüssel verwendete Datenbank muss in einer geschützten Umgebung betrieben werden.
- Key Recovery darf nur unter besonderen Umständen, die genau definiert werden müssen, erlaubt sein.
- Key Recovery darf nur für solche private Schlüssel vorgesehen werden, die zum Verschlüsseln von Daten verwendet werden, die längere Zeit gespeichert werden. Wird ein privater Schlüssel nur zum Signieren benutzt, dann ist kein Key Recovery notwendig. Dasselbe gilt für Schlüssel, die nur zum Verschlüsseln von Nachrichten während der Übertragung (etwa während eines Telefongesprächs) oder zu Authentifizierung verwendet werden.

Für die Durchführung eines Key Recovery sieht CMP einige Protokollnachrichten vor, die beispielsweise zwischen der CA und der Recovery-Datenbank zum Einsatz kommen können. Im Signaturgesetz und dem dazugehörenden ISIS-MTT-Standard ist dagegen kein Key Recovery erlaubt. Einen ausführlichen Überblick zum Thema Key Recovery und Alternativen dazu gibt es in Abschnitt 28.2.

Sperren eines Zertifikats

Zu den großen Vorteilen einer hierarchischen PKI gegenüber einem Web of Trust gehört, dass das Sperren von Zertifikaten einfach durchgeführt und durchgesetzt werden kann. Für Sperrungen ist stets die CA zuständig. Im Fall einer Schlüsselkompromittierung muss jedoch Alice die Kompromittierung des Schlüssels melden und damit einen Sperrantrag abgeben. Für einen solchen Sperrantrag sieht CMP Protokollnachrichten vor. Alice kann eine solche Protokollnachricht selbst an die CA senden, wenn ihr privater Schlüssel kompromittiert worden ist. Es ist jedoch auch möglich, dass die CA einen telefonischen Sperrdienst betreibt, der Sperranträge mündlich entgegennimmt und dann in digitaler Form an die CA weiterleitet.

26.6 Identitätsbasierte Krypto-Systeme

Der Betrieb einer PKI ist eine aufwendige Sache. Vor allem die Prozesse für das Enrollment, die Sperrung und die Sperrprüfung binden viele Ressourcen, obwohl sie nicht dem eigentlichen Zweck – dem Verschlüsseln und Signieren – dienen. Schon seit langem suchen Experten daher nach Infrastrukturen, die bei weniger Aufwand einen ähnlichen Nutzen bieten wie eine PKI. Als wichtigste PKI-Alternative gilt derzeit die **identitätsbasierte Kryptografie** (man spricht auch von **identitätsbasierten Krypto-Systemen**). In diesem Unterel will ich Ihnen diese interessante Technologie vorstellen. Da identitätsbasierte Krypto-Systeme bisher jedoch in der Praxis kaum eingesetzt werden (im Gegensatz zur Public-Key-Infrastrukturen), wird das Thema in den darauffolgenden Kapiteln nicht weiter betrachtet.

26.6.1 Funktionsweise

Hinter der identitätsbasierten Kryptografie steht ein einfaches Prinzip: Alices öffentlicher Schlüssel ist mit ihrem Namen (bzw. einer Kennung oder ihrer E-Mail-Adresse) identisch. Wenn Bob Alice eine verschlüsselte Nachricht schicken will, dann benötigt er dazu weder ein Zertifikat noch einen unhandlichen RSA- bzw. Diffie-Hellman-Schlüssel, sondern lediglich eine kurze Zeichenkette. Wir nehmen für die folgenden Betrachtungen an, dass es sich dabei um Bobs E-Mail-Adresse handelt.

Leider lässt sich diese einfache Idee nicht ohne weiteres in die Praxis umsetzen. Schließlich sehen alle bekannten asymmetrischen Verfahren vor, dass Alice aus einem (innerhalb bestimmter Grenzen frei wählbaren) privaten Schlüssel mit einer Einwegfunktion den öffentlichen Schlüssel berechnet. Wenn wir also einen frei wählbaren öffentlichen Schlüssel haben wollen, dann müssen wir die Einwegfunktion umdrehen. Genau das geht jedoch nicht, ohne das Verfahren zu knacken.

Die identitätsbasierte Kryptografie löst dieses Dilemma durch die Hinzunahme eines Servers, der sich an der Kommunikation zwischen Alice und Bob beteiligt. Dieser Server hat einen öffentlichen Schlüssel – allerdings einen, der nicht mit seinem Namen bzw. seiner E-Mail-Adresse identisch ist. Will Alice eine Nachricht an Bob verschlüsseln, dann benötigt sie dazu den öffentlichen Server-Schlüssel und Bobs E-Mail-Adresse. Bobs privater Schlüssel wird vom Server berechnet, wobei der private Server-Schlüssel in die Berechnung eingeht. Bob erhält seinen privaten Schlüssel vom Server. Er ändert sich nicht, solange sich Bobs E-Mail-Adresse und der private Schlüssel des Servers nicht ändern. In ähnlicher Weise lassen sich auch identitätsbasierte digitale Signaturen realisieren.

26.6.2 Das Boneh-Franklin-Verfahren

Die identitätsbasierte Kryptografie stammt aus den achtziger Jahren und ist damit nicht gerade eine neue Erfindung. Schon in Bruce Schneiers Standardwerk [Schn96] findet sich ein kurzes Kapitel zu diesem Thema – allerdings mit dem Hinweis, dass die damals bekannten Verfahren dieser Art unpraktikabel oder unsicher waren. An dieser Situation hat sich erst in den letzten Jahren einiges geändert. Als erstes praxistaugliches identitätsbasiertes Krypto-System gilt das 2001 veröffentlichte **Boneh-Franklin-Verfahren** [BonFra]. Da dieses nur zur Verschlüsselung dient (für digitale Signaturen ist es nicht geeignet), redet man in diesem Zusammenhang auch von *identitätsbasierter Verschlüsselung*. Auch die Abkürzung **IBE** (*Identity-Based Encryption*) wird dafür verwendet.

Funktionsweise

Das Boneh-Franklin-Verfahren basiert auf dem Diffie-Hellman-Schlüsselaustausch. Ich übernehme daher die Bezeichnungen aus Abschnitt 11.2. Im Folgenden sei x Alices privater Schlüssel, y sei der private Schlüssel des Servers. Für das Boneh-Franklin-Verfahren spielt zudem eine sogenannte **Paarungsfunktion** (auch Pairing genannt) eine wichtige Rolle. Eine Paarungsfunktion f hat im Zusammenhang mit Diffie-Hellman folgende Eigenschaft (g_1 und g_2 sind natürliche Zahlen):

$$f(g_1^x, g_2^y) = f(g_1^y, g_2^x) \pmod{p}$$

Will Alice eine Nachricht an Bob verschicken, dann generiert sie zunächst einen Sitzungsschlüssel. Dazu nimmt sie eine aus Bobs E-Mail-Adresse generierte Zahl i und wendet folgende Formel an:

$$k = f(i^x, g^y) \pmod{p}$$

Mit k als Schlüssel verschlüsselt sie die Nachricht mit einem symmetrischen Verfahren. Um den Sitzungsschlüssel berechnen zu können, benötigt Empfänger Bob einen privaten Schlüssel z, den ihm der Server zur Verfügung stellt. Der Server berechnet z nach folgender Formel:

$$z = i^y \pmod{p}$$

Nun kann Bob den Sitzungsschlüssel auf folgende Weise berechnen:

$$k = f(i^y, g^x) \pmod{p}$$

Die Eigenschaften der Paarungsfunktion stellen sicher, dass Alice und Bob tatsächlich denselben Sitzungsschlüssel erhalten.

Leider hat der beschriebene Ablauf einen kleinen Schönheitsfehler: Bisher ist keine Paarungsfunktion bekannt, die sich in diesem Zusammenhang nutzen ließe. Die Situation ändert sich jedoch, wenn wir den Diffie-Hellman-Schlüsselaustausch auf Basis elliptischer Kurven (ECDH) verwenden. Hier sind zwei Paarungsfunk-

tionen bekannt (diese heißen *Weil-Pairing* und *Tate-Pairing*). Das Boneh-Franklin-Verfahren verwendet das Weil-Pairing. Auf einen Nenner gebracht, lässt sich also sagen: Das Bonneh-Franklin-Verfahren ist eine Weiterentwicklung von ECDH, bei der das Weil-Pairing angewendet wird.

Bewertung des Boneh-Franklin-Verfahrens

2002 gründete Dan Boneh, einer der beiden Erfinder des Boneh-Franklin-Verfahrens, zusammen mit einigen Kollegen die Firma Voltage Security. Diese vermarktet seitdem eine Softwareproduktlinie, die das Boneh-Franklin-Verfahren in die Praxis umsetzt. Daher ist meistens die Lösung von Voltage Security gemeint, wenn außerhalb der Krypto-Szene von identitätsbasierter Verschlüsselung die Rede ist. Dies sollte jedoch nicht darüber hinwegtäuschen, dass es inzwischen mehrere Dutzend andere identitätsbasierte Krypto-Systeme gibt, sowie zahlreiche weitere Verfahren, die ebenfalls eine Paarungsfunktion verwenden. [Gagné] und [Barret] sind empfehlenswerte Literaturquellen dazu.

Voltage Security konnte einige namhafte Kunden (unter anderem Banken) gewinnen. Vom großen Durchbruch ist die identitätsbasierte Verschlüsselung jedoch noch ein gutes Stück entfernt. Voltage Security hält mehrere Patente zur identitätsbasierten Verschlüsselung und will sich dadurch offensichtlich Exklusivität verschaffen. Diese Vorgehensweise könnte dazu führen, dass andere Anbieter einen Bogen um das Boneh-Franklin-Verfahren machen, was der Akzeptanz nicht besonders förderlich sein dürfte. Zudem steht die Standardisierung der identitätsbasierten Kryptografie noch am Anfang (P1363.3 könnte ein entsprechender Standard werden, siehe Abschnitt 18.3.2).

Unabhängig von solchen Überlegungen hat das Boneh-Franklin-Verfahren unbestreitbare Vorteile. Insbesondere kommt es ohne digitale Zertifikate aus, was den Betrieb einer entsprechenden Infrastuktur deutlich vereinfacht. So gibt es keine Notwendigkeit für einen Zertifikateserver, für Zertifikats-Sperrungen oder für Sperrlisten. Auch das Enrollment ist einfacher: Empfänger Bob muss sich erst registrieren, wenn er eine Nachricht von Alice entschlüsseln will, die er bereits empfangen hat. Abgesehen davon gibt es bezüglich der Sicherheit des Boneh-Franklin-Verfahrens bisher nichts Negatives zu berichten.

Dem stehen allerdings auch Nachteile gegenüber. Viele davon haben mit dem Server zu tun, ohne den die identitätsbasierte Kryptografie nicht auskommt. Häufig ist beispielsweise unklar, wer diesen Server betreibt. Neutrale Anbieter von solchen Servern gibt es bisher kaum. Wenn Alice eine Mail an Bob schreibt, dann muss sie daher entweder dem Server von Bobs Arbeitgeber vertrauen – oder umgekehrt. Beides ist problematisch, denn immerhin hat der Server Zugriff auf alle privaten Schlüssel.

Die zentrale Rolle des Servers birgt noch ein weiteres Problem. Da dieser – im Gegensatz zu einer CA in einer PKI – auf alle privaten Schlüssel zugreifen kann, muss er gut geschützt sein. Ein erfolgreicher Angriff auf den Server ermöglicht es Mallory, das gesamte System auszuhebeln. Ein solcher Schutz ist nicht einfach zu bewerkstelligen. Zwar ist es möglich, den privaten Schlüssel des Servers auf mehrere Stellen zu verteilen, doch das macht die Sache deutlich umständlicher.

Neben dem Server gibt es noch andere wunde Punkte in der identitätsbasierten Kryptografie. So ist etwa das Sperren eines öffentlichen Schlüssels schwierig. Zum einen gibt es keine Sperrabfragen, und zum anderen will Bob vermutlich nicht bei einer Sperrung auch gleich die E-Mail-Adresse ändern (diese ist ja sein öffentlicher Schlüssel). Daher bleibt nichts anderes übrig, als allen Schlüsseln eine kurze Laufzeit zu geben (z.B. eine Woche). Dadurch erhält Bob ständig einen neuen Schlüssel, was sicherlich auch nicht ideal ist.

Ein weiterer Nachteil: Digitale Signaturen spielen in der Voltage-Lösung keine Rolle. Es gibt auch keine naheliegende Möglichkeit, dies zu ändern. Das Thema Authentisierung lässt sich ebenfalls nicht ohne weiteres einbinden. Im Vergleich zu einer PKI bietet das Boneh-Franklin-Verfahren also nur einen eingeschränkten Funktionsumfang.

Trotz der genannten Nachteile ist die identitätsbasierte Verschlüsselung eine interessante Technik, die es weiter zu betrachten lohnt. Ob sie eine ernsthafte Alternative zur PKI werden wird, werden die nächsten Jahre zeigen.

27 Digitale Zertifikate

Es versteht sich von selbst, dass digitale Zertifikate standardisiert werden müssen. Das Maß aller Dinge hierbei ist der bereits erwähnte X.509-Standard [X.509], von dem es drei Versionen gibt und der zahlreiche Profile nach sich gezogen hat. Auch die Standards PKIX und Common PKI verwenden X.509-Zertifikate und spezifizieren jeweils ein Profil dazu. Daneben gibt es mit Open-PGP- und CV-Zertifikaten, auch solche, die nicht auf X.509 basieren.

27.1 X.509v1- und X.509v2-Zertifikate

Die erste Version von X.509 erschien 1988. Zertifikate, die dem darin beschriebenen Format entsprechen, werden **X.509v1-Zertifikate** genannt. 1993 folgte **X.509v2**.

27.1.1 Das Format

Gemäß X.509v1 hat ein Zertifikat folgende Bestandteile (**Felder**):

- *Version*: Dies ist die X.509-Versionsnummer des Zertifikats. Bei X.509v1 ist dies die Zahl Null für Version 1 (dies entspricht der Informatik-Gepflogenheit, beim Zählen mit Null anzufangen). Bei späteren Versionen steht an dieser Stelle entsprechend eine Eins oder eine Zwei.
- *Serial Number*: Die Seriennummer des Zertifikats ist hiermit gemeint. Dies ist eine maximal 20 Byte lange Zahl, die unter allen von einer CA ausgestellten Zertifikaten eindeutig sein muss. Da die CA einen eindeutigen Namen haben muss, liefert die Kombination aus CA-Name und Seriennummer eine weltweit eindeutige Kennung für jedes Zertifikat. Viele CA-Implementierungen verwenden eine Zufallszahl oder einen Hashwert oder die Uhrzeit als Seriennummer. Dadurch kann sich die CA einen Abgleich mit bereits vergebenen Seriennummern sparen.
- *Signature*: Dies ist nicht die Signatur, sondern die Kennung des Signaturverfahrens (eventuell ergänzt durch zugehörige Parameter), mit dem das Zertifikat signiert ist. Meist ist dies RSA, manchmal auch DSA oder ein Verfahren auf Basis elliptischer Kurven. Dieses Feld ist eigentlich überflüssig, da die Signatur des Zertifikats ebenfalls mit einer Kennung des Signaturverfahrens versehen ist. Aus Gründen der Kompatibilität darf es eine CA jedoch nicht weglassen.
- *Issuer*: Hier ist der Name der CA gespeichert, die das Zertifikat signiert hat. Dabei ist ein Name vorgesehen, welcher der X.500-Namensgebung entspricht (dies ist eine der erwähnten X.500-Altlasten).
- *Validity*: Dieses Feld bezeichnet den Gültigkeitszeitraum des Zertifikats. Dieser wird durch ein Anfangs- und ein Enddatum angegeben.
- *Subject*: Dies ist der Name des Zertifikatsinhabers. Auch hier ist ein X.500-Name vorgesehen. Entspricht das Zertifikat dem X.509v3-Standard (siehe unten), dann kann eine CA dieses Feld auch leer lassen und stattdessen nur den Namen im Feld Subject Alternative Name verwenden.
- *Subject Public Key Info*: In diesem Feld ist der öffentliche Schlüssel des Zertifikatsinhabers gespeichert. Dieser öffentliche Schlüssel ist der Grund für die Existenz eines Zertifikats. Das Subject-Public-Key-Info-Feld ist strukturiert. Neben dem öffentlichen Schlüssel enthält es die Kennung des Signaturverfahrens, für das dieser verwendet werden soll.

X.509v2 fügte zu den vorhandenen Feldern zwei neue hinzu:

- *Issuer Unique ID*: Dies ist eine eindeutige Kennung der CA. Wenn mehrere CAs denselben Namen haben, ist dadurch eine Unterscheidung möglich.

- *Subject Unique ID*: Diese eindeutige Kennung des Zertifikatsinhabers ermöglicht eine Unterscheidung, wenn mehrere Zertifikatsinhaber denselben Namen haben.

Die meisten X.509-Implementierungen verwenden die beiden X.509v2-Felder nicht. Auch in den X.509-Profilen von PKIX und Common PKI wird empfohlen, diese Felder einfach leer zu lassen. Stattdessen sollte eine CA dafür sorgen, dass Namensgleichheiten nicht vorkommen. X.509v2 erwies sich somit als überflüssige Erweiterung des X.509-Standards.

27.1.2 Nachteile von X.509v1 und v2

Im Laufe der Jahre zeigte sich, dass X.509v1-Zertifikate einige Nachteile haben, die durch die zweite X.509-Version nicht behoben wurden:

- X.509v1 sieht als Inhaber- und CA-Namen jeweils X.500-Namen vor, die einem bestimmten Format entsprechen müssen. Da X.509-Zertifikate meist unabhängig von X.500 eingesetzt werden, ist diese Einschränkung aus heutiger Sicht unsinnig. X.509v1 erlaubt es beispielsweise nicht, als Name lediglich eine E-Mail-Adresse zu verwenden, obwohl dies oftmals sinnvoll wäre.
- X.509v1-Zertifikate lassen keine Rückschlüsse auf den Verwendungszweck des zertifizierten öffentlichen Schlüssels zu. Es gibt kein Feld, das eine entsprechende Aussage machen könnte. Es ist jedoch sinnvoll, beispielsweise zwischen Schlüsseln zum Verschlüsseln und zur Verifikation von Signaturen zu unterscheiden.
- X.509v1-Zertifikate machen keine Aussage über die Policy, die ihnen zugrunde liegt. Alices Zertifikat ist daher zum Beispiel nicht anzusehen, ob Alice es ohne nennenswerte Authentifizierung per E-Mail bei der CA bestellt oder nach Vorlage ihres Ausweises persönlich abgeholt hat.

Um diese Mängel zu beheben, entwickelten einige Unternehmen den Standard **PKCS#6**, der das X.509-Format erweitert [PKCS#6]. PKCS#6 erwies sich jedoch schnell als überflüssig, da mit der dritten X.509-Version eine leistungsfähigere Alternative entstand.

27.2 X.509v3-Zertifikate

1996 erschien die dritte Version des X.509-Standards. X.509v3 sah zunächst keine neuen Felder für X.509-Zertifikate vor, sondern spezifizierte eine Syntax, mit der neue Felder (**Erweiterungen**) definiert werden können. Jede Erweiterung enthält dabei ein Teilfeld, das angibt, ob die Erweiterung **kritisch** oder **unkritisch** ist. Wenn eine Software eine kritische X.509v3-Erweiterung entdeckt, die sie nicht kennt, dann wird das Zertifikat als ungültig betrachtet. Eine der Software nicht bekannte unkritische Erweiterung wird dagegen übergangen.

27.2.1 Die X.509v3-Standarderweiterungen

Die Erweiterungssyntax von X.509v3 hat den Vorteil, dass mit fehlenden Zertifikatsfeldern endlich Schluss ist. Was auch immer für ein zusätzliches Feld in einem X.509-Zertifikat benötigt wird – X.509v3 bietet die Möglichkeit, es zu definieren. Durch diese Offenheit handelte man sich jedoch ein neues Problem ein: Ein X.509v3-Zertifikat ist nicht automatisch für alle Anwendungen lesbar, die X.509v3-Zertifikate unterstützen. Sobald eine Erweiterung darin enthalten ist, die eine Anwendung nicht kennt, kommt es zu Inkompatibilitäten. Um einem Wildwuchs im Bereich der Zertifikatserweiterungen vorzubeugen, wurde X.509v3 1997 noch einmal erweitert (durch sogenannte Amendments). Dabei wurden mithilfe der in X.509v3 festgelegten Erweiterungssyntax einige Erweiterungen spezifiziert, die nun ein fester Bestandteil des Standards sind. Die folgende Liste nennt diese Erweiterungen:

- *Authority Key Identifier*: Dies ist eine Kennung des CA-Schlüssels. Wenn eine CA mehrere Schlüssel zum Signieren benutzt, dann kann Alice durch dieses Feld feststellen, welcher davon für dieses Zertifikat verwendet wurde. Dass eine CA mehrere Schlüssel verwendet, kann etwa in einer Übergangszeit vorkommen, wenn der CA-Schlüssel routinemäßig gewechselt wird. Das PKIX-Profil schreibt vor, dass diese Erweiterung genutzt, aber als unkritisch markiert wird.
- *Subject Key Identifier*: Dieses Feld enthält eine Kennung des Inhaberschlüssels. Wenn Alices öffentlicher Schlüssel in mehreren digitalen Zertifikaten enthalten ist, kann dies durch diese Kennung schnell festgestellt werden. Dass Alice mehrere Zertifikate für den gleichen Schlüssel hat, kann etwa vorkommen, wenn die Gültigkeitsdauer eines Zertifikats zu Ende geht und Alice sich ein neues mit dem gleichen Schlüssel ausstellen lässt. Alice kann auch denselben Schlüssel von verschiedenen CAs zertifizieren lassen. Das PKIX-Profil schreibt vor, dass diese Erweiterung verwendet und als unkritisch markiert wird. Common PKI lässt dies offen.
- *Key Usage*: Diese Erweiterung nennt den Verwendungszweck des Schlüssels. Durch dieses Feld kann festgeschrieben werden, zu welchem Zweck der im Zertifikat enthaltene öffentliche Schlüssel verwendet werden darf. Beispiele sind Signatur, Verschlüsselung und Signatur von Zertifikaten. Das PKIX-Profil schreibt die Verwendung dieses Felds nicht vor; es empfiehlt lediglich, es als kritisch zu markieren, falls es verwendet wird. Bei Common PKI ist es dagegen obligatorisch.
- *Private Key Usage Period*: Dies ist die Nutzungsdauer für den privaten Schlüssel. Der Sinn dieser Erweiterung besteht darin, dass digitale Signaturen oftmals auch dann noch verifiziert werden müssen, wenn der Inhaber das zugehörige Schlüsselpaar nicht mehr verwendet. Es bietet sich daher an, für den

privaten Schlüssel einen anderen Gültigkeitszeitraum vorzusehen als für das Zertifikat selbst. Dies ist mit dieser Erweiterung möglich. Das PKIX-Profil sieht die Nutzung dieses Felds nicht vor. Common PKI empfiehlt ebenfalls, darauf zu verzichten.

- *Certificate Policies*: In diesem Feld kann die CA mit einer OID auf eine Certificate Policy verweisen, die für dieses Zertifikat gültig ist (siehe 29.3). Das PKIX-Profil schreibt diese Erweiterung nicht zwingend vor, gemäß Common PKI muss sie dagegen verwendet werden.
- *Policy Mappings*: Diese Erweiterung gibt es nur in CA-Zertifikaten. Die CA, die das Zertifikat ausgestellt hat, erklärt damit, dass eine bestimmte Certificate Policy der zertifizierten CA als gleichwertig zu einer eigenen akzeptiert wird. Die Policies werden mit OIDs identifiziert. Ausführliche Informationen zum Thema Policy Mapping gibt es in Abschnitt 29.4.2. Das PKIX-Profil lässt die Verwendung dieser Erweiterung zu (als unkritisch markiert), schreibt sie aber nicht vor, Common PKI ebenfalls nicht.
- *Subject Alternative Name*: Der X.509-Standard sieht im Namensfeld für den Zertifikatsbesitzer einen X.500-Namen vor. Im Erweiterungsfeld Subject Alternative Name kann die CA einen weiteren Namen oder eine sonstige Kennung des Zertifikatsbesitzers speichern. Gängig ist es etwa, an dieser Stelle eine E-Mail-Adresse, IP-Adresse oder Webadresse abzulegen. Manche CAs lassen das eigentliche X.509-Namensfeld sogar leer und verwenden stattdessen nur die Subject-Alternative-Name-Erweiterung. Diese muss in einem solchen Fall als kritisch markiert werden. Das PKIX-Profil lässt die Nutzung dieser Erweiterung auf diese Weise zu. Common PKI lässt die Verwendung dieses Felds offen.
- *Issuer Alternative Names*: X.509 verlangt, dass eine CA ihren Namen in einem Zertifikat als X.500-Name angibt. In der Erweiterung Issuer Alternative Name hat die CA die Möglichkeit, weitere Namen und Kennungen zu nennen. Auch hier sieht das PKIX-Profil die Möglichkeit vor, den X.500-Namen leer zu lassen und stattdessen nur den Namen in dieser Erweiterung zu verwenden. Common PKI lässt die Verwendung weitgehend offen.
- *Subject Directory Attributes*: In dieser Erweiterung können X.500-Attribute des Inhabers aus einem Verzeichnisdienst stehen. PKIX empfiehlt, auf diese Erweiterung zu verzichten. Gemäß Common PKI können darin Angaben aufgeführt werden, die alternativ auch in den Common-PKI-spezifischen Erweiterungen stehen können.
- *Basic Constraints*: Diese Erweiterung, die übersetzt »grundlegende Einschränkungen« heißt, ist nur in CA-Zertifikaten von Bedeutung. Darin wird festgelegt, wie viele Stufen im Zertifizierungsbaum unterhalb dieser CA vorkommen dürfen (siehe auch Abschnitt 29.4.2 zum Thema Policy Mapping). Das PKIX-Profil schreibt vor, dass diese Erweiterung für CA-Zertifikate verwendet und als kritisch markiert wird, Common PKI ebenso.

- *Name Constraints*: Dieses Feld (»Namenseinschränkungen«) wird nur für CA-Zertifikate benutzt. Es gibt Einschränkungen für die Namen an, die die CA beim Ausstellen von Zertifikaten beachten muss. PKIX ermöglicht die Verwendung dieser Erweiterung (als kritisch markiert), Common PKI dagegen verbietet sie.
- *Policy Constraints*: Diese Erweiterung, die übersetzt »Policy-Einschränkungen« heißt, ist ebenfalls nur für CA-Zertifikate vorgesehen. Ihr Zweck wird in Abschnitt 29.4.2 beschrieben. Das PKIX-Profil empfiehlt, diese Erweiterung zu verwenden (unkritisch), Common PKI lässt diesen Punkt offen.
- *Inhibit Any-Policy*: Dieses Feld spielt im Zusammenhang mit Policy Mapping eine Rolle. Siehe Abschnitt 29.4.2.
- *Freshest CRL*: Diese stets nichtkritische Erweiterung wird auch als »Delta CRL Distribution Point« bezeichnet. Dadurch wird auch der Zweck dieses Felds klar: Die CA verweist damit auf einen oder mehrere Stellen im Netz, wo eine aktuelle Delta-Sperrliste zum Download bereit steht. Wenn eine CA mit Delta-Sperrlisten arbeitet, ist die Verwendung des Freshest-CRL-Felds eine sinnvolle Sache.
- *Private Internet Extensions*: Diese Erweiterung ermöglicht es einer CA, weitere Informationen in einem Zertifikat abzulegen. Noch ist die genaue Nutzung dieses Felds unklar.

Mit diesen zusätzlichen Zertifikatsfeldern kann eine CA zweifellos aus dem Vollen schöpfen. Wie wir gleich sehen werden, gibt es jedoch noch mehr Erweiterungen, die von bestimmten Implementierungen verwendet werden.

27.3 Weitere X.509-Profile

Die in PKIX und Common PKI beschriebenen Profile machen nicht nur Aussagen darüber, wie die Standardfelder und die Standarderweiterungen von X.509v3-Zertifikaten verwendet werden. Vielmehr werden darin auch einige zusätzliche Erweiterungen beschrieben. Davon abgesehen bietet der X.509-Standard sogenannte Attributzertifikate.

27.3.1 Die PKIX-Erweiterungen

Der PKIX-Standard RFC 5280 fügt folgende Erweiterungen zum X.509v3-Format hinzu [RFC5280]:

- *Authority Information Access*: In dieser stets nichtkritischen Erweiterung kann eine CA auf zusätzliche Dienste oder Informationen verweisen, die sie anbietet. Die wichtigste Anwendung dieses Felds besteht darin, auf einen OCSP-Responder zu verweisen, den die CA betreibt und an dem Alice den

Sperrstatus des Zertifikats abfragen kann. Theoretisch könnte die CA im Authority-Information-Access-Feld auch auf einen CRL Distribution Point verweisen. Dies ist jedoch nicht vorgesehen, da es dafür ein eigenes Zertifikatsfeld gibt. Common PKI hat die Erweiterung Authority Information Access übernommen, schreibt deren Verwendung aber nicht zwingend vor.

- *Subject Information Access*: Diese Erweiterung verweist auf Dienste und Informationen, die der Zertifikatsinhaber anbietet. Ist das Zertifikat auf eine CA ausgestellt, können dies dieselben Informationen sein, die im Authority-Information-Access-Feld der von dieser CA ausgestellten Zertifikate stehen. Interessant ist diese Erweiterung auch, wenn der Zertifikatsinhaber ein Zeitstempeldienst ist. In diesem Fall kann das Feld dessen Webadresse enthalten. Für Endanwender wie Alice und Bob ist das Subject-Information-Access-Feld dagegen kaum von Interesse.

Ein weiterer RFC führt zusätzlich folgende zwei Erweiterungen ein [RFC3739]:

- *Biometric Information*: Darin wird der Hashwert eines biometrischen Templates gespeichert.
- *Qualified Certificate Statements*: Diese Erweiterung erlaubt einen Verweis auf ein bestimmtes Signaturgesetz oder eine ähnliche Aussage.

27.3.2 Die Common-PKI-Erweiterungen

Common PKI definiert eine Reihe zusätzlicher Erweiterungen, die alle optional sind. Diese Erweiterungen sind vor allem im Zusammenhang mit dem deutschen Signaturgesetz relevant. Hier eine Liste der Erweiterungen:

- *Liability Limitation Flag*: Gibt an, dass ein Attributzertifikat existiert, das die Anwendbarkeit dieses Zertifikats einschränkt.
- *Date of Cert Gen*: In diesem Feld kann die CA angeben, wann sie ein Zertifikat erstellt hat. Das Erstellungsdatum muss nicht mit dem Beginn der Gültigkeit übereinstimmen.
- *Procuration*: Hier wird Alices Vertretungsmacht gegenüber einer Organisation (oder Person) angezeigt, die sie vertritt. Dieses Feld entspricht dem Kürzel »i. A.«, »i. V.« oder »ppa.« vor einer Unterschrift von Hand.
- *Admission*: Mit diesem Feld kann eine CA Zertifikatsinhaberin Alice eine Zulassung (etwa als Ärztin, Notarin, Rechtsanwältin, ...) bescheinigen.
- *Monetary Limit*: Hiermit kann eine CA festlegen, bis zu welchem Geldbetrag Alice mit ihrem Zertifikat geradestehen muss.
- *Declaration of Majority*: In dieser Erweiterung kann angezeigt werden, ob Zertifikatsinhaberin Alice volljährig ist.

- *Restriction*: Mit diesem Feld kann eine CA verfügen, dass Zertifikatsinhaberin Alice ihren Signaturschlüssel nur für bestimmte Fälle verwenden darf. Dadurch wird der Inhalt des Felds Key Usage weiter eingeschränkt.
- *Additional Information*: Hier können weitere Informationen untergebracht werden.
- *ICCSN*: In dieser Erweiterung kann die CA die Seriennummer von Alices Smartcard speichern.

Genauere Informationen zu diesen Erweiterungen sowie zum X.509v3-Profil von Common PKI finden sich in [Common].

27.3.3 Attributzertifikate

In manchen Fällen ist es sinnvoll, einige Felder eines digitalen Zertifikats in eine zusätzliche Datenstruktur auszulagern. Diese zusätzliche Datenstruktur wird meist von derselben CA signiert und ähnelt daher einem digitalen Zertifikat – sie enthält jedoch keinen öffentlichen Schlüssel. Eine solche Datenstruktur wird **Attributzertifikat** genannt. Um Attributzertifikate von den normalen Zertifikaten abzugrenzen, werden Letztere in diesem Zusammenhang auch als **Schlüsselzertifikate** bezeichnet. Der X.509-Standard beinhaltet ein Format für Attributzertifikate. Dieses Format ist weitgehend das gleiche wie bei anderen X.509-Zertifikaten, nur dass eben der öffentliche Schlüssel fehlt.

Das Auslagern von Feldern aus einem Schlüssel- in ein Attributzertifikat kann folgende Gründe haben:

- Ein Schlüsselzertifikat mit vielen Feldern wird oft zu groß, insbesondere wenn es auf einer Smartcard gespeichert werden soll. Durch das Auslagern von Feldern in ein Attributzertifikat bleibt die Größe des Schlüsselzertifikats im Rahmen.
- Ein Attributzertifikat kann eine andere (meist kürzere) Gültigkeitsdauer als das zugehörige Schlüsselzertifikat haben. Es kann auch unabhängig vom Schlüsselzertifikat gesperrt werden. Neue Attributzertifikate können bei Bedarf generiert werden, ohne das zugehörige Schlüsselzertifikat zu ändern.
- Ein Attributzertifikat kann unter Verschluss gehalten werden, während das zugehörige Schlüsselzertifikat veröffentlicht wird. Dies ist aus Sicht des Datenschutzes oft sinnvoll.

Im RFC 5280, der ein X.509-Profil für das Internet spezifiziert, werden Attributzertifikate ausgeklammert. Dafür gibt es mit RFC 3281 einen PKIX-Standard, der Attributzertifikaten und deren Anwendung im Internet gewidmet ist [RFC5755]. Eine Instanz, die Attributzertifikate ausstellt, wird darin auch als **Attribute Authority** (**AA**) bezeichnet. In der Regel ist die AA mit der CA identisch. Trotz aller Vorteile werden Attributzertifikate bisher recht selten verwendet.

Im Common-PKI-Standard spielen Attributzertifikate eine wichtigere Rolle als in PKIX. Attributzertifikate werden sogar im ansonsten sehr allgemein gehaltenen Signaturgesetz erwähnt. Hierbei gibt es Attributzertifikate nur mit zugehörigem Schlüsselzertifikat. Gemäß Common PKI kann ein Attributzertifikat weitgehend die gleiche Form wie ein Schlüsselzertifikat annehmen (nur dass eben kein Schlüssel enthalten ist). Name des Inhabers, Versionsnummer, Seriennummer und CA-Name müssen in jedem Attributzertifikat vorhanden sein. Besonders gut geeignet sind Attributzertifikate, um Erweiterungen wie *Declaration of Majority* (Volljährigkeit), *Procuration* (Vertretungsmacht) und *Restriction* (Nutzungsbeschränkungen) aus Schlüsselzertifikaten auszulagern.

27.3.4 X.509-Fazit

X.509 ist nicht gerade ein Glanzlicht unter den Krypto-Standards. Die erste Version war nicht praxistauglich, die zweite brachte kaum Verbesserungen. Die dritte X.509-Version tat dann wiederum des Guten zu viel und musste nachgebessert werden. Alle drei Versionen haben zudem die Eigenschaft, dass sie zu viel Raum für Interpretationen lassen, wodurch es viele X.509-Implementierungen gibt, die untereinander nicht vollständig kompatibel sind. Durch die zahlreichen unterschiedlichen Profile wird die Lage auch nicht besser.

Die Mängel von X.509 haben jedoch zwei einleuchtende Begründungen: Zum einen war X.509 einer der ersten Krypto-Standards überhaupt und enthält daher einige Anfängerfehler. Zum anderen war X.509 zunächst nur für die Nutzung im Zusammenhang mit X.500 geplant. X.509 für andere Anwendungen einzusetzen, stellt im Grunde einen Missbrauch des ursprünglichen Standards dar. So gesehen ist X.509 eine Altlast, mit der man leben muss. Vieles wäre einfacher, wenn man den Standard noch einmal von Grund auf neu entwickeln könnte. Da es jedoch schon viele Implementierungen gibt, kommt dies nicht infrage. Trotz aller Nachteile ist X.509 heute einer der wichtigsten Krypto-Standards überhaupt. X.509-Zertifikate werden Ihnen daher noch öfter in diesem Buch begegnen.

27.4 PGP-Zertifikate

Wie bereits erwähnt, ist OpenPGP als konkurrierender Ansatz zu PKIX zu werten [RFC2440]. Dies zeigt sich auch an den digitalen Zertifikaten. OpenPGP verwendet keine X.509-Zertifikate, sondern ein vollständig eigenes Zertifikatsformat (**OpenPGP-Zertifikat** oder **PGP-Zertifikat** genannt).

27.4.1 OpenPGP-Pakete

OpenPGP arbeitet mit Dateneinheiten, die Pakete (Packets) genannt werden. Es gibt 14 unterschiedliche Pakettypen, die jeweils in einzelne Felder aufgeteilt und mit einer Nummer (Tag) bezeichnet sind. So gibt es beispielsweise einen Pakettypen für geheime Schlüssel (*Secret Key Packet*) und einen weiteren Pakettypen für symmetrisch verschlüsselte Daten (*Symmetrically Encrypted Data Packet*). Ersterer hat den Tag 5, letzterer den Tag 9. Für digitale Zertifikate sind drei weitere Pakettypen relevant, die im Folgenden beschrieben werden.

Signature Packet (Tag 2)

Ein *Signature Packet* ist ein Datensatz, der eine digitale Signatur sowie einige Zusatzinformationen enthält. Der OpenPGP-Standard [RFC2440] kennt mehrere Versionen davon, die aktuelle ist die Version 4 (nicht zu verwechseln mit der Versionsnummer der Software PGP). Ein *Signature Packet* umfasst in Version 4 folgende Felder:

- *Version Number*: Dies ist die Versionsnummer, in diesem Fall also 4.
- *Signature Type*: Dieses Feld gibt an, zu welchem Zweck die Signatur erstellt worden ist. Der OpenPGP-Standard nennt 13 verschiedene Werte, die dieses Feld annehmen kann, von einer einfachen Signatur für binäre Inhalte bis zu einer Zeitstempelsignatur.
- *Public Key Algorithm*: Hier wird angegeben, welches Verfahren zum Erstellen der Signatur verwendet wurde. OpenPGP fordert die Unterstützung von DSA, empfiehlt die Unterstützung von RSA und lässt weitere Algorithmen zu.
- *Hash Algorithm*: Hier wird die verwendete kryptografische Hashfunktion genannt. OpenPGP fordert die Unterstützung von SHA-1 und empfiehlt die Unterstützung von MD5 (aus Kompatibilitätsgründen).
- *Count for Hashed Subpacket Data*: Dieses Feld gibt die Länge des folgenden Felds an. Es dient einer einfacheren Verarbeitung eines *Signature Packet* und hat keine kryptografische Bedeutung.
- *Hashed Subpacket Data*: In diesem Feld steht eine Liste von Zusatzinformationen zur Signatur. OpenPGP bietet hierfür eine ganze Reihe von Möglichkeiten an, darunter den Ersteller und die Erstellungszeit der Signatur.
- *Count for Unhashed Subpacket Data*: Dieses Feld gibt die Länge des folgenden Felds an. Es dient einer einfacheren Verarbeitung eines *Signature Packet* und hat keine kryptografische Bedeutung.
- *Unhashed Subpacket Data*: Auch in diesem Feld können Zusatzinformationen zur digitalen Signatur stehen. Signiererin Alice kann sich aussuchen, ob sie derartige Daten lieber hier oder in die Hashed Subpacket Data ablegt. Der Unterschied besteht darin, dass das Feld Hashed Subpacket Data in den

Hashwert (und damit in die Signatur) eingeht, Unhashed Subpacket Data jedoch nicht. Ersteres Feld kann Mallory daher nicht unbemerkt verändern, letzteres dagegen schon. Unhashed Subpacket Data hat also die Funktion einer unverbindlichen Zusatzinformation, die man bei Bedarf auch ändern kann, ohne die Signatur ungültig zu machen.

- *Left 16 bits of Signed Hash Value*: Dieses Feld enthält die ersten 16 Bit des Hashwerts, auf den sich die Signatur bezieht. Damit kann Bob in vielen Fällen eine falsche Signatur erkennen, bevor er sie verifiziert.
- *Signature*: Hier steht schließlich die eigentliche Signatur. Diese bezieht sich auf einen Hashwert, in den (in dieser Reihenfolge) die folgenden Daten eingehen: Hashwert der zu signierenden Nachricht, Felder dieser Datenstruktur bis einschließlich Hashed Subpacket Data. Man beachte, dass Alice zum Erstellen einer Signatur zweimal einen Hashwert generieren muss.

Die zu signierende Nachricht selbst ist nicht Bestandteil eines Signature Packet.

Public Key Packet (Tag 6)

Das Public Key Packet ist ein weiterer OpenPGP-Pakettyp. Ein solches enthält einen öffentlichen Schlüssel und einige Zusatzinformationen. Dazu gehören in der aktuellen Version 4 folgende Felder:

- *Version Number*: Dies ist die Nummer 4.
- *Time Key Was Created*: Dies ist der Zeitpunkt, zu dem der Schlüssel generiert wurde.
- *Public Key Algorithm*: Dies ist das Verfahren, zu dem der Schlüssel gehört, also beispielsweise DSA oder RSA.
- *Key Material*: Hier ist der Schlüssel abgelegt.

User ID Packet (Tag 13)

Ein User ID Packet besteht nur aus einem Feld. Dieses enthält die Kennung einer Person oder eines sonstigen Objekts. In aller Regel handelt es sich dabei um eine E-Mail-Adresse.

Public Subkey Packet (Tag 14)

Ein Paket dieses Typs hat genau denselben Aufbau wie ein *Public Key Packet*. Der darin gespeicherte öffentliche Schlüssel ist einem anderen öffentlichen Schlüssel zugeordnet, der in einem *Public Key Packet* gespeichert ist. Meist gibt es folgende Abgrenzung: Der Schlüssel im *Public Key Packet* wird für Signaturen verwendet, während der Schlüssel im zugehörigen *Public Subkey Packet* zum Verschlüsseln vorgesehen ist.

27.4.2 PGP-Zertifikatsformat

Ein PGP-Zertifikat ist aus den vier Pakettypen zusammengesetzt, die wir soeben betrachtet haben. Wenn wir annehmen, dass es sich um Bobs Zertifikat handelt, dann hat ein PGP-Zertifikat folgende Bestandteile:

- *Public Key Packet*: Dieses Paket enthält Bobs öffentlichen Schlüssel, eine Versionsnummer des Paketformats und die Entstehungszeit.
- *User ID Packet*: Dieses Paket enthält eine Kennung, die Bob identifiziert. In aller Regel ist es Bobs E-Mail-Adresse.
- Weitere *User ID Packets* (optional): Hier können weitere Kennungen stehen, etwa weitere E-Mail-Adressen Bobs.
- Zu jedem User ID Packet gibt es null oder mehr *Signature Packets* mit jeweils einer digitalen Signatur. Mit diesen Signaturen wird die Zugehörigkeit von Bobs öffentlichem Schlüssel zur jeweiligen Kennung (und damit zu Bob selbst) bestätigt. Eine solche Signatur kann von jeder beliebigen Person oder Organisation (etwa von Alice) angefertigt werden. Gibt es zu einer Kennung mehrere Signature Packets, dann stammen diese im Normalfall von unterschiedlichen Personen.
- *Subkey Packets* (optional): Hier können weitere öffentliche Schlüssel stehen, die Bob gehören.
- Pro Subkey Packet ein *Signature Packet*: Diese Signatur bezieht sich auf das jeweilige Subkey Packet und ist von Bob selbst angefertigt worden. Dazu hat er den privaten Schlüssel verwendet, der zum öffentlichen Schlüssel im Public Key Packet gehört.

Eine wichtige Rolle in einem PGP-Zertifikat spielt das Feld *Hashed Subpacket Data* in den Signature Packets. Dieses Feld wird genutzt, um den Ersteller und den Erstellungszeitpunkt der Signatur anzugeben. OpenPGP sieht für dieses Feld außerdem zahlreiche weitere mögliche Inhalte vor, die bei der Nutzung für ein digitales Zertifikat eine Rolle spielen. Dazu gehören das Ende der Gültigkeitsdauer der Signatur, eine Policy URL (verweist auf ein Policy-Dokument) und Key Flags (gibt den Verwendungszweck des Schlüssels an). Mehrere der Zertifikatsfelder aus X.509 haben somit eine OpenPGP-Entsprechung in Form eines Hashed-Subpacket-Data-Eintrags. Keine OpenPGP-Entsprechung hat dagegen das X.509 Policy Mapping (siehe auch Abschnitt 29.4.2). Dafür kann der Signierer eines PGP-Zertifikats jedoch die Signaturtypen verwenden, die ein Signature Packet bietet. Wie erwähnt, gibt es 13 verschiedene Signaturtypen, die OpenPGP festlegt. Vier davon sind speziell für digitale Zertifikate gedacht (wir nehmen an, Alice signiert ein Zertifikat mit Bobs Schlüssel):

- *Generic Certification*: Bei dieser Stufe sagt Alice überhaupt nichts darüber aus, ob und wie sie sich vergewissert hat, dass Bob tatsächlich der Besitzer des Schlüssels ist.
- *Persona Certification*: Bei dieser Stufe hat Alice nicht überprüft, ob Bob der Besitzer des Schlüssels ist (»Persona« ist ein englisches Wort für »Rolle«).
- *Casual Certification*: Bei dieser Stufe hat Alice zwar überprüft, dass Bob den Schlüssel besitzt. Es handelt sich dabei jedoch nur um eine grobe, flüchtige Überprüfung (was das genau heißt, ist nicht definiert).
- *Positive Certification*: Bei dieser Stufe hat Alice genau überprüft, ob Bob Besitzer des Schlüssels ist.

Die einzelnen Stufen sind recht schwammig definiert. Dies ist jedoch Absicht.

27.4.3 Unterschiede zu X.509

PGP- und X.509-Zertifikate folgen unterschiedlichen Philosophien. Dies wird klar, wenn wir uns die Unterschiede betrachten:

- Der wesentliche Unterschied zwischen X.509- und PGP-Zertifikaten besteht darin, dass bei Ersteren *ein* öffentlicher Schlüssel durch *eine* CA-Signatur an einen Anwendernamen gebunden wird. Bei PGP-Zertifikaten können dagegen in einem Zertifikat beliebig viele öffentliche Schlüssel und beliebig viele Signaturen vorkommen. Dieser Unterschied ist durch das Vertrauensmodell Web of Trust begründet. Er hat zur Folge, dass zu einem Schlüsselpaar oft sehr viele PGP-Zertifikate im Umlauf sind, die unterschiedliche Signaturen tragen.
- Viele Erweiterungsfelder von X.509v3 haben bei PGP-Zertifikaten keine Entsprechung. So gibt es bei PGP-Zertifikaten beispielsweise kein Feld, in dem auf eine Policy verwiesen wird, ganz zu schweigen von Policy Mapping.
- PGP-Zertifikate haben zwar weniger Felder als X.509-Zertifikate. Trotzdem ist der Umgang mit ihnen häufig komplexer. Dadurch, dass ein PGP-Zertifikat mehrere Schlüssel und Signaturen beinhalten kann, muss Anwender Bob stets überlegen, welcher Signatur er nun in welcher Situation trauen kann.

Obwohl PGP-Zertifikate für ein Web of Trust gemacht sind, werden sie in der Praxis häufig mit dem Vertrauensmodell Hierarchical Trust eingesetzt. Dies bedeutet, dass ein PGP-Zertifikat nur eine Signatur trägt, die von einer CA ausgestellt worden ist. Durch diesen »Missbrauch« geht zwar ein Teil der PGP-Idee verloren. Die typischen Vorteile von Hierarchical Trust wie das Durchsetzen einer Policy oder ein hohes Maß an Verbindlichkeit werden dadurch jedoch gewonnen.

27.5 CV-Zertifikate

Es gibt Szenarien, in denen eine Smartcard in der Lage sein muss, das digitale Zertifikat eines Kommunikationspartners zu prüfen. Vor allem im Zusammenhang mit Gesundheitskarten und elektronischen Ausweisen ist dies häufig der Fall. So kann es beispielsweise erwünscht sein, dass Alices Gesundheitskarte nur mit berechtigten Instanzen (z.B. Arzt-PC) kommuniziert, während alle anderen keine Chance haben sollen, auf die Karte zuzugreifen – selbst wenn Alice ihre PIN eingegeben und die Karte dadurch freigeschaltet hat. In ähnlicher Form kann es vorkommen, dass der Staat Kryptoland bestimmte Zugriffe auf Alices Ausweis-Chip (z.B. das Auslesen von Alices Adresse) nur staatlichen Prüfgeräten erlauben will – nichtstaatliche Geräte sollen dagegen selbst dann keinen Zugriff haben, wenn Alice ihre Karte eigenhändig einsteckt und ihre PIN eingibt. Der Hintergedanke hierbei ist meist folgender: Wenn Alice nach ihrer Gesundheitskarte gefragt wird, kann sie oft nicht unterscheiden, ob sie es mit einem berechtigten Arzt-PC oder um mit einem von Mallory betriebenen Gerät zu tun hat; einem Ausweis-Prüfgerät sieht sie ebenfalls nicht unbedingt an, ob dieses die geforderte Berechtigung hat. Mit digitalen Zertifikaten lässt sich dieses Problem lösen: Wenn jedes Gerät mit der entsprechenden Zugriffsberechtigung ein Zertifikat erhält und der Kartenchip dieses prüfen kann, dann ist die Gefahr, dass Alices Karten mit einem unberechtigten Gerät kommuniziert, gebannt.

Allerdings ist das Überprüfen eines X.509-Zertifikats relativ komplex. Ein entsprechendes Programm muss das Vorhandensein bestimmter Erweiterungen prüfen, auswerten, ob diese kritisch oder unkritisch sind, und schließlich die Inhalte der einzelnen Felder interpretieren. Auf einem PC oder Server ist normalerweise genügend Speicherplatz vorhanden, um den dazu notwendigen Code abzulegen. Auf einer Smartcard, deren Speicherangebot in KByte gemessen wird, wird es dagegen eng. Mit anderen Worten: Das X.509-Format ist für die Verarbeitung auf einer Smartcard ungeeignet. Ein einfacheres Format ist notwendig.

Der Chipkarten-Standard ISO/IEC 7816-8 beschreibt ein Zertifikatsformat, das speziell für die Auswertung auf Smartcards geschaffen wurde [ISO7816-8]. Zertifikate, die diesem Format entsprechen, werden als **CV-Zertifikate** bezeichnet (CV steht für card-verifiable). Das Format ist nicht mit X.509 kompatibel. CV-Zertifikate haben eine sehr einfache Syntax und benötigen wenig Speicherplatz. In einem CV-Zertifikat sind in der einfachsten Form lediglich die ausstellende CA, der Zertifikatsinhaber, mit dem Zertifikat verbundene Zugriffsrechte, der öffentliche Schlüssel des Zertifikatsinhabers sowie der Gültigkeitszeitraum vermerkt. Zur Signatur kann eine besonders platzsparende Variante des RSA-Verfahrens (mit Message Recovery, siehe Abschnitt 12.4) verwendet werden. CV-Zertifikate werden beispielsweise im neuen deutschen Personalausweis und in der deutschen elektronischen Gesundheitskarte eingesetzt.

28 PKI-Prozesse im Detail

In diesem Kapitel nehmen wir einige Prozesse innerhalb einer PKI unter die Lupe: Anwender-Enrollment, Recovery und den Abruf von Sperrinformationen.

28.1 Anwender-Enrollment

Alice möchte sich gerne ein Zertifikat von einer CA ausstellen lassen. Den Vorgang, der die Anmeldung, das Generieren des Schlüsselpaars und das Signieren des Zertifikats umfasst, nennt man **Enrollment**. Wie ein Enrollment abläuft, dafür gibt es verschiedene Möglichkeiten. In allen Fällen kann man jedoch drei Schritte unterscheiden.

28.1.1 Schritt 1: Registrierung

Bevor Alice von einer CA ein Zertifikat erhält, muss die CA sie erst einmal kennen. Alice muss sich also bei der CA registrieren lassen. Dazu gibt es verschiedene Varianten.

Die persönliche Variante

Wenn Alice ihr Zertifikat bei einem öffentlichen Trust Center beantragt, dann gibt sie ihre Personalien in der Regel selbst an. Dies kann persönlich, per Post oder online erfolgen. Je nach Sicherheitsanforderungen kann der CA-Betreiber dazu Alices Ausweis oder eine Kopie davon zur Authentifizierung verlangen. Für weniger sicherheitskritische Fälle genügt es oft auch, wenn Alice belegt, dass sie in Besitz der E-Mail-Adresse ist, die im Zertifikat erwähnt wird.

Die Identity-Management-Variante

Bekommt Alice ihr Zertifikat von der CA ihres Arbeitgebers Krypt & Co., dann sind ihre Personalien in der Regel bereits in einer Datenbank der Personalabteilung gespeichert. Krypt & Co. muss also die vorhandenen Mitarbeiterdaten an die CA übermitteln. Wie man sich leicht klar macht, gibt es eine solche Aufgabenstellungen auch bei anderen Computersystemen: Zum Anlegen eines E-Mail-Accounts, einer Telefonnummer oder eines Intranetzugangs wird das Unternehmen ebenfalls die bereits vorhandenen Daten nutzen. Diese Überlegungen machen deutlich, warum viele Unternehmen das Thema PKI-Enrollment längst in einem größeren Zusammenhang sehen. Das Schlagwort in diesem Zusammenhang heißt **Identity Management**. Ziel des Identity Management ist es, Daten und Berechtigungen von Anwendern plattformübergreifend zu verwalten. Im Idealfall sieht das so aus:

- Wenn Alice von der Firma Krypt & Co. eingestellt wird, dann muss der dortige IT-Administrator ihre Daten nur einmal eingeben, und zwar in das Identity-Management-System. Dieses sorgt anschließend über entsprechende Schnittstellen zu anderen IT-Systemen dafür, dass Alice eine E-Mail-Adresse, eine Telefonnummer und einen Intranetzugang (und eventuell noch mehr) erhält. Im gleichen Zug wird sie auch automatisch bei der CA registriert.
- Wenn sich Alices Nachname oder ihre Position im Unternehmen ändert, dann vermerkt der IT-Administrator diese Änderung im Identity-Management-System. Dieses sorgt anschließend – wiederum über die entsprechenden Schnittstellen – dafür, dass sich (falls notwendig) auch Alices E-Mail-Adresse, ihre Telefonnummer und ihr Intranetzugang ändern. Innerhalb desselben Prozesses wird gegebenenfalls auch die CA darüber informiert, dass Alice ein neues Zertifikat benötigt.

- Wenn Alice die Firma Krypt & Co. verlässt, dann erfordert dies wiederum nur eine einzige Aktion des IT-Administrators. Dieser löscht Alice im Identity-Management-System, woraufhin dieses automatisch eine Löschung der E-Mail-Adresse, der Telefonnummer und des Intranetzugangs veranlasst. Gleichzeitig beantragt das Identity-Management-System die Sperrung von Alices Zertifikat bei der CA.

Eine derartige Einbindung der PKI in das Identity Management bringt für ein Unternehmen enorme Vorteile mit sich. Der Identity-Management-Boom, der seit einigen Jahren zu verzeichnen ist, hat daher auch dem Thema PKI einen erheblichen Aufschwung beschert. Da ein Identity-Management-System nebenbei auch für einheitliche Authentifizierungsvorgänge auf unterschiedlichen Plattformen sorgen kann, ist Identity Management auch eine wesentliche Triebkraft für die Credential-Synchronisation (siehe Abschnitt 22.1). Identity Management ergänzt sich zudem ideal mit Single Sign-On: Während Ersteres den Administratoren den Umgang mit unterschiedlichgen Serversystemen erleichtert, erfüllt Letzteres den gleichen Zweck für die Anwender.

28.1.2 Schritt 2: Zertifikate-Generierung

Kennt die CA erst einmal Alices Personalien, dann kann sie mit der Generierung eines Zertifikats beginnen. Die CA kann hierbei unmittelbar nach der Registrierung die Zertifizierung automatisch durchführen oder auf einen entsprechenden Anstoß des PKI-Administrators warten.

Variante mit RA

Wenn sich Alice an einer RA persönlich registrieren lässt, dann bietet es sich an, die Zertifikats-Generierung im gleichen Zug vorzunehmen. Auch wenn Alices Daten der CA schon bekannt sind, kann diese Alice bitten, persönlich an der RA zu erscheinen, um sich an der Zertifikats-Generierung zu beteiligen. An dieser Stelle stellt sich die Frage, ob Alice ihr Schlüsselpaar selbst generieren soll oder ob die RA bzw. die CA dies übernimmt. Wenn Alice der CA misstraut, dann ist eine selbst durchgeführte Generierung in der Regel die sicherere Methode. Andererseits wird ein Key Recovery später nur dann möglich sein, wenn die RA oder CA Alices privaten Schlüssel kennt und ihn speichern kann.

Wird das Schlüsselpaar von der RA oder von Alice selbst generiert, dann muss die RA dieses an die CA schicken. Dazu kann der Registrator einen signierten Zertifizierungsantrag (Certification Request) verwenden. Hat die CA Alices öffentlichen Schlüssel erhalten (bzw. hat sie ihn selbst generiert), dann stellt sie einen entsprechenden Datensatz zusammen, den sie durch eine Signatur zu einem digitalen Zertifikat macht. Dieses digitale Zertifikat stellt die CA im Verzeichnisdienst zum Download bereit.

Variante ohne RA

Den Anstoß zur Zertifikate-Generierung kann Alice auch selbst geben. Dies geht etwa über eine Webschnittstelle, welche die CA zur Verfügung stellt. In diesem Fall muss sich Alice über die Webseite gegenüber der CA authentisieren. Zur Authentisierung nutzt sie beispielsweise ein Passwort, das ihr die CA in einem versiegelten Brief (**PIN-Brief**) zugeschickt hat. Hat die CA Alices Zertifikat generiert, dann wird dieses über den Zertifikateserver den anderen Anwendern zugänglich gemacht.

28.1.3 Schritt 3: PSE-Übergabe

Wenn Alice ihr Schlüsselpaar nicht selbst generiert hat, dann muss die CA Alice das PSE (siehe Abschnitt 26.5.1) in einem zuvor festgelegten Prozess zukommen lassen.

Variante mit RA

Wenn Alice eine RA aufsucht, dann kann ihr der Registrator das PSE mit dem neuen privaten Schlüssel persönlich überreichen. Handelt es sich um ein Hardware-PSE (also etwa eine Smartcard), dann übergibt der Registrator dieses. Ist der private Schlüssel dagegen in einem Software-PSE gespeichert, dann erhält Alice eine CD oder einen Memorystick, auf dem der private Schlüssel verschlüsselt abgelegt ist. Bei einem Roaming-PSE erhält Alice die Webadresse, unter der dieses zugänglich ist. In allen Fällen muss die RA Alice zudem das Passwort (bzw. die PIN) zukommen lassen. Dies kann dadurch erfolgen, dass sich Alice selbst ein Passwort ausdenken und es an einem Terminal eintippen kann. Alternativ kann auch die RA das Passwort festlegen und auf einem Stück Papier an Alice übergeben.

Variante ohne RA

Natürlich kann die CA Alice eine Smartcard auch per Post zuschicken. Aus Sicherheitsgründen sollte die PIN hierbei getrennt versandt werden (in einem PIN-Brief). Wird ein Software-PSE verwendet, dann kann die CA dieses auch per E-Mail an Alice versenden (beispielsweise als PKCS#12-Datei). Auch hier sollte die CA Alice das PSE-Passwort in einem zusätzlichen PIN-Brief per Post zuschicken. Bei einem Roaming-PSE reicht eine Webadresse und ein PIN-Brief.

28.1.4 Enrollment-Beispiele

Nachdem wir uns die drei Schritte eines PKI-Enrollments angeschaut haben, wollen wir diese nun zu einem vollständigen Prozess zusammensetzen. Dafür gibt es viele verschiedene Möglichkeiten. Ein paar Beispiele schauen wir uns an.

Beispiel 1: Dezentrale Registrierung mit Smartcard

In einem Unternehmen (Krypt & Co.) könnte ein Enrollment wie folgt ablaufen.

1. Nachdem Alice ihren Arbeitsvertrag bei Krypt & Co. unterschrieben hat, trägt ein Mitarbeiter der Personalabteilung Alices Daten in die Personalverwaltungssoftware ein.
2. Das Identity-Management-System von Krypt & Co. liest Alices Daten aus der Personalverwaltungssoftware und gibt sie an andere Computersysteme weiter. Neben dem Mailserver und der Telefonanlage erhält auch die CA Alices Personalien.
3. An ihrem ersten Arbeitstag sucht Alice die RA des Unternehmens auf (dies ist beispielsweise ein Raum in der IT-Abteilung). In der RA verbindet sich der Registrator mit der CA. Dort hat er Alices Daten bereits vorliegen. Der Registrator nimmt nun einen Smartcard-Rohling und steckt ihn in den Leser des RA-Rechners. Der Chip auf dem Kartenrohling generiert ein Schlüsselpaar und sendet den öffentlichen Schlüssel an die CA.
4. Die CA generiert ein Zertifikat mit Alices Namen und ihrer E-Mail-Adresse. Dieses Zertifikat wird in den Verzeichnisdienst gestellt.
5. Alice tippt auf dem RA-Rechner eine von ihr gewählte PIN für die Smartcard ein (natürlich so, dass sie dabei niemand beobachtet).
6. Der Registrator händigt Alice die Smartcard aus.

Alice kann nun mit der Smartcard arbeiten – also verschlüsseln, signieren und sich authentifizieren.

Beispiel 2: Zentrale Registrierung ohne Smartcard

Nehmen wir nun an, Alice will sich bei einem öffentlichen Trust Center ein digitales Zertifikat besorgen. Dies könnte so ablaufen:

1. Alice holt sich von der Webseite des Trust Centers ein Registrierungsformular, das sie sich ausdruckt. Sie füllt das Formular aus und schickt es zusammen mit einer Fotokopie ihres Personalausweises an das Trust Center.
2. Das Trust Center überprüft den Antrag und registriert Alice im positiven Fall bei der CA. Alice erhält nun einen Brief mit einem Passwort (PIN-Brief) zugeschickt.
3. Alice verbindet sich per Internet mit dem Trust Center und authentisiert sich dort mit dem ihr zugegangenen Passwort. Alices Software (z.B. ihr Webbrowser) generiert nun ein Schlüsselpaar. Den privaten Schlüssel speichert die Software in einem Software-PSE. Den öffentlichen Schlüssel schickt sie als Teil eines Zertifizierungsantrags über das Netz an die CA.
4. Die CA nimmt den öffentlichen Schlüssel und generiert ein Zertifikat, das dessen Echtheit bestätigt. Das Zertifikat wird im Verzeichnisdienst abgelegt.

Alice hat nun den privaten Schlüssel in ihrem Software-PSE vorliegen und kann ihn nutzen.

Beispiel 3: SCEP-Enrollment

Das **Simple Certificate Enrollment Protocol** (**SCEP**) ist ein Protokoll, das ein Enrollment über ein Computernetz unterstützt [LiMaMN]. Das Protokoll wurde von der Firma Cisco entwickelt (Gleiches gilt auch für das Vorgängerprotokoll CEP), wird jedoch inzwischen auch von zahlreichen anderen Herstellern unterstützt. Demnächst wird es voraussichtlich in Form eines RFC zum offiziellen Standard werden. Der Name des Protokolls ist etwas irreführend, denn SCEP dient nicht nur dem Enrollment. Vielmehr sieht es auch Protokollnachrichten für die Verteilung des öffentlichen CA- oder RA-Schlüssels, für die Sperrung eines Zertifikats sowie für das Herunterladen eines Zertifikats oder einer Sperrliste vor. An dieser Stelle soll uns jedoch nur das Enrollment interessieren.

SCEP ist ein Protokoll für die Kommunikation zwischen einer CA oder RA und einem Stück Hardware. Es wird vor allem für das Enrollment von Routern, VPN-Komponenten, Switches und ähnlichen Geräten verwendet. Wie der Name andeutet, handelt es sich um ein recht einfaches Protokoll, das wenig Flexibilität bietet. Zur Erklärung von SCEP wollen wir annehmen, dass eine CA ein Zertifikat für einen Router ausstellen will. Kämen etwa eine RA und ein Switch zum Einsatz, wäre der Ablauf derselbe.

Voraussetzung für ein SCEP-Enrollment ist, dass der Router das CA-Zertifikat kennt. Falls notwendig, kann er dieses über eine SCEP-Protokollnachricht anfordern, wobei der Administrator zusätzlich einen Hashwert des Zertifikats überprüfen sollte. Optional kann der Administrator des Routers mit der CA ein Passwort vereinbaren. SCEP kennt zwei Enrollment-Modi. Der einfachere heißt *Automatic Mode* und hat folgenden Ablauf:

1. Der Router generiert ein Schlüsselpaar und schickt eine Nachricht (*PKCSReq*) an den Server. Diese Nachricht enthält einen PKCS#10-Zertifizierungsantrag sowie optional ein Passwort.
2. Die CA überprüft den Antrag und generiert im positiven Fall ein Zertifikat. Falls kein Passwort zur Authentifizierung des Antrags verwendet wird, muss die CA auf anderem Weg sicherstellen, dass Angreifer Mallory sich kein Zertifikat erschleichen kann (dies kann etwa durch die persönliche Anwesenheit des Administrators erfolgen). Das neu generierte Zertifikat schickt die CA verschlüsselt und signiert im PKCS#7-Format an den Router zurück (*CertRep*).

Im zweiten Modus (*Manual Mode*) sendet die CA das Zertifikat nicht notwendigerweise in der zweiten Nachricht zurück, sondern schickt (gegebenenfalls mehrfach) eine Meldung, die besagt, dass die Generierung noch nicht abgeschlossen ist.

Wie Sie sehen, ist ein SCEP-Enrollment wirklich sehr simpel. Es erfüllt jedoch seinen Zweck und ermöglicht es Administratoren, ohne großen Aufwand ein Enrollment für eine Hardwarekomponente durchzuführen. Mit SCEP hat sich einmal mehr eine pragmatische Lösung durchgesetzt.

Beispiel 4: Auto-Enrollment

Ein weiterer pragmatischer Ansatz ist das sogenannte **Auto-Enrollment**. Diese Enrollment-Variante wurde zuerst von Microsoft als Funktion von Windows 2000 angeboten, wird inzwischen jedoch auch von anderen PKI-Lösungen unterstützt. Ein Auto-Enrollment ist vor allem für Unternehmen oder Behörden geeignet, die ihre eigene PKI betreiben. Es setzt voraus, dass Alice einen zentral administrierten Account hat, über den sie sich in das Betriebssystem einloggt. Diese Voraussetzung ist in praktisch jedem Unternehmen und jeder Behörde gegeben. Zudem muss die CA auf den Datenbestand des Anmeldeservers zugreifen können, über den Alices Login abläuft. CA und Anmeldeserver verschmelzen dadurch zu einer Komponente.

Eine weitere Voraussetzung für ein Auto-Enrollment ist, dass auf Alices PC ein spezieller Client (Auto-Enrollment-Client) vorhanden ist. Dieser Client kann in das Betriebssystem integriert sein, oder es kann sich um eine Zusatzsoftware handeln. In letzterem Fall bietet es sich an, das Softwareverteilungssystem von Krypt & Co. zu nutzen, um den Client aufzubringen. Wenn wir wieder annehmen, dass Alice ein Zertifikat von der CA ihres Arbeitgebers Krypt & Co. erhält, dann läuft ein Auto-Enrollment etwa so ab:

1. Ein Registrator schaltet Alice an der CA (bzw. am Anmeldeserver) für die Zertifikats-Generierung frei.
2. Wenn sich Alice das nächste Mal an ihrem PC einloggt, startet ihr Auto-Enrollment-Client automatisch die Generierung eines Schlüsselpaars und legt den privaten Schlüssel in einem Software-PSE ab.
3. Der Auto-Enrollment-Client schickt den öffentlichen Schlüssel an die CA (bzw. an den Anmeldeserver). Da sich Alice kurz zuvor mit ihrem Passwort oder einer sonstigen Authentifizierungsmethode am Anmeldeserver angemeldet hat, besteht eine sichere Verbindung.
4. Die CA generiert ein Zertifikat. Dieses leitet sie an den Zertifikateserver weiter, damit es dort abrufbar ist.

Die Sicherheit dieses Ablaufs ergibt sich dadurch, dass nur Alice sich an ihrem PC-Account einloggen und damit eine sichere Verbindung zum Anmeldeserver aufbauen kann. Das Einloggen ersetzt beim Auto-Enrollment also das persönliche Erscheinen an der RA oder das Hantieren mit einem PIN-Brief. Beachten Sie, dass das Enrollment für Alice in diesem Szenario weitgehend transparent abläuft – Alice muss nichts tun, außer sich wie gewohnt einloggen (deshalb heißt dieser Vorgang »Auto-Enrollment«). Eine RA ist hierbei nicht notwendig.

28.1.5 Zertifizierungsanträge

In einigen Enrollment-Varianten spielen **Zertifizierungsanträge** eine Rolle. Ein Zertifizierungsantrag ist ein Datensatz, der Alices Namen, ihren öffentlichen Schlüssel sowie gegebenenfalls weitere Angaben enthält und in der Regel digital signiert ist. Für Zertifizierungsanträge gibt es zwei gängige Standards: PKCS#10 und CRMF. Letzterer ist der deutlich jüngere Standard. Beide sind ausschließlich für X.509-Zertifikate gedacht und lassen offen, wer den Antrag entgegennimmt (CA oder RA). PKCS#10 und CRMF sind nicht miteinander kompatibel.

PKCS#10

PKCS#10 ist ein weiterer Standard aus der PKCS-Reihe [PKCS#10]. Er spezifiziert ein Format für Zertifizierungsanträge. Im Vergleich zu einigen anderen PKCS-Standards ist PKCS#10 recht simpel aufgebaut. Ein PKCS#10-Datensatz enthält den Namen des Antragsstellers (in diesem Fall Alice), die gewünschte X.509-Versionsnummer sowie den öffentlichen Schlüssel. Weitere mögliche Inhalte (Attribute) eines PKCS#10-Antrags sind Angaben über gewünschte X.509v3-Erweiterungsfelder und ein Sperrpasswort. Diese Attribute werden jedoch nicht im PKCS#10-Standard selbst, sondern in PKCS#9 beschrieben (PKCS#9 spezifiziert zudem Attribute für verschiedene andere Standards der PKCS-Familie).

Ein PKCS#10-Antrag muss von Alice selbst signiert sein (mit demjenigen privaten Schlüssel, dessen öffentliches Gegenstück im Antrag enthalten ist). Durch eine Verifizierung der Signatur kann die RA bzw. CA zum einen die Korrektheit des Zertifizierungsantrags überprüfen und zum anderen sicherstellen, dass Alice auch tatsächlich den zum Antrag passenden privaten Schlüssel besitzt (**Proof of Possession**). PKCS#10 ist nicht dazu vorgesehen, dass ein Registrator einen von Alice gestellten Zertifikatsantrag signiert, obwohl dies eine durchaus sinnvolle Einsatzmöglichkeit wäre.

Bei einem PKCS#10-Antrag ist eines zu beachten: Falls das von Alice generierte Schlüsselpaar ein RSA-Schlüsselpaar ist, bereitet das Erstellen einer Signatur keine Probleme, da sich RSA für digitale Signaturen nutzen lässt. Handelt es sich dagegen um ein Diffie-Hellman-Schlüsselpaar, dann ist die Sache nicht ganz so einfach, da Diffie-Hellman kein Signaturverfahren ist. Diesem Problem widmet sich der PKIX-Standard RFC 2875, der die zwei folgenden Proof-of-Possession-Verfahren (PoP) für Diffie-Hellman-Schlüssel spezifiziert [RFC2875]:

- **Statischer PoP**: Bei dieser Variante kommt keine digitale Signatur zum Einsatz, sondern lediglich eine kryptografische Hashfunktion (SHA-1). Der Standard sieht vor, dass Alice und die CA bzw. RA einen Diffie-Hellman-Schlüsselaustausch durchführen. Aus dem resultierenden gemeinsamen geheimen Schlüssel sowie aus dem Zertifizierungsantrag generiert Alice einen Hashwert, den sie anstelle der digitalen Signatur in den Antrag einfügt. Die CA bzw. RA berechnet nach Erhalt des Antrags auf dieselbe Weise wie Alice einen

Hashwert. Sind die beiden Hashwerte gleich, dann kann die CA bzw. RA daraus schließen, dass Alice tatsächlich den betreffenden privaten Diffie-Hellman-Schlüssel besitzt. Hierbei ist zu beachten, dass der statische PoP nach RFC 2875 nur für Fälle vorgesehen ist, in denen alle Anwender denselben Modulus *p* und dieselbe Basis *g* verwenden.

- **DLSS-PoP**: Falls jeder Anwender seine eigene Basis und seinen eigenen Modulus verwendet, können Alice und die CA bzw. RA auch ein Proof-of-Possession-Verfahren verwenden, das eine Signatur auf Basis des diskreten Logarithmus vorsieht (Diffie-Hellman-Schlüssel lassen sich für eine solche nutzen). Genauer gesagt, verwendet Alice eine Variante des DSA, um ihren Zertifizierungsantrag mit einem privaten Diffie-Hellman-Schlüssel zu signieren.

Noch gibt es keinen standardisierten Proof of Possession für ECC-Schlüssel. Dies sollte sich jedoch in nicht allzu ferner Zukunft ändern.

CRMF

CRMF (**Certificate Request Message Format**) ist neben PKCS#10 das zweite bedeutende Format für Zertifizierungsanträge. Es wird im PKIX-Standard RFC 4211 beschrieben [RFC4211]. Schon die RFC-Nummer deutet an, dass CRMF eng mit dem Certificate Management Protocol (CMP) verbunden ist, das die Nummer 4210 trägt. Man kann sogar sagen, dass CRMF ein ausgelagerter Bestandteil von CMP ist. Ein Blick in die CMP-Spezifikation verrät, dass an der entsprechenden Stelle auf CRMF verwiesen und dessen Einsatz empfohlen wird. Zwar unterstützt CMP auch PKCS#10, doch dies ist nur aus Kompatibilitätsgründen der Fall. CRMF ist etwas komplexer als PKCS#10. Das Format sieht drei Felder vor:

- *certReq*: In diesem Feld werden die Informationen übertragen, die in das Zertifikat aufgenommen werden sollen. Dazu gehören beispielsweise die X.509-Versionsnummer, der Name des Inhabers und die Gültigkeitsdauer. Alle diese Angaben sind optional – die CA kann die jeweiligen Inhalte der Zertifikatsfelder daher auch unabhängig vom Zertifizierungsantrag festlegen.
- *popo*: Dieses Feld mit dem schönen Namen enthält einen Proof of Possession (dies kann etwa eine digitale Signatur sein, siehe unten).
- *regInfo*: Dieses Feld enthält beliebige Zusatzinformationen und Anmerkungen.

Beachten Sie, dass es kein spezielles Feld für eine digitale Signatur gibt. Wenn der Zertifizierungsantrag sich auf einen Signaturschlüssel bezieht und signiert sein soll, dann steht die Signatur im popo-Feld. Alternativ kann dieses Feld auch leer bleiben (dann muss die CA oder RA auf andere Weise die Integrität des Antrags überprüfen). Auch wenn Alice einen Verschlüsselungsschlüssel oder einen Diffie-Hellman-Schlüssel zertifizieren lassen will, bietet CRMF jeweils mehrere Möglichkeiten (teilweise mit Signatur, teilweise ohne) an.

28.2 Recovery

Nachdem Alice von der CA ihres Arbeitgebers Krypt & Co. ein digitales Zertifikat und eine Smartcard erhalten hat, verschlüsselt sie alle wichtigen Daten damit. Dies geht einige Monate gut, doch dann passiert das Unvorhergesehene: Alice verliert ihre Smartcard. Nun kann sie ihre Daten nicht mehr entschlüsseln, und die Arbeit von Monaten ist dahin. Diese Situation zeigt, dass sich der Nutzen einer PKI schnell ins Gegenteil verkehren kann. Ein Unternehmen, das eine PKI aufbaut, muss sich daher stets Gedanken über eine Hintertür machen, die im Notfall einen Zugriff auf verschlüsselte Daten erlaubt. Man bezeichnet dies als **Recovery**.

Die Sache ist allerdings etwas komplizierter, als das eingangs erwähnte Beispiel glauben macht. In der Praxis muss Krypt & Co. beim PKI-Aufbau nämlich nicht nur eine verlorene Smartcard, sondern mehrere weitere Szenarien berücksichtigen, die ebenfalls eine Hintertür zu verschlüsselten Daten erfordern. Dabei ist es wohl überflüssig zu erwähnen, dass jede Hintertür eine potenzielle Sicherheitslücke darstellt. Doch die Schäden, die bei einem Verzicht darauf entstehen können, sind oft so hoch, dass viele PKI-Betreiber Sicherheitsbedenken hintanstellen. Im Folgenden gehen wir davon aus, dass Alice verschlüsselte Nachrichten erhält und dass ihr Kollege Paul (PKI-Administrator) in bestimmten Situationen Zugriff darauf benötigt. Unabhängig von den konkreten Umständen stehen Krypt & Co. immer zwei Vorgehensweisen zur Verfügung, um Paul Zugang zu Alices verschlüsselten Daten zu verschaffen:

- **Key Recovery**: In diesem Fall besitzt Paul eine Kopie von Alices privatem Schlüssel (siehe auch Abschnitt 24.1.8).
- **Message Recovery**: In diesem Fall muss der Absender einer Nachricht zur Verschlüsselung zwei öffentliche Schlüssel einsetzen. Einer davon gehört Alice, der andere gehört Paul. Dabei verschlüsselt der Absender nicht die ganze Nachricht doppelt, sondern nur den Sitzungsschlüssel (es kommt also ein Hybridverfahren zum Einsatz). Der Begriff »Message Recovery« hat in diesem Zusammenhang naturgemäß eine andere Bedeutung als bei den Signaturverfahren mit Message Recovery (siehe Abschnitt 12.4).

Bei allen Recovery-Szenarien spielt es eine wichtige Rolle, ob Alice eine Nachricht selbst verschlüsselt oder ob jemand anderer (etwa Bob) ihr etwas Verschlüsseltes zugeschickt hat. Ersterer Fall entspricht einer Dateiverschlüsselung, die Alice auf ihrem PC durchführt. Letzterer Fall ist beispielsweise gegeben, wenn Bob Alice eine verschlüsselte E-Mail schickt. Es bietet sich daher an, in diesem Zusammenhang zwischen *Dateiverschlüsselung* und *E-Mail-Verschlüsselung* zu unterscheiden. An dieser Stelle noch ein Wort zu digitalen Signaturen und zur Authentifizierung: Wenn Alice ihren Signatur- bzw. Authentifizierungsschlüssel verliert, ist kein Recovery notwendig. Es reicht aus, wenn Alice ein neues Schlüsselpaar erhält bzw. generiert und das alte Zertifikat gesperrt wird. Wir müssen das Thema Recovery daher nur im Zusammenhang mit Verschlüsselung betrachten.

28.2.1 Schlüsselverlust-Problem

Kommen wir zurück zur Ausgangssituation: Alice hat ihre Smartcard verloren und kann ihre Daten nicht mehr entschlüsseln. Eine ähnliche Situation entsteht, wenn Alices Smartcard kaputt geht oder wenn sie das Passwort ihres Software-PSE vergisst (bei einer Smartcard kann ein Administrator mit der SO-PIN eine neue PIN vergeben, doch bei einem Software-PSE geht dies nicht). Außerdem ist es denkbar, dass Alice aus Ärger über ihren Arbeitgeber ihre Smartcard absichtlich vernichtet. In jedem Falle können wir von einem Schlüsselverlust bzw. vom **Schlüsselverlust-Problem** sprechen. Wie man sich leicht klar macht, ist ein Schlüsselverlust nur dann ein Problem, wenn Daten über längere Zeit verschlüsselt gespeichert werden. Wenn es dagegen um die Verschlüsselung einer Kommunikationsverbindung (z. B. eines Telefongesprächs) geht, kann die Firma Krypt & Co. den verlorenen Schlüssel durch einen neuen ersetzen und benötigt daher keine Recovery-Mechanismen.

Key Recovery bei Schlüsselverlust

Key Recovery bei Schlüsselverlust setzt voraus, dass Krypt & Co. Kopien der privaten Verschlüsselungsschlüssel in einer Datenbank speichert. Hat Alice ihre Smartcard verloren, dann sucht sie Administrator Paul auf und erhält von diesem nach einem zuvor festgelegten Prozess eine neue Karte mit ihrem alten Schlüssel. Dieser Vorgang funktioniert sowohl bei Dateiverschlüsselung als auch bei E-Mail-Verschlüsselung. Natürlich muss sich Krypt & Co. genau überlegen, ob und unter welchen Bedingungen ein Key Recovery zulässig ist. In Abschnitt 24.1.8 gibt es ein paar Gedanken dazu.

Message Recovery bei Schlüsselverlust

Message Recovery bei Schlüsselverlust ist für die Dateiverschlüsselung gut geeignet, da Alice problemlos alle ihre Dateien mit zwei öffentlichen Schlüsseln verschlüsseln kann. Message Recovery bei E-Mail-Verschlüsselung ist dagegen meist nicht praktikabel. Es setzt voraus, dass der Absender einer E-Mail diese doppelt verschlüsselt (einmal für Alice und einmal für Paul). Leider kann man einen Absender in den seltensten Fällen zu einer solchen Doppelverschlüsselung zwingen, weshalb sich Krypt & Co. lieber von vornherein nach einer anderen Möglichkeit umsehen sollte.

Alternativen

Wenn Alice eingehende Mails nicht verschlüsselt abspeichert, kann sie auf Recovery für E-Mails verzichten. Falls sie ihre Smartcard verliert, erhält sie in diesem Fall einen neuen Schlüssel. Anschließend muss sie die Absender der letzten eingegangenen Mails bitten, ihre Nachrichten noch einmal zu schicken (mit ihrem

neuen Schlüssel verschlüsselt). Zum Key Recovery oder Message Recovery bei der Dateiverschlüsselung gibt es jedoch kaum eine Alternative. Allenfalls für weniger wichtige Daten kann man darauf verzichten.

28.2.2 Chef-Sekretärin-Problem

Der Chef von Krypt & Co. hat eine Sekretärin, die alle seine empfangenen E-Mails entschlüsseln können soll. Diese in der Praxis ausgesprochen häufig anzutreffende Situation führt zu einem weiteren Recovery-Szenario, das man als **Chef-Sekretärin-Problem** bezeichnet.

Key Recovery beim Chef-Sekretärin-Problem

Key Recovery heißt im Falle des Chef-Sekretärin-Problems, dass die Sekretärin eine Kopie des privaten Chef-Schlüssels erhält. Ob dies mit den Sicherheitsvorstellungen der Firma Krypt & Co. vereinbar ist, ist eine interessante Frage, die das Unternehmen selbst beantworten muss. Da Kryptografie nie ohne ein gewisses Maß an Pragmatismus funktioniert, haben viele Unternehmen keine Berührungsängste gegenüber einer solchen Lösung. Technisch möglich ist eine solche Schlüsselkopie jedenfalls, und das sowohl bei der Datei- als auch bei der E-Mail-Verschlüsselung.

Message Recovery beim Chef-Sekretärin-Problem

Um gegenüber seiner Sekretärin Message Recovery zuzulassen, muss der Chef jede Datei doppelt verschlüsseln – einmal für sich und einmal für die Sekretärin. Dies ist möglich und praktikabel. Anders sieht es dagegen bei der E-Mail-Verschlüsselung aus, denn hier muss der Absender jede E-Mail doppelt verschlüsseln. Kaum anzunehmen, dass sich jeder Kommunikationspartner des Chefs daran hält.

Alternativen

Es gibt nur zwei Möglichkeiten, im Zusammenhang mit dem Chef-Sekretärin-Problem Key Recovery und Message Recovery komplett zu vermeiden: Entweder der Chef entschlüsselt alles selbst oder er überlässt sein PSE komplett seiner Sekretärin.

28.2.3 Urlauber-Vertreter-Problem

Alice nimmt ein paar Tage Urlaub. Währenddessen soll ihre Kollegin Carol ihre E-Mails entgegennehmen. Durch eine entsprechende Weiterleitung ist zwar gewährleistet, dass Carol alle Mails enthält – sie kann für Alice bestimmte Mails jedoch nicht entschlüsseln. Dieses sogenannte **Urlauber-Vertreter-Problem** ähnelt dem Chef-Sekretärin-Problem, ist aber im Gegensatz zu diesem zeitlich begrenzt.

Außerdem hat das Chef-Sekretärin-Problem in der Praxis meist eine deutlich höhere Priorität, da es ja schließlich um eine Angelegenheit des Chefs geht.

Key Recovery beim Urlauber-Vertreter-Problem

Key Recovery bedeutet in diesem Fall, dass der Vertreter eine Schlüsselkopie des Urlaubers erhält. Obwohl dies technisch kein Problem ist, ist mir keine PKI bekannt, in der so etwas erlaubt ist – weder für die E-Mail-Verschlüsselung noch für die Dateiverschlüsselung. Im Falle eines Software-PSEs können es Anwender manchmal auch ohne Erlaubnis praktizieren, indem sie das PSE kopieren oder den Schlüssel exportieren. Allerdings lassen sich viele PSE-Implementierungen (etwa der Windows Keystore) so konfigurieren, dass dies nicht ohne weiteres möglich ist.

Message Recovery beim Urlauber-Vertreter-Problem

Message Recovery im Zusammenhang mit einem Urlauber heißt, dass der Absender einer E-Mail diese doppelt verschlüsseln muss – einmal für den Urlauber und einmal für den Vertreter. Eine solche Vorgehensweise ist jedoch gegenüber den Absendern kaum durchzusetzen, zumal die doppelte Verschlüsselung nur für einen begrenzten Zeitraum notwendig ist. Dateiverschlüsselung mit Message Recovery ist dagegen schon eher realistisch. Dazu muss der Urlauber alle fraglichen Dateien entsprechend doppelt verschlüsseln.

Alternativen

In der Praxis wird das Urlauber-Vertreter-Problem oft pragmatisch gelöst, indem der Urlauber seinem Vertreter seine Karte oder sein PC-Passwort überlässt. Dies ist zwar in so gut wie jeder PKI verboten, wird aber häufig praktiziert. Sinnvoller ist es natürlich, mit automatischen Antworten an den Absender zu arbeiten. Eine solche teilt dem Absender mit, dass der Urlauber derzeit nicht erreichbar ist und der Vertreter in dringenden Fällen zur Verfügung steht. Im Idealfall enthält die automatische Benachrichtigung sogar den öffentlichen Schlüssel des Vertreters oder einen Verweis darauf.

28.2.4 Virenscanner-Problem

Die Firma Krypt & Co. lässt Dateien und eingehende E-Mails nach Viren und anderer Malware scannen. Dabei ergibt sich ein Problem: Verschlüsselte Daten lassen sich nicht scannen (**Virenscanner-Problem**). In der Tat ist Content Security (so nennt man den Schutz vor Viren und anderen gefährlichen Inhalten) ein natürlicher Feind der Verschlüsselung. Da Content-Security-Maßnahmen weiter verbreitet und meist schon seit längerem im Einsatz sind, ist es fast immer die Verschlüsselung, die zurückweichen muss. Mir sind mehrere Kryptografie-Projekte

bekannt, die an dieser Frage gescheitert sind. Beim Aufbau einer PKI ist daher eine frühe Betrachtung des Virenscanner-Problems Pflicht.

Key Recovery beim Virenscanner-Problem

Das Virenscanner-Problem lässt sich über Key Recovery lösen. Dies bedeutet beispielsweise, dass der Virenscanner auf dem Mailserver oder in der Firewall eine Kopie von Alices Schlüssel benötigt. Zahlreiche Content-Security-Produkte unterstützen eine derartige Entschlüsselung vor dem Scan. Zwar ist es aus Sicht eines Kryptografen völlig unbefriedigend, wenn jede E-Mail auf einem Server entschlüsselt wird, doch in vielen Fällen bleibt keine andere Wahl. Ähnliches gilt für die Dateiverschlüsselung.

Message Recovery beim Virenscanner-Problem

Message Recovery für E-Mails zur Lösung des Virenscanner-Problems ist zwar möglich, jedoch einmal mehr unpraktikabel, da in der Praxis nicht jeder Absender eine Mail an Alice doppelt verschlüsseln wird. Wenn es um Dateiverschlüsselung geht, sieht der Fall dagegen anders aus. Hier kann Krypt & Co. eine doppelte Verschlüsselung für alle Dateien vorsehen, die dem Virenscanner Zugang gewährt.

Alternativen

Die wichtigste Alternative zu Recovery-Maßnahmen für den Virenscan besteht darin, nur auf der Anwenderplattform zu scannen. Im Falle von verschlüsselten Dateien ist dies sicherlich durchführbar. Zentrale Content-Security-Maßnahmen auf dem Mailserver oder in der Firewall zu umgehen, widerstrebt dagegen den meisten Netzbetreibern. Es gibt eine weitere Möglichkeit, die sich **Encrypted Mail Virus Scan** (**EMVS**) nennt. Diese sieht ein Protokoll mit folgendem Ablauf vor:

1. Bob schickt eine verschlüsselte E-Mail an Alice (diese arbeitet bei Krypt & Co.).
2. Der Mailserver von Krypt & Co. fängt die verschlüsselte E-Mail ab und extrahiert den verschlüsselten Sitzungsschlüssel. Diesen leiter er an Alice weiter.
3. Alice (bzw. ihr E-Mail-Programm) entschlüsselt den Sitzungsschlüssel und sendet ihn an den Mailserver zurück.
4. Der Mailserver entschlüsselt die Mail mit dem Sitzungsschlüssel und untersucht sie auf Viren. Findet er nichts Verdächtiges, dann leitet er die verschlüsselte Mail an Alice weiter.
5. Alice empfängt die E-Mail und entschlüsselt sie.

EMVS ist patentiert. Wer das Protkoll implementieren will, sollte den Patentinhaber kontaktieren [Hoffme].

28.2.5 Geht es auch ohne Recovery?

Die vorhergehenden Absätze haben gezeigt, dass das Thema Recovery recht komplex ist und deutlich mehr umfasst als nur das Szenario eines Anwenders mit verlorenem Schlüssel. Trotzdem gibt es in der Praxis viele PKIs, die ganz ohne Recovery auskommen. Dies liegt zum einen daran, dass viele PKI-Anwendungen – etwa Authentifizierung und digitale Signaturen – ohne Recovery auskommen. Auch bei E-Mail kann man oft darauf verzichten – vorausgesetzt Krypt & Co. begnügt sich mit einem Virenscan auf der Anwenderplattform, verzichtet auf Urlauber-Vertreter- sowie Chef-Sekretärin-Lösungen und sieht nicht das verschlüsselte Ablegen von E-Mails vor. Wenn es um die Datei-Verschlüsselung geht, interessieren sich viele Unternehmen ohnehin nur für Dateien auf dem Laptop – Dateien auf dem Server werden dagegen meist als sicher angenommen. Wenn Alice die Daten auf ihrem Laptop regelmäßig sichert (das sollte sie ohnehin tun, schließlich kann ein solches Gerät jederzeit kaputt oder verloren gehen) und die jeweiligen Backups nicht verschlüsselt sind, kann sie auch hier auf Recovery verzichten.

28.3 Abruf von Sperrinformationen

Eine CA muss in der Lage sein, Zertifikate für ungültig zu erklären (zu sperren). Dies ist beispielsweise dann notwendig, wenn ein Anwender seine Smartcard mit dem privaten Signaturschlüssel verloren hat. Wenn ein Mitarbeiter der Firma Krypt & Co. das Unternehmen verlässt, dann muss die firmeninterne CA dessen Zertifikat ebenfalls sperren. Eine Sperrung hat allerdings nur dann einen Sinn, wenn Anwenderin Alice sich bei Bedarf darüber informieren kann, ob ein bestimmtes Zertifikat eines anderen Anwenders gesperrt ist. Ist dies der Fall, dann darf sie dieses Zertifikat nicht mehr verwenden. Damit Alice einen leichten Zugang zu Sperrinformationen hat, werden diese sinnvollerweise über den Zertifikateserver abrufbar gemacht. Zertifikate und Informationen über deren Sperrung gibt es daher meist an derselben Stelle. Für das Vorgehen beim Abruf von Sperrinformationen stehen mehrere Möglichkeiten zur Verfügung, die wir im Folgenden betrachten.

28.3.1 Sperrlisten

Der gängigste Ansatz zum Abruf von Sperrinformationen sind sogenannte **Sperrlisten**. Diese werden auch als **Certificate Revocation Lists** (**CRLs**) bezeichnet. Eine Sperrliste ist eine digitale Liste, auf der die Seriennummern der gesperrten Zertifikate aufgeführt sind und die (meist von einer CA) digital signiert ist. Wenn Alice sich in regelmäßigen Abständen (etwa einmal am Tag) die jeweils aktuelle Sperrliste herunterlädt, dann weiß sie, welche Zertifikate sie nicht mehr verwenden darf. Eine Sperrliste wird von der CA regelmäßig oder bei Bedarf durch eine aktualisierte Version ersetzt. Sperrlisten gibt es in unterschiedlichen Variationen:

- *Komplette Sperrlisten*: Die einfachste Form einer Sperrliste ist die **komplette Sperrliste**. In einer solchen veröffentlicht ein Trust Center die Seriennummern aller Zertifikate, die gesperrt sind. Dies kann in einer großen PKI sehr unhandlich werden. So kann es durchaus vorkommen, dass eine komplette Sperrliste mehrere Megabyte groß wird. Wenn alle PKI-Anwender sich einmal am Tag eine solche Datenmenge auf ihren Rechner laden, dann lohnt es sich, nach Alternativen zu suchen.
- *Teilsperrlisten*: Komplette Sperrlisten lassen sich vermeiden, indem die CA die Seriennummern der gesperrten Zertifikate auf mehrere Sperrlisten verteilt. Man spricht dabei von **Teilsperrlisten**. Diese sind besonders wirkungsvoll, wenn ein digitales Zertifikat einen Verweis auf die Stelle enthält, an der die zugehörige Teilsperrliste abrufbar ist (der X.509v3-Standard sieht eine Erweiterung vor, die diese Information aufnehmen kann). Diese Stelle wird als **Sperrlisten-Verteilungspunkt** oder auch als **CRL Distribution Point** (**CDP**) bezeichnet. Jede Teilsperrliste sollte einen eigenen Sperrlisten-Verteilungspunkt erhalten. Wenn Alice sich die neuesten Sperrlisten besorgen will, dann schaut sie sich zunächst die Zertifikate ihrer Kommunikationspartner und die darin angegebenen Sperrlisten-Verteilungspunkte an. Danach kann sie sich dort genau die Teilsperrlisten besorgen, die sie benötigt. Wenn Alice nur mit einem geringen Teil der anderen PKI-Anwender kommuniziert, lässt sich mit Teilsperrlisten und Sperrlisten-Verteilungspunkten das Datenaufkommen deutlich verringern.
- *Delta-Sperrlisten*: Eine weitere Möglichkeit, den Umfang von Sperrlisten zu reduzieren, sind **Delta-Sperrlisten** (**Delta-CRLs**). Eine Delta-Sperrliste enthält nur diejenigen Zertifikate, die seit Erstellung der letzten kompletten oder Teil-Sperrliste (**Basis-Sperrliste**) gesperrt worden sind. Um festzustellen, ob ein Zertifikat gesperrt ist, muss Alice die letzte komplette Sperrliste und alle seitdem veröffentlichten Delta-Sperrlisten kennen. Dies ist etwas umständlich, aber es lohnt sich, da das Datenaufkommen auf diese Weise deutlich sinkt. Delta-Sperrlisten und verteilte Sperrlisten lassen sich gleichzeitig einsetzen – in Form verteilter Delta-Sperrlisten.

Standards für Sperrlisten

Der X.509-Standard beschreibt nicht nur ein Format für digitale Zertifikate, sondern auch ein solches für Sperrlisten [X.509]. Es werden sowohl komplette als auch Teil- und Delta-Sperrlisten unterstützt. Ähnlich wie X.509-Zertifikate sind auch X.509-Sperrlisten in Felder eingeteilt, wobei sich die Signatur auf alle Felder bezieht. Als 1988 die erste Version von X.509 erschien, waren folgende Sperrlistenfelder vorgesehen:

- *Version*: In diesem Feld wird die Version des X.509-Sperrlistenformats angegeben. In der 1988 erschienenen ersten Version war dies die Zahl 0 (die Zählung beginnt also bei Null, nicht bei Eins).
- *Signature*: Dieses Feld enthält die Kennung des zur Signatur der Sperrliste verwendeten Signatur-Algorithmus.
- *Issuer*: Dieses Feld enthält den Namen (Distinguished Name nach X.500) des Herausgebers der Sperrliste, also in der Regel einer CA.
- *This Update*: Hier wird der Zeitpunkt angegeben, zu dem die Sperrliste herausgegeben wurde.
- *Next Update*: Hier wird der Ausgabezeitpunkt der nächsten Sperrliste angegeben.
- *Revoked Certificates*: Dieses Feld enthält die Seriennummern der gesperrten Zertifikate, jeweils mit dem Sperrdatum.

Ähnlich wie X.509v1-Zertifikate zeigten auch X.509v1-Sperrlisten schnell Unzulänglichkeiten. 1996 erschien daher mit der dritten Version des X.509-Standards ein erweitertes Format für Sperrlisten. Verwirrenderweise spricht man in diesem Zusammenhang von X.509v2-Sperrlisten, da in der drei Jahre zuvor erschienenen zweiten X.509-Version das Sperrlistenformat nicht geändert worden war. X.509v2-Sperrlisten sehen die Möglichkeit vor, beliebige weitere Felder (Erweiterungen) selbst festzulegen. Es gibt einerseits Erweiterungen, die pro Sperrliste nur einmal auftauchen, und andererseits solche, die innerhalb des Felds *Revoked Certificates* einmal pro aufgeführter Zertifikatsseriennummer vorhanden sind. Jede Erweiterung ist entweder als kritisch oder als unkritisch markiert (mit gleicher Bedeutung wie bei X.509v3-Zertifikaten). Um Wildwuchs zu vermeiden, nahmen die Standardisierer zum X.509v2-Format einige vordefinierte Erweiterungen hinzu. Diese wurden auch von PKIX und Common PKI übernommen. Folgende vordefinierte Erweiterungen beziehen sich jeweils auf die gesamte Sperrliste:

- *Authority Key Identifier*: In dieser Erweiterung wird eine eindeutige Kennung des Schlüssels des Herausgebers der Sperrliste abgelegt. Damit lassen sich beispielsweise unterschiedliche Schlüssel einer CA unterscheiden. Das PKIX- und das Common-PKI-Profil sehen beide vor, dass diese Erweiterung verwendet wird und als kritisch (PKIX) bzw. unkritisch (Common PKI) markiert wird.
- *Issuer Alternative Name*: Im Namensfeld der Sperrliste muss ein X.500-Name angegeben werden. In dieser Erweiterung kann ein weiterer Name oder eine sonstige Kennung des Herausgebers stehen (etwa seine E-Mail- oder IP-Adresse). PKIX sieht vor, diese Erweiterung dann zu verwenden, wenn ein alternativer Name in Gebrauch ist. Dabei wird empfohlen, die Erweiterung als unkritisch zu markieren. Gemäß Common PKI kann diese Erweiterung verwendet werden, muss dann aber unkritisch sein.

- *CRL Number*: Hiermit ist eine eindeutige Seriennummer der Sperrliste gemeint. PKIX schreibt vor, diese Erweiterung als unkritisch zu verwenden, Common PKI ebenfalls.
- *Delta CRL Indicator*: Diese in jedem Fall kritische Erweiterung zeigt an, dass es sich bei der vorliegenden Sperrliste um eine Delta-Sperrliste handelt. Gemäß PKIX und Common PKI können Delta-Sperrlisten und damit diese Erweiterung verwendet werden.
- *Issuing Distribution Point*: In dieser in jedem Fall kritischen Erweiterung werden ein Sperrlisten-Verteilungspunkt und gegebenenfalls einige Angaben über Einschränkungen in der Verwendung der Sperrliste angegeben. Gemäß PKIX muss diese Erweiterung nicht vorhanden sein, Common PKI verbietet sie sogar.

Die PKIX-Arbeitsgruppe der IETF führte eine weitere Erweiterung ein:

- *Freshest CRL*: Diese stets unkritische Erweiterung wird nur eingesetzt, wenn Delta-Sperrlisten im Einsatz sind. Eine Delta-Sperrliste selbst enthält *Freshest CRL* jedoch nicht. Vielmehr ist diese Erweiterung für Basis-Sperrlisten gedacht. Der Zweck besteht darin, den Verteilungspunkt der zugehörigen Delta-Sperrlisten zu nennen.

Zusätzlich gibt es in X.509v2 vordefinierte Erweiterungen, die pro gesperrtem Zertifikat einmal vorkommen:

- *Reason Code*: Hier wird der Grund für die Sperrung des jeweiligen Zertifikats angegeben (siehe unten). PKIX empfiehlt die Verwendung dieser Erweiterung. Gemäß Common PKI kann sie verwendet werden.
- *Hold Instruction Code*: In dieser Erweiterung werden zusätzliche Angaben gemacht, falls die Sperrung temporär ist. Da Common PKI keine temporären Sperrungen erlaubt, verbietet es diese Erweiterung.
- *Invalidity Date*: In dieser Erweiterung wird angegeben, zu welchem Zeitpunkt das jeweilige Zertifikat ungültig geworden ist. Common PKI verbietet die Verwendung dieser Erweiterung.
- *Certificate Issuer*: In dieser Erweiterung steht der Name der CA, die das jeweilige gesperrte Zertifikat signiert hat. Gemäß Common PKI kann dieses Feld verwendet werden.

Sowohl Common PKI als auch PKIX verzichten darauf, eigene Sperrlisten-Erweiterungen einzuführen. Die folgende Liste nennt die Sperrgründe, die im Feld *Reason Code* genannt werden können (in Klammern jeweils der englische Ausdruck für den Sperrgrund sowie dessen Nummer):

- *Unspezifiziert* (*unspecified*, 0)
- *Schlüsselkompromittierung* (*Key Compromise*, 1): Alices Zertifikat muss natürlich dann gesperrt werden, wenn Mallory Zugang zu Alices privatem Schlüssel hat oder ein entsprechender Verdacht besteht (Kompromittierung). Wenn Alice ihre Smartcard verloren oder Mallory ihr Software-PSE kopiert hat, dann ist dieser Fall eingetreten.
- *CA-Kompromittierung* (*CA Compromise*, 2): Wenn der Verdacht besteht, dass Mallory Zugang zum Schlüssel der CA hat, dann ist dies ebenfalls ein Grund zum Sperren von Zertifikaten. In diesem Fall müssen sogar alle Zertifikate gesperrt werden, die mit dem betreffenden Schlüssel signiert wurden.
- *Änderung des Zertifikatsinhalts* (*Affiliation Changed*, 3): Wenn sich am Inhalt des Zertifikats etwas ändert (zum Beispiel Alices Nachname, nachdem sie geheiratet hat), dann muss das Zertifikat gesperrt und gegebenenfalls ein neues generiert werden.
- *Neues Zertifikat vorhanden* (*superseded*, 4): Wird Alices Zertifikat durch ein neues ausgetauscht, dann muss das alte gesperrt werden. Ein solcher Austausch kann etwa kurz vor Ende der Gültigkeitsdauer durchgeführt werden.
- *Einstellung des Betriebs* (*Cessation of Operation*, 5): Wenn Alice nicht mehr Kundin bei der CA ist, die ihr Zertifikat ausgestellt hat, dann ist dies für das Trust Center ein Grund, das Zertifikat zu sperren. Dieser Sperrgrund tritt beispielsweise dann ein, wenn die Firma Krypt & Co. eine CA für die Mitarbeiter betreibt und ein Mitarbeiter das Unternehmen verlässt.
- *Suspendierung* (*Certificate Hold*, 6): Eine Suspendierung ist eine vorübergehende Sperrung, die nach kurzer Zeit wieder aufgehoben wird, falls zwischenzeitlich keine endgültige Sperrung vorgenommen wurde. Eine Suspendierung soll vor allem Denial-of-Service-Attacken verhindern: Wenn eine Sperrung nach kurzer Zeit wieder rückgängig gemacht werden kann, dann kann Mallory weniger Schaden anrichten, wenn er für die Zertifikate anderer Leute eine Sperrung beantragt. In der Praxis gibt es jedoch kaum eine PKI, in der Suspendierungen vorgesehen sind.

28.3.2 Online-Sperrprüfung

Die zwei wichtigsten Nachteile von Sperrlisten liegen auf der Hand: Zum einen wird eine Sperrung erst wirksam, wenn die CA eine neue Sperrliste erstellt – zu diesem Zeitpunkt kann die Sperrung jedoch schon Stunden oder Tage zurückliegen. Zum anderen muss Alice immer eine ganze Liste herunterladen, obwohl sie vielleicht nur ein einziges Zertifikat überprüfen will. Die einfachste Möglichkeit zur Behebung dieser Mängel ist die **Online-Sperrprüfung**. Eine solche ist denkbar einfach: Wenn Alice wissen will, ob ein Zertifikat gesperrt ist, dann sendet sie eine Nachricht mit der Seriennummer des Zertifikats an den Zertifikateserver.

Dieser überprüft dann, ob das betreffende Zertifikat gesperrt ist und sendet die entsprechende Information in einer (meist signierten) Nachricht zurück. Online-Sperrprüfungen haben den Vorteil, dass jede Sperrung unmittelbar wirksam wird. Es gibt jedoch auch Nachteile. So ist es für Alice beispielsweise recht aufwendig (und oftmals nicht möglich), ständig eine Online-Verbindung zum Zertifikateserver herzustellen. Darüber hinaus stellt es einen erheblichen Aufwand dar, wenn der Zertifikateserver jede Antwort signieren muss.

OCSP

Für Online-Sperrprüfungen hat das Internet-Standardisierungsgremium IETF ein Netzwerkprotokoll namens **OCSP** (**Online Certificate Status Protocol**) ins Leben gerufen. Es ist in [RFC2560] standardisiert. OCSP ist zustandslos und auch ansonsten ein recht einfaches Protokoll. Es sieht vor, dass Alice (**OCSP-Requester**) eine Protokollnachricht an einen Server (**OCSP-Responder**) schickt, der diese mit einer weiteren Protokollnachricht beantwortet. In der Nachricht von Alice an den OSCP-Responder ist eine Liste von Zertifikatsseriennummern enthalten (diese Liste hat oft nur einen Eintrag), zu denen Alice den Sperrstatus wissen möchte. Die Nachricht kann von Alice signiert sein.

Die Antwort des OCSP-Responders ist in der Regel signiert und enthält die Information, ob die Zertifikate mit den genannten Seriennummern gesperrt sind. Außerdem wird jeweils ein Sperrgrund mitgeliefert, wobei die in Abschnitt 28.3.1 genannten Möglichkeiten zur Auswahl stehen. OCSP macht keine Aussage über die Aktualität der Sperrinformationen. Eine OCSP-Responder-Implementierung kann ihre Informationen daher aus einer Sperrliste beziehen, auch wenn diese nicht auf dem aktuellen Stand ist. Wer OCSP in einer PKI einsetzen will, sollte sich daher schon in der Konzeptionsphase überlegen, wie aktuell die per OCSP abrufbaren Sperrinformationen sein sollen.

SCVP

Von der PKIX-Arbeitsgruppe stammt ein Protokoll mit dem Namen **Server-based Certificate Validation Protocol** (**SCVP**) [RFC5055]. Man kann sich dieses als Verallgemeinerung von OCSP vorstellen. Auch SCVP ist ein vergleichsweise einfaches, zustandsloses Protokoll, bei dem die Protokollnachrichten per HTTP oder E-Mail übertragen werden. Alice kann SCVP einerseits für eine Online-Sperrprüfung (wie OCSP) verwenden. Andererseits kann sie die Überprüfung eines Zertifikats per SCVP an einen Server delegieren. Vertraut Alice dem Server nicht, dann kann sie sich über SCVP die notwendigen Informationen (vor allem CA-Zertifikate, die innerhalb einer Hierarchie vorkommen, inklusive Sperrlisten) zuschicken lassen, um die Überprüfung selbst vorzunehmen. Im Gegensatz zum weit verbreiteten OCSP hat sich SCVP bisher nicht durchgesetzt.

28.3.3 Weitere Formen des Abrufs von Sperrinformationen

Neben Sperrlisten und der Online-Sperrabfrage gibt es für Alice noch einige weitere Möglichkeiten zum Abruf von Sperrinformation.

Weiße Listen

Sperrlisten bezeichnet man manchmal auch als *Schwarze Listen*. Eine Schwarze Liste enthält alle gesperrten Zertifikate. Es gibt jedoch auch die Möglichkeit, nur die gültigen Zertifikate auf eine Liste zu packen. Eine solche heißt dann folgerichtig **Weiße Liste**. Nutzt Alice eine Weiße Liste, dann darf sie nur Zertifikate verwenden, die auf dieser stehen. Weiße Listen sind meist nur dann sinnvoll, wenn wenige Zertifikate darauf stehen, da ansonsten beim Prüfen ein großer Aufwand entsteht. Manche Implementierungen, in denen nur wenige Zertifikate in Verwendung sind, nutzen interne, hartkodierte Weiße Listen, die meist nicht signiert sind. Eine weitere Variante der Weißen Listen ist ebenfalls in manchen kleineren PKIs im Einsatz: Alice versucht, ein Zertifikat beim Zertifikateserver abzurufen. Findet sie es nicht, dann betrachtet sie es als ungültig. Weiße Listen sind also ein pragmatischer Ansatz für kleine PKIs.

Sperrbäume

Sperrbäume, die auch als **Certificate Revocation Trees** (**CRTs**) bezeichnet werden, sind eine Mischung aus Sperrliste und Online-Sperrabfrage, die mithilfe von Hashbäumen (siehe Abschnitt 14.6.1) realisiert werden [Koch98]. Zur Erstellung eines Sperrbaums muss die CA zunächst eine Liste aller Zertifikatsseriennummern aufstellen. Im Gegensatz zu einer Sperrliste sind auf dieser Liste sowohl gesperrte als auch nicht gesperrte Zertifikate aufgeführt. Jeder Listeneintrag enthält neben der Seriennummer eine Information darüber, ob das jeweilige Zertifikat gesperrt ist. Die Liste ist anhand der Seriennummern sortiert. Eine solche Liste könnte (bei zweistelligen Seriennummern) etwa so aussehen:

1. 46 gültig
2. 47 gültig
3. 48 gültig
4. 50 gesperrt
5. 51 gesperrt
6. 59 gültig
7. 60 gesperrt
8. 65 gültig
9. 66 gültig
10. 70 gültig
11. 71 gesperrt

Diese Liste lässt sich wie folgt vereinfachen:

1. 46–48 gültig
2. 50–51 gesperrt
3. 59 gültig
4. 60 gesperrt
5. 65–70 gültig
6. 71 gesperrt

Auf jeden Listeneintrag wendet die CA nun eine kryptografische Hashfunktion an. Anschließend baut sie einen Hashbaum auf, wie er in Abschnitt 14.6.1 beschrieben wird (siehe Abbildung 28–1). Wenn Alice wissen will, ob ein bestimmtes Zertifikat gesperrt ist (beispielsweise das mit der Nummer 66), dann sendet sie eine Nachricht mit der Seriennummer des Zertifikats an den Zertifikateserver. Dieser sucht anschließend den passenden Listeneintrag aus der vereinfachten Liste (im Beispiel ist dies Eintrag 5) und schickt ihn zusammen mit den relevanten Hashwerten des Hashbaums (in diesem Fall a_5 und c_0) an Alice zurück.

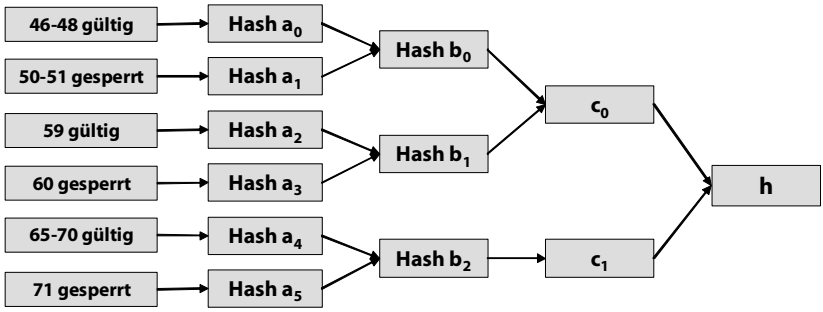


Abb. 28–1 *Beispiel für einen Sperrbaum. Je mehr Zertifikate gesperrt sind, desto mehr lohnt es sich, statt mit Sperrlisten mit Sperrbäumen zu arbeiten.*

Sperrbäume sind für Alice ähnlich zu handhaben wie eine Online-Sperrabfrage. Alice muss keine ganze Sperrliste herunterladen, sondern benötigt nur eine kurze Nachricht, in der die gewünschte Sperrinformation und eine Kette von Hashwerten enthalten ist. Selbst wenn es sehr viele gesperrte Zertifikate gibt, bleibt die Antwort des Zertifikateservers überschaubar (1 Million gesperrte Zertifikate erfordern gerade einmal 20 Hashwerte, was etwa 400 Byte entspricht). Für den Zertifikateserver und die CA sind Sperrbäume von Vorteil, da nicht jede Antwort einzeln signiert werden muss. Trotz dieser Vorteile haben sich Sperrbäume bisher nicht durchgesetzt. Vermutlich liegt dies daran, dass besonders große PKIs mit Millionen gesperrter Zertifikate bisher eine Seltenheit sind.

Kurzlebige Zertifikate

In der Kryptografie sind pragmatische Lösungen oft besonders erfolgreich. Eine solche ist zweifellos die Verwendung **kurzlebiger Zertifikate**. Dabei handelt es sich um Zertifikate, die nur ein paar Stunden oder Tage gültig sind. Ein kurzlebiges Zertifikat muss nicht gesperrt werden, wenn beispielsweise ein Mitarbeiter das Unternehmen verlässt oder wenn sich der Name des Inhabers ändert. Auch der Verlust einer Smartcard ist oft tolerierbar, wenn die Gültigkeit innerhalb kurzer Zeit erlischt. Kurzlebige Zertifikate haben also den Vorteil, dass ein PKI-Anwender gänzlich auf eine Sperrprüfung verzichten kann.

Der Nachteil kurzlebiger Zertifikate besteht darin, dass die CA ständig neue Zertifikate für dieselbe Person ausstellen muss. Wenn sich dies jedoch automatisieren lässt – etwa mit einem Auto-Enrollment –, dann ist dies oft kein wirkliches Problem. Kurzlebige Zertifikate lassen sich außerdem sehr gut mit Roaming-PSEs kombinieren. Mehr Informationen zum Thema kurzlebige Zertifikate gibt es in [CalKir].

29 Spezielle Fragen beim Betrieb einer PKI

In diesem Kapitel wollen wir einige weitere Fragen betrachten, die sich der Betreiber einer PKI stellen muss. Diese Fragen sind recht unterschiedlicher Natur, sind aber jeweils wichtig.

29.1 Outsourcing oder Eigenbetrieb?

Wenn wir annehmen, dass die Firma Krypt & Co. eine PKI für den Eigenbedarf aufbauen will, dann lautet eine der wichtigsten Fragen: Soll die PKI als Outsourcing-Lösung oder im Eigenbetrieb realisiert werden? Verkürzt lässt sich dies auch so formulieren: Make or buy? Eine ähnliche Frage stellt sich auch bei vielen anderen Investitionen eines Unternehmens – von der Gebäudereinigung bis zur Herstellung von Teilkomponenten lässt sich in einem Unternehmen so ziemlich alles auslagern.

Beim **PKI-Eigenbetrieb** baut Krypt & Co. die CA selbst auf – beispielsweise im eigenen Rechenzentrum. Dafür braucht das Unternehmen eine CA-Software, wie sie von verschiedenen Anbietern erhältlich ist. Entscheidet sich Krypt & Co. dagegen für **PKI-Outsourcing**, dann wird ein bereits bestehendes Trust Center genutzt, das seine Dienste am Markt anbietet. Für den Eigenbetrieb einer PKI sprechen aus Sicht von Krypt & Co. folgende Argumente:

- Beim Outsourcing verlagert Krypt & Co. eine sicherheitskritische Dienstleistung nach außen. Beim Eigenbetrieb ist dies nicht der Fall, die Sicherheit bleibt also im Unternehmen.
- Bei vorhandenen Ressourcen (Rechenzentrum, Administrationspersonal, ...) kann der Eigenbetrieb einer PKI die kostengünstigere Lösung sein. Dies gilt zumindest dann, wenn die Anforderungen an die PKI in puncto Ausfallsicherheit und Performanz nicht außerordentlich hoch sind.
- Eigenbetrieb ist in der Regel flexibler, da sich Krypt & Co. nicht an den Vorgaben des Outsourcing-Anbieters orientieren muss.

Für PKI-Outsourcing sprechen folgende Argumente:

- Outsourcing ist die einfachere und nervenschonendere Lösung. Krypt & Co. überlässt dabei den PKI-Betrieb Spezialisten und kann sich auf sein Kerngeschäft konzentrieren.
- Outsourcing ist in der Regel unter dem Strich die billigere Variante (vieles hängt jedoch von den genauen Umständen ab). Dies gilt vor allem dann, wenn die Anforderungen von Krypt & Co. an die Ausfallsicherheit und die Performanz besonders hoch sind.
- Qualifizierte Zertifikate nach deutschem Signaturgesetzes sind bei einer unternehmensinternen PKI derzeit nur mit Outsourcing sinnvoll möglich. Die Anforderungen des Signaturgesetzes sind so hoch, dass der Aufbau eines Trust Centers, das qualifizierte Zertifikate ausstellt, für den Eigenbedarf viel zu teuer ist.

Obwohl Outsourcing bei einer nüchternen Analyse durchaus seine Vorteile hat, entscheiden sich bisher mehr Unternehmen für den Eigenbetrieb. Der Gedanke, die CA unter dem eigenen Dach und innerhalb des direkten Einflussbereichs zu haben, ist für viele IT-Leiter offensichtlich von großer Bedeutung. In manchen Fällen spielen sicherlich auch persönliche Eitelkeiten eine wichtige Rolle: Welcher Mitarbeiter einer IT-Abteilung möchte nicht gerne den Aufbau einer CA im Lebenslauf stehen haben?

29.2 Gültigkeitsmodelle

Am 1. Januar eines Jahres fertigt Alice eine digitale Signatur an. Am 31. Januar läuft ihr Zertifikat aus. Am 31. März läuft das Zertifikat ihrer CA aus. Am 31. Mai läuft dann schließlich auch das Zertifikat der Root-CA aus, die das CA-Zertifikat erstellt hat. Was passiert nun, wenn Bob Alices Signatur am 28. Februar verifiziert? Ist die Signatur gültig, obwohl Alices Zertifikat zu diesem Zeitpunkt bereits abgelaufen ist? Und was passiert, wenn Bob Alices Signatur am 30. April oder am 30. Juni überprüft.

Keine Frage, das Verifizieren einer digitalen Signatur in einer Public-Key-Infrastruktur kann zu einer komplexen Angelegenheit werden. Überprüft Bob Alices Signatur unmittelbar nachdem diese sie erstellt hat, kann nicht viel passieren. Die Probleme entstehen dann, wenn eine Überprüfung auch noch Monate oder Jahre später möglich sein muss. Um solchen Problemen aus dem Weg zu gehen, muss der Betreiber einer PKI Kriterien festlegen, nach denen die Gültigkeit oder Ungültigkeit einer Signatur in Abhängigkeit der involvierten Zertifikate bestimmt wird. Man nennt so etwas ein **Gültigkeitsmodell**. Die zwei wichtigsten Gültigkeitsmodelle heißen Kettenmodell und Schalenmodell. Wir werden sie weiter unten genauer betrachten. Dabei gehen wir von einer zweistufigen CA-Hierarchie aus. Alice hat also ein Zertifikat von einer CA, die selbst ein Zertifikat von einer Wurzel-CA besitzt. Alle Überlegungen lassen sich leicht auf komplexere CA-Hierarchien übertragen.

Um ein Gültigkeitsmodell richtig anwenden zu können, muss klar sein, wann Alice ihre Signatur angefertigt hat. Ein signierter Zeitstempel von einem Zeitstempeldienst kann dabei hilfreich sein. Die Signatur des Zeitstempeldiensts muss hierbei selbst einem Gültigkeitsmodell unterliegen. Außerdem spielt in diesem Zusammenhang eine Rolle, wann die Gültigkeit eines Zertifikats endet. Dies ist natürlich dann der Fall, wenn der im Zertifikat angegebene Gültigkeitszeitraum abläuft. Außerdem beendet eine Sperrung die Gültigkeit eines Zertifikats. Bevor Bob eine Verifizierung durchführt, muss er daher zunächst wissen, ob von den drei beteiligten Zertifikaten (Alices Zertifikat, das CA-Zertifikat und das Wurzel-Zertifikat) eines oder mehrere gesperrt wurden.

Deutlich schwieriger als eine Sperrung ist eine Kompromittierung zu handhaben. Eine solche liegt etwa vor, wenn das verwendete Signaturverfahren geknackt oder der private CA-Schlüssel gestohlen wird. Die Sache ist nicht zuletzt deshalb so kompliziert, weil oftmals gleich mehrere Verfahren mit unterschiedlichen Schlüssellängen in einem Gültigkeitsmodell eine Rolle spielen. So kann Alice etwa selbst 1.024-Bit-RSA verwenden, während die CA und die Wurzel-CA zum Signieren von Zertifikaten dasselbe Verfahren mit 2.048 Schlüssel-Bits einsetzen. Gleichzeitig arbeitet der Zeitstempeldienst vielleicht mit einem ECC-Verfahren. Welche Konsequenzen es hat, wenn etwa Mallory den CA-Schlüssel klaut oder ein neuer Angriff auf das RSA-Verfahren bekannt wird, wollen wir an dieser Stelle nicht betrachten und klammern daher im Folgenden eine Kompromittierung aus.

29.2.1 Schalenmodell

Das älteste und einfachste Gültigkeitsmodell ist das **Schalenmodell**. Im Grunde handelt es sich dabei um zwei unterschiedliche, jedoch recht ähnliche Gültigkeitsmodelle. Das erste wird meist als **PEM-Schalenmodell** bezeichnet (PEM ist ein veralteter E-Mail-Verschlüsselungsstandard, der dieses Modell einführte [RFC1421, RFC1422, RFC1423, RFC1424]). Das PEM-Schalenmodell lässt sich wie folgt beschreiben: Eine Signatur ist dann gültig, wenn zum Verifikationszeitpunkt sowohl Alices Zertifikat als auch das CA-Zertifikat als auch das Wurzel-CA-Zertifikat gültig sind. Wie Sie leicht nachvollziehen können, benötigt man in diesem Fall keinen Zeitstempeldienst.

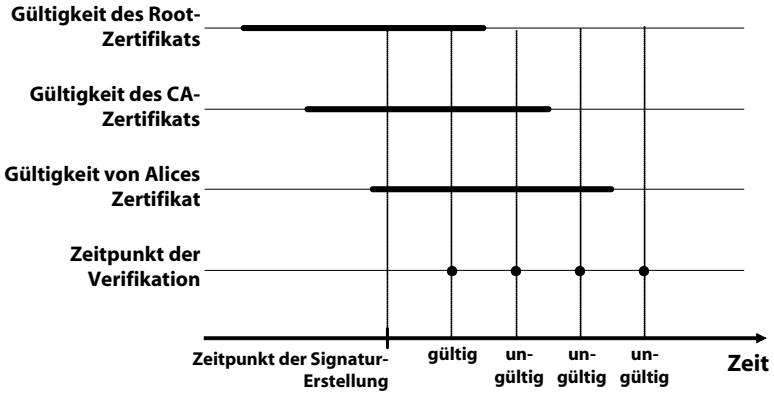


Abb. 29–1 *Im PEM-Schalenmodell ist eine Signatur gültig, wenn zum Verifikationszeitpunkt sowohl Alices Zertifikat als auch das CA-Zertifikat als auch das Wurzel-CA-Zertifikat gültig sind.*

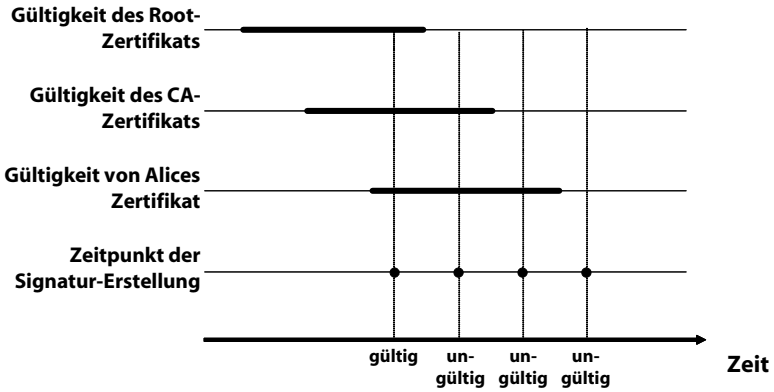


Abb. 29–2 *Im modifizierten Schalenmodell ist eine Signatur dann gültig, wenn zum Signierzeitpunkt alle beteiligten Zertifikate gültig sind.*

Das PEM-Schalenmodell ist zwar einfach, jedoch für viele Anwendungen nicht geeignet. Sein größter Nachteil besteht darin, dass alle Signaturen ungültig werden, sobald das erste beteiligte Zertifikat ausläuft. Etwas leistungsfähiger ist daher das **modifizierte Schalenmodell**. Dieses besagt, dass eine Signatur dann gültig ist, wenn zum Signierzeitpunkt alle beteiligten Zertifikate gültig sind. Wann genau der Signierzeitpunkt war, lässt sich gegebenenfalls über einen signierten Zeitstempel herausfinden. Das modifizierte Schalenmodell impliziert, dass weder die CA noch die Wurzel-CA Zertifikate ausstellen sollten, die länger gültig sind als das eigene Zertifikat.

29.2.2 Kettenmodell

Das heute vorherrschende Gültigkeitsmodell ist das **Kettenmodell**. Dieses kommt insbesondere im Zusammenhang mit fast allen Signaturgesetzen zum Einsatz. Laut Kettenmodell ist Alices Signatur gültig, wenn folgende Voraussetzungen gegeben sind:

- Alices Zertifikat war zum Signaturzeitpunkt gültig.
- Alices Zertifikat wurde zu einem Zeitpunkt ausgestellt, als das CA-Zertifikat gültig war.
- Das CA-Zertifikat wurde zu einem Zeitpunkt ausgestellt, als das Wurzel-CA-Zertifikat gültig war.

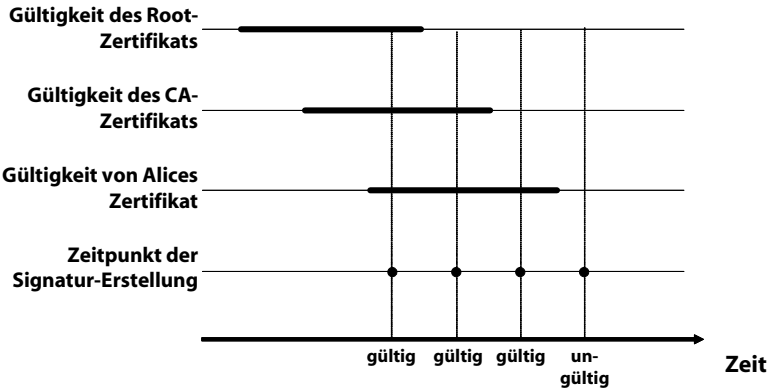


Abb. 29–3 *Das Kettenmodell ist das wichtigste Gültigkeitsmodell.*

Das Kettenmodell ist dem Schalenmodell klar überlegen. Es setzt allerdings voraus, dass eine Verifikationskomponente mit der etwas komplizierten Logik umgehen kann.

29.3 Certificate Policy und CPS

Bevor sich Alice ein Zertifikat von einer CA ausstellen lässt, möchte sie normalerweise wissen, worauf sie sich dabei einlässt. Es interessiert sie beispielsweise, ob sich die CA an Vorgaben des Signaturgesetzes hält, welche Zertifikatstypen zur Auswahl stehen und welche Sicherheitsmaßnahmen die CA getroffen hat. Natürlich will sie auch wissen, welche Pflichten sie als Zertifikatsinhaberin übernimmt und in welchen Fällen der CA-Betreiber für Schäden haftet. Aus diesen Gründen sollte eine CA eine Beschreibung ihrer Dienste in einem geeigneten Dokument veröffentlichen. Ein solches Dokument, das oft als *Policy* bezeichnet wird, kann als Grundlage eines Vertrags verwendet werden und hat eine ähnliche Funktion wie die Allgemeinen Geschäftsbedingungen eines Unternehmens.

Da es sehr viele CAs gibt, die eine Policy veröffentlichen, entschied sich die IETF, ein Rahmenwerk für die Erstellung solcher Dokumente zu entwickeln. Das Ergebnis dieser Bemühungen ist der 2003 veröffentlichte RFC 3647 [RFC3647]. RFC 3647 ist streng genommen kein Standard, da er zu den Informational RFCs gehört. Er ermöglicht jedoch – zumindest in der Theorie – das Erstellen einheitlicher Policys und hat damit Standard-Charakter. Eine gute Übersicht zu RFC 3647 gibt [BarKel].

RFC 3647 sieht vor, dass es zwei unterschiedliche Policy-Dokumente gibt, die sich gegenseitig ergänzen: die **Certificate Policy** (**CP**) und das **Certification Practice Statement** (**CPS**). Eine Certificate Policy bezieht sich auf einen bestimmten Zertifikatstypen. Wenn eine CA unterschiedliche Zertifikatstypen unterstützt, sollte sie für jeden davon eine eigene Certificate Policy erstellen. Es ist auch möglich, dass dieselbe Certificate Policy von mehreren CAs verwendet wird. Zudem kann eine CA einer Certificate Policy einen Namen in Form einer OID geben. X.509v3-Zertifikate haben die Eigenschaft, dass sie eine solche OID speichern können. Wenn Alice wissen will, worauf sie sich bei einem Zertifikat einlässt, das ihr jemand zugeschickt hat, dann kann sie dementsprechenden X.509-Feld die Certificate-Policy-OID entnehmen und das zugehörige Dokument im Internet einsehen.

Ein CPS bezieht sich im Gegensatz zu einer Certificate Policy nicht auf eine bestimmte Art von Zertifikaten, sondern auf eine gesamte CA. Im CPS einer CA sind der Aufbau der zugehörigen PKI, die ablaufenden Prozesse und weitere Eigenschaften beschrieben. Die diversen Certificate Policys, die eine CA unterstützt, werden darin aufgelistet. Im Gegensatz zu einer Certificate Policy ist ein CPS einer CA fest zugeordnet und ist nicht für eine Übernahme durch andere CAs geeignet.

Das CPS und die Certificate Policys einer CA werden meist gegenüber den Anwendern veröffentlicht. Falls jedoch geheime Informationen in diesen Dokumenten enthalten sind, sieht RFC 3647 die Möglichkeit vor, nichtvertrauliche Aspekte in ein Dokument namens **PDS** (**PKI Disclosure Statement**) auszulagern und nur dieses öffentlich zugänglich zu machen. Ein CPS oder eine CP ersetzt zudem kein Betriebskonzept, in dem alle Komponenten, Rollen und Prozesse einer CA beschrieben werden.

29.3.1 Was steht in einem CPS und einer Certification Policy?

RFC 3647 schreibt keine Inhalte zwingend vor. Stattdessen wird eine größere Auswahl möglicher **Bestimmungen** (Provisions) aufgezählt, aus dem sich der CPS/CP-Autor bedienen kann. Für diese Bestimmungen ist zudem eine Gliederung in neun Kapitel enthalten. Die inhaltliche Gestaltung der einzelnen Bestimmungen ist ebenfalls Sache des CPS/CP-Autors, der RFC macht lediglich Vorschläge. Man kann sich RFC 3647 insgesamt als eine Art Baukasten für die CPS/CP-Erstellung vorstellen.

Im Folgenden will ich genauer auf die neun Kapitel eingehen, die in RFC 3647 aufgeführt sind. Da es bisher keine offizielle deutsche Übersetzung für diese Kapitel und deren Unterkapitel (ein Unterkapitel entspricht einer Bestimmung) gibt, möchte ich an dieser Stelle einen entsprechenden Übersetzungsvorschlag machen, in der Hoffnung, dass dieser von deutschsprachigen CPS/CP-Autoren übernommen wird. Alle Kapitel und Unterkapitel werden daher auf Deutsch übersetzt angegeben, der englische Orignaltitel steht jeweils kursiv in Klammern.

Kapitel 1:
Einführung (*Introductions*)

Jede Certificate Policy und jedes CPS sollte laut RFC 3647 eine Einführung enthalten. In dieser Einleitung können der Zweck des Dokuments, die Anwendungen, der Urheber und eine mögliche OID der Policy beschrieben werden. Der RFC sieht für die Einführung folgende Unterkapitel vor:

1. Überblick (*Overview*)
2. Name und Kennung des Dokuments (*Document Name and Identification*)
3. PKI-Teilnehmer (*PKI Participants*)
4. Verwendung der Zertifikate (*Certificate Usage*)
5. Policy-Verwaltung (*Policy Administration*)
6. Definitionen und Abkürzungen (*Definitions and Acronyms*)

Kapitel 2:
Veröffentlichung und Aufbewahrung (*Publication and Repository Responsibilities*)

In diesem Kapitel stehen der Name der CA und gegebenenfalls weitere relevante Namen. Darüber hinaus wird an dieser Stelle festgehalten, welche Informationen die an der PKI beteiligten Komponenten veröffentlichen müssen. Dieses Kapitel hat gemäß RFC 3647 keine Unterkapitel.

Kapitel 3:
Identifizierung und Authentifizierung (*Identification and Authentication*)

In diesem dritten von RFC 3647 vorgesehenen Kapitel geht es um die wichtige Frage, wie die potenziellen PKI-Anwender identifiziert und authentifiziert werden, bevor sie ein Zertifikat erhalten. Welche Authentifizierung verlangt wird, hängt auch davon ab, ob es um ein Enrollment oder nur um die Erneuerung eines Zertifikats geht. Neben Authentifizierung spielt an dieser Stelle außerdem die Frage eine Rolle, welche Namen in einem Zertifikat verwendet werden (echter Name oder Pseudonym). Die folgende Liste nennt die vorgesehenen Unterkapitel:

1. Namensgebung (*Naming*)
2. Initiale Überprüfung der Identität (*Initial Identity Validation*)
3. Identifizierung und Authentifizierung für Erneuerungsanträge (*Identification and Authentication for Re-key Requests*)
4. Identifizierung und Authentifizierung für Sperranträge (*Identification and Authentication for Revocation Requests*)

Kapitel 4:
Anforderungen an das Zertifikate-Management (*Certificate Life-Cycle Operational Requirements*)

In diesem Teil einer Certificate Policy oder eines CPS werden einige der Prozesse beschrieben, die in einer PKI ablaufen. Dazu zählen das Anwender-Enrollment, Sperrungen, Sicherheitsüberprüfungen, die Archivierung von Daten, Schlüsselwechsel, die Reaktion auf Krisensituationen und einiges mehr. Die folgenden Unterkapitel sind vorgesehen:

1. Zertifizierungsanträge (*Certificate Application*)
2. Verarbeitung der Zertifizierungsanträge (*Certificate Application Processing*)
3. Ausstellen der Zertifikate (*Certificate Issuance*)
4. Entgegennahme von Zertifikaten (*Certificate Acceptance*)
5. Nutzung von Schlüsselpaaren und Zertifikaten (*Key Pair and Certificate Usage*)
6. Zertifikatserneuerung (*Certificate Renewal*)
7. Zertifikatsschlüsselerneuerung (*Certificate Re-key*)
8. Zertifikatsmodifikation (*Certificate Modification*)
9. Zertifikatssperrung und -suspendierung (*Certificate Revocation and Suspension*)
10. Zertifikatsstatus-Dienste (*Certificate Status Services*)
11. Ende der Geschäftsbeziehung (*End of Subscription*)
12. Recovery (*Key Escrow and Recovery*)

Kapitel 5:
Bauliche und organisatorische Bestimmungen (*Facility, Management and Operational Controls*)

In diesem Kapitel werden die baulichen und organisatorischen Bestimmungen behandelt, die für den Betrieb der PKI eine Rolle spielen. Es gibt folgende Unterkapitel:

1. Physikalische Sicherheitsmaßnahmen (*Physical Security Controls*)
2. Organisatorische Sicherheitsmaßnahmen (*Procedural Controls*)
3. Personelle Sicherheitsmaßnahmen (*Personnel Security Controls*)
4. Auditierung (*Audit Logging Procedures*)
5. Archivierung (*Records Archival*)
6. CA-Schlüsselwechsel (*Key Changeover*)
7. Notfall-Management (*Compromise and Disaster Recovery*)
8. CA- und RA-Betriebsende (*CA or RA Termination*)

Kapitel 6:
Technische Sicherheitsbestimmungen (*Technical Security Controls*)

In dieses Kapitel fallen die Sicherheitsmaßnahmen, die für den Betrieb einer PKI getroffen werden. Sicherheit bedeutet in diesem Zusammenhang nicht nur Security, sondern auch Safety (siehe Abschnitt 2.2). Folgende Unterkapitel gibt es:

1. Generierung und Initialisierung der Schlüssel
   (*Key Pair Generation and Installation*)
2. Schutz der privaten Schlüssel
   (*Private Key Protection and Cryptographic Module Engineering Controls*)
3. Weitere Aspekte des Schlüsselpaar-Managements
   (*Other Aspects of Key Pair Management*)
4. Aktivierungsdaten (*Activation Data*)
5. IT-Sicherheitsbestimmungen (*Computer Security Controls*)
6. Sicherheitsbestimmungen für das Zertifikate-Management
   (*Life Cycle Security Controls*)
7. Bestimmungen zur Netzwerksicherheit (*Network Security Controls*)
8. Zeitstempeldienst (*Time-stamping*)

Kapitel 7:
Zertifikats-, Sperrlisten- und OCSP-Profile (Certificate, CRL and OCSP Profiles)

In einer Certificate Policy oder einem CPS müssen geeignete Formate ausgewählt werden. Dies erfolgt in folgenden Unterkapiteln:

1. Zertifikatsprofil (*Certificate Profile*)
2. Sperrlistenprofil (*CRL Profile*)
3. OCSP-Profil (*OCSP Profile*)

Kapitel 8:
Evaluierung und Überprüfung (*Compliance Audit and Other Assessment*)

Wenn ein CPS/CP und deren korrekte Umsetzung in die Praxis geprüft werden soll, dann ist dieses Kapitel der vorgesehene Platz, um die entsprechenden Bestimmungen festzuhalten. Dieses Kapitel hat keine Unterkapitel.

Kapitel 9:
Sonstige gesetzliche und geschäftliche Aspekte (*Other Business and Legal Matters*)

In diesem Teil einer Certificate Policy oder eines CPS geht es um rechtliche Fragen. Folgende Unterkapitel sind laut RFC 3647 enthalten:

1. Gebühren (*Fees*)
2. Finanzielle Aspekte (*Financial Responsibility*)
3. Vertraulichkeit geschäftlicher Daten (*Confidentiality of Business Information*)
4. Vertraulichkeit persönlicher Daten (*Privacy of Personal Information*)
5. Urheberrecht (*Intellectual Property Rights*)
6. Vertretung und Garantien (*Representations and Warranties*)
7. Ausschlussklauseln (*Disclaimers of Warranties*)
8. Haftungsbeschränkungen (*Limitations of Liability*)
9. Entschädigungen (*Indemnities*)
10. Fristen und Beendigung (*Term and Termination*)
11. Persönliche Mitteilungen und Kommunikation mit Teilnehmern (*Individual notices and communications with participants*)
12. Zusätze (*Amendments*)
13. Schlichtung (*Dispute Resolution Procedures*)
14. Einschlägige Gesetze (*Governing Law*)
15. Sonstige gesetzliche Bestimmungen (*Compliance with Applicable Law*)
16. Weitere Bestimmungen (*Miscellaneous Provisions*)
17. Sonstige Bestimmungen (*Other Provisions*)

29.3.2 Nachteile von RFC 3647

Ich gebe offen zu, dass ich kein Freund von RFC 3647 bin. Meiner Meinung nach ist dieser RFC unübersichtlich, hat einen Geburtsfehler und weist einige unnötige Mängel auf.

Ungünstige Struktur

Schon der Aufbau eines CPS- oder CP-Dokuments nach RFC 3647 ist meiner Meinung nach nicht optimal gelöst. Hier ein paar Beobachtungen, die diese Einschätzung belegen:

- Meines Erachtens ist es sinnvoll, die Beschreibung eines IT-Systems (ein CPS/CP ist eine solche) in drei Teile zu gliedern: Komponenten, Rollen und Prozesse. Oft bietet es sich an, als weiteren Bestandteil eine Einführung oder ein Kapitel über vertragliche Details vorzusehen. RFC 3647 folgt diesem bewährten Vorgehen jedoch nicht, sondern beschreibt eine neunteilige Kapitelstruktur, die unnötig kompliziert wirkt.
- Die Inhalte eines CPS/CP gemäß RFC 3647 sind sehr ungleichmäßig auf die neun Kapitel verteilt. Kapitel 2 (*Veröffentlichung und Aufbewahrung*) und Kapitel 8 (*Evaluierung und Überprüfung*) haben keine Unterkapitel und fallen daher meist recht kurz aus. Kapitel 9 (*Sonstige gesetzliche und geschäftliche Aspekte*) ist in den meisten mir bekannten CPS/CP-Dokumenten sogar leer, während Kapitel 5 (*Bauliche und organisatorische Bestimmungen*) und Kapitel 6 (*Technische Sicherheitsbestimmungen*) wenig Inhalt haben. Der eigentliche CPS/CP-Inhalt konzentriert sich auf die verbleibenden Kapitel 1, 3 und 7, die dadurch vergleichsweise umfangreich werden.
- Die RFC-3647-Kapitelstruktur sieht kein dediziertes Kapitel zum Thema Prozesse vor. Stattdessen muss der CPS/CP-Autor Prozessbeschreibungen über die Kapitel 4, 5 und 6 verstreuen, was wahrlich nicht übersichtlich ist.
- Der wichtigste Prozess in einer PKI ist das Anwender-Enrollment. Die Bedeutung dieses Vorgangs sollte sich auch in einem CPS oder CP widerspiegeln. RFC 3647 kennt leider kein Enrollment als kompletten Prozess. Stattdessen wird der entsprechende Vorgang in vier Teile zerlegt (*Zertifizierungsanträge*, *Verarbeitung der Zertifizierungsanträge*, *Ausstellen der Zertifikate*, *Entgegennahme von Zertifikaten*). Diese Vierteilung erscheint unnötig und unübersichtlich.
- RFC 3647 fordert eine Trennung zwischen CPS und CP. In der Praxis wünscht ein PKI-Betreiber jedoch meist, alle Informationen in einem Dokument zu haben. Aus Sicht eines Beraters ist es ohnehin schwierig, einen Kunden von der Notwendigkeit mehrerer Policy-Dokumente zu überzeugen. Besser ist es in den meisten Fällen, ein CPS und mehrere CPs in ein Dokument zu packen. Dabei ist es natürlich trotzdem möglich, verschiedene Zertifikatstypen zu unterscheiden und dazu jeweils eigene Policy-Inhalte zu definieren.

Die genannten Probleme machen es zu einem schwierigen Unterfangen, ein gutes CPS/CP-Dokument zu schreiben, das RFC-3647-konform ist. Viele CPS/CP-Autoren halten sich daher erst gar nicht an die RFC-3647-Struktur.

Unverständliche Mängel

Neben der ungünstigen Struktur weist RFC 3647 weitere Mängel auf. Hier die wichtigsten:

- Die Begriffe »CPS« und »CP« sind unvorteilhaft gewählt. Die beiden Abkürzungen sehen sich recht ähnlich, zumal der Plural von »CP« »CPs« lautet, was wiederum leicht mit »CPS« zu verwechseln ist. Es wäre sicherlich besser

gewesen, andere Begriffe zu verwenden. Möglich wären zum Beispiel *CAP* (*CA Policy*) statt CPS und *DCP* (*Digital Certificate Policy*) statt *CP*.

- Schlecht gestaltet ist in RFC 3647 zudem die Behandlung des Themas Zertifikatsersetzung. Der RFC sieht drei unterschiedliche Prozesse hierfür vor: Zertifikatserneuerung, Zertifikatsschlüsselerneuerung und Zertifikatsmodifikation. Leider werden diese drei Prozesse nicht sauber voneinander abgegrenzt. So ist nicht klar, welche Zertifikatsinhalte sich bei einer Zertifikatsschlüsselerneuerung ändern. Einerseits ist es der Sinn dieses Prozesses, dass außer dem Schlüssel alle Inhalte gleich bleiben (ansonsten wäre es eine Zertifikatsmodifikation). Andererseits müssen sich aus naheliegenden Gründen wenigstens die Gültigkeitsdauer und die Seriennummer des Zertifikats im Rahmen dieses Prozesses ändern. In RFC 3647 ist nichts Näheres dazu zu erfahren.
- RFC 3647 geht nicht auf PGP-Zertifikate ein. Dies ist einerseits verständlich, da RFC 3647 zu PKIX gehört, das wiederum in der X.509-Welt angesiedelt ist. CPS/CP-Autoren interessiert die Herkunft des Standards jedoch wenig, weshalb sie das Fehlen von PGP oft als Mangel betrachten.

Ich hoffe, dass einige dieser Mängel in der nächsten Version von RFC 3647 behoben werden. Eine Umbenennung von CPS und CP wird es jedoch kaum geben, da sich die beiden existierenden Bezeichnungen längst eingebürgert haben.

Veraltete Inhalte

RFC 3647 wurde im Jahr 2003 veröffentlicht. Bei Redaktionsschluss dieses Buchs war der Inhalt des Standards somit bereits neun Jahre alt. Da sich im PKI-Bereich zwischenzeitlich einiges getan hat, weist RFC 3647 zwangsläufig Lücken auf. Die folgende Liste nennt einige Beispiele:

- Roaming-PSEs werden in RFC 3647 nicht erwähnt.
- Kurzlebige Zertifikate werden in RFC 3647 nicht erwähnt.
- CV-Zertifikate werden in RFC 3647 nicht erwähnt.
- Das Thema Identity Management wird in RFC 3647 nicht erwähnt.
- Smartcards werden in RFC 3647 nicht erwähnt.

Das Fehlen dieser Begriffe heißt natürlich nicht, dass die entsprechenden Themen im CPS oder in der CP nicht vorkommen dürfen. Vor allem für den weniger erfahrenen CPS/CP-Autoren wäre es jedoch sinnvoll, sie zu erwähnen. Außerdem lassen sich diese Mängel in der nächsten Version von 3647 beheben. Man muss sich jedoch die Frage stellen, ob ein RFC überhaupt das richtige Medium für ein CPS/CP-Rahmenwerk ist. Ein RFC ist im Normalfall einige Jahre lang gültig, was etwa für die Beschreibung eines Protokolls durchaus sinnvoll ist. Ein CPS/CP-Rahmenwerk ist jedoch kein Protokoll und kann sich deshalb ändern, ohne dass Interoperabilitätsprobleme entstehen. Es ist daher zu überlegen, ob RFC 3647 durch ein anders geartetes Dokument ersetzt werden sollte, bei dem kontinuierliche Nachbesserungen möglich sind.

Probleme für nichtenglischsprachige Autoren

Es liegt in der Natur der Sache, dass RFC 3647 für nichtenglischsprachige Autoren einige Probleme aufwirft. Insbesondere gibt es bisher keine einheitliche Übersetzung der RFC-3647-Bestimmungen ins Deutsche. Vielleicht setzen sich ja die in Abschnitt 29.3.1 aufgeführten Übersetzungsvorschläge durch. Auch für die Fachbegriffe in RFC 3647 gibt es keine offizielle Übersetzung. Es ist daher unter anderem unklar, ob man »CA« mit »Zertifizierungsstelle« oder »Zertifizierungsinstanz« oder anders übersetzen soll. Natürlich wäre es nicht sinnvoll, in einem Dokument wie RFC 3647 Übersetzungsfragen zu berücksichtigen. Stattdessen sind an dieser Stelle nationale Organisationen (in Deutschland etwa Teletrust oder das BSI) gefordert. Diese könnten für eine Anpassung von RFC 3647 auf die jeweiligen nationalen Gegebenheiten sorgen. Ein solche Anpassung könnte auch nationale Besonderheiten wie das deutsche Signaturgesetz berücksichtigen – es versteht sich von selbst, dass ein international gültiger Standard auf so etwas nicht eingehen kann.

Fazit und Lösungsvorschläge

Kein Zweifel, RFC 3647 ist alles andere als ein Glanzlicht unter den PKIX-RFCs. Und das, obwohl RFC 3647 im Gegensatz zu anderen PKIX-Bestandteilen nicht durch Altlasten aus der PKI-Steinzeit betroffen ist. Was zu tun ist, um dieses Problem anzugehen, habe ich teilweise schon angesprochen. So sollte die nächste RFC-Version einige zusätzliche Inhalte (z.B. Smartcards, Literaturverzeichnis) enthalten. Auch eine andere Gliederung wäre von Vorteil. Zudem sollten nationale Organisationen eine Anpassung und Übersetzung von RFC 3647 vornehmen. Unabhängig davon wäre es sinnvoll, statt eines RFC eine flexiblere Form der Veröffentlichung zu wählen. Es sollte möglich sein, den Inhalt eines solchen CPS-CP-Standards regelmäßig zu aktualisieren, was bei einem RFC nicht vorgesehen ist.

Interessant wäre es außerdem, einen alternativen CPS/CP-Standard ins Leben zu rufen. Diesen könnte man *Simple CA Policy* (*SCAP*) nennen. Ein SCAP-Dokument würde ein CPS und mehrere CPs durch ein Dokument ersetzen. Die Gliederung eines SCAP-Dokuments sollte übersichtlicher und einfacher gestaltet werden, als dies in RFC 3647 der Fall ist. Auch bei SCAP sollte es sich nicht um einen RFC handeln, sondern um ein Rahmenwerk, dessen Inhalt laufend angepasst werden kann. Da CPS/CP-Autoren jedoch weder auf die nächste RFC-Version noch auf SCAP warten können, empfehle ich für das Schreiben eines CPS/CP-Dokuments Folgendes:

- In vielen Fällen reicht ein Dokument aus. Dieses sollte sowohl das CPS als auch die relevanten CPs enthalten.
- Die gesamte PKI-Architektur sollte in einem Betriebskonzept beschrieben werden, das vom CPS/CP-Dokument unabhängig ist.

- Ich empfehle, trotz aller Mängel die in RFC 3647 vorgegebene Struktur beizubehalten. Dadurch werden CPS/CP-Dokumente untereinander vergleichbar. Dabei muss man jedoch in Kauf nehmen, dass ein CPS bzw. CP zahlreiche leere Unterkapitel enthält.

Auf die beschriebene Weise sollte es möglich sein, brauchbare CPS/CP-Dokumente zu schreiben, die RFC 3647 entsprechen.

29.4 Policy-Hierarchien

Das Hantieren mit Certificate Policys kann schnell kompliziert werden, wenn wir es mit einer CA-Hierarchie zu tun haben. Nehmen wir beispielsweise an, CA_1 stelle ein Zertifikat für CA_2 aus. Wenn sich CA_1 an eine bestimmte Certificate Policy hält, CA_2 aber nicht, dann steht CA_1 am Ende für Zertifikate gerade, die nicht nach ihren Richtlinien generiert worden sind. Damit dies nicht passiert, muss CA_1 CA_2 dazu verpflichten, die eigene Certificate Policy zu beachten oder zumindest eine gleichwertige Certificate Policy einzuhalten. In einer CA-Hierarchie gibt es daher zwangsläufig auch eine **Policy-Hierarchie**, zu der sich die Beteiligten ihre Gedanken machen müssen.

29.4.1 Hierarchietiefe

Um Certificate Policys über die Ebenen einer CA-Hierarchie hinweg durchsetzen zu können, stellt der X.509v3-Standard eine Reihe von Zertifikatsfeldern zur Verfügung. Es handelt sich dabei um mehrere X.509v3-Standarderweiterungen sowie um eine PKIX-spezifische X.509v3-Erweiterung. Alle relevanten Felder wurden bereits in Abschnitt 27.2.1 erwähnt. Zunächst ist das Zertifikatsfeld *Certificate Policies* von Bedeutung. Dieses kann beispielsweise in Alices Zertifikat stehen und dort die Kennung der Certificate Policy aufnehmen, die für das Zertifikat gültig ist (es können auch mehrere Policy-Kennungen in einem Zertifikat enthalten sein). Allerdings sagt dieses Feld nichts darüber aus, ob und wie eine Certificate Policy in einer CA-Hierarchie vererbt werden kann.

Ein weiteres wichtiges Zertifikatsfeld ist *Basic Constraints*. Dieses kommt nur in CA-Zertifikaten zum Einsatz. Darin wird festgelegt, wie viele Hierarchieebenen unterhalb der jeweiligen CA vorkommen dürfen. Wenn CA_1 ein Zertifikat für CA_2 ausstellt und dieses Feld dabei auf 0 setzt, dann darf CA_2 nur Endanwender-Zertifikate, aber keine CA-Zertifikate ausstellen. Tut CA_2 dies doch (z.B. für CA_3), dann sollte Bobs X.509v3-konforme Anwendung diese Policy-Verletzung erkennen und alle von CA_3 ausgestellten Zertifikate als ungültig ablehnen. Setzt CA_1 den Basic-Constraints-Wert dagegen auf 1, dann darf CA_2 zwar CA-Zertifikate (z.B. für CA_3) ausstellen, CA_3 darf dies jedoch nicht tun. Mit anderen Worten: Mit dem Feld *Basic Constraints* kann CA_1 verhindern, dass die CA-Hierarchie unter ihr beliebig viele Ebenen enthält.

29.4.2 Policy Mapping

Einer CA reicht es oft nicht aus, die Zahl der Hierarchieebenen zu begrenzen. Vielmehr kommt es der CA in der Regel zusätzlich darauf an, dass alle CAs unter ihr in der Hierarchie eine akzeptable Certificate Policy verwenden. Zur Durchsetzung dieses Ziels kann das X.509v3-Zertifikatsfeld *Policy Mapping* verwendet werden. Um dieses Feld zu erklären, nehmen wir wieder an, dass CA_1 ein Zertifikat für CA_2 ausstellt. CA_1 halte sich an die Certificate Policy CP_1, während CA_2 die Certificate Policy CP_2 beachtet. Wenn CA_1 mit CP_2 einverstanden ist, dann kann sie dies im CA-Zertifikat von CA_2 vermerken. Dazu verwendet sie das besagte Feld *Policy Mapping*, in dem sie in kodierter Form angibt, dass CP_1 und CP_2 aus ihrer Sicht gleichwertig sind.

Was das bringt, sehen wir, wenn wir uns in die Lage von Bob versetzen, der Alices Zertifikat überprüfen will, das von CA_2 ausgestellt worden ist. Da Bob nur CA_1 und deren Policy CP_1 kennt, kann er mit Alices Zertifikat zunächst nichts anfangen. Mithilfe des öffentlichen Schlüssels von CA_1 kann Bob jedoch das Zertifikat von CA_2 und damit auch Alices Zertifikat überprüfen. Außerdem sieht er im Policy-Mapping-Feld des Zertifikats von CA_2, dass CP_2 aus Sicht von CA_1 gleichwertig mit CP_2 ist. Da CP_2 in Alices Zertifikat angegeben ist, weiß er, dass er diesem vertrauen kann.

Für unser Beispiel nehmen wir als nächstes Folgendes an:

- CA_1 habe ein Zertifikat ohne Policy-Mapping-Feld.
- CA_2 habe ein Policy-Mapping-Feld mit dem Inhalt CP_1=CP_2.
- CA_3 habe ein Policy-Mapping-Feld mit dem Inhalt CP_2=CP_3.
- CA_4 habe ein Policy-Mapping-Feld mit dem Inhalt CP_3=CP_4.

Nun beziehen wir ein weiteres X.509v3-Feld in unsere Betrachtungen mit ein: *Policy Constraints*. Dieses sieht zwei mögliche Teilfelder vor: *inhibitPolicyMapping* und *requireExplicitPolicy*. Beide Teilfelder können jeweils die Werte 0, 1, 2, 3 usw. annehmen. *inhibitPolicyMapping* hat den Zweck, das Policy Mapping auf eine bestimmte Zahl von Hierarchiestufen zu begrenzen. Stehen im Zertifikat von CA_1 folgende Werte für *inhibitPolicyMapping*, dann ergeben sich mit der genannten Belegung der Policy-Mapping-Felder die jeweils angegebenen Auswirkungen:

- *inhibitPolicyMapping* = 0: kein Policy Mapping erlaubt.
- *inhibitPolicyMapping* = 1: Policy Mapping für CA_2 erlaubt.
- *inhibitPolicyMapping* = 2: Policy Mapping für CA_2 und CA_3 erlaubt.
- *inhibitPolicyMapping* = 3: Policy Mapping für CA_2, CA_3 und CA_4 erlaubt.

Das Teilfeld *requireExplicitPolicy* gibt an, ab welcher Hierarchie-Ebene eine Certificate Policy im CA-Zertifikat vorhanden sein muss. Die folgenden Aussagen gelten für das Zertifikat von CA_1 und die oben genannten Belegungen der Policy-Mapping-Felder:

- *requireExplicitPolicy* =
  0: Für CA_1, CA_2, CA_3 und CA_4 ist keine Policy-Kennung gefordert.
- *requireExplicitPolicy* =
  1: Für CA_2, CA_3 und CA_4 ist keine Policy-Kennung gefordert.
- *requireExplicitPolicy* =
  2: Für CA_3 und CA_4 ist keine Policy-Kennung gefordert.
- *requireExplicitPolicy* =
  3: Für CA_4 ist keine Policy-Kennung gefordert.

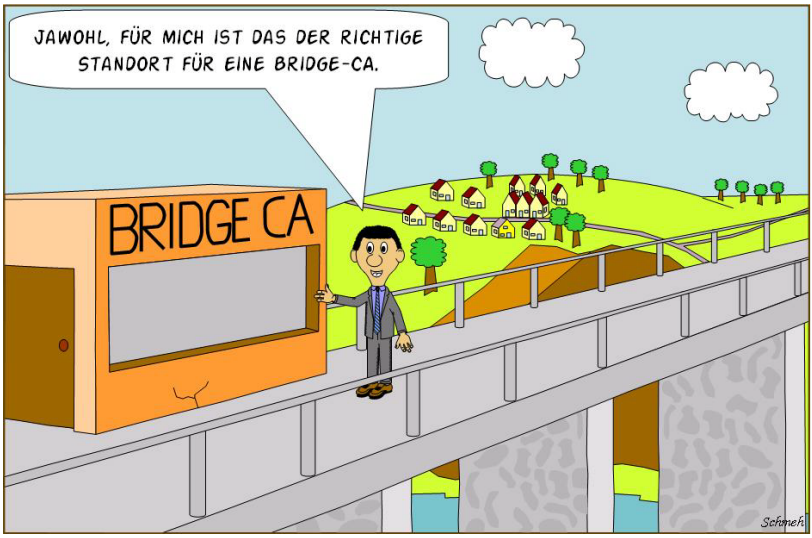
Es gibt noch ein weiteres X.509v3-Zertifikatsfeld, das an dieser Stelle von Belang ist: *Inhibit Any-Policy*. Dieses bezieht sich auf die **Any-Policy**, eine leere Certificate Policy, die eine eigene OID hat. Steht in einem Policy-Constraints-Feld, dass eine Policy gleichwertig mit der Any-Policy ist, dann bedeutet dies, dass jede Certificate Policy als gleichwertig akzeptiert wird. Der Zweck des Felds *Inhibit Any-Policy* ist es, die Verwendung der Any-Policy zu unterbinden. In unserem Beispiel hat dies folgende Konsequenzen:

- *inhibitAny-Policy* = 0: Any-Policy nicht anerkannt.
- *inhibitAny-Policy* = 1: Any-Policy wird nur in CA_1 anerkannt.
- *inhibitAny-Policy* = 2: Any-Policy wird in CA_1 und CA_2 anerkannt.
- *inhibitAny-Policy* = 3: Any-Policy wird in CA_1, CA_2 und CA_3 anerkannt.

29.4.3 Policy-Hierarchien in der Praxis

Möglicherweise werden Sie angesichts dieser komplexen Mapping-Arithmetik vermuten, dass Policy-Hierarchien kaum mehr als eine Spielerei sind. In der Tat spielt dieses Thema in der Praxis bisher keine große Rolle. Policy-Hierarchien fehlt einfach die Pragmatik, die für eine erfolgreiche Karriere im PKI-Sektor erforderlich ist. Da jedoch derzeit ein zaghaftes Zusammenwachsen von bisher isolierten PKIs zu verzeichnen ist und der Umgang mit Policys dadurch an Bedeutung gewinnt, könnte das Policy Mapping in den nächsten Jahren ebenfalls populärer werden.

30 Beispiel-PKIs



Mitte der neunziger Jahre, als die Entwicklung von Public-Key-Infrastrukturen noch in den Kinderschuhen steckte, träumten viele Kryptografen von einer weltweiten Zertifizierungshierarchie. An deren Wurzel sollte eine globale Wurzel-CA stehen, auf die sich alle digitalen Zertifikate dieser Welt zurückführen ließen. Jeder Mensch sollte dadurch auf einfache Weise mit einem beliebigen anderen Menschen sicher kommunizieren können. Heute wissen wir, dass eine solche allumfassende PKI keine Realität geworden ist und dies vermutlich auch nie werden wird (siehe dazu auch [EllSch]). Es gibt schlichtweg zu viele Beteiligte mit zu unterschiedlichen Interessen und zu unterschiedlichen Sicherheitsanforderungen, um eine solche Infrastruktur aufzubauen. Hinzu kommt, dass sich viele Unternehmen und Organisationen aus Imagegründen nicht in eine Hierarchie einordnen wollen. Statt einer weltweiten Super-PKI sind daher in den letzten Jahren viele kleinere Public-Key-Infrastrukturen entstanden. Die wichtigsten davon werden wir in diesem Kapitel betrachten.

30.1 Signaturgesetze und dazugehörende PKIs

Als Mitte der neunziger Jahre das Thema digitale Signaturen in das Bewusstsein der Öffentlichkeit vordrang, stellten sich viele eine Frage: Wird eine digitale Signatur vor Gericht als Beweismittel anerkannt? Die Anwort lautete überraschenderweise ja – zumindest war eine solche Anerkennung möglich. Da in Deutschland und zahlreichen anderen Ländern das Prinzip der freien Beweiswürdigung gilt, konnte ein Richter bereits damals eine digitale Signatur anerkennen, wenn die gegebenen Umstände dies plausibel machten. Ausnahmen gab es lediglich in einigen Spezialfällen, in denen das Gesetz eine Unterschrift von Hand ausdrücklich forderte – etwa auf einem Testament oder beim Kauf einer Immobilie.

Obwohl digitale Signaturen in den meisten Ländern auch ohne spezielles Gesetz juristisch anerkannt werden, haben inzwischen alle industrialisierten Staaten Rechtsvorschriften zum Thema digitale Signatur (meist **Signaturgesetze** genannt) erlassen. Der Sinn dieser Vorschriften ist offensichtlich: Anstatt den Umgang mit digitalen Signaturen der freien Beweiswürdigung und damit dem Ermessen eines Richters zu überlassen, wollten die Gesetzgeber klare Verhältnisse schaffen. Signaturgesetze sollen also für Rechtssicherheit sorgen. Betrachtet man die diversen Signaturgesetze in den verschiedenen Staaten, dann fällt auf, dass es sich dabei fast immer in gewissem Sinn um PKI-Gesetze handelt. Das heißt, dass im jeweiligen Gesetz und in den zugehörigen Rahmenbestimmungen Anforderungen an eine PKI genannt werden. In aller Regel ist es nicht generell vorgeschrieben, sich an diese Anforderungen zu halten. Stattdessen gilt: Wer sich an die Anforderungen hält, hat im Falle eines Rechtsstreits deutlich bessere Karten. Außerdem gibt es in vielen Ländern spezielle Situationen (z. B. Abgabe der Steuererklärung), bei denen nur digitale Signaturen zulässig sind, die dem jeweiligen Gesetz entsprechen.

30.1.1 EU-Signaturrichtlinie

Alle Signaturgesetze, die in den Staaten der Europäischen Union gelten, müssen der **EU-Signaturrichtlinie** aus dem Jahre 1999 entsprechen [EUSIG, BreWel]. Leider hatten die EU-Mitglieder seinerzeit recht unterschiedliche Vorstellungen davon, wie ein Signaturgesetz aussehen sollte, weshalb aus der EU-Signaturrichtlinie am Ende ein nicht gerade übersichtliches Kompromisswerk wurde. In der EU-Signaturrichtlinie werden Begriffe wie »Zertifizierungsdiensteanbieter« (damit ist ein Trust Center gemeint), »Zertifikat«, »elektronische Signatur« (damit wird eine Verallgemeinerung der digitalen Signatur bezeichnet) und »Signaturerstellungseinheit« eingeführt. Uns interessieren an dieser Stelle zunächst die Unterteilungen, die es für elektronische Signaturen gibt:

- **Elektronische Signatur**: Diese ist definiert als eine beliebige Information, die sich einer Person zuordnen lässt (also etwa deren Name im ASCII-Format).
- **Fortgeschrittene elektronische Signatur**: Diese bezeichnet eine digitale Signatur im üblichen Sinn.

Bei Zertifikaten entscheidet die EU-Signaturrichtlinie zwischen einfachen Zertifikaten und **qualifizierten Zertifikaten**. Außerdem gibt es einfache und sichere Signaturerstellungseinheiten. Den Gesetzgebern in den einzelnen Mitgliedstaaten macht die EU-Signaturrichtlinie folgende Vorgaben:

- Eine fortgeschrittene Signatur, die auf einem qualifizierten Zertifikat beruht, hat die gleiche Wirkung wie eine Unterschrift von Hand.
- Ein Diensteanbieter, der ein qualifiziertes Zertifikat ausstellt, haftet gegenüber jeder Person, die begründetermaßen auf dieses Zertifikat vertraut.
- EU-Mitgliedstaaten erkennen ihre qualifizierten Signaturen gegenseitig an.
- Ein Trust Center darf ohne behördliche Genehmigung (Akkreditierung) betrieben werden.

30.1.2 Deutsches Signaturgesetz

Deutschland war 1997 der erste Staat weltweit, der ein nationales Signaturgesetz einführte. Im Vorfeld hatte es heftige Diskussionen gegeben. Einige Hardliner forderten ein strenges Gesetz, das hohe Sicherheitsanforderungen an eine PKI stellt. Trust-Center-Betreiber sollten dadurch zu wirkungsvollen Schutzmaßnahmen animiert werden. Andere Experten lehnten ein derart strenges Gesetz dagegen ab. Sie wollten die Trust-Center-Betreiber durch geeignete Haftungsbestimmungen dazu bringen, auch ohne detaillierte Vorschriften ein hohes Maß an Sicherheit zu gewährleisten.

Am Ende setzten sich die Hardliner durch. Das 1997 in Kraft getretene erste deutsche Signaturgesetz forderte von jedem Trust-Center-Betreiber, der sich das Etikett »signaturgesetzkonform« umhängen wollte, zahlreiche teure Sicherheitsmaßnahmen. Erst zwei Jahre nach dem deutschen Signaturgesetz folgte die bereits erwähnte EU-Signaturverordnung. Sie zwang die Bundesregierung dazu, das deutsche Signaturgesetz anzupassen. Das resultierende Signaturgesetz von 2001 ist im Wesentlichen bis heute gültig. Es gibt darin vier Abstufungen, die an die EU-Signaturrichtlinie angelehnt sind [SigG, Reimer]:

- *Elektronische Signatur*: Als solche gilt – wie in der EU-Signaturrichtlinie – eine beliebige Information, die sich einer Person zuordnen lässt (also etwa deren Name im ASCII-Format).
- *Fortgeschrittene elektronische Signatur*: Dies ist – wie in der EU-Signaturrichtlinie – eine digitale Signatur im üblichen Sinne.

- *Qualifizierte elektronische Signatur*: Eine solche entspricht der fortgeschrittenen elektronischen Signatur mit qualifiziertem Zertifikat und sicherer Signaturerstellungseinheit aus der EU-Richtlinie.
- *Qualifizierte Signatur mit Anbieterakkreditierung*: Das deutsche Signaturgesetz von 1997 forderte, dass sich ein signaturgesetzkonformes Trust Center eine staatliche Genehmigung einholen musste. Die EU-Signaturrichtlinie schreibt jedoch fest, dass eine solche Genehmigung nicht notwendig ist. Deshalb erfand der deutsche Gesetzgeber die freiwillige Trust-Center-Akkreditierung. Besitzt ein Trust Center eine solche Akkreditierung, dann wird aus einer qualifizierten Signatur eine qualifizierte Signatur mit Anbieterakkreditierung – verwirrend, aber wahr.

Gesetz, Verordnung, Rahmenwerk

Das deutsche Signaturgesetz besteht aus 25 Paragrafen und ist recht abstrakt gehalten. Darin werden Begriffe wie »digitale Signatur«, »Zeitstempel« und »Zertifizierungsstelle« (Behördendeutsch für Trust-Center) definiert. Konkretere Angaben wurden aus dem Signaturgesetz in die aus 19 Paragrafen bestehende **Signaturverordnung** ausgelagert, die von der Bundesregierung beschlossen wurde und von dieser geändert werden kann [SigV]. In der Signaturverordnung werden beispielsweise Angaben zum Betrieb von Trust-Centern, Genehmigungsverfahren, Sicherheitskonzepten und Kosten für die Genehmigung gemacht. Weitere Informationen finden sich im sogenannten **Rahmenwerk** [SigR].

Welche Krypto-Verfahren im Zusammenhang mit dem Signaturgesetz verwendet werden dürfen, wird im Bundesanzeiger veröffentlicht [SigB]. Laut Veröffentlichung vom 18. Januar 2012 sind RSA, DSA und drei ECDSA-Varianten zulässig. Als kryptografische Hashfunktionen werden RIPEMD-160, SHA-1, SHA-224, SHA-256, SHA-384 und SHA-512 genannt. Schließlich äußert sich die Veröffentlichung noch zum Thema Zufallsgenerierung, wobei sowohl echte Zufallsgeneratoren als auch Pseudozufallsgeneratoren unter jeweils bestimmten Umständen zulässig sind.

Der Standard, der die Anforderungen des Signaturgesetzes in konkrete Formate und Verfahren umsetzen soll, ist der in Abschnitt 26.4.3 erwähnte Common-PKI-Standard [Common]. Wer also ein Trust Center betreiben will, das qualifizierte Zertifikate ausstellt, muss sich durch einen dicken Papierstapel kämpfen: Gesetz, Verordnung, Rahmenwerk, Bundesanzeiger-Veröffentlichungen und die Common-PKI-Spezifikation – die deutsche Gründlichkeit überließ nichts dem Zufall.

Will der Betreiber einer PKI qualifizierte Zertifikate nach dem deutschen Signaturgesetz ausstellen, dann muss er für die CA einen besonders hohen Sicherheitsstandard einhalten. Dazu gehören bauliche Maßnahmen (etwa Sicherheitstüren, Vereinzelungsschleusen an den Eingängen und dicke Wände) und organisatorische Maßnahmen (Vier-Augen-Prinzip, Dokumentation aller Vorgänge).

Außerdem ist für die sicherheitskritischen Softwarebestandteile der CA eine Evaluierung nach Common Criteria oder ITSEC gefordert.

Anwendung des Signaturgesetzes

Das deutsche Signaturgesetz ist für sich genommen eine reine Kann-Bestimmung. Es schreibt niemandem vor, dass er eine fortgeschrittene oder qualifizierte Signatur (mit oder ohne Anbieterakkreditierung) zu verwenden hat. Im Falle eines Rechtsstreits gilt nach wie vor die freie Beweiswürdigung. Dennoch ist es von Vorteil, beispielsweise qualifizierte Signaturen einzusetzen, denn es ist nicht zu erwarten, dass ein Richter einer solchen im Streitfall die Anerkennung versagt. Wer auf qualifizierte Signaturen verzichtet, darf sich dagegen nicht wundern, wenn er vor Gericht Schwierigkeiten mit der Anerkennung bekommt.

Seine eigentliche Wirkung erzielt das deutsche Signaturgesetz dadurch, dass in den letzten Jahren zahlreiche andere Gesetze, die bis dahin eine Unterschrift von Hand forderten, um digitale Signaturen erweitert wurden. So heißt es beispielsweise in § 3a des Verwaltungsverfahrensgesetzes: »Eine durch Rechtsvorschrift angeordnete Schriftform kann, soweit nicht durch Rechtsvorschrift etwas anderes bestimmt ist, durch die elektronische Form ersetzt werden. In diesem Fall ist das elektronische Dokument mit einer qualifizierten elektronischen Signatur nach dem Signaturgesetz zu versehen.« Auch das Bundesgesetzbuch und das Prozessrecht enthalten ähnliche Stellen. Dadurch ist es inzwischen in vielen Bereichen des täglichen Lebens möglich, eine Unterschrift durch eine digitale Signatur zu ersetzen. Beispielsweise ist es für Unternehmen unter bestimmten Voraussetzungen erlaubt, Vorsteuerabzug auf digital signierten Rechnungen geltend zu machen.

Die Signaturgesetz-PKI

Auf Grundlage des deutschen Signaturgesetzes und des Common-PKI-Standards ist eine eigene Signaturgesetz-PKI entstanden. Es handelt sich dabei um eine zweistufige CA-Hierarchie. An deren Wurzel steht die Wurzel-Zertifizierungsstelle der Bundesnetzagentur. Diese hat für jedes akkreditierte deutsche Trust Center ein CA-Zertifikat ausgestellt. Mitte 2012 waren folgende neun akkreditierte Trust Center aktiv: Deutsche Post, Deutsche Telekom, Bundesnotarkammer, Deutsches Gesundheitsnetz, TC TrustCenter, Deutscher Sparkassen Verlag, D-Trust (Bundesdruckerei), DATEV, Medisign und AuthentiDate (nur Zeitstempeldienstanbieter). Es gibt zudem einige nichtakkreditierte Trust Center, die qualifizierte Zertifikate nach Signaturgesetz anbieten. Deren öffentliche CA-Schlüssel sind nicht von der Wurzel-CA der Bundesnetzagentur zertifiziert. Sie gehören daher nicht zur Signaturgesetz-PKI.

30.1.3 Österreichisches Signaturgesetz

Nachdem die EU-Signaturrichtlinie 1999 in Kraft trat, war Österreich das erste Land, das ein dieser Richtlinie entsprechendes Signaturgesetz einführte [MenSch]. Dieses trägt den offiziellen Namen *Bundesgesetz über elektronische Signaturen* und wurde 1999 verabschiedet [SigGÖ]. Ein Jahr später kam die österreichische Signaturverordnung dazu [SigVÖ]. Das österreichische Signaturgesetz setzt die EU-Signaturrichtlinie recht genau um. Es gibt also einfache und fortgeschrittene Signaturen, außerdem einfache und qualifizierte Zertifikate sowie einfache und sichere Signaturerstellungseinheiten. Eine fortgeschrittene Signatur mit qualifiziertem Zertifikat und sicherer Signaturerstellungseinheit wird als qualifizierte Signatur (früher: sichere Signatur) bezeichnet. Sie entspricht der qualifizierten Signatur aus dem deutschen Signaturgesetz.

Wie im deutschen Signaturgesetz kann sich ein österreichisches Trust Center von einer Behörde (die zur Rundfunk und Telekom Regulierungs-GmbH gehörende **Telekom-Control-Kommission**) akkreditieren lassen. Es gibt jedoch auch hier keine Akkreditierungspflicht. Die Akkreditierung spielt in Österreich eine geringere Rolle als in Deutschland, da das österreichische Signaturgesetz keine Unterscheidung zwischen einer qualifizierten Signatur mit und ohne Anbieterakkreditierung vorsieht.

2008 trat eine Neufassung des österreichischen Signaturgesetzes in Kraft [SigGÖÄ]. Auch die Signaturverordnung wurde erneuert [SigVÖÄ]. Die neue Gesetzesversion schrieb einige kleinere Änderungen fest und ersetzte den bis dahin verwendeten Begriff »sichere Signatur« durch den bedeutungsgleichen Ausdruck »qualifizierte Signatur«. Dies entspricht der Ausdrucksweise in anderen EU-Ländern.

30.1.4 Schweizer ZertES

Die Schweiz ist als Nicht-EU-Mitglied nicht an die EU-Signaturrichtlinie gebunden. Trotzdem ist die schweizerische Signaturgesetzgebung eng an diese Richtlinie angelehnt. Das Schweizer Signaturgesetz heißt *Bundesgesetz über die elektronische Signatur* (**ZertES**) [SigGCH], die zugehörige Verordnung ist die **VZertES** [SigVCH]. Das ZertES trat erst Anfang 2005 in Kraft, als alle EU-Staaten die EU-Signaturrichtlinie bereits umgesetzt hatten. Es gab zuvor allerdings andere gesetzliche Bestimmungen zur digitalen Signatur in der Schweiz. Nach Vorbild der EU-Signaturrichtlinie sieht das ZertES einfache, fortgeschrittene und qualifizierte Signaturen vor, wobei eine qualifizierte Signatur eine sichere Signaturerstellungseinheit und ein qualifiziertes Zertifikat erfordert.

Die in der EU-Signaturrichtlinie geforderte Gleichstellung einer qualifizierten digitalen Signatur mit einer handschriftlichen Unterschrift findet sich nicht in der ZertES selbst, sondern im Obligationenrecht (dieses ist ein Teil des Zivilgesetzbuchs). Im Gegensatz zur EU-Signaturrichtlinie schreibt das ZertES vor, dass ein

Trust Center eine Akkreditierung von einer zuständigen Behörde benötigt, wenn es qualifizierte Zertifikate ausstellen will.

30.1.5 Fazit

Der Aufwand, der in den letzten 15 Jahren in das Thema digitale Signaturen und deren rechtliche Einordnung gesteckt wurde, steht leider in keinem Verhältnis zum daraus resultierenden Erfolg. Das Interesse der Endanwender war bisher gering, und nur wenige Behörden und Unternehmen entschieden sich für den Einsatz qualifizierter Signaturen. Größter Hoffnungsträger sind im Moment Projekte, die elektronische Personalausweise, Krankenversichertenkarten oder Dienstausweise mit der Fähigkeit zum digitalen Signieren vorsehen. Derartige Vorhaben gibt es derzeit in mehreren EU-Ländern, wobei in der Regel auch qualifizierte Signaturen unterstützt werden. Tatsächlich könnten die Signaturgesetze der EU-Staaten dadurch in den nächsten Jahren erheblich an Bedeutung gewinnen. Im folgenden Abschnitt 30.2 werden wir einige dieser Projekte genauer anschauen.

30.2 Die PKIs elektronischer Ausweise

Wie Sie aus Abschnitt 23.3 wissen, geben einige Staaten Reisepässe, Personalausweise, Krankenversicherungskarten oder ähnliche Dokumente aus, die mit privaten Schlüsseln bestückt sind (einen Überblick zu diesem Thema finden Sie in [Schm09/2]). Zur Ausweis-Infrastruktur gehört hierbei stets eine PKI, die digitale Zertifikate zu den privaten Schlüsseln ausstellt. Wie man sich leicht vorstellen kann, zählen die diversen PKI-Projekte, die zu diesem Zweck in den letzten Jahren in vielen Ländern gestartet wurden, zu den größten überhaupt. Die Umsetzung solcher Vorhaben ist ein Milliardengeschäft.

30.2.1 Die PKI des elektronischen Reisepasses

Elektronische Reisepässe sind in Dutzenden von Ländern bereits Realität. Erfreulicherweise gibt es eine internationale Standardisierung, wodurch praktisch jedes Prüfgerät der Welt mit jedem beliebigen Reisepass zurechtkommt. Insbesondere kann jedes Prüfgerät die digitale Signatur verifizieren, die sich auf einem elektronischen Reisepass befindet. Diese Signatur stammt von der Ausweisbehörde des jeweiligen Landes. Für die Beglaubigung des zugehörigen öffentlichen Schlüssels sorgt eine internationale PKI, die speziell für diesen Zweck aufgebaut wurde. Jeder Staat, der elektronische Reisepässe ausgibt, betreibt innerhalb dieser PKI eine CA, die als **CSCA** (Country Signing Certification Authority) bezeichnet wird. Für den Betrieb der CSCA ist eine nationale Sicherheitsbehörde (in Deutschland beispielsweise das BSI) zuständig. Leider gibt es aus politischen Gründen inner-

halb dieser Reisepass-PKI keine Wurzel-CA, die Zertifikate für CSCAs ausstellt. Auch eine Bridge-CA ist nicht vorgesehen.

Eine CSCA stellt für jeden Ausweishersteller des Landes (in Deutschland ist die Bundesdruckerei der einzige) ein digitales Zertifikat aus, das **DS-Zertifikat** genannt wird (DS steht für »digital signing«). Mit dem zum DS-Zertifikat gehörenden privaten Schlüssel signiert der Ausweishersteller die Daten auf den ausgestellten Reisepässen. Ein Prüfgerät muss zur Überprüfung einer solchen Signatur das CSCA-Zertifikat des betreffenden Staats kennen. Mit diesem kann es die Echtheit des betreffenden DS-Zertifikats prüfen und anschließend die Signatur verifizieren.

Neben der digitalen Signatur hat jeder elektronische Reisepass einen privaten Schlüssel für eine Challenge-Response-Authentifizierung gespeichert. Für diesen privaten Schlüssel ist kein Zertifikat vorgesehen. Stattdessen ist der zugehörige öffentliche Schlüssel Teil des signierten Reisepassinhalts. Ein Prüfgerät kann dadurch im Rahmen der Signaturverifikation zunächst die Echtheit des öffentlichen Schlüssels sicherstellen und mit diesem anschließend eine Authentifizierung durchführen.

Optional kann ein elektronischer Reisepass die Identität eines Prüfgeräts überprüfen, um bestimmte Inhalte (etwa biometrische Daten) nur unter geeigneten Umständen preiszugeben. Diese Funktion wird **Extended Access Control** (**EAC**) genannt. EAC setzt voraus, dass ein Prüfgerät ein eigenes digitales Zertifikat besitzt. Dieses ist ein CV-Zertifikat (siehe Abschnitt 27.5) und kommt von einer CA, die von einem sogenannten **Document Verifier** (**DV**) betrieben wird. Ein DV ist eine organisatorische Einheit, die innerhalb eines Staats die Prüfgeräte eines bestimmten Typs betreibt (beispielsweise alle Prüfgeräte, die an der Grenze zum Einsatz kommen). Die DV erhält wiederum ein Zertifikat, das von einer **CVCA** (Country Verifying CA) ausgestellt wird, die vom jeweiligen Staat betrieben wird. Auch hier ist keine weltweite Wurzel-CA vorgesehen. Der Chip auf dem Reisepass muss daher die öffentlichen CVCA-Schlüssel derjenigen Staaten gespeichert haben, in denen ein EAC-Zugriff möglich sein soll.

30.2.2 PKIs elektronischer Personalausweise

Während elektronische Reisepässe weltweit einheitlich realisiert sind, herrscht im Bereich der elektronischen Personalausweise ein ziemliches Durcheinander. Mehrere Dutzend Staaten weltweit geben inzwischen solche Dokumente an ihre Bürger aus, und erst langsam gewinnen Initiativen wie *European Citizen Card* an Bedeutung, die eine Interoperabilität unterschiedlicher Ausweissysteme zum Ziel haben. Zu den europäischen Staaten, die bereits elektronische Personalausweise eingeführt haben, gehören Belgien, Deutschland, Finnland, Italien, Portugal und Spanien. Auch einige arabische und südostasiatische Länder haben diesen Schritt inzwischen vollzogen. Weitere werden folgen.

Gemeinsam haben praktisch alle elektronischen Personalausweise, dass sie private Schlüssel auf dem Ausweischip vorsehen, die sich (anders als beim Reisepass) nicht nur für eine Echtheitskontrolle nutzen lassen, sondern auch zum digitalen Signieren und zum Zugriff auf Online-Angebote. Zu diesem Zweck haben die jeweiligen Staaten meist eigene CAs eingerichtet, die Zertifikate zu den privaten Schlüsseln ausgeben. Eine internationale Wurzel-CA, die durchaus sinnvoll wäre, ist leider nicht in Sicht.

30.2.3 PKIs elektronischer Krankenversichertenkarten

Elektronische Gesundheitskarten mit PKI-Unterstützung sind noch nicht so weit verbreitet wie elektronische Personalausweise. Allerdings ist auch dieses Segment am Wachsen, denn der Nutzen von Signaturen und zertifikatsbasierter Authentifizierung im Gesundheitswesen ist offensichtlich. Die Anwendungsmöglichkeiten reichen vom elektronischen Rezept über die elektronische Krankenakte bis zum elektronischen Impfpass. Leider sind wir von einer internationalen Standardisierung noch weit entfernt.

Das weltweit umfangreichste Kartensystem im Gesundheitswesen entsteht derzeit in Deutschland, und zwar in Form der elektronischen Gesundheitskarte. Diese ist Bestandteil einer umfangreichen Infrastruktur (Telematikinfrastruktur), die zahlreiche neue Dienste im deutschen Gesundheitswesen ermöglichen soll. Im Zusammenhang mit der elektronischen Gesundheitskarte kommen an mehreren Stellen private Schlüssel und dazu gehörende digitale Zertifikate zum Einsatz. Um diese ausstellen und verwalten zu können, gehört eine PKI zu den festen Bestandteilen der Telematikinfrastruktur. Dabei handelt es sich um eine der komplexesten und ambitioniertesten PKIs der Welt. Es werden sechs unterschiedliche Zertifikatstypen unterschieden, für die es jeweils eigene CAs gibt. Neben X.509-Zertifikaten sind auch CV-Zertifikate mit im Spiel. Für die X.509-Zertifikate sind keine übergeordneten Zertifizierungsstellen vorgesehen. Stattdessen wird jede CA in eine von zwei Listen (Trust-Lists) eingetragen, die von einer CA (Gematik Bridge CA) signiert werden.

Während in Deutschland noch der Aufbau läuft, ist in Österreich eine vergleichbare Krankenversichertenkarte in Form der e-Card bereits Realität. Auch die e-Card unterstützt asymmetrische Kryptografie, allerdings ist die zugehörige PKI deutlich weniger komplex. Im Wesentlichen gibt es eine CA, die Zertifikate für alle Karten ausstellt. Darüber hinaus lassen sich die Karten optional mit zusätzlichen Schlüsseln bestücken. Diese Aufgabe übernimmt ein öffentliches Trust Center. In der Schweiz gibt es ebenfalls eine elektronische Gesundheitskarte mit PKI-Unterstützung. Noch befindet sich diese jedoch in einem frühen Stadium.

30.3 Weitere PKIs

Unabhängig von Signaturgesetzen und Bürgerkartenprojekten sind in den letzten Jahren zahlreiche weitere Public-Key-Infrastrukturen entstanden. Einige davon wurden von Unternehmen oder Behörden für den Eigenbedarf eingerichtet. Andere bieten ihre Dienste am Markt an. Die folgenden Abschnitte geben einen Überblick.

30.3.1 Organisationsinterne PKIs

Zahlreiche Organisationen haben für ihre Zwecke interne PKIs aufgebaut, die die eigenen Mitarbeiter sowie Geschäftspartner mit Zertifikaten versorgen. Leider sind solche organisationsinternen PKIs meist Insellösungen, die nicht in eine CA-Hierarchie integriert sind. Die meisten Betreiber kümmern sich zunächst einmal um einen reibungslosen Aufbau und Betrieb, bevor sie sich für Hierarchien und Zertifizierungspfade interessieren. Zudem sind viele organisationsinterne PKIs nur für einen bestimmten Zweck oder eine bestimmte Abteilung gedacht. Viele Unternehmen nehmen beispielsweise den Aufbau eines VPN oder eines Wireless LAN zum Anlass, um für die jeweiligen Anwender digitale Zertifikate auszustellen. Auf ähnliche Weise entstanden CAs für Webportale oder für die Dateiverschlüsselung. In der Regel werden diese PKIs nicht strategisch geplant, sondern als Bestandteil eines anderen Projekts umgesetzt. Daher gibt es heute zahlreiche Unternehmen, die mehrere PKIs gleichzeitig betreiben, ohne dass eine Cross-Zertifizierung oder eine übergeordnete CA existiert. Zwar gibt es meist Bestrebungen, diese Insellösungen miteinander zu verbinden, doch das dauert seine Zeit.

Ohnehin ist nicht zu übersehen, dass fast alle Unternehmen und Behörden einen großen Pragmatismus an den Tag legen, wenn es um Public-Key-Infrastrukturen geht. So verzichten nahezu alle deutschen Betreiber organisationsinterner PKIs auf den flächendeckenden Einsatz qualifizierter Zertifikate nach Signaturgesetz. Auch Smartcards, HSMs, CPS/CP-Dokumente, die Verwendung mehrerer Schlüsselpaare pro Anwender und andere aus Sicherheitssicht sinnvolle PKI-Bestandteile sind nicht selbstverständlich, sondern werden nach Kosten-Nutzen-Überlegungen eingesetzt. Die PKI-Branche musste dadurch lernen, dass ihre Kunden nicht an einer bestimmten Technik, sondern an der Lösung bestimmter Probleme interessiert war. Statt für eine Hochsicherheitslösung entschieden sich die PKI-Betreiber nahezu immer für einen praktikablen Kompromiss zwischen Sicherheit, Kosten und Benutzerfreundlichkeit.

Immerhin gibt es Fortschritte. Die am Markt verfügbaren PKI-Produkte sind in den letzten Jahren immer stabiler und praxistauglicher geworden. Die zahlreichen kleinen PKI-Projekte haben viele Unternehmen dazu veranlasst, ihre PKI-Aktivitäten zu koordinieren und das Thema strategisch anzugehen. Derweil haben längst auch Abteilungen, die nicht unmittelbar mit der IT-Sicherheit befasst sind, die Vorteile einer PKI zu schätzen gelernt, was PKI-Aufbauprojekte

deutlich erleichtert. Das Idealbild einer weltweiten CA-Hierarchie werden wir zwar trotzdem wohl nicht erreichen. Digitale Zertifikate als alltägliche Hilfsmittel, die für eine höhere Sicherheit sorgen, sind jedoch heute schon Realität.

30.3.2 Kommerzielle Trust Center

Neben organisationsinternen PKIs gibt es auch zahlreiche Public-Key-Infrastrukturen, in deren Mittelpunkt eine kommerziell betriebene CA (und damit ein Trust Center) steht. Diese Trust Center bieten Zertifikate am Markt an. Theoretisch kann jede Privatperson dort ein Zertifikat erwerben, doch in der Praxis sind es bisher nur Unternehmen und Behörden, die ihre Mitarbeiter mit Schlüsselpaaren ausstatten wollen, ohne dazu eine eigene CA aufzubauen.

Im Vergleich zu den unternehmensintern betriebenen PKIs sind die CAs kommerzieller Trust Center meist deutlich aufwendiger und funktionsreicher. Praktisch alle Trust-Center-Betreiber realisieren einen hohen Sicherheitsstandard, verwenden ein HSM für die Schlüsselspeicherung und bieten die Nutzung von Smartcards. Die meisten deutschen Trust-Center-Betreiber stellen auf Wunsch qualifizierte Zertifikate nach Signaturgesetz aus und haben eine Anbieterakkreditierung. Qualifizierte Zertifikate sind jedoch bisher nicht gerade ein Verkaufsschlager, weshalb alle Anbieter auch nichtqualifizierte Zertifikate im Angebot haben. Hier eine Liste der wichtigsten Trust-Center-Betreiber:

- *A-Trust*: Dies ist das bedeutendste österreichische Trust Center. Es besitzt eine Akkreditierung nach österreichischem Signaturgesetz.
- *D-Trust*: D-Trust ist eine Tochter der Bundesdruckerei. Das Unternehmen betreibt ein Trust Center, das nach Signaturgesetz akkreditiert ist.
- *Signtrust*: Signtrust ist das Trust Center der Deutschen Post. Es besitzt eine Akkreditierung nach Signaturgesetz.
- *Swisscom*: Das Schweizer Gegenstück zur Deutschen Telekom betreibt ein nach Schweizer Signaturgesetz (ZertES) akkreditiertes Trust Center.
- *SwissSign*: SwissSign gehört zur schweizerischen Post und ist ein ebenfalls nach Schweizer Signaturgesetz (ZertES) akkreditiertes Trust Center.
- *TC TrustCenter*: Dieses in Hamburg lokalisierte Trust Center zählte zu den CA-Betreibern der ersten Stunde. Heute gehört es zu Symantec. TC TrustCenter besitzt eine Akkreditierung nach Signaturgesetz.
- *QuoVadis*: Dies ist ein weiteres Schweizer Trust Center mit Akkreditierung nach ZertES.
- *T-Systems*: Das von T-Systems betriebene Trust Center der Deutschen Telekom zählt zu den wichtigsten Anbietern in Deutschland. T-TeleSec besitzt eine Akkreditierung nach Signaturgesetz.

- *Verisign*: Die US-Firma Verisign ist Betreiber des ältesten Trust Centers am Markt. Verisign ging 1995 an den Start und hat sich bis heute als weltweit größter CA-Betreiber behaupten können.

Einige Trust Center sind inzwischen wieder vom Markt verschwunden. Es ist bisher schwierig, mit dem Betrieb eines Trust Centers Geld zu verdienen.

30.4 Übergreifende PKIs

Auch wenn es wohl nie eine weltweite, allumfassende CA-Hierarchie geben wird, so gibt es derzeit immerhin einige Projekte, die uns diesem Idealziel wenigstens einen kleinen Schritt näher bringen. Die Rede ist von Vorhaben, die den Aufbau einer CA-Hierarchie oder den systematischen Einsatz von Cross-Zertifizierungen zum Inhalt haben. Die folgenden Abschnitte nennen die wichtigsten Beispiele.

30.4.1 European Bridge-CA

Die **European Bridge-CA** ist eine Einrichtung, die im Oktober 2000 von der Deutschen Bank und der Deutschen Telekom ins Leben gerufen wurde (im Internet unter www.bridge-ca.de zu finden). Im Januar 2001 ging sie als Projekt des Branchenverbands Teletrust in Betrieb. Dem Namen entsprechend handelt es sich dabei um eine Bridge-CA – also um eine CA, die Cross-Zertifizierungen mit anderen CAs eingeht, um PKI-übergreifende Zertifikatsketten bilden zu können (siehe Abschnitt 26.3.4). Das Ziel der European Bridge-CA ist es, möglichst viele europäische PKIs durch eine Cross-Zertifizierung miteinander zu verbinden. Bisher haben etwa ein Dutzend Unternehmen und Behörden im deutschsprachigen Raum eine Cross-Zertifizierung mit der European Bridge-CA durchgeführt. Auf diese Weise ist eines der größten zusammenhängenden PKI-Systeme der Welt entstanden. Zu den beteiligten Organisationen gehören unter anderem die Deutsche Bank und die Deutsche Telekom. Außerhalb von Deutschland, Österreich und der Schweiz gibt es bisher keine Mitglieder.

Die Notwendigkeit für eine Einrichtung wie die European Bridge-CA ist derzeit kaum zu übersehen. Zahlreiche Unternehmen bauen zurzeit eine interne PKI auf, wollen sich jedoch nicht einer Wurzel-CA unterordnen. Die European Bridge-CA sorgt daher für gegenseitige Anerkennung, ohne dass eine große Anzahl von Cross-Zertifizierungen notwendig ist. Die Anforderungen, die an Bridge-CA-Teilnehmer gestellt werden, sind aus diesem Grund recht pragmatisch gehalten.

30.4.2 Verwaltungs-PKI

Die **Verwaltungs-PKI** ist eine vom Bundesamt für Sicherheit in der Informationstechnik (BSI) initiierte Public-Key-Infrastruktur [V-PKI]. Sie hat das Ziel, von deutschen Behörden und öffentlichen Einrichtungen betriebene CAs zu einer Hierarchie zusammenzufassen. Die Wurzel der Verwaltungs-PKI wird vom BSI selbst betrieben. Diese Wurzel-CA ist mit der European Bridge-CA verbunden, wodurch eine Verbindung zwischen der öffentlichen Verwaltung und zahlreichen Wirtschaftsunternehmen gegeben ist. Eine CA, die an der Verwaltungs-PKI teilnimmt, lässt sich von der Wurzel-CA ein CA-Zertifikat ausstellen. Es ist zulässig, dass eine solche CA selbst Zertifikate für untergeordnete CAs ausstellt.

Die Verwaltungs-PKI sieht nicht vor, dass die beteiligten CAs qualifizierte Zertifikate nach Signaturgesetz ausstellen. Dies ist auf den ersten Blick erstaunlich, da man von Behörden erwarten würde, dass sie die entsprechenden Vorschriften des Signaturgesetzes beachten. Wie in Abschnitt 30.1.2 erklärt, stellt das Signaturgesetz jedoch sehr hohe und oftmals unpraktikable Anforderungen, die auch für Behörden nur schwer zu erfüllen sind. Man kann die Verwaltungs-PKI daher als eine Kompromisslösung betrachten: Einerseits sah das BSI ein, dass qualifizierte Zertifikate für den flächendeckenden Einsatz in den Behörden nicht geeignet sind. Andererseits wollte man den PKI-Aufbau im öffentlichen Sektor nicht völlig dem Zufall überlassen. Deshalb schuf das BSI die Verwaltungs-PKI und gleichzeitig ein Regelwerk, das eine Reihe von Anforderungen vorschreibt, die eine CA erfüllen muss, wenn sie sich in diese Hierarchie eingliedern will. Diese Anforderungen sind deutlich lockerer als die des Signaturgesetzes, andererseits aber deutlich strenger als die der European Bridge-CA.

30.4.3 Wurzel-CAs

Wenn Krypt & Co. eine eigene CA betreibt und mit dieser ein SSL-Zertifikat für einen Webserver ausstellt, steht die Firma vor folgendem Problem: Die gängigen Webbrowser kennen die CA von Krypt & Co. nicht und zeigen daher eine Warnmeldung an, wenn Anwenderin Alice auf eine SSL-geschützte Seite zugreifen will. Ähnliches passiert, wenn Bob an Alice eine verschlüsselte E-Mail schicken will, sein E-Mail-Programm jedoch Alices CA nicht kennt. Ein solches Problem lässt sich zwar durch einen Import des CA-Zertifikats von Krypt & Co. in den Webbrowser bzw. in das E-Mail-Programm beheben, doch diese nur wenige Mausklicks erfordernde Aktion ist für viele Anwender bereits zu kompliziert.

Alternativ kann Krypt & Co. den Hersteller des E-Mail-Programms bzw. des Browsers (z.B. Microsoft) darum bitten, das CA-Zertifikat schon ab Werk in die Liste der unterstützten CAs aufzunehmen. Einen solchen Service wird der Hersteller jedoch kaum bieten, wenn das entsprechende Unternehmen nicht dafür bezahlt (mir sind Fälle bekannt, in denen der CA-Betreiber bis zu 500.000 Euro bezahlte, um in einen Browser aufgenommen zu werden).

Es gibt jedoch noch eine weitere Alternative: Krypt & Co. lässt das firmeneigene CA-Zertifikat von einer Wurzel-CA zertifizieren, deren Zertifikat bereits in die gängigen Browser und E-Mail-Programme aufgenommen worden ist. Alle gängigen Browser und E-Mail-Programme erkennen, dass ein SSL-Zertifikat von einer CA signiert worden ist, deren Zertifikat von einer bekannten Wurzel-CA ausgestellt worden ist. Die unschöne Warnmeldung entfällt dadurch. Leider haben nicht alle in den Browsern und E-Mail-Programmen gelisteten CA-Betreiber Lust, ihre privilegierte Stellung an andere CAs zu vermitteln. Einige (z.B. GlobalSign, QuoVadis und TC TrustCenter) tun dies aber doch und verlangen für diesen Service meist einige Zehntausend Euro im Jahr.

Teil 5

Kryptografische Netzwerkprotokolle

31 Kryptografie im OSI-Modell

Kryptografische Netzwerkprotokolle mit ihren diversen Eigenschaften haben Sie bereits in Kapitel 20 kennengelernt. An dieser Stelle wollen wir uns einen Überblick über die wichtigsten davon verschaffen, wobei das Internet als bedeutendstes Computernetz im Mittelpunkt stehen soll. Diesen Überblick gewinnt man am besten mit dem sogenannten **OSI-Modell**. Dieses ist ein von der Standardisierungsorganisation ISO genormtes Modell für das Zusammenspiel von Netzwerkprotokollen, das sich auf nahezu alle Computernetze anwenden lässt. Die Abkürzung OSI steht für Open Systems Interconnection.

31.1 Das OSI-Modell

Das OSI-Modell sieht bewusst kein Superprotokoll vor, das alleine die gesamte Kommunikation regelt. Vielmehr kommen zwischen den Beteiligten stets mehrere Protokolle zum Einsatz. Jedes Protokoll gehört zu einer von sieben Klassen

(genannt **Schichten**), die streng voneinander abgegrenzt sind. Die Schichten des OSI-Modells sind von 1 bis 7 durchnummeriert und befolgen vor allem zwei Grundsätze:

- Je mehr ein Protokoll mit dem physikalischen Signal zu tun hat, das über den Übertragungskanal fließt, desto niedriger ist die Schicht, der es angehört. Je mehr ein Protokoll mit einem Anwendungsprogramm zu tun hat, das die Kommunikation nutzt, desto höher ist die Schicht, zu der es gehört.
- Jedes Protokoll einer bestimmten Schicht verwendet ausschließlich das Protokoll der darunter liegenden Schicht und keine anderen Protokolle. Damit hat nur das Protokoll der Schicht 1 direkt mit der Hardware zu tun und nur das Protokoll der Schicht 7 direkt mit dem Anwendungsprogramm.

31.1.1 Die Schichten des OSI-Modells

Die sieben Schichten des ISO-OSI-Modells haben im Einzelnen folgende Aufgaben:

- *Schicht 1* (**Bit-Übertragungsschicht**) wandelt die Signale, die aus dem Übertragungskanal kommen, in Nullen und Einsen um. Dies bedeutet, dass jeder Draht, der zur Datenübertragung benutzt wird, an seinen Enden ein Gerät benötigt, das ein Schicht-1-Protokoll verwendet.
- *Schicht 2* (**Sicherungsschicht**) nimmt die Nullen und Einsen aus Schicht 1 entgegen, sucht über Prüfsummen nach Fehlern und fasst Bit-Gruppen zusammen (beispielsweise zu ASCII-Zeichen). Auch Schicht-2-Protokolle kommen praktisch an jedem Drahtende zur Anwendung.
- *Schicht 3* (**Vermittlungsschicht**) entscheidet, wohin die von Schicht 2 erhaltenen Zeichen weitergeleitet werden sollen. Diese Wegewahl (Routing) ist eine Aufgabe der Router, die also ein Schicht-3-Protokoll abarbeiten. Router (und damit Schicht-3-Protokolle) kommen hauptsächlich an Netzknoten zum Einsatz, an denen mehr als zwei Leitungen aufeinandertreffen.
- *Schicht 4* (**Transportschicht**) bietet mithilfe von Schicht 3 eine Verbindung zwischen dem Sender und Empfänger an, bei der die Router dazwischen nicht bemerkt werden. Schicht-4-Protokolle werden daher vor allem von solchen Netzknoten abgearbeitet, die Daten nicht nur weiterleiten, sondern sie selbst verwenden (dies gilt auch für die Schichten 5 bis 7).
- *Schicht 5* (**Kommunikationssteuerungsschicht**) steuert die Kommunikation auf der Verbindung, die Schicht 4 herstellt. Hier wird beispielsweise festgelegt, wer gerade sendet und wer empfängt.
- *Schicht 6* (**Darstellungsschicht**) wandelt auf Senderseite Daten in das Format um, in dem sie an Schicht 5 weitergegeben werden. Auf Empfängerseite werden in Schicht 6 die ankommenden Daten in ein vom Empfänger gewünschtes Format konvertiert.

- *Schicht 7* (**Anwendungsschicht**) stellt die Schnittstelle zu den Programmen her, die die Kommunikation durchführen (zum Beispiel einem Webbrowser oder E-Mail-Programm).

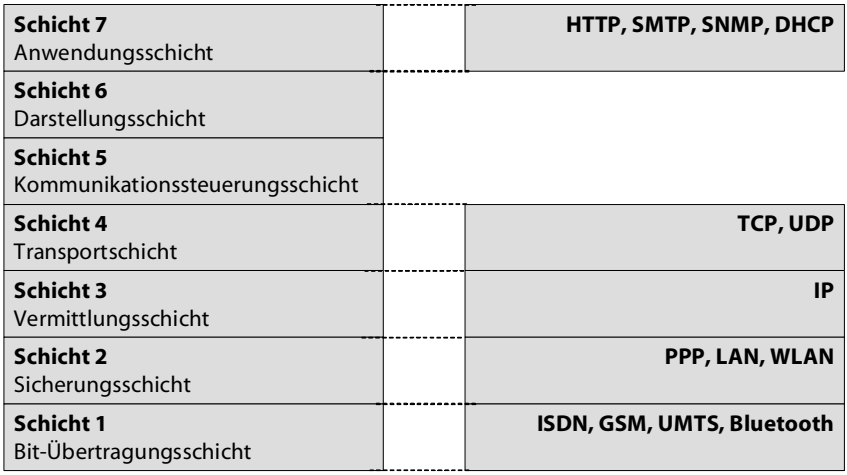


Abb. 31–1 *Die Protokolle der TCP/IP-Familie lassen sich in die Schichten 2, 3, 4 und 7 des OSI-Modells einordnen.*

Der Vorteil eines solchen Schichtenmodells für Protokolle liegt auf der Hand: Anstatt ein Riesenprotokoll für alle Eventualitäten zu konstruieren, entwickelt man sieben kleinere Protokolle, die aufeinander aufbauen. Wenn die Schnittstelle zwischen den Protokollen standardisiert ist, dann kann jedes einzelne Protokoll geändert werden, ohne dass dies die anderen betrifft. Außerdem wird den Entwicklern von Anwendungsprogrammen die Arbeit enorm erleichtert, denn sie müssen sich nur um die Protokolle der Schicht 7 kümmern.

31.1.2 Die wichtigsten Netzwerkprotokolle im OSI-Modell

Das OSI-Modell hat durchaus auch seine Schwächen. Schicht 2 muss in der Praxis meist in zwei Unterschichten aufgeteilt werden. Dagegen hat sich herausgestellt, dass man auf Schicht 5 und 6 gut verzichten kann, weshalb es kaum entsprechende Protokolle gibt. Zudem passen einige real existierende Protokolle nicht so recht in das OSI-Modell. Dennoch ist dieses nach wie vor ein beliebtes Hilfsmittel zur Beschreibung von Protokollen. Schauen wir uns einmal an, welche bekannten Netzwerkprotokolle sich den jeweiligen OSI-Schichten zuordnen lassen:

- In Schicht 1 lassen sich ISDN und Mobilfunkprotokolle wie GSM und UMTS einordnen.
- Zu Schicht 2 gehören beispielsweise zwei Protokolle der TCP/IP-Familie: das Serial Line Internet Protocol (SLIP) und das Point to Point Protocol (PPP).

Darüber hinaus werden auch die üblichen Protokolle für LANs und WLANs in diese Schicht eingeordnet.

- Zu Schicht 3 des OSI-Modells gehört das Internet Protocol IP. Auch dieses zählt zur TCP/IP-Familie. Alle anderen Schicht-3-Protokolle können wir vernachlässigen.
- In Schicht 4 befinden sich mit TCP und UDP zwei weitere TCP/IP-Protokolle. Diese verwenden IP und stellen damit eine Ende-zu-Ende-Verbindung her.
- Schicht 5 und 6 gibt es im Internet und den meisten anderen Netzen nicht.
- In Schicht 7 gehören zahlreiche Protokolle aus der TCP/IP-Familie: HTTP, FTP, Telnet, DNS, NTP und andere.

Wenn Alice und Bob über das Internet miteinander kommunizieren, sieht das im einfachsten Fall so aus: Alice und Bob arbeiten je auf einem Rechner, auf dem ein Anwendungsprotokoll (zum Beispiel HTTP), TCP, IP und darunter liegende Protokolle implementiert sind. Dazwischen stehen mehrere Router, deren Protokollturm nur bis IP reicht. Auf IP-Ebene sind alle transportierten Daten bekanntlich in handliche Pakete verpackt. Router haben die Aufgabe, diese Pakete zu befördern. Da sich ein Router nicht für den Inhalt eines Pakets interessiert, ist in einem Router kein TCP und kein Anwendungsprotokoll notwendig. Natürlich ist eine Kommunikation zwischen Alice und Bob in der Praxis meist etwas komplizierter als hier dargestellt, etwa wenn eine Firewall dazwischen steht. Als Modellvorstellung soll diese Ansicht jedoch erst einmal genügen.

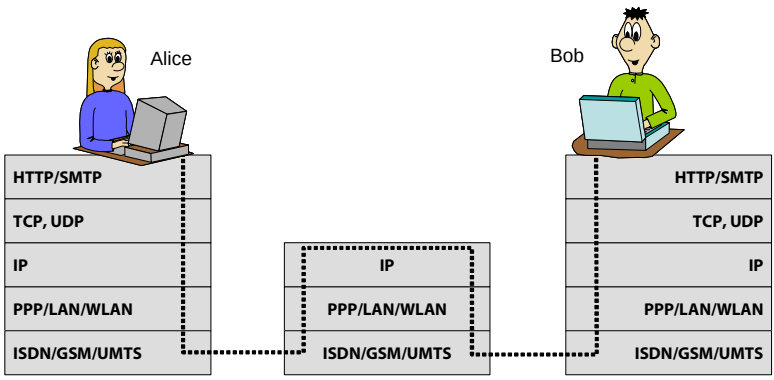


Abb. 31–2 *Eine Kommunikation im Internet läuft über einen oder mehrere Router. In einem Router ist IP die höchste Schicht.*

31.2 In welcher Schicht wird verschlüsselt?

Viele ältere Netzwerkprotokolle sind recht einfach konzipiert. Manche sind inzwischen sogar ein ärgerlicher Anachronismus. TCP, IP, UDP und zahlreiche weitere TCP/IP-Protokolle sind Beispiele dafür. Die Fähigkeit zur Echtzeitübertragung fehlt diesen Protokollen genauso wie Multicasting (Senden an mehrere Empfänger) oder die Möglichkeit zur Reservierung von Übertragungskapazitäten. Ein weiterer Mangel: Zahlreiche Protokolle sehen in ihrer ursprünglichen Version keinerlei Verschlüsselung oder sonstige Kryptografie vor. Praktisch die gesamte TCP/IP-Familie gehört dazu. Bei der immensen Abhörgefahr und sonstigen Sicherheitslücken im Internet ist dies durchaus Grund zur Sorge. Glücklicherweise waren die IETF und einige andere Organisationen nicht untätig und entwickelten Standards, die Kryptografie nachträglich in vorhandene Protokolle einbauten. Die Verfahren, die dabei eingesetzt werden, kennen Sie bereits aus den vorhergehenden Kapiteln. In diesem Teil des Buchs werden Sie erfahren, wie diese Verfahren innerhalb von kryptografischen Netzwerkprotokollen zum Einsatz kommen.

An dieser Stelle stellt sich eine der zentralen Fragen der praktischen Kryptografie: In welcher OSI-Schicht sollen Alice und Bob die in diesem Buch beschriebenen Krypto-Verfahren einsetzen? Wenn es um Verschlüsselung geht, lässt sich diese Frage nicht eindeutig beantworten, denn Verschlüsselung ist in jeder OSI-Schicht sinnvoll – jeweils mit unterschiedlichen Vor- und Nachteilen. Ähnlich sieht es mit der Authentifizierung aus. Anders liegt der Fall bei digitalen Signaturen: Das Signieren von Nutzdaten ist nur in Schicht 7 sinnvoll möglich. Mehr zum Thema Kryptografie im OSI-Modell gibt es in den folgenden Unterkapiteln. Die in der Praxis unnötigen Schichten 5 und 6 lasse ich bei dieser Betrachtung weg. Wir wollen uns außerdem an den OSI-Grundsatz halten, dass die Schnittstellen zwischen den Schichten nicht angetastet werden, sondern dass in einer Schicht nur Änderungen vorgenommen werden, von denen die darüber und darunter liegende Schicht nichts mitbekommt (Schnittstellenparadigma).

31.2.1 Kryptografie in Schicht 7 (Anwendungsschicht)

Kryptografische Verfahren in Schicht 7 einzusetzen bedeutet, dass ein Anwendungsprogramm – zum Beispiel ein Webbrowser oder ein E-Mail-Programm – das Verschlüsseln, Signieren und Authentifizieren übernimmt. Ein solches Programm muss diese Verfahren entweder selbst beherrschen oder sie müssen ihm durch Plug-ins, Java-Applets oder über eine geeignete Schnittstelle beigebracht werden.

Krypto-Verfahren in Schicht 7 anzuwenden, ist in vielerlei Hinsicht die beste Lösung. Wenn Alice mit Bob kommuniziert, dann können die beiden durch entsprechende Features ihrer Anwendungssoftware Kryptografie äußerst flexibel einsetzen. Alice kann für jede Nachricht festlegen, ob und wie diese verschlüsselt

wird, und sie kann digitale Signaturen sowie schlüsselabhängige Hashfunktionen benutzen. Bob kann empfangene Daten gleich oder später entschlüsseln, er kann sie verschlüsselt weiterleiten und kann ein digitales Dokument samt Signatur speichern. Wenn Alice irgendwann einmal bestreitet, die signierte Nachricht abgeschickt zu haben, dann kann Bob anhand der Signatur nachweisen, dass dies nicht stimmt.

Digitale Signaturen für Nutzdaten haben ohnehin nur in Schicht 7 wirklich einen Sinn (anders verhält es sich nur mit digitalen Signaturen, die in einem Challenge-Response-Verfahren eingesetzt werden). Dies liegt daran, dass in unteren Schichten stets nur Pakete sichtbar sind, in denen die eigentliche Nachricht zerstückelt untergebracht ist. Alice kann dort also nur Pakete signieren, nicht die ganze Nachricht. Da Pakete üblicherweise nicht aufbewahrt werden, ist zudem eine spätere Verifizierung nicht möglich. Auch Authentifizierungsmaßnahmen passen am besten in Schicht 7, denn darunter liegende Schichten können vom Anwender kein Passwort abfragen und keine Smartcard verlangen, ohne Schicht 7 zu Hilfe zu nehmen und damit das Schnittstellenparadigma zu verletzen.

Der wesentliche Nachteil des Krypto-Einsatzes auf Schicht 7 besteht darin, dass alle Anwendungsprogramme, die Alice und Bob zum Verschlüsseln nutzen wollen, Krypto-Verfahren unterstützen müssen. Von Nachteil ist außerdem, dass Informationen, die in tieferen Schichten relevant sind (zum Beispiel Absender- und Empfängeradressen), in Schicht 7 nicht verschlüsselt werden können. Dies erleichtert Verkehrsflussanalysen.

31.2.2 Kryptografie in Schicht 4 (Transportschicht)

Schicht 4 umfasst im Internet die beiden Protokolle TCP (Transmission Control Protocol) und UDP (User Datagram Protocol). Die meisten Protokolle der Schicht 7 arbeiten über TCP. Schicht 7 übergibt an TCP eine sogenannte Portnummer, anhand derer auch auf dieser Ebene noch festgestellt werden kann, zu welcher Anwendung die zu befördernden Daten gehören. Kryptografische Verfahren in Schicht 4 einzubauen, heißt, einen sicheren Tunnel zwischen zwei Anwendungsprogrammen (zum Beispiel Webserver und Webbrowser) zu schaffen. Anwendungsprogramme bekommen von diesem Tunnel nichts mit, was in der Praxis der wichtigste Vorteil eines Kryptografie-Einsatzes unter der Anwendungsebene ist. Statt zahlreicher Applikationen muss lediglich dasjenige Programm geändert werden, welches das entsprechende Protokoll realisiert.

Durch die Portnummer ist in Schicht 4 bekannt, welche Anwendung gerade aktiv ist. Somit kann die Art der Verschlüsselung oder Authentifizierung immer noch davon abhängig gemacht werden, welche Anwendung damit geschützt wird. Da die Portnummer selbst mitverschlüsselt werden kann, ist es Abhörer Mallory dennoch nicht möglich, Informationen über die Art der verschlüsselten Daten zu gewinnen, was wiederum eine Verkehrsflussanalyse erschwert.

Allerdings hat dieser Ansatz auch Nachteile. Für Alice ist es schwerer, auf die Verschlüsselung Einfluss zu nehmen: Wenn das Schnittstellenparadigma nicht verletzt werden soll, dann kann sie mit einer speziellen Software lediglich die Verschlüsselung für eine bestimmte Portnummer aus- und einschalten, nicht jedoch über ihr Anwendungsprogramm eine bestimmte Nachricht verschlüsseln. Digitale Signaturen können aus dem gleichen Grund nicht sinnvoll eingesetzt werden. Bob würde mit seinem Anwendungsprogramm die Daten ohnehin nicht signiert erhalten. Das Gleiche gilt für die Verschlüsselung: Bobs Anwendung erhält die Daten unverschlüsselt, da sie bereits in Schicht 4 entschlüsselt wurden. Er kann die Daten also nicht verschlüsselt weiterleiten oder abspeichern, obwohl sie vielleicht gar nicht für seine Augen bestimmt sind.

Der langen Rede kurzer Sinn: Kryptografie unterhalb der Anwendungsebene schafft zwar einen sicheren Tunnel; wenn eine Nachricht jedoch den Tunnel erst einmal verlassen hat, dann ist es mit der Sicherheit vorbei. Dies wirkt sich vor allem bei E-Mails kritisch aus, da E-Mail-Gateways stets als Anwendungsprogramme realisiert sein müssen. Dadurch können E-Mails nur in der Anwendungsschicht so verschlüsselt werden, dass nur der Empfänger sie lesen kann. Es gibt noch weitere Nachteile: Mit TCP und UDP müssen immerhin zwei Protokolle geändert werden, wenn Kryptografie eingesetzt wird. Zudem können in Schicht 4 die IP-Adressen nicht verschlüsselt werden, da diese in Schicht 3 zum Routen benötigt werden. Die wichtigste Information für eine Verkehrsflussanalyse liegt für Mallory also auch hier offen.

31.2.3 Schicht 3 (Vermittlungsschicht)

Auch in Schicht 3 kann ein sicherer Tunnel zwischen zwei Anwendern hergestellt werden. Die Vor- und Nachteile hierbei sind weitgehend die gleichen wie in Schicht 4. Jedoch muss hier mit IP nur ein Protokoll geändert werden, wenn kryptografische Verfahren im Internet eingesetzt werden. Portnummern sind in Schicht 3 allerdings nicht mehr bekannt, weshalb eine Verschlüsselung keinen Bezug mehr auf die Anwendung nehmen kann, ohne das Schnittstellenparadigma zu verletzen. Dafür sind die Portnummern auch für Mallory nicht lesbar, was ihm keine Rückschlüsse auf die Art der übertragenen Daten erlaubt. IP-Adressen können dagegen auch in Schicht 3 nicht verschlüsselt werden, da sie zum Routen gebraucht werden.

Ein gefährlicher Angriff auf Schicht 3 ist das IP-Spoofing (siehe Abschnitt 3.4.3). Hierbei fälscht Mallory die Absenderadresse von IP-Paketen und macht den Empfänger so glauben, das Paket käme von einem vertrauenswürdigen Rechner (zum Beispiel einem Rechner im gleichen Firmennetz). Mallory kann auch andere Daten im IP-Header fälschen und so beispielsweise einen Router dazu veranlassen, alle Pakete über einen bestimmten Router zu leiten, den Mallory in seiner Gewalt hat. Einigen Unfug kann Mallory auch anrichten, wenn er Nachrich-

ten des Internet Control Message Protocol (ICMP) fälscht. Dieses ist ebenfalls in Schicht 3 angesiedelt und verwendet IP-Pakete, um Steuerungsmeldungen zwischen Routern zu verschicken. Durch gefälschte Nachrichten kann Mallory also dem einen Router vormachen, der andere sei überlastet oder eine größere Menge an Paketen sei nicht angekommen. Denial-of-Service-Attacken sind hierbei auf verschiedene Weise denkbar. Die genannten Probleme zeigen, dass eine Authentifizierung und ein Integritätscheck bei IP-Paketen durchaus einen Sinn hat. Dies ist in höheren Schichten nicht möglich.

Da nicht nur in Anwendungen, sondern auch auf Routern eine IP-Schicht vorhanden ist, kann in Schicht 3 auch eine Teilstückverschlüsselung durchgeführt werden. Wenn also zwei Router durch ein besonders abhörgefährdetes Übertragungsmedium verbunden sind, ist eine Verschlüsselung auf IP-Ebene sinnvoll.

31.2.4 Schicht 2 (Sicherungsschicht)

Alice und Bob können in Schicht 2 verschlüsseln – jedoch nur von einem Schicht-2-Knoten zum nächsten. Dadurch entsteht meist keine sichere Ende-zu-Ende-Verbindung, sondern lediglich eine sichere Teilstrecke. Dies hat vor allem praktische Vorteile: Viele Datenleitungen (Telefonnetz, Weitverkehrsnetze, lokale Netze) haben unterschiedliche Betreiber. Auf Schicht 2 kann jeder davon nach eigenem Wunsch verschlüsseln, ohne die anderen fragen zu müssen. Dabei kann der Betreiber auch auf das Übertragungsmedium Rücksicht nehmen. Bei einer einfach abzuhörenden Funkstrecke ist es möglicherweise sinnvoll, die darüber fließenden Daten zu verschlüsseln. Bei einer Drahtverbindung ist eine solche Maßnahme dagegen oft unnötig. Von Vorteil bei einer Verschlüsselung in Schicht 2 ist zudem, dass die Anwender davon nichts mitbekommen. Zudem sind auf Schicht 2 IP-Adressen nicht relevant und werden deshalb mitverschlüsselt.

Der Nachteil von Schicht 2 gegenüber Schicht 3 liegt darin, dass kein sicherer Tunnel von Alice zu Bob entsteht. Greift Mallory einen Schicht-3-Knoten an (also einen Router), so erhält er die Nachricht unverschlüsselt.

31.2.5 Schicht 1 (Bit-Übertragungsschicht)

Daten auf Schicht 1 zu verschlüsseln, hat gegenüber Schicht 2 zunächst einmal keine Vorteile. In Schicht 2 werden meist Prüfsummen verwendet, um Fehler zu entdecken, und es hat wenig Sinn, auf diese Fehlerkorrektur vor dem Entschlüsseln zu verzichten. Eine Ausnahme besteht lediglich dann, wenn zwischen Schicht 1 und 2 eine wichtige Schnittstelle liegt. Dies ist beispielsweise bei ISDN der Fall, das zu Schicht 1 gezählt wird. Es ist durchaus sinnvoll, auf ISDN-Ebene zu verschlüsseln, um die daran angeschlossenen Geräte unbehelligt zu lassen. Gleiches gilt auch für GSM und UMTS.

31.2.6 Fazit

Bei der Betrachtung von Kryptografie im OSI-Modell fallen drei Dinge auf:

- Je tiefer die Schicht, in der verschlüsselt wird, desto besser lässt sich ein Minimum Disclosure realisieren. Adressen, Routing-Informationen und Ähnliches aus einer höheren Schicht werden mitverschlüsselt.
- Je höher die Schicht, in der verschlüsselt wird, desto mehr haben Alice und Bob Einfluss darauf, was und wie verschlüsselt wird. Dass bestimmte Daten anders verschlüsselt werden als andere, lässt sich in tieferen Schichten nicht mehr regeln.
- Je höher die Schicht, in der verschlüsselt wird, desto besser lassen sich Zwischenstationen überbrücken. In vielen Fällen ist es beispielsweise nicht wünschenswert, dass Daten unterwegs bei einem Router entschlüsselt und wieder verschlüsselt werden.

Nach dieser Betrachtung dürfte außerdem klar sein, dass es keine gute und keine schlechte Schicht für Kryptografie gibt. In welche Schicht man Verschlüsselung und ähnliche Mechanismen einbaut, ist vielmehr vom Einzelfall abhängig. Meistens ist es sogar sinnvoll, verschiedene Ansätze miteinander zu kombinieren.

31.3 Design eines kryptografischen Netzwerkprotokolls

Kryptografische Netzwerkprotokolle gibt es in großer Zahl und für unterschiedliche Anwendungsgebiete. Dennoch erfüllen viele von ihnen einem recht ähnlichen Zweck: Sie ermöglichen die verschlüsselte Kommunikation zwischen zwei Instanzen (in unserem Fall zwischen Alice und Bob). Man bezeichnet so etwas auch als **sicheren Kanal**. Protokolle, die einen sicheren Kanal realisieren, gibt es in allen OSI-Schichten – als Beispiele seien die in späteren Kapiteln beschriebenen Protokolle WPA, IPsec, SSL und Secure Shell genannt. Ein sicherer Kanal sieht in der Regel zwei Teile vor: die Initialisierungsroutine und die Datenaustauschroutine.

31.3.1 Initialisierungsroutine

Die **Initialisierungsroutine** eines sicheren Kanals hat den Zweck, Alice und Bob auf sichere Weise mit einem gemeinsamen geheimen Schlüssel zu versorgen. Um vor Mallory sicher zu sein, müssen sich Alice und Bob hierbei gegenseitig authentisieren (manchmal reicht es auch aus, wenn nur einer der beiden sich authentisiert). Recht einfach ist die Sache, wenn Alice und Bob bereits einen gemeinsamen geheimen Schlüssel besitzen (etwa wenn sie diesen zuvor persönlich übergeben haben). Dieser Fall ist vor allem auf unteren OSI-Schichten häufig gegeben, wenn eine Komponente nur wenige (oder gar nur einen) Kommunikationspartner hat.

Die Initialisierungsroutine besteht dann meist aus einem (evtl. wechselseitigen) Challenge-Response-Verfahren.

Falls kein gemeinsamer geheimer Schlüssel vorhanden ist, führen Alice und Bob im Rahmen der Initialisierungsroutine einen Schlüsselaustausch durch. Meist erfolgt dies asymmetrisch mit RSA oder Diffie-Hellman. Viele Protokolle stützen sich hierbei auf eine PKI. Aber auch ein symmetrischer Schlüsselaustausch mit Kerberos oder einem ähnlichen Verfahren ist möglich. Viele kryptografische Netzwerkprotokolle sind flexibel und unterstützen mehrere Schlüsselaustauschverfahren. Ist die Initialisierungsroutine eines kryptografischen Netzwerkprotokolls in sachgerechter Weise umgesetzt, dann sind Spoofing- und Man-in-the-Middle-Angriffe nahezu ausgeschlossen. Vor allem, wenn asymmetrische Verfahren im Spiel sind, sind jedoch manchmal Denial-of-Service-Angriffe möglich.

31.3.2 Datenaustauschroutine

In der **Datenaustauschroutine** eines sicheren Kanals wird vorausgesetzt, dass Alice und Bob – als Ergebnis der Initialisierungsroutine – einen gemeinsamen geheimen Schlüssel besitzen. Mit diesem Schlüssel verschlüsseln sie nun die Daten, die sie sich gegenseitig zuschicken. Viele moderne Krypto-Protokolle sehen hierbei vor, dass Alice und Bob den gemeinsamen Schlüssel nicht direkt, sondern nur als Masterschlüssel verwenden. Aus diesem generieren sie mit einer kryptografischen Hashfunktion einen Sitzungsschlüssel, der für die Verschlüsselung der Daten genutzt wird. In die Hashwert-Bildung gehen neben dem Masterschlüssel noch einige andere Informationen ein (z.B. ein Zufallswert, der nicht geheim gehalten werden muss) ein. Ab und zu sollten Alice und Bob den Sitzungsschlüssel wechseln, indem sie durch eine erneute Hashwert-Bildung (mit einem geänderten Zufallswert) einen neuen generieren. Auf diese Weise erschweren sie Known-Key-Attacken.

Eine interessante Frage ist, welches symmetrische Verschlüsselungsverfahren Alice und Bob innerhalb eines sicheren Kanals verwenden sollen. Die meisten (wenn auch nicht alle) Netzwerkprotokolle lassen verschiedene Verfahren zu, aus denen der Implementierer wählen kann oder die verhandelbar sind (viele kryptografische Netzwerkprotokolle sind verhandlungsfähig). Am weitesten verbreitet ist sicherlich der AES. Als Betriebsart sind CBC, CFB und CTR für einen sicheren Kanal geeignet. Die meisten Protokolle sind auch diesbezüglich flexibel und stellen mehrere Betriebsarten zur Wahl.

Die nächste interessante Frage lautet nun, ob die verschlüsselten Daten zusätzlich noch mit einer schlüsselabhängigen Hashfunktion vor Manipulationen durch Mallory geschützt werden sollen. Auf den ersten Blick erscheint dies nicht notwendig, denn wenn Mallory verschlüsselte Daten manipuliert, dann kommt beim Entschlüsseln nur Unsinn heraus – Alice und Bob bemerken dies im Normalfall sofort. Wenn der Klartext ein geeignetes Format hat, das beispielsweise

Prüfsummen und nummerierte Datenpakete vorsieht, dann ist es nahezu ausgeschlossen, dass durch Zufall ein sinnvoller Inhalt entsteht. Doch besonders vorsichtigen Kryptografen (und derer gibt es viele) reichen diese Überlegungen nicht. Sie weisen auf zwei Dinge hin

- Mallory hat in jeder Betriebsart die Möglichkeit, den Geheimtext in irgendeiner Form zu manipulieren. In manchen Fällen kann er das in einer Weise tun, bei der das Format des Klartexts unverletzt bleibt. Die Frage, ob und unter welchen Voraussetzungen Mallory dies zu einem wirksamen Angriff nutzen kann, erübrigt sich, wenn Alice und Bob von vornherein eine Integritätskontrolle mit einer schlüsselabhängigen Hashfunktion vorsehen.
- Die Sicherheit einer kryptografischen Implementierung sollte immer so wenig wie möglich von äußeren Faktoren abhängig sein. Die Erfahrung zeigt, dass so manche Implementierung in Umgebungen (wieder-)verwendet wird, an die der Entwickler gar nicht dachte und in denen nicht alle ursprünglich geplanten Voraussetzungen gegeben sind. In diesem Fall bedeutet das: Wer einen sicheren Kanal entwickelt, sollte sich besser nicht darauf verlassen, dass die transportierten Daten ein bestimmtes Format haben – auch wenn ein solches eigentlich vorgesehen ist.

Viele kryptografische Netzwerkprotokolle sehen aus diesen beiden Gründen neben der Verschlüsselung auch eine Integritätsprüfung mit einer schlüsselabhängigen Hashfunktion vor. Besonders elegant funktioniert dies, wenn Alice und Bob eine Stromchiffre verwenden, die sowohl verschlüsselt als auch einen Hashwert liefert (siehe Abschnitt 16.1.1). Nutzen sie kein derartiges Verfahren, dann müssen sie sich fragen, ob sie erst verschlüsseln und dann hashen oder umgekehrt. Beide Varianten sind möglich. Die Unterschiede sind in der Praxis so gering, dass ich an dieser Stelle nicht näher darauf eingehen will. Für eine ausführliche Diskussion der Frage, in welcher Reihenfolge verschlüsselt und gehasht werden soll, empfehle ich [FeScKo].

32 Krypto-Standards für OSI-Schicht 1

Nachdem Sie im vorhergehenden Kapitel erfahren haben, welche Vor- und Nachteile der Einsatz von Kryptografie in bestimmten OSI-Schichten bietet, schauen wir uns nun die praktischen Umsetzungen an. Wir beginnen unten im OSI-Modell (Schicht 1) und betrachten die Anstrengungen, die unternommen wurden, um die in diesem Bereich angesiedelten Protokolle mit kryptografischen Mechanismen auszurüsten.

32.1 Krypto-Erweiterungen für ISDN

ISDN habe ich Ihnen bereits in Abschnitt 3.3.3 vorgestellt. Einige Jahre lang war ISDN eine beliebte Technik, um sich über die Telefonleitung mit dem Internet zu verbinden, doch in diesem Segment wurde es inzwischen von DSL verdrängt. Für die Sprachtelefonie ist ISDN jedoch nach wie vor weit verbreitet. Krypto-Lösungen für ISDN gibt es bereits seit den Neunzigern, doch im Gegensatz zu fast allen

anderen Netzwerkprotokollen, um die es in diesem Buch geht, existiert kein Standard. Eine nennenswerte Nachfrage nach ISDN-Verschlüsselung gibt es nur im Militär- und Regierungsbereich – alle am Markt angebotenen Produkte sprechen diese Zielgruppe an.

Die bekannteste ISDN-Verschlüsselungslösung ist Elcrodat 6-2 von der Firma Rohde & Schwarz SIT [BSI]. Elcrodat 6-2 realisiert einen sicheren Kanal (siehe Abschnitt 31.3) zwischen den jeweiligen Kommunikationspartnern. Die genaue Funktionsweise ist nicht öffentlich bekannt. Zur Verschlüsselung in der Datenaustauschroutine wird ein proprietäres, symmetrisches Verfahren eingesetzt. Der Schlüsselaustausch in der Initialisierungsroutine basiert auf einem asymmetrischen Verfahren. Elcrodat 6-2 wird von allen deutschen Sicherheitsbehörden sowie innerhalb der NATO eingesetzt.

Abb. 32–1 *ISDN-Verschlüsselungshardware der Firma Rohde & Schwarz SIT*

32.2 Kryptografie im GSM-Standard

Wenn Sie in den letzten Jahren mit einem Handy telefoniert haben, dann arbeitete dieses mit großer Wahrscheinlichkeit nach dem Standard **GSM** (**Global System for Mobile Communication**). In Europa sowie in vielen anderen Ländern der Welt setzte sich GSM als Standard für die Mobiltelekommunikation durch und ersetzte die bis dahin üblichen analogen Technologien. Inzwischen hat GSM seinen Zenit zwar überschritten und wird derzeit vom Nachfolger UMTS verdrängt, doch bis zum Ende von GSM wird es noch einige Jahre dauern.

Mit GSM verhält es sich zunächst einmal ähnlich wie mit ISDN: Wird es als Übertragungsmedium für das Internet genutzt, dann muss man es als Schicht-1-Protokoll betrachten. Eine Verschlüsselung ist in diesem Fall nur selten notwendig, da es in höheren OSI-Schichten bessere Möglichkeiten dazu gibt. Ähnliche Überlegungen gelten, wenn GSM als Basis für die Datenübertragung mit WAP genutzt wird. Anders sieht es dagegen aus, wenn Alice und Bob GSM zum Telefonieren nutzen. In diesem Fall gibt es keine höheren Schichten, weshalb eine Verschlüsselung auf GSM-Ebene ansetzen muss. Dass es durchaus sinnvoll ist, GSM-

Verbindungen zu verschlüsseln, sollte nach der Lektüre von Abschnitt 3.3.5 klar sein.

Schon den Urhebern des GSM-Standards war die Wichtigkeit von Verschlüsselung bewusst. GSM gehörte daher zu den ersten Kommunikationsstandards, bei denen Kryptografie von Anfang an ein Bestandteil war. Jedes GSM-Handy verschlüsselt automatisch und führt eine Authentifizierung über ein Challenge-Response-Verfahren durch. Die Authentifizierung ist notwendig, damit Mallory nicht mit seinem Handy auf Alices Kosten telefonieren kann. Leider bildet die GSM-Verschlüsselung keinen sicheren Kanal zwischen den beiden Gesprächspartnern Alice und Bob, sondern jeweils nur zwischen dem Handy und der Basisstation. Dies hat zwar den Vorteil, dass Mallory nicht einfach mit einer Antenne das Gespräch der beiden abhören kann. Das Abhören auf dem restlichen Teil der Übertragungsstrecke ist jedoch nach wie vor möglich.

32.2.1 Wie GSM Kryptografie einsetzt

GSM-Teilnehmerin Alice erhält von ihrem Netzbetreiber eine Smartcard (**SIM-Karte**), die sie in ihr Handy steckt. Die SIM-Karte ist mit einer PIN geschützt. Auf der SIM-Karte ist ein geheimer Schlüssel der Länge

128 Bit gespeichert (dies ist ein Masterschlüssel), der auch dem Netzbetreiber bekannt ist. GSM sieht keine asymmetrische Kryptografie vor. Der Masterschlüssel wird zunächst zur Authentifizierung verwendet, wobei ein Verfahren namens *A3* zum Einsatz kommt. Anschließend wird aus dem Masterschlüssel ein 64-Bit-Sitzungsschlüssel generiert. Diese Generierung erfolgt mithilfe eines Verfahrens, das im GSM-Standard *A8* genannt wird. Mit dem Sitzungsschlüssel wird dann die Kommunikation zwischen Handy und Basisstation verschlüsselt, wozu das in Abschnitt 16.3 beschriebene Verschlüsselungsverfahren A5 eingesetzt wird.

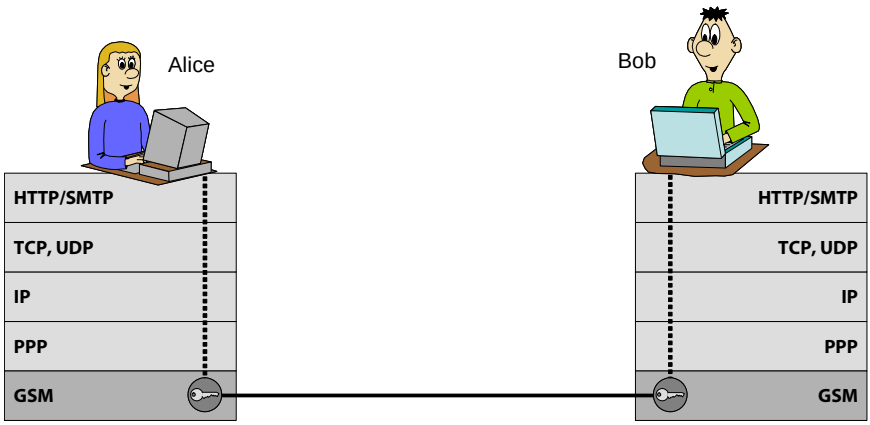


Abb. 32–2 *Auch GSM dient im Internet als Schicht-1-Protokoll. GSM war eines der ersten Protokolle, bei dem kryptografische Mechanismen fest eingebaut waren.*

Während A5 im GSM-Standard beschrieben wird, kann jeder Netzbetreiber A3 und A8 selbst festlegen. Lediglich die Größe von Challenge, Response, Masterschlüssel und Sitzungsschlüssel ist vorgegeben. Da A3 und A8 nur zwischen Alices SIM-Karte und der Basisstation zur Anwendung kommen, die beide vom Netzbetreiber kontrolliert werden, sind durch eine proprietäre Realisierung dieser Verfahren keine Inkompatibilitäten zu befürchten. Es liegt nahe, A3 und A8 als schlüsselabhängige Hashfunktionen zu realisieren und in beiden Fällen dasselbe Design zu verwenden. Ist dies der Fall, dann nutzt A3 den Masterschlüssel, um aus der Challenge einen Hashwert zu berechnen, der als Response verwendet wird. Ein weiterer Hash mit demselben Verfahren und demselben Schlüssel ergibt den Sitzungsschlüssel. Die meisten GSM-Netzbetreiber verwenden für A3 und A8 die schlüsselabhängige Hashfunktion COMP128. Diese habe ich bereits in Abschnitt 14.5.2 erwähnt. Zusammengefasst sieht ein GSM-Verbindungsaufbau wie folgt aus:

1. Die Basisstation sendet einen 128-Bit-Wert als Challenge an Alice (bzw. an die SIM-Karte in Alices Handy).
2. Alice berechnet aus der Challenge und dem Masterschlüssel mit dem Algorithmus A3 die Response (32 Bit) und sendet diese an die Basisstation.
3. Die Basisstation überprüft die Korrektheit der Response mithilfe des geheimen Schlüssels auf Alices Smartcard (dieser ist dem Netzbetreiber bekannt).
4. Alice verwendet den Masterschlüssel und die Challenge, um mit dem Algorithmus A3 den Sitzungsschlüssel zu berechnen.
5. Die Basisstation berechnet auf die gleiche Weise ebenfalls den Sitzungsschlüssel.
6. Alices Handy und die Basisstation kommunizieren verschlüsselt, wobei das Verschlüsselungsverfahren A5 und der zuvor generierte Sitzungsschlüssel verwendet werden. Die Verschlüsselung wird aus Performanzgründen nicht auf der SIM-Karte, sondern vom Handy selbst durchgeführt.

32.2.2 Sicherheit von GSM

Die kryptografische Sicherheit von GSM ist unbefriedigend. Schon das Protokolldesign entspricht nicht mehr dem Stand der Technik und sieht beispielsweise keine Authentisierung der Basisstation gegenüber Alice vor. Außerdem bietet das Protokoll keine sinnvolle Trennung zwischen Masterschlüssel und Sitzungsschlüssel, da der Masterschlüssel unverändert für die Authentifizierung verwendet wird. Forward Security oder Backward Security ist ebenfalls nicht gegeben.

Noch schlechter sieht es bei den verwendeten Krypto-Verfahren aus. Bereits in Abschnitt 16.3 haben Sie erfahren, dass das symmetrische Verschlüsselungsverfahren A5 heute geknackt werden kann. Auch COMP128 ist alles andere als sicher. 1998 veröffentlichten Ian Goldberg und David Wagner einen Angriff, mit dem man einer SIM-Karte den Masterschlüssel entlocken kann (eine Beschreibung findet sich in [Dingfe]). Dazu benötigten sie etwa 185.000 Challenge-Werte,

mit denen sie eine Karte fütterten und anschließend jeweils die Response betrachteten. Die für einen solchen Angriff benötigte Zeit liegt bei etwa acht Stunden. Gelingt es Mallory, auf diese Weise den Masterschlüssel von Alices SIM-Karte zu bestimmen, dann kann er diesen auf eine andere Smartcard übertragen (man nennt diesen Vorgang **Klonen**). Die geklonte SIM-Karte ist für die Basisstation vom Original nicht zu unterscheiden. Mallory kann damit also unerkannt auf Alices Kosten mobil telefonieren. Außerdem erleichtert der aus einer SIM-Karte extrahierte Schlüssel das Abhören ungemein.

Bruce Schneier kommentiert die Sicherheit der Verfahren A5 und COMP128 wie folgt [Schn96]: »What's most interesting about these algorithms is how robustly lousy they are.« Trotzdem glaube ich nicht, dass die GSM-Verschlüsselung absichtlich schwach konstruiert wurde, um ein einfaches Abhören zu ermöglichen. Vermutlich war es eher die Unerfahrenheit der namentlich nicht bekannten Entwickler dieser Verfahren. Die Geheimniskrämerei, die um die Funktionsweise von A5 und COMP128 getrieben wurde, trug auch nicht gerade zur Vertrauensbildung bei.

32.3 Kryptografie im UMTS-Standard

Nachfolger des Mobilfunkstandards GSM ist das leistungsfähigere **UMTS** (**Universal Mobile Telecommunications System**). UMTS ermöglicht nicht nur um ein Vielfaches höhere Datenraten, sondern bietet auch deutlich mehr Sicherheit. Insbesondere haben die Schöpfer des UMTS-Standards – diese sind unter dem Dach einer Organisation namens **3GPP** (3rd Generation Partnership Project) aktiv – aus den Fehlern von GSM gelernt und die verwendeten kryptografischen Verfahren und Protokolle auf ein erkennbar höheres Niveau gehoben.

32.3.1 Von UMTS verwendete Krypto-Verfahren

UMTS verwendet sieben kryptografische Verfahren, die mit **f1**, **f2**, **f3**, **f4**, **f5**, **f8** und **f9** bezeichnet werden [MaPüSc, BoHoHN]. Hinter diesen nicht gerade aussagekräftigen Namen verbergen sich in sechs Fällen Methoden, die als schlüsselabhängige Hashfunktionen realisiert werden können. Das siebte Verfahren dient der Verschlüsselung. Alle sieben Verfahren sind symmetrisch. UMTS sieht keine asymmetrische Kryptografie vor. Die Verfahren f1 bis f5 sind im UMTS-Standard nicht fest vorgegeben. Stattdessen kann sie der Netzbetreiber im Rahmen verschiedener Vorgaben selbst aussuchen. Der Standard enthält jedoch eine unverbindliche Empfehlung für diese fünf Verfahren, die als *MILENAGE* bezeichnet wird. Hier ein Überblick:

- *f1, f2*: Diese beiden werden im Standard als »Authentifikationsfunktionen« bezeichnet. Dabei handelt es sich jeweils um eine schlüsselabhängige Hashfunktion mit 128 Bit Block- und Schlüssellänge. MILENAGE sieht für f1 und f2 jeweils eine erweiterte CBC-MAC-Version des AES vor. Die Erweiterung besteht aus einer zusätzlichen Datenrotation und der Addition zweier Konstanten per Exklusiv-oder-Verknüpfung. Interessanterweise entschied sich das UMTS-Standardisierungsgremium bereits vor Ende des AES-Wettbewerbs dafür, den damaligen AES-Kandidaten Rijndael zu verwenden. Erst später wurde dieses Verfahren zum Sieger des Wettbewerbs gekürt und in AES umgetauft.
- *f3, f4, f5*: Diese werden im UMTS-Standard als »Schlüsselgenerierungsfunktionen« bezeichnet. Sie haben jeweils die Aufgabe, mit einem geheimen Schlüssel aus einer Zufallszahl eine 128-Bit-Zahl zu generieren, die als Schlüssel verwendet werden kann. In der Praxis werden diese Anforderungen am besten von einer schlüsselabhängigen Hashfunktion erfüllt. Wir können uns daher auch f3, f4 und f5 als schlüsselabhängige Hashfunktion vorstellen. Der in MILENAGE beschriebene Aufbau von f3, f4 und f5 entspricht dem von f1 und f2, wobei jedoch andere Rotationen und Konstanten zum Einsatz kommen.

Man kann die Sache auf einen einfachen Nenner bringen: f1 bis f5 sind schlüsselabhängige Hashfunktionen auf Basis des AES, die identisch aufgebaut sind und sich lediglich durch Konstanten und die Art der zusätzlichen Rotation voneinander unterscheiden. Im Gegensatz zu den Verfahren f1 bis f5 sind die beiden Algorithmen f8 und f9 im UMTS-Standard fest vorgegeben:

- *f8*: Dies ist ein Verschlüsselungsverfahren. Es handelt sich dabei um das Verfahren KASUMI, das in einer Mischung aus OFB- und CTR-Modus betrieben wird.
- *f9*: Dies ist eine weitere schlüsselabhängige Hashfunktion. Sie ist als CBC-MAC-Version von KASUMI realisiert und generiert einen 32-Bit-Hashwert.

Wie in Abschnitt 10.1.2 beschrieben, ist KASUMI eine Variante von MISTY. MISTY wurde von den UMTS-Standardisierern ausgesucht, weil es sich um ein in Hardware performantes Verfahren handelt, das bereits damals gut untersucht war. Dem Gremium stand nur ein halbes Jahr zur Auswahl zur Verfügung, was für die Neuentwicklung eines Verfahrens zu kurz war. Im Gegensatz zu GSM entschied man sich bei der Standardisierung von UMTS, die verwendeten Krypto-Verfahren von Anfang an offenzulegen.

32.3.2 UMTS-Krypto-Protokolle

Die soeben beschriebenen Verfahren haben die Aufgabe, innerhalb von UMTS-Netzen für Authentizität, Integrität und Vertraulichkeit zu sorgen. Zu diesem Zweck sieht der UMTS-Standard zwei kryptografische Netzwerkprotokolle vor, die zusammen einen sicheren Kanal realisieren. Das erste Protokoll heißt **AKA** (**Authentication and Key Agreement**). Es handelt sich dabei um die Initialisierungsroutine des sicheren Kanals. Das zweite Protokoll – die Datenaustauschroutine des sicheren Kanals – ist nicht eigenständig, sondern bildet einen Bestandteil der UMTS-Datenübertragung und hat daher keinen speziellen Namen. Während sich bei GSM nur Anwenderin Alice (bzw. ihre SIM-Karte) gegenüber dem Netz authentisiert, gibt es bei UMTS auch eine Authentisierung in der Gegenrichtung. Dadurch kann sich Angreifer Mallory gegenüber Alice nicht als Basisstation ausgeben. An den beiden UMTS-Krypto-Protokollen sind drei Parteien beteiligt:

1. *Alice*: Anwenderin Alice verwendet ein Handy und die darin enthaltene Smartcard. Diese Smartcard wird in diesem Zusammenhang als **USIM-Karte** bezeichnet.
2. *UMTS-Netz*: Dies ist das UMTS-Netz, in dem Alice sich gerade aufhält. In der Regel ist dies das UMTS-Netz des Mobilfunkanbieters, bei dem Alice Kundin ist. Wenn sich Alice im Ausland aufhält, kann es sich jedoch auch um das Netz eines anderen Anbieters handeln. Das UMTS-Netz ist für Alices Handy über die heutzutage fast überall verfügbaren Basisstationen zugänglich.
3. *Authentifizierungsserver*: Dieser wird von Alices UMTS-Anbieter betrieben. Der Authentifizierungsserver hat alle Kundendaten gespeichert und ermöglicht eine verteilte Authentifizierung inklusive Credential-Synchronisation.

Statt asymmetrischer Kryptografie kommen bei UMTS Preshared Keys zum Einsatz. Auf Alices USIM-Karte ist ein 128-Bit-Schlüssel K gespeichert, den auch der Authentifizierungsserver kennt. Es handelt sich dabei um einen Masterschlüssel. Der Schlüssel K ist für jede USIM-Karte verschieden und darf die Karte nie verlassen. Er ist durch geeignete Konstruktionsmaßnahmen vor einem Auslesen geschützt.

AKA-Protokoll

Für das AKA-Protokoll gilt die Voraussetzung, dass Alice (gemeint ist im Folgenden immer Alices USIM-Karte) und der Authentifizierungsserver den gemeinsamen Schlüssel K besitzen, der als Masterschlüssel dient. Ziel ist es, dass Alice und das UMTS-Netz sich gegenseitig authentifizieren und dass dabei zwei Sitzungsschlüssel aus K abgeleitet werden, mit denen anschließend eine verschlüsselte Übertragung von Nutzdaten sowie eine integritätsgeschützte Übertragung von Steuerdaten durchgeführt werden. Natürlich könnten Alice und der Authentifi-

zierungsserver auch den Schlüssel *K* zur Verschlüsselung und zum Integritätsschutz nutzen, doch dann wäre das Protokoll anfälliger gegenüber einer Known-Key-Attacke.

Das AKA-Protokoll gliedert sich in zwei Teile. Im ersten Teil versorgt der Authentifizierungsserver das UMTS-Netz mit den notwendigen Informationen, damit dieses mit Alice eine gegenseitige Authentifizierung inklusive Schlüsselaustausch vornehmen kann. Dieser erste Protokollteil hat folgenden Ablauf:

1. Das UMTS-Netz fordert vom Authentifizierungsserver die zur Authentifizierung notwendigen Informationen (einen sogenannten *Authentifizierungsvektor*) an.
2. Der Authentifizierungsserver stellt einen Authentifizierungsvektor zusammen und sendet ihn an das UMTS-Netz.

Ein Authentifizierungsvektor besteht aus mehreren gleich strukturierten Einträgen, wobei ein Vektoreintrag pro Authentifizierungsvorgang vorgesehen ist. Um einen Vektoreintrag zu erstellen, benötigt der Authentifizierungsserver eine Nachrichtennummer (*SQN*), eine Zufallszahl (*RAND*) und eine betreiberspezifische Information, die *AMF* genannt wird. Ein Vektoreintrag hat folgende Bestandteile:

- *MAC*: Ein mit f1 und dem Schlüssel *K* erstellter Hashwert von *SQN*, *RAND* und *AMF*. Es gilt also $MAC = f1_K(SQN, RAND, AMF)$.
- *XRES*: Ein mit f2 und *K* erstellter Hashwert von *RAND*. Es gilt $XRES = f2_K(RAND)$. *RAND* entspricht hierbei einer Challenge, *XRES* entspricht der erwarteten Response.
- *CK*: Ein mit f3 und *K* erstellter Hashwert von *RAND*. Es gilt $CK = f3_K(RAND)$. *CK* ist einer der beiden Schlüssel, den Alice und das UMTS-Netz am Ende gemeinsam haben sollen. *CK* dient der Verschlüsselung.
- *IK*: Ein mit f4 und *K* erstellter Hashwert von *RAND*. Es gilt $IK = f4_K(RAND)$. *IK* ist der zweite Schlüssel, den Alice und das UMTS-Netz am Ende gemeinsam haben sollen.
- *AK*: Ein mit f5 und *K* erstellter Hashwert von *RAND*. Es gilt $AK = f5_K(RAND)$. *AK* ist ein dritter Schlüssel, den Alice und das UMTS-Netz gemeinsam haben werden. Er spielt jedoch nur für das AKA-Protokoll selbst, nicht für den späteren Datenaustausch eine Rolle.

Der zweite Teil des AKA-Protokolls kann starten, nachdem das UMTS-Netz einen Authentisierungsvektor vom Authentifizierungsserver empfangen hat. Da ein Authentisierungsvektor mehrere Einträge hat, reicht eine Abarbeitung des ersten AKA-Teils für mehrere Durchführungen des zweiten Teils aus. Teil 2 hat den Zweck, die gegenseitige Authentifizierung zwischen Alice und dem UMTS-Netz zu realisieren. Am Ende sollen Alice und das UMTS-Netz zudem die zwei gemeinsamen Schlüssel *CK* und *IK* besitzen. Der zweite AKA-Teil läuft wie folgt ab:

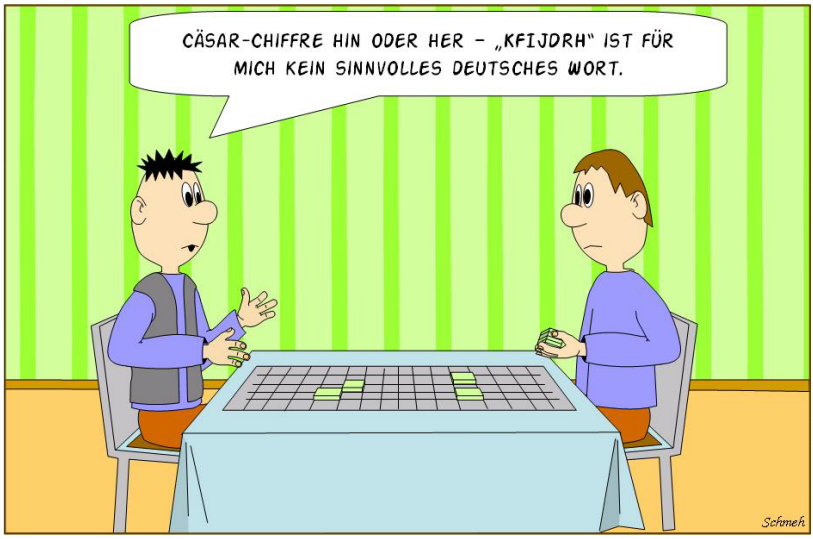
1. Das UMTS-Netz sendet *RAND* und ein sogenanntes Authentifizierungs-Token an Alice. Das Authentifizierungs-Token enthält *AMF*, *MAC* und $SQN \oplus AK$ (die Exklusiv-oder-Verknüpfung von *SQN* und *AK* kommt einer Verschlüsselung von *SQN* mit dem Schlüssel *AK* gleich).
2. Alice berechnet zunächst $AK = f5_K(RAND)$. Mit *AK* entschlüsselt sie anschließend *SQN*. Dann wendet sie f1 mit dem Schlüssel *K* auf *SQN*, *RAND* und *AMF* an. Das Ergebnis vergleicht sie mit *MAC*. Ist dieser Vergleich positiv, dann berechnet Alice $RES = f2_K(RAND)$ als Response und schickt diese an das UMTS-Netz zurück. Zudem berechnet Alice $CK = f3_K(RAND)$ und $IK = f4_K(RAND)$.
3. Das UMTS-Netz vergleicht anschließend den von Alice erhaltenen Wert *RES* mit *XRES*. Sind die beiden Werte gleich, dann ist das Protokoll positiv abgeschlossen.

Nach Abarbeitung dieses Protokolls mit wechselseitigem Challenge-Response weiß Handynutzerin Alice, dass sie tatsächlich mit einem vertrauenswürdigen UMTS-Netz verbunden ist, während der Netzbetreiber sicher sein kann, dass er es tatsächlich mit Alice zu tun hat. Zudem haben Alice und das UMTS-Netz die beiden Schlüssel *CK* und *IK* gemeinsam. Diese benötigen sie für die weitere Kommunikation. *AK* brauchen sie nicht mehr.

Kommunikationsteil

Nach erfolgter Authentifizierung und nach erfolgtem Schlüsselaustausch kann der verschlüsselte UMTS-Datenaustausch beginnen. Aus kryptografischer Sicht ist dieser recht einfach. Alice und das Netz nutzen den gemeinsamen Schlüssel *CK* zur Verschlüsselung und den Schlüssel *IK* für den Integritätsschutz der unverschlüsselten Steuerungsdaten. Dazu werden die beiden Verfahren f8 und f9 verwendet. Die Verschlüsselung findet nur zwischen Handy und Netz statt. Eine Ende-zu-Ende-Verschlüsselung gibt es nicht.

33 Krypto-Standards für OSI-Schicht 2



Nachdem wir es in OSI-Schicht 1 mit Technologien aus dem Telefonbereich (ISDN, GSM und UMTS) zu tun hatten, schauen wir uns nun Schicht 2 an. Hier finden wir beispielsweise WLAN und Bluetooth, die eine drahtlose Kommunikation über kurze Strecken ermöglichen. Da bei einer drahtlosen Datenübertragung die Abhörgefahr offensichtlich ist, haben sie alle Kryptografie fest eingebaut – wenn auch in recht unterschiedlicher Form. Nicht speziell für die drahtlose Kommunikation ist das Protokoll PPP gedacht, das ebenfalls in Schicht 2 eingeordnet wird. Trotzdem ist es nicht sinnvoll, PPP-Nachrichten kryptografisch abzusichern. Wie eine PPP-, WLAN- oder PPP-Verschlüsselung aussieht, erfahren Sie in diesem Kapitel.

33.1 Krypto-Erweiterungen für PPP

Von den zahlreichen Protokollen der TCP/IP-Familie werden nur zwei der Schicht 2 des OSI-Modells zugerechnet. Neben dem veralteten Serial Line Internet Protocol (SLIP) ist dies vor allem das **Point to Point Protocol** (**PPP**). PPP ist unterhalb von IP angesiedelt und kommt zwischen zwei festen Punkten (zum Beispiel Routern) zum Einsatz, wodurch es die typischen Eigenschaften eines Schicht-2-Protokolls hat. PPP wird vor allem in zwei Situationen eingesetzt: Zum einen kann Bob sich damit mit dem Einwahl-Rechner seines Internet-Providers verbinden. Zum anderen können damit zwei lokale Netze miteinander verbunden werden. Über die PPP-Verbindung fließen in beiden Fällen die IP-Pakete, in denen die Nachricht transportiert wird.

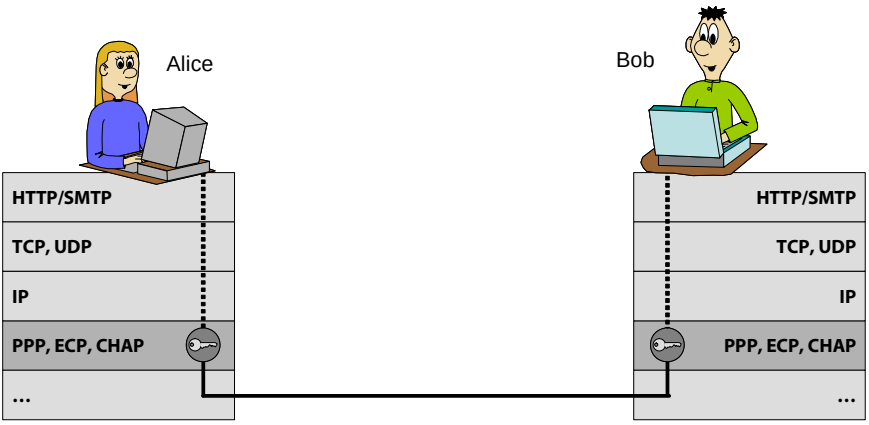


Abb. 33–1 *ECP (Verschlüsselung) und CHAP (Authentifizierung) sichern Schicht 2 des OSI-Modells kryptografisch ab.*

PPP wird in [RFC1661] beschrieben und in einigen späteren RFCs erweitert. Es handelt sich dabei um ein verhandlungsfähiges, zustandsbehaftetes Protokoll. Zu den Erweiterungen, die für PPP entwickelt wurden, gehören insbesondere verschiedene kryptografische Ergänzungen. Diese ermöglichen den Betrieb eines sicheren Kanals über PPP. Sie werden über das sogenannte *Link Control Protocol* (LCP) in PPP eingefügt. LCP ist derjenige Teil von PPP, der am Anfang einer Kommunikation abgearbeitet wird, um den beiden Kommunikationspartnern das Aushandeln von Protokollparametern zu ermöglichen.

33.1.1 CHAP und MS-CHAP

Das **Challenge Handshake Protocol** (**CHAP**) ist ein Authentifizierungsmechanismus für PPP. CHAP wird in [RFC1994] beschrieben. Eine von Microsoft entwickelte Variante namens MS-CHAP findet sich in [RFC2433] und [RFC2759].

CHAP realisiert ein einfaches Challenge-Response-Verfahren. Dieses wird beim Aufbau einer PPP-Verbindung abgearbeitet und kann zu einem späteren Zeitpunkt beliebig oft wiederholt werden. Es entspricht der Initialisierungsroutine des sicheren Kanals.

Wollen sich Alice und Bob mit CHAP gegenseitig authentisieren, so müssen sie vorher einen gemeinsamen Schlüssel vereinbart haben (Preshared Key). Zur Authentifizierung schickt Alice Bob einen Zufallswert (Challenge) zu, den Bob zusammen mit dem Schlüssel einer kryptografischen Hashfunktion zuführt. Dabei kann es sich um MD5 handeln, CHAP lässt aber auch andere Verfahren zu. Das Ergebnis (Response) sendet er zurück an Alice, die es überprüfen kann, da sie den Schlüssel ebenfalls kennt.

CHAP ist ein sehr einfaches Verfahren, das ohne Public-Key-Kryptografie auskommt. Es arbeitet nicht mit einem AAA-Server zusammen und bietet keine Forward Security oder Backward Security. Da Alice und Bob einen Schlüssel manuell vereinbaren müssen, funktioniert CHAP nur dann, wenn die Zahl potenzieller Kommunikationspartner begrenzt ist. Genau dies ist jedoch bei PPP in der Regel der Fall, da es stets zwischen zwei konstanten Netzknoten abgearbeitet wird.

33.1.2 EAP

CHAP ermöglicht nur eine sehr einfache Authentifizierung. Deutlich mehr bietet das **Extended Authentication Protocol** (**EAP**), das ebenfalls eine Authentifizierung über PPP ermöglicht und damit ebenfalls als Initialisierungsroutine in einem sicheren Kanal dient. EAP, das in [RFC3748] beschrieben wird, ist genau genommen kein Protokoll, sondern ein Rahmenwerk, in das ein nahezu beliebiges Authentifizierungsprotokoll integriert werden kann. Ein mithilfe von EAP realisierter Authentifizierungsablauf wird EAP-Methode genannt. Bei Redaktionsschluss dieses Buchs gab es bereits über 40 EAP-Methoden. Einige davon wurden in RFCs standardisiert, andere wurden von Herstellern speziell für deren Produkte entwickelt. Viele EAP-Methoden sehen eine RADIUS- oder DIAMETER-Abfrage vor und ermöglichen dadurch eine Credential-Synchronisation. Viele gängige Authentifizierungsverfahren gibt es inzwischen als EAP-Methode:

- **EAP-OTP** ist eine EAP-Methode, die eine Authentifizierung über Einmal-Passwörter bietet [RFC3748].
- **EAP-MD5** ermöglicht ein Challenge-Response-Verfahren auf Basis der kryptografischen Hashfunktion MD5 [RFC3748].
- **EAP-TLS** ist eine der wichtigsten EAP-Methoden [RFC2716]. Sie sieht vor, dass Alice und Bob auf PPP-Ebene ein Protokoll verwenden, das SSL bzw. TLS ähnelt (siehe Kapitel 35). Die Authentifizierung erfolgt mithilfe eines AAA-Servers (z.B. RADIUS oder DIAMETER).

- **EAP-TTLS** ist eine Erweiterung von EAP-TLS [BlaFun]. Es unterstützt nicht nur digitale Zertifikate, sondern auch andere EAP-Mechanismen wie MD5-Challenge-Response oder Einmal-Passwörter. Die Authentifizierung erfolgt mithilfe eines AAA-Servers.

EAP wurde zwar ursprünglich für PPP konzipiert, doch inzwischen kommt es auch in anderen Bereichen zum Einsatz. Viele der neueren EAP-Methoden sind sogar für PPP ungeeignet. Eine große Bedeutung hat EAP vor allem im Bereich Wireless LAN erlangt. Dort wird eine EAP-Variante eingesetzt, die im Standard IEEE 802.1x festgelegt ist und kurz 802.1x genannt wird. Einige der existierenden EAP-Methoden wurden speziell für 802.1x entwickelt. Von diesem Thema wird in diesem Kapitel noch die Rede sein.

33.1.3 ECP und MPPE

Das **Encryption Control Protocol** (**ECP**) sorgt für eine Verschlüsselung von Daten, die per PPP übertragen werden. Es ist damit die Datenaustauschroutine des sicheren Kanals. ECP ist in [RFC1968] spezifiziert, [RFC2419] und [RFC2420] beschreiben Erweiterungen davon. ECP setzt voraus, dass Alice und Bob einen gemeinsamen geheimen Schlüssel besitzen – dieser Schlüssel kann beispielsweise zuvor mit EAP ausgetauscht worden sein. ECP bietet Alice und Bob die Möglichkeit, ein Verschlüsselungsverfahren auszuhandeln. In RFC 2419 sind Details für die Verwendung des DES im Zusammenhang mit ECP beschrieben, in RFC 2419 geht es um die Nutzung von Triple-DES. Die Firma Microsoft hat eine eigene Variante von ECP entwickelt, die als **MPPE** (**Microsoft Point-To-Point Encryption**) bezeichnet wird. MPPE sieht die Verwendung von RC4 mit einer Schlüssellänge von 40 oder 128 vor. Die Details werden in [RFC3078] beschrieben.

33.1.4 Virtuelle Private Netze in Schicht 2

Die Firma Krypt & Co. betreibt an zwei verschiedenen Standorten jeweils ein lokales Netz. Diese beiden Netze möchte das Unternehmen miteinander verbinden, ohne sie gegenüber dem Internet zu öffnen. Dazu bietet sich folgende Lösung an: Ein Router im lokalen Netz wird mit einem Router im anderen lokalen Netz auf Schicht 2 über PPP verbunden. Eine derartige Verbindung zweier Netze wird als **LAN-Kopplung** bezeichnet. Eine LAN-Kopplung kann über das Internet realisiert werden. Dies bedeutet, dass zwischen den beiden Routern unterhalb von PPP ein für diesen Zweck geeignetes Schicht-7-Protokoll verwendet wird. Man nennt einen solchen Vorgang **Tunneln** (obwohl »Überbrücken« sicherlich eine etwas glücklichere Bezeichnung wäre). Das besagte Schicht-7-Protokoll wird **Tunnelprotokoll** genannt. Wie die unterschiedlichen TCP/IP-Protokolle bei einer LAN-Kopplung zusammenspielen, ist in Abbildung 33–2 ersichtlich. Das Tunnelprotokoll verwendet (wie andere Schicht-7-Protokolle) TCP oder UDP, darunter

folgt IP. Eine Besonderheit der LAN-Kopplung über das Internet liegt somit darin, dass einige OSI-Schichten doppelt vorkommen.

Betrachten wir nun einen weiteren Fall aus der Praxis: Krypt & Co. will die PCs einiger Außendienstmitarbeiter an das lokale Netz des Unternehmens anbinden (**Client-Anbindung**). Auch hier ist wiederum ein Tunneln durch das Internet mithilfe eines Tunnelprotokolls eine gängige Möglichkeit. Die Client-Anbindung ist so gesehen eine ähnliche Sache wie die LAN-Kopplung.

Das Tunneln des Internets zur LAN-Kopplung oder zur Client-Anbindung wird als **Virtuelles Privates Netz** (**VPN**) bezeichnet. Das Entscheidende an einem VPN lässt sich wie folgt zusammenfassen: Das lokale Netz nutzt zwar das Internet, um ein anderes lokales Netz oder einzelne Rechner anzubinden. Dennoch bleiben die Protokollwelten zwischen Internet und lokalem Netz strikt getrennt. Das Tunnelprotokoll muss nicht unbedingt kryptografisch abgesichert sein, auch wenn der Ausdruck »privat« dies nahelegt. In der Regel sieht ein VPN jedoch Verschlüsselung vor (in Form eines sicheren Kanals), und in diesem Buch soll uns nur dieser Fall interessieren. VPNs lassen sich in den Schichten 2, 3 und 4 realisieren. In diesem Kapitel ist nur Schicht 2 relevant. Hierfür gibt es derzeit zwei Tunnelprotokolle, die eine Rolle spielen:

- Das **Point to Point Tunneling Protocol** (**PPTP**) ist eine Entwicklung von Microsoft [RFC2637]. Es sieht zwei verschiedene Teilprotokolle vor: das Tunnelprotokoll für den eigentlichen Zweck von PPTP und das Steuerungsprotokoll für den Aufbau und die Verwaltung des Tunnels. Das Steuerungsprotokoll verwendet TCP in Schicht 4, während das Tunnelprotokoll auf ein Protokoll namens GRE (Generic Routing Encapsulation) aufbaut. Letzteres können wir uns an dieser Stelle als UDP-ähnliches Protokoll vorstellen. PPTP sieht keine Kryptografie vor, sondern überlässt dies anderen Protokollschichten. Die von Microsoft bevorzugte Variante sieht vor, dass die Krypto-Erweiterungen von PPP verwendet werden, um PPTP abzusichern. Microsoft hat für diesen Zweck eigene Versionen von CHAP und ECP entwickelt. Diese als MS-CHAP und MPPE bezeichneten Protokolle kennen Sie bereits aus Abschnitt 33.1.1.
- **L2TP** (**Layer Two Tunneling Protocol**) wurde innerhalb der IETF entwickelt. L2TP, das in [RFC2661] beschrieben wird, sieht wie PPTP zwei Teilprotokolle für Tunnel und Steuerung vor. Die Steuerung erfolgt über TCP, für den Tunnel wird UDP verwendet. Für den Einsatz von Kryptografie gelten bei L2TP die gleichen Überlegungen wie für PPTP. Eine Besonderheit von L2TP ist jedoch, dass es selbst eine Authentifizierung ermöglicht: Es unterstützt eine CHAP-ähnliche Passwortabfrage.

Für die LAN-Kopplung werden von vielen Herstellern Gateway-Produkte angeboten, mit denen die beiden Endpunkte eines Tunnels realisiert werden können. Solche Gateway-Funktionen sind oft auch in Router, Firewalls oder Betriebssys-

teme integriert. Wenn größere Datenmengen über den Tunnel gesendet werden, dann lohnt es sich, als Gateway eine spezielle Hardwarebox zu verwenden. Für die Client-Anbindung sind ebenfalls viele Produkte erhältlich. Meist bestehen sie aus einer Client- und einer Gateway-Komponente. Die Client-Komponente ist in der Regel eine Software, die auf dem anzubindenden Einzelrechner installiert wird. Die Gateway-Komponente kann wiederum als Hardwarebox oder als Software realisiert werden. Einige Lösungen unterstützen inzwischen auch PKI-Funktionen, wobei jeder Benutzer eines angebundenen Rechners ein eigenes Zertifikat erhält. Bei einer VPN-Lösung für die Anbindung von Einzelrechnern spielt die Administration der Client-Komponenten eine wichtige Rolle. Vor allem bei VPNs mit vielen Client-Komponenten sollte eine VPN-Lösung daher eine leistungsfähige Administrationskomponente beinhalten.

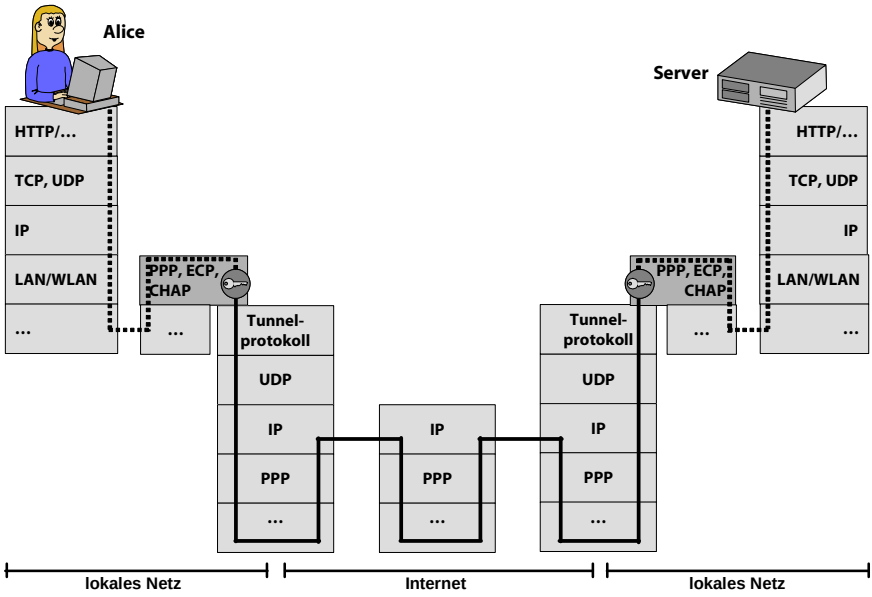


Abb. 33–2 *Beispiel für ein VPN: PPP-Pakete werden hierbei durch ein Tunnelprotokoll über das Internet übertragen. Das Tunnelprotokoll gehört in Schicht 7 des OSI-Modells.*

33.2 Kryptografie im WLAN

Kaum eine Netzwerktechnik hat sich in den letzten Jahren so schnell durchgesetzt wie das **Wireless Local Area Network** (auch Wireless LAN oder **WLAN** genannt). Wie der Name sagt, ermöglicht ein WLAN die kabellose Übertragung von Daten über kürzere Strecken und bietet dabei eine vergleichsweise hohe Bandbreite. Ein WLAN ist daher die Methode der Wahl, um PCs im Büro ohne Kabelsalat zu vernetzen. Außerdem bieten viele Hotels, Restaurants, Flughäfen oder öffentliche

Gebäude WLAN-Stationen (*Access Points*) an, damit sich Leute wie Laptop-Nutzerin Alice ohne großen Aufwand (allerdings oft gegen Bezahlung) unterwegs ins Internet einwählen können.

33.2.1 WEP

Angesichts der offensichtlichen Sicherheitsprobleme bei der drahtlosen Datenübertragung spielte die Kryptografie in der WLAN-Entwicklung schon früh eine Rolle. 1999 veröffentlichte die IEEE (siehe Abschnitt 18.1.1) eine Spezifikation für ein kryptografisches Netzwerkprotokoll zur Absicherung von WLANs, das den Namen **WEP** (**Wireless Equivalent Privacy**) trug (eine gute Einführung bietet [Chandr]). WEP realisiert einen sicheren Kanal zwischen Alices Endgerät und dem Access Point. Es handelt sich dabei um ein recht simples Protokoll, das nur ein symmetrisches Verfahren einsetzt und keine PKI- oder Public-Key-Unterstützung vorsieht. WEP ist nicht verhandlungsfähig, es bietet weder Forward Security noch Backward Security und ist außerdem zustandslos, da jedes Datenpaket unabhängig von den anderen verschlüsselt wird.

Abb. 33–3 *WEP sichert die drahtlose Teilstrecke zwischen einem Endgerät und einem WLAN-Access-Point ab.*

Funktionsweise von WEP

WEP verwendet das symmetrische Verschlüsselungsverfahren RC4 zur Verschlüsselung. Die Schlüssellänge für den RC4-Schlüssel beträgt 64 Bit, 128 Bit oder 256 Bit (RC4 unterstützt auch andere Schlüssellängen, diese sind hier jedoch nicht vorgesehen). Wie wir sehen werden, sind jeweils 24 Schlüssel-Bits nicht geheim, weshalb die effektive Schlüssellänge bei 40, 104 oder 232 Bit liegt. Einen Schlüssel dieser Länge müssen die Kommunikationspartner (also typischerweise die WLAN-Karte in Alices PC und der Access Point) gemeinsam haben. Er wird *WEP-Schlüssel* genannt. Da keine Public-Key-Kryptografie zum Einsatz kommt, muss der WEP-Schlüssel manuell übergeben werden. Meist händigt der WLAN-Betreiber Alice den WEP-Schlüssel als Hexadezimalzahl auf einem Stück Papier

aus, und Alice muss ihn eintippen. Falls Alice den Access Point selbst betreibt, muss sie den WEP-Schlüssel selbst festlegen und sowohl an ihren PC als auch am Access Point eintragen.

Da RC4 eine Stromchiffre ist, wäre es unklug, jedes im WLAN verschickte Datenpaket mit demselben Schlüssel zu verschlüsseln. Deshalb sieht WEP nicht vor, dass WLAN-Karte und Access Point den WEP-Schlüssel unverändert zur Verschlüsselung nutzen. Stattdessen wird ein Sitzungsschlüssel verwendet, der sich aus zwei Bestandteilen zusammensetzt: dem WEP-Schlüssel und einem 24-Bit-Wert, der *Initialisierungsvektor* genannt wird. Der Initialisierungsvektor ist nicht geheim. Dadurch erklärt sich, warum die effektive Schlüssellänge um 24 Bit kürzer ist als die Länge des Sitzungsschlüssels.

Die Initialisierungsroutine des sicheren Kanals, den WEP realisiert, gibt es in zwei Varianten. In der Variante *Open System Authentication* (OSA) erfolgt keine Authentifizierung. Die WEP-Kommunikation startet in diesem Fall also gleich mit der Datenaustauschroutine. Da diese verschlüsselt abläuft, kann Angreifer Mallory trotz fehlender initialer Authentisierung nicht allzu viel ausrichten, denn er kennt den WEP-Schlüssel nicht. Es sind jedoch verschiedene Angriffe auf dieses Szenario denkbar, weshalb von der Verwendung von OSA abzuraten ist. In der Variante *Shared Key Authentification* (SKA) erfolgt dagegen eine Challenge-Response-Authentifizierung von Alice gegenüber dem Access Point. Letzterer schickt eine Challenge, die Alice mit RC4 verschlüsselt, wobei sie den WEP-Schlüssel und einen Initialisierungsvektor verwendet. Eine Authentifizierung des Access Points findet nicht statt. Es ist im Rahmen von SKA auch nicht vorgesehen, den Sitzungsschlüssel mit kryptografischen Mitteln aus dem WEP-Schlüssel (der ja ein wertvoller Masterschlüssel ist) abzuleiten. Stattdessen ist der Sitzungsschlüssel bis auf einen öffentlich bekannten Teil mit dem WEP-Schlüssel identisch. Die Datenaustauschroutine von WEP läuft wie folgt ab:

1. Alice (genau genommen ihre WLAN-Karte) nimmt das zu verschickende Datenpaket und berechnet dazu eine Prüfsumme. Das zu verwendende Prüfsummenverfahren heißt CRC (Cyclic Redundancy Check) und ist vom WLAN-Standard vorgegeben. Es ist kein kryptografisches Verfahren.
2. Alice verschlüsselt das Datenpaket inklusive Prüfsumme mit dem RC4-Verfahren, wobei sie den Sitzungsschlüssel verwendet. Als Initialisierungsvektor nimmt sie eine Zufallszahl der Länge 24 Bit.
3. Alice schickt das verschlüsselte Datenpaket zusammen mit dem (unverschlüsselten) Initialisierungsvektor an den Access Point.
4. Der Access Point setzt aus dem erhaltenen Initialisierungsvektor und dem ihm bekannten WEP-Schlüssel den Sitzungsschlüssel zusammen. Damit entschlüsselt er das Datenpaket.
5. Der Access Point überprüft die Prüfsumme. Ist sie korrekt, dann kann er sicher sein, dass es sich tatsächlich um ein korrektes Datenpaket handelt und nicht etwa um Datenmüll, den Mallory ihm zugeschickt hat.

Wenn umgekehrt der Access Point eine Nachricht an Alice schickt, dann läuft das Protokoll mit vertauschten Rollen ab. Der WEP-Schlüssel bleibt dabei über längere Zeit konstant, während sich der Initialisierungsvektor mit jeder verschlüsselten Nachricht ändert. Dadurch ist gewährleistet, dass ein Sitzungsschlüssel nur einmal eingesetzt wird.

WEP-Sicherheitsprobleme

WEP entstand zu einer Zeit, als die Entwicklung kryptografischer Netzwerkprotokolle längst kein Hexenwerk mehr war. Fachleute und Fachliteratur zu diesem Thema gab es damals bereits genug. Umso erstaunlicher ist es, dass WEP einige erhebliche Schwächen aufweist, die bereits in der Konzeptionsphase hätten auffallen müssen. Eine Schwachstelle ist beispielsweise der Initialisierungsvektor, dessen Länge nur 24 Bit beträgt. Wie man leicht nachrechnet, gibt es 2^{24}, also etwa 16,7 Millionen Werte, die der Initialisierungsvektor annehmen kann. Nach dem Geburtstagsproblem benötigt Mallory nur etwa 4.000 abgefangene Nachrichten, um mit einer Wahrscheinlichkeit von 50 Prozent auf einen doppelt vorkommenden Initialisierungsvektor zu stoßen. Ein solcher führt zu zwei mit gleichem Schlüssel verschlüsselten Paketen, und das ist bei einer Stromchiffre eine Todsünde (siehe Abschnitt 16.1).

Einen noch besseren Angriff fanden 2001 Fluhrer, Mantin und Shamir [FlMaSh]. Es handelt sich dabei genau genommen um einen Angriff auf RC4, der im Zusammenhang mit WEP besonders gut funktioniert. Diesen Angriff kennen Sie bereits aus Abschnitt 16.2. Ioannidis, Rubin und Stubblefield setzten diese Attacke in eine praktische Implementierung um und testeten diese in einem WEP-gesicherten WLAN [IoRuSt]. Sie benötigten jeweils etwa 5 bis 6 Millionen abgefangene Datenpakete, um den Schlüssel zu rekonstruieren. In einem stark frequentierten Netz dauert das Sammeln einer solchen Datenmenge nur ein paar Stunden.

Inzwischen haben Andrei Pyshkin, Erik Tews und Ralf-Philipp Weinmann [PyTeWe] sowie Andreas Klein [Klein] die Fluhrer-Mantin-Shamir-Attacke sogar noch verbessert (siehe Abschnitt 16.2). Mithilfe der neuen Erkenntnisse genügen 40.000 Datenpakete, um mit 50-prozentiger Wahrscheinlichkeit den Schlüssel bestimmen zu können. Bei 85.000 Paketen beträgt die Wahrscheinlichkeit 95 Prozent. In der Praxis dauert es etwa eine Minute, um auf diese Weise den Schlüssel eines WEP-gesicherten WLANs zu ermitteln. Auf weitere Schwachstellen in WEP machten Nikita Borisov und Ian Goldberg aufmerksam [BoGoWa]. Unter anderem wiesen sie darauf hin, dass manche WLAN-Karten als Initialisierungsvektor keine Zufallszahl verwenden, sondern lediglich einen Wert hochzählen. Das Fazit der beiden war daher eindeutig: »Wired Equivalent Privacy (WEP) isn't.«

33.2.2 WPA

Die zahlreichen Schwächen von WEP zwangen die Verantwortlichen, schnell für Verbesserungen zu sorgen. Daher entwickelte die zuständige IEEE-Arbeitsgruppe den Standard IEEE 802.11i, in dem ein Nachfolger von WEP beschrieben wird [Chandr]. Dieser neue Standard setzt auf den AES statt auf RC4 und erfordert deshalb etwas leistungsfähigere Hardware als WEP. Daher war 802.11i für viele bestehende Geräte ungeeignet. Dies nahm die Wi-Fi-Allianz – dies ist die Interessenvertretung der 802.11-Hersteller – zum Anlass, eine abgespeckte Version von 802.11i zu entwickeln, die nach wie vor RC4 verwendet und dadurch auch ohne neue Hardware ins Feld zu bringen ist. Diese abgespeckte Version hat den Namen **WPA** (**Wi-Fi Protected Access**). Für 802.11i hat sich derweil die Bezeichnung **WPA2** etabliert. Diese Bezeichnungen sind etwas verwirrend, da WPA2 vor WPA entstanden ist. Andererseits steckt dennoch eine gewisse Logik dahinter, weil WPA2 komplexer und leistungsfähiger ist als WPA.

Funktionsweise von WPA

WPA sieht zwei Varianten der Verschlüsselung vor: Die eine heißt **Temporal Key Integrity Protocol** (**TKIP**) und ist als Bestandteil einer WPA-Implementierung vorgeschrieben. Daneben gibt es noch CCMP, das in WPA noch optional, in WPA2 aber Pflicht ist. Im Folgenden geht es nur um TKIP, wogegen CCMP anschließend im Zusammenhang mit WPA2 beschrieben wird. TKIP verwendet zwar wie WEP RC4 zur Verschlüsselung, doch der Umgang mit den Daten und den Schlüsseln ist etwas komplexer. TKIP sieht einen Masterschlüssel vor, der *Pairwise Master Key* (*PMK*) genannt wird. Er hat eine Länge von 256 Bit und wird nur zur Ableitung anderer Schlüssel verwendet, damit eine Known-Key-Attacke erschwert wird. Der PMK muss Alice und dem WLAN-Betreiber bekannt sein. Die Ableitung anderer Schlüssel aus dem PMK ist wie folgt geregelt (der WLAN-Betreiber muss dieselben Schritte wie Alice durchführen):

1. Alice wendet auf den PMK eine Funktion an, die auf dem HMAC-Verfahren mit SHA-1 basiert. Als Ergebnis erhält sie einen 512-Bit-Wert. Diesen zerlegt sie in vier Schlüssel zu je 128 Bit. Diese vier Schlüssel heißen *EAPol-Schlüssel*, *EAPol-MIC-Schlüssel*, *Daten-Schlüssel* und *Daten-MIC-Schlüssel*.
2. Alice bildet aus dem Daten-Schlüssel, einem Initialisierungsvektor und der Geräteadresse ihrer WLAN-Karte einen *Per-Packet-Encryption-Schlüssel* (PPE-Schlüssel). Dazu verwendet sie ein Verfahren, in dem zweimal eine Funktion angewandt wird, die auf einer AES-S-Box basiert. Der PPE-Schlüssel ist 128 Bit lang. TKIP sieht vor, dass Alice bzw. der Access Point für jedes versendete Datenpaket einen neuen PPE-Schlüssel generieren.

Der Initialisierungsvektor besteht aus einem ersten Teil von 16 Bit und einem 32 Bit langen zweiten Teil. Der Wert des ersten Teils erhöht sich von Datenpaket zu Datenpaket um eins. Da der Initialisierungsvektor insgesamt 48 Bit lang ist, gibt es diesbezüglich eine deutlich höhere Sicherheit als bei WEP, das 24 Bit als Initialisierungsvektor-Länge vorsieht. Eine Verschlüsselung läuft wie folgt ab:

1. Alice nimmt die zu versendenden Nutzdaten und berechnet mithilfe des Daten-MIC-Schlüssels einen Hashwert. Dazu verwendet sie die schlüsselabhängige Hashfunktion MICHAEL. Dabei handelt es sich nicht um eine kryptografische Hashfunktion (diese würde zu viel Rechenkapazität verbrauchen), sondern um ein im Vergleich zu CRC etwas komplexeres konventionelles Verfahren.
2. Die Daten mit dem Hashwert bearbeitet Alice mit CRC. Dadurch entsteht ein zweiter Hashwert. Die Verwendung von CRC dient der Abwärtskompatibilität.
3. Alice verschlüsselt die Daten inklusive der beiden Hashwerte mit RC4. Als Schlüssel verwendet sie den aktuellen PPE-Schlüssel.
4. Den entstehenden Geheimtext schickt Alice zusammen mit dem unverschlüsselten Initialisierungsvektor als WLAN-Paket an den Access Point.

Beachten Sie, dass der PPE-Schlüssel dieselbe Länge wie der Sitzungsschlüssel von WEP hat und ebenfalls für RC4 verwendet wird. Dadurch ist eine Kompatibilität mit WEP gegeben, die sich insbesondere auf die verwendete Hardware bezieht.

WPA und EAP

Der WEP-Standard sagt nichts darüber aus, wie Alice und die Basisstation ihren gemeinsamen Schlüssel vereinbaren. Bei WPA ist das anders. WPA unterstützt das in Abschnitt 33.1.2 vorgestellte EAP, das ein Rahmenwerk für die Authentifizierung und den Schlüsselaustausch (Initialisierungsroutine des sicheren Kanals) darstellt. Genauer gesagt wird die EAP-Variante 802.1x unterstützt, die EAP an die Nutzung in LANs (nicht nur in WLANs, sondern auch in drahtgebundenen LANs) anpasst [Chandr].

Über 802.1x stehen WPA mehrere EAP-Methoden zur Verfügung. Die erwähnten EAPoL- und EAPoL-MIC-Schlüssel können in einigen dieser Methoden verwendet werden (*EAPol* steht für *EAP over LAN*). Mit anderen Worten heißt dies, dass der Betreiber eines WLANs beim WPA-Einsatz aus mehreren Authentifizierungsprotokollen wählen kann. Derzeit sind fünf EAP-Methoden fester Bestandteil von WPA. Das wichtigste davon ist das bereits erwähnte EAP-TLS, das einen TLS-ähnlichen Authentifizierungsablauf vorsieht und dabei digitale Zertifikate unterstützt. EAP-TLS nutzt einen RADIUS- oder DIAMETER-Server für die Speicherung der Authentifizierungsdaten. Dadurch ist es möglich, dass mehrere Access Points durch einen Zugriff auf denselben RADIUS- bzw. DIAMETER-Server auf dieselbe Weise authentifizieren können.

33.2.3 WPA2

WPA2 ist mit dem IEEE-Standard 802.11i identisch. Es soll WPA, das provisorischen Charakter hat, ablösen. WPA2 sieht das erwähnte **CCMP** (**Counter Mode with Cipher Block Chaining Message Authentication Code Protocol**) als festen Bestandteil vor und empfiehlt dessen Verwendung. TKIP wird nur aus Kompatibilitätsgründen unterstützt. CCMP verwendet den AES. Es handelt sich dabei genau genommen um eine Betriebsart des AES, die sowohl verschlüsselt als auch einen schlüsselabhängigen Hashwert bildet. Eine weitere Änderung: Die (nichtkryptografische) Hashfunktion MICHAEL wird bei CCMP nicht benötigt, da die Hashwert-Generierung in die Betriebsart integriert ist. Die Ableitung der Schlüssel erfolgt wie bei TKIP, außer dass der Daten-MIC-Schlüssel nicht benötigt wird. WPA2 unterstützt ansonsten dieselben EAP-Methoden wie WPA.

33.3 Kryptografie für Bluetooth

Bluetooth ist ein Standard für die drahtlose Kommunikation über kurze Distanzen [Chandr]. Bluetooth ist vor allem für sogenannte Ad-hoc-Netzwerke gedacht, also für Computernetze, die ohne spezielle Infrastruktur spontan eingerichtet werden können. Typische Anwendungen sind das drahtlose Verbinden von Tastatur, Maus und Drucker mit einem PC oder das Verbinden eines Headsets mit einem Telefon. Auch Smartphones bieten oft eine Bluetooth-Unterstützung. Mit dieser lässt sich beispielsweise eine Verbindung zu einem Bankautomaten herstellen, wodurch Alice über ihr Smartphone den Kontostand abrufen oder eine Überweisung veranlassen kann.

Abb. 33–4 *Bluetooth ist eine Technologie für die drahtlose Datenübertragung über kurze Strecken. Kryptografie ist fest eingebaut.*

Der Bluetooth-Standard sieht vor, dass es zwei Arten von Geräten gibt: Master und Slaves. In der einfachsten Form (und nur diese soll uns interessieren) gibt es in einem Bluetooth-Netz einen Master und bis zu sieben Slaves. Der Master kann alle Slaves ansprechen. Die Slaves können nur mit dem Master kommunizieren, eine Kommunikation untereinander ist nicht möglich. Bluetooth-Kommunikation findet daher immer zwischen einem Master und einem Slave statt. Der Master kann beispielsweise ein PC sein, an den die Slaves Tastatur, Maus und Drucker angebunden sind.

33.3.1 Grundlagen der Bluetooth-Kryptografie

Wie bei den meisten anderen drahtlosen Netztechnologien sah der Bluetooth-Standard von Anfang an die Verwendung von Kryptografie vor (in Form eines sicheren Kanals zwischen Master und Slave). Der Grund dafür ist offensichtlich: Durch die drahtlose Datenübertragung wäre eine Technik wie Bluetooth ohne Kryptografie ein leichtes Angriffsziel für einen Bösewicht wie Mallory. Bluetooth unterstützt (wie bei einem sicheren Kanal üblich) sowohl ein Verfahren für die Authentifizierung als auch eine Verschlüsselungsfunktion. Ob diese beiden Werkzeuge (oder nur eines davon oder gar keines) zum Einsatz kommen, hängt von der verwendeten Sicherheitsstufe ab. Bluetooth definiert drei Sicherheitsstufen (von 1 bis 3 durchnummeriert), wobei der Master nicht notwendigerweise dieselbe Stufe verwenden muss wie der Slave. Dies bedeutet, dass es insgesamt neun Kombinationen von Sicherheitsstufen gibt. Alle neun sind gemäß der Bluetooth-Spezifikation zulässig.

Beim Einsatz der drei Sicherheitsstufen sind zwei Aspekte zu beachten: Zum einen kann der Betreiber einer Bluetooth-Architektur sowohl auf dem Master als auch auf dem Slave eine Sicherheits-Policy festlegen, die eine Auswirkung auf die Verwendung der beiden genannten kryptografischen Werkzeuge hat. Zum anderen kann auch die jeweilige Anwendung (z.B. eine Smartphone-Software für Bankgeschäfte oder ein Tastaturtreiber auf dem PC) Einfluss auf Authentifizierung und Verschlüsselung nehmen. Vor diesem Hintergrund ist der folgenden Tabelle zu entnehmen, wie Bluetooth Authentifizierung und Verschlüsselung in Abhängigkeit von den jeweiligen Sicherheitsstufen einsetzt:

	Slave		
Master	**Stufe 1**	**Stufe 2**	**Stufe 3**
Stufe 1	**Authentifizierung**: nein	**Authentifizierung**: wenn Master-Anwendung es verlangt	**Authentifizierung**: ja
	Verschlüsselung: nein	**Verschlüsselung**: wenn Master-Anwendung es verlangt	**Verschlüsselung**: wenn Master-Policy es verlangt
Stufe 2	**Authentifizierung**: wenn Slave-Anwendung es verlangt	**Authentifizierung**: wenn Master- oder Slave-Anwendung es verlangt	**Authentifizierung**: ja
	Verschlüsselung: wenn Slave-Anwendung es verlangt	**Verschlüsselung**: wenn Master- oder Slave-Anwendung es verlangt	**Verschlüsselung**: wenn Master-Policy oder Slave-Anwendung es verlangt
Stufe 3	**Authentifizierung**: ja	**Authentifizierung**: ja	**Authentifizierung**: ja
	Verschlüsselung: wenn Slave-Policy es verlangt	**Verschlüsselung**: wenn Master-Anwendung oder Slave-Policy es verlangt	**Verschlüsselung**: wenn Master- oder Slave-Policy es verlangt

Der Bluetooth-Standard schreibt vor, dass jede konforme Implementierung Authentifizierung unterstützen muss. Verschlüsselung wird dagegen nicht zwingend gefordert. Ein Blick auf die Tabelle zeigt, dass auch eine Implementierung, die keine Verschlüsselung unterstützt, in Sicherheitsstufe 3 betrieben werden kann. Dies bedeutet, dass auch die höchste Sicherheitsstufe kein Garant dafür ist, dass übertragene Daten verschlüsselt werden.

Verwendete Verfahren

Bei Bluetooth kommen folgende fünf kryptografische Verfahren zur Anwendung:

- E_0: E_0 ist eine Stromchiffre, die Sie aus Abschnitt 16.4 kennen.
- E_{21}: Diese Funktion kann man als kryptografische Hashfunkion mit einem Urbild fester Länge (176 Bit) interpretieren. Sie basiert auf dem symmetrischen Verschlüsselungsverfahren SAFER+. Ein Teil des Urbilds wird als SAFER+-Schlüssel verwendet, ein anderer Teil als Klartext. Das Ergebnis der Verschlüsselung ist der Hashwert (128 Bit). Damit die Berechnung des Hashwerts nicht umkehrbar ist, wird eine leicht geänderte Variante von SAFER+ verwendet, bei der eine Entschlüsselung auch bei bekanntem Schlüssel nicht funktioniert.
- E_{22}: Auch hierbei handelt es sich um eine kryptografische Hashfunktion, die auf SAFER+ basiert. Die Länge des Urbilds kann innerhalb bestimmter Grenzen variieren. Ein Teil des Urbilds (im Bluetooth-Einsatzszenario sind dies ein Schlüssel und eine Geräteadresse) wird zu einem SAFER+-Schlüssel verarbei-

tet, mit dem der Rest (in der Bluetooth-Praxis eine Zufallsfolge) verschlüsselt wird. Auch hier kommt die erwähnte, leicht geänderte Variante von SAFER+ zum Einsatz, bei der die Entschlüsselung nicht funktioniert.

- E_1: Dieses Verfahren kann man sich als schlüsselabhängige Hashfunktion vorstellen, allerdings hat das Urbild eine feste Länge (176 Bit). Die Funktionsweise von E_1 basiert auf einer doppelten Anwendung von SAFER+ (einmal unverändert, einmal in der erwähnten Modifizierung). Die Schlüssellänge sowie die Länge des Hashwerts betragen jeweils 128 Bit.
- E_3: Auch E_3 kann man als schlüsselabhängige Hashfunktion mit Urbild fester Länge betrachten. Die Funktionsweise ähnelt der von E_1. Auch hier kommt SAFER+ doppelt zur Anwendung (einmal unverändert, einmal modifiziert). Das Urbild hat eine Länge von 224 Bit, der Schlüssel von 128 Bit. Der Hashwert ist 128 Bit lang.

Was die Kryptografie betrifft, setzt Bluetooth also auf ein bewährtes Verfahren (SAFER+) in spezieller Anwendung sowie auf einen speziell entwickelten Algorithmus (E_0). SAFER+ wird nicht zum Verschlüsseln, sondern nur als Baustein für vier verschiedene kryptografische Hashfunktionen (teilweise schlüsselabhängig) verwendet.

Bluetooth-Schlüssel

Das Komplexeste an der Kryptografie, die in Bluethooth eingesetzt wird, sind die Schlüssel. Die für uns relevanten Schlüssel heißen *PKEY*, *Initialisation Key*, *Link Key*, *Encryption Key* und *Constraint Key*. Es gibt noch einige weitere Schlüssel, die etwa dann von Belang sind, wenn ein Master mit mehreren Slaves kommuniziert. Dieses Szenario wollen wir jedoch nicht betrachten. Bei den genannten Schlüsseln handelt es sich in allen Fällen um Schlüssel für symmetrische Verfahren (also um Zufallsfolgen). Die Länge beträgt meist 128 Bit. Asymmetrische Verfahren kommen in der Bluetooth-Spezifikation nicht vor.

Der Bluetooth-Standard setzt voraus, dass Master und Slave einen gemeinsamen Schlüssel besitzen. Dieser wird *PKEY* (Pass-Key) genannt. Er hat eine Länge von maximal 128 Bit, kann aber auch kürzer sein. Ein Protokoll zur Festlegung des *PKEY* oder zu dessen Übertragung vom Master zum Slave (beispielsweise mit RSA) gibt es im Bluetooth-Standard nicht. Der *PKEY* muss daher von einer nicht standardisierten Software eingerichtet oder manuell eingegeben werden. Viele Anwendungen leiten den *PKEY* von einer PIN (also von einer meist vierstelligen Geheimzahl) ab, die der Anwender eingeben muss. Dabei ist zu beachten, dass ein aus einer vierstelligen Zahl abgeleiteter Schlüssel nicht gerade die höchste Sicherheit bietet.

Ist der *PKEY* sowohl dem Master als auch dem Slave bekannt, dann werden die anderen Bluetooth-Schlüssel nach folgendem Protokoll festgelegt:

1. Der Master generiert aus dem *PKEY* und einem Zufallswert (*IN_RAND*) mit der kryptografischen Hashfunktion E_{22} den *Initialisation Key*. *IN_RAND* sendet er an den Slave.
2. Der Slave verfährt auf gleiche Weise wie zuvor der Master: Aus dem *PKEY* und *IN_RAND* berechnet er mit E_{22} den *Initialisation Key*. Der Zweck des *Initialisation Key* ist es, dass der *PKEY* nicht direkt am Protokoll beteiligt und dadurch besser geschützt ist.
3. Der Master generiert einen weiteren Zufallswert (LK_RAND_A). Daraus sowie aus seiner Geräteadresse berechnet er mit der kryptografischen Hashfunktion E_{21} den Wert K_A. Anschließend berechnet er die Exklusiv-oder-Verknüpfung von LK_RAND_A und dem *Inititialisation Key*. Diese Verknüpfung kommt einer Verschlüsselung (One-Time-Pad) von LK_RAND_A mit dem *Initialisation Key* gleich. Das Ergebnis der Verschlüsselung sendet er an den Slave.
4. Der Slave entschlüsselt LK_RAND_A mit dem *Initialisation Key*. Aus LK_RAND_A sowie aus der Geräteadresse des Masters berechnet er mit E_{21} den Wert K_A. Außerdem generiert er einen weiteren Zufallswert (LK_RAND_B) und berechnet anschließend die Exklusiv-oder-Verknüpfung von LK_RAND_B und dem *Initialisation Key*. Dies kommt einer One-Time-Pad-Verschlüsselung von LK_RAND_B gleich. Das Ergebnis der Verschlüsselung sendet er an den Master. Außerdem wendet der Slave das Verfahren E_{21} auf LK_RAND_B und seine eigene Geräteadresse an, wodurch er K_B erhält. Der *Link Key* ergibt sich aus $K_A \oplus K_B$.
5. Der Master entschlüsselt LK_RAND_B mit dem *Initialisation Key*. Aus LK_RAND_B und der Geräteadresse des Slave berechnet er mit dem Verfahren E_{21} den Wert K_B. Nun kann auch der Master den *Link Key* berechnen, der sich aus $K_A \oplus K_B$ ergibt.
6. Aus dem *Link Key*, einem weiteren Zufallswert (*EN_RAND*) sowie einer weiteren Variablen (es handelt sich um die Kennung des Authentifizierungsprozesses) generiert der Master den *Encryption Key*. Der *Link Key* dient dabei als Schlüssel für E_3, die anderen beiden Eingaben als Urbild. *EN_RAND* sendet er an den Slave. Außerdem berechnet der Master aus dem Encryption Key den Constraint Key. Der *Constraint Key* hat den Zweck, die effektive Schlüssellänge zu verkürzen, damit die jeweilige Implementierung Gesetzen entspricht, die die Verwendung oder den Export von Kryptografie beschränken (in den USA gab es früher Exportbeschränkungen, die nur die Ausfuhr von maximal 40 Bit effektiver Schlüssellänge zuließen). Die Berechnung des Constraint Key sieht vor, einen konfigurierbaren Teil des Encryption Key abzuschneiden und das Ergebnis nach einem nichtkryptografischen Verfahren auf 128 Bit zu expandieren. Da die US-Exportbeschränkungen nicht mehr aktuell sind, verwenden die meisten Implementierungen den *Encryption Key* als *Constraint Key*.

7. Der Slave berechnet ebenfalls den *Encryption Key*. Dazu wendet er E_3 mit dem *Link Key* als Schlüssel auf *EN_RAND* und die zusätzliche Variable an. Anschließend berechnet er den *Constraint Key* aus dem *Encryption Key*.

Nach Abarbeitung dieses Protokolls besitzen der Master und der Slave einen gemeinsamen *Link Key* sowie einen gemeinsamen *Constraint Key*. Diese beiden Schlüssel werden zur Authentifizierung und zur Verschlüsselung verwendet. Zum beschriebenen Protokoll ist noch zu bemerken, dass es verschiedene Abwandlungen davon gibt. So können die letzten beiden Schritte entfallen, wenn keine Verschlüsselung durchgeführt werden soll. Bei Teilen des Protokolls kann zudem die Rolle von Master und Slave vertauscht werden. Und schließlich gibt es noch eine erweiterte Variante des Protokolls für den Fall, dass mehrere Slaves daran beteiligt sind.

33.3.2 Bluetooth-Authentifizierung und -Verschlüsselung

Nach dieser Vorarbeit können der Master und der Slave eine Authentifizierung durchführen sowie die übertragenen Daten verschlüsseln. Dabei ist die Sicherheitsstufen-Tabelle in Abschnitt 33.3.1 zu beachten. Diese besagt, dass nicht in allen Fällen eine Authentifizierung stattfindet. Falls ja, muss diese nicht gegenseitig sein. Zudem wird gemäß der Tabelle nicht in allen Fällen verschlüsselt. Wenn jedoch eine Verschlüsselung oder Authentifizierung durchgeführt wird, dann geschieht dies wie im Folgenden beschrieben.

Authentifizierung

Die von Bluetooth vorgesehene Authentifizierung folgt einem einfachen Challenge-Response-Protokoll. Dieses sieht wie folgt aus:

1. Der Master schickt eine Challenge (also einen Zufallswert) an den Slave.
2. Der Slave wendet auf die Challenge und seine Bluetooth-Geräteadresse das Verfahren E_1 an. Als E_1-Schlüssel verwendet er den *Link Key*. Das Ergebnis geht als Response an den Master zurück.

Dieses Protokoll kann auch umgekehrt ablaufen (der Slave authentifiziert den Master). Auch eine wechselseitige Authentifizierung ist möglich. Dabei kommt der beschriebene Ablauf zweimal zum Einsatz, wobei beim zweiten Mal die Rollen vertauscht werden.

Verschlüsselung

Falls die verwendeten Sicherheitsstufen sowie die Anwendungen oder Policys es vorsehen, können Master und Slave den Bluetooth-Datenaustausch verschlüsseln. Dazu verwenden sie das Verfahren E_0. Als Schlüssel wird der *Constraint Key* verwendet. Für jedes Datenpaket wird das Verfahren E_0 neu initialisiert. Da bei einer Stromchiffre (E_0 ist eine solche) derselbe Schlüssel nicht mehrfach verwen-

det werden darf, gehen in die Initialisierung von E_0 zusätzlich 26 Bits aus der Systemuhr sowie die Geräteadresse des jeweiligen Absenders (Master oder Slave) und der Zufallswert *EN_RAND* mit ein.

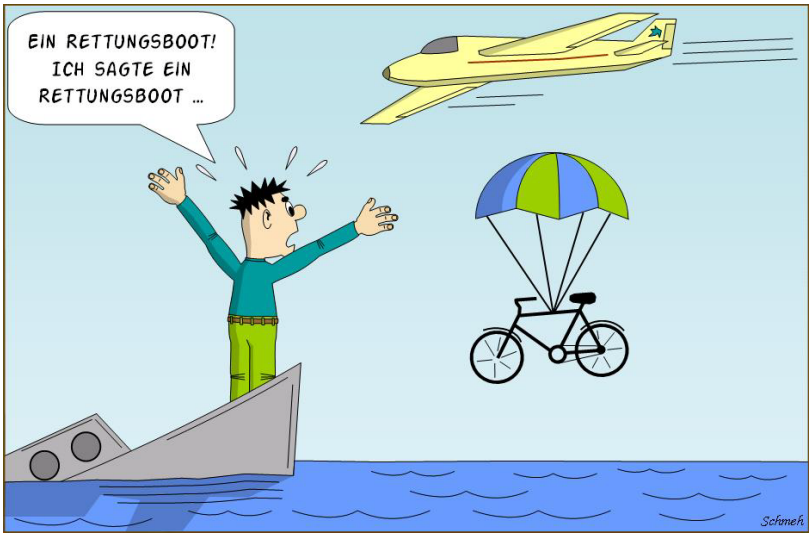
Logischerweise werden jeweils nur die Nutzdaten eines Bluetooth-Pakets verschlüsselt. Der Header eines Pakets wird dagegen im Klartext übertragen. Vor dem Verschlüsseln wird auf die Nutzdaten eine nichtkryptografische Hashfunktion angewandt, deren Ergebnis mit verschlüsselt wird. Durch den zusätzlichen Hashwert kann der empfangende Master oder Slave nach dem Entschlüsseln unmittelbar feststellen, ob es sich um einen zulässigen Klartext handelt. Falls Mallory also ein gefälschtes Datenpaket verschickt, bei dem eine sinnlose Bit-Folge die verschlüsselte Nutzlast ersetzt, fällt dies sofort auf.

33.3.3 Angriffe auf die Bluetooth-Sicherheitsarchitektur

Bluetooth gehört mit GSM und 802.11-WLAN zu den Netzwerktechnologien, die Kryptografie als festen Bestandteil vorsehen. Obwohl die Bluetooth-Standardisierer hätten gewarnt sein können, machten auch sie unnötige Fehler beim Umgang mit den kryptografischen Werkzeugen. Ein Problem ist zweifellos das Verschlüsselungsverfahren E_0, das speziell für Bluetooth entwickelt wurde. Wie in Abschnitt 16.4 beschrieben, bietet E_0 trotz eines 128-Bit-Schlüssels nur die Sicherheit von etwa 84 Bit Schlüssellänge. Würde man E_0 für größere Datenmengen verwenden, ohne den Schlüssel zu ändern (bei Bluetooth ist dies nicht der Fall), dann wäre das Verfahren sogar mit moderatem Aufwand zu knacken. Zwar gibt es bisher noch keinen praxistauglichen Angriff auf Bluetooth, der die Schwäche von E_0 nutzt, doch der Sicherheitspuffer erscheint nicht gerade groß, und so könnte die nächste Kryptoanalyse schon verheerende Auswirkungen haben.

Während die Verschlüsselung von Bluetooth bisher noch ausreichend sicher scheint, kann man das für andere Teile der Bluetooth-Architektur nicht behaupten. 2005 veröffentlichten Shaked und Wool einen Angriff auf das Bluetooth-Protokoll, der voraussetzt, dass der *PKEY* von einer PIN abgeleitet ist [ShaWoo]. Der Angriff ist aus kryptografischer Sicht eher simpel. Er sieht vor, dass Angreifer Mallory die Generierung der diversen Bluetooth-Schlüssel sowie die Authentifizierung zwischen Master und Slave abhört. Anschließend ermittelt Mallory per Brute Force, welche PIN zu den abgehörten Werten passt. Bei einer vierstelligen PIN (es gibt also 10.000 Möglichkeiten) dauern diese Berechnungen auf einem PC nur Sekundenbruchteile. Natürlich ist dieser Angriff nichts Sensationelles – es war schließlich von Anfang an klar, dass eine vierstellige PIN keine große Sicherheit bietet. Da jedoch zahlreiche Bluetooth-Produkte eine vierstellige PIN vorsehen, funktioniert diese Methode in der Praxis oft genug. Wer eine Bluetooth-Implementierung sicher betreiben will, sollte eine ausreichend lange PIN oder am besten gleich einen zufälligen 128-Bit-Schlüssel verwenden. Darüber hinaus sollte man vor allem darauf hoffen, dass keine neuen Angriffe auf E_0 bekannt werden.

34 IPsec (Schicht 3)



Bei unserem Streifzug durch die OSI-Schichten kommen wir nun zu Schicht 3 (Vermittlungsschicht). Dort haben wir es in der Praxis meist mit IP (Internet Protocol) zu tun – dem Netzwerkprotokoll, das dem Internet seinen Namen gab. IP sah ursprünglich keine Kryptografie vor. Seit Ende der neunziger Jahre gibt es jedoch mit dem Standard **IPsec** eine entsprechende Erweiterung. IPsec wird innerhalb der IETF entwickelt und ist unter anderem ein fester Bestandteil der neuen IP-Generation IP Version 6 (IPv6).

Die Protokollnachrichten von IP werden IP-Pakete genannt. Jedes Paket enthält neben der Nutzlast einen Header und wird unabhängig von den anderen Paketen über das Netz geschickt. Der Header ist in zahlreiche Felder unterteilt, die Angaben über Sender, Empfänger, Länge der Nutzlast und vieles andere machen. IPsec ermöglicht die Verschlüsselung von IP-Paketen und kann deren Integrität durch eine schlüsselabhängige Hashfunktion sicherstellen – es realisiert also die Datenaustauschroutine eines sicheren Kanals. Zur Kommunikation wer-

den Sicherheitsassoziationen (siehe Abschnitt 20.2.1) genutzt. IPsec setzt voraus, dass Alice und Bob einen gemeinsamen geheimen Schlüssel besitzen.

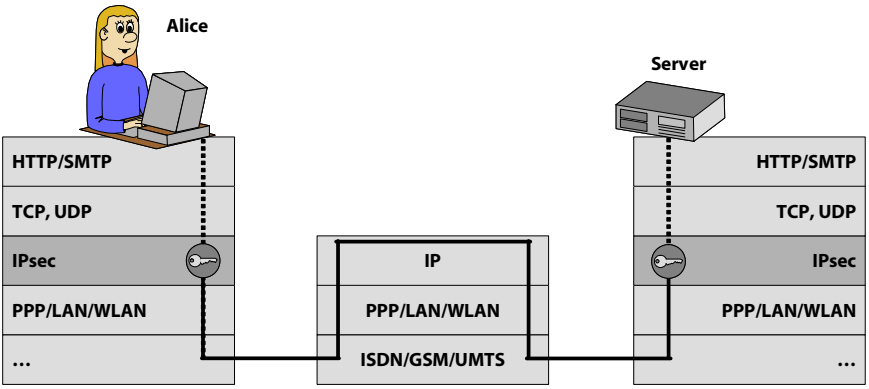


Abb. 34–1 *IPsec ermöglicht den Einsatz von Kryptografie in Schicht 3 des OSI-Modells. Mit IPsec kann auch nur eine Teilstrecke des Internets kryptografisch abgesichert werden.*

Der Aufbau und das Management von Sicherheitsassoziationen sowie der Schlüsselaustausch (also die Initialisierungsroutine des sicheren Kanals) sind nicht Bestandteil von IPsec. Diese Aufgaben werden stattdessen meist von einem Protokoll namens **IKE** (**Internet Key Exchange**) übernommen. IKE ist zwar kein Bestandteil von IPsec, doch de facto ist es so eng damit verbunden, dass man es dazuzählen kann. Für sich allein ist IPsec ein zustandsloses, nicht verhandlungsfähiges Protokoll. Mit IKE gelten diese Einschränkungen dagegen nicht.

34.1 Bestandteile von IPsec

IPsec sieht (ohne IKE) zwei Bestandteile vor: ESP und AH. Diese beiden Komponenten haben unterschiedliche Aufgaben und können gleichzeitig eingesetzt werden.

34.1.1 ESP

Der erste Bestandteil von IPsec wird **ESP** (**Encapsulated Security Payload**) genannt und ist in [RFC4303] beschrieben. ESP ermöglicht die Verschlüsselung der Nutzdaten (Payload) eines IP-Pakets und erlaubt zusätzlich die Verwendung einer schlüsselabhängigen Hashfunktion. Die meisten Felder des Headers werden nicht verschlüsselt, damit diese für einen Router weiterhin lesbar sind. ESP fügt zu den bisherigen Header-Feldern weitere hinzu, die unter anderem den Kommunikationspartnern die Zuordnung eines Pakets zu einem bestimmten Kontext erlauben. Wenn nicht IPv6 eingesetzt wird, dann sind die zusätzlichen Header-

Felder die letzten vor Beginn der Nutzdaten und werden selbst schon zum Teil verschlüsselt.

Welches Verschlüsselungsverfahren und welche schlüsselabhängige Hashfunktion innerhalb von ESP eingesetzt werden, wird in der ESP-Beschreibung offen gelassen. [RFC2405] beschreibt die Verwendung des DES im CBC-Modus als mögliches Verschlüsselungsverfahren. In [RFC2451] werden Triple-DES, RC5, CAST, IDEA und Blowfish als Alternativen eingeführt. In [RFC4309] kommt schließlich auch der AES zu seinem Recht. Als kryptografische Hashfunktion können gemäß [RFC2403] MD5, gemäß [RFC2404] SHA-1 und gemäß [RFC2857] RIPEMD-160 eingesetzt werden, jeweils in schlüsselabhängiger Form als HMAC. In [RFC4308] werden einige Kombinationen aus Verschlüsselungs- und Hashverfahren genannt, die für den IPsec-Einsatz geeignet sind, wobei gleich auch dazu passende Verfahren für IKE aufgeführt werden. Wenn Alice mit all den genannten Verfahren nicht glücklich wird, dann kann sie auch andere verwenden – sofern Kommunikationspartner Bob diese unterstützt.

ESP kennt zwei Modi. Der erste ist der **Transport-Modus**. In diesem werden die Nutzdaten eines IP-Pakets unverändert verschlüsselt. Das bedeutet, dass diese so verschlüsselt werden, wie sie vorliegen, nachdem sie von TCP oder UDP heruntergereicht und in kleinere Teile aufgeteilt worden sind. Im **Tunnel-Modus** wird dagegen ein gesamtes IP-Paket verschlüsselt und anschließend in den Nutzdatenteil eines weiteren IP-Pakets gepackt. Dadurch ist der Header des ursprünglichen IP-Pakets nicht mehr lesbar, was ein Minimum Disclosure ermöglicht. Alice und Bob können den Tunnel-Modus verwenden, wenn auf einer Teilstrecke des Internets Header-Informationen wie die Sender- oder Empfängeradresse nicht mehr lesbar sein sollen. Der Tunnel-Modus ist vor allem im Zusammenhang mit VPNs interessant (siehe Abschnitt 34.4).

34.1.2 AH

Authentication Header (**AH**) ist neben ESP der zweite Bestandteil von IPsec. Die Beschreibung von AH erfolgt in [RFC4302]. AH sieht ein zusätzliches Header-Feld für die Zuordnung zu einem Kontext vor und fügt ein weiteres Header-Feld hinzu, in dem das Resultat einer schlüsselabhängigen Hashfunktion übertragen wird. Diese schlüsselabhängige Hashfunktion wird auf die Nutzlast und auf einige Header-Felder des IP-Pakets angewendet. Der Sinn von AH ist damit klar: Wenn Alice mit Bob über IP kommuniziert, dann kann Abhörer Mallory keine unbemerkten Änderungen an den IP-Paketen vornehmen. Da AH jedoch keinerlei Verschlüsselung vorsieht, hindert es Mallory nicht am Lesen eines IP-Pakets.

Sendet Bob ein IP-Paket an Alice, dann berechnet er das Resultat der schlüsselabhängigen Hashfunktion und schreibt es in den AH-Header. Empfängerin Alice berechnet den schlüsselabhängigen Hashwert ebenfalls und vergleicht ihn mit dem erhaltenen. Vorausgesetzt, dass nur Alice und Bob den verwendeten

Schlüssel kennen, hat Mallory keine Chance, das Paket unbemerkt zu ändern. Die AH-Spezifikation legt nicht fest, welche kryptografische Hashfunktion hierbei verwendet wird. Die RFCs, die den Einsatz von kryptografischen Hashfunktionen in ESP behandeln, sehen für AH dieselben Verfahren wie für ESP vor. Als kryptografische Hashfunktion können demnach MD5 [RFC2403], SHA-1 [RFC2404] und RIPEMD-160 [RFC2857] eingesetzt werden. Alice und Bob können auch andere Verfahren verwenden, wenn beide diese unterstützen. Prinzipiell kann statt einer schlüsselabhängigen Hashfunktion auch ein digitales Signaturverfahren zum Einsatz kommen. Da digitale Signaturen jedoch bekanntermaßen recht langsam sind und unterhalb der Anwendungsebene kaum einen Sinn haben, ist dies nicht üblich.

Ähnlich wie ESP kann auch AH im Transport- und im Tunnel-Modus eingesetzt werden. Die Funktionsweise ist analog zu der von ESP: Im Transport-Modus wird ein bestehendes IP-Paket erweitert, im Tunnel-Modus wird ein gesamtes IP-Paket in den Nutzdatenteil eines neuen IP-Pakets gepackt.

34.2 IKE

IPsec unterstützt zwar Kontexte und Assoziationen, kann diese jedoch nicht verwalten und ist für sich genommen weder verhandlungsfähig noch zustandsbehaftet. Zudem setzt IPsec voraus, dass Alice und Bob einen gemeinsamen geheimen Schlüssel besitzen, wobei kein Schlüsselaustausch spezifiziert wird. Ein leistungsfähiges Werkzeug wird aus IPsec daher nur im Zusammenspiel mit einem zusätzlichen Protokoll, das sich um die fehlenden Belange kümmert. Das von der IETF hierfür vorgesehene Protokoll hat den Namen **IKE** (**Internet Key Exchange**). Es wird in [RFC5996] beschrieben.

Während bei der Entwicklung von IPsec halbwegs Einigkeit über den zu wählenden Ansatz herrschte, sorgte IKE innerhalb der IETF für heftige Diskussionen. Es gab im Wesentlichen zwei Lager: Die einen wollten IKE als möglichst flexibles und funktionsreiches Schicht-7-Protokoll realisieren. Dazu schlugen sie einen Entwurf vor, der auf drei bereits vorhandenen Protokollen (ISAKMP, Oakley und SKEME) basierte. Den anderen war dieser Ansatz zu kompliziert. Sie bevorzugten das deutlich einfachere **Simple Key Management Protocol** (**SKIP**), das in Schicht 3 des OSI-Modells ansetzt und weniger Funktionalität bietet. Nach langen Diskussionen machte die ISAKMP-Oakley-SKEME-Fraktion das Rennen. So kommt es, dass IKE ein recht komplexes Protokoll ist, das dafür jedoch eine Vielzahl von Möglichkeiten bietet.

34.2.1 ISAKMP

Um IKE zu verstehen, muss man zuerst mit **ISAKMP** vertraut sein [RFC2408]. ISAKMP steht für **Internet Security Association and Key Management Protocol**. Es wurde von der NSA entwickelt. ISAKMP ist genau genommen kein Protokoll, sondern ein Baukasten, mit dem ein kryptografisches Netzwerkprotokoll zusammengebastelt werden kann. Zu diesem Zweck definiert ISAKMP ein Format für Protokollnachrichten, die zur Spezifikation eines Protokolls verwendet werden können. Die Protokollnachrichten von ISAKMP werden stets in Schicht 7 (Anwendungsschicht) des OSI-Modells übertragen, wobei in der darunter liegenden Transportschicht UDP zum Einsatz kommt. Dass ISAKMP im Zusammenhang mit IPsec (Schicht 3) verwendet wird, ist ein Verstoß gegen das Schnittstellenparadigma, was jedoch in Kauf genommen wird.

ISAKMP-Protokollnachrichten bestehen – wie allgemein üblich – aus einem Header und einem Nutzdatenteil. Für den Nutzdatenteil definiert die ISAKMP-Spezifikation insgesamt 13 unterschiedliche Typen. Von einem Hashwert über eine Signatur bis zu einem digitalen Zertifikat können so beliebige kryptografische Daten in einer ISAKMP-Protokollnachricht transportiert werden, wobei jeweils ein anderer Nutzdatentyp eingesetzt wird.

Im Header einer ISAKMP-Protokollnachricht stehen keine Adressen. Um Sender und Empfänger kümmert sich das darunter liegende UDP. Dafür finden sich in jedem ISAKMP-Header zwei Felder für ein Sender-Cookie und ein Empfänger-Cookie. Ein **Cookie** ist in diesem Fall ein 32-Bit-Wert. Bei einer Kommunikation über ISAKMP-Protokollnachrichten sucht sich Alice einen beliebigen Wert als Cookie für Bob aus, umgekehrt wählt Bob ein beliebiges Cookie für Alice. Schickt Alice eine Protokollnachricht an Bob, dann fügt sie das für Bob vorgesehene Cookie als Empfänger-Cookie ein. Hat sie selbst bereits eine ISAKMP-Nachricht von Bob erhalten, dann kennt sie außerdem das Cookie, das Bob ihr zugedacht hat. Dieses fügt sie als Absender-Cookie in die Nachricht ein. Ein Cookie erlaubt es Alice und Bob, eine Nachricht einer bestimmten Sicherheitsassoziation zuzuordnen. Außerdem werden durch Cookies primitive Denial-of-Service-Attacken verhindert: Versucht Mallory, Alice mit falschen ISAKMP-Nachrichten zu überschwemmen, dann kann Alice sofort alle Nachrichten mit falschem Cookie aussortieren, bevor sie irgendwelche zeitraubenden Public-Key-Operationen startet. Allerdings funktioniert dieser Cookie-Trick nur dann, wenn Mallory die Kommunikation von Alice und Bob nicht abhört (und so an die verwendeten Cookies kommt), sondern einfach nur ins Blaue falsche ISAKMP-Nachrichten an Alice schickt.

ISAKMP sieht zwei Phasen vor. In der ersten Phase wird eine Sicherheitsassoziation aufgebaut, wobei in der Regel asymmetrische Verfahren zum Einsatz kommen. Diese Sicherheitsassoziation wird **ISAKMP-SA** genannt. In der zweiten Phase wird die ISAKMP-SA verwendet, um beliebig viele weitere Sicherheitsassoziationen (beispielsweise für IPsec) aufzubauen. Im zweiten Schritt kann dann in

der Regel auf asymmetrische Verfahren verzichtet werden (asymmetrische Verfahren sind bekanntlich recht aufwendig).

Dieser Zwei-Phasen-Ansatz klingt zwar etwas umständlich, hat aber einige Vorteile. So kann die erste Phase etwa von den Betriebssystemen von Alices und Bobs Rechnern übernommen werden. In der zweiten Phase können Alice und Bob dann schnell und ohne den Einsatz aufwendiger asymmetrischer Verfahren Kommunikationsverbindungen aufbauen und schließen.

34.2.2 Wie IKE ISAKMP nutzt

Der Zusammenhang zwischen IKE und ISAKMP ist leicht zu erklären: IKE ist ein Protokoll, das vollständig aus ISAKMP-Protokollnachrichten aufgebaut ist. Auch der Zwei-Phasen-Ansatz von ISAKMP wird von IKE übernommen: In der ersten Phase wird eine ISAKMP-SA aufgebaut (diese wird in diesem Zusammenhang **IKE-SA** genannt), in der zweiten Phase werden auf deren Basis eine oder mehrere Sicherheitsassoziationen für IPsec **IPsec-SAs**) erstellt. Der genaue Ablauf von IKE wurde von den zwei bereits erwähnten Protokollen Oakley und SKEME abgeschaut, auf die ich nicht näher eingehen will. Ein wesentlicher Bestandteil von IKE ist der Diffie-Hellman-Schlüsselaustausch – dieser wird verwendet, um für Alice und Bob einen gemeinsamen geheimen Schlüssel zu generieren (aus diesem werden zudem weitere geheime Schlüssel abgeleitet). IKE kennt insgesamt fünf Diffie-Hellman-Varianten: Alice und Bob können zum einen das klassische Verfahren mit 768, 1.024 oder 1.680 Bit Schlüssellänge einsetzen, zum anderen sind zwei ECDH-Varianten mit 155 bzw. 185 Schlüssel-Bits möglich.

IKE Phase 1

In Phase 1 des IKE-Protokolls tauschen Alice und Bob Cookies aus, handeln Verfahren und Parameter aus, authentifizieren sich gegenseitig, führen einen Diffie-Hellman-Schlüsselaustausch durch und bauen auf dieser Basis eine IKE-SA auf. Zu diesem Zweck bietet IKE zwei Varianten. Die erste wird als **Main Mode** bezeichnet. Der Main Mode sieht vor, dass Alice und Bob abwechselnd insgesamt sechs Protokollnachrichten an den jeweils anderen verschicken:

- In den ersten beiden Nachrichten werden die Cookies übergeben und die zu verwendenden Verfahren und Parameter ausgehandelt.
- Die Nachrichten drei und vier dienen der Durchführung eines Diffie-Hellman-Schlüsselaustauschs.
- Mit der fünften und sechsten Nachricht werden die Authentizität und Integrität der zuvor übermittelten öffentlichen Diffie-Hellman-Schlüssel sichergestellt. Wenn Alice und Bob bereits einen gemeinsamen geheimen Schlüssel besitzen, dann erfolgt die Authentifizierung mithilfe einer schlüsselabhängigen Hashfunktion, die auf die Diffie-Hellman-Schlüssel angewendet wird. Ist

kein gemeinsamer geheimer Schlüssel vorhanden, dann können die beiden statt eines Hashwerts jeweils auch eine digitale Signatur schicken, die sich auf den jeweils eigenen öffentlichen Diffie-Hellman-Schlüssel bezieht. Mit der Signatur kann auch ein digitales Zertifikat verschickt werden, das eine Überprüfung der Signatur erlaubt. Eine weitere Möglichkeit zur Authentifizierung ist ein spezielles Challenge-Response-Verfahren. Wenn Alice und Bob dieses nutzen, senden sie sich bereits in Nachricht drei und vier je einen Zufallswert (**Nonce**) zu, den sie mit einem öffentlichen Schlüssel des jeweils anderen verschlüsseln. Dieser öffentliche Schlüssel ist in der Regel ein RSA-Schlüssel und kann Bestandteil eines digitalen Zertifikats sein, das von einem Zertifikateserver bezogen werden kann. In der fünften und sechsten Nachricht senden sich die beiden gegenseitig den Hashwert des erhaltenen Nonce zu.

Natürlich wäre es auch möglich, gleich den Diffie-Hellman-Schlüssel in Form eines Zertifikats zu verschicken – dies wird jedoch nicht gemacht, um Forward Security und Backward Security zu erreichen (siehe Abschnitt 20.3.5).

Neben dem Main Mode gibt es für Phase 1 von IKE noch den **Aggressive Mode**, der etwas einfacher, aber weniger leistungsfähig ist. Im Aggressive Mode sendet Alice in der ersten Nachricht an Bob ein Cookie, die von ihr vorgeschlagenen Verfahren und Parameter, gegebenenfalls das verschlüsselte Nonce sowie ihren öffentlichen Diffie-Hellman-Schlüssel. Bob anwortet darauf mit einer Nachricht, die sein für Alice gedachtes Cookie, die von ihm akzeptierten Verfahren, seinen öffentlichen Diffie-Hellman-Schlüssel sowie Hashwert bzw. Signatur enthält. Anschließend verschickt Alice eine dritte und letzte Phase-1-Nachricht an Bob, in der ihr Hashwert bzw. ihre Signatur enthalten sind.

Am Ende von Phase 1 besitzen Alice und Bob eine Sicherheitsassoziation und einen gemeinsamen geheimen Schlüssel, den sie als Basis für eine IPsec-Kommunikation verwenden können. Durch die Authentifizierung wissen sie außerdem, dass sie es wirklich miteinander zu tun haben.

IKE Phase 2

Mit der in Phase 1 aufgebauten ISAKMP-SA und den dabei entstandenen gemeinsamen geheimen Schlüsseln können Alice und Bob in der zweiten Phase weitere Sicherheitsassoziationen (IPsec-SAs) aufbauen und diese für IPsec nutzen. In Ergänzung zu den beiden Modi aus Phase 1 (Main und Aggressive Mode) sieht die IKE-Spezifikation hierzu einen weiteren Modus namens **Quick Mode** vor. Im Quick Mode sendet Alice zunächst eine Protokollnachricht an Bob, erhält von diesem eine zurück und sendet schließlich eine weitere an Bob – es sind also drei Nachrichten vorgesehen.

Im Quick Mode können Alice und Bob auf eine aufwendige Authentifizierung und einen Schlüsselaustausch mit asymmetrischen Verfahren verzichten, da dies bereits in Phase 1 durchgeführt wurde. Wenn allerdings an dieser Stelle For-

ward Security und Backward Security gewünscht ist, dann kann auch ein erneuter Diffie-Hellman-Schlüsselaustausch durchgeführt werden.

Weitere IKE-Protokollnachrichten

Neben den Protokollnachrichten von Main Mode, Aggressive Mode und Quick Mode gibt es in IKE noch zwei weitere Typen von Protokollnachrichten. Diese dienen nicht der Generierung von Sicherheitsassoziation. Mit dem einen der beiden Typen ist das Versenden von Fehler- und Statusmeldungen möglich. Mit dem anderen können Alice und Bob neue Parameter für den Diffie-Hellman-Schlüsselaustausch aushandeln.

34.3 Kritik an IPsec

IPsec (und das dazugehörende IKE) sind nicht gerade unumstritten. Eine vernichtende Kritik kam von den Kryptografen Bruce Schneier und Niels Ferguson [FeSc00]. Sie schrieben: »Für uns war IPsec eine große Enttäuschung. In Anbetracht der Qualität der Leute, die daran gearbeitet haben, und der Zeit, die dafür aufgebracht wurde, haben wir ein viel besseres Ergebnis erwartet.« Der Hauptvorwurf von Schneier und Ferguson: IPsec und IKE sind viel zu komplex. Einige der von ihnen genannten Kritikpunkte sind zwar inzwischen durch neuere IPsec-Versionen und eine verbesserte Dokumentation behoben, doch von ihrem grundsätzlichen Standpunkt sind die beiden nicht abgerückt.

Ein Dorn im Auge sind den beiden Kryptografen insbesondere die vielen Varianten, Modi und Typen, die IPsec vorsieht. Es sei praktisch nicht möglich, IPsec und IKE ausreichend fehlerfrei zu implementieren, lautet der Standpunkt von Schneier und Ferguson. Nicht ganz zu Unrecht monieren sie zudem, dass IPsec in mehreren Varianten betrieben werden kann, die sich untereinander in der Funktionalität nicht wesentlich unterscheiden. In der Tat schließt der Funktionsumfang von ESP, das neben Verschlüsselung auch schlüsselabhängige Hashwerte unterstützt, die Funktionalität von AH ein. Auch den Transport-Modus sehen die beiden Autoren als überflüssig an, da der Tunnel-Modus ihrer Meinung nach ausreicht. Um an Komplexität einzusparen, schlagen Schneier und Ferguson vor, auf AH und auf den Transport-Modus zu verzichten. Für eine weitere Vereinfachung soll zudem nur noch die Verschlüsselung optional sein, während ein schlüsselabhängiger Hashwert stets obligatorisch eingesetzt werden soll.

Die Schuld an der besagten Komplexitätsmisere sehen Schneier und Ferguson jedoch nicht bei den Entwicklern der beiden Standards, sondern im Entstehungsprozess. Da IPsec und IKE von einem Gremium entwickelt wurde, in dem Personen mit ganz unterschiedlichen Interessen saßen, entstand am Ende ein kaum noch überschaubares Sammelsurium von Kompromissen und Varianten. Schneier und Ferguson schlagen daher vor, dass wichtige Standards nicht mehr von Gremien, sondern in Form eines Wettbewerbs (wie beim AES) entwickelt werden.

Die beiden betonen in ihrem Dokument jedoch, dass es derzeit keine Alternative zu IPsec gibt. Wir werden also damit leben müssen.

34.4 Virtuelle Private Netze mit IPsec

In Abschnitt 33.1.4 haben wir uns mit Virtuellen Privaten Netzen (VPNs) beschäftigt. Dabei ging es um das Tunneln eines Schicht-2-Protokolls (meist PPP) über ein Internetprotokoll der Anwendungsschicht (PPTP oder L2TP). Durch ein solches Tunneln kann ein lokales Netz (Intranet) über das Internet mit anderen Netzen oder Einzelrechnern verbunden werden, wobei dennoch eine strikte Trennung zwischen Intranet und Internet bestehen bleibt.

Mit IPsec lässt sich ein ähnlicher Effekt erreichen wie bei der in Abschnitt 33.1.4 beschriebenen Vorgehensweise, wobei jedoch andere OSI-Schichten für den Tunnel verwendet werden. Aus diesem Grund lassen sich VPNs auch mit IPsec realisieren, ohne dabei PPTP oder L2TP einzusetzen. Wie die Realisierung eines VPNs mit IPsec funktioniert, sehen Sie in Abbildung 34–2. Als Tunnelprotokoll wird dabei IPsec im Tunnel-Modus genutzt. Dieses kommt zwischen den beiden verbundenen Netzen (bei einer LAN-Kopplung) oder zwischen Einzelrechner und Netz (bei der Anbindung von Einzelrechnern) zum Einsatz. Das Protokoll, das getunnelt wird, ist in diesem Fall kein Schicht-2-Protokoll, sondern IP. Das Intranet und das Internet sind dabei – wie bei einem VPN üblich – logisch voneinander getrennt.

Ein Vorteil bei der Verwendung von IPsec als Tunnelprotokoll besteht darin, dass dieses Verschlüsselung und Authentifizierung von Hause aus unterstützt. Nicht zuletzt deshalb hat IPsec die beiden Alternativen PPTP und L2TP als Tunnelprotokoll längst überholt.

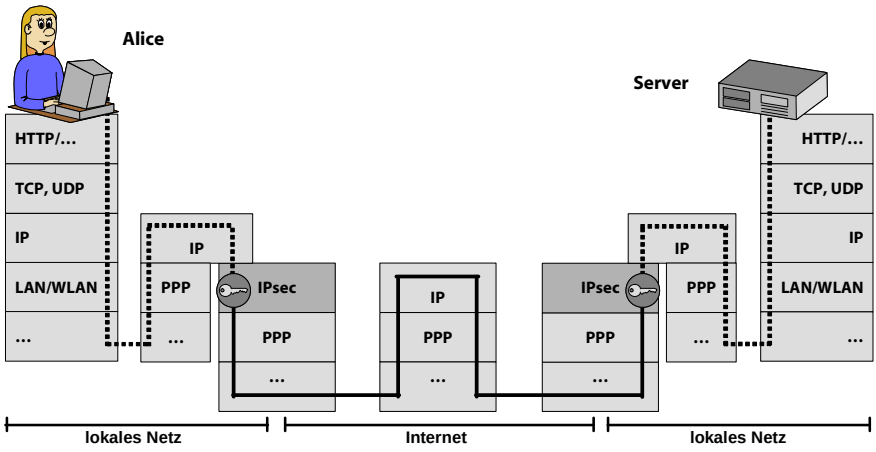


Abb. 34–2 *IPsec kann auch als Tunnelprotokoll für ein VPN verwendet werden. Dabei werden IP-Pakete innerhalb von IP-Paketen übertragen.*

35 SSL und TLS (Schicht 4)

Nachdem wir im letzten Kapitel das Schicht-3-Protokoll IPsec betrachtet haben, klettern wir nun im OSI-Modell um eine Etage nach oben. In Schicht 4 (Transportschicht) gibt es im Internet zwei Protokolle: das verbindungsorientierte TCP und das verbindungslose UDP. Wie alle frühen Internetprotokolle enthalten diese beiden in ihrer ursprünglichen Version keine Kryptografie. Mit dem kryptografischen Netzwerkprotokoll **SSL** (**Secure Socket Layer**) ist es jedoch möglich, diesen Mangel für TCP zu beheben. Für UDP hat dagegen meines Wissens bisher niemand eine Krypto-Erweiterung entwickelt. Dies ist wohl auch nicht notwendig, da UDP im Wesentlichen IP-Pakete unverändert durchreicht, und für diese gibt es ja IPsec.

Während IPsec in einem langwierigen Prozess innerhalb der IETF aus mehreren bereits existierenden Bestandteilen zusammengebastelt wurde, ist SSL ein eher pragmatisches Konstrukt, das von einigen Mitarbeitern der Firma Netscape entwickelt wurde. SSL ist dadurch weit weniger komplex als IPsec. Es war zudem

früher in Produkten verfügbar und hat sich am Markt schneller durchgesetzt. SSL wurde 1997 von der IETF unter dem Namen **TLS** (**Transport Layer Security**) standardisiert – an die Stelle von SSL 3.1 trat TLS 1.0. Inzwischen ist TLS in der Version 1.2 verfügbar [RFC5246], doch nach wie vor wird häufig die Bezeichnung SSL verwendet. Dies werde ich im Folgenden übernehmen.

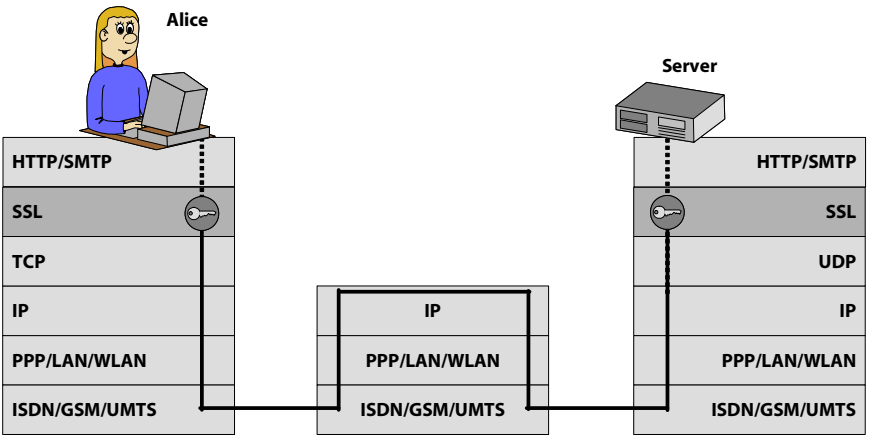


Abb. 35–1 *SSL setzt zwischen TCP und einem Anwendungsprotokoll wie HTTP oder FTP an. Das Anwendungsprotokoll und die Anwendungssoftware müssen nicht geändert werden.*

35.1 Funktionsweise von SSL

SSL realisiert, ähnlich wie IPsec, einen sicheren Kanal. Während IPsec dazu Änderungen an IP-Paketen vornimmt, greift SSL nicht direkt in die TCP-Kommunikation ein. Stattdessen fügt SSL eine zusätzliche Schicht zwischen TCP und der darüber liegenden Anwendung ein (Abbildung 35–1). Alle Krypto-Operationen finden in dieser Schicht statt, ohne dass das darunter liegende TCP etwas davon mitbekommt. Auch die darüber liegende Anwendungsschicht bleibt (weitgehend) unbehelligt. Obwohl SSL eine eigene Schicht überhalb von TCP bildet, wird es in der Regel immer noch zu Schicht 4 (Transportschicht) des OSI-Modells gezählt.

SSL kann, wie alle Protokolle, die unterhalb der Anwendungsschicht angesiedelt sind, von der jeweiligen Anwendung unabhängig betrieben werden. Durch diese Flexibilität hat sich SSL in unterschiedlichen Anwendungsszenarien durchgesetzt. Am bekanntesten ist der Einsatz unterhalb von HTTP. SSL kann jedoch auch zur kryptografischen Absicherung von FTP, Telnet, SAP R/3, X-Windows, POP, IMAP, IIOP, WAP oder Lotus Notes verwendet werden. Der Name »Secure Socket Layer« leitet sich von der Schnittstelle zwischen TCP und der darüber liegenden Anwendung ab, die **Socket** genannt wird. SSL setzt auf den Socket eine weitere Schicht und emuliert diesen. Für die Anwendung ändert sich dabei nichts, wodurch das Schnittstellen-Paradigma nicht verletzt wird.

35.1.1 Protokolleigenschaften

SSL ist ein verbindungsorientiertes Protokoll und arbeitet nur mit dem ebenfalls verbindungsorientierten TCP zusammen, nicht aber mit dem verbindungslosen UDP. SSL ist ein zustandsbehaftetes Protokoll, es gibt also Kontexte und Sicherheitsassoziationen. SSL legt sich – wie die meisten anderen kryptografischen Netzwerkprotokolle – nicht auf bestimmte Krypto-Verfahren fest, ist also verhandlungsfähig. Die aktuelle Spezifikation erlaubt beispielsweise die symmetrischen Verfahren RC2, RC4, Triple-DES, und AES. RFC 4132 beschreibt zudem den Einsatz von Camellia innerhalb von SSL [RFC4132]. Als schlüsselabhängige Hashfunktion ist HMAC basierend auf SHA-256 spezifiziert, Alternativen sind zulässig.

Für den Schlüsselaustausch unterstützt SSL die Verfahren RSA und Diffie-Hellman, als Signaturverfahren zur Authentifizierung kann RSA oder DSA eingesetzt werden. In RFC 4492 wird der Einsatz von ECC-Verfahren innerhalb von SSL beschrieben [RFC4492]. An dieser Stelle sei noch einmal erwähnt, dass digitale Signaturen in einem Protokoll, das wie SSL unterhalb der Anwendungsebene ansetzt, nur zur Authentifizierung sinnvoll sind. Die von SSL verwendeten Zertifikate entsprechen in der Regel (aber nicht notwendigerweise) dem X.509v3-Standard.

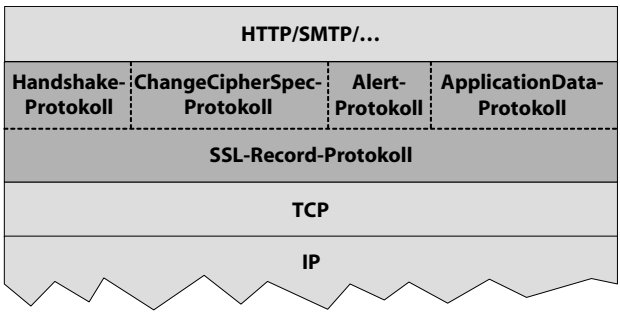


Abb. 35–2 *SSL gliedert sich in zwei Teilschichten. Die obere ist für das Aushandeln von Verfahren und Fehlermeldungen zuständig. Die untere Schicht übernimmt die Verschlüsselung.*

35.1.2 SSL-Teilprotokolle

Der Aufbau von SSL sieht fünf Teilprotokolle vor, die zwei übereinander liegende Teilschichten bilden (Abbildung 35–2). In der unteren Schicht wird das **Record-Protokoll** abgearbeitet, in der oberen Schicht die vier anderen Protokolle. Das Record-Protokoll hat die Aufgabe, Daten von oben über den Socket entgegenzunehmen und kryptografische Verfahren darauf anzuwenden – es realisiert Datenaustauschroutine des sicheren Kanals. Dazu können die erwähnten Verschlüsselungsverfahren und schlüsselabhängigen Hashfunktionen zum Einsatz kommen. Die beiden Verfahren, die gerade im Einsatz sind, werden aktuelles Verschlüsse-

lungsverfahren und aktuelle Hashfunktion genannt. Am Anfang einer Kommunikation sind beide aktuelle Verfahren auf Null gesetzt, was bedeutet, dass weder eine Hashfunktion angewendet noch verschlüsselt wird. Die obere Teilschicht kann nun veranlassen, dass sich dies ändert. So kann beispielsweise der AES zum aktuellen Verschlüsselungsverfahren werden und SHA-256 zur aktuellen Hashfunktion. Hierbei ist es möglich, dass eines oder beide der aktuellen Verfahren auf Null gesetzt bleibt.

35.2 SSL-Protokollablauf

SSL liegt die Vorstellung zugrunde, dass ein Client (zum Beispiel ein Webbrowser) mit einem Server (zum Beispiel ein Webserver) kommuniziert. Es gibt mehrere Teilprotokolle.

35.2.1 Das Handshake-Protokoll

Das erste SSL-Teilprotokoll ist das *Handshake-Protokoll*. Es wird auf der oberen SSL-Schicht abgearbeitet und dient dazu, Client (in unserem Fall Alice) und Server zu authentifizieren, die anzuwendenden Krypto-Verfahren zu vereinbaren und einen symmetrischen Schlüssel auszutauschen (aus diesem einen Schlüssel können in der oberen SSL-Schicht zwei berechnet werden, die zur Verschlüsselung und für die Hashfunktion verwendet werden). Das Handshake-Protokoll übernimmt somit (zusammen mit dem ChangeCipherSpec-Protokoll) die Aufgabe der Initialisierungsroutine des sicheren Kanals. Der genaue Ablauf ist von verschiedenen Optionen abhängig. Typischerweise läuft das Handshake-Protokoll folgendermaßen ab:

1. Alice sendet eine Nachricht an den Server, die diesem anzeigt, dass sie SSL einsetzen will (Client-Hello). In dieser Nachricht nennt Alice zudem die Krypto-Verfahren, die sie unterstützt. Dabei muss mindestens je ein symmetrisches Verschlüsselungsverfahren, eine kryptografische Hashfunktion und ein Schlüsselaustauschverfahren genannt werden.
2. Der Server schickt eine Nachricht zurück (Server-Hello). Darin gibt er an, welche Verfahren er sich aus Alices Vorschlägen ausgesucht hat. Außerdem enthält die Nachricht das digitale Zertifikat des Servers.
3. Alice schickt nun ihr eigenes digitales Zertifikat und einen Sitzungsschlüssel, der mit dem öffentlichem Schlüssel des Servers verschlüsselt ist. Dadurch haben Alice und Bob einen gemeinsamen geheimen Schlüssel, den sie in der unteren SSL-Schicht zur Verschlüsselung und zur Integritätskontrolle mit einer schlüsselabhängigen Hashfunktion einsetzen können.

Der geschilderte Ablauf ist nur ein Beispiel, denn die Kommunikation kann auch anders verlaufen. Wenn etwa eine unterbrochene Kommunikation wieder aufgenommen wird, dann entfällt der Schlüsselaustausch. Wenn zur Authentifizierung

digitale Signaturen eingesetzt werden sollen, dann kommen zusätzliche Nachrichten hinzu. Es ist außerdem möglich, dass nur der Server ein Zertifikat besitzt oder dass beide Seiten auf ein Zertifikat verzichten und Preshared Keys verwendet werden [RFC4279]. Eine weitere Möglichkeit ist der Einsatz von Kerberos, was in [RFC2712] beschrieben wird.

35.2.2 Das ChangeCipherSpec-Protokoll

Nach Abarbeiten des Handshake-Protokolls kommt das *ChangeCipherSpec-Protokoll* zum Einsatz, um die vereinbarten Verfahren der Record-Schicht mitzuteilen. Dadurch werden diese zu den aktuellen Verfahren. Dieses Teilprotokoll wird zudem immer dann abgearbeitet, wenn an den aktuellen Verfahren etwas geändert werden soll. Erst wenn das ChangeCipherSpec-Protokoll – das nur aus einer Nachricht besteht – beendet ist, ändert das Record-Protokoll die bis dahin verwendeten Verfahren.

35.2.3 Das Alert-Protokoll

Ein weiteres SSL-Teilprotokoll, das in der oberen Schicht abgearbeitet wird, ist das *Alert-Protokoll*. Dieses wird nur dann gebraucht, wenn der Datenaustausch nicht planmäßig läuft. Seine Aufgabe ist das Versenden von Fehlermeldungen und Warnungen.

35.2.4 Das ApplicationData-Protokoll

Das vierte und letzte Teilprotokoll der oberen SSL-Schicht ist das *ApplicationData-Protokoll*. Dieses ist für das Durchreichen der von oben kommenden Anwendungsdaten an das Record-Protokoll zuständig. Damit ist das ApplicationData-Protokoll dasjenige Protokoll, das im Regelfall in der oberen Schicht im Einsatz ist. Die anderen Protokolle werden nur zur Initialisierung, zum Verfahrenswechsel und bei Störungen abgearbeitet und bilden daher den Ausnahmefall. Das ApplicationData-Protokoll kann auch abhängig von der darüber liegenden Anwendung realisiert werden. Dies ist zwar ein Verstoß gegen das Schnittstellenparadigma, erlaubt jedoch, dass an dieser Stelle spezielle Anforderungen des darüber liegenden Protokolls beachtet werden.

35.3 SSL in der Praxis

SSL dürfte das gegenwärtig meistverwendete Verschlüsselungsprotokoll im Internet sein. Wie es scheint, ist die Lage zwischen Anwendungsschicht und TCP für viele Fälle ideal: SSL lässt sich ohne (oder mit geringen) Anwendungsänderungen in bestehende Strukturen integrieren. Gleichzeitig ist es jedoch so nahe an der Anwendung, dass es (mit einer kleinen Verletzung des Schnittstellenparadigmas)

noch von der Anwendung beeinflussbar ist. Von Nachteil ist jedoch, dass SSL nur über TCP, nicht aber über UDP einsetzbar ist. Dadurch lassen sich Dienste wie TFTP, DNS und insbesondere Internettelefonie nicht mit SSL absichern.

Wären in den Protokollen der TCP/IP-Familie Sicherheitsvorkehrungen von Anfang an vorgesehen gewesen, dann hätte sich ein Protokoll wie SSL, das eine eigene Schicht für sich in Anspruch nimmt, sicherlich nicht durchgesetzt. Durch die Versäumnisse früherer Internettage hat sich SSL jedoch zum Erfolgsmodell unter den Krypto-Protokollen entwickelt.

35.3.1 Vergleich zwischen IPsec und SSL

IPsec und SSL sind sich in ihrer Wirkungsweise auf den ersten Blick recht ähnlich. Beide bieten einen sicheren Kanal unterhalb der Anwendungsebene zwischen zwei ans Internet angeschlossenen Rechnern. Dennoch gibt es einige Unterschiede:

- SSL lässt nur eine Ende-zu-Ende-Verbindung zu. IPsec kann dagegen auch Teilstrecken zwischen zwei Routern verschlüsseln.
- SSL ist näher an der Anwendung und kann die verwendeten Verfahren auf die übertragenen Daten abstimmen. IPsec kann dagegen nur durch gravierende Verletzungen des Schichtenparadigmas auf die entsprechende Anwendung eingehen.
- SSL arbeitet nicht mit UDP-basierten Schicht-7-Protokollen wie TFTP, DNS, L2TP und PPTP zusammen. Auch Internettelefonie wird mit UDP realisiert und kann daher nicht mit SSL abgesichert werden.
- SSL lässt sich leichter in bestehende Netze integrieren, da es zwischen zwei Schichten ansetzt. IPsec erfordert jedoch die Änderung bestehender Protokolle.
- IPsec verwendet zum Schlüsselmanagement ein Schicht-7-Protokoll (IKE). Bei SSL spielt sich dagegen alles in einer Schicht (oder genauer: in den zwei Teilschichten) ab.
- IPsec ist aus mehreren Bestandteilen aufgebaut (ESP, AH, IKE), die teilweise einen unterschiedlichen Ursprung haben und innerhalb der IETF zusammengefügt wurden. SSL ist im Vergleich dazu ein homogeneres Protokoll, das von einem Unternehmen (Netscape) entwickelt wurde. Durch die unterschiedliche Entstehung ist es auch zu erklären, dass IPsec komplexer, dafür aber auch leistungsfähiger ist. Dieser Umstand hat auch dazu geführt, dass die Standardisierung von SSL schneller und unspektakulärer ablief als bei IPsec.

Eine häufig gestellte Frage ist, ob man eine sicherheitskritische Client-Server-Lösung, die über das Internet zugänglich sein soll, als Webportal mit SSL oder als VPN-Lösung mit IPsec realisieren soll. Eine solche Fragestellung taucht etwa dann auf, wenn eine Versicherung ihren Vertretern eine Online-Datenbank mit Preis- und Kundeninformationen zur Verfügung stellen will. Die Anwort lautet meist: Im heterogenen Umfeld ist SSL zu bevorzugen, während in homogenen

Strukturen IPsec die bessere Wahl ist. Etwas präziser ausgedrückt lässt sich dies so formulieren:

- Wenn die Vertreter der Versicherung Angestellte sind, die alle mit der gleichen PC-Konfiguration arbeiten, dann kann die IT-Abteilung des Unternehmens auf jedem der Rechner einen VPN-Client installieren, der sich zentral administrieren lässt und der über das VPN unterschiedliche Anwendungen zulässt. IPsec ist daher die Methode der Wahl.
- Wenn die Vertreter der Versicherung dagegen externe Vertragspartner mit eigener IT-Infrastruktur sind, dann kann die IT-Abteilung kaum die Installation eines VPN-Clients auf deren PC verordnen. Ein solches Unterfangen wäre ohnehin schwierig, da bei den Vertretern in diesem Fall unterschiedliche Betriebssysteme und Konfigurationen anzutreffen sind. Ein Webbrowser mit SSL-Unterstützung ist jedoch auf nahezu jedem PC verfügbar. Für die Versicherung empfiehlt es sich in diesem Szenario daher, ihre Vertreter-Applikation über ein Webportal anzubieten.

Vor diesem Hintergrund ist sicherlich verständlich, warum beispielsweise Online-Banking, Versicherungskundenportale und ähnliche Serversysteme fast immer über SSL angebunden werden, während Unternehmensfilialen und die Heimarbeitsplätze von Mitarbeitern ein Fall für IPsec oder eine andere VPN-Technologie sind.

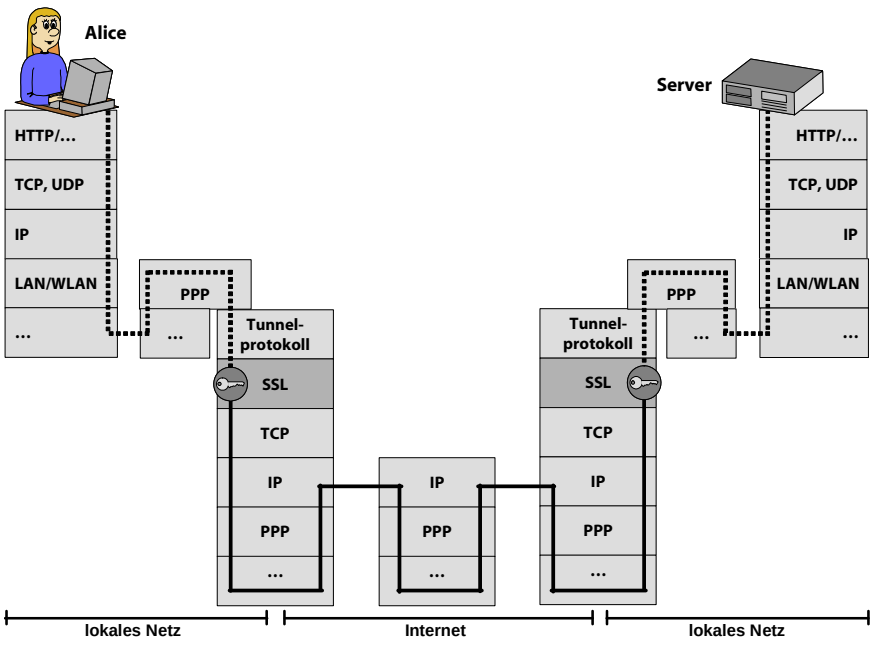


Abb. 35–3 *Mit SSL lässt sich ein VPN realisieren.*

35.3.2 VPNs mit SSL

Das SSL-Protokoll lässt sich auch zur Absicherung eines VPN nutzen. **SSL-VPNs** sind in den letzten Jahren sogar recht populär geworden. Leider wird dieser Begriff in mehreren unterschiedlichen Bedeutungen verwendet, wobei nicht alle Verwendungen wirklich sinnvoll sind. In diesem Buch will ich ein SSL-VPN als Variante eines VPNs in Schicht 2 (siehe Abschnitt 33.1.4) definieren, in dem SSL zur Absicherung des Tunnelprotokolls eingesetzt wird. Wie ein solches Konstrukt aussieht, ist in Abbildung 35–3 ersichtlich. Wie Sie sehen, unterscheidet sich ein SSL-VPN auf den ersten Blick nicht wesentlich von einem VPN, das mit PPTP oder L2TP realisiert wird. Das Problem hierbei ist jedoch, dass die beiden genannten Tunnelprotokolle UDP als Schicht-4-Protokoll nutzen und daher nicht mit SSL abgesichert werden können. Anbieter von SSL-VPNs nach der beschriebenen Definition setzen daher bisher proprietäre Tunnelprotokolle ein, die TCP verwenden und daher auch einen SSL-Einsatz ermöglichen. In den nächsten Jahren dürfte es einen Standard für diesen Zweck geben.

Ein SSL-VPN hat gegenüber einem IPsec-VPN vor allem netzwerktechnische Vorteile. So gibt es bei IPsec oft Probleme mit der Network Address Translation (NAT), während SSL, das keine IP-Adressen kennt, davon nicht betroffen ist. SSL-VPNs werden zudem von Firewalls seltener blockiert, da sie den Port von HTTPS (also von HTTP über SSL) nutzen, der meist freigeschaltet ist. Diese Vorteile machen SSL-VPNs vor allem für Client-Anbindungen interessant, während die LAN-Kopplung nur selten auf diese Weise abgesichert wird.

36 E-Mail-Verschlüsselung und -Signierung (Schicht 7)

Von den zahlreichen Kommunikationstechnologien, die sich in Schicht 7 (Anwendungsschicht) des OSI-Modells einordnen lassen, ist E-Mail die populärste. Selbst das World Wide Web – ebenfalls in Schicht 7 angesiedelt – kann da nicht mithalten. Gerade E-Mail ist jedoch auch ein äußerst interessanter Dienst für einen Bösewicht wie Mallory. Denn zum einen ist eine E-Mail – beispielsweise auf einem Mailserver – vergleichsweise einfach für ungebetene Mitleser zugänglich. Zum anderen werden per E-Mail oft sensible Daten verschickt. Trotz dieser Gefahren sahen die ersten E-Mail-Systeme weder Verschlüsselung noch digitale Signaturen vor. Dieses Problem wurde jedoch früh erkannt, und so war E-Mail der erste Internetdienst überhaupt, in den kryptografische Mechanismen integriert wurden. In den neunziger Jahren setzte sich S/MIME als E-Mail-Krypto-Format durch.

36.1 E-Mail und Kryptografie

Es gibt zwar verschiedene Techniken zur Übertragung von E-Mails, doch die mit Abstand größte Verbreitung hat Internet Mail. Dieses verwendet, wie der Name ausdrückt, das Internet als Übertragungsmedium und ist am @-Zeichen in der E-Mail-Adresse zu erkennen. Das Format einer Internet-Mail-Nachricht wird in [RFC822] beschrieben. Wenn Alice eine E-Mail an Bob sendet, dann kann man dies als einfaches, zustandsloses Protokoll betrachten, bei dem die E-Mail die einzige Protokollnachricht darstellt. Diese Protokollnachricht besteht – wie üblich – aus einem Header und aus Nutzdaten. Der Header wiederum besteht aus mehreren Feldern, beispielsweise einem für die E-Mail-Adresse des Absenders und einem für die E-Mail-Adresse des Empfängers. Zur Übertragung von E-Mail-Nachrichten (beispielsweise von einem Mailserver zum nächsten) sieht Internet Mail hauptsächlich das Protokoll SMTP (Simple Mail Transfer Protocol) vor, das in Schicht 7 des OSI-Modells eingeordnet wird. SMTP wird auch vom Mail-Client verwendet, um eine E-Mail vom Client an einen Mailserver zu übertragen. In umgekehrter Richtung kommen dagegen meist POP oder IMAP zum Einsatz.

Das in RFC 822 beschriebene E-Mail-Format ist nur für ASCII-Text geeignet und sieht keine Strukturierung des E-Mail-Inhalts vor. Dieser Makel wurde durch den MIME-Standard (Multipurpose Internet Mail Extensions) behoben, der in [RFC1521] spezifiziert ist. MIME ermöglicht die Aufteilung einer E-Mail in beliebig viele Blöcke. Jeder Block erhält dabei einen eigenen Header, in dem angegeben werden kann, welchem Typ die Daten darin entsprechen (Inhaltstyp). Beispiele für Inhaltstypen sind ASCII-Text und Binärdaten. Der Header kann außerdem zusätzliche Felder (Attribute) beinhalten, in denen weitere Eigenschaften der Daten im zugehörigen Block angegeben werden können.

36.1.1 Kryptografie für E-Mails

Will Alice eine E-Mail an Bob verschlüsseln, dann sieht dies wie folgt aus (wir nehmen an, dass RSA zum Schlüsselaustausch und der AES zur Verschlüsselung zum Einsatz kommen):

1. Alice besorgt sich den öffentlichen Schlüssel von Bob und generiert per Zufallsgenerator einen AES-Sitzungsschlüssel. Mit diesem verschlüsselt sie die Mail. Außerdem verschlüsselt sie den Sitzungsschlüssel per RSA mit Bobs öffentlichem Schlüssel (es handelt sich also um ein Hybridverfahren). Zum Schluss schickt Alice den verschlüsselten Text zusammen mit dem verschlüsselten Schlüssel an Bob.
2. Bob entschlüsselt den Sitzungsschlüssel und damit anschließend die Mail.

Anstatt RSA können Alice und Bob auch Diffie-Hellman oder ein ECC-Verfahren verwenden. Möglich ist außerdem der Einsatz von MQV (One-Pass-Variante) für den Schlüsselaustausch. Allerdings hat sich MQV für E-Mails bisher nicht durch-

gesetzt. Forward Security und Backward Security bezüglich der privaten Schlüssel bietet ohnehin keiner der gängigen E-Mail-Krypto-Standards, obwohl dies in den typischen Einsatzszenarien – es kann durchaus vorkommen, dass Mallory Alice den privaten Schlüssel von der Festplatte klaut – durchaus sinnvoll wäre. Neben der Verschlüsselung sind digitale Signaturen für E-Mails eine sinnvolle Sache. Es kann schließlich vorkommen, dass Mallory eine Mail verändert oder dass Absenderin Alice im Nachhinein abstreitet, eine bestimmte Mail gesendet zu haben. Ist ihre Nachricht signiert, dann funktionieren diese Angriffe nicht. Digitale Signaturen gehören daher zum Standardrepertoire einer E-Mail-Krypto-Lösung. Sinnvollerweise sollte Alice ihre E-Mail zunächst signieren und dann erst verschlüsseln.

Clientbasierte E-Mail-Absicherung

Wenn Alice ihre E-Mails an Bob verschlüsseln oder signieren will, dann kann sie das mithilfe eines geeigneten E-Mail-Clients tun. Microsoft Outlook, Lotus Notes, Novell Groupwise und viele andere bieten verschiedene Krypto-Funktionen, mit denen das möglich ist. Ist Alice mit den eingebauten Krypto-Mechanismen ihres E-Mail-Clients dagegen nicht zufrieden (etwa, weil Features wie Gruppenverschlüsselung, Smartcard-Anbindung oder ECC-Unterstützung fehlen), dann kann sie ein Krypto-Plugin von einem Drittanbieter verwenden. Ein solches klinkt sich über eine geeignete Schnittstelle in die E-Mail-Software ein und bietet auf diese Weise die gewünschten Krypto-Funktionen an.

Serverbasierte E-Mail-Absicherung

Ursprünglich galt es als Selbstverständlichkeit, das Verschlüsseln und Signieren von E-Mails auf dem PC des Anwenders vorzunehmen. In den letzten Jahren wurde jedoch ein neuer Ansatz für den kryptografischen Schutz von E-Mails immer populärer: die **serverbasierte E-Mail-Absicherung**. Eine ausführliche Betrachtung dieses Themas enthält [Schm08/4]. Um die serverbasierte E-Mail-Absicherung zu erklären, nehmen wir an, die Firma Krypt & Co. betreibe einen Mailserver für alle Angestellten. Alle E-Mails, die das Unternehmen verlassen oder an eine Adresse innerhalb des Unternehmens gesendet werden, passieren diesen. E-Mail-Nutzerin Alice befinde sich innerhalb des Unternehmens, Bob außerhalb davon. Will Krypt & Co. serverbasierte E-Mail-Absicherung nutzen, dann schließt sie an den Mailserver eine spezielle Komponente an, die **E-Mail-Krypto-Gateway** (ECG) genannt wird und die für das Verschlüsseln sowie das Signieren von E-Mails zuständig ist. Die serverbasierte E-Mail-Absicherung funktioniert nun wie folgt:

1. Alice schickt ihre E-Mail an Bob im Klartext ab.
2. Bevor Alices E-Mail das Unternehmen verlässt, landet diese auf dem E-Mail-Krypto-Gateway. Dieses verschlüsselt und/oder signiert die E-Mail.
3. Das E-Mail-Krypto-Gateway leitet die verschlüsselte bzw. signierte Mail weiter.

Im beschriebenen Ablauf ist es unerheblich, ob Empfänger Bob die E-Mail mit einem geeigneten Client entschlüsselt bzw. verifiziert oder ob er selbst ebenfalls ein E-Mail-Krypto-Gateway in Anspruch nimmt. Zudem ist offensichtlich, dass eine Mail auch in die umgekehrte Richtung fließen kann. Geht eine von Bob verschlüsselte E-Mail bei Krypt & Co. ein, dann durchläuft diese ebenfalls das E-Mail-Krypto-Gateway, wo sie entschlüsselt und anschließend im Klartext an Alice weitergeleitet wird.

Die Vorteile der serverbasierten E-Mail-Verschlüsselung sind leicht zu erkennen. Krypt & Co. kann auf diese Weise flächendeckend das Verschlüsseln von E-Mails einführen, ohne auch nur einen Mitarbeiter mit einem kryptofähigen E-Mail-Client ausstatten zu müssen. Noch wichtiger ist die Tatsache, dass sich Anwenderin Alice nicht um das Verschlüsseln oder Signieren kümmern muss. Da Anwender bekanntlich notorisch faul und desinteressiert sind, kann ein E-Mail-Krypto-Gateway der IT-Abteilung von Krypt & Co. Zeit und Nerven sparen.

Allerdings hat die serverbasierte E-Mail-Absicherung auch erhebliche Nachteile. Vor allem ist zu bemängeln, dass die Teilstrecke zwischen Mail-Client und Mailserver nicht verschlüsselt wird – dabei kommen bekanntlich viele Abhörangriffe von innen. Außerdem ist Alice in diesem Fall nicht Herrin ihres privaten Schlüssels, sondern muss ihn an das Gateway abgeben. Digitale Signaturen verlieren bei einer serverbasierten Lösung ihren Sinn, da die Beweiskraft einer Signatur, die nicht Alice, sondern das Gateway mit Alices Schlüssel angefertigt hat, nicht besonders hoch ist.

Trotz dieser Nachteile sind E-Mail-Krypto-Gateways derzeit ein Verkaufsschlager. Der Grund dafür ist die einfache Handhabbarkeit, die für viele Unternehmen und Behörden offensichtlich wichtiger ist als eine hohe Sicherheitsstufe. Die serverbasierte E-Mail-Absicherung ist also wieder einmal ein Beispiel dafür, dass sich in der Kryptografie oft pragmatische Lösungen durchsetzen, auch wenn sie keine optimale Sicherheit bieten.

E-Mail-Krypto-Gateways gibt es von zahlreichen Anbietern. Die Hersteller von Mailservern (also vor allem Microsoft, Lotus und Novell) haben ihre Produkte dagegen bisher nicht mit einer Krypto-Gateway-Funktion ausgestattet – vermutlich wird das irgendwann kommen. Der Funktionsumfang der verfügaren E-Mail-Krypto-Gateway-Produkte ist oft beträchtlich. So bieten einige Lösungen die Möglichkeit, dass Alice in die Betreffzeile ihrer Mail eine Kombination von Buchstaben tippen kann, mit der sie festlegt, ob und wie die Mail verschlüsselt bzw. signiert wird. Zudem kann der Betreiber für das Gateway eine beliebige Policy konfigurieren, in der beispielsweise bestimmt wird, dass alle ausgehenden Mails verschlüsselt werden, dass die Mails bestimmter Absender signiert werden und dass für bestimmte Empfänger nur verschlüsselte Mails akzeptiert werden.

36.2 S/MIME

S/MIME ist ein standardisiertes Format für verschlüsselte und signierte E-Mails. Die Abkürzung steht für *Secure MIME* und bezieht sich auf den bereits erwähnten MIME-Standard, der das Strukturieren von E-Mails ermöglicht. S/MIME wurde in den neunziger Jahren von der US-Firma RSA Data Security (heute RSA Security) entwickelt und setzte sich gegen andere damals verbreitete E-Mail-Krypto-Standards wie PEM oder MOSS durch. Ab der 1998 erschienenen Version 2 wurde S/MIME innerhalb einer Arbeitsgruppe der IETF weiterentwickelt. Inzwischen liegt Version 3.2 vor [RFC5751]. S/MIME sieht vor, dass Absenderin Alice eine E-Mail digital signieren und verschlüsseln kann. Diese beiden Vorgänge sind voneinander unabhängig – es sind also auch unsignierte, verschlüsselte E-Mails sowie signierte, unverschlüsselte E-Mails zulässig. Das Verschlüsseln erfolgt stets über ein Hybridverfahren.

36.2.1 S/MIME-Format

Das von S/MIME verwendete Format für verschlüsselte und signierte E-Mails ist aus PKCS#7 abgeleitet. Es wird auch als CMS (Cryptographic Message Syntax) bezeichnet und ist unter diesem Namen in die S/MIME-Standardisierung eingegangen. RFC 3852 beschreibt die für S/MIME relevante CMS-Variante. Von den sechs Inhaltstypen, die CMS vorsieht, verwendet S/MIME nur vier: Data, SignedData, EnvelopedData und CompressedData. Wie der Name andeutet, verwendet S/MIME für die Strukturierung einer E-Mail-Nachricht das MIME-Format. Das bedeutet, dass die verschlüsselte Nachricht, der asymmetrisch verschlüsselte Schlüssel und andere Bestandteile einer S/MIME-E-Mail in verschiedene MIME-Blöcke aufgeteilt werden. Man kann S/MIME daher als MIME-Version von PKCS#7 betrachten. S/MIME schreibt die Unterstützung folgender Verfahren vor:

- *Verschlüsselung*: Triple-DES-Unterstützung ist Pflicht. Die zusätzliche Unterstützung von RC2 (mit 40 Bit Schlüssellänge) und des AES wird im Standard empfohlen.
- *Schlüsselaustausch*: RSA-Unterstützung ist Pflicht. Die zusätzliche Unterstützung von Diffie-Hellman wird im Standard empfohlen.
- *Digitale Signatur*: Die Möglichkeit zum Signieren mit DSA oder RSA ist Pflicht. Eine konforme Implementierung muss beide Verfahren verifizieren können.
- *Kryptografische Hashfunktion*: Kommt DSA für digitale Signaturen zum Einsatz, dann ist SHA-1-Unterstützung Pflicht. Wird RSA eingesetzt, dann muss MD5 oder SHA-1 unterstützt werden. Eine konforme Anwendung muss außerdem alle genannten Kombinationen verifizieren können.

Es gibt noch zahlreiche weitere Algorithmen, die innerhalb von S/MIME eingesetzt werden können. Deren Implementierung ist jedoch keine Pflicht. Vorgeschrieben ist dagegen die Unterstützung von X.509v3-Zertifikaten und X.509v2-

Sperrlisten nach PKIX-Profil. Neben RFC 5751 gibt es mehrere Dutzend weiterer RFCs zum Thema S/MIME. Diese beschreiben größtenteils Erweiterungen. So gibt es zahlreiche kryptografische Verfahren, die in der Grundversion von S/MIME nicht unterstützt werden, deren Verwendung jedoch möglich ist und in zusätzlichen RFCs spezifiziert wird. Dazu gehören Skipjack, KEA, CAST, IDEA, Camellia, SEED und GOST sowie verschiedene ECC-Verfahren. Weitere RFCs sehen beispielsweise signierte Empfangsbestätigungen, Sicherheitsetiketten (Security Labels) zur Übermittlung von Zusatzinformationen sowie Verschlüsselungsfunktionen für Mailinglisten vor. Weitere RFCs der S/MIME-Arbeitsgruppe behandeln den S/MIME-Einsatz in bestimmten Umgebungen (z.B. X.400), geben Hinweise zur Vermeidung von Problemen (z.B. mögliche Angriffe auf das Diffie-Hellman-Verfahren) oder nennen Einsatzbeispiele. Ein Blick auf die Webseite der S/MIME-Arbeitsgruppe der IETF verrät mehr zu diesem Thema.

36.2.2 S/MIME-Profil von Common PKI

Von den acht Teilen von Common PKI (siehe Abschnitt 26.4.3) ist einer (Teil 3) dem Thema E-Mail-Verschlüsselung und -Signierung gewidmet. Da Common PKI vor allem für die Anwendung des deutschen Signaturgesetzes geschaffen worden ist, ist das darin beschriebene Nachrichtenformat nicht zuletzt für E-Mails mit qualifizierten Signaturen nach Signaturgesetz gedacht. Die Common-PKI-Entwickler vermieden es, das Rad neu zu erfinden. Daher ist das beschriebene Nachrichtenformat nahezu mit S/MIME identisch. Bisher wird jedoch nur S/MIME 3.0 unterstützt, während die aktuelle Version 3.1 nicht berücksichtigt wird. Die Beschreibung des S/MIME-Profils von Common PKI umfasst nur wenige Seiten und besteht fast nur aus Empfehlungen, S/MIME in einer bestimmten Weise zu verwenden. Zudem wird die Nutzung der kryptografischen Verfahren eingeschränkt. Während S/MIME die verwendeten Verfahren offen lässt und lediglich einige Möglichkeiten vorgibt, schreibt Common PKI vor, dass nur die im Teil 6 des Standards aufgeführten Verfahren verwendet werden dürfen. Dabei handelt es sich in allen Fällen um bewährte Algorithmen wie RSA, DSA, ECDSA, AES, Triple-DES und SHA-1.

36.2.3 Bewertung von S/MIME

S/MIME hat sich in den letzten Jahren als wichtigster Standard für die E-Mail-Verschlüsselung und -Signierung etabliert. Es wird von zahlreiche E-Mail-Clients, Plugins und E-Mail-Krypto-Gateways unterstützt. Der einzige ernsthafte Konkurrent von S/MIME ist OpenPGP. Letzteres ist unter Privatanwendern und in kleineren Installationen durchaus beliebt, doch in professionellen Umgebungen ist S/MIME praktisch konkurrenzlos. Durch seine X.509-Unterstützung ist es zu zahlreichen anderen PKI-Anwendungen interoperabel und lässt sich sehr gut mit Smartcards kombinieren.

36.3 OpenPGP

OpenPGP habe ich in Abschnitt 26.4.4 als PKI-Standard vorgestellt. Historisch gesehen handelt es sich bei OpenPGP jedoch eher um einen E-Mail-Krypto-Standard mit eigenem Zertifikatsformat. Dieser ist aus der Software PGP entstanden, die sich bis heute großer Beliebtheit erfreut. OpenPGP ist derzeit die einzige praxisrelevante Alternative zu S/MIME. OpenPGP basiert auf zahlreichen proprietären Bestandteilen, die inzwischen von der IETF in einem RFC standardisiert wurden [RFC2440]. Abgesehen von diesem gibt es nur einen weiteren RFC zum Thema OpenPGP, während die S/MIME-Arbeitsgruppe Dutzende von RFCs veröffentlichte. Schon diese Tatsache zeigt, dass S/MIME im Vergleich zu OpenPGP deutlich umfangreicher, dafür aber auch komplexer ist.

36.3.1 OpenPGP

Von der Idee her sind sich OpenPGP und S/MIME recht ähnlich. OpenPGP bietet digitale Signaturen, Verschlüsselung mit einem Hybridverfahren und digitale Zertifikate. Das Format kann nicht nur für E-Mails, sondern für beliebige Daten verwendet werden. Das unterstützte Zertifikatsformat entspricht dem OpenPGP-Format. Zum Signieren wird in OpenPGP das RSA-Verfahren oder der DSA verwendet, für den Schlüsselaustausch ist ebenfalls RSA oder Diffie-Hellman vorgesehen.

Da PGP-Erfinder Phil Zimmermann seinerzeit dem DES nicht traute, sah er als symmetrisches Verfahren zunächst IDEA vor, was sich auch im OpenPGP-Standard niedergeschlagen hat. Inzwischen sieht die OpenPGP-Spezifikation zusätzlich Triple-DES, Safer, Blowfish und CAST vor. Die unterstützten kryptografischen Hashfunktionen sind MD5, RIPEMD-160 und SHA-1. Zusammen mit einer verschlüsselten Mail wird das Zertifikat des Absenders geschickt. Dieses entspricht nicht dem X.509-Format, vielmehr werden OpenPGP-Zertifikate verwendet.

36.3.2 Bewertung von OpenPGP

OpenPGP ist zweifellos ein interessanter E-Mail-Krypto-Standard, der vor allem für Heimanwender und kleinere Firmeninstallationen geeignet ist. Viele größere Unternehmen nutzen OpenPGP zudem als Provisorium vor der Einführung einer umfassenden E-Mail-Sicherheitslösung – wobei Provisorien bekanntermaßen oft ein langes Leben haben. Bei den genannten Einsatzzwecken kommt OpenPGP zugute, dass es einfacher und pragmatischer ist als S/MIME. Erfahrungsgemäß kommt es dadurch zu weniger technischen Schwierigkeiten und Interoperabilitätsproblemen. Von Vorteil ist in diesem Zusammenhang zweifellos auch, dass OpenPGP ein leicht anarchistisches Image hat, in dem die Geschichte von PGP und Phil Zimmermann mitschwingt. OpenPGP wird seine oftmals sehr treuen Fans daher auch in den nächsten Jahren nicht verlieren.

Trotz seiner Popularität hat OpenPGP allerdings auch ernsthafte Nachteile. Dazu gehört vor allem, dass es sich dabei nach wie vor um eine komplette Insellösung handelt. X.509, PKCS oder PKIX vertragen sich nicht mit OpenPGP. Dies ist schon allein deshalb ein Mangel, weil diese Standards außerhalb des E-Mail- und Dateiverschlüsselungsbereichs fast konkurrenzlos sind, während OpenPGP dort so gut wie keine Rolle spielt. Es ist daher beispielsweise kaum möglich, ein OpenPGP-Zertifikat für ein VPN, für eine SSL-Verbindung oder im WLAN einzusetzen. Es gibt schlichtweg keine Produkte, die dies unterstützen.

Als weiterer Mangel von OpenPGP gilt das Web of Trust. Zwar können auch OpenPGP-Zertifikate von einer CA signiert werden und somit ein hierarchisches Vertrauensmodell bilden. OpenPGP-Zertifikate sind für diesen Zweck jedoch nicht geschaffen und bieten nicht die Zertifikatsfelder, die man dafür braucht. Ein weiterer Nachteil: Für die Unterstützung von Smartcards im OpenPGP-Umfeld gibt es so gut wie keine Implementierung und schon gar keinen Standard.

Wer mit OpenPGP arbeitet, muss also damit leben, dass er es mit einer Nischenlösung zu tun hat, die obendrein mit dem Mainstream nicht interoperabel ist. Für ambitionierte E-Mail-Krypto-Projekte, die eine PKI-Unterstützung vorsehen, ist daher normalerweise nicht OpenPGP, sondern S/MIME erste Wahl. Allerdings gibt es durchaus auch Unternehmen, die zweigleisig fahren. Diese unterstützen S/MIME als bevorzugten Standard, bieten ihren Mitarbeitern aber auch die Nutzung von OpenPGP an. Da viele Kommunikationspartner mit OpenPGP arbeiten, ist dies zweifellos eine sinnvolle Maßnahme.

36.4 Abholen von E-Mails: POP und IMAP

Für den Transport von E-Mails über das Internet kommt in der Regel das Netzwerkprotokoll SMTP (Simple Mail Transfer Protocol) zum Einsatz. Dieses leitet eine E-Mail von Alices PC zum Mailserver und sorgt für die Übertragung einer E-Mail von einem Mailserver zum anderen. Wenn Alice allerdings ihre E-Mails vom Mailserver abruft, dann wird im Normalfall ein anderes Protokoll verwendet. Dabei handelt es sich meist um das *Post Office Protocol* (*POP*) oder um dessen Nachfolger *IMAP* (*Internet Message Access Protocol*). Beide Protokolle sind speziell dafür geschaffen, eine Anfrage an einen Mailserver zu stellen und dort gespeicherte E-Mails herunterzuladen.

POP ist nicht nur das ältere, sondern auch das einfachere der beiden Protokolle. In seiner einfachsten Form funktioniert es so: Alice verbindet sich mit dem Mailserver, gibt ein Passwort ein und bekommt anschließend die für sie bestimmten Mails auf ihren Rechner geliefert. Anschließend löscht der Server die an Alice übertragenen Nachrichten. Neben diesem simplen Ablauf unterstützt POP auch einige weitere Funktionen, die es Alice erlauben, Mails nur teilweise vom Server herunterzuladen oder verschiedene Verzeichnisse zu verwenden.

IMAP bietet im Vergleich zu POP deutlich mehr an Funktionalität. Alice kann damit mehrere Postfächer verwalten, sich ein Postfach mit Bob teilen und

E-Mails auf verschiedene Weise markieren, ohne sie auf ihren Rechner herunterzuladen. IMAP ist zudem nicht nur für das Abholen von E-Mails geeignet, sondern dient auch dem Zugang zu News-Gruppen und anderen Datensammlungen.

36.4.1 Gefahren beim Abholen von E-Mails

Es versteht sich von selbst, dass das Abholen von E-Mails mit POP oder IMAP ein enormes Sicherheitsrisiko in sich birgt. Wenn es Mallory gelingt, sich gegenüber dem Server als Alice auszugeben, dann kann er alle für Alice bestimmten E-Mails auf seinen Rechner laden. Alice bekommt diese E-Mails nicht und bemerkt möglicherweise nicht einmal, dass etwas fehlt. Zudem kann Mallory sich auch damit begnügen, Alices Kommunikation mit dem Mailserver abzuhören, um so die übertragenen Mails mitzulesen.

Wenn Alice nur E-Mails zugesendet bekommt, die mit S/MIME oder OpenPGP verschlüsselt sind, dann kann Mallory mit einer unberechtigten POP- oder IMAP-Anfrage wenig ausrichten. Allerdings werden in der Praxis nur wenige E-Mails verschlüsselt. Außerdem kann Mallory bei einer verschlüsselten Mail erkennen, von wem sie stammt, was eine Verkehrsflussanalyse ermöglicht. Wenn Mallory besonders durchtrieben ist (wovon wir ja ausgehen), dann bietet sich folgender Angriff an: Über einen unrechtmäßigen POP- oder IMAP-Zugriff lässt er alle verschlüsselten E-Mails verschwinden. Alice denkt dann vielleicht, dass der Mailserver mit dem neumodischen Verschlüsselungskram nicht zurechtkommt, und bittet Bob, seine Mails an sie nicht mehr zu verschlüsseln. Anschließend hat Mallory leichtes Spiel.

Trotz des offensichtlichen Sicherheitsrisikos sahen die ursprünglichen Versionen von POP und IMAP keinerlei Kryptografie vor – lediglich eine Passwortabfrage wurde unterstützt. Zum Glück hat sich inzwischen einiges getan, und deshalb gibt es heute mehrere (vielleicht sogar zu viele) Krypto-Zusätze für IMAP und POP.

36.4.2 Krypto-Zusätze für IMAP

IMAP ist im Vergleich zu POP das modernere Protokoll. Deshalb gibt es hier auch mehr Möglichkeiten für eine kryptografische Absicherung.

IMAP Authentication Mechanisms

In [RFC1731] werden drei IMAP-Zusätze für eine sichere Authentifizierung (und teilweise auch für die Verschlüsselung) von E-Mail-Abfragen beschrieben (**IMAP Authentication Mechanisms**). Dabei handelt es sich um folgende Zusätze:

- *Kerberos*: Dieser Zusatz ermöglicht eine Nutzung von Kerberos zur Authentifizierung und Verschlüsselung.

- *GSS-API*: Hier wird eine Authentifizierung und Verschlüsselung mit den Mitteln der GSS-API ermöglicht.
- *S/Key*: Dieser Zusatz ermöglicht den Einsatz von Einmal-Passwörtern nach der von S/Key vorgesehenen Methode. Verschlüsselt wird dabei nicht.

CRAM

Die IMAP Authentication Mechanisms sehen zwar verschiedene Möglichkeiten zur Authentifizierung vor. Ein Challenge-Response-Verfahren nach Vorbild der Digest Access Authentication oder CHAP ist jedoch nicht dabei. Dies wird in [RFC2195] nachgeholt. Darin wird ein speziell für IMAP entwickeltes Verfahren namens **Challenge-Response Authentication Mechanism** (**CRAM**) spezifiziert. Dabei handelt es sich um ein einfaches Challenge-Response-Verfahren, bei dem der Server einen Zufallswert, seinen Servernamen und einen Zeitstempel als Challenge an Bob sendet. Bob wendet auf diese Daten eine schlüsselabhängige Hashfunktion an und schickt das Resultat als Response an den Server zurück. Das vorgesehene Verfahren ist ein HMAC auf Basis von MD5.

IMAP über TLS oder SASL

Als typisches Client-Server-Protokoll, das über TCP arbeitet, lässt sich IMAP auch über SSL (bzw. TLS) absichern. In [RFC2595] wird diese Kombination unter dem Namen *IMAP-TLS* beschrieben. Zu guter Letzt kann auch SASL verwendet werden, um IMAP abzusichern. IMAP war sogar als wichtigste Anwendung von SASL geplant. Aus diesem Grund waren die ersten drei Sicherheitsmechanismen, die für SASL definiert wurden, Kerberos, GSS-API und S/Key – also genau dieselben Methoden, die auch die IMAP Authentication Mechanisms vorsehen.

36.4.3 Krypto-Zusätze für POP

Alle beschriebenen Krypto-Zusätze für IMAP sind auch für POP nutzbar:

- *IMAP Authentication Mechanisms*: [RFC1734] spezifiziert eine POP-Erweiterung, über die die IMAP Authentication Mechanisms auch bei POP anwendbar sind.
- *CRAM*: CRAM kann auch zur Absicherung von POP eingesetzt werden.
- *SSL, SASL*: Auch SSL und TLS lassen sich für POP nutzen. Der erwähnte RFC 2595, der IMAP-TLS beschreibt, sieht auch die Kombination *POP-TLS* vor. Die Nutzung von POP über SASL ist ebenfalls möglich.

37 Weitere Krypto-Protokolle der Anwendungsschicht

Schicht 7 des OSI-Modells ist diejenige Schicht, in der es die meisten kryptografischen Netzwerkprotokolle gibt. Das Verschlüsseln und Signieren von E-Mails, das ich im vorhergehenden Kapitel vorgestellt habe, ist dabei nur die Spitze des Eisbergs. In diesem Kapitel wollen wir uns die wichtigsten Krypto-Lösungen für Schicht 7 anschauen, die es sonst noch gibt.

37.1 Kryptografie im World Wide Web

Das wohl wichtigste Netzwerkprotokoll in Schicht 7 des OSI-Modells ist HTTP (Hypertext Transfer Protocol). Es kommt im World Wide Web zum Einsatz und wird daher von jedem Webbrowser und -server unterstützt. HTTP ist ein recht einfaches Protokoll. Dessen typischer Ablauf sieht vor, dass Alice eine Anfrage an einen Server schickt, in der die Kennung (URL) einer Datei enthalten ist. Als Antwort erhält sie diese Datei (oder eine Fehlermeldung) zurück. Der Inhalt der von

Alice angeforderten Datei kann beliebig sein – meist handelt es sich jedoch um den Quellcode einer Webseite, die anschließend von Alices Webbrowser dargestellt wird. Die Beschreibung einer Webseite erfolgt in der Regel im HTML-Format. HTTP war in seiner Version 1.0 zustandslos. In der aktuellen Version 1.1 ist es immerhin möglich, Dateien hintereinander herunterzuladen, ohne jedes Mal eine neue Verbindung aufbauen zu müssen.

In HTTP 1.0 gab es keine Kryptografie. Dabei sind Verschlüsselung und digitale Signaturen im Web durchaus sinnvoll. Wenn die Firma Krypt & Co. beispielsweise ihre aktuelle Preisliste für Außendienstmitarbeiterin Alice, aber nicht für die Konkurrenz abrufbar machen will, dann kann sie diese Liste auf eine Webseite setzen, die nur nach einer Authentifizierung zugänglich ist. Wenn Bob auf einer Webseite seine neue Waschmaschine bestellt, dann will er, dass seine Kreditkartennummer nur verschlüsselt an den Server übertragen wird. Solche und ähnliche Anwendungen führten dazu, dass man sich schon früh Gedanken machte, wie man HTTP um Krypto-Verfahren erweitern konnte. In der Folge entstand eine ganze Reihe teilweise vollkommen unterschiedlicher Krypto-Erweiterungen für HTTP.

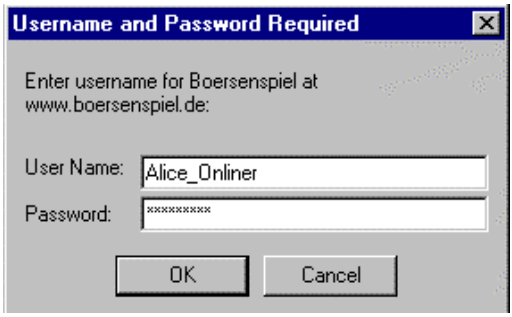


Abb. 37–1 *Die Basic Authentication ist eine einfache Passwortabfrage, die von allen Browsern unterstützt wird. Leider wird dabei keine Kryptografie eingesetzt.*

37.1.1 Basic Authentication

HTTP 1.0 enthält zwar keinerlei Kryptografie, eine Authentifizierung durch eine Passwortabfrage wird jedoch unterstützt. Der entsprechende Bestandteil von HTTP heißt **Basic Authentication**. Die Basic Authentication ist eine Passwort-Authentifizierung einfachster Prägung: In der Anfrage, die Alice an den Server verschickt, sind eine Anwender-ID und ein Passwort enthalten. Nur wenn das Passwort korrekt ist, sendet der Server anschließend die von Alice gewünschte Datei zurück. Das bei der Basic Authentication verwendete Passwort wird zwar in einem Format übertragen, das nicht menschenlesbar ist, es ist jedoch nicht verschlüsselt.

37.1.2 Digest Access Authentication

Da die in Version 1.0 von HTTP enthaltene Basic Authentication Mallory kaum einen Schrecken einjagt, wurde mit der Version 1.1 ein weiterer Authentifizierungsmechanismus eingeführt. Dieser wird als **Digest Access Authentication** bezeichnet und wird in [RFC2617] beschrieben. Es handelt sich dabei um ein Challenge-Response-Verfahren. Die Funktionsweise der Digest Access Authentication ist recht einfach: Will Bob seine Webseite nicht für die Allgemeinheit, sondern nur für Alice zugänglich machen, dann vereinbart er mit ihr ein Passwort (wie bei der Basic Authentication). Will Alice diese Seite mit ihrem Webbrowser laden, dann schickt Bobs Webserver eine Zufallszahl (Challenge) an Alice. Alice wendet auf diese Zufallszahl, das vereinbarte Passwort, die Webadresse und einen Teil des HTTP-Headers eine kryptografische Hashfunktion an, deren Ergebnis sie an Bobs Webserver zurückschickt. Dieser prüft das Ergebnis und sendet im Erfolgsfall die Webseite. Als Hashfunktion für die Digest Access Authentication ist MD5 vorgesehen, es können jedoch auch andere verwendet werden. Wie im RFC 2617 vermerkt wird, ist die Digest Access Authentication »keine komplette Antwort für die Sicherheitsbedürfnisse des World Wide Web«. Die Digest Access Authentication ist vielmehr ein Mittelweg: Ist SSL zu aufwendig und die Basic Authentication zu unsicher, dann ist die Digest Access Authentication die richtige Wahl.

37.1.3 NTLM

NTLM (NT LAN Manager) ist ein Netzwerkprotokoll der Firma Microsoft [Glass]. Es wird auch als *NTCR* (NT challenge/response) bezeichnet. Mit NTLM kann sich Alice gegenüber einem Webserver authentisieren. Dieser Vorgang erfolgt automatisch, nachdem sie sich an ihrem Windows-Client angemeldet hat. NTLM bietet also ein Single Sign-On. NTLM ist eine proprietäre Lösung der Firma Microsoft und ist daher vor allem in Produkten dieses Herstellers implementiert. Allerdings unterstützen beispielsweise auch der Mozilla Firefox, Opera und der Apache Web Server dieses Protokoll. NTLM sieht eine Challenge-Response-Authentifizierung vor, die den DES sowie MD4 oder MD5 zur Berechnung der Response verwendet.

37.1.4 HTTP über SSL (HTTPS)

Mit der Digest Access Authentication und NTLM gibt es zwar kryptografische Sicherheitsmechanismen für HTTP. Diese dienen jedoch nur der Authentifizierung, nicht aber der Verschlüsselung. Wenn neben Authentizität auch Vertraulichkeit gefordert ist, kommt im World Wide Web oft SSL ins Spiel. Von SSL, einem Krypto-Protokoll für die Transportschicht, war bereits in Kapitel 35 die Rede. Durch seine Lage zwischen TCP und einem Protokoll der Anwendungs-

schicht kann SSL sehr gut im Zusammenspiel mit HTTP verwendet werden (man nennt diese Kombination auch *HTTPS*). Alle gängigen Webbrowser und Webserver unterstützen daher nicht nur HTTP, sondern liefern die darunter liegende SSL-Schicht gleich mit. Vorteilhaft hierbei ist, dass SSL mit dem ApplicationData-Protokoll einen Bestandteil hat, der auf das darüber liegende Protokoll (in diesem Fall HTTP) abgestimmt werden kann.

Historisch gesehen waren die Unsicherheiten von HTTP der Grund, warum SSL überhaupt entwickelt wurde. Wenn Alice HTTPS einsetzen will, dann tippt sie eine Adresse in den Browser ein, die mit »https« beginnt. Der Browser versucht anschließend, eine SSL-Verbindung aufzubauen. Unterstützt der Server ebenfalls SSL, so kann die folgende Kommunikation mit SSL geschützt werden. Dies wird von den gängigen Browsern mit dem Symbol eines geschlossenen Schlosses angezeigt (ohne SSL wird das Schloss geöffnet dargestellt). HTTPS bietet eine Authentifizierung, die deutlich sicherer ist als die Digest Access oder Basic Authentication. Außerdem wird mit HTTPS die Kommunikation verschlüsselt. HTTPS hat jedoch alle Nachteile, die Krypto-Protokolle unterhalb der Anwendungsschicht haben. Insbesondere können mit HTTPS keine digitalen Signaturen zum Signieren von Webinhalten eingesetzt werden.

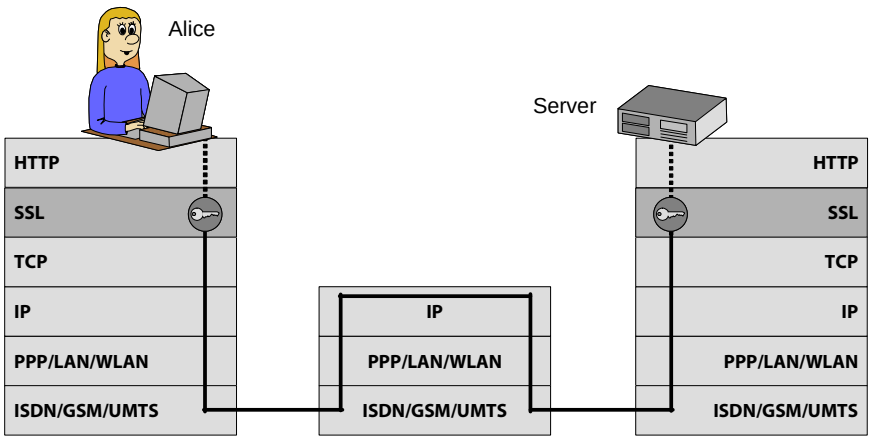


Abb. 37–2 *Wird HTTP zusammen mit SSL eingesetzt, so nennt man dieses Protokollpaar HTTPS.*

37.1.5 Was es sonst noch gibt

Einige weitere Krypto-Ansätze aus der Frühzeit des Web haben sich nicht durchgesetzt. Dazu gehört das Schicht-7-Protokoll *S-HTTP* (nicht zu verwechseln mit HTTPS). Ein ähnliches Schicksal wurde dem SSL-Gegenstandard *PCT* (*Private Communications Technology*) von Microsoft zuteil. In den gängigen Webbrowsern noch nicht implementiert sind die beiden XML-Standards XML Encryption und XML Signature (siehe Abschnitt 18.4.2). Sollte sich dies in zukünftigen

Browserversionen ändern, dann sind zahlreiche interessante Anwendungen – etwa das digitale Signieren von Formularen (**Form Signing**) – denkbar.

37.2 Kryptografie für Echtzeitdaten im Internet (RTP)

Das *Real Time Transport Protocol* (*RTP*) ist ein Schicht-7-Protokoll für die Übertragung von Audio- und Videodaten im Internet. Es behebt zwei Mängel der TCP/IP-Protokolle, die für Audio und Video eine wichtige Rolle spielen. Der eine Mangel besteht darin, dass TCP und IP auf die Übertragung von Daten an *einen* Empfänger ausgerichtet sind (Unicast), während bei Radio und Fernsehen Multicast gefragt ist. Der andere Nachteil ist, dass TCP und IP nicht echtzeitfähig sind. RTP arbeitet im Normalfall über UDP. Zusätzlich gibt es noch das *Real Time Control Protocol* (*RTCP*), das zur Steuerung von RTP-Verbindungen dient. RTP und RTCP werden meist zusammen eingesetzt.

37.2.1 SRTP

RTP sah ursprünglich keine Kryptografie vor. Mitarbeiter der Firmen Cisco und Ericsson entwickelten jedoch eine Sicherheitserweiterung, die sie **SRTP** (**Secure RTP**) nannten. SRTP wurde 2004 als RFC standardisiert [RFC3711]. Zusätzlich gibt es auch **SRTCP** (**Secure RTCP**), das dieselben Sicherheitsmechanismen wie SRTP realisiert, allerdings für RTCP eingesetzt wird. Da es zwischen SRTP und SRTCP keine kryptografisch relevanten Unterschiede gibt, betrachten wir im Folgenden nur Ersteres.

SRTP ermöglicht einen sicheren Kanal zwischen Senderin Alice und Empfänger Bob (es kann auch mehrere Empfänger geben), stellt dazu jedoch nur die Datenaustauschroutine bereit. Es wird also vorausgesetzt, dass die beiden einen gemeinsamen geheimen Schlüssel besitzen. Dieser ist ein Masterschlüssel, der nur zur Ableitung anderer Schlüssel verwendet wird, um Known-Key-Attacken zu verhindern. Die vom Masterschlüssel abgeleiteten Sitzungsschlüssel sind ein Authentifizierungsschlüssel, ein Verschlüsselungsschlüssel und ein sogenannter Salt-Schlüssel. Letzterer dient zur Generierung von Salt-Werten (siehe Abschnitt 14.1.3). Die Ableitung der Schlüssel erfolgt mit einem AES-basierten Verfahren. SRTP wendet zur Authentifizierung und zum Integritätsschutz HMAC mit SHA-1 an, wobei der Authentifizierungsschlüssel verwendet wird. Die Verschlüsselung erfolgt mit dem AES und dem Verschlüsselungsschlüssel. Dabei kommt eine Variante des CTR-Modus zur Anwendung, wobei in die Verschlüsselung ein Salt-Wert eingeht, der mit dem Salt-Schlüssel gebildet wird.

37.2.2 SRTP-Initialisierungsroutinen

SRTP geht davon aus, dass Alice und Bob bereits über einen gemeinsamen Masterschlüssel verfügen. Deshalb gibt es Bedarf für ein Schlüsselaustausch-Protokoll – also für eine Initialisierungsroutine des sicheren Kanals. Es gibt für diesen Zweck mehrere, voneinander unabhängige Protokolle. Die zwei wichtigsten davon schauen wir uns im Folgenden an.

ZRTP

ZRTP ist ein Netzwerkprotokoll, das einen Diffie-Hellman-Schlüsselaustausch für SRTP realisiert [RFC6189]. Es wurde unter der Mitwirkung von PGP-Erfinder Phil Zimmermann entwickelt und ist für den Einsatz beim Telefonieren im Internet vorgesehen. Zimmermann entwickelte ZRTP als Bestandteil des von ihm ersonnenen Produkts Zfone, das die Verschlüsselung von Internettelefonie mit SRTP vorsieht. Das »Z« in ZRTP steht demnach für Zfone. ZRTP ist ein vergleichsweise simples Protokoll. Mithilfe eines Diffie-Hellman-Schlüsselaustauschs versorgt es Alice und Bob mit einem gemeinsamen geheimen Schlüssel, den diese als Masterschlüssel für das Internettelefonieren über SRTP verwenden können. ZRTP sieht zwar die Verwendung eines asymmetrischen Verfahrens (Diffie-Hellman) vor, verzichtet jedoch auf eine PKI-Unterstützung. Laut ZRTP-Erfinder Phil Zimmermann, der nicht gerade als PKI-Fan bekannt ist, war dieser Verzicht Absicht, da ihm eine PKI-Unterstützung zu komplex und außerdem unnötig erschien.

ZRTP verwendet das Diffie-Hellman-Verfahren mit 3.078 oder 4.096 Bit Schlüssellänge. Da für jedes Telefonat ein neuer Masterschlüssel generiert wird, bietet Zfone Forward Security und Backward Security, selbst wenn Mallory in den Besitz des Masterschlüssels gelangt. Eine der Funktionen von ZRTP ist das Aushandeln von kryptografischen Verfahren für den SRTP-Nachrichtenaustausch. AES (128 Bit Schlüssellänge) und SHA-256 müssen unterstützt werden, andere Verfahren sind möglich.

Einer der offensichtlichen Nachteile von ZRTP ist die fehlende Authentifizierung. Alice und Bob haben (im allgemeinen Fall) zu Beginn des Protokollablaufs keinen gemeinsamen geheimen Schlüssel, und sie können keine digitalen Zertifikate zur Authentifizierung einsetzen. Mallory kann daher beispielsweise Alice anrufen und sich für Bob ausgeben. Er kann außerdem eine Man-in-the-Middle-Attacke gegen das Diffie-Hellman-Verfahren starten, indem er seinen öffentlichen Schlüssel gegenüber Alice als den von Bob und gegenüber Bob als den von Alice ausgibt. Auf diese Weise gelangt er in den Besitz je eines gemeinsamen geheimen Schlüssels mit Alice und mit Bob. Beim anschließenden Nachrichtenaustausch kann er dies zum Mithören nutzen. Um derartige Angriffe zu verhindern, bietet ZRTP einige Gegenmaßnahmen, die zwar in der Theorie angreifbar sind, für praktische Zwecke jedoch ausreichen dürften:

- Um zu verhindern, dass Mallory eine Man-in-the-Middle-Attacke auf das Diffie-Hellman-Verfahren startet, bietet ZRTP die Möglichkeit, einen Hashwert, der unter anderem aus den beiden Schlüsseln generiert wird, mündlich zu übertragen. Zwar kann Mallory auch hier durch eine Man-in-the-Middle-Attacke eingreifen, doch dazu müsste er Alices und Bobs Stimmen imitieren.
- ZRTP arbeitet mit dem Baby-Duck-Verfahren (siehe Abschnitt 20.3.5). Wenn Mallory ein Telefongespräch abhören will, dann benötigt er sämtliche Masterschlüssel, die (per Diffie-Hellman) seit der letzten mündlichen Hashwert-Authentifizierung generiert wurden.
- ZRTP ermöglicht die Verwendung eines gemeinsamen symmetrischen Schlüssels, der manuell ausgetauscht wird. Alice und Bob können sich also bei einem persönlichen Treffen auf einen Schlüssel einigen.

Auch wenn die Authentifizierung nicht der reinen Lehre entspricht, ist ZRTP eine sinnvolle Entwicklung. Die aus Kryptografensicht nicht optimale Authentifizierung halte ich für tolerierbar, sofern ZRTP zwischen Personen eingesetzt wird, die sich kennen. In diesem Fall ist es für einen Angreifer äußerst schwierig, am Telefon eine falsche Identität vorzuspielen. Für Anwendungen, in denen die Authentifizierung besonders wichtig ist (z.B. telefonische Entgegennahme einer Bestellung oder Telefon-Banking) muss Zfone mit einem anderen Werkzeug (z.B. mündliche Passwortabfrage) kombiniert werden.

MIKEY

MIKEY (**Multimedia Internet Keying**) ist ein weiteres Protokoll, mit dem sich ein Masterschlüssel für die Nutzung mit RTP vereinbaren lässt. Es wird in RFC 3830 beschrieben [RFC3830]. MIKEY ist deutlich komplexer und flexibler als ZRTP und lässt sich im Gegensatz zu diesem nicht nur für die Internettelefonie einsetzen. Im Gegensatz zu ZRTP unterstützt MIKEY auch Multicasting (Versenden von Nachrichten an mehrere Empfänger).

MIKEY unterstützt drei Varianten für die Schlüsselvereinbarung:

- *Pre-shared Key*: Diese Variante sieht vor, dass Alice und Bob bereits einen gemeinsamen geheimen Schlüssel besitzen. Wenn es mehrere Empfänger gibt, muss Alice mit jedem von diesen einen Schlüssel gemeinsam haben.
- *Public Key*: Hierbei kommt das RSA-Verfahren für den Schlüsselaustausch zum Einsatz. Die verwendeten öffentlichen Schlüssel sollten durch digitale Zertifikate beglaubigt sein.
- *Diffie-Hellman*: Hier kommt ein Diffie-Hellman-Schlüsselaustausch zur Ausführung.

Weitere RFCs beschreiben zusätzliche Methoden. So geht es in [RFC4650] um einen Diffie-Hellman-Schlüsselaustausch mit HMAC-Authentifizierung, und [RFC4738] spezifiziert eine weitere Variante für den Schlüsselaustausch mit RSA.

37.2.3 Bewertung von SRTP

SRTP ist zweifellos ein interessantes kryptografisches Netzwerkprotokoll, für das es einigen Bedarf gibt. Allerdings zeigt die Erfahrung, dass die Bereitschaft der Anwender, Telefongespräche oder Videokonferenzen zu verschlüsseln, eher gering ist. Der zukünftige Erfolg von SRTP wird daher sicherlich auch davon abhängen, wie viele Abhörskandale es im Zusammenhang mit der Internettelefonie und Internetvideokonferenzen geben wird.

37.3 Secure Shell (SSH)

Die **Secure Shell** (**SSH**) ist ein kryptografisches Netzwerkprotokoll, das in Schicht 7 des OSI-Modells angesiedelt ist. Es ermöglicht den Betrieb eines sicheren Kanals zwischen zwei Kommunikationspartnern – meist zwischen einem Client und einem Server. Die Secure Shell lässt sich für verschiedene Verwendungszwecke nutzen:

- *Sicherer Serverzugriff*: Alice nutzt den sicheren Kanal in diesem Fall, um Befehle an einen Server zu übermitteln. Hierbei ersetzt die Secure Shell unsichere Netzwerkprotokolle wie Telnet oder Rlogin. Vor allem zur Administration eines Servers ist diese Art des Zugriffs beliebt.
- *Sicherer Dateitransfer*: In dieser Anwendungsvariante lädt Alice über den sicheren Kanal Dateien herunter oder hoch. Hierbei ersetzt die Secure Shell das unsichere FTP oder ein anderes Dateitransfer-Programm.
- *Sicherer Kanal für andere Protokolle*: Die Secure Shell kann auch als sicherer Kanal für andere Netzwerkprotokolle der Schicht 7 verwendet werden. Beispiele dafür sind HTTP und FTP. Diese Anwendung ist möglich, wenn das jeweilige Protokoll den Austausch von ASCII-Texten vorsieht. Die Secure Shell spielt in diesem Fall eine ähnliche Rolle wie SSL.

37.3.1 Entstehungsgeschichte der Secure Shell

Die Geschichte der Secure Shell hat einige Parallelen mit der von PGP. Am Anfang stand die Idee einer Einzelperson, die in diesem Fall Tatu Ylönen hieß. Der Finne entwickelte 1995 die erste Version der Secure Shell für den Eigenbedarf und veröffentlichte das Ergebnis als Freeware. Zu seiner eigenen Überraschung stieß Ylönen damit auf großes Interesse, und so erlangte die Secure Shell schnell eine beachtliche Verbreitung. Da die Software nicht nur kostenlos, sondern auch gut implementiert und dokumentiert war, entwickelte sich die Secure Shell zum nach PGP zweitwichtigsten freien Krypto-Programm. Das Netzwerkprotokoll, das die Software verwendete, wurde im Nachhinein als **SSH-1** bezeichnet.

Angesichts des überraschenden Erfolgs gründete Ylönen noch im Jahr 1995 ein Unternehmen namens SSH Communications Security, um die Secure Shell zu vermarkten. 1998 brachte diese Firma eine neue Version der Software heraus, die im Gegensatz zum Vorgänger nicht frei verfügbar war. Deren Protokoll wurde **SSH-2** genannt. SSH-2 behob einige Mängel von SSH-1 und beruhte auf einem durchdachteren Design. Allerdings war SSH-2 nicht mit SSH-1-kompatibel. 1997 gründete sich eine IETF-Arbeitsgruppe, die SSH-2 zu einem Internetstandard machte. Es dauerte jedoch einige Jahre, bevor 2006 schließlich die erste Standardisierung abgeschlossen war. Mittlerweile sind über ein Dutzend RFCs zur Secure Shell erschienen. Neben der kommerziellen Implementierung von SSH Communications Security gibt es die Open-Source-Lösung **OpenSSH** sowie mindestens zehn weitere Implementierungen.

37.3.2 Funktionsweise der Secure Shell

Die Funktionsweise der Secure Shell wird in den vier RFCs 4251 bis 4254 beschrieben. Der wichtigste davon ist RFC 4251, in dem die Protokollarchitektur standardisiert wird [RFC4251]. Das Secure-Shell-Protokoll sieht drei übereinander liegende Schichten vor, in denen jeweils ein eigenes Protokoll zum Einsatz kommt. Diese drei Schichten bzw. Protokolle heißen *Authentication Protocol* [RFC4252], *Transport Layer Protocol* [RFC4253] und *Connection Protocol* [RFC4254].

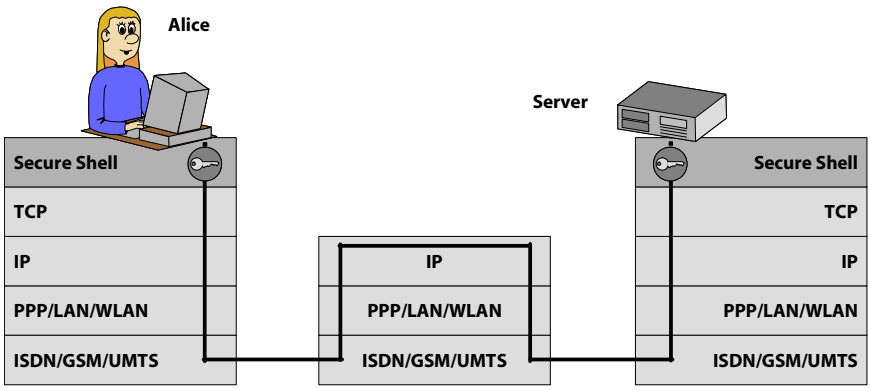


Abb. 37–3 *Die Secure Shell ist eine Mehrzweck-Kryptolösung für Schicht 7.*

Connection Protocol

Das Connection Protocol ist das unterste der drei Secure-Shell-Protokolle. Es führt zwar keine kryptografischen Operationen durch, ist für uns aber dennoch wichtig. Die Aufgabe dieses Protokolls ist es, über TCP einen oder mehrere Kanäle für die Daten aufzubauen, die aus der darüber liegenden Schicht kommen

oder an diese weitergereicht werden. Es ist eine Besonderheit der Secure Shell, dass sie für diesen Zweck mehrere parallele Kanäle unterstützt, die unabhängig voneinander über dieselbe gesicherte Verbindung Daten übertragen können. Dies ist ein wesentlicher Vorteil gegenüber SSL, das keine vergleichbare Funktion vorsieht. Alice kann die verschiedenen Kanäle, die die Secure Shell aufbaut, beispielsweise nutzen, um gleichzeitig eine Datei herunterzuladen und Befehle an den Server zu schicken.

Transport Layer Protocol

Dieses Protokoll wird in der mittleren der drei Secure-Shell-Schichten abgearbeitet. Es hat zum einen die Aufgabe, eine sichere Verbindung zwischen Alice und dem Server aufzubauen, wobei ein Schlüssel ausgetauscht wird und sich der Server gegenüber Alice authentisiert. Die umgekehrte Authentisierung (Alice gegenüber dem Server) ist dagegen Sache des darüber liegenden Authentication Protocol. Sie erfolgt erst, wenn der sichere Verbindungsaufbau im Transport Layer Protocol erledigt ist. Zum anderen kümmert sich das Transport Layer Protocol darum, dass die Kommunikation zwischen Alice und dem Server nach erfolgtem Schlüsselaustausch verschlüsselt oder mit einer schlüsselabhängigen Hashfunktion abgesichert wird (auch beides gleichzeitig ist möglich). Innerhalb des Transport Layer Protocol kommen vier Arten von kryptografischen Verfahren zur Anwendung. Da die Secure Shell ein verhandlungsfähiges Protokoll ist, können alle vier benötigten Verfahren zwischen Alice und dem Server ausgehandelt werden. Folgende Algorithmen werden im RFC 4253 genannt:

- *Symmetrisches Verschlüsselungsverfahren*: Triple-DES muss, der AES (128 Bit Schlüssellänge) sollte unterstützt werden. Optional sind im Standard Blowfish, Twofish, Serpent, RC4, CAST und IDEA (teilweise mit unterschiedlichen Schlüssellängen) aufgeführt.
- *Schlüsselabhängige Hashfunktion*: Alle im RFC genannten Verfahren sind HMACs, basieren also auf kryptografischen Hashfunktionen. Die Unterstützung von SHA-1 ist vorgeschrieben, die von SHA-96 (dies ist eine inoffizielle SHA-Variante mit 96 Bit Hashwert-Länge) wird empfohlen. MD5 ist als Option vorgesehen.
- *Schlüsselaustausch*: Hier werden zwei Diffie-Hellman-Varianten genannt, die jeweils mit der kryptografischen Hashfunktion SHA-1 ergänzt werden (Letztere wird verwendet, um aus dem Masterschlüssel die Sitzungsschlüssel zu generieren). Diese beiden Varianten müssen unterstützt werden. Weitere Verfahren nennt der RFC nicht.
- *Asymmetrisches Verfahren*: Dies ist im Normalfall ein Verfahren für die digitale Signatur. DSA ist hier vorgeschrieben, RSA wird empfohlen. Für beide Verfahren kann optional das OpenPGP-Schlüsselformat verwendet werden.

Abgesehen von den genannten Verfahren können Alice und der Server auch andere einsetzen – vorausgesetzt, dass beide sie unterstützen. Insbesondere gibt es einige Verfahren, deren Verwendung innerhalb der Secure Shell in anderen RFCs beschrieben wird. So geht es in [RFC4432] um RSA als Schlüsselaustausch-Verfahren anstelle von Diffie-Hellman.

Der erste Schritt der Kommunikation im Transport Layer Protocol sieht vor, dass Alice und der Server die Verfahren aushandeln, die sie verwenden. Dabei können die beiden jeweils zwei symmetrische Verschlüsselungsverfahren und schlüsselabhängige Hashfunktionen festlegen – jeweils eines davon für den Weg von Alice zum Server sowie für den Weg vom Server zu Alice. Der Ablauf des darauffolgenden Schlüsselaustauschs hängt von den gewählten Verfahren ab. Der Standard schreibt vor, dass ein auf der Nutzung von Diffie-Hellman basierendes Vorgehen unterstützt wird, das wie folgt aussieht:

1. Alice generiert ein Diffie-Hellman-Schlüsselpaar. Sie wählt also einen Wert x (privater Schlüssel) und berechnet $g^x \pmod p$ (öffentlicher Schlüssel). Den öffentlichen Schlüssel schickt sie an den Server.
2. Der Server generiert ebenfalls ein Diffie-Hellman-Schlüsselpaar bestehend aus y (privater Schlüssel) und $g^y \pmod p$ (öffentlicher Schlüssel). Außerdem berechnet er $g^{xy} \pmod p$. Das Ergebnis wird K genannt und ist der gemeinsame geheime Schlüssel von Alice und dem Server. Nun signiert der Server seinen öffentlichen Diffie-Hellman-Schlüssel (zusammen mit einigen weiteren Informationen, die hier nicht interessieren), wobei er das RSA-Verfahren nutzt. Anschließend sendet er seinen öffentlichen Diffie-Hellman-Schlüssel mit der Signatur sowie seinen öffentlichen RSA-Schlüssel (beispielsweise als digitales Zertifikat) an Alice.
3. Alice überprüft die Signatur des öffentlichen Diffie-Hellman-Schlüssels. Dazu muss sie wissen, ob der öffentliche RSA-Schlüssel des Servers echt ist. Liegt ihr dieser als digitales Zertifikat vor, dann ist dies über die bekannten PKI-Mechanismen möglich. Falls nicht, kann Alice den öffentlichen RSA-Schlüssel auch mit einer manuell konfigurierten Schlüsselliste abgleichen. Eine dritte Möglichkeit wird in [RFC4255] beschrieben: Alice kann das Domain Name System (DNS) nutzen, um sich einen Hashwert des Schlüssels zu besorgen, mit dem sie eine Echtheitsprüfung vornehmen kann. Ist die Verifizierung der Signatur erfolgreich, dann hat Alice den Server authentifiziert. Sie berechnet nun $K = g^{xy} \pmod p$.

Am Ende dieses Ablaufs haben Alice und der Server den geheimen Schlüssel K gemeinsam. Mallory, der die gesamte Kommunikation abgehört hat, kann K nicht berechnen, sofern er das Diffie-Hellman-Verfahren nicht knacken kann. K dient als Masterschlüssel, der aus Sicherheitsgründen nicht direkt verwendet wird. Stattdessen leiten Alice und der Server daraus mithilfe der gewählten kryptografischen Hashfunktion (in der Regel SHA-1) vier Sitzungsschlüssel ab: einen

für die Verschlüsselung von Alice zum Server, einen für die Verschlüsselung vom Server zu Alice, einen für den Integritätsschutz von Alice zum Server sowie einen für den Integritätsschutz vom Server zu Alice. Wie bereits erwähnt, ist es möglich, für die Datenübertragung zum Server ein anderes Verfahren zu verwenden als in der Gegenrichtung. In allen vier Fällen kann auch auf die Nutzung eines Verfahrens verzichtet werden.

Wenn bei jedem Verbindungsaufbau neue Diffie-Hellman-Schlüssel eingesetzt werden, bietet die Secure Shell Forward Security und Backward Security. Durch die Unterstützung des Baby-Duck-Verfahrens (siehe Abschnitt 20.3.5) werden diese noch verstärkt. Eine weitere Verstärkung ergibt sich dadurch, dass die Secure Shell jeweils nach einer Stunde oder nach einem Gigabyte übertragener Daten einen erneuten Schlüsselaustausch vorsieht.

Authentication Protocol

Das Authentication Protocol der Secure Shell kommt in der obersten der drei Protokollschichten zum Einsatz. Sein Zweck ist die Authentifizierung des Anwenders (diese wird im Transport Layer Protocol ausgespart). Alice kann sich mit der Secure Shell auf folgende Weise gegenüber einem Server authentisieren:

- *Passwort*: Die einfachste Variante ist die Nutzung eines Passworts. Wenn im Transport Layer Protocol Verschlüsselung eingesetzt wird, dann ist diese Form der Authentifizierung sogar relativ sicher.
- *Öffentlicher Schlüssel*: Alice kann sich auch auf die gleiche Weise authentifizieren wie der Server, also mit einem öffentlichen Schlüssel. Dieser kann Bestandteil eines digitalen Zertifikats sein.
- *Einmal-Passwörter*: Alice kann auch Einmal-Passwörter nutzen. Als solche gelten in diesem Zusammenhang auch die von einem OTP-Token generierten Authentifizierungsinformationen.
- *GSS-API*: Alice kann auch die GSS-API nutzen, um sich beispielsweise über Kerberos oder NTLM zu authentisieren.

37.3.3 Bewertung der Secure Shell

Die Secure Shell ist eine kryptografische Mehrzweckwaffe, die vor allem im Unix-Umfeld eine gewisse Popularität erlangt hat. Zwar lassen sich die meisten Anwendungen dieses Protokolls auch über ein VPN oder eine SSL-Verbindung realisieren, doch das konnte die Secure Shell bisher nicht vom Markt verdrängen. Handwerklich gesehen ist die Secure Shell gute Arbeit. Das Protokoll ist übersichtlich aufgebaut, verhandlungsfähig, unterstützt zahlreiche Verfahren und bietet Forward Security sowie Backward Security. Ein Kritikpunkt ist jedoch die Anfälligkeit gegenüber Denial-of-Service-Attacken. Diese kommt dadurch zustande, dass der Server zuerst einen aufwendigen Schlüsselaustausch inklusive mehrere asym-

metrischer Operationen durchlaufen muss, bevor er Alice authentifiziert. Mallory kann den Server daher mit Verbindungsaufbau-Anfragen bombardieren, die dieser schnell nicht mehr bewältigen kann. Den Entwicklern der Secure-Shell-Protokolle war dieser Mangel bewusst. Sie empfehlen, ein solches Fluten mit Anfragen durch netzwerktechnische Maßnahmen zu verhindern.

37.4 Online-Banking mit HBCI

Beim Online-Banking sind die Sicherheitsrisiken so offensichtlich, dass bereits Mitte der neunziger Jahre die ersten kryptografischen Lösungen für diesen Zweck entstanden. Der heute in Deutschland relevante Standard, der allerdings längst nicht von allen Banken genutzt wird, hieß ursprünglich **HBCI** (**Homebanking Computer Interface**) und wurde inzwischen in **FinTS** (**Financial Transaction Services**) umbenannt [FinTS]. Es handelt sich dabei um ein kryptografisches Netzwerkprotokoll für Schicht 7 des OSI-Modells, das vom Zentralen Kreditausschuss (ZKA), also der Interessenvertretung der deutschen Banken, entwickelt wurde.

37.4.1 Der Standard

Die immer noch weit verbreitete Version 2.2 von HBCI stammt aus dem Jahr 2000. Mit der darauffolgenden Version FinTS 3.0, die 2002 erschien, erfolgte der besagte Namenswechsel. Inzwischen gibt es auch FinTS 4.0.

HBCI 2.2

Mit HBCI wollte der ZKA einen einheitlichen Standard für das Online-Banking in Deutschland schaffen. Dieser sollte auf der Seite des Anwenders durch den Webbrowser oder durch eine geeignete Finanzsoftware unterstützt werden. Alle gängigen Bankgeschäfte sollten damit möglich sein. Um auf die Anforderungen verschiedener Banken vorbereitet zu sein, definiert HBCI eine größere Anzahl von Geschäftsvorfällen – von der Kontostandsabfrage über die Einzelüberweisung bis zum Dauerauftrag. Inzwischen gibt es zudem Geschäftsvorfälle, die sich nicht auf Bankgeschäfte beziehen und HBCI damit auch für Behörden und andere Einrichtungen interessant machen. Für jeden Geschäftsvorfall sind spezielle Protokollnachrichten vorgesehen. Die meisten davon sind jedoch optional, um Implementierungen nicht unnötig aufzublähen.

HBCI sieht die Verwendung diverser kryptografischer Verfahren vor. Leider mussten die Entwickler hierbei zwei verschiedene Interessen unter einen Hut bringen: Der Verband der Privatbanken favorisierte eine Verschlüsselung ohne spezielle Hardware, während der Deutsche Sparkassen- und Giroverband auf eine bereits vorhandene Smartcard-Implementierung (die ZKA-Chipkarte) setzte. Deshalb gibt es zwei HBCI-Varianten: ohne und mit Smartcard.

In der HBCI-Variante ohne Smartcard wird zur Verschlüsselung ein Hybridverfahren – bestehend aus RSA und Triple-DES – verwendet. Außerdem werden digitale Signaturen eingesetzt, die ebenfalls mit dem RSA-Verfahren erstellt werden. Eine Nachricht kann auch mehrfach von unterschiedlichen Personen signiert sein. Der Einsatz mehrerer RSA-Schlüsselpaare pro Anwender ist möglich, oft wird dasselbe Paar jedoch sowohl zum Schlüsselaustausch als auch zum Signieren eingesetzt. Die RSA-Schlüssellänge ist mit 768 Bit für heutige Verhältnisse etwas knapp bemessen. Alices privater RSA-Schlüssel ist in einer Datei auf Festplatte, Memorystick oder einem anderen Datenträger gespeichert. Wie diese (in der Regel verschlüsselte) Speicherung aussieht, ist im Standard nicht festgelegt. Die meisten Implementierungen verwenden ein mit PKCS#5 vergleichbares Verfahren. HBCI sieht keine PKI-Unterstützung vor. Stattdessen muss Alice einen Hashwert ihres öffentlichen RSA-Schlüssels – diesen generiert sie selbst – per Post an die Kryptobank schicken. Die Bank verfährt ebenso in umgekehrter Richtung.

In der HBCI-Variante mit Smartcard kommt keine Public-Key-Kryptografie zum Einsatz, da die besagte ZKA-Chipkarte ursprünglich nicht genügend Speicherplatz bot. Stattdessen erhält Kundin Alice eine Smartcard, auf der ein 112-Bit-Schlüssel gespeichert ist. Dabei handelt es sich um einen Masterschlüssel, mit dem per Triple-DES-Verschlüsselung ein weiterer 112-Bit-Schlüssel übertragen wird, der als Sitzungsschlüssel für die weitere Kommunikation dient (für diese wird ebenfalls der Triple-DES verwendet). An die Stelle der digitalen Signatur tritt in diesem Fall ein schlüsselabhängiger Hashwert, der als CBC-MAC gebildet wird. Um den Ablauf möglichst ähnlich wie bei der HBCI-Variante ohne Smartcard zu gestalten, wird der schlüsselabhängige Hashwert nicht direkt aus dem Urbild berechnet, sondern aus einem Hashwert davon. Dieser Hashwert wird gebildet wie derjenige, der in der HBCI-Variante ohne Smartcard signiert wird. Mehrere schlüsselabhängige Hashwerte von verschiedenen Personen innerhalb einer Nachricht sind zulässig.

Beide HBCI-Varianten stellen einen Kompromiss dar: Im einen Fall wird auf Public-Key-Kryptografie verzichtet, um die ZKA-Chipkarte verwenden zu können. Im anderen Fall wird auf eine Smartcard verzichtet, um Public-Key-Kryptografie einsetzen zu können. Beide Varianten verzichten auf die Verwendung eines Passworts bzw. einer PIN und bieten dafür eine deutlich stärkere Authentifizierung über ein kryptografisches Verfahren. Dadurch wird Mallory beispielsweise das Phishing erschwert. Trotzdem wollten einige Banken lieber mit PINs und TANs (also mit Passwörtern und Einmal-Passwörtern) arbeiten. Eine nachgeschobene HBCI-Variante namens *HBCI+* ermöglicht dies. HBCI+ unterstützt PINs und TANs, sieht jedoch keinerlei Kryptografie innerhalb der Protokollnachrichten vor. Stattdessen werden unterhalb von HBCI+ stets HTTP und SSL (also HTTPS) eingesetzt.

FinTS 3.0

Trotz des neuen Namens unterscheidet sich FinTS 3.0 nicht grundlegend von HBCI 2.2. Aus kryptografischer Sicht sind es folgende Funktionen, die zusätzlich zu den bisher vorhandenen unterstützt werden:

- Die Nutzung digitaler Signaturen ist nun auch mit einer geeigneten Smartcard möglich. Erklärtes Ziel der FinTS-Entwickler ist es, dass sich die Variante durchsetzt.
- Die Nutzung eines PIN/TAN-Verfahrens (ähnlich dem von HBCI+) ist nun ein Teil des Standards.
- Die RSA-Schlüssellänge kann nun auch 1.024 bis 2.048 Bit betragen.

FinTS 4.0

Die 2004 erschienene vierte Generation (FinTS 4.0) brachte einige Neuheiten. Darin wurden alle internen Datenstrukturen komplett auf XML umgestellt. Dabei kommen insbesondere die Formate XML Signature und XML Encryption (Abschnitt 18.4.2) zum Einsatz. Eine weitere Neuerung in FinTS 4.0 sind sogenannte *verteilte Signaturen*. Diese ermöglichen es, dass von den mehreren zulässigen Signaturen in einer Nachricht alle bis auf eine zu einem späteren Zeitpunkt erstellt werden können.

37.4.2 Bewertung von HBCI und FinTS

HBCI bzw. FinTS sind sicherlich keine herausragenden Beispiele für modernes Kryptoprotokoll-Design. So könnte man bezüglich Verhandlungsfähigkeit, Unterstützung unterschiedlicher Verfahren, Sicherheit gegenüber Known-Key-Attacken sowie Forward Security und Backward Security einiges verbessern. Zugunsten der Pragmatik halte ich diese Versäumnisse aber für tolerierbar. Bemerkenswert an HBCI/FinTS ist in jedem Fall, dass es sich um einen in Deutschland entwickelten Standard handelt, der den Sprung zum Einsatz in der Praxis geschafft hat. Die Einführung verlief zunächst zwar eher schleppend, doch inzwischen unterstützen etwa 2.000 Banken in Deutschland HBCI oder FinTS. Dies entspricht etwa der Hälfte der deutschen Banken. Damit zählt Deutschland international zu den Vorreitern im Bereich des Online-Banking. Die Hoffnung, andere Länder könnten HBCI/FinTS übernehmen, hat sich leider bisher nicht erfüllt.

37.5 Weitere Krypto-Protokolle in Schicht 7

Schauen wir uns nun noch einige weitere Krypto-Protokolle an, die sich in Schicht 7 des OSI-Modells einordnen lassen.

37.5.1 Krypto-Erweiterungen für SNMP

SNMP (Simple Network Management Protocol) ist ein Protokoll aus der TCP/IP-Familie, das dem Management von Netzkomponenten dient. Mit SNMP kann ein Administrator Router, Switches, Drucker und andere Komponenten konfigurieren und warten, ohne physikalischen Zugang dazu zu haben. Es versteht sich von selbst, dass SNMP in puncto Sicherheit zu den kritischsten Internetprotokollen gehört. Schon allein das Abhören einer SNMP-Kommunikation kann einem Angreifer wie Mallory interessante Informationen verschaffen. Gelingt es Mallory gar, mittels SNMP eine Komponente zu manipulieren, dann kann dies einen enormen Schaden verursachen. Mallory kann auf diese Weise etwa Pakete auf einen ihm zugänglichen Router umleiten oder für beträchtliche Leistungseinbußen sorgen.

Leider sah SNMP – wie alle anderen frühen TCP/IP-Protokolle – zunächst keine Kryptografie vor. Spötter meinten gar, SNMP stehe für »Security's not my problem«. Lediglich eine unverschlüsselte Passwortabfrage gehörte zur ursprünglichen Ausstattung. Die großen Gefahren, die von einem unsicheren SNMP ausgehen, sorgten allerdings dafür, dass es vergleichsweise früh Sicherheitsüberlegungen zu diesem Thema gab [Stal99]. Der heute gültige Standard heißt SNMPv3. Er spezifiziert unter dem Namen **User Security Model** (**USM**) folgende Krypto-Mechanismen für SNMP [RFC3414]:

- Die SNMP-Kommunikation kann mit dem DES (oder mit einem anderen Verfahren, falls die Kommunikationspartner dieses unterstützen) verschlüsselt werden.
- Um die Authentizität und Integrität von Managementdaten zu gewährleisten, kann eine SNMP-Implementierung eine schlüsselabhängige Hashfunktion verwenden. Hierzu nennt der RFC zwei HMAC-Verfahren auf Basis von MD5 bzw. SHA-1 (ersteres muss, letzteres sollte eine Implementierung unterstützen). Alternativ kann auch ein anderes Hashverfahren oder eine digitale Signatur eingesetzt werden, falls die Kommunikationspartner dies unterstützen.
- Zusätzlich können Zeitstempel eingesetzt werden, um Replay-Attacken zu verhindern.

USM entspricht der Datenaustauschroutine eines sicheren Kanals. Über die Initialisierungsroutine macht der Standard dagegen keine Aussage. Die Entwickler gingen davon aus, dass die Anzahl der Netzkomponenten, die ein Administrator mit SNMP verwaltet, überschaubar ist und dass dieser die Schlüssel daher von Hand konfigurieren kann.

37.5.2 DNSSEC und TSIG

Das Domain Name System (DNS) habe ich Ihnen bereits in Abschnitt 3.4.4 als wichtigen Internetdienst vorgestellt. Zweck des DNS ist es, eine Internetadresse wie www.kryptoboerse.kl in die zugehörige IP-Adresse (z.B. 153.125.34.43) umzuwandeln. DNS ist ein vergleichsweise alter Internetdienst, dessen Netzwerkprotokolle in den achtziger Jahren entstanden sind. Wie damals üblich, sahen diese zunächst keine kryptografischen Mechanismen vor. Deshalb ist das DNS bis heute anfällig gegenüber Angriffen wie dem in Abschnitt 3.4.4 beschriebenen DNS-Spoofing.

Natürlich gibt es längst kryptografische Zusätze für das DNS. Diese verzichten zwar auf Verschlüsselung, da es hier nicht um vertrauliche Daten geht. Dafür spielen digitale Signaturen in diesem Zusammenhang eine wichtige Rolle. Diese können verhindern, dass Mallory eine DNS-Anfrage von Alice mit einer falschen IP-Adresse beantwortet. Der wichtigste Standard, der das DNS mit kryptografischen Funktionen ausstattet, nennt sich **DNSSEC**. Die erste Version davon wurde 1999 veröffentlicht [RFC2535]. Diese ursprüngliche Variante erwies sich allerdings als untauglich, da sie die gewaltigen Datenmengen im Internet nicht performant genug bewältigen konnte. Inzwischen gibt es eine überarbeitete DNSSEC-Fassung, die in vier RFCs beschrieben wird [RFC3658, RFC4033, RFC4034, RFC4035]. Einige weitere RFCs spezifizieren Erweiterungen. Leider ist DNSSEC zu komplex, um es an dieser Stelle im Detail zu behandeln. Ich will daher im Folgenden nur eine idealisierte Funktionsweise vorstellen, die auf einem vereinfachten DNS-Modell basiert. Wenn Sie mehr zum Thema wissen wollen, dann ist [Schwen] eine gute Quelle.

Funktionsweise des DNS

In unserer vereinfachten DNS-Welt nehmen wir an, Alice benötige die IP-Adresse zur WWW-Adresse www.kryptoboerse.kl. Um diese zu erfahren, nutzt Alice den für sie zuständigen DNS (dieser wird beispielsweise von ihrem Internet-Provider betrieben). Nun beginnt folgender Ablauf:

1. Kennt Alices DNS die zugehörige IP-Adresse nicht, dann stellt er eine Anfrage an den *Root-DNS*. Dieser ist in unserer idealisierten Umgebung ein DNS, der für weltweit alle IP-Adressen des Internets zuständig ist.
2. Kennt der Root-DNS die IP-Adresse nicht, dann verweist er Alices DNS auf den *kl-DNS* (dies tut er, indem er Alices DNS dessen IP-Adresse liefert). Der kl-DNS ist in unserer idealisierten Umgebung für alle Internetadressen zuständig, die mit kl enden.
3. Alices DNS stellt eine Anfrage an den kl-DNS.
4. Kennt der kl-DNS die IP-Adresse nicht, dann verweist er Alices DNS an den *kryptoboerse.kl-DNS*. Dieser ist in unserer idealisierten Umgebung für alle Internetadressen zuständig, die mit kryptoboerse.kl enden.

5. Alices DNS stellt eine Anfrage an den kryptoboerse.kl-DNS.
6. Der kryptoboerse.kl-DNS kennt die zu www.kryptoboerse.kl gehörende IP-Adresse und sendet sie an Alices DNS.

Durch dieses Prozedere erfährt Alices DNS bei Bedarf jede beliebige IP-Adresse im Internet. In der Praxis wird das Verfahren meist dadurch erheblich beschleunigt, dass jeder DNS auch eine große Anzahl von IP-Adressen kennt, die nicht in seinem Einflussbereich liegen.

Funktionsweise von DNSSEC

DNSSEC sieht vor, dass alle DNS-Antworten, die Alices DNS erhält, digital signiert sind (die Anfragen sind dagegen nicht signiert). Um dies zu ermöglichen, werden in unserer Umgebung sowohl der Root-DNS als auch der kl-DNS als auch der kryptoboerse.kl-DNS mit einem RSA-Schlüsselpaar ausgestattet. Alices DNS kennt lediglich den öffentlichen Schlüssel des Root-DNS. Eine mit DNSSEC abgesicherte Ermittlung einer IP-Adresse läuft wie folgt ab:

1. Alices DNS fragt den Root-DNS nach der IP-Adresse von www.kryptoboerse.de.
2. Falls der Root-DNS die angefragte IP-Adresse nicht kennt, verweist er Alices DNS auf den kl-DNS, indem er ihr dessen IP-Adresse liefert. Zusätzlich liefert er Alices DNS den Hashwert des öffentlichen Schlüssels des kl-DNS (in signierter Form).
3. Alices DNS verifiziert die Signatur des Hashwerts mithilfe des öffentlichen Schlüssels des Root-DNS (dieser ist ihm ja bekannt) und fragt anschließend den kl-DNS nach der IP-Adresse von www.kryptoboerse.de.
4. Falls der kl-DNS die angefragte IP-Adresse nicht kennt, verweist er Alice auf den kryptoboerse.kl-DNS, indem er ihr dessen IP-Adresse liefert. Zusätzlich liefert er Alices DNS seinen eigenen öffentlichen Schlüssel sowie den Hashwert des öffentlichen Schlüssels des kryptoboerse.kl-DNS (Letzteren in signierter Form).
5. Alices DNS verifiziert die Signatur mithilfe des öffentlichen Schlüssels des kl-DNS (dessen Echtheit kann er mithilfe des Hashwerts überprüfen, den er im zweiten Schritt erhalten hat) und fragt anschließend den kryptoboerse.kl-DNS nach der IP-Adresse von www.kryptoboerse.de.
6. Der kryptoboerse.kl-DNS kennt die angefragte IP-Adresse und sendet sie an Alices DNS (in signierter Form). Zusätzlich liefert er seinen eigenen öffentlichen Schlüssel.
7. Alices DNS überprüft die Signatur mithilfe des öffentlichen Schlüssels des kryptoboerse.kl-DNS (dessen Echtheit kann er mithilfe des Hashwerts überprüfen, den er im vierten Schritt erhalten hat). Im positiven Fall hat er nun die gewünschte IP-Adresse erfahren und kann zudem sicher sein, dass ihm Mallory keine falsche Adresse untergeschoben hat.

Auffällig an diesem Ablauf ist, dass der jeweils angefragte DNS nie den signierten öffentlichen Schlüssel des untergeordneten DNS liefert, sondern immer nur einen signierten Hashwert davon. Dies hat unter anderem den Vorteil, dass ein DNS die Hashwerte aller akzeptierten öffentlichen Schlüssel in einer Konfigurationsdatei speichern kann, wodurch sich der beschriebene Ablauf in vielen praktischen Fällen abkürzen lässt. So könnte Alices DNS in unserem Beispiel auch direkt den kryptoboerse.kl-DNS ansprechen, sofern er dessen IP-Adresse und Schlüssel-Hashwert kennt. Eine weitere Besonderheit von DNSSEC ist, dass ein DNS mehrere öffentliche Schlüssel besitzen kann, wobei einer davon als Masterschlüssel eingesetzt wird. Mit diesem signiert der DNS seine anderen öffentlichen Schlüssel, die wiederum zum Signieren der DNS-Antworten verwendet werden.

TSIG

TSIG (Transaction Signature) ist eine Alternative zu DNSSEC, die immer dann interessant ist, wenn nur einige wenige DNS-Systeme miteinander kommunizieren. TSIG ermöglicht es, die Kommunikation zwischen zwei DNS oder zwischen einem Server und einem DNS abzusichern. Im Gegensatz zu DNSSEC kommen dabei jedoch keine digitalen Signaturen zum Einsatz, sondern nur schlüsselabhängige Hashwerte. Die Schlüssel müssen von den zuständigen Administratoren manuell eingerichtet werden, da es kein Schlüsselaustausch-Protokoll gibt. Die verwendete schlüsselabhängige Hashfunktion ist ein HMAC-Verfahren auf Basis von MD5. TSIG ist in [RFC2845] spezifiziert. [RFC3645] ermöglicht die Nutzung der GSS-API für TSIG.

37.5.3 Kryptografie für SAP R/3

R/3 von SAP ist bekanntlich die führende Software für die Ressourcenplanung im Unternehmen. Aus unserer Sicht ist R/3 in erster Linie ein System von Clients und Servern, wobei die Clients über ein proprietäres Schicht-7-Protokoll mit den Servern kommunizieren. Unter diesem proprietären Protokoll wird TCP eingesetzt. Dass Mallory innerhalb eines R/3-Systems einigen Schaden verursachen kann, dürfte klar sein. Wenn es ihm gelingt, anstelle der berechtigten R/3-Anwenderin Alice auf den Server zuzugreifen, dann kann er dort beliebiges Unheil anrichten. Auch das Abhören von Protokollnachrichten ist eine Gefahr. R/3 unterstützte zwar von Anfang an Passwörter, aber – wie so viele Systeme – ursprünglich keine Kryptografie. Natürlich erkannte man bei SAP diese Gefahr und entwickelte entsprechende Gegenmaßnahmen. Die wichtigsten davon sind zwei kryptografische Schnittstellen [SNCSSF].

Secure Network Communication (SNC)

Die erste Krypto-Schnittstelle von R/3 trägt den Namen **Secure Network Communication** (**SNC**). SNC hat die Aufgabe, Daten vor dem Transport über das Netz zu verschlüsseln bzw. nach dem Empfang zu entschlüsseln. Anstatt das Rad neu zu erfinden, setzte SAP dabei auf Bewährtes: SNC basiert auf der GSS-API, die Sie bereits aus Abschnitt 24.3.6 kennen. R/3 kann so konfiguriert werden, dass zu versendende Daten zunächst an die SNC-Schnittstelle übergeben werden. Auf der anderen Seite dieser Schnittstelle muss sich dann ein Modul befinden, das die eingereichten Daten verschlüsselt (oder in anderer Form kryptografisch verarbeitet). Die verschlüsselten Daten werden anschließend vom Modul über die SNC-Schnittstelle an R/3 zurückgegeben.

Hat R/3-Anwenderin Alice ihre Daten über die SNC-Schnittstelle kryptografisch bearbeitet, dann verschickt sie diese. Auf Serverseite wiederholt sich der Vorgang in umgekehrter Form: Die empfangenen Daten werden über die SNC-Schnittstelle an ein Modul weitergereicht, das sie entschlüsselt und wieder zurückreicht. Der Server kann dann die entschlüsselten Daten in gewohnter Weise verarbeiten. Verschickt der Server eine Nachricht an Alice, dann erfolgt dieser Ablauf in umgekehrter Reihenfolge.

Secure Store & Forward (SSF)

Secure Store & Forward (**SSF**) ist der Name der zweiten Krypto-Schnittstelle von R/3. Über diese werden die Daten jedoch nicht für den Transport, sondern für das Speichern auf dem Datenbankserver kryptografisch bearbeitet. Ein hinter der Schnittstelle befindliches Modul verschlüsselt oder signiert die Daten hierzu und reicht sie im PKCS#7-Format zurück. In dieser Form können sie dann gespeichert werden. Beim Lesen der Daten wird dieser Vorgang in umgekehrter Form abgearbeitet.

37.5.4 SASL

Simple Authentication and Security Layer (**SASL**) ist ein Protokoll der Schicht 7, das die Authentisierung eines Clients gegenüber einem Server (und gegebenenfalls auch umgekehrt) ermöglicht [RFC4422]. SASL ist dazu gedacht, in ein beliebiges anderes Schicht-7-Protokoll integriert zu werden. Mit SASL können die beiden Kommunikationspartner außerdem die Art der Verschlüsselung in der weiteren Kommunikation vereinbaren. SASL realisiert also die Initialisierungsroutine eines sicheren Kanals, wobei die zugehörige Datenaustauschroutine nicht festgelegt ist. Entwickelt wurde SASL in erster Linie für IMAP, es kann jedoch auch für andere Protokolle verwendet werden – beispielsweise für SMTP, POP3, LDAP und XMPP. SASL ermöglicht eine Authentifizierung über Kerberos, GSS-API oder S/Key. Darüber hinaus sind weitere Authentifizierungsmechanismen möglich. In

[RFC2808] wird beispielsweise die Verwendung der SecurID-Karte mit SASL beschrieben, in [RFC2831] geht es um die Nutzung der Digest Access Authentication.

Die Idee einer vielseitig einsetzbaren, vom Rest des Protokolls unabhängigen Initialisierungsroutine ist sicherlich gut. Trotzdem ist SASL bisher kein großer Erfolg. Anstatt SASL nutzen die Hersteller von Krypto-Software bisher lieber komplette Protokolle wie SSL oder die Secure Shell. Diese sind zwar nicht so flexibel, haben sich aber in der Praxis bereits bewährt. SASL wird den großen Durchbruch daher voraussichtlich nie schaffen.

37.5.5 Sicheres NTP und sicheres SNTP

NTP (Network Time Protocol) und SNTP (Simple Network Time Protocol) sind Protokolle zur Zeitsynchronisation zwischen Computern. SNTP ist das neuere und zugleich einfachere unter den beiden Protokollen. Wie man sich leicht vorstellen kann, kann Mallory einen großen Schaden anrichten, wenn er NTP- bzw. SNTP-Nachrichten fälscht. Daher unterstützen die beiden Protokolle verschiedene kryptografische Sicherheitsmaßnahmen. Informationen dazu gibt es in [RFC5905] und [RFC5906].

38 Noch mehr Kryptografie in der Anwendungsschicht

In diesem Kapitel wollen wir uns einige weitere Kryptografie-Anwendungen anschauen, die man in Schicht 7 des OSI-Modells einordnen kann oder die in Anwendungsprogramme integriert sind.

38.1 Dateiverschlüsselung

Wie Sie bereits aus Abschnitt 3.3.2 wissen, zählt Laptop-Diebstahl in den Industrieländern zu den häufigsten Straftaten. Schon allein durch diese Tatsache dürfte klar sein, dass das Verschlüsseln von Dateien auf einem Laptop zu den wichtigsten Anwendungen der Kryptografie überhaupt gehört. Davon abgesehen gibt es für Alice auch gute Gründe, Dateien auf ihrem Büro-PC oder auf dem Server zu verschlüsseln – unter anderem, weil sie auch ihrem Administrator nicht immer vertrauen kann.

Das Verschlüsseln von Dateien ist also eine ebenso naheliegende wie wichtige Anwendung der Kryptografie. Beschränkt man sich auf das einfachste Szenario (Zugriff nur für eine Person, keine Smartcards, kein Recovery), dann ist die Dateiverschlüsselung zudem eine vergleichsweise einfache Krypto-Anwendung. So ist es auch kein Wunder, dass es längst Dutzende von Dateiverschlüsselungslösungen auf dem Markt gibt, von denen die meisten recht simple Programme sind. Ein Netzwerkprotokoll müssen diese in der Regel nicht unterstützen. Stattdessen genügt es, wenn eine derartige Software ein geeignetes Format für verschlüsselte Daten verwendet. Infrage kommen etwa PKCS#7 oder XML Encryption (siehe Abschnitt 18.4). Auch die vor allem für E-Mails gedachten Formate S/MIME und OpenPGP können an dieser Stelle eingesetzt werden. Viele Produkte setzen allerdings auf proprietäre Formate, da Interoperabilität bei der Dateiverschlüsselung nicht ganz so wichtig ist.

In der Regel bietet eine Software zur Dateiverschlüsselung einen zusätzlichen Menüpunkt im Kontextmenü einer Datei an. Alice genügt also ein Mausklick, um die Verschlüsselung zu starten. Im einfachsten Fall verwendet das Programm ein symmetrisches Verschlüsselungsverfahren (etwa den AES) und bildet den Schlüssel aus einem Passwort (beispielsweise nach PKCS#5). Das Entschlüsseln wird erneut über das Kontextmenü oder durch das doppelte Anklicken der Datei gestartet. Anschließend muss Alice das Passwort eingeben und erhält schließlich die Datei im Klartext zurück. Gute Programme zur Dateiverschlüsselung bieten neben diesem einfachen Ablauf weitere Funktionen an, wie etwa folgende:

- *PKI-Anbindung*: Eine PKI-Anbindung ermöglicht es Alice, ein asymmetrisches Verfahren zu nutzen, wobei das Schlüsselpaar von einer CA zertifiziert wurde. Dies hat den Vorteil, dass Alice für die Dateiverschlüsselung denselben Schlüssel oder dasselbe PSE verwenden kann wie für andere PKI-Anwendungen (etwa VPN oder E-Mail-Verschlüsselung). Da private Schlüssel innerhalb einer PKI oft auf Smartcards gespeichert werden, sollte ein Dateiverschlüsselungsprogramm auch eine Smartcard-Anbindung (über PKCS#11 oder MS-CAPI) unterstützen.
- *Gruppenfunktion*: Dieses Feature, das einige Produkte zur Dateiverschlüsselung bieten, sieht vor, dass Alice beim Verschlüsseln festlegen kann, wer die Datei entschlüsseln darf. Dies funktioniert wie bei einer E-Mail an unterschiedliche Empfänger. Das Programm verschlüsselt die Datei mit einem symmetrischen Verfahren und verschlüsselt anschließend den verwendeten Schlüssel mit den öffentlichen Schlüsseln aller zum Entschlüsseln Berechtigter.
- *Aktives Löschen*: Wie in Kapitel 17 beschrieben, sind Speicherleichen ein großes Problem bei der Dateiverschlüsselung. Gute Verschlüsselungsprodukte überschreiben daher entsprechende Speicherbereiche nach dem Verschlüsseln, damit keine verräterischen Informationen zurückbleiben.

- *Sicheres Löschen*: Dies ist eine kleine, aber nützliche Zusatzfunktion, die viele Produkte zur Dateiverschlüsselung bieten. Wenn Alice eine Datei löscht, dann wird diese (wie gewohnt) in den Papierkorb verschoben. In diesen Vorgang greift das Programm ein und verschlüsselt die Datei, bevor sie in den Papierkorb gelegt wird. Will Alice die Datei wiederherstellen, muss sie das entsprechende Passwort eingeben oder ihre Smartcard einsetzen. Der Vorteil dieses sicheren Löschens liegt auf der Hand: Wenn Alice eine vertrauliche Datei löscht, dann ist diese nicht mehr lesbar. Klaut Mallory ihren Laptop, dann wird er auch im Papierkorb nichts Verräterisches finden.
- *Recovery*: Bei keiner anderen Anwendung ist ein Schlüsselverlust so schmerzhaft wie bei der Dateiverschlüsselung. Deshalb bieten gute Produkte Key Recovery oder Message Recovery an (siehe Abschnitt 28.2).

Sie sehen: Obwohl die Dateiverschlüsselung zu den simpelsten Anwendungen der Kryptografie gehört, gibt es durchaus Kriterien, um die Spreu vom Weizen zu trennen.

38.2 Festplattenverschlüsselung

Einen Schritt weiter als die Dateiverschlüsselung geht die **Festplattenverschlüsselung**. Diese auch als **Full Disk Encryption** (**FDE**) bezeichnete Technik sieht vor, dass alle Daten auf Alices Festplatte automatisch verschlüsselt werden, sobald Alice sie dort ablegt. Die Entschlüsselung erfolgt ebenfalls automatisch, wenn Alice auf gespeicherte Daten zugreift. Weder beim Ver- noch beim Entschlüsseln muss Alice auf einen Button klicken oder in anderer Form eingreifen – beide Vorgänge erledigt ihr Rechner in transparenter Form. Lediglich beim Einloggen muss Alice aktiv werden, indem sie ihr Passwort zum Freischalten des Schlüssels oder ihre Smartcard eingibt.

Festplattenverschlüsselung ist vor allem für Laptops eine sinnvolle Sache. Es kann Alice schnell passieren, dass ihr mobiler Rechner gestohlen wird oder dass sie ihn irgendwo vergisst – ein gefundenes Fressen für Mallory (siehe Abschnitt 3.2.2). Die Festplattenverschlüsselung schafft Abhilfe. Entsprechende Produkte sind zwar aufwendiger herzustellen als Dateiverschlüsselungsprogramme, doch dafür sind sie für Anwenderin Alice sehr bequem – ein wichtiger Vorteil, denn Anwenderfreundlichkeit gehört zu den bedeutendsten Erfolgskriterien in der Kryptografie. Die meisten Produkte zur Festplattenverschlüsselung nutzen nur symmetrische Verschlüsselungsverfahren (meist den AES). Asymmetrische Verfahren sind in der Regel nicht notwendig, da der Verschlüssler mit dem Entschlüssler identisch ist. Einige Lösungen ermöglichen jedoch das verschlüsselte Ablegen des verwendeten symmetrischen Schlüssels mit einem asymmetrischen Verfahren, damit die Festplattenverschlüsselung in eine PKI integriert werden kann.

Natürlich sollte auch eine Festplattenverschlüsselungslösung eine Recovery-Funktion bieten – für den Fall, dass Alice ihr Passwort vergisst oder ihre Karte verliert. Bei vielen Produkten ist außerdem das Betriebssystem-Login mit dem Freischalten des Schlüssels zu einem Vorgang zusammengefasst, wobei dieselbe Smartcard oder dasselbe Passwort genutzt wird. Wenn Mallory einen durch ein solches Mini-Single-Sign-on gesicherten Rechner klaut, dann hat er doppeltes Pech: Er kann sich nicht einloggen und findet auf der Festplatte nur verschlüsselte Daten.

Das Betriebssystem wird normalerweise auf der Festplatte mitverschlüsselt. Dies ist sinnvoll, da die Betriebssystemdateien für Mallory durchaus interessante Inhalte haben können. Eine Festplattenverschlüsselungslösung muss daher ein vom Betriebssystem unabhängiges Modul enthalten, das mithilfe von Alices Passwort bzw. ihrer Smartcard den Schlüssel freischaltet und anschließend das Betriebssystem entschlüsselt. Ohne Schlüssel oder Karte kann Mallory nicht einmal das Betriebssystem starten (**Pre-Boot Authentication**).

Es gibt zwei Varianten der Festplattenverschlüsselung: Software- und Hardwarelösungen. Die Frage, ob man Kryptografie lieber in Hardware oder Software realisieren sollte (siehe Kapitel 23), spielt also auch in der Festplattenverschlüsselungstechnik eine wichtige Rolle. Bei einer Software zur Festplattenverschlüsselung ist die Verschlüsselungsfunktion ins Dateisystem integriert. Teilweise gehört eine solche Lösung zum Funktionsumfang eines Betriebssystems (z.B. Microsoft BitLocker [Schm12/2]), in anderen Fällen kann sie durch Produkte von Drittanbietern nachgerüstet werden. Der Softwareansatz hat folgende Vorteile:

- Alice benötigt keine zusätzliche Hardware, wodurch Softwarefestplattenverschlüsselung der billigere Ansatz ist.
- Gute Software zur Festplattenverschlüsselung bietet die Möglichkeit, zentrale Richtlinien und Konfigurationseinstellungen vorzugeben. Dies ist vorteilhaft, wenn beispielsweise ein Unternehmen seine Mitarbeiter mit Festplattenverschlüsselungssystemen ausstattet und eine einheitliche Nutzung durchsetzen will. Diese zentralen Vorgaben funktionieren bei Softwarelösungen naturgemäß besser als bei Hardwarelösungen.

Bei Hardwarelösungen zur Festplattenverschlüsselung ist ein Verschlüsselungsmodul in die Festplatte integriert. Man spricht in diesem Zusammenhang von einem **Self-Encrypting Drive** (**SED**). Auf dem Markt werden zahlreiche SEDs mit unterschiedlichen Features angeboten. Einige sind für den Einbau in einen PC gedacht (intern), andere werden über die USB-Schnittstelle angesprochen (extern). Externe SEDs verfügen teilweise über eine eingebaute Zehnertastatur für die PIN-Eingabe, manche haben sogar einen Fingerabdruckleser für die Freischaltung des Schlüssels integriert. Einbruchschutz, Einbruchevidenz (siehe Abschnitt 17.3.2) sind weitere mögliche Merkmale, genauso wie das automatische Löschen des Schlüssels bei physikalischer Beeinträchtigung der Platte. Der Hardwareansatz hat folgende Vorteile:

- Hardwarefestplattenverschlüsselung ist eindeutig der sicherere Weg, da die Verschlüsselung komplett vom Betriebssystem getrennt ist. Features wie eine eingebaute Tastatur oder Einbruchevidenz machen den Sicherheitsgewinn sogar noch größer.
- Es kommt vor, dass sich Lösungen zur Softwarefestplattenverschlüsselung nicht mit Virenscannern oder anderen Programmen vertragen, die ebenfalls im Dateisystem aktiv sind. Dieser Nachteil entfällt bei SEDs.
- Hardwarefestplattenverschlüsselung ist etwas performanter als die Softwarevariante.

Für die Festplattenverschlüsselung gilt also wie an anderer Stelle in der Kryptografie: Hardware ist sicherer, dafür aber auch teurer.

38.3 Code Signing

Wenn sich Alice für ihren PC eine Software aus dem Internet besorgt, dann besteht immer die Gefahr, dass Viren oder Trojaner darin versteckt sind. Ein Lösungsansatz ist hierbei das bereits behandelte Trusted Computing (Abschnitt 24.2). Darüber hinaus gibt es Methoden wie Virenscanner oder das Sandbox-Prinzip. Eine weitere Maßnahme zur Bekämpfung von Schadsoftware ist das **Code Signing**. Dieses sieht vor, dass der Urheber einer Software diese digital signiert. Wenn Alice eine Datei ausführen will, dann prüft sie zuvor die Signatur. Ist das Ergebnis positiv, dann kann Alice sicher sein, dass es sich um eine Originalsoftware des betreffenden Herstellers handelt. Falls Bösewicht Mallory Alice eine von ihm manipulierte Version der Software zuspielt, kann Alice dies an der falschen Signatur erkennen. Zwar sagt eine Signatur für sich genommen nichts über das Fehlen schädlicher Funktionen in einer Software aus. Wenn Alice jedoch dem Hersteller vertraut (bei einem namhaften Softwareanbieter kann sie dies in der Regel tun), dann kann sie ziemlich sicher sein, dass eine korrekt signierte Software keine bösen Überraschungen beschert.

Das Code Signing war in den neunziger Jahren die erste Anwendung für digitale Signaturen, die sich in der Praxis durchsetzte. Vor allem Java-Applets und ActiveX-Controls werden oft durch digitale Signaturen geschützt, da diese häufig über das Inhalt zur Ausführung kommen und Anwenderin Alice den Inhalt kaum beurteilen kann. Darüber hinaus kommt Code Signing häufig für Softwareupdates zur Anwendung. Die bedeutendste Code-Signing-Lösung ist **Authenticode** von Microsoft [Authen]. Dieses ist in die Windows-Betriebssysteme sowie in den Internet Explorer integriert. Authenticode verwendet das PKCS#7-Format für das Signieren von Software. Außerdem kommen X.509-Zertifikate zum Einsatz, die als **Software Publishing Certificates** (**SPC**) bezeichnet werden und ein spezielles Profil aufweisen. Wer als Hersteller eine Software signieren will, muss sich bei einer CA ein SPC besorgen. Im Zertifikatsspeicher von Windows, auf den auch der Internet Explorer zugreift, kann Alice die SPCs derjenigen Entwickler oder

Unternehmen speichern, denen sie vertraut. Darüber hinaus kann sie einem SPC auch dann vertrauen, wenn es von einer vertrauenswürdigen CA ausgestellt worden ist.

38.4 Bezahlkarten

Bargeld ist etwas Umständliches. Als Kunde ärgert man sich, wenn der Händler auf große Scheine nicht herausgeben kann oder wenn die passende Münze für den Automaten nicht zur Hand ist. Für den Händler hat Bargeld sogar noch größere Nachteile. Er muss sich mit ausreichend Wechselgeld versorgen, sich vor Dieben schützen und jeden Abend die eingenommenen Scheine und Münzen zur Bank bringen. Weil Bargeld somit einige unbestreitbare Nachteile hat, kam bereits in den achtziger Jahren die Idee auf, Chipkarten an dessen Stelle einzusetzen. Im Folgenden will ich diese Technik als **Bezahlkarten** bezeichnen. Eine Bezahlkarte ist eine Karte, auf der ein Geldbetrag gespeichert ist, der nur nach geeigneter Authentifizierung verändert werden kann. So ist beispielsweise die Bank berechtigt, den Betrag auf Alices Karte zu erhöhen, wenn Alice eine entsprechende Einzahlung geleistet hat. Andererseits darf ein Händler den Betrag auf der Karte heruntersetzen, nachdem er Alice eine Ware verkauft hat. Den Differenzbetrag lässt sich der Händler gutschreiben. Kreditkarten und Debitkarten zählen nicht zu den Bezahlkarten nach dieser Definition, da sie keine Beträge speichern, sondern lediglich eine Überweisung der jeweiligen Summe veranlassen. Bei einem solchen Ablauf entstehen höhere Transaktionskosten als bei einer Bezahlkarte. Bezahlkarten sind daher auch für kleinere Beträge geeignet. Projekte, die Bezahlkarten realisieren, gibt es zu Dutzenden. Leider herrscht dabei vom kanadischen Dexit bis zum britisch-amerikanischen Mondex durchweg Inkompatibilität. Wir schauen uns im Folgenden die drei wichtigsten Systeme aus dem deutschsprachigen Raum an.

Geldkarte

Die **Geldkarte** ist eine deutsche Bezahlkarte. Es gibt sie kontogebunden und kontoungebunden. Im ersteren Fall ist der Geldkartenchip auf der Kundenkarte einer Bank oder auf einer Maestro-Karte aufgebracht. Alice kann ihn aufladen, indem sie am Automaten Geld von ihrem Konto auf die Karte transferiert. Zum Aufladen einer kontoungebundenen Karte benötigt Alice entweder Bargeld oder eine weitere Karte (z. B. Kreditkarte). Das Bezahlen funktioniert mit beiden Kartentypen gleich.

Die Geldkartenspezifikation sieht drei Typen von Terminals vor: Bankterminals, an denen Alice ihre Karte aufladen kann, Händlerterminals, an denen Alice bezahlen kann, und Sonderfunktionsterminals, mit deren Hilfe die Bank einige spezielle Aktionen durchführen kann (etwa Entladen der Karte oder Überprüfen der Echtheit). Eine Bank, die Geldkarten herausgibt, hat einen sogenannten

Masterschlüssel, der in einer speziellen Hardware gespeichert ist, auf die das Bankterminal zugreifen kann. Bevor die Bank eine Geldkarte an Alice ausgibt, bildet sie einen Hashwert aus dem Masterschlüssel und der Kartennummer, der als *Kartenschlüssel* auf der Karte abgelegt wird. Als kryptografische Hashfunktion wird ein DES-basiertes Verfahren verwendet. Will Alice ihre Geldkarte am Bankterminal aufladen, dann authentifizieren sich Karte und Terminal gegenseitig mithilfe des Kartenschlüssels (mit einem Challenge-Response-Verfahren auf Basis des DES). Zusätzlich muss Alice noch eine PIN eingeben, ohne die die Karte nicht aufgeladen werden kann. Nach erfolgter Authentifizierung gibt Alice den entsprechenden Betrag ein, der dann beispielsweise von ihrem Konto abgebucht und auf der Karte gespeichert wird.

Mit der geladenen Geldkarte kann Alice beispielsweise im Parkhaus, am Fahrkartenautomat oder im Supermarkt bezahlen. Zum Bezahlen steckt sie die Karte in das Händlerterminal. Natürlich ist auch vor dem Heruntersetzen des Betrags durch das Händlerterminal eine Authentifizierung notwendig. Dies geschieht mithilfe eines weiteren Kartenschlüssels, der aus einem *Händlerschlüssel* generiert wird. Diesen wiederum hat jeder Händler auf einer speziellen *Händlerkarte* gespeichert. Um die Sache so einfach wie möglich zu machen, ist bei der Geldkarte zum Bezahlen keine PIN notwendig. Dies heißt auch, dass eine verlorene Karte verlorenes Geld bedeutet.

Aus Performanzgründen und weil man keine zu aufwendige Infrastruktur haben will, wird bei der Geldkarte auf Public-Key-Kryptografie verzichtet. Dadurch kommt der sicheren Speicherung der geheimen Schlüssel auf der Smartcard und in spezieller Hardware eine besonders wichtige Rolle zu. Außerdem wird für jedes Kartenexemplar ein Schattenkonto geführt, durch das sich sechs Jahre lang alle Zahlungen nachvollziehen lassen. Das Schattenkonto hat unter anderem den Vorteil, dass sich gefälschte Geldkarten zweifelsfrei erkennen lassen.

Obwohl die Geldkarte inzwischen ihren 15. Geburtstag gefeiert hat, gibt es bisher wenig Grund zum Jubeln. Die Akzeptanz dieses Zahlungsmittels ist nach wie vor gering, obwohl es praktisch jeder in seiner Brieftasche mit sich trägt. Während der Einzelhandel die Geldkarte bisher kaum unterstützt, gibt es immerhin zahlreiche Parkschein-, Fahrkarten-, Briefmarken- und Zigarettenautomaten, an denen man auf diese Weise bezahlen kann.

Quick

Quick ist das österreichische Gegenstück zur deutschen Geldkarte. Die Funktionsweise ist ähnlich, doch die beiden Systeme sind nicht kompatibel. Wie die Geldkarte verzichtet Quick auf eine PIN-Eingabe beim Bezahlen. Quick ist heute auf allen österreichischen Maestro-Karten vertreten. Dazu kommen auch hier kontounabhängige Karten.

CASH

CASH ist die bedeutendste schweizerische Bezahlkarte. Die Funktionsweise ähnelt Quick und der Geldkarte, ohne dass eine Kompatibilität gegeben ist. Es ist auf den Schweizer Maestro-Karten verfügbar. Wie die beiden Gegenstücke aus Deutschland und Österreich kommt auch CASH nur schwer in die Gänge.

38.5 Online-Bezahlsysteme

Mithilfe der Kryptografie lassen sich Protokolle entwickeln, mit denen Alice Geld über das Netz an Online-Händler Otto transferieren kann (Online-Bezahlen). Es ist wohl unnötig zu erwähnen, dass es sich dabei um eine äußerst wichtige Anwendung der Kryptografie handelt. Während wir sonst davon ausgehen, dass Mallory der einzige Bösewicht ist, müssen wir an dieser Stelle zusätzlich in Betracht ziehen, dass auch Alice und Online-Händler Otto nicht ehrlich sind. Alice hat naturgemäß Grund dazu, sich vor dem Bezahlen zu drücken, während Otto gerne mehr kassieren würde, als ihm zusteht. Leider gibt es kein simples Protokoll, das einen Geldtransfer ermöglicht, ohne dass Alice, Otto und Mallory betrügen könnten – wir müssen also etwas tiefer in die kryptografische Trickkiste greifen. Insbesondere benötigen wir neben Alice und Otto als dritte Instanz eine Zentrale (z. B. eine Bank oder eine Clearing-Stelle), die den Geldtransfer überwacht.

Ende der neunziger Jahre, als das Internet seinen Durchbruch schaffte, schossen **Online-Bezahlsysteme** wie Pilze aus dem Boden. Sie wurden zu einem willkommenen Betätigungsfeld für Kryptografen. 15 Jahre später kann man auf die damaligen Entwicklungen nur noch mit einem mitleidigen Lächeln zurückblicken. Das Online-Bezahlen erwies sich nämlich nicht als die erwartete Goldgrube, sondern entwickelte sich zu einem Fiasko ersten Ranges. Von Ecash und Cybercash über Millicent und Micromint bis zu Clickshare und First Virtual floppte ein Online-Bezahlsystem nach dem anderen und ließ ratlose Anbieter sowie gleichgültige Kunden zurück. Selbst das von Branchengrößen wie Microsoft und IBM unterstützte Netzwerkprotokoll SET, das Kreditkartenzahlungen über das Internet ermöglichte, entwickelte sich zur digitalen Investitionsruine. In der Liste der größten Krypto-Flops (siehe Abschnitt 40.5) nehmen die Online-Bezahlsysteme der ersten Generation daher einen Ehrenplatz ein.

Die Gründe für das Scheitern sind vielschichtig. Offensichtlich waren die meisten frühen Online-Bezahlsysteme zu kompliziert. In der Kryptografie setzen sich erfahrungsgemäß vor allem einfache, pragmatische Lösungen durch, und darauf achteten die Anbieter zu wenig. Es zeigte sich sogar, dass die Rolle der Kryptografie beim Online-Bezahlen deutlich überschätzt wurde. Neuere Online-Bezahlsysteme verzichten daher größtenteils auf digitale Signaturen, kryptografische Authentifizierungsmaßnahmen und komplexe Protokolle. Stattdessen gehen sie schlicht davon aus, dass die meisten Kunden und Händler ehrlich sind und dass sich eventuelle Betrugsfälle mit Kulanz regeln lassen. Praktisch alle Online-

Bezahlsysteme kann man in drei Klassen einteilen: Kreditkartensysteme, Kontensysteme und Bargeldsysteme. Im Folgenden schauen wir uns diese etwas genauer an.

38.5.1 Kreditkartensysteme

Derzeit sind allein in Deutschland über 40 Millionen Kreditkarten im Umlauf. Anstatt das Rad (sprich: Online-Bezahlsystem) neu zu erfinden, lohnt es sich daher, auf diese etablierte Zahlungsform mit ihrer bereits vorhandenen Infrastruktur zurückzugreifen. Online-Bezahlsysteme, die auf Kreditkarten basieren, werden als **Kreditkartensysteme** bezeichnet. Zu den Kreditkartensystemen gehört somit auch eines der einfachsten Online-Bezahlsysteme. Dieses sieht folgenden Ablauf vor:

1. Alice schickt ihre Kreditkartennummer (und das Gültigkeitsdatum, was ich im Folgenden stillschweigend als Teil der Nummer betrachte) an Online-Verkäufer Otto, beispielsweise indem sie diese in ein Formular auf einer Webseite eintippt.
2. Otto gibt die Nummer mit dem Betrag an die Zentrale (Clearing-Stelle des Kreditkarten-Unternehmens) weiter.
3. Die Zentrale bucht den entsprechenden Betrag von Alices Konto ab und überweist ihn auf das Konto von Otto.

Wenn die Verbindung zwischen Alice und Otto verschlüsselt ist (in der Praxis meist mit SSL), dann kann Mallory die Kreditkartennummer nicht abhören. Genau diese Angst vor einem Abhörer wie Mallory war einer der wesentlichen Beweggründe für die Entwicklung des SSL-Protokolls. Leider ist das Übermitteln einer Kreditkartennummer gleichzeitig auch ein Fall, in dem SSL an seine Grenzen stößt. So kann Alice im Nachhinein bestreiten, eine Zahlung in Auftrag gegeben zu haben, weil SSL keine digitalen Signaturen für Nutzdaten unterstützt. Verkäufer Otto kann die Summe in Alices Zahlungsauftrag ändern und so mehr Geld kassieren als vorgesehen. Dies liegt daran, dass SSL als Schicht-4-Protokoll nur einen sicheren Tunnel bietet, sich aber nicht um die weitere Verarbeitung der Daten kümmert. Aus demselben Grund kann sich Mallory Alices Kreditkartennummer merken und später Zahlungsaufträge erfinden. Zwar sind dies zum großen Teil keine Probleme des Online-Bezahlens, sondern des Kreditkartenwesens allgemein (auch an der Tankstelle kann es vorkommen, dass sich der Verkäufer eine Kreditkartennummer notiert), doch im großen, anonymen Internet ist das Vertrauen, das man einem Händler entgegenbringt, normalerweise geringer.

Ende der neunziger Jahre entstanden Online-Bezahlsysteme, die die Online-Nutzung von Kreditkarten sicherer machen sollten. Diese arbeiteten in Schicht 7 des OSI-Modells, sahen digitale Signaturen vor und verbargen Alices Kreditkartennummer mittels Verschlüsselung vor den Augen des Händlers. Unter der Mit-

wirkung von Unternehmen wie Microsoft, IBM, Visa und Mastercard entstand sogar ein spezielles Netzwerkprotokoll zu diesem Zweck, das den Namen **SET** (**Secure Electronic Transaction**) trug. Wie so viele andere Ideen für das Online-Bezahlen floppte jedoch auch SET – unter anderem, weil viele es als zu kompliziert empfanden. Daher ist die SSL-verschlüsselte Übertragung der Kreditkartennummer derzeit das Einzige, was in Sachen Kreditkartensysteme verfügbar ist. Den Betrugsmöglichkeiten bei dieser Zahlungsmethode begegnen die Anbieter nicht mit Kryptografie, sondern auf die inzwischen bewährte Weise: Hoffen, dass es nur selten passiert, und bei Bedarf den Schaden tragen.

38.5.2 Kontensysteme

Die meisten Online-Bezahlsysteme folgen einem recht einfachen Ansatz: Alice und Otto haben ein Konto beim selben Anbieter. Wenn Alice bei Otto etwas kauft, dann veranlasst sie eine entsprechende Umbuchung. Zahlungssysteme, die auf diesem Ansatz beruhen, werden als **Kontensysteme** bezeichnet. Die Kommunikation zwischen Alice und Otto sollte über einen sicheren Kanal ablaufen. Im Idealfall ist die Zahlungsanweisung signiert. Bei Kontensystemen sind die Transaktionskosten im Vergleich zu Kreditkartensystemen niedriger. Sie sind daher auch für Geldsummen unterhalb der Fünf-Euro-Grenze (sogenanntes **Micropayment**) geeignet.

In der Anfangszeit des Internets waren verschiedene Kryptografen maßgeblich an der Entwicklung von Kontensystemen beteiligt. Viele der damaligen Lösungen (etwa Netbill und Cybercoin) basierten daher auf relativ komplexen kryptografischen Netzwerkprotokollen. Trotzdem oder gerade deswegen floppte praktisch die gesamte erste Kontensystemgeneration. Nur mühsam konnten sich in den letzten Jahren einige neue Angebote etablieren. Diese sind kryptografisch nicht besonders anspruchsvoll.

Ein typisches Beispiel ist **Click&Buy** von der Firma Firstgate. Will Alice eine Überweisung vornehmen, dann verbindet sie sich per SSL mit einem Server des Anbieters und erteilt dort den entsprechenden Auftrag. Digitale Signaturen kommen (abgesehen vom SSL-Handshake) nicht zum Einsatz. Ähnlich funktioniert **PayPal**, das von eBay betrieben wird. Ein weiteres Online-Bezahlsystem ist **T-Pay**, das von der Deutschen Telekom angeboten wird. Es gibt unterschiedliche Varianten davon: Alice kann ihren Zahlungsauftrag als Überweisung aufgeben oder eine gebührenpflichtige Nummer wählen oder den Betrag mit der Telefonrechnung bezahlen oder eine Kreditkarte bzw. ein Lastschriftverfahren nutzen. Je nach Variante ist T-Pay also ein Konten- oder ein Kreditkartensystem. Außer SSL kommt keine Kryptografie zum Einsatz.

In Österreich recht bekannt ist **Paybox**. Es hat die Besonderheit, dass ein Handy in eine Zahlung involviert ist. Wenn Alice Paybox nutzen will, dann muss sie sich zunächst beim Betreiber anmelden und dort eine Kontonummer samt Ein-

zugsermächtigung hinterlegen. Dafür erhält sie ein Passwort in Form einer PIN. Ein Bezahlvorgang sieht dann so aus:

1. Alice übergibt ihre Mobiltelefonnummer an Händler Otto.
2. Otto schickt die Transaktion über eine sichere Datenverbindung zum Betreiber.
3. Der Betreiber ruft Alice unter der angegebenen Nummer an.
4. Alice gibt die Transaktion durch Eingabe ihrer PIN frei.
5. Der Betreiber zieht das Geld per Lastschriftverfahren ein und leitet es an Otto weiter.

38.5.3 Bargeldsysteme

Bei Kreditkarten- und Kontensystemen lässt sich der Weg des Geldes prinzipiell zurückverfolgen. Die Bank oder die Kreditkartenfirma weiß schließlich, wie viel Geld Alice an Otto übergeben hat. Zwar gibt es keine Zentralstelle, die einen Überblick über alle Zahlungen hat, der gläserne Konsument ist hierdurch dennoch ein gutes Stück nähergerückt. Um diese Überwachungsmöglichkeit zu umgehen, benötigt man ein digitales Äquivalent zum Bargeld, das eine anonyme Geldübergabe ermöglicht. Zahlungssysteme, die so etwas realisieren, werden oft als Electronic Cash oder digitales Bargeld bezeichnet. Ich werde den etwas weniger plakativen Ausdruck **Bargeldsystem** verwenden.

Das Design eines Bargeldsystems ist nicht gerade trivial. Digitale Daten haben nun einmal keinen Materialwert und sind leicht zu kopieren – Geldscheine lassen sich so schwer realisieren. Zwar kann der Herausgeber digitale Geldscheine digital signieren, doch dadurch wird niemand daran gehindert, einen Geldschein samt Signatur zu kopieren (**Double-Spending-Problem**). Eine Lösungsmöglichkeit besteht darin, Smartcards (oder eine andere Art von Hardware) zu nutzen – eine Karte ist schließlich nicht ohne Weiteres kopierbar. Es ist jedoch auch möglich, das Double-Spending-Problem ohne Hardware zu lösen. Dazu ist der Einsatz eines speziell konzipierten kryptografischen Protokolls (**Bargeld-Protokoll**) notwendig. Mithilfe eines solchen überwacht eine Zentrale den Geldfluss, ohne dabei Zugriff auf die Identitäten der Beteiligten zu haben. Die Idee eines Bargeld-Protokolls ist eng mit dem Namen des US-Kryptografen David Chaum verbunden, der sich bereits in den achtziger Jahren zahlreiche Verfahren zu diesem Thema patentieren ließ. Um seine Erfindungen zu vermarkten, gründete Chaum die Firma Digicash, die das Bargeldsystem **Ecash** auf den Markt brachte.

Ecash war aus kryptografischer Sicht zweifellos eine sehr interessante Lösung. Das kryptografische Protokoll, das es vorsah, dürfte zu den komplexesten gehören, die je kommerziell implementiert wurden. Im Mittelpunkt des Protokolls standen sogenannte blinde Signaturen, die es einer Bank ermöglichten, digitale Geldscheine zu signieren, ohne die Seriennummer der Scheine sehen zu können. Gab Alice einen digitalen Geldschein aus, dann tauschte der Händler diesen anschließend bei der Bank ein, woraufhin diese die Seriennummer auf eine

Sperrliste setzte. Das erneute Verwenden eines Scheins war nicht vorgesehen und aufgrund von Sperrlistenabfragen bei der Zahlung auch nicht möglich. Damit war das Double-Spending-Problem gelöst. Die gewünschte Anonymität war gewährleistet, da die Bank die Seriennummern der ausgegebenen Geldscheine nicht kannte und dadurch den Lauf eines Geldscheins nicht verfolgen konnte. Leider war Ecash – wie so viele Online-Bezahlsysteme der damaligen Zeit – wirtschaftlich ein Flop. Immer wieder gab es technische Probleme, für Laien war das Funktionsprinzip kaum zu verstehen. Darüber hinaus war die Nachfrage nach digitalem Bargeld längst nicht so hoch, wie viele es erwartet hatten. Viele Online-Bezahler verzichteten offensichtlich auf die vollständige Anonymität, wenn sie dadurch ein paar Mausklicks sparen konnten. 1998 ging Chaums Firma Digicash pleite.

38.6 Elektronische Ausweise

Ein **elektronischer Ausweis** ist ein Ausweis (z. B. Personalausweis, Reisepass, Dienstausweis oder Krankenversichertenkarte), der mit einem Computerchip ausgestattet ist. Ein elektronischer Ausweis ist damit ein Spezialfall einer Chipkarte. Der Chip kann hierbei (wie bei einer Chipkarte allgemein) als reiner Speicherbaustein oder als kleiner Computer mit eigener Rechenlogik realisiert sein. Eine Übersicht zum Thema elektronische Ausweise gibt [Schm09/2]. Kryptografisch interessant sind vor allem elektronische Ausweise, die einen Smartcard-Chip aufweisen. Diese haben oft Kreditkartengröße, kommen aber auch in anderen Formfaktoren vor. Zwischen der Kryptografie und elektronischen Ausweisen besteht eine gegenseitige Abhängigkeit. Auf der einen Seite lässt sich die Sicherheit eines elektronischen Ausweises meist nur mithilfe der Kryptografie gewährleisten. Dies erfolgt in der Regel auf zwei Arten:

- Ein elektronischer Ausweis kann auf seinem Chip einen privaten Schlüssel speichern, den er für ein Challenge-Response-Verfahren nutzt. Wenn man davon ausgeht, dass jeder Ausweis seinen eigenen, unauslesbar gespeicherten Schlüssel hat, dann lässt sich auf diese Weise schnell und automatisch die Echtheit eines Ausweises prüfen. Damit dies funktioniert, muss dem Prüfgerät der jeweils zugehörige öffentliche Schlüssel bekannt sein. Dies lässt sich mit einer Public-Key-Infrastruktur gewährleisten.
- Die Daten im Chip eines elektronischen Ausweises lassen sich durch eine digitale Signatur vor Veränderungen schützen. Diese Signatur wird typischerweise von der Ausweisbehörde angefertigt. Deren öffentlicher Schlüssel muss den Prüfgeräten bekannt sein, was durch eine Public-Key-Infrastruktur gewährleistet werden kann. Im Chip eines elektronischen Ausweises sind normalerweise dieselben Daten gespeichert, die auch aufgedruckt sind – also Name, Geburtsdatum und ähnliche Informationen. Zusätzlich speichern manche elektronischen Ausweise einen biometrischen Referenzwert (z. B. einen Fingerabdruck).

Mit diesem lässt sich verhindern, dass jemand anderer als der rechtmäßige Inhaber den Ausweis nutzt. Wenn beispielsweise ein Beamter an der Grenze Alices Ausweis kontrolliert, kann er deren Fingerabdruck einlesen und einen Vergleich mit dem Referenzwert auf dem Chip durchführen.

Nutzt eine Ausweisbehörde diese beiden kryptografischen Maßnahmen in sachgerechter Weise, dann sind elektronische Ausweise sehr schwer zu fälschen und sehr schwer zu manipulieren. Kommt zusätzlich Biometrie in der beschriebenen Weise zum Einsatz, dann ist für Bösewicht Mallory auch die Nutzung eines geliehenen oder gestohlenen Ausweises nahezu unmöglich. Man benötigt nicht einmal einen wachsamen Grenzbeamten, um derartigen Missbrauch zu erkennen, denn sowohl Fälschungen als auch Manipulationen als auch eine unberechtigte Ausweisnutzung sind maschinell erkennbar. Es dürfte also klar sein, dass elektronische Ausweise die Sicherheit im Ausweiswesen deutlich erhöhen können.

Ein elektronischer Ausweis profitiert jedoch nicht nur von der Kryptografie, sondern kann umgekehrt auch deren Nutzung für Zwecke unterstützen, die nichts mit den ursprünglichen Anwendungsgebieten von Ausweisen zu tun haben. Insbesondere lassen sich auf dem Chip eines elektronischen Ausweises private Schlüssel für beliebige Zwecke unterbringen. Ausweisinhaberin Alice kann ihren Ausweis dadurch zusätzlich zum Verschlüsseln (genauer gesagt: zum Entschlüsseln), zur Online-Authentifizierung und für das digitale Signieren nutzen. Voraussetzung dafür ist eine entsprechende Infrastruktur, die beispielsweise Online-Behördengänge oder die Dateiverschlüsselung mit einem elektronischen Ausweis erlaubt.

38.6.1 Elektronische Reisepässe

Die größte Verbreitung unter den elektronischen Ausweisen haben derzeit **elektronische Reisepässe**. Für diese gibt es einen Standard namens MRTD (Machine Readable Travel Documents), der von der Luftfahrtorganisation ICAO entwickelt wurde und weltweit anerkannt ist [MRTD]. Der MRTD-Standard sieht vor, dass auf einem Reisepass ein Chip enthalten ist. Dieser wird kontaktlos angesprochen und kann biometrische Referenzwerte speichern. Praktisch alle Industrieländer (einschließlich Deutschland, Österreich und der Schweiz) stellen seit einigen Jahren nur noch MRTD-Reisepässe aus. Die weite Verbreitung von MRTD-Reisepässen ist unter anderem darauf zurückzuführen, dass die USA seit dem 26. Oktober 2005 Ausländern eine Einreise ohne Visum nur noch mit MRTD-Reisepass erlauben (eine zusätzliche Voraussetzung ist, dass der Einreisende Bürger eines Staats ist, der im sogenannten Visa-Waiver-Programm der USA anerkannt ist). Diese Regelung entstand als Folge der Terroranschläge vom 11. September 2001.

Im Vergleich zu anderen elektronischen Ausweisen ist der elektronische Reisepass eher simpel aufgebaut. Die Daten auf dem Reisepass-Chip sind digital signiert, und der Chip enthält einen privaten Schlüssel für ein Challenge-Response-

Verfahren. Optional kann ein Reisepass auch die Echtheit des Prüfgeräts per Challenge-Response überprüfen. Die Abfrage der Chip-Inhalte (inklusive der biometrischen Daten) bei einer Überprüfung des Passes erfolgt nach einer Authentifizierung (mit einem symmetrischen Verfahren) und symmetrisch verschlüsselt, wobei die beiden verwendeten Schlüssel von den aufgedruckten Daten ableitbar sind. Durch die Authentifizierung wird gewährleistet, dass Angreifer Mallory Alices Reisepass nicht unbefugt auslesen kann, während diese das Dokument in der Jackentasche trägt (da es sich um einen kontaktlosen Chip handelt, wäre dies sonst möglich). Durch die Verschlüsselung kann Mallory nicht mitlesen, wenn ein befugtes Prüfgerät den Inhalt von Alices Reisepass (kontaktlos) ausliest. Wenn Mallory jedoch den Reisepass selbst in die Hände bekommt, dann kann er die benötigten Schlüssel aus den aufgedruckten Daten berechnen und den Inhalt auslesen.

38.6.2 Elektronische Personalausweise

Neben elektronischen Reisepässen sind vor allem **elektronische Personalausweise** von Bedeutung. Einige Staaten – darunter Deutschland, Finnland, Belgien, Malaysia und Spanien – haben einen solchen bereits eingeführt. Im Gegensatz zu elektronischen Reisepässen gibt es für elektronische Personalausweise keinen international anerkannten Standard. Die diversen nationalen Lösungen, die bereits in Betrieb sind oder geplant werden, sind daher größtenteils nicht miteinander kompatibel. Dafür sind elektronische Personalausweise meist leistungsfähiger als elektronische Reisepässe. So sind die verwendeten Chips in der Regel mit mehreren privaten Schlüsseln ausgestattet, die sich zum Verschlüsseln, Signieren und Online-Authentifizieren verwenden lassen.

In einigen Ländern gibt es noch weitere Anwendungen für den elektronischen Personalausweis. Besonders weit geht Malaysia, dessen als *Mykad* bezeichnete elektronische Ausweiskarte unter anderem als Personalausweis, Bezahlkarte, Krankenversichertenkarte, Führerschein, Bankkarte und Zutrittskontrollkarte nutzbar ist [Schm09/2]. In Hongkong können die Bürger Bibliotheken und verschiedene Freizeitangebote mit ihrem elektronischen Personalausweis nutzen [Schm09/2]. Die oben genannten europäischen Staaten begnügen sich dagegen vorläufig mit einem kleineren Anwendungsspektrum. Trotz aller Vorteile sind elektronische Personalausweise nicht unumstritten. Vor allem in Frankreich und Großbritannien gibt es beträchtlichen Widerstand gegen entsprechende Vorhaben. Die Gegner haben meist Bedenken bezüglich des Datenschutzes und kritisieren die hohen Kosten der Kartenprojekte. Trotz aller Kritik gehe ich davon aus, dass sich elektronische Personalausweise in den nächsten Jahren immer mehr durchsetzen werden.

38.6.3 Elektronische Gesundheitskarten

Bereits seit 1994 gibt es in Deutschland Ausweise für gesetzlich Krankenversicherte. Diese *Krankenversichertenkarten* (KVK) genannten Karten sind mit einem Speicherchip bestückt (es handelt sich also nicht um Smartcards), der lediglich verwaltungstechnische Daten speichert. Medizinische Informationen finden sich dagegen nicht auf dem Chip. Derzeit wird eine völlig neue und deutlich leistungsfähigere Karte eingeführt, die als **elektronische Gesundheitskarte** bezeichnet wird [Rispen]. Es handelt sich dabei um eine kontaktbehaftete Smartcard. Sie wird auch an privat Versicherte ausgegeben.

Die elektronische Gesundheitskarte ist ein Vielzweckinstrument. Sie soll dem Patienten die Übergabe seiner Personalien an den Arzt erleichtern, notfallrelevante Daten (z.B. die Blutgruppe) des Inhabers speichern, als Organspenderausweis dienen, die bisher üblichen Rezepte auf Papier durch eine digitale Alternative ersetzen, den Online-Zugang zu verschiedenen Diensten ermöglichen und einiges mehr. Dabei ist die elektronische Gesundheitskarte nur ein kleiner Bestandteil einer größeren Infrastruktur. Der derzeit laufende Aufbau des Gesamtsystems zählt zu den umfangreichsten IT-Projekten, die weltweit jemals stattgefunden haben.

Wie man sich leicht vorstellen kann, spielt die Kryptografie im Zusammenhang mit der elektronischen Gesundheitskarte eine wichtige Rolle. Auf der Karte sind mehrere private Schlüssel gespeichert, die sich in verschiedener Weise für die Authentifizierung gegenüber medizinischen Online-Angeboten nutzen lassen. Außerdem wird die elektronische Gesundheitskarte (wenn auch nicht von Anfang an) digitale Signaturen unterstützen. Als asymmetrisches Verfahren wird zunächst RSA mit einer Schlüssellänge von 2.048 Bit zum Einsatz kommen. Die Spezifikation der elektronischen Gesundheitskarte nennt zudem Verfahren auf Basis elliptischer Kurven, die in einem späteren Stadium eingeführt werden können. Darüber hinaus nutzt die Karte zur symmetrischen Verschlüsselung Triple-DES und als kryptografische Hashfunktion SHA-1.

Österreich ist derweil schon etwas weiter. Die dort verwendete Krankenversichertenkarte heißt **e-Card** und ist bereits seit einigen Jahren mit einem Smartcard-Chip ausgestattet. Die Infrastruktur und die Anwendungen sind nicht ganz so komplex wie bei der elektronischen Gesundheitskarte in Deutschland. Dafür lässt sich die e-Card bereits jetzt zum digitalen Signieren nutzen. Erwähnenswert ist, dass die e-Card Krypto-Verfahren auf Basis elliptischer Kurven verwendet, während die meisten anderen derartigen Projekte bisher RSA nutzen. Auch in der Schweiz gibt es Pläne für eine elektronische Krankenversichertenkarte, die jedoch noch nicht so weit fortgeschritten sind wie bei den deutschsprachigen Nachbarn.

38.6.4 Weitere elektronische Ausweise

Neben Reisepässen, Personalausweisen und Gesundheitskarten lassen sich auch andere Ausweise mit einem Chip bestücken. Beispielsweise gibt es in Deutschland einen elektronischen Dienstausweis, der derzeit bereits von der Bundeswehr genutzt wird. In den nächsten Jahren sollen andere Behörden diesen übernehmen. Auch der elektronische Dienstausweis ist mit mehreren privaten Schlüsseln ausgerüstet und lässt sich dadurch für digitale Signaturen und zur Authentifizierung nutzen. Darüber hinaus ist ein solcher Ausweis mit einem Mifare-, Hitag- oder Legic-Chip ausgestattet, damit er auch für den Gebäudezutritt zum Einsatz kommen kann.

Nicht zu vergessen sind zudem elektronische Mitarbeiterausweise für Unternehmen, die ebenfalls immer beliebter werden. Auch diese lassen sich für unterschiedlichste Anwendungen nutzen – beispielsweise für die Authentifizierung, das digitale Signieren, den Gebäudezutritt oder die Zeiterfassung. Es versteht sich von selbst, dass ein elektronischer Mitarbeiterausweis praktisch und kostensparend ist. In zahlreichen Großunternehmen ist diese Technik daher bereits Realität.

38.7 Digital Rights Management

Bereits in Abschnitt 24.2 bin ich darauf eingegangen, dass Alice auf ihrem eigenen PC mit ihren Dateien machen kann, was sie will. Sie kann Dateien beliebig kopieren, ausführen, löschen oder an andere verschicken – so wie es ihr gerade gefällt. Wie Sie aus demselben Abschnitt wissen, haben diese Freiheiten in manchen Fällen auch ihre Nachteile, wobei sich insbesondere das Virenproblem und das Rechteproblem stellen. An dieser Stelle soll uns das Rechteproblem interessieren. Dieses gibt es in vielerlei Spielarten:

- Wenn Bob eine vertrauliche Mail an Alice schickt, dann möchte er möglicherweise vermeiden, dass Alice diese an andere weiterleitet.
- Wenn Alice eine Musikdatei herunterlädt, dann möchte der Verkäufer in der Regel nicht, dass Alice diese kopiert und anderen zur Verfügung stellt.
- Der Verkäufer einer Software möchte möglicherweise nicht, dass Kundin Alice alle Funktionen des Programms nutzt. Stattdessen soll ihr zunächst nur ein Teil der Funktionalität zur Verfügung stehen. Wenn sie jedoch eine zusätzliche Gebühr bezahlt hat, soll sie das gesamte Programm nutzen können.

Maßnahmen, die zur Lösung des Rechteproblems dienen, fasst man unter dem Begriff **Digital Rights Management** (**DRM**) zusammen. Ein Werkzeug zur Durchführung von Digital Rights Management wird **DRM-System** genannt. Dieser Begriff ist etwa deckungsgleich mit **Conditional Access System** und **Zugangsberechtigungssystem**. Die beiden letzteren Begriffe werden vor allem im Zusammenhang mit dem Bezahlfernsehen verwendet. Wie Sie sich denken können, spie-

len in der Praxis vor allem solche DRM-Systeme eine Rolle, in denen Alice für den Zugang zu den jeweiligen Daten eine Gebühr bezahlen muss. Es geht also ums Geld, und genau das ist der Grund dafür, dass das Digital Rights Management zwischenzeitlich einen enormen Boom erlebte. Treibende Kräfte waren vor allem die Film- und die Musikindustrie, die verhindern wollten, dass ihre Erzeugnisse nach Belieben vervielfältigt werden.

Das Rechteproblem ist nicht neu. Kopierte Video- und Musikkassetten gab es schon vor Jahrzehnten. In den achtziger Jahren entwickelte sich das Kopieren von Computerprogrammen dank dem legendären Commodore 64 zum illegalen Volkssport. Wirklich bedrohlich wurde es beispielsweise für die Musikindustrie, als es dank immer billiger werdender CD-Brenner möglich wurde, Tonträger ohne Qualitätsverlust zu kopieren. So rutschten die Musikkonzerne in die tiefste Krise ihrer Geschichte und mussten oft zweistellige Umsatzverluste einstecken. Ähnliche Probleme hatte die Filmindustrie, die sich nun mit kopierten DVDs herumschlagen musste. Mit dem Aufkommen breitbandiger Internetanschlüsse wurde schließlich der Austausch der kopierten Ware deutlich einfacher.

Vor diesem Hintergrund verwundert es kaum, dass die Industrie händeringend nach wirksamen DRM-Strategien suchte. Verbraucherschützer und anarchistisch angehauchte Computernutzer kritisierten dies, da sie im DRM eine Gängelung der Anwender sahen. Diese DRM-Diskussion, die äußerst emotional geführt wurde, ist jedoch inzwischen wieder abgeebbt. Der Grund dafür ist, dass die meisten Anbieter auf allzu restriktive DRM-Maßnahmen verzichteten, um ihre Kunden nicht zu vergraulen. Angebote wie iTunes oder Musicload zeigen, dass sich auch mit ungeschützten Dateien Geld verdienen lässt. Offenbar gibt es genügend Kunden, die beispielsweise für eine Musikdatei lieber ein bis zwei Euro bezahlen, anstatt im Bekanntenkreis oder auf illegalen Plattformen umständlich danach suchen zu müssen. Dennoch ist Digital Rights Management nach wie vor ein aktuelles Thema, wie Sie im Folgenden sehen werden.

38.7.1 Containment und Marking

Digital Rights Management hat zunächst einmal nichts mit Kryptografie zu tun. Die Kryptografie ist jedoch ein wichtiges Hilfsmittel für das DRM. Für die weiteren Betrachtungen müssen wir uns vom klassischen Alice-Bob-Mallory-Modell verabschieden. Stattdessen benötigen wir ein neues Szenario, in dem ein Anbieter (nennen wir ihn Theo) Alice digitale Daten zur Verfügung stellt. Für diese Daten will Theo eine Nutzungsbeschränkung durchsetzen – er will also beispielsweise verhindern, dass Alice die Daten kopiert. Alice hat jedoch ein Interesse daran, diese Nutzungsbeschränkungen auszuhebeln. Theo muss Alice daher als seine Gegenspielerin betrachten und DRM-Maßnahmen anwenden, um seine Interessen zu wahren. Wie man sich leicht vorstellen kann, ist dies eine schwierigere Aufgabe als das herkömmliche Verschlüsseln, bei dem lediglich Abhörer Mallory

ausgeschaltet werden soll. Andererseits ist der Aufwand, den Alice in der Praxis treibt, oft nicht besonders hoch. Es lohnt sich für sie nun einmal nicht, eine starke Verschlüsselung zu knacken, nur um eine Musikdatei im Wert von 1,49 Euro vervielfältigen zu können. Theo muss jedoch auch Zeitgenossen mit krimineller Energie fürchten, die geschützte Daten als Raubkopien verbreiten.

Ein mögliches Hilfsmittel für das Digital Rights Management ist das Trusted Computing (siehe Abschnitt 24.2). Dieses ermöglicht die Bearbeitung von Daten auf Alices Computer, ohne dass Alice auf beliebige Weise in diesen Vorgang eingreifen kann. Das im besagten Abschnitt beschriebene Trusted Platform Module (TPM) ist jedoch nicht für eine DRM-Nutzung ausgelegt.

Die Kryptografie spielt im Digital Rights Management an zwei Stellen eine Rolle. Am offensichtlichsten ist folgendes Szenario: Bevor Theo Alice seine Daten liefert, dann verschlüsselt er diese. Erst wenn Alice ihre Gebühr bezahlt hat, erhält sie den Schlüssel. Dies wird auch als **Containment** bezeichnet. Durch Containment kann Theo zwar nicht das Kopieren der Daten verhindern. Er kann jedoch dafür sorgen, dass nicht jeder die Daten unbefugt nutzen kann. Containment funktioniert besonders gut im Zusammenhang mit Trusted Computing. Dann nämlich kann das verwendete Trusted-Computing-Modul die Entschlüsselung der Daten übernehmen und gleichzeitig dafür sorgen, dass Alice die erhaltenen Daten nicht kopiert.

Die zweite Stelle, an der die Kryptografie eine Rolle im DRM spielt, nutzt digitale Signaturen. Hierbei signiert Theo seine Daten inklusive einer Zusatzinformation, die angibt, wofür Alice die Daten verwenden darf. Dies wird **Marking** genannt. Marking ist auf Trusted Computing angewiesen. Die übliche Vorgehensweise sieht vor, dass Theo die signierten Daten an das Trusted-Computing-Modul von Alice schickt. Dieses wertet die Zusatzinformation aus und verifiziert die Signatur. Anschließend lässt es nur diejenigen Aktionen zu, die Theo in der Zusatzinformation angegeben hat. Durch die Signatur kann Alice die Zusatzinformation nicht zu ihren Gunsten verändern.

Containment und Marking stellen im Zusammenspiel mit Trusted Computing wirksame Werkzeuge für das Digital Rights Management dar. Eine hohe Sicherheit wird man damit alleine jedoch meist nicht erreichen. Beispielsweise kann Alice eine Musikdatei immer dadurch kopieren, dass sie ein Mikrofon vor den Lautsprecher hält. Die Entwickler von DRM-Systemen müssen also damit leben, dass im Alice-Theo-Modell eine Lösung schwieriger zu finden ist als im Alice-Bob-Mallory-Modell. Das Ziel muss es daher sein, DRM-Systeme zu entwickeln, die die Verletzung des Urheberrechts auf ein erträgliches Maß eindämmen. Das illegale Kopieren digitaler Daten wird man jedoch nie vollständig verhindern können.

38.7.2 Beispiele für DRM-Systeme

DRM-Systeme gibt es in zahlreichen Variationen. Einige Beispiele schauen wir uns im Folgenden an.

Bezahlfernsehen

Ein Bereich, in dem das Digital Rights Management seit Jahren gut funktioniert, ist das Bezahlfernsehen. Die hierzu verwendete Technik basiert auf dem Containment-Prinzip. Von Vorteil für die DRM-Entwickler ist, dass beim Bezahlfernsehen nur der direkte Datenzugang verhindert werden soll, während dem Anwender das Kopieren der Daten erlaubt wird. Von Nachteil ist dagegen, dass im Bezahlfernsehen dieselbe Nachricht (das Fernsehsignal) an Millionen von Empfängern geschickt wird, was das Schlüsselmanagement nicht gerade einfach macht.

Das Fernsehen im Allgemeinen und das Bezahlfernsehen im Besonderen machen derzeit eine Entwicklung von der analogen zur digitalen Übertragung durch. Uns soll nur die moderne, also die digitale Version interessieren. Digitales Fernsehen wird größtenteils im Rahmen des *DVB-Standards* (die Abkürzung steht für Digital Video Broadcasting) betrieben. Auch das Bezahlfernsehen bewegt sich innerhalb dieses Standards. Wenn ein Fernsehsender ein kostenpflichtiges Programm nach DVB-Standard anbietet, dann verschlüsselt er das Programmsignal mit dem Verfahren CSA (siehe Abschnitt 16.7.1). CSA ist eine speziell für den DVB-Standard entwickelte Kombination aus Block- und Stromchiffre mit einer Schlüssellänge von 64 Bit. Ein CSA-Schlüssel wird auch **Control Word** genannt. Sinnvollerweise verwendet jeder Sender für jedes Programm ein eigenes Control Word und kann dieses beliebig oft ändern. Das Ziel des digitalen Bezahlfernsehens besteht darin, dass jeder zahlende Kunde stets über das passende Control Word verfügt, während Nichtkunden nicht in dessen Besitz gelangen dürfen. Dabei ist zu beachten, dass das Control Word zur Entschlüsselung eines bestimmten Programms zu einem bestimmten Zeitpunkt überall dasselbe ist.

Die Entschlüsselung des CSA-verschlüsselten Signals übernimmt ein Hardwarebauteil, das in das Empfangsgerät (z.B. in den Satelliten-Receiver) eingebaut wird. Dieses wird **Conditional Access Module** (**CAM**) genannt und ist als recht einfache Krypto-Hardware realisiert. Diese nimmt über eine standardisierte Schnittstelle den CSA-verschlüsselten Datenstrom sowie ein Control Word entgegen und gibt den Datenstrom im Klartext aus. Diesen Klartext kann der Fernseher wie gewohnt darstellen.

Während die Ver- und Entschlüsselung des Fernsehsignals im DVB-Standard festgelegt ist und daher praktisch überall gleich abläuft, ist das Schlüsselmanagement dem Anbieter überlassen. Standardisiert ist lediglich, dass das Fernsehsignal außer dem verschlüsselten Programm Steuerinformationen enthält (sogenannte *Service Informations*), die der Sender nach seinem eigenen Bedarf nutzen kann. In aller Regel läuft das Schlüsselmanagement wie folgt ab: Will Alice Kunde eines

Bezahlsenders werden, dann kauft sie eine von diesem angebotene Smartcard. Diese Smartcard enthält einen Masterschlüssel, der für alle Karten dieses Fernsehprogramms identisch ist. Die Länge dieses Schlüssels, der unauslesbar gespeichert ist, kann 128 Bit oder mehr betragen. Wie bei einem Masterschlüssel üblich, wird er nur zur Berechnung eines Sitzungsschlüssels (in diesem Fall ist dies das Control Word) verwendet. Hat Alice eine Smartcard gekauft, dann steckt sie diese in einen Leser, der mit dem CAM in ihrem Receiver verbunden ist.

Das eigentliche Protokoll sieht meist etwa so aus:

1. Der Fernsehsender nimmt einen Zufallswert und wendet darauf sowie auf den Masterschlüssel eine kryptografische Hashfunktion an (dieselbe Hashfunktion muss auch auf der Smartcard implementiert sein). Das Ergebnis ist ein 64-Bit-Hashwert, der als Control Word dient.
2. Der Fernsehsender teilt dem Empfangsgerät über eine Steuerinformation den Zufallswert mit und verschlüsselt von da an das Fernsehsignal mit dem Control Word.
3. Das Empfangsgerät leitet die Challenge an das CAM weiter, das diese an die Smartcard übergibt. Die Smartcard berechnet daraus sowie aus dem Masterschlüssel einen 64-Bit-Hashwert, wobei sie dieselbe kryptografische Hashfunktion verwendet wie der Fernsehsender. Das Ergebnis ist das Control Word.
4. Das CAM entschlüsselt nun alle eintreffenden Signale mit dem Control Word.

Will der Fernsehsender das Control Word ändern (was er aus Sicherheitsgründen regelmäßig tun sollte), dann generiert er einen neuen Zufallswert und teilt diesen dem Empfangsgerät per Steuerinformation mit. Auf dieselbe Weise kann der Sender die Smartcard dazu veranlassen, ihren Betrieb einzustellen. Dies wird beispielsweise dann notwendig, wenn Alice ihr Abonnement kündigt. Dabei können auf einer Smartcard auch mehrere Masterschlüssel gespeichert sein, wenn ein Sender verschiedene Programme anbietet.

Windows Rights Management Services

Rights Management Services (*RMS*) ist der Name eines DRM-Systems, das von Microsoft entwickelt wurde und in die Windows-Betriebssysteme integriert ist. Bisher wird RMS nur wenig genutzt. Im Gegensatz zur Mehrzahl der DRM-Systeme ist RMS nicht auf kostenpflichtige Inhalte wie Audio- und Videodateien ausgerichtet. Es ist also keine Technologie, die der Musik- und Videobranche ihre Einnahmen sichern soll. Stattdessen zielt RMS vor allem auf eine Verwendung im Büro ab. Es soll beispielsweise sicherstellen, dass Bob eine von Alice erhaltene E-Mail nicht ausdrucken oder weiterleiten kann. Ebenso kann Alice mit RMS erreichen, dass eine von ihr erstellte und an Bob geschickte Datei nach einer bestimmten Frist automatisch gelöscht wird.

RMS sieht drei Beteiligte vor. Zum einen ist dies der Autor eines Dokuments (in unserem Fall Alice), zum anderen der Empfänger des Dokuments (Bob). Drit-

ter Beteiligter ist ein RMS-Server, der für Alice und Bob erreichbar ist. Im einfachsten Fall wird der RMS-Server vom Unternehmen betrieben, in dem Alice und Bob arbeiten. Microsoft stellt jedoch auch einen über das Internet erreichbaren RMS-Server zur Verfügung, und es könnte zukünftig weitere Anbieter geben. RMS setzt in seiner derzeitigen Version Kryptografie nur für das Containment ein. Das Markieren der Daten ist zwar ebenfalls vorgesehen, es gibt jedoch keine digitalen Signaturen und keine kryptografischen Hashwerte, die diese Markierungen schützen.

Windows Media DRM

Windows Media DRM (**WMDRM**) ist ein weiteres DRM-System, das von Microsoft entwickelt wurde [WMDRM]. Im Gegensatz zum vom selben Anbieter stammenden Windows RMS ist es nicht für Office-Dokumente und E-Mails gedacht, sondern soll den Umgang mit Musik- und Videodaten regeln. Genauer gesagt ist Windows Media DRM ein DRM-System für WMA- und WMV-Dateien, das für den Einsatz im Internet konzipiert ist. Windows Media DRM ist also die Antwort Microsofts auf das Problem des massenhaften Kopierens von Audio- und Videodaten.

Leider hat Microsoft die technischen Details des Windows Media DRM bisher nicht veröffentlicht. Klar ist aber, dass es drei Beteiligte gibt: den Eigentümer einer Datei (z. B. ein Musikportal), den Kunden (z. B. Alice) sowie einen Lizenzierungsserver. Alle drei nutzen Softwarepakete von Microsoft. Sowohl Containment als auch Marking kommen zum Einsatz.

Jede Musikdatei, die Alice vom Musikportal herunterlädt, ist verschlüsselt. Den geheimen Schlüssel zur Entschlüsselung erhält Alice vom Lizenzierungsserver, wobei sie sich zuvor gegenüber diesem authentisieren muss (der Schlüssel dazu ist in die Client-Software integriert). Mit der Musikdatei erhält Alice zudem ein sogenanntes Lizenzzertifikat, in dem in signierter Form festgehalten ist, auf welche Weise sie die Datei nutzen darf. Ihre Client-Software sorgt dafür, dass Alice diese Bestimmungen einhält.

Da Microsoft die verwendeten Krypto-Verfahren nicht veröffentlicht hat, ist man auf die Ergebnisse von Codeuntersuchungen angewiesen. Offensichtlich verwendet Windows Media DRM drei verschiedene symmetrische Verfahren: den DES, RC4 sowie ein proprietäres Verfahren. Als kryptografische Hashfunktion wird SHA-1 eingesetzt. Darüber hinaus verwendet Windows Media DRM Verfahren auf Basis elliptischer Kurven für die Signatur und die Authentifizierung.

Wie alle DRM-Lösungen ist auch Windows Media DRM nur bedingt sicher. Beispielsweise lässt sich schlecht verhindern, dass Alice ihre Client-Software so manipuliert, dass diese das Lizenzzertifikat missachtet. Die bisher einzige wirksame Gegenmaßnahme besteht darin, dass Microsoft den Programmcode geheim hält, um eine solche Manipulation zu erschweren. Dies ist jedoch nur bedingt wirksam, und in der Tat gibt es längst verschiedene Softwareprogramme zum Knacken von Windows Media DRM.

38.8 Elektronische Wahlen und Online-Wahlen

Elektronische Wahlen (**Electronic Voting**) sind Wahlen, bei denen elektronische Hilfsmittel für die Stimmabgabe und für das Auszählen der Stimmen genutzt werden. Typisch ist beispielsweise ein Wahlcomputer, der im Wahllokal steht und Wählerin Alice einen Bildschirm zum Anklicken ihrer Wahlentscheidung zur Verfügung stellt. **Online-Wahlen** gehen noch einen Schritt weiter: Hier kann Wählerin Alice ihre Stimme sogar vom heimischen PC aus über das Internet abgeben.

Aus Sicht der Kryptografie sind vor allem Online-Wahlen interessant (einen Überblick gibt es in [Starts]). Diese lassen sich nur mithilfe eines geeigneten kryptografischen Protokolls (**Wahlprotokoll**) sinnvoll durchführen. Die Anforderungen an ein solches Protokoll sind äußerst komplex. So muss ein Wahlprotokoll gewährleisten, dass das Wahlgeheimnis gewahrt bleibt, dass nur Wahlberechtigte eine Stimme abgeben können, dass kein Wähler mehrfach wählt und dass Bösewicht Mallory keine Stimme verändern kann. Auch Verkehrsflussanalysen und Denial-of-Service-Attacken können bei einer Online-Wahl durchaus zum Problem werden.

In den letzten 20 Jahren wurden unzählige Wahlprotokolle (größtenteils in Form von Konzeptprotokollen) vorgeschlagen. Viele davon bieten neben den genannten Anforderungen auch einige weitere Besonderheiten. So haben einige Protokolle die Eigenschaft, dass Alice nach dem Wählen nicht belegen kann, dass sie eine bestimmte Partei gewählt hat (es nützt daher nichts, Alice mit Gewalt dazu zu zwingen, ihre Wahlentscheidung offenzulegen). Außerdem ermöglichen es einige Protokolle, dass man das Wahlergebnis auf zuverlässige Weise prüfen kann (ein Insider kann dadurch nicht ohne Weiteres Stimmen hinzufügen oder wegfallen lassen).

Die zahlreichen Diskussionen, die es in den vergangenen Jahren zum Thema Online-Wahlen gegeben hat, drehten sich allerdings meist nicht um die kryptografische Seite des Themas. Stattdessen stand eher die Frage im Vordergrund, ob Online-Wahlen generell sinnvoll sind. Manche Experten meinen, dass man sich bei Wahlen nicht von Maschinen abhängig machen sollte, zumal selbst das beste kryptografische Protokoll nicht alle Angriffe verhindern kann (man denke nur an fehlerhafte Clients oder manipulierte Software). Andere befürchten, dass durch Online-Wahlen wohlhabende und gebildete Bürger bevorzugt werden, da diese eher einen Computer besitzen und sich im Umgang damit leichter tun. Auf der anderen Seite sind Online-Wahlen ohne Zweifel bequem für den Bürger und könnten die Wahlbeteiligung erhöhen. Erste Erfahrungen mit Online-Wahlen gibt es aus den USA, aus Estland und der Schweiz. Bisher ist noch nicht erkennbar, wohin die Reise gehen wird.

Teil 6

Mehr über Kryptografie

39 Wo Sie mehr zum Thema erfahren

Die vergangenen Kapitel haben Ihnen zwar einen guten Überblick über die Kryptografie und deren Anwendungen gegeben. Auf dem Weg zum gestandenen Kryptografie-Experten haben Sie damit jedoch erst die erste Etappe zurückgelegt. Wo Sie für Ihren weiteren Weg geeignete Informationsquellen finden, erfahren Sie in diesem Kapitel.

39.1 Buchtipps

Wer sich heute in einem gut sortierten Buchladen umsieht, kommt kaum auf den Gedanken, dass die Kryptografie einst eine Geheimwissenschaft war. Im Gegenteil: Die öffentlich verfügbare Literatur zum Thema Kryptografie ist heute so umfangreich, dass selbst Experten kaum noch den Überblick behalten können. Die folgende Liste nennt nur die Grundlagenwerke und ist noch nicht einmal vollständig:

- Bauer, Friedrich L.: *Entzifferte Geheimnisse. Methoden und Maximen der Kryptologie* (500 Seiten) [Bauer]. Dies ist ein Klassiker der deutschen Kryptografie-Literatur. Das Buch geht auch ausführlich auf historische Verfahren ein, ist aber leider nicht mehr ganz aktuell.
- Beutelspacher, Albrecht; Neumann, Heike B.; Schwarzpaul, Thomas: *Kryptografie in Theorie und Praxis* (320 Seiten) [BeNeSc]. Gut verständliches Buch über die mathematische Seite der Kryptografie. Die Autoren gehen auch auf das SSL-Protokoll und die Software PGP ein, womit sie über den mathematischen Tellerrand hinausschauen.
- Beutelspacher, Albrecht; Schwenk, Jörg; Wolfenstetter, Klaus-Dieter: *Moderne Verfahren der Kryptographie: Von RSA zu Zero-Knowledge* (164 Seiten) [BeScWo]. Kompakte Einführung ins Thema.
- Buchmann, Johannes: *Einführung in die Kryptographie* (270 Seiten) [Buch03]. Eines von vielen Büchern, das Kryptografie auf etwa 300 Seiten mathematisch erklärt, leider nicht mehr ganz aktuell.
- Delfs, Hans; Knebl, Helmut: *Introduction to Cryptography* (380 Seiten) [DelKne]. Relativ aktuelle Einführung.
- Ertel, Wolfgang: *Angewandte Kryptographie* (220 Seiten) [Ertel]. Weitere kompakte Einführung.
- Esslinger, Bernhard: *Das CrypTool-Skript: Kryptographie, Mathematik und mehr* (230 Seiten), 11. Auflage [Esslin]. Dies ist das Begleitbuch zur kostenlosen Software CrypTool (siehe Abschnitt 39.4.4). Das gesamte Buch ist auf der CrypTool-Webseite (www.cryptool.de) kostenlos erhältlich und wird zudem regelmäßig aktualisiert (die aktuelle Auflage ist die elfte). Es gibt sowohl eine deutsche als auch eine englische Version. Das über 300 Seiten umfassende Werk erklärt viele der in CrypTool verfügbaren Krypto-Verfahren mathematisch fundiert und liefert zudem die wichtigsten Grundlagen. Die inhaltliche Qualität ist mit einem kommerziellen Buch vergleichbar.
- Horster, Patrick: *Kryptologie* (332 Seiten) [Horste]. Inzwischen veraltetes deutschsprachiges Buch aus dem Jahr 1985.
- Hromkovic, Juraj; Freiermuth, Karin; Keller, Lucia; Steffen, Björn: *Einführung in die Kryptologie* (370 Seiten) [HrFrKS]: Ein weiteres mathematisches Kryptografie-Buch.
- Karpfinger, Christian; Kiechle, Hubert: *Kryptologie: Algebraische Methoden und Algorithmen* [KarKie] (270 Seiten): Noch ein mathematisches Kryptografie-Buch.
- Katz, Jonathan; Lindell, Yehuda: *Introduction to Modern Cryptography* (530 Seiten) [KatLin]. Im deutschsprachigen Raum wenig verbreitetes Kryptografie-Buch.

- Kuhn, Nico: *Das Buch der geheimen Verschlüsselungstechniken* (320 Seiten) [Kuhn]. Ein guter Tipp für alle, die einen Einstieg in die Kryptografie ohne allzu viel Mathematik suchen.
- Mollin, Richard: *An Introduction to Cryptography* (380 Seiten) [Mollin]. Dies ist ein in Europa wenig bekanntes Kryptografie-Buch mit Mathematik-Schwerpunkt.
- Menezes, Alfred; van Oorschot, Paul; Vanstone, Scott: *Handbook of Applied Cryptography* (780 Seiten) [MeOoVa]: Dieser Klassiker wird von Insidern meist »MOV« abgekürzt. Leider ist dieses Werk inzwischen veraltet.
- Nichols, Randal K.: *ICSA Guide to Cryptography* (830 Seiten) [Nichol]: Sehr umfangreiches Kryptografie-Buch mit vergleichsweise wenig Mathematik. Leider veraltet.
- Oppliger, Rolf: *Contemporary Cryptography* (500 Seiten) [Opplig]. Alternative zum Buch von Stinson.
- Paar, Christof; Pelzl, Jan: *Understanding Cryptography* (370 Seiten) [PaaPel]. Von zwei deutschen Autoren auf Englisch verfasstes Kryptografie-Buch.
- Schneier, Bruce: *Angewandte Kryptographie* (780 Seiten) [Schn96]. Dieses Kompendium galt einst als das Buch der Bücher in der Kryptografie. Eine ganze Generation von Kryptografen und Krypto-Interessierten ist damit aufgewachsen. Die Übersichtlichkeit, der lebendige Schreibstil und der Verzicht auf allzu viel Mathematik heben dieses Buch meilenweit über den Durchschnitt hinaus. Schade nur, dass Schneier seit 1996 keine Neubearbeitung mehr vorgenommen hat. Es ist jedoch kein Geheimnis, dass das Buch, das Sie gerade in den Händen halten, den Zweck hat, diese Lücke zu schließen. Was den Umfang anbelangt, hat es Schneiers *Angewandte Kryptographie* inzwischen überholt. Was die Qualität des Inhalts anbelangt, müssen Sie sich selbst ein Urteil bilden.
- Selke, Gisbert: *Kryptographie* (230 Seiten) [Selke]. Kurze Kryptographie-Einführung, leider veraltet.
- Swoboda, Joachim; Spitz, Stephan; Pramateftakis, Michael: *Kryptographie und IT-Sicherheit: Grundlagen und Anwendungen* (280 Seiten) [SwSpPr]. Deutschsprachige Kryptografie-Einführung, die auch auf Chipkarten und Trusted Computing eingeht.
- Stallings, William: *Cryptography and Network Security* (680 Seiten) [Stal06]. Dies ist bereits die vierte Ausgabe eines durchaus lesenswerten Buchs. Es behandelt neben der Kryptografie auch einige andere Themen der Netzwerksicherheit, allerdings nur sehr knapp.
- Stinson, Douglas: *Cryptography in Theory and Practice* (590 Seiten) [Stinso]. Ausführliches und mathematisch exaktes Kryptografie-Buch, das zu den Klassikern des Genres zählt.

- Tilborg, Henk van: *Fundamentals of Cryptology. A Professional Reference and Interactive Tutorial* (510 Seiten) [Tilb99]. Mathematisch fundierte Kryptografie-Einführung, leider recht teuer und inzwischen veraltet.
- Tilborg, Henk van: *Encyclopedia of Cryptography and Security* (680 Seiten) [Tilb05]. Dies ist ein Kryptografie-Lexikon, dessen Einträge von unterschiedlichen, teilweise namhaften Kryptografen verfasst worden sind.
- Trappe, Wade; Washington, Lawrence C.: *Introduction to Cryptography with Coding Theory* (600 Seiten) [TraWas]. Mathematisches Kryptografie-Buch mit einem zusätzlichen Schwerpunkt in der Kodierungstheorie.
- Wätjen, Dietmar: *Kryptographie* (310 Seiten) [Wätjen]. Deutschsprachiges Kryptografie-Buch.
- Wobst, Reinhard: *Abenteuer Kryptologie* (270 Seiten) [Wobst]. Deutschsprachiger Klassiker, leider veraltet.

Es ist nicht einfach, sich aus diesem enormen Angebot die lohnenswertesten Lektüren herauszupicken. Zweifellos ist Schneiers Klassiker *Angewandte Kryptografie* auch heute noch mit Gewinn zu lesen. Wenn Sie mehr über die Mathematik der Kryptografie erfahren wollen, dann ist das CrypTool-Skript von Bernhard Esslinger eine gute Wahl – es ist kompetent, kostenlos und wird regelmäßig aktualisiert. Falls Ihnen das CrypTool-Skript nicht genug Tiefgang bietet, sollten Sie es mit *Cryptography in Theory and Practice* von Stinson versuchen. Auch in den meisten anderen genannten Büchern werden Sie viel Mathematik finden – viele Krypto-Experten haben nun einmal einen mathematischen Hintergrund.

Neben mehreren Dutzend Grundlagenwerken gibt es längst auch zahlreiche Bücher, die nur ein Teilgebiet der Kryptografie abdecken, beispielsweise SSL [Davies], Kerberos [Pröhl], Kryptografie für Entwickler [StDeni], Virtuelle private Netze [Lipp06], Wireless-Verschlüsselung [Chandr] oder Kryptografie mit Java [Lipp00, Hook]. Im Folgenden beschreibe ich einige Bücher, die ich besonders empfehlenswert finde, und gehe am Ende zum Vergleich auf dasjenige ein, das Sie gerade in den Händen halten.

39.1.1 Anderson: Security Engineering

Bei der Lektüre des vorliegenden Buchs sollte Ihnen klar geworden sein, dass die Kryptografie für sich genommen eine recht nutzlose Angelegenheit ist. Wenn AES, RSA und all die anderen Methoden einen wirklichen Wert entwickeln sollen, dann müssen sie zunächst einmal richtig implementiert und mit anderen Sicherheitsmaßnahmen (z.B. manipulationssichere Hardware, Diebstahlsicherung, Rollenmodell, Rechtevergabe) kombiniert werden. Wer sich ernsthaft mit Kryptografie beschäftigt, sollte sich daher unbedingt auch mit der Entwicklung von Sicherheitssystemen (Security Engineering) befassen. Das beste Buch zu diesem Thema ist ohne Zweifel *Security Engineering* von Ross Anderson [Ande08].

Die aktuelle Ausgabe ist 2008 erschienen und hat über 1.000 Seiten. Es gibt keine deutsche Übersetzung.

Anderson liefert in seinem Buch einen Rundumschlag zur Entwicklung sicherer Computersysteme. So erfährt der Leser einiges zu den Werkzeugen eines Sicherheitsingenieurs (dazu gehört auch die Kryptografie) und deren Anwendung in konkreten Szenarien, die vom sicheren Betrieb eines Druckers bis zum Einsatz von Atomwaffen reichen. Spannend zu lesen ist das alles nicht zuletzt wegen der zahlreichen Beispiele aus dem wirklichen Leben, die der Autor angibt. Zu jedem Thema gibt es zudem einen Abschnitt über typische Fehler, die im jeweiligen Bereich gemacht werden. So manches Aha-Erlebnis ist dabei garantiert. Mein Tipp daher: Wenn Sie Ihren Horizont über die Kryptografie hinaus erweitern wollen, dann sollten Sie Andersons Buch lesen. Falls Sie eine deutschsprachige Alternative suchen, ist *IT-Sicherheit: Konzepte – Verfahren – Protokolle* von Claudia Eckert eine gute Wahl [Eckert].

39.1.2 Schwenk: Sicherheit und Kryptographie im Internet

Wenn Sie einige Inhalte von Teil 5 dieses Buchs (dort geht es um Kryptografie in Computernetzen) vertiefen wollen, dann ist *Sicherheit und Kryptographie im Internet* von Jörg Schwenk eine empfehlenswerte Lektüre [Schwen]. Dort erfahren Sie zahlreiche Details über kryptografische Netzwerkprotokolle wie SSL, S/MIME, OpenPGP, IPsec und DNSSEC. Das mit 270 Seiten nicht gerade umfangreiche Werk ist auch ohne größere mathematische Kenntnisse gut lesbar.

39.1.3 Schneier, Ferguson, Kohno: Cryptography Engineering

Eines vorweg: *Cryptography Engineering* [FeScKo] ist keine Neubearbeitung von Schneiers Klassiker *Angewandte Kryptographie*. Schon eher kann man sich unter diesem Werk von Bruce Schneier, Niels Ferguson und Tadayoshi Kohno ein Kryptografie-Buch für Fortgeschrittene vorstellen, die ihr Wissen mit Erfahrungen aus der Praxis vertiefen wollen. Wer *Cryptography Engineering* liest, sollte in jedem Fall mit der Materie vertraut sein und die Funktionsweise der behandelten Verfahren und Protokolle kennen. Sind diese Voraussetzungen gegeben, dann ist dieses Buch eine wahre Fundgrube, in der selbst Profis viel Neues entdecken können.

39.1.4 Karamanian, Tenneti, Dessart: PKI Uncovered

Public-Key-Infrastrukturen sind längst mehr als nur ein Teilbereich der Kryptografie. Kein Wunder also, dass es inzwischen eine ganze Reihe von Büchern gibt, die sich ausschließlich diesem Thema widmen. Mein Favorit ist *PKI Uncovered* von Karamanian, Tenneti und Dessart [KaTeDe]. Dieses Werk ist aufgrund einer ausführlichen Einführung auch für Einsteiger empfehlenswert. Fortgeschrittene

werden vor allem die Informationen über den Aufbau und den Betrieb einer PKI nützlich finden. Leider spart *PKI Uncovered* als amerikanisches Buch einige für Europa wichtige Themen aus. Dazu gehören insbesondere das deutsche Signaturgesetz und die EU-Signaturrichtlinie. Aber für solche Themen gibt es ja das Buch, das Sie gerade in den Händen halten.

39.1.5 Schmeh: Nicht zu knacken

Wenn Sie sich an Kapitel 5 dieses Buchs erinnern, dann wissen Sie, dass die Kryptografie eine bewegte Geschichte hat. Vor allem das Voynich-Manuskript, die Enigma und einige ungelöste Verschlüsselungsrätsel üben bis heute eine ungeheure Faszination aus. Da auch ich selbst die Geschichte der Kryptografie äußerst spannend finde, habe ich bereits mehrere populärwissenschaftliche Bücher darüber geschrieben. Dazu gehört *Nicht zu knacken* (2012) [Schm12/1], in dem es um die zehn wichtigsten ungelösten Rätsel der Verschlüsselungsgeschichte geht. Das Voynich-Manuskript ist in dieser Top-Ten-Liste genauso vertreten wie der Codex Rohonci, der Zodiac-Killer, der Somerton-Mann und ungelöste Enigma-Nachrichten aus dem Zweiten Weltkrieg.

Ein weiteres Buch von mir zur Kryptografie-Geschichte ist *Codeknacker gegen Codemacher* (2007) [Schm07/1]. Dieses Werk erzählt die Geschichte der Verschlüsselung von den alten Mesopotamiern bis zur Quantenkryptografie der Zukunft. 2008 erschien mit *Versteckte Botschaften* [Schm08/2] ein Buch von mir über die Geschichte der Steganografie.

39.1.6 Kahn: The Codebreakers

David Kahns *The Codebreakers* ist das Standardwerk schlechthin zur Kryptografie-Geschichte [Kahn96]. David Kahn, von Beruf Journalist und Historiker, war der Erste, der sich systematisch mit der Geschichte dieses spannenden Themas beschäftigte. Die 1967 veröffentlichte erste Ausgabe von *The Codebreakers* war die erste Arbeit, die alle wesentlichen Aspekte der Krypto-Geschichte beschrieb. Kahn begründete dadurch eine neue wissenschaftliche Disziplin, die in den letzten Jahrzehnten ein immer breiteres Publikum erreicht hat. Zahlreiche bekannte Kryptografen sind über *The Codebreakers* erstmals mit der Verschlüsselungstechnik in Berührung gekommen. Die 1996 erschienene Neubearbeitung von *The Codebreakers* hat 1.200 Seiten und ist eine nahezu unerschöpfliche Quelle für Informationen zur Kryptografie-Geschichte. Leider wird es vorläufig keine Aktualisierung geben. Dafür hat Kahn im Laufe der Jahre mehrere andere lesenswerte Bücher zur Verschlüsselungs- und Geheimdienstgeschichte veröffentlicht.

39.1.7 Schmeh: Kryptografie – Verfahren, Protokolle, Infrastrukturen

Zum Abschluss meiner Buchtipps will ich noch ein paar Worte zu dem Buch sagen, das Sie gerade in den Händen halten. Folgende Ziele habe ich mir beim Schreiben gesetzt:

- *Einsteigertauglichkeit*: Dieses Buch soll auch und gerade für Krypto-Anfänger interessant sein.
- *Verzicht auf mathematische Tiefe*: Im Gegensatz zu vielen anderen Kryptografie-Büchern ist dieses auch für Menschen ohne umfangreiche mathematische Kenntnisse geeignet.
- *Umfang*: Leider kann kein Kryptografie-Buch auch nur annähernd vollständig sein. Dieses Werk deckt jedoch ein breites Spektrum ab und ist damit meines Wissens das weltweit ausführlichste Kryptografie-Fachbuch auf dem Markt.
- *Anschaulichkeit und Lesespaß*: Wie bei allen meinen Büchern habe ich mich auch in diesem um eine gute Lesbarkeit bemüht. Dies ist keine Selbstverständlichkeit, denn viele andere Kryptografie-Bücher sind ausgesprochen trocken.

Damit dürfte klar sein, wie dieses Buch im reichhaltigen Markt für Kryptografie-Bücher positioniert ist. Die meisten anderen Werke zu diesem Thema sind deutlich weniger umfangreich, dafür aber mathematisch anspruchsvoller. Ich denke daher, dass dieses Buch eine Marktlücke schließt, zumal das populärste aller Krypto-Bücher – Bruce Schneiers *Angewandte Kryptographie* – inzwischen 17 Jahre auf dem Buckel hat und damit hoffnungslos veraltet ist.

39.2 Veranstaltungen zum Thema Kryptografie

Wer die Kryptografie nicht nur in Büchern, Zeitschriften und im Internet erleben will, der sollte sich den Besuch einer kryptografischen Fachkonferenz oder Fachmesse gönnen. Auswahl an solchen Veranstaltungen gibt es genug. Wenn Sie auf einen wissenschaftlichen Anspruch Wert legen und keine Berührungsängste mit der Mathematik haben, dann sind Sie bei den folgenden Fachkonferenzen bestens aufgehoben:

- **Crypto**: Als wichtigster Kongress für die wissenschaftliche Seite der Kryptografie gilt die jährlich im August in Santa Barbara stattfindende Crypto-Konferenz. Hier trifft sich alles, was in der Kryptografie Rang und Namen hat.
- **Eurocrypt**: Dies ist das europäische Gegenstück zur Crypto. Die Eurocrypt wird jährlich im Frühjahr an wechselnden Orten ausgetragen.
- **Asiacrypt:** Hierbei handelt es sich um das asiatische Gegenstück zur Crypto. Der Austragungsort variiert von Jahr zu Jahr.

- **Indocrypt**: Dies ist ein jährlich in Indien stattfindender Kongress, der ebenfalls der Crypto ähnelt.
- **SantaCrypt**: Findet jedes Jahr Anfang Dezember in Prag statt.
- **CHES**: Die Fachkonferenz **Cryptographic Hardware and Embedded Systems** (CHES) behandelt Themen rund um Krypto-Hardware und Kryptografie in eingebetteten Systemen. Der Austragungsort wechselt jährlich.
- **FSE**: Die **Fast Software Encryption** (FSE) ist eine weitere hoch angesehene kryptografische Fachkonferenz.
- **PKC**: Die **Public Key Cryptography** behandelt Themen rund um die asymmetrische Kryptografie.

Das Niveau aller genannten Veranstaltungen ist hoch. Wer einen Vortrag platzieren will, muss sich an einem »Call for Papers« beteiligen, wobei jeweils ein Komitee aus anerkannten Fachleuten über die Zulassung entscheidet. Wem Crypto und ähnliche Konferenzen zu abgehoben sind, sollte sich eine praxisorientiertere Veranstaltung aussuchen. Hier eine Auswahl:

- **RSA-Konferenz**: Diese jährlich im Frühjahr in San Francisco abgehaltene Kongressmesse der US-Firma RSA Security ist die wohl bedeutendste Krypto-Veranstaltung überhaupt. Sowohl das Vortragsprogramm als auch die zugehörige Messe sind einen Besuch wert. Im Vergleich zur Crypto geht es hier wesentlich weniger theoretisch zu, dafür stehen wirtschaftliche Aspekte im Vordergrund. Alle namhaften Krypto-Firmen sind präsent. Inzwischen gibt es einen europäischen Ableger der RSA-Konferenz, der jeweils an wechselnden Orten im Herbst stattfindet, sowie einen asiatischen Ableger.
- **EuroPKI**: Dies ist die wichtigste PKI-Veranstaltung auf europäischem Boden.
- **ISSE**: Die ISSE (*Information Security Solutions Europe*) ist keine reine Krypto-Veranstaltung, sondern eine Kongressmesse zum Thema IT-Sicherheit. Sie wird jedes Jahr im Herbst abgehalten, wobei der Veranstaltungsort immer in Europa liegt. Im Vergleich zur RSA-Konferenz ist die ISSE etwas wissenschaftlicher orientiert.

Wenn Sie sich für historische Verschlüsselungstechnik (z.B. die Enigma oder das Voynich-Manuskript) interessieren, dann sind folgende Veranstaltungen einen Besuch wert:

- **Cryptologic History Symposium**: Findet alle zwei Jahre in Laurel (US-Bundesstaat Maryland) statt. Die Veranstaltung wird von der NSA organisiert. Das National Cryptologic Museum befindet sich in der Nähe.
- **Charlotte International Cryptologic Symposium**: Die Premiere dieses Symposiums gab es im März 2012, und zwar in Charlotte (US-Bundesstaat North Carolina). Die Veranstaltung soll von nun an alle zwei Jahre an gleicher Stelle stattfinden.

- **Voynich 100**: Dieser Kongress zum Voynich-Manuskript fand 2012 zum ersten Mal statt. Im zweijährigen Rhythmus sollen weitere Veranstaltungen folgen.

Zusätzlich gibt es auch empfehlenswerte Veranstaltungen, die nicht nur Kryptografie, sondern IT-Sicherheit allgemein zum Inhalt haben. Hier ein paar Beispiele aus dem deutschsprachigen Raum:

- **D-A-CH Security**: Bei dieser Veranstaltung mit wissenschaftlichem Anspruch treffen sich jedes Jahr Szenegrößen aus Deutschland (D), Österreich (A) und der Schweiz (CH).
- **IT Defense**: Die IT Defense ist die exklusivste unter den deutschen IT-Sicherheitsveranstaltungen. Hier treten hochkarätige Referenten aus Europa und den USA auf. Es gibt keinen »Call for Papers«, sondern nur eingeladene Vorträge.
- **DFN Workshop Sicherheit in vernetzten Systemen**: Dieser jährlich in Hamburg stattfindende Kongress gehört seit 20 Jahren zu den wichtigsten seiner Art in Deutschland.
- **IT-SA**: Die einzige Spezialmesse zum Thema IT-Sicherheit in Deutschland findet jährlich im Herbst in Nürnberg statt.
- **Deutscher IT-Sicherheitskongress**: Diese Kongressmesse wird alle zwei Jahre vom BSI organisiert. Bei der jeweils in Bonn stattfindenden Veranstaltung sind vor allem zahlreiche IT-Sicherheitsexperten von Behörden sowie deren Partner und Lieferanten präsent.

Ein weiterer interessanter Termin für Krypto-Interessierte ist die allseits bekannte Computermesse *Cebit*. Dort gibt es einen Bereich, in dem sich die IT-Sicherheitsanbieter konzentrieren. Interessant für Kryptologen sind auch Kongresse wie *Cartes* (Paris), *Omnicard* (Berlin), *SIT-SmartCard Workshop* (Darmstadt), *Security Document World* (London) und *ID World* (verschiedene Veranstaltungsorte). Wenn Sie sich nicht nur für IT-Sicherheit, sondern auch für Tresore und gepanzerte Limousinen interessieren, dann lohnt sich ein Besuch auf der alle zwei Jahre stattfindenden Messe *Security* in Essen.

39.3 Zeitschriften zum Thema Kryptografie

Es gibt auch Zeitschriften, die sich ausschließlich mit dem Thema Kryptografie beschäftigen. Vier davon sind mir bekannt:

- **Cryptogram** (nicht zu verwechseln mit Bruce Schneiers Newsletter Crypto-Gram) ist eine US-Zeitschrift, die sich hauptsächlich mit kryptografischen Rätseln (Kryptogrammen) beschäftigt. Für die meisten Leser dieses Buchs dürfte diese Publikation eher uninteressant sein.

- **Cryptologia** ist eine Fachzeitschrift mit Peer-Review. Sie erscheint in den USA und berichtet vor allem über historische Aspekte der Kryptografie. Darüber hinaus gibt es darin allerlei Nützliches, Kurioses und Aktuelles rund um die Verschlüsselung.
- Das **Journal of Cryptology** der International Association of Cryptologic Research (IACR) ist eine weitere wissenschaftliche Fachzeitschrift zum Thema Kryptografie. Wie bei der IACR üblich, ist das mathematische Niveau sehr hoch, während viele praktische Aspekte der Kryptografie ausgeblendet werden.
- Das **Journal of Mathematical Cryptology** ist ebenfalls eine Fachzeitschrift zum Thema Kryptografie mit mathematischem Schwerpunkt.

Auf dem deutschsprachigen Markt gibt es einige Zeitschriften, die neben der Kryptografie auch andere Sicherheitsthemen behandeln. Hier eine Auswahl:

- **Datenschutz und Datensicherheit** (**DuD**): Die DuD ist zusammen mit der KES die wichtigste deutsche Zeitschrift zum Thema IT-Sicherheit. Sie erscheint monatlich und hat den Charakter einer wissenschaftlichen Fachzeitschrift.
- **KES** (**Kommunikations- und EDV-Sicherheit**): Dies ist eine weitere bedeutende IT-Sicherheits-Zeitschrift in Deutschland. Das zweimonatlich erscheinende Magazin hat den Charakter einer Computerzeitschrift.
- **IT-Security**: Die IT-Security ist eine Schweizer Zeitschrift, die viermal jährlich erscheint. Man kann sie als schweizerisches Gegenstück zur KES betrachten.
- **IT-Sicherheit**: Dieses »Fachmagazin für Informationssicherheit und Compliance« erscheint zweimonatlich.

Diese Zeitschriften sind leider nicht im Handel erhältlich und müssen daher abonniert werden. Gleiches gilt für Zeitschriften wie **Sicherheits-Berater** und **Security+Management**, die über Sicherheit allgemein (vom Brandschutz bis zur Alarmanlage) berichten und dabei auch die Kryptografie abdecken. Falls Sie Zeitschriften bevorzugen, die im Fachhandel zu erwerben sind, dann sollten Sie es mit Computermagazinen wie c't oder iX versuchen. Diese bieten in fast jeder Ausgabe einen oder zwei Artikel zum Thema.

Abb. 39–1 *Zeitschriften zum Thema IT-Sicherheit gibt es auch in deutscher Sprache.*

39.4 Weitere Informationsquellen

39.4.1 Lehrveranstaltungen

Der Aufschwung der Kryptografie in den letzten Jahrzehnten ist zu einem erheblichen Teil den Universitäten zu verdanken, die in diesem Bereich forschen und lehren. Dementsprechend gibt es inzwischen längst zahlreiche Vorlesungen, Seminare, Praktika und andere Lehrveranstaltungen zum Thema Kryptografie an deutschen Hochschulen. Hochburgen sind die Universitäten in Bochum, Darmstadt, Dresden, Essen, Hamburg, Ilmenau, Karlsruhe, Klagenfurt, München, Saarbrücken, Siegen, Weimar und Zürich. Auch an vielen Fachhochschulen werden Kryptografie-Vorlesungen angeboten.

39.4.2 Museen

Es gibt auch Museen mit Exponaten zum Thema Kryptografie. Das wichtigste davon ist das **National Cryptologic Museum** der NSA in Fort Meade, Maryland. Ein Besuch lohnt sich für alle, die an der Geschichte der Kryptografie interessiert sind. Etwas näher liegen das *Deutsche Museum* in München sowie das *Heinz Nixdorf MuseumsForum* in Paderborn, die beide interessante Krypto-Sammlungen zu bieten haben. Ein ausgesprochen empfehlenswertes Museum ist zudem in

Bletchley Park bei London untergebracht, wo im Zweiten Weltkrieg die Enigma und die Lorenz-Maschine geknackt wurden.

Abb. 39–2 *Drei Enigmas im National Cryptologic Museum*

39.4.3 Newsgruppen

Die Newsgruppen **sci.crypt** und **sci.crypt.research** (letztere ist moderiert, erstere nicht) gelten in der Kryptografie als Institution. Mit dem Aufkommen anderer Informationsquellen ist ihre Bedeutung jedoch deutlich gesunken.

39.4.4 Software

Auch in der Kryptografie funktioniert das Prinzip »Learning by Doing«. Wenn Sie ein Gefühl für die Funktionsweise von Verschlüsselungsverfahren bekommen wollen, empfehle ich Ihnen daher die Lernsoftware **CrypTool**. Die jeweils aktuelle Version können Sie im Internet kostenlos unter www.cryptool.org herunterladen. Mit CrypTool können Sie etwa 200 verschiedene Krypto-Verfahren auf anschauliche Weise anwenden. CrypTool gibt es in vier verschiedenen Versionen:

- *CrypTool 1* existiert seit 1999 und ist die weltweit verbreitetste Lernsoftware im Bereich Kryptologie.
- *CrypTool 2* ist .NET-basiert und bietet eine völlig neue Benutzeroberfläche, die unter anderem das visuelle Programmieren kryptografischer Abläufe ermöglicht.
- *JCrypTool* ist Java-basiert und damit plattformunabhängig.
- *CrypTool-Online* läuft im Browser auf dem PC und auf dem Smartphone.

Die unterstützten Algorithmen reichen von der einfachen Caesar-Chiffre über alle gängigen modernen Verschlüsselungs-, Signatur- und Hashverfahren bis zu Protokollen wie dem Diffie-Hellman-Schlüsselaustausch oder Plugins zur verteilten

Faktorisierung mittels dem Verfahren GNFS. Äußerst hilfreich sind zudem verschiedene Kryptoanalyse-Verfahren.

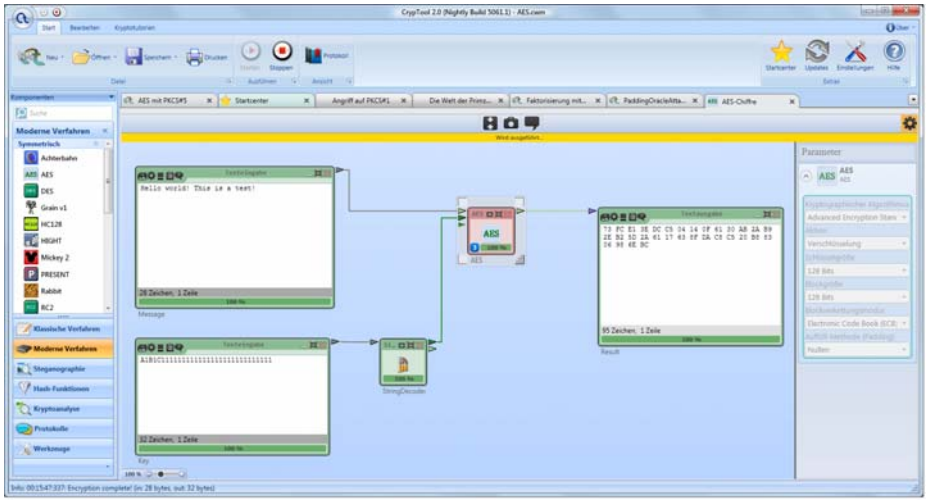


Abb. 39–3 *CrypTool ist eine kostenlose Kryptografie-Lernsoftware, die dem Nutzer zahlreiche kryptografische Verfahren zur Verfügung stellt.*

39.4.5 Webseiten

Frühere Ausgaben dieses Buch enthielten eine Liste der zehn wichtigsten Webseiten zum Thema Kryptografie. Inzwischen ist so eine Zusammenstellung überflüssig, denn für Grundlagen und als Nachschlagewerk kann ich nur noch eine Seite empfehlen: Wikipedia. Die Menge der dort verfügbaren Informationen ist beachtlich. Interessant finde ich auch die Webseite von Tobias Schrödel (www.sichere.it), wo es Fakten zu mehreren Hundert aktuellen und historischen Kryptografie-Büchern gibt. Wenn Sie aktuelle Nachrichten zur Kryptografie suchen, dann sollten Sie sich nach den Webportalen Datensicherheit.de, IT SecCity oder Heise Security umschauen. Sie alle berichten allerdings nicht nur über Kryptografie, sondern über IT-Sicherheit allgemein. Das gilt auch (trotz des Titels) für Bruce Schneiers Newsletter Crypto-Gram.

40 Kryptografisches Sammelsurium

In diesem letzten Kapitel erfahren Sie einige Dinge, die Sie in den meisten anderen Kryptografie-Büchern vergeblich suchen werden.

40.1 Die zehn wichtigsten Personen der Kryptografie

Wie jede andere Szene wird auch die Kryptografie-Szene durch die Personen geprägt, die in ihr aktiv sind. In diesem Unterkapitel will ich Ihnen die zehn wichtigsten davon etwas genauer vorstellen. Dabei sollten Sie immer im Hinterkopf behalten, dass viele geniale Kryptografen für die NSA oder andere Geheimorganisationen arbeiten, ohne dass über ihre Entwicklungen jemals etwas bekannt wird. In der hier vorgestellten Liste kommen diese zwangsläufig nicht vor.

40.1.1 Vater der Kryptografie: William Friedman (1891–1969)

Den Namen **William Friedman** werden Sie in so manchem aktuellen Kryptografie-Buch vergeblich suchen. Der Grund: Friedman war in der Vor-Computer-Ära aktiv und erlebte die in den siebziger Jahren ansetzende Blüte der akademischen Kryptografie nicht mehr mit. Dennoch wäre eine Liste der zehn bedeutendsten Kryptografen ohne William Friedman unvollständig. Der 1969 verstorbene US-Amerikaner gilt nämlich als erfolgreichster Kryptoanalytiker aller Zeiten und als Vater der wissenschaftlichen Kryptografie.

Abb. 40–1 *William Friedman gilt als bedeutendster Kryptograf der Vor-Computer-Ära.*

Geboren wurde Friedman in Moldawien, von wo er bereits als Kleinkind mit seinen Eltern in die USA auswanderte. Dort absolvierte er ein Genetikstudium und ließ sich 1915 von der privaten Forschungseinrichtung Riverbanks Laboratories verpflichten. Eigentlich sollte sich Friedman dort um genetische Forschungsarbeiten kümmern, doch schon bald gelangte er mit einem spannenderen Riverbanks-Projekt in Kontakt. In diesem wurde die damals recht populäre Vermutung überprüft, dass die William Shakespeare zugeschriebenen Werke in Wirklichkeit von dessen Zeitgenossen Francis Bacon verfasst wurden. Bacon, so die Theorie, habe seinen Namen an verschiedenen Stellen in die vermeintlichen Shakespeare-Verse einkodiert. Derartigen Codes glaubten die Riverbanks-Forscher damals auf der Spur zu sein.

William Friedman stellte zwar schnell fest, dass die Shakespeare-Code-These nicht zu halten war, doch immerhin wurde dadurch sein Interesse an Verschlüsselungsfragen geweckt. Dieses fiel auf fruchtbaren Boden, denn das US-Militär hatte damals nach dem Eintritt in den Ersten Weltkrieg Bedarf an fähigen Verschlüsselungsexperten. Als die Riverbanks Laboratories ihr diesbezügliches Know-how der US-Regierung zur Verfügung stellten, wurde Friedman unversehens zum Militär-Kryptografen. Friedman zeigte ein außerordentliches Geschick im Knacken fremder Verschlüsselungen und entwickelte sich so schnell zum

wichtigsten Krypto-Spezialisten der USA. In den zwanziger Jahren gab es kaum ein Verschlüsselungsverfahren auf der Welt, das Friedman nicht hätte lösen können. Sein erstes Meisterstück lieferte er mit dem Dechiffrieren der von Edward Hebern konstruierten Rotormaschine ab, deren Schwächen er schonungslos aufdeckte. Friedmans Verbesserungsvorschläge machten die damalige Rotorchiffren-Technik deutlich sicherer.

Geradezu legendär ist Friedmans Kryptoanalyse der damals populären Kryha-Verschlüsselungsmaschine (siehe Abschnitt 5.2.1). In nur 2 Stunden und 41 Minuten knackte er einen Kryha-verschlüsselten Text [Schm10]. Weniger bedeutende Verschlüsselungsverfahren löste Friedman nahezu im Akkord. Angeblich soll es keine einzige Chiffre gegeben haben, vor der er kapitulieren musste. Als Friedmans größter Erfolg gilt das Brechen der japanischen Verschlüsselungsmaschine Purple im Zweiten Weltkrieg. Friedman und sein Team hatten nie eine Purple gesehen und wussten nichts über deren Konstruktionsweise. Zur Verfügung standen ihnen nur abgefangene Geheimtexte, einige Klartexte sowie Informationen über einige andere japanische Verschlüsselungsmaschinen. Trotz dieser scheinbar aussichtslosen Ausgangsposition schafften es Friedman und seine Mitarbeiter innerhalb von zwei Jahren, die Funktionsweise der Purple zu durchschauen und eine Methode zur Ermittlung des Schlüssels zu entwickeln. Ab 1942 konnten die Amerikaner Purple-Nachrichten routinemäßig entschlüsseln. Viele Experten betrachten das Knacken der Purple heute als größten Kryptoanalyse-Erfolg der Geschichte.

Nach dem Zweiten Weltkrieg wurde Friedman Chef-Kryptograf der Armed Forces Security Agency (AFSA) und deren Nachfolgeorganisation NSA. Neben seinen unerreichten Fähigkeiten als Codeknacker war Friedman auch an der Entwicklung vieler Verschlüsselungsverfahren beteiligt, die in den USA zum Einsatz kamen. Außerdem betätigte er sich als Krypto-Ausbilder. Die Lehrbücher, die Friedman verfasste, waren die mit Abstand besten ihren Zeit und gelten bis heute als didaktisch vorbildlich. Friedman war damit der Erste, den man als wissenschaftlich arbeitenden Kryptografen bezeichnen kann. Als Friedman 1969 starb, hatte die Kryptografie ihren bedeutendsten Vertreter der Vor-Computer-Ära verloren.

40.1.2 Begründer der Krypto-Geschichte: David Kahn (*1930)

Der US-amerikanische Historiker und Journalist David Kahn erkannte die Bedeutung der Kryptografie bereits, als außerhalb von Militär und Geheimdiensten kaum jemand davon Notiz nahm. 1967 erschien sein Buch *The Codebreakers*, in dem er auf über 1.000 Seiten die Geschichte der Verschlüsselungstechnik erzählte – es wurde zum Klassiker. Die NSA zeigte sich allerdings weniger begeistert von Kahns Buch, da sie kryptografisches Know-how von der Öffentlichkeit fernhalten wollte. *The Codebreakers* durfte daher erst nach einigen inhaltlichen Änderungen erscheinen. Dabei geht es darin fast nur um die Geschichte der Kryptografie, während Kahn über die damals aktuellen Verfahren aufgrund der allgemein üblichen Geheimhaltung kaum etwas wissen konnte.

Obwohl *The Codebreakers* eher ein Geschichtsbuch als ein Fachbuch ist, hat es die Entwicklung der Kryptografie maßgeblich beeinflusst. Es machte die Verschlüsselungstechnik einer breiten Öffentlichkeit bekannt, und es ist sicherlich kein Zufall, dass um 1970 – also etwa drei Jahre nach Erscheinen des Buchs – die Kryptografie erstmals öffentlich als Wissenschaft betrieben wurde. Für die Kryptografen der ersten Generation war *The Codebreakers* eine Pflichtlektüre, zumal es damals kaum andere Literatur gab. Whitfield Diffie, Martin Hellman und viele weitere Koryphäen der ersten Stunde wären ohne dieses Buch vermutlich nie mit der Kryptografie in Berührung gekommen.

Ein weiteres Verdienst David Kahns ist es, die Geschichte der Kryptografie als eigenständige Disziplin etabliert zu haben. Bis heute gilt Kahn als der bedeutendste Experte auf diesem Gebiet, das nicht erst seit Hollywood-Filmen wie *Enigma* oder *Windtalkers* einen Boom erlebt. Nach *The Codebreakers* schrieb Kahn mehrere weitere bedeutende Bücher zur Kryptografie-Geschichte, darunter *Seizing The Enigma* [Kahn12] und *The Reader of Gentlemen's Mail* [Kahn04]. Hinzu kommen unzählige Zeitschriftenartikel und Fachaufsätze. Auch mit über 80 Jahren ist Kahn noch sehr aktiv und als Redner äußerst begehrt. 2010 ehrte ihn die Universität Luxemburg mit einem zweitägigen Symposium anlässlich seines 80. Geburtstags. Unter den zahlreichen teilnehmenden Kryptografen war ein Satz immer wieder zu hören: »Ohne David Kahns Buch wäre ich nie zur Kryptografie gekommen.«

40.1.3 Guru und Rebell: Whitfield Diffie (*1944)

Whitfield Diffie gilt zusammen mit Martin Hellman und Ralph Merkle als Erfinder der asymmetrischen Kryptografie. Der 1944 geborene Diffie studierte am Massachusetts Institute of Technology (MIT) und beschäftigte sich bereits Anfang der 70er Jahre mit Computersicherheit und Kryptografie. Dabei wurde ihm klar, dass die Lösung des Schlüsselaustausch-Problems die Kryptografie einen großen Schritt nach vorne bringen würde. 1974 erfuhr er, dass sich ein gewisser Martin Hellman ebenfalls mit diesem Thema beschäftigte. Diffie nahm Kontakt mit Hellman auf, der Rest ist Geschichte. Nach zweijähriger gemeinsamer Forschung entstand der Diffie-Hellman-Schlüsselaustausch, und die asymmetrische Kryptografie war geboren. Die von Diffie und Hellman 1976 veröffentlichte Forschungsarbeit »New Directions in Cryptography« gilt bis heute als wichtigster Aufsatz in der Geschichte der Kryptografie [DifHel].

Durch seine Pioniertat wurde Diffie in der Krypto-Szene zu einer lebenden Legende. Daran änderte sich auch während seiner Karriere in der Industrie nichts, die er in den achtziger Jahren startete. Zunächst war er als Sicherheitsfachmann für eine Telefongesellschaft aktiv, bevor er 1991 bei Sun Microsystems anheuerte. Diffie betätigt sich heute als Lobbyist gegen Krypto-Regulierungen und tritt häufig als Redner auf. Sein Buch »Privacy on the Line«, in dem es um

Kryptografie und Politik geht, erreichte auch außerhalb der Krypto-Szene gute Verkaufszahlen [DifLan]. Im kryptografischen Tagesgeschäft spielt Diffie zwar keine Rolle mehr, doch seinen Platz unter den zehn bedeutendsten Kryptografen hat er sicher.

40.1.4 Der Pionier: Martin Hellman (*1946)

Martin Hellman ist ein weiterer Pionier der asymmetrischen Kryptografie und damit eine Ikone der Verschlüsselungstechnik. Er wurde 1946 im New Yorker Stadtteil Bronx geboren. Bereits Anfang der 70er Jahre forschte er als Professor an der Universität Stanford in Kalifornien im Bereich Kryptografie, wobei ihn vor allem das damals noch ungelöste Schlüsselaustausch-Problem faszinierte. Zunächst arbeitete Hellman als Einzelkämpfer, doch 1974 erhielt er einen Anruf von einem Kollegen namens Whitfield Diffie. Dieser beschäftigte sich ebenfalls mit dem Schlüsselaustausch-Problem und suchte in Hellman einen Verbündeten. Die beiden trafen sich und beschlossen, fortan zusammenzuarbeiten.

Nach zwei Jahren gemeinsamer Forschung gelang schließlich Martin Hellman der Durchbruch: Er entdeckte das Verfahren, das wir noch heute als Diffie-Hellman-Schlüsselaustausch kennen. Diffie und Hellman veröffentlichten ihr Verfahren in einer amerikanischen Fachzeitschrift unter dem Titel »New Directions in Cryptography« [DifHel]. Dieser Zeitschriftenaufsatz gilt bis heute als die bekannteste wissenschaftliche Arbeit in der Geschichte der Kryptografie.

Seine 1971 angetretene Professur an der Universität Stanford behielt Hellman bis 1996. Er veröffentlichte in dieser Zeit zahlreiche weitere kryptografische Fachaufsätze, die allerdings nicht mehr die Bedeutung seiner Pionierarbeit erreichten. Dafür engagierte sich Hellman für junge Unternehmen sowie für das Thema Ethik in der technischen Entwicklung. Wie sein Weggefährte Whitfield Diffie gehört er heute nicht mehr zur Elite der Kryptografie, doch seine Verdienste sichern ihm einen Ehrenplatz in deren Geschichte.

40.1.5 Der bedeutendste Kryptograf der Gegenwart: Ron Rivest (*1947)

Ronald Rivest ist meiner Meinung nach der größte lebende Kryptograf. Der Professor an der Elite-Universität MIT (Massachusetts Institute of Technology) schloss 1969 sein Studium der Mathematik ab und erhielt 1974 sein Ph.D. (vergleichbar mit dem deutschen Doktortitel) in Informatik. Seine größte Erfindung gelang Rivest 1978, als er zusammen mit Adi Shamir und Leonard Adleman das RSA-Verfahren erfand. Die Kryptografie war damals noch eine recht junge akademische Disziplin, und auch Ron Rivest stand noch am Anfang seiner Karriere. Mit dem RSA-Verfahren hatte er eines der ersten asymmetrischen Krypto-Verfahren überhaupt erfunden, und wie wir inzwischen wissen, ist es auch eines der praktikabelsten. In den Folgejahren hat sich RSA zum wichtigsten Krypto-Ver-

fahren überhaupt entwickelt. Von den zahlreichen Produkten, die für asymmetrische Verschlüsselung oder digitale Signaturen verwendet werden können, gibt es kaum eines, das nicht das RSA-Verfahren unterstützt.

Nach seiner wichtigsten Erfindung blieb Rivest der Kryptografie treu und ließ immer wieder mit interessanten Arbeiten aufhorchen. So entwickelte er die symmetrischen Verschlüsselungsverfahren RC2, RC4, RC5 und RC6 sowie die kryptografischen Hashfunktionen MD2, MD4, MD5 und MD6. Außer MD6 wurden oder werden alle genannten Verfahren in der Praxis eingesetzt. Dass eine Person so viele erfolgreiche Krypto-Verfahren entwickelt hat, ist einzigartig. Es würde den Rahmen dieses Kapitels sprengen, wollte man alle weiteren Arbeiten Rivests aufzählen. Unter anderem gibt es von ihm Veröffentlichungen zum Thema Online-Bezahlsysteme, Information Hiding, Online-Wahlen und PKI. Zudem gründete Rivest zusammen mit Shamir und Adleman die Firma RSA Data Security, die 1996 von Security Dynamics aufgekauft wurde (Security Dynamics benannte sich später in RSA Security um). Auch über 30 Jahre nach Erfindung des RSA-Verfahrens ist Ron Rivest immer noch aktiv. Wir dürfen also gespannt sein, was wir von ihm in den nächsten Jahren noch hören werden.

40.1.6 Deutschlands bester Codeknacker: Hans Dobbertin (1952–2006)

Hans Dobbertin, Deutschlands wohl bedeutendster Kryptograf, zeigte sein Talent bereits in jungen Jahren. 1971 wurde er Bundessieger im Wettbewerb »Jugend forscht« in der Sparte Mathematik. Danach dauerte es jedoch noch 20 Jahre, bevor sich Dobbertin nach einem Mathematikstudium, Promotion und Habilitation 1991 beim Bundesamt für Sicherheit in der Informationstechnik (BSI) verdingte und sich dort zu einem hervorragenden Kryptografen entwickelte. Zu seinem Schwerpunkt wurden kryptografische Hashfunktionen (für eine ausführlichere Betrachtung von Dobbertins Leistungen siehe [Schm07/1]).

1994 erhielt das BSI vom Zentralen Kreditausschuss (ZKA), der Interessenvertretung der deutschen Banken, den Auftrag, die Qualität der kryptografischen Hashfunktion RIPEMD zu prüfen. Der ZKA wollte dieses Verfahren einsetzen und hatte bereits ein positives Gutachten vorliegen. Hans Dobbertin gab sich damit jedoch nicht zufrieden und machte sich an die Arbeit. RIPEMD zählt zu den zahlreichen MD4-Weiterentwicklungen und arbeitet wie das Vorbild in drei Runden. Im Gegensatz zu MD4 sieht RIPEMD jedoch zwei parallele Ablaufstränge vor, die für eine doppelte Sicherheit sorgen sollen. Dobbertin schaffte es dennoch, eine Zwei-Runden-Version von RIPEMD zu knacken (also Kollisionen zu finden). Dies war zwar nur eine theoretische Schwachstelle, doch das genügte, um erhebliche Zweifel an der Sicherheit des Verfahrens aufkommen zu lassen. Der ZKA musste sich also nach einer anderen kryptografischen Hashfunktion umsehen.

Nebenbei hatte Dobbertin mit seiner Arbeit auch MD4 geknackt, das einen ähnlichen Aufbau wie RIPEMD hat, mangels einer Doppelung des Ablaufstrangs jedoch weniger sicher ist. Zum ersten Mal in der modernen Geschichte der Kryptografie hatte damit ein Deutscher ein Verfahren gebrochen, das in der Praxis eingesetzt wurde. Als Nächstes nahm sich Dobbertin MD5 vor. Auch hier entdeckte er Schwachstellen und konnte am Ende sogar Kollisionen erzeugen, sofern das Verfahren mit geänderten Initialisierungsvektoren arbeitete. Dies war zwar erneut nur ein theoretischer Angriff, doch der in der Kryptografie übliche Sicherheitspuffer war damit aufgebraucht. MD5 wurde nach Dobbertins Angriff in Neuimplementierungen nicht mehr eingesetzt und ist heute weitgehend aus der Krypto-Welt verschwunden.

Nachdem Dobbertin ausreichend Erfahrung mit dem Knacken kryptografischer Hashfunktionen gesammelt hatte, entwickelte er selbst eine solche. 1996 veröffentlichte er zusammen mit den belgischen Kryptografen Antoon Bosselaers und Bart Preneel das Verfahren RIPEMD-160, das eine Weiterentwicklung von RIPEMD darstellt. RIPEMD-160 ist eines der wenigen kryptografischen Verfahren deutscher Herkunft, das es zu einer nennenswerten Verbreitung in der Praxis gebracht hat. An den neueren Entwicklungen im Bereich der kryptografischen Hashfunktionen, die die Qualität seines Verfahrens RIPEMD-160 bestätigten, konnte Dobbertin durch seine zwischenzeitlich diagnostizierte Krebserkrankung nur noch eingeschränkt teilhaben. Am 2. Februar 2006 starb Hans Dobbertin in Bochum.

40.1.7 Das »S« in RSA: Adi Shamir (*1952)

Adi Shamir ist Professor am israelischen Weizman-Institut für Wissenschaften und außerdem das »S« in RSA. Die Erfindung des RSA-Verfahrens glückte dem Israeli zusammen mit Rivest und dem Mathematiker Leonard Adleman während seines mehrjährigen Wirkens in den USA. 1980 kehrte er in seine Heimat zurück und hatte einen wesentlichen Anteil daran, dass sich Israel zu einer Hochburg der Kryptografie entwickelte. Weitere bekannte Kryptografen aus diesem Land sind Amos Fiat und Eli Biham.

Auch ohne seine Beteiligung an der Erfindung des RSA-Verfahrens müsste man Adi Shamir zu den bedeutendsten Kryptografen der Welt zählen. Niemand sonst in der Krypto-Szene kann auf so viele Kryptoanalyse-Erfolge zurückblicken wie er. Unter anderem ist Shamir Miterfinder der differenziellen Kryptoanalyse, die als bedeutendste kryptografische Analysemethode der Computer-Ära gilt. Auch die (theoretischen) RSA-Knack-Maschinen TWINKLE und TWIRL (siehe Abschnitt 11.3.3) sind sein Werk. Shamir gelang zudem ein erfolgreicher Angriff auf das Protokoll WEP, das RC4 als Verschlüsselungsverfahren nutzt. Darüber hinaus entwickelte Shamir einige Verfahren und Erkenntnisse, die in diesem Buch nicht betrachtet werden. Dazu gehört insbesondere das nach ihm benannte Fiat-

Shamir-Protokoll. Shamir ist außerdem Mitgründer der Firma RSA Data Security, die inzwischen in RSA Security (ehemals Security Dynamics) aufgegangen ist.

40.1.8 Der Volksheld: Phil Zimmermann (*1954)

Phil Zimmermann hat kein nennenswertes Krypto-Verfahren erfunden, keines geknackt und hat auch kein außergewöhnliches Buch geschrieben. Dennoch ist Zimmermann seit über einem Jahrzehnt die bekannteste Persönlichkeit der Kryptografie-Szene überhaupt. Seine Popularität ist untrennbar mit der von ihm entwickelten Software PGP verbunden, die man als sein Lebenswerk bezeichnen kann.

Zimmermann studierte in Florida Physik und Informatik. In den 80er Jahren engagierte er sich in der Friedensbewegung. Er demonstrierte gegen Aufrüstung und Atomtests, wobei auch eine Verhaftung bei einer Kundgebung sein Engagement nicht bremsen konnte. Als sich die Gefahr eines Atomkriegs Ende der 80er Jahre zunehmend verringerte, verlagerte Zimmermann sein Engagement in einen anderen Bereich. Er beschäftigte sich mit Kryptografie und sah darin eine Möglichkeit, mit der sich der Normalbürger gegen den Staat, dem Zimmermann immer noch misstraute, schützen konnte. Er beschloss, eine Software zu entwickeln, die moderne Kryptografie für PC-Anwender nutzbar machte. Pretty Good Privacy (PGP) war geboren.

Abb. 40–2 *Phil Zimmermann ist der Erfinder von PGP.*

Zimmermann ging mit PGP keine Kompromisse ein. Er verwendete – ohne die Patentlage zu beachten – RSA für den Schlüsselaustausch und für digitale Signaturen. Der DES war ihm nicht sicher genug, also setzte er auf IDEA. Eine geschwächte Version für den Export kam ihm dagegen nicht in den Sinn, obwohl die diesbezüglichen US-Gesetze seinerzeit recht streng waren. Das hierarchische Vertrauensmodell des X.509-Standards vermied er ebenfalls und setzte stattdessen auf ein Web of Trust. 1991 stellte Zimmermann PGP zum kostenlosen Download ins Netz.

PGP verbreitete sich schnell, und genauso schnell hatte Zimmermann Ärger am Hals. Die Firma RSA Data Security warf Zimmermann die Verletzung des RSA-Patents vor und bezeichnete PGP bei dieser Gelegenheit als »Pretty Good Piracy«. Außerdem wurde Zimmermann wegen Verstoßes gegen die US-Exportbestimmungen angeklagt. Diese Querelen sorgten für eine ausgeprägte Berichterstattung in der Presse und machten Zimmermann zum Volkshelden. Die Popularität von PGP stieg dadurch immer weiter an. 1996 wurde das Verfahren gegen Zimmermann eingestellt, auch der Patentstreit wurde beigelegt. Zu diesem Zeitpunkt war PGP längst die populärste Kryptografie-Software überhaupt.

In den folgenden Jahren schaffte es Zimmermann, seine Entwicklung in bare Münze umzuwandeln. Er hielt gut bezahlte Vorträge und gründete eine Firma (PGP Inc.), die 1997 von Network Associates (NAI) übernommen wurde. NAI baute PGP zu einer benutzerfreundlichen PKI-Lösung aus, die mit dem primitiven Krypto-Tool aus den Anfangstagen nicht mehr viel zu tun hatte. Einige Hardcore-Fans fühlten sich durch diese Entwicklung abgeschreckt und boykottierten die neuen PGP-Versionen.

Während nun andere Unternehmen PGP-kompatible Produkte auf den Markt brachten, stellte Network Associates im Jahr 2002 die Weiterentwicklung von PGP aufgrund mangelnder Umsätze ein. Kurz darauf sorgten jedoch einige Investoren unter Beteiligung von Phil Zimmermann für eine Wiederbelebung. Nicht zuletzt dank des Standards OpenPGP und der Open-Source-Implementierung GnuPG ist PGP immer noch aktuell. Zimmermann selbst muss sich ohnehin keine Sorgen machen, denn als hervorragender Redner kann er nach wie vor gut bezahlte Vorträge halten.

40.1.9 Der Krypto-Papst: Bruce Schneier (*1963)

Bruce Schneier ist Gründer eines Unternehmens (Counterpane Systems) sowie Erfinder einiger Krypto-Verfahren (u. a. Blowfish und Twofish). Bekannt geworden ist er jedoch in erster Linie durch sein Buch *Angewandte Kryptographie*, das Mitte der neunziger Jahre erschienen ist [Schn96]. Dieses Werk – insbesondere die zweite Ausgabe von 1996 – galt lange Zeit als die Bibel unter den Kryptografie-Büchern. Schneier gehört als Forscher sicherlich nicht zu den ganz Großen der Zunft. Durch sein äußerst erfolgreiches Buch wurde er jedoch zu einem der größten Stars der Krypto-Szene. Oft wird er sogar als »Krypto-Papst« bezeichnet, was nicht zuletzt durch seine umfangreiche Präsenz in den Medien begründet ist. Neben Phil Zimmermann hat er wohl den größten Bekanntheitsgrad unter allen Krypto-Prominenten. Dies zeigt sich nicht zuletzt darin, dass Schneier und Zimmermann in dem Bestseller *Sakrileg* von Dan Brown als Beispiele für bedeutende Krypto-Experten genannt werden.

Leider hat Schneier seit 1996 keine Neubearbeitung seines Erfolgsbuchs in Angriff genommen. Dafür hat er zwischenzeitlich einige andere Bücher veröffent-

licht (z. B. *Cryptography Engineering* [FeScKo]) und einen monatlichen Newsletter (Crypto-Gram) etabliert. Spätestens seitdem das von ihm mitentwickelte Verschlüsselungsverfahren Twofish beim AES-Wettbewerb unter die besten drei kam, wird Schneier auch als Forscher ernst genommen. Seine Hauptkompetenz liegt jedoch nach wie vor in der Beratung und Berichterstattung zur Kryptografie.

40.1.10 Zweifacher Wettbewerbssieger: Joan Daemen (*1965)

Der belgische Krypto-Experte Joan Daemen hat geschafft, was außer ihm niemandem gelungen ist: Er hat die beiden wichtigsten Standardisierungswettbewerbe der Kryptografie-Geschichte gewonnen. Sowohl beim AES-Wettbewerb als auch beim SHA-3-Wettbewerb erwies sich das jeweils von ihm mitentwickelte Verfahren in den Augen der Jury als das beste. Durch diese Erfolge hat sich inzwischen herumgesprochen, dass »Joan« im niederländischen Sprachraum ein Männername ist. Im Englischen verhält es sich bekanntlich anders, und das hat durchaus schon für Verwirrung gesorgt.

Joan Daemen studierte in den achtziger Jahren an der Universität Löwen (Belgien). Diese gilt als weltweite Kryptografie-Hochburg und hat schon zahlreiche bedeutende Kryptografen hervorgebracht (beispielsweise die in diesem Buch erwähnten Antoon Bosselaers, Bart Preneel und Vincent Rijmen). In den neunziger Jahren beschäftigte sich Daemen mit symmetrischen Verschlüsselungsverfahren und arbeitete dabei teilweise mit seinem Landsmann Vincent Rijmen zusammen. Dabei entstand unter anderem ein Verschlüsselungsverfahren namens Square. Dieses enthielt zwar einige interessante Ansätze, fiel jedoch in der Vielzahl der damals veröffentlichten Verfahren kaum auf.

Für den 1997 startenden AES-Wettbewerb überarbeiteten Daemen und Rijmen das Square-Verfahren und passten es an die AES-Kriterien an. Den neuen Algorithmus nannten sie Rijndael. Rijndael galt zunächst als Außenseiter im AES-Wettbewerb, erwies sich jedoch als erstklassiges Verfahren. Am Ende ging es als Sieger hervor und wurde zum AES ernannt (siehe Abschnitt 18.5.2). Der AES katapultierte die bis dahin nur zur zweiten Garde der Kryptografie zählenden Daemen und Rijmen an die Spitze.

Nach dem AES kam es zu keiner größeren Zusammenarbeit der beiden Belgier mehr. Daemen entwickelte dafür mit drei Kollegen die kryptografische Hashfunktion RadioGatún und anschließend deren Nachfolger Keccak. Das Vierer-Team reichte Keccak beim SHA-3-Wettbewerb ein – es setzte sich gegen 63 Konkurrenten durch (siehe Abschnitt 18.5.3). Spätestens seit diesem Erfolg hat Joan Daemen seinen Platz unter den zehn wichtigsten Personen der Kryptografie sicher.

40.2 Die wichtigsten Unternehmen

Die Kryptografie ist ein Geschäft, in dem Milliarden bewegt werden. Das größte Stück vom Kuchen nehmen nach wie vor die Vereinigten Staaten für sich in Anspruch, wo sowohl viele Abnehmer als auch die meisten Krypto-Anbieter ihren Sitz haben. Doch auch deutsche Firmen sind auf dem Krypto-Weltmarkt gut vertreten. Viele davon konnten sich in der zweiten Hälfte der neunziger Jahre etablieren, als das Interesse an Kryptografie deutlich anstieg, US-Hersteller aufgrund der dortigen Exportbestimmungen jedoch nur schwache kryptografische Lösungen ausführen durften. Heute sind diese Beschränkungen zwar längst abgeschafft, doch die deutsche Krypto-Wirtschaft kann sich nach wie vor gut gegen die US-Konkurrenz behaupten.

Im Folgenden schauen wir uns die aus deutscher Sicht wichtigsten Firmen an, die sich auf Kryptografie spezialisiert haben. Dies sollte jedoch nicht darüber hinwegtäuschen, dass ein großer Teil des Bedarfs heute von IT-Riesen wie Microsoft oder IBM sowie von einigen Open-Source-Lösungen abgedeckt wird. Produkte wie Windows, Outlook, Mozilla Firefox, Internet Explorer und Notes haben längst kryptografische Funktionen eingebaut, die den Grundbedarf decken. Für Heimanwender und mittelständische Unternehmen reicht es daher oftmals völlig aus, sich aus diesem kostenlosen Angebot zu bedienen. Die in den folgenden Unterkapiteln aufgeführten Krypto-Spezialfirmen rekrutieren ihre Kundschaft vor allem aus Großunternehmen und Großbehörden mit hohem Sicherheitsbedarf. Ihre Produkte sind oft besonders funktionsreiche oder besonders sichere Lösungen, deren Leistungsfähigkeit über die Standardfunktionen von Microsoft und Co. hinausgehen. Gerade deutsche Firmen profitieren dabei häufig davon, dass Kunden mit hohem Sicherheitsbedarf US-Hersteller generell meiden, weil sie einen Einfluss der NSA oder anderer Organisationen fürchten.

Obwohl die Nachfrage nach Kryptografie aus allen Teilen der Welt kommt, konzentrieren sich die kommerziellen Anbieter auf wenige Staaten. Die meisten haben ihren Sitz in den USA. Deutschland würde ich bereits auf Platz zwei einordnen. Daneben sind vor allem Israel, Finnland, Großbritannien und die Schweiz auf dem Krypto-Markt aktiv. In der folgenden Zusammenstellung sind Unternehmen aus dem deutschsprachigen Raum überrepräsentiert, da dies ein deutschsprachiges Buch ist.

40.2.1 Applied Security

Applied Security (apsec) ist ein auf Kryptografie spezialisiertes Unternehmen aus Stockstadt am Main. Es ist seit 1998 am Markt, wurde also in einer Zeit gegründet, als deutsche Krypto-Anbieter davon profitierten, dass ihre Konkurrenten aus den USA per Gesetz keine wirksame Verschlüsselungstechnik exportieren durften. Mit speziellen Krypto-Produkten, Auftragsarbeiten und Beratungsleistungen

konnte sich Applied Security bis heute behaupten. Kunden des Unternehmens sind vor allem Banken, Behörden und das Gesundheitswesen.

40.2.2 Crypto AG

Die **Crypto AG** mit Sitz in Zug (Schweiz) gilt als stiller Star der Krypto-Branche. Von den hier betrachteten Anbietern hat dieses Unternehmen die mit Abstand längste und interessanteste Geschichte [Schm07/1]. Gegründet wurde die Firma 1932 unter dem Namen *A. B. Kryptoteknik* von dem Schweden Boris Hagelin. Firmensitz war Schweden. Der gewiefte Geschäftsmann Hagelin vermarktete zunächst mechanische Verschlüsselungsmaschinen, die damals mit der Enigma konkurrierten. Der Zweite Weltkrieg bescherte dem Unternehmen enorme Umsätze und etablierte es an der Spitze der Krypto-Branche. 1952 verlegte Hagelin seine Firma in die Schweiz und benannte sie in Crypto AG um. Bis heute beliefert das Unternehmen Kunden in aller Welt mit Krypto-Technik. Die Kunden der Crypto AG sind vor allem staatliche Organisationen, also Regierungen, Ministerien, das Militär und Geheimdienste. Im Gegensatz zu den meisten anderen Anbietern arbeitet die Crypto AG mit proprietären und geheimen kryptografischen Verfahren. Das Unternehmensgebäude in Zug ähnelt einem Hochsicherheitstrakt, auch ansonsten ist Verschlossenheit ein Teil der Firmenphilosophie. Der Erfolg gibt der Crypto AG Recht: Auch nach über 75 Jahren Firmengeschichte ist das Unternehmen bestens im Geschäft.

40.2.3 Cryptovision

Cryptovision ist ein Unternehmen mit Sitz in Gelsenkirchen, das auf kryptografische Produkte und Dienstleistungen spezialisiert ist. Man kann auch von einem Krypto-Zulieferer sprechen, denn die Lösungen des Unternehmens sind größtenteils Komponenten, die in andere Produkte integriert werden. Cryptovision-Entwicklungen finden sich beispielsweise in elektronischen Ausweisen, Smartcard-Betriebssystemen, Navigationssystemen, Schließanlagen, Messgeräten und Motorsteuerungen. Das bekannteste Produkt des Unternehmens ist die Smartcard-Middleware *cv act sc/interface*. Zum Portfolio zählen außerdem eine PKI-Software und eine E-Mail-Verschlüsselungslösung. Durch die Vielzahl der Angebote und die konsequente Spezialisierung kann man Cryptovision als Hardcore-Krypto-Anbieter bezeichnen. Im Bereich der ECC-Verfahren zählt Cryptovision sogar weltweit zu den wichtigsten Anbietern und verfügt über zahlreiche Patente.

40.2.4 CryptWare

Die Firma CryptWare mit Sitz in Bad Camberg bietet ein interessantes Portfolio an Verschlüsselungsprodukten mit dem Schwerpunkt auf Festplatten-, Datei- und Verzeichnisverschlüsslung. Das bekannteste Produkt ist *CryptoPro Secure Disk for BitLocker*, das die in Windows-Betriebssysteme eingebaute Festplattenverschlüsselung BitLocker um einige wichtige Funktionen ergänzt [Schm12/2].

40.2.5 Entrust Technologies

Entrust ist eine kanadische Firma, die im Bereich PKI aktiv ist. Das wichtigste Produkt dieses Anbieters ist *Entrust Authority*, eine Softwarelösung für den CA- und RA-Betrieb. Außerdem bietet Entrust eine ganze Reihe von PKI-Anwendungen und Identity-Management-Lösungen. Nach der Windows CA gilt Entrust Authory als die am zweitmeisten verbreitete PKI-Lösung auf dem Markt. Sie kann mit einer Vielzahl von Funktionen und unterstützten Partnerprodukten aufwarten. Auch Faktoren wie Stabilität, Support und die Qualität der Beratung sprechen für diese Lösung. Für große, komplexe PKI-Projekte ist Entrust Authority daher oft erste Wahl. Allerdings gibt es für weniger anspruchsvolle PKIs oft kostengünstigere Alternativen.

40.2.6 Rohde & Schwarz SIT

Rohde & Schwarz SIT ist ein Tochterunternehmen des traditionsreichen Elektronik-Herstellers Rohde & Schwarz. Im Gegensatz zur Konzernmutter, die in München ansässig ist, hat Rohde & Schwarz SIT seinen Sitz in Berlin. Rohde & Schwarz SIT ging nach der deutschen Wiedervereinigung aus der für Verschlüsselung zuständigen Behörde ZCO (Zentrales Chiffrierorgan) der DDR hervor [Schm11]. In den neunziger Jahren konnte sich das Unternehmen am Markt etablieren. Rohde & Schwarz SIT ist auf Krypto-Hardware für niedrige OSI-Ebenen spezialisiert. Im Portfolio des Unternehmens finden sich unter anderem Hardwaremodule für die ISDN-, ATM- und GSM-Verschlüsselung. Kunden sind größtenteils Organisationen mit besonders hohem Sicherheitsbedarf, also vor allem Militär- und Staatsorganisationen.

40.2.7 RSA Security

RSA Security, eine Firma mit Sitz in den USA, gehört zweifellos zu den weltweit ersten Adressen in der Kryptografie. Der Name geht auf Ron Rivest, Adi Shamir und Leonard Adleman zurück, die nicht nur das RSA-Verfahren erfanden, sondern auch eine Firma namens *RSA Data Security* gründeten. Dieses nur mäßig erfolgreiche Unternehmen wurde 1996 vom deutlich größeren Anbieter Security Dynamics übernommen, der sich später in RSA Security umbenannte. RSA Secu-

rity ist vor allem durch die SecurID-Karte bekannt, die ich bereits in Abschnitt 21.3.1 vorgestellt habe. Obwohl oder gerade weil dieses Produkt kryptografisch eher anspruchslos ist, wurde es enorm erfolgreich, was wieder einmal zeigt, dass pragmatische Lösungen oft den größten Erfolg haben.

40.2.8 Secude

Secude, ein Unternehmen mit Hauptsitz in Luzern, gehört seit 15 Jahren zum Establishment der deutschsprachigen Krypto-Branche. Allerdings haben sich Besitzer, Standorte und Produktportfolio in dieser Zeit mehrfach geändert. Gegründet wurde Secude 1997 als Spin-off der Gesellschaft für Mathematik und Datenverarbeitung (GMD) in Darmstadt, wobei SAP und die Deutsche Bank Starthilfe leisteten. 2002 übernahm Secude den Schweizer Krypto-Anbieter IT_SEC und wuchs damit zu einem der führenden Krypto-Produkthersteller mit breiter Angebotspalette. Das Geschäft lief allerdings schlechter als erwartet, und so schlitterte das Unternehmen in die Insolvenz. Es fand sich jedoch eine Schweizer Investorengruppe, die neues Kapital zur Verfügung stellte und Secude neu formierte. Der Hauptsitz wurde in die Schweiz verlegt. 2011 übernahm SAP einen Teil des Unternehmens inklusive Mitarbeitern und Produkten. Zum Portfolio des Unternehmens zählen heute verschiedene Krypto-Lösungen für die Datei- und Festplattenverschlüsselung.

40.2.9 Secunet

Die Essener Firma **Secunet** ist das größte deutsche Beratungsunternehmen im Bereich der IT-Sicherheit. Über 200 Mitarbeiter sind in den diversen Niederlassungen aktiv. Zu den Geschäftsfeldern gehören Public-Key-Infrastrukturen, digitale Signaturen, Virtuelle Private Netze und Smartcards, aber auch nichtkryptografische Themen wie Firewalls und Sicherheitsmanagement. Nachdem das Unternehmen 1996 als reine Beratungsfirma startete, betätigt sich Secunet inzwischen auch als Produktanbieter. Das mit Abstand erfolgreichste Produkt im Portfolio ist die VPN-Lösung *SINA*, die sich vor allem bei Behörden und im Hochsicherheitsumfeld großer Beliebtheit erfreut. Secunet war ursprünglich ein Spinoff des TÜV Informationstechnik und als solches eine Tochter des Rheinisch-Westfälischen TÜV. Inzwischen gehört das Unternehmen mehrheitlich dem Smartcard- und Wertdruck-Spezialisten Giesecke & Devrient. Nicht zuletzt dank der mächtigen Anteilseigner im Rücken gilt Secunet seit Jahren als solide Größe in der deutschen IT-Sicherheit.

40.2.10 Secusmart

Secusmart ist ein noch junges Unternehmen mit Sitz in Düsseldorf, das sich auf die Verschlüsselung von Mobilfunkverbindungen spezialisiert hat. Das Produkt *SecuVOICE* lässt sich auf verschiedenen Smartphones nutzen.

40.2.11 Sophos

Bis vor einigen Jahren zählte die 1983 gegründete Firma *Utimaco Safeware* zu den wichtigsten Krypto-Anbietern in Deutschland. Zum Portfolio gehörten Produkte für die E-Mail-Verschlüsselung, für die Absicherung des PC-Zugangs, für die Festplattenverschlüsselung, für den Betrieb von VPNs und einige mehr. 2009 wurde Utimaco vom Virenschutz-Spezialisten **Sophos** übernommen. Die meisten ehemaligen Utimaco-Produkte werden seitdem unter dem Herstellernamen Sophos angeboten – zusammen mit einigen weiteren Krypto-Anwendungen. Der Markenname Utimaco blieb jedoch erhalten – unter diesem verkauft Sophos heute HSMs und Lösungen für die Kommunikationsüberwachung.

40.2.12 Zertificon

Die Berliner Firma Zertificon ist seit über zehn Jahren am Markt. Ihr wichtigstes Produkt ist das *Z1 SecureMail* Gateway, eine der erfolgreichsten Lösungen für die Gateway-basierte E-Mail-Verschlüsselung (siehe Abschnitt 36.1.1).

40.3 Non-Profit-Organisationen

Neben kommerziellen Unternehmen spielen auch verschiedene Behörden und Verbände eine wesentliche Rolle in der Krypto-Szene. Die wichtigsten davon will ich Ihnen nun vorstellen.

40.3.1 BSI

Das 1990 entstandene **Bundesamt für Sicherheit in der Informationstechnik** (**BSI**) ist eine deutsche Bundesbehörde, die für Fragen rund um die IT-Sicherheit zuständig ist. Dies ist eine äußerst vielfältige Aufgabe. Das BSI entwickelt beispielsweise Verschlüsselungslösungen und andere IT-Sicherheitswerkzeuge für Bundesbehörden. Darüber hinaus vergibt es Common-Criteria- und ITSEC-Zertifikate. Alle zwei Jahre richtet das BSI den deutschen Deutschen IT-Sicherheitskongress aus, der in der Branche eine große Bedeutung hat. Auch in der Gesetzgebung redet das BSI häufig ein Wort mit – unter anderem hat es bei der Entwicklung des deutschen Signaturgesetzes eine wesentliche Rolle gespielt. In der Öffentlichkeit ist das BSI vor allem dadurch bekannt, dass es bei Bedarf (etwa, wenn ein Computervirus sein Unwesen treibt) Sicherheitswarnungen veröffentlicht, über die an-

schließend in den Medien berichtet wird. Eine andere Aufgabe des BSI besteht darin, im Auftrag von Polizei und Geheimdiensten Verschlüsselungen zu knacken. Naturgemäß ist nur wenig Konkretes über diese Tätigkeit bekannt. Mehr zur Geschichte des BSI und deren Vorgängerbehörde findet sich in [Schm11].

40.3.2 Bundesnetzagentur

Mit vollständigem Namen heißt diese Behörde *Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen*. Sie ist die Nachfolgerin des Bundespostministeriums, das nach der Privatisierung der Post aufgelöst wurde. Hauptsitz der Bundesnetzagentur ist Bonn. Aus Kryptografen-Sicht ist die Bundesnetzagentur interessant, weil sie die Wurzel-CA des deutschen Signaturgesetzes betreibt.

40.3.3 IACR

Die **International Association for Cryptologic Research** (**IACR**) ist ein nichtkommerzieller, internationaler Zusammenschluss von Kryptografen. Bekannt ist die IACR zum einen als Veranstalterin der Crypto- und Eurocrypt-Konferenzen, die als Mekka der wissenschaftlichen Kryptografie gelten. Zum anderen gibt die 1982 gegründete Organisation mit dem *Journal of Cryptology* die bedeutendste kryptografische Fachzeitschrift heraus. Die IACR hat einen mathematisch geprägten Blick auf die Kryptografie. Netzwerkprotokolle, der Krypto-Markt, Evaluierung und einige weitere praxisrelevante Themen spielen in dieser Organisation keine große Rolle.

40.3.4 NSA

Die amerikanische Geheimbehörde **NSA** (**National Security Agency**) wurde in diesem Buch schon so oft erwähnt, dass ein Kapitel darüber fast schon überfällig ist. Gegründet wurde diese sagenumwobene Organisation 1952 als Nachfolgeorganisation der AFSA (Armed Forces Security Agency) innerhalb des US-Verteidigungsministeriums. Die wichtigste Aufgabe der NSA ist das Abhören von allen ausländischen Kommunikationskanälen, die für die Sicherheit der Vereinigten Staaten von Interesse sind. Daneben ist die NSA auch bei der Entwicklung kryptografischer Verfahren aktiv.

Um ihre Aufgabe zu erfüllen, unterhält die NSA nicht nur zahlreiche Bodenstationen für das Abhören von Funkstrecken, sondern setzt auch Satelliten und Aufklärungsflugzeuge ein. Kompromittierende Abstrahlung gehört ebenfalls zum NSA-Repertoire. Gerüchte besagen zudem, dass jedes Telefonat, das aus den USA mit einem ausländischen Gesprächspartner geführt wird, von der NSA mitgeschnitten wird. Auch in Europa dürfte kaum ein Telefonat und kaum eine E-Mail vor der NSA sicher sein.

Abb. 40–3 *Die NSA-Zentrale in Fort Meade (US-Bundesstaat Maryland)*

Die Zahl der NSA-Angestellten wird auf mehrere Zehntausend geschätzt. Dies ist mehr, als jeder andere US-Geheimdienst beschäftigt. Außerdem gilt die NSA als der weltweit größte Abnehmer von Hardware. Man geht zudem davon aus, dass die NSA bei der computergestützten Verarbeitung großer Datenmengen weltweit führend ist. Bekannt ist etwa, dass die Geheimbehörde ein hochwertiges Sprach-erkennungssystem namens Oratory verwendet, das Telefongespräche auswerten kann. Das Computersystem Echelon (siehe Abschnitt 3.5.4) verarbeitet riesige Informationsmengen, die einem Programm namens MEMEX zugeführt werden, das mithilfe künstlicher Intelligenz die interessanten Daten herausfiltert. Nur mit diesen beschäftigen sich die NSA-Angestellten anschließend persönlich.

Über das kryptografische Know-how der NSA werden wahre Wunderdinge erzählt. Die NSA gilt als weltweit größter Arbeitgeber für Mathematiker. Gerüchten zufolge ist die NSA in Sachen Kryptografie der öffentlichen Forschung um Jahre voraus (auch wenn der Vorsprung inzwischen deutlich kleiner geworden sein dürfte). Viele Kryptografen sehen die NSA kritisch. Dem Einfluss der NSA ist es beispielsweise zu verdanken, dass der DES nur mit 56 Schlüssel-Bits ausgestattet wurde. Die Designkriterien für den DES und den ebenfalls unter NSA-Mithilfe entstandenen DSA wurden nicht oder erst sehr spät bekannt gemacht. Das im Clipper-Chip eingebaute Skipjack-Verfahren wurde zunächst gar nicht veröffentlicht. Die NSA war es auch, die den Streit mit Phil Zimmermann um den Export von PGP austrug, und selbst die RSA-Erfinder Rivest, Shamir und Adleman wollte man zur Geheimhaltung ihres Verfahrens zwingen. Nicht zuletzt ist auch das ehemalige US-Exportverbot für starke Kryptografie auf den Einfluss der NSA zurückzuführen.

40.3.5 Teletrust

Teletrust ist der Verband der deutschen Computer-Security-Branche. Über 150 Anbieter von IT-Sicherheit, darunter auch viele aus dem Bereich Kryptografie, sind Mitglied dieser Vereinigung. Teletrust ist einerseits eine Lobbyorganisation, die sich beispielsweise gegen eine gesetzliche Regulierung von Kryptografie starkmachte (inzwischen ist dieses Thema vom Tisch), und andererseits eine Plattform für gemeinsame Aktivitäten der Mitglieder. Teletrust-Gremien beschäftigen sich unter anderem mit Identity Management, Public-Key-Infrastrukturen und Biometrie. Der Verband ist zudem Mitveranstalter der *ISSE*-Konferenz (*Information Security Solutions Europe*) und betreibt die European Bridge-CA. Auf der RSA-Konferenz in San Francisco gibt es alljährlich einen von Teletrust organisierten Stand, auf dem sich deutsche Krypto-Anbieter präsentieren. Außerdem wurde innerhalb von Teletrust das Expertenzertifikat *TISP* (*Teletrust Information Security Professional*) entwickelt, mit dem sich IT-Sicherheitsfachleute ihr Wissen offiziell bestätigen lassen können.

40.4 Kryptoanalyse-Wettbewerbe

Die wichtigste Antriebskraft in der Kryptografie ist auch nach über 3.000 Jahren immer noch der Wettlauf zwischen Codeerfindern und Codeknackern. Bereits mehrfach haben Organisationen oder Einzelpersonen versucht, diesen Wettlauf in Form eines Kryptoanalyse-Wettbewerbs zu institutionalisieren. Einige davon schauen wir uns im Folgenden an.

40.4.1 Die RSA-Challenges

Die US-Firma RSA Data Security (heute RSA Security) hat bereits mehrere Wettbewerbe veranstaltet, in denen es für die Entschlüsselung eines Geheimtexts oder für die Faktorisierung eines Primzahlprodukts einen Geldpreis zu gewinnen gab. Diese Wettbewerbe wurden als **RSA-Challenges** bezeichnet. Sie sind nach der Firma RSA Data Security (und nicht etwa nach dem RSA-Verfahren) benannt. Die RSA-Challenges erregten zeitweise großes Aufsehen, allerdings ließ das öffentliche Interesse im Laufe der Jahre deutlich nach. 2007 beendete der Veranstalter schließlich die zu diesem Zeitpunkt noch laufenden Wettbewerbe.

Faktorisierungswettbewerbe

1991 rief man im Hause RSA Data Security einen Faktorisierungswettbewerb ins Leben. In dieser Langzeitveranstaltung ging es darum, vom Ausrichter vorgegebene Primzahlprodukte in ihre beiden Faktoren zu zerlegen. Gelingt eine solche Faktorisierung, dann kommt dies dem Knacken eines RSA-Schlüssels gleich (jetzt ist das Verfahren RSA gemeint, nicht die Firma). Die erste Zahl, die der Veran-

stalter im März 1991 vorgab, hatte 100 Stellen (330 Bit) und wurde daher als *RSA-100* bezeichnet.

Schon im April 1991 gelang es dem bekannten Kryptografen Arjen Lenstra, RSA-100 zu faktorisieren. Damit war klar, dass eine RSA-Schlüssellänge von 330 Bit nicht mehr ausreichte. In den Folgejahren fielen auch *RSA-110*, *RSA-120* und einige weitere Zahlen aus dem Faktorisierungswettbewerb. Für den Höhepunkt sorgte im Mai 2005 ein deutsches Team unter der Leitung des Bonner Mathematikers Jens Franke, als es die Faktorisierung von *RSA-200* bekannt gab. Mit 663 Bit ist RSA-200 bis heute die längste Zahl, die öffentlich faktorisiert wurde. Der Fortschritt in der Faktorisierungstechnik führte dazu, dass mittlerweile 1.024 Bit als Mindestlänge für einen RSA-Schlüssel gelten.

2001 präsentierte der inzwischen in RSA Security umbenannte Veranstalter acht weitere Zahlen, die es zu faktorisieren galt. Dieses Mal wurden die Zahlen nicht nach ihrer Länge im Dezimalsystem, sondern nach ihrer Bit-Länge benannt: *RSA-576*, *RSA-640*, *RSA-704*, *RSA-768*, *RSA-896*, *RSA-1024*, *RSA-1536* und *RSA-2048*. Zudem gab es nun attraktive Geldpreise zu gewinnen. Für das Faktorisieren von RSA-2048 (also einer 2.048-Bit-Zahl) lobte der Veranstalter immerhin 200.000 Dollar aus. Die bereits erwähnte Forschergruppe um Jens Franke blieb auch bei den neuen Zahlen am Ball und knackte im November 2005 RSA-640, wofür sie 20.000 Dollar erhielt. Dies war allerdings nur die zweitlängste bis dahin faktorisierte Zahl, da die bereits erwähnte RSA-200 mit 663 Bit noch ein Stück länger ist. Die kleinste Zahl, die die RSA-Challenges unfaktorisiert überstand, ist RSA-704. Die Prämie dafür lag bei 30.000 Dollar. Auch wenn es diese Summe seit Beendigung des Wettbewerbs nicht mehr zu gewinnen gibt, können Sie sich nach wie vor an einer Faktorisierung versuchen. Hier ist die Zahl (bevor Sie loslegen, bitte die Korrektheit auf der Internetseite des Wettbewerbs überprüfen):

```
74037563479561712828046796097429573142593188889231289084936232638972765034028266276891996419625117843995894330502127585370118968098286733173273108930900552505116877063299072396380786710086096962537934650563796359
```

DES-Wettbewerbe

1997 setzten die RSA-Leute erstmals einen Geldpreis für das Knacken eines symmetrischen Verfahrens aus. 10.000 US-Dollar sollte erhalten, wer eine DES-verschlüsselte Nachricht per Known-Plaintext-Attacke dechiffrieren konnte. Nach vier Monaten war unter Beteiligung von 14.000 Rechnern der Schlüssel gefunden (siehe Abschnitt 6.3). Die Öffentlichkeit nahm mit Erstaunen zur Kenntnis, dass das wichtigste aller Verschlüsselungsverfahren erstmals gebrochen worden war – wenn auch nur mit bekanntem Klartext. Kein anderer Kryptoanalyse-Wettbewerb hat bis heute ein so großes Medienecho gefunden.

Der große Erfolg des ersten DES-Wettbewerbs veranlasste den Veranstalter der RSA-Challenges zu mehreren Neuauflagen. In der zweiten Runde gab es zwei DES-Geheimtexte zu knacken (*DES II-1 Challenge* und *DES II-2 Challenge*), was in 39 Tagen bzw. 56 Stunden auch gelang. Im ersteren Fall war die **distributed.net**-Gruppe am schnellsten, die sich auf das verteilte Berechnen rechenzeitintensiver Aufgabenstellungen spezialisiert hat. Die DES II-2 Challenge wurde von einem Spezialrechner der Internet-Bürgerrechtsbewegung **Electronic Frontier Foundation** (**EFF**) gelöst. Im Januar 1999 startete die dritte DES-Runde (*DES III Challenge*). Dieses Mal arbeiteten die distributed.net-Gruppe und die EFF zusammen, und in weniger als 23 Stunden war der Schlüssel gefunden. Dieser Erfolg markierte gleichzeitig den Höhe- und Wendepunkt in der Geschichte der Kryptoanalyse-Wettbewerbe. Das öffentliche Interesse ließ in Folge der ständig neuen Erfolgsmeldungen deutlich nach. Der Ausrichter verzichtete auf neue Wettbewerbe, und so wurde der Rekord im DES-Knacken bis heute nicht mehr verbessert.

RC5-Wettbewerb

Neben dem DES unterzog RSA Data Security auch das im eigenen Haus entwickelte symmetrische Verfahren RC5 einem Kryptoanalyse-Wettbewerb. Der Aufruf zum Knacken von RC5 diente unter anderem der Demonstration, dass das Verfahren bei ausreichender Schlüssellänge sicher ist. Da RC5 eine variable Blocklänge, Schlüssellänge und Rundenzahl vorsieht, bot es sich an, mehrere unterschiedliche Varianten des Verfahrens knacken zu lassen. Die Veranstalter entschieden sich dazu, Blocklänge (64 Bit) und Rundenzahl (12) konstant zu lassen, dafür aber die Schlüssellänge zu variieren.

Im Januar 1997 startete der RC5-Wettbewerb. Die Teilnehmer konnten sich an 12 RC5-Geheimtexten versuchen, die mit den Schlüssellängen 40 Bit, 48 Bit, 56 Bit, 64 Bit, 72 Bit, 80 Bit, 88 Bit, 96 Bit, 104 Bit, 112 Bit, 120 Bit und 128 Bit verschlüsselt worden waren. Da ein Teil des jeweils zugehörigen Klartexts bekannt gemacht wurde, war auch hier eine Known-Plaintext-Attacke gefragt. Da RC5 bisher keine praktisch verwertbaren Schwächen offenbart hat, mussten es die Teilnehmer mit einer vollständigen Schlüsselsuche versuchen.

Der 40-Bit-Geheimtext des RC5-Wettbewerbs war bereits nach gut drei Stunden geknackt. Die 48-Bit-Variante fiel nach 13 Tagen. Nach 265 Tagen fand die bereits erwähnte distributed.net-Gruppe auch den 56-Bit-Schlüssel. Danach wurde es erst einmal still um den RC5-Wettbewerb. Im Juli 2002 gelang es schließlich erneut distributed.net, nach fünfjähriger Rechenzeit den 64-Bit-Geheimtext zu ermitteln. Dieser Erfolg bildet bis heute den Weltrekord in Sachen vollständiger Schlüsselsuche. Es gibt also keinen anderen öffentlich bekannten Fall, in dem ein Schlüssel der Länge 64 Bit oder mehr durch Brute Force geknackt wurde. Mit dem Ende der RSA-Challenges im Jahr 2007 ist auch der RC5-Wettbewerb ausgelaufen. Dies sollte jedoch niemanden daran hindern, durch das Knacken des 72-Bit-Schlüssels einen neuen Weltrekord aufzustellen.

40.4.2 Mystery Twister C3

Mystery Twister C3 (**MTC3**) ist der derzeit bedeutendste Kryptoanalyse-Wettbewerb weltweit. Im Rahmen von MTC3, das Ende 2010 an den Start ging, sind derzeit etwa 140 kryptografische Rätsel (Challenges) auf der Webseite www.mysterytwisterc3.org abrufbar. Jeder kann mitmachen und für eventuell gefundene Lösungen Punkte einsammeln. Auch Preise gibt es zu gewinnen. Die Schwierigkeitsgrade der Rätsel sind recht unterschiedlich, sie reichen von recht einfach (Level I) über moderat (Level II) und schwierig (Level III) bis zu bisher ungelöst (Level X). Neben vielen klassischen Krypto-Verfahren spielen auch moderne Algorithmen wie RSA und AES eine Rolle. So hat MTC3 die noch ungelösten Faktorisierungsaufgaben der RSA-Challenges übernommen. Eine andere Aufgabe sieht vor, 65 Bits eines AES-Schlüssels per vollständiger Schlüsselsuche zu ermitteln (die restlichen 63 Bits sind bekannt). Sollte dies gelingen, dann wäre es ein neuer Weltrekord. Die bisherige Bestleistung beträgt 64 Bits und stammt aus dem RC5-Wettbewerb der RSA-Challenges.

MTC3 ist ein Gemeinschaftsprojekt mehrerer Universitäten und Initiativen. Beteiligt sind unter anderem die Professoren Bernhard Esslinger (Universität Siegen), Alexander May (Universität Bochum) und Arno Wacker (Universität Kassel). Ziel von MTC3 ist es, die Verschlüsselungstechnik für ein breiteres Publikum attraktiv zu machen. Wer Lust hat, kann nicht nur Challenges lösen, sondern auch selbst welche einreichen.

40.5 Die zehn größten Krypto-Flops

Schadenfreude ist auch für Kryptografen die schönste Freude. Werfen wir deshalb im Folgenden einen Blick auf die größten Flops, die den Weg der Kryptografie in den letzten Jahrzehnten pflastern.

40.5.1 Flop in Massenproduktion: GSM-Verschlüsselung

Das Mobilfunksystem GSM unterstützte als eines der ersten Kommunikationsnetze von Haus aus Verschlüsselung und Authentifizierung. Zu diesem Zweck nutzen GSM-Handys und Basisstationen die Algorithmen A3, A5 und A8. Wie Sie in den Abschnitten 16.3.2 und 32.2 erfahren haben, ist die Qualität dieser Verfahren leider recht bescheiden. Zwar ist nicht davon auszugehen, dass die Urheber dieser Algorithmen die diversen Schwächen absichtlich einbauten (obwohl dies teilweise vermutet wurde), doch die notwendige Sorgfalt ließen die namentlich nicht bekannten Chiffren-Designer offensichtlich vermissen. Bedenkt man, dass A3, A5 und A8 durch die weite Verbreitung von GSM zu den meistverwendeten Krypto-Algorithmen überhaupt gehören, dann muss man diese leider zu den größten Flops der Krypto-Geschichte zählen.

40.5.2 Die Rechnung ohne den Wirt gemacht: Clipper

Clipper war der Name eines Krypto-Chips für die Verschlüsselung von Telefongesprächen, der in den neunziger Jahren von der NSA entwickelt wurde. Ein ähnlicher Chip für Computerdaten hieß **Capstone**. Clipper verwendete ein symmetrisches Verschlüsselungsverfahren namens Skipjack, dessen Funktionsweise geheim gehalten wurde. Die wichtigste Besonderheit des Chips bestand darin, dass er eine kryptografische Hintertür enthielt. Er verwendete ein kryptografisches Protokoll, das bei jeder verschlüsselten Datenübertragung das Entschlüsseln mit einem Zusatzschlüssel (*Unit Key*) erlaubte. Der Unit Key war für jedes Exemplar des Chips verschieden.

Die Entwicklung von Clipper erfolgte im Auftrag der US-Regierung, die damit einen besonderen Plan verfolgte. Sie wollte den Chip zum Bestandteil sämtlicher in den USA verwendeten Telefone machen. Dadurch sollte der US-Telefonverkehr flächendeckend abhörsicher gestaltet werden, wobei eine staatliche Behörde mithilfe des Unit Key die Möglichkeit zum Mithören erhalten sollte. Um dies sicher zu gestalten, sollte jeder Unit Key in zwei Teile zerlegt werden, die von zwei voneinander unabhängigen Einrichtungen aufbewahrt wurden. Ein Abhören war nur nach einem entsprechenden Gerichtsbeschluss zulässig.

Auf den ersten Blick war Clipper eine sinnvolle Sache. Erstmals in der Geschichte sollte damit ein öffentliches Telefonnetz mit fest eingebauter Verschlüsselung ausgestattet werden. Kriminelle und Spione hätten dadurch keine Chance mehr zum Abhören eines Telefongesprächs gehabt. Nur der Staat sollte mithören dürfen, und das nur unter gesetzlich festgelegten Voraussetzungen. Wem diese Einschränkung nicht passte, der konnte problemlos und legal ein zusätzliches Verschlüsselungsmodul einsetzen, das ohne Hintertür arbeitete. Bestehende Telefon-Verschlüsselungslösungen wurden von Clipper nicht beeinträchtigt.

Bedenkt man, dass es in den USA Mitte der neunziger Jahre so gut wie gar keine Telefonverschlüsselung gab, dann war das Clipper-Projekt ein großer Fortschritt. Dies dachte wohl auch die US-Regierung, doch sie hatte die Rechnung ohne den Wirt gemacht. Denn Kryptografen, Bürgerrechtler und kritische Zeitgenossen hielten reichlich wenig von Clipper. Ein unbekannter Algorithmus und eine eingebaute staatliche Hintertür erschien ihnen per se suspekt. Außerdem befürchteten viele, dass auf Clipper irgendwann ein Verbot der freien Verwendung von Kryptografie folgen könnte. In den Augen der Skeptiker sorgte eine breite Anwendung des Chips zudem für ein falsches Gefühl von Sicherheit und behinderte die Verbreitung besserer Lösungen.

Zusätzliche Munition erhielten die Clipper-Gegner durch eine Schwachstelle des Chips, die der Kryptograf Matt Blaze entdeckte. Ihm fiel auf, dass sich durch eine leichte Manipulation der Protokollnachrichten der zur Verschlüsselung verwendete Schlüssel ändern ließ. Dadurch wurde der Unit Key wirkungslos. Auf diesen Protokollangriff war die NSA offenbar nicht gekommen, was wieder ein-

mal bestätigte, dass das Design eines kryptografischen Protokolls oft schwieriger ist als das eines Krypto-Verfahrens.

All diese Misserfolge bewogen die US-Regierung schließlich zu einem Rückzieher. Einer der größten Flops der Krypto-Geschichte war damit perfekt. Ob die Verhinderung von Clipper wirklich etwas gebracht hat, ist dagegen umstritten. Denn nach wie vor wird praktisch der gesamte Telefonverkehr der USA unverschlüsselt abgewickelt. Der Verzicht auf Clipper hat also nicht dafür gesorgt, dass sich bessere Lösungen verbreiteten. Im Gegenteil: Im Vergleich zur jetzigen Situation wäre Clipper wohl das kleinere Übel gewesen.

40.5.3 Das Bare war nicht das Wahre: Ecash

Zu den größten Flops der Krypto-Geschichte muss man leider auch das vielgelobte Online-Bezahlsystem **Ecash** zählen. Ecash wurde von dem Kryptografen David Chaum erfunden und von dessen Firma Digicash vermarktet. Ecash war vermutlich die anspruchsvollste Krypto-Implementierung überhaupt auf dem Markt. Der Zweck von Ecash war es, Bargeld allein mit Softwaremitteln zu simulieren. Um dieses scheinbar unmögliche Vorhaben zu realisieren, entwickelte Chaum ein ausgeklügeltes Protokoll, das einerseits das doppelte Ausgeben des digitalen Bargelds verhinderte, andererseits aber für den Bezahler anonym ablief. Die Krypto-Welt war von Ecash begeistert.

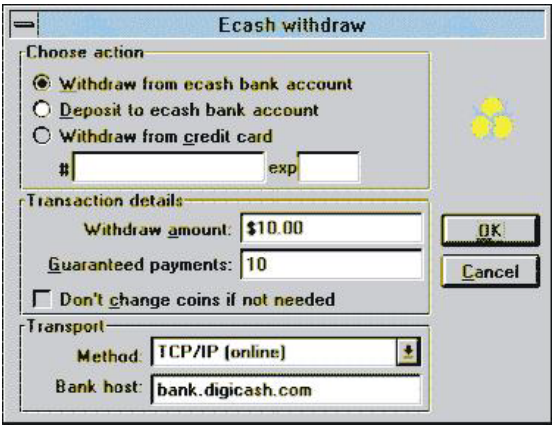


Abb. 40–4 *Ecash erwies sich als zu kompliziert und floppte daher.*

Leider floppte Ecash in der Praxis, und die Firma Digicash ist inzwischen vom Markt verschwunden (eine ausführliche Beschreibung der Hintergründe gibt es in [Schm02]). Der Misserfolg lag zum einen sicherlich daran, dass Internet-Zahlungssysteme allgemein zunächst kaum Fuß fassen konnten. Zusätzlich erwies sich Ecash trotz einfacher Bedienung als zu kompliziert für die Praxis. Immer wieder wurde von technischen Problemen berichtet. Wie Ecash funktioniert, ver-

stand ohnehin kaum ein Kunde. So musste wieder einmal eine technisch brillante Lösung weniger anspruchsvollen Konkurrenten den Vortritt lassen. Schade.

40.5.4 Kryptografie unterschätzt: WEP

Das kryptografische Netzwerkprotokoll WEP (siehe Abschnitt 33.2) entstand zu einer Zeit, als das Design eines hochwertigen Krypto-Systems längst kein Hexenwerk mehr war. Als die WEP-Entwickler Mitte der neunziger Jahre ihre Arbeit aufnahmen, gab es bereits zahlreiche Experten, die bei einem solchen Unterfangen hätten helfen können, und an Fachliteratur mangelte es ebenfalls nicht. Umso peinlicher ist es, dass WEP aufgrund eines fehlerhaften Einsatzes des Verschlüsselungsverfahrens RC4 eine völlig unnötige Sicherheitslücke aufwies. 2004 gelang es Fluhrer, Mantin und Shamir, das Protokoll mit moderatem Aufwand zu knacken. Den Standardisierern blieb nichts anderes übrig, als eine Nachbesserung in Angriff zu nehmen, die zu den bereits im Einsatz befindlichen Geräten kompatibel sein musste und daher äußerst kompliziert ausfiel. Die Moral dieser Geschichte: Wer die Kryptografie unterschätzt, ist auf dem besten Weg, einen Flop zu produzieren.

40.5.5 FEAL hilft nicht viel: FEAL

1987 veröffentlichten zwei japanische Kryptografen das Verschlüsselungsverfahren FEAL (Fast Encryption Algorithm) [Schn96]. FEAL war als Alternative zum DES gedacht. Die 56 Schlüssel-Bits des DES galten bereits damals als zu wenig, und zudem störte viele, dass der DES als Softwareimplementierung recht langsam ist. FEAL sollte diese Schwächen beheben. Das Verfahren, das mit 64 Schlüssel-Bits arbeitete, ähnelte dem DES, hatte jedoch nur vier Runden und verwendete Operationen, die bei einer Softwarerealisierung besonders schnell sind. Untersuchungen der Diffusions- und Konfusionseigenschaften erbrachten teilweise bessere Werte als beim DES.

War damit zehn Jahre nach der Veröffentlichung des DES eine ernst zu nehmende Alternative zu diesem gefunden? Nein, denn diverse Kryptografen, die FEAL unter die Lupe nahmen, entdeckten darin Schwachstellen wie Löcher im Schweizer Käse. Bereits einige Monate nach Veröffentlichung des Verfahrens erschien die erste erfolgreiche Kryptoanalyse. Weitere wirksame Angriffe veranlassten die Entwickler zu Gegenmaßnahmen: Getreu dem Motto »viel hilft viel« erhöhten sie die Rundenzahl zunächst auf acht, später auf 16 und schließlich auf eine variable Zahl. Von einem ursprünglich sehr performanten Verfahren entwickelte sich FEAL dabei zu einer äußerst lahmen Angelegenheit.

Der beste bisher bekannte Angriff auf FEAL betrifft die Vier-Runden-Variante. Es handelt sich um eine Known-Plaintext-Attacke mit nur fünf Textblöcken [Schn96]. Kein anderes Verfahren aus der wissenschaftlichen Kryptografie ist

auch nur annähernd so schwach. Doch FEAL zeigte immerhin, dass auch der Misserfolg berühmt machen kann. Da sich das Verfahren als dankbares Opfer zahlreicher Codeknacker erwies, kann man ihm eine entscheidende Rolle in der Geschichte der Kryptografie zubilligen. Noch heute ist das Verfahren in Gebrauch – als Anschauungsobjekt in Kryptoanalyse-Kursen.

40.5.6 Tag der offenen Tür: DigiNotar

Das Jahr 2011 ging als das Jahr der gehackten Zertifizierungsstellen in die Geschichte ein. Zunächst musste der US-Anbieter Comodo einräumen, dass ein Hacker nach Überwindung der Sicherheitsbarrieren einige falsche digitale Zertifikate ausgestellt hatte [Schmid]. Einige Monate später wurde die israelische Firma StartCom Opfer eines ähnlichen Angriffs [Bach11]. Zur Ausstellung falscher Zertifikate kam es dieses Mal zwar nicht, doch aus Sicherheitsgründen musste das Unternehmen einige Wochen lang den CA-Betrieb unterbrechen. Auch bei GemNet und GlobalSign wurde 2011 Hacker-Alarm ausgelöst, allerdings war bei ersterem CA-Betreiber die Zertifizierungsstelle nicht betroffen, und bei letzterem handelte es sich (angeblich) um einen falschen Alarm.

Während die vier genannten Firmen mit einem blauen Auge davonkamen, rutschte der niederländische CA-Betreiber DigiNotar nach einem Hacker-Einbruch in die Pleite [Borche]. Anfang September 2011 war bekannt geworden, dass ein Angreifer über 500 falsche Zertifikate (unter anderem für die Domain google.com) ausgestellt hatte. Alle Browserhersteller entfernten daraufhin die CA-Zertifikate von DigiNotar aus ihren Programmen – auch legitime Zertifikate wurden dadurch nicht mehr anerkannt. Da DigiNotar in den Niederlanden auch für den Staat arbeitete, waren zahlreiche Behörden betroffen. Der finanzielle Schaden war erheblich. Am 20. September 2011 erklärte ein Gericht DigiNotar für insolvent. Experten zufolge hat das Unternehmen zwei entscheidende Fehler gemacht: Zum einen hat es bei den Sicherheitsvorkehrungen geschlampt, zum anderen hat es nach der Entdeckung des Vorfalls zu lange versucht, die Sache unter den Teppich zu kehren.

40.5.7 Protokolle, die die Welt nicht braucht: PCT

In Kapitel 35 dieses Buchs haben Sie erfahren, dass SSL (Secure Socket Layer) eines der erfolgreichsten kryptografischen Netzwerkprotokolle überhaupt ist. SSL wurde ursprünglich von Netscape entwickelt und später von der IETF als TLS (Transport Layer Security) standardisiert. Die Firma Microsoft wollte den Erfolg von SSL jedoch nicht einfach so hinnehmen, denn schließlich war Netscape seinerzeit ihr größter Herausforderer. So entwickelte man in Redmond einen SSL-Konkurrenzstandard namens *PCT* (*Private Communications Technology*). PCT unterschied sich von SSL nur geringfügig, war aber mit diesem nicht

kompatibel. Angesichts dieser unnötigen Zweigleisigkeit sahen es viele Experten mit Erleichterung, dass sich PCT nicht durchsetzen konnte. Da praktisch niemand dieses Protokoll implementierte, geriet es schnell zum Flop. Während heute alle Welt SSL einsetzt, ist PCT von der Bildfläche verschwunden.

40.5.8 Secure Electronic Tragedy: SET

Kaum ein Teilbereich der Kryptografie floppte so heftig wie die kryptografischen Online-Bezahlsysteme. Als das Internet vor etwa zehn Jahren erwachsen wurde, schossen die Anbieter derartiger Lösungen wie Pilze aus dem Boden. Kein Wunder, schließlich war ein großer Boom des Online-Einkaufens abzusehen, und da wollte sich niemand das Milliardengeschäft des Online-Bezahlens entgehen lassen. Was in der Theorie einleuchtete, ging in der Praxis jedoch gründlich daneben. Während der Online-Handel mit der Zeit tatsächlich zum Boom ansetzte, landete von Millicent bis Clickshare und von First Virtual bis Cybercash ein Online-Bezahlsystem nach dem anderen in der Pleite.

Den wohl größten Misserfolg der Online-Bezahl-Sparte erlebte das Netzwerkprotokoll **SET** (Secure Electronic Transaction). Dabei handelte es sich um ein Protokoll, mit dem Kundin Alice ihre Kreditkartennummer an einen Online-Händler übermitteln und dieser (ohne die Nummer lesen zu können) eine Weiterleitung an eine Clearing-Stelle veranlassen konnte. SET sollte die Sicherheit beim Online-Bezahlen mit Kreditkarte deutlich erhöhen und gleichzeitig den Kreditkartenanbietern ihren Marktanteil im Internet langfristig sichern.

An einflussreichen Fürsprechern mangelte es SET nicht. Neben den Kreditkarten-Riesen Visa, Mastercard und American Express unterstützten auch namhafte Computerfirmen wie Microsoft und IBM das Protokoll. Als 1997 die ersten SET-Pilotprojekte starteten, glaubten viele, die Zukunft des Online-Bezahlens hätte begonnen. Doch aus »Secure Electronic Transaction« wurde schnell eine »Secure Electronic Tragedy«. Offenbar hatten weder die Kunden noch die Verkäufer auf SET gewartet. Zu hohe Gebühren für die Clearing-Stellen und Interessenkonflikte unter den beteiligten Unternehmen wirkten sich ebenfalls wenig förderlich aus. So fing SET nie wirklich Feuer und konnte sich keine nennenswerte Bedeutung verschaffen. Am Ende blieb eine bittere Erkenntnis: Selbst wenn mehrere Weltunternehmen an einem Strang ziehen, kann am Ende ein Flop herauskommen.

40.5.9 Das Stehaufmännchen bleibt liegen: Skipjack

Skipjack (»Stehaufmännchen«) ist das symmetrische Verschlüsselungsverfahren, das von dem Verschlüsselungschip Clipper (siehe Abschnitt 40.5.2) eingesetzt wird. Clipper ist an sich schon einer der größten Krypto-Flops, doch Skipjack sorgte noch für weitere Irritationen. Als 1998 die Funktionsweise des ursprüng-

lich geheimen Verfahrens veröffentlicht wurde, entdeckten die Kryptografen Biham, Biryukov und Shamir eine theoretische Schwachstelle darin (siehe Abschnitt 7.4.6). Zum ersten Mal kamen damit Zweifel an der Kompetenz der NSA auf. Für die scheinbar allmächtige Geheimorganisation war dies zweifellos ein echter Flop.

40.5.10 Stumpfe Allzweckwaffe: S-HTTP

S-HTTP ist ein Mitte der neunziger Jahre entstandenes kryptografisches Netzwerkprotokoll. Es sollte das von Webbrowsern verwendete Protokoll HTTP, das keine Verschlüsselung vorsieht, um kryptografische Mechanismen erweitern. S-HTTP wurde als kryptografische Allzweckwaffe konzipiert, die so ziemlich alle damaligen kryptografischen Formate und Verfahren unterstützte. So konnte man zur Datenübertragung wahlweise PKCS#7, das PGP-Format oder PEM verwenden. Zur symmetrischen Verschlüsselung standen der DES, Triple-DES, IDEA und RC4 zur Verfügung. Das Schlüsselmanagement erfolgte entweder symmetrisch über Kerberos oder asymmetrisch per RSA inklusive digitalen Zertifikaten, wobei neben X.509- auch PKCS#6-Zertifikate unterstützt wurden. Digitale Signaturen waren mit RSA und DSA möglich, wobei MD2, MD5 oder SHA-1 als Hashfunktion genutzt werden konnten.

Doch so schön der kryptografische Gemischtwarenladen auch ausschaute – implementieren wollte dieses komplexe Protokoll kaum jemand. Stattdessen setzten die meisten Hersteller auf das simplere SSL, das zwar nur eine Leitungsverschlüsselung bot, dadurch aber einfacher zu handhaben war. So zeichnete sich spätestens im Lauf des Jahres 1996 ab, dass S-HTTP wohl nie Feuer fangen würde, während SSL einen bis heute andauernden Siegeszug antrat. Wieder einmal hatte sich eine pragmatische Lösung gegen einen ambitionierten Konkurrenten durchgesetzt.

40.6 Murphys zehn Gesetze der Kryptografie

Ganz zum Schluss will ich Ihnen noch die Wahrheit zum Thema Kryptografie verraten. Diese ist in treffender Weise in Murphys zehn Gesetzen der Kryptografie enthalten.

40.6.1 Mallory ist immer schlauer, als man denkt

Dies ist die wichtigste Regel überhaupt in der Kryptografie. Alle folgenden Gesetze sind Spezialfälle dieses Gesetzes, das in seiner Allgemeingültigkeit und praktischen Anwendbarkeit seinesgleichen sucht. Eine vergleichbare Bedeutung hat allenfalls Murphys 10. Gesetz der Kryptografie.

40.6.2 Krypto-Verfahren sind nicht geheim zu halten

Da es nicht möglich ist, ein Krypto-Verfahren geheim zu halten, sollte man es erst gar nicht versuchen.

40.6.3 Neue Krypto-Verfahren sind immer fehlerhaft

Trauen Sie daher nur solchen Verfahren, die gut untersucht sind.

40.6.4 Selbst entwickelte Krypto-Verfahren sind fehlerhaft

Dies ergibt sich eigentlich bereits aus Gesetz 3, wird aber dennoch immer wieder missachtet.

40.6.5 Schlüssel werden kompromittiert

Die Arbeit eines Krypto-Analytikers erleichtert sich ungeheuer, wenn er durch Nachlässigkeiten des Anwenders in Besitz des Schlüssels kommt.

40.6.6 Zufallszahlen sind nicht zufällig

Nichts in dieser Welt geschieht wirklich zufällig, weshalb es Zufallszahlen gar nicht gibt.

40.6.7 Vertraue nichts und niemandem

Dieses Gesetz ist besonders hart, da man in der Kryptografie stets irgendjemandem vertrauen muss.

40.6.8 Anwender sind dumm, faul und interesselos

Anwender sind nicht nur desinteressiert und dumm, sondern auch unendlich faul. Anwender sind grundsätzlich zu faul, um ein Krypto-System überhaupt einzusetzen.

40.6.9 Kryptografie ist fehlerhaft implementiert

Selbst die stärksten Algorithmen ringen Mallory kein müdes Lächeln ab, wenn sie in ein fehlerhaftes Programm integriert sind.

40.6.10 Kryptografie ist sinnlos

Egal, wie sicher eine Nachricht verschlüsselt wird, es gibt trotzdem noch Wege, an sie heranzukommen. Es ist alles nur eine Frage des Aufwands, der Skrupellosigkeit und des Bestechungsgelds. Mit dieser Erkenntnis will ich mein Buch abschließen, ich hoffe, Sie hatten Ihren Spaß damit.

Anhang

Bildnachweis

Vorwort	Bernhard Esslinger
3–1	Public Domain
3–2	GFI
4–4	Public Domain
4–5	Klaus Schmeh
4–6	Public Domain
5–2	Klaus Schmeh
5–4	Klaus Schmeh
5–5	Klaus Schmeh
5-–6	Klaus Schmeh
5–7	Crypto AG
5–8	Klaus Schmeh
5–11	Klaus Schmeh
5–14	Klaus Schmeh
5–15	Klaus Schmeh
8–2	Ferguson, Schroeppel, Whiting
11–3	Klaus Schmeh
17–1	cv cryptovision gmbh, Gelsenkirchen
17–3	Klaus Schmeh
21–3	ekey biometric systems Gmbh & CoKG
21–4	byometric systems AG, Mitterfelden
21–5	FREYLANCE, München
21–7	RSA Security
21–8	Secure Computing
23–2	cv cryptovision gmbh, Gelsenkirchen
23–3	CE-Infosys GmbH
23–4	Public Domain
32–1	Rohde & Schwarz SIT
39–2	Klaus Schmeh
40–1	NSA
40–2	Phil Zimmermann
40–3	NSA

Literatur

[AACRR] Dakshi Agrawal, Bruce Archambeault, Suresh Chari, Josyula R. Rao, Pankaj Rohatgi: *Advances in Side-Channel Cryptanalysis. Electromagnetic Analysis and Template Attacks*. Cryptobytes 1/2003

[AIKMMN] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, Toshio Tokita: *Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms – Design and Analysis*. (im Internet erhältlich) 2000

[AnBiKn] Ross Anderson, Eli Biham and Lars Knudsen: *Serpent. A Candidate Block Cipher for the Advanced Encryption Standard*. (im Internet erhältlich) 1998

[AndBih] Ross Anderson, Eli Biham: *Tiger: A Fast New Hash Function*. (im Internet erhältlich) 1996

[Ande93] Ross Anderson: *Why Cryptosystems Fail*. (im Internet erhältlich) 1993

[Ande94] Ross Anderson: *On Fibonacci Keystream Generators*. Fast Software Encryption, 1994

[Ande08] Ross Anderson: *Security Engineering. A Guide to Building Dependable Distributed Systems*. Wiley & Sons, New York, 2008

[AndKuh] Ross Anderson, Markus Kuhn: *Low cost attacks on tamper resistant devices*. (im Internet erhältlich) 1997

[Aswest] Karl Wassilios Aswestopoulos: *Hellasgate*. Telepolis vom 03.02.2006

[Aust02] *BSI-PP-0006-2002 for Protection Profile – Secure Signature-Creation Device Type 3, Version 1.05*. CEN/ISSS – Information Society Standardization System, Workshop on Electronic Signatures, 2002

[Aust05] *Security Target BSI-DSZ-CC-0346 Version 1.1. Evaluation of the ACOS EMV-A03V1, Configuration A*. Austria Card 2005

[Aust06] *Certification Report BSI-DSZ-CC-0346-2006 for ACOS EMV-A03V1 Configuration A*. Bundesamt für Sicherheit in der Informationstechnik, 2006

[Authen] *Introduction to Code Signing*. MSDN, 2007

[BabDod] Steve Babbage, Matthew Dodd: *The stream cipher MICKEY 2.0*. (im Internet erhältlich) undatiert

[Bach05] Daniel Bachfeld: *Schwachstelle in Verschlüsselung bei Microsofts Office-Paket*. Heise Newsticker vom 21.01.2005

[Bach11] Daniel Bachfeld: *Angriff auf israelischen Zertifikatsherausgeber.* Heise Newsticker vom 20.06.2011

[BaHaJK] Backslash, Hack-Tic, Jansen & Janssen, Keine Panik: *Der kleine Abhörratgeber.* Edition ID-Archiv, Berlin, 1994

[BarKel] Petra Barzin, Stefan Kelm: *Das Policy-Rahmenwerk einer PKI*. (im Internet erhältlich) 2007

[Barret] Paulo S.L.M. Barreto: *The Pairing-Based Crypto Lounge*. (im Internet erhältlich) 2005

[BarRij] Paulo S.L.M. Barreto, Vincent Rijmen: *The WHIRLPOOL Hashing Function*. (im Internet erhältlich) 2003

[Bauer] Friedrich L. Bauer: *Entzifferte Geheimnisse. Methoden und Maximen der Kryptologie*. Springer Verlag, Berlin, 2000

[BBCCGG] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, H. Sibert: *Sosemanuk, a fast software-oriented stream cipher.* (im Internet erhältlich) undatiert

[BBFKSS] Eli Biham, Alex Biryukov, Niels Ferguson, Lars Knudsen, Bruce Schneier, Adi Shamir: *Cryptanalysis of Magenta*. (im Internet erhältlich) 1999

[BeDaAP] Guido Bertoni, Joan Daemen, Gilles Van Assche, Michael Peeters: *RadioGatun, a belt-and-mill hash function*. (im Internet erhältlich) 2006

[BeKiRo] Mihir Bellare, Joe Kilian, Phillip Rogaway: *The Security of the Cipher Block Chaining Message Authentication Code*. (im Internet erhältlich) 1999

[BeNeSc] Albrecht Beutelspacher, Heike Neumann, Thomas Schwarzpaul: *Kryptografie in Theorie und Praxis*. Vieweg, Wiesbaden, 2005

[BePSSS] K. Bentahar, D. Page, J.H. Silverman, M.-J. O. Saarinen, N.P. Smart: *LASH*. (im Internet erhältlich) 2005

[BeRiRT] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, Till Stegers: *Format-Preserving Encryption*. (im Internet erhältlich) 2009

[Bernst] Daniel Bernstein: *Salsa20 specification*. (im Internet erhältlich) undatiert

[BerOst] Omer Berkman, Odelia Moshe Ostrovsky: *The unbearable lightness of PIN cracking*. (im Internet erhältlich) 2006

[BeScWo] Albrecht Beutelspacher, Jörg Schwenk, Klaus-Dieter Wolfenstetter: Moderne Verfahren der Kryptographie. Von RSA zu Zero-Knowledge. Vieweg, Wiesbaden, 2001

[BHJKKL] Jae Hyun Baek, Hyo Sun Hwang, Chung Ryong Jang, Shin Gak Kang, Chul Kim, Eun Jeong Lee, Kyung Seok Lee, Pil Joong Lee, Chae Hoon Lim, Sang Jae Moon, Sung Jun Park, Sang Bae Park, Jong Tae Shin, Sang Gyoo Sim, Dong Ho Won: *The Korean Certicate-based Digital Signature Algorithm*. (im Internet erhältlich) 1998

[BiBiSh] Eli Biham, Alex Biryukov, Adi Shamir: *Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials*. EUROCRYPT, 1999

[BiDuKe] Eli Biham, Orr Dunkelman, Nathan Keller: *A Related-Key Rectangle Attack on the Full KASUMI*. ASIACRYPT, 2005

[BihFur] Eli Biham, Vladimir Furman: *Impossible Differential on 8-Round MARS' Core*. AES Candidate Conference, 2000

[BihSha] Eli Biham, Adi Shamir: *Differential Cryptanalysis of DES-like Cryptosystems*. CRYPTO, 1990

[BirKho] Alex Biryukov, Dmitry Khovratovich: *Related-key Cryptanalysis of the Full AES-192 and AES-256*. (im Internet erhältlich) 2009

[BirKus] Alex Biryukov, Eyal Kushilevitz: *Improved Cryptanalysis of RC5*. EUROCRYPT, 1998.

[BiSh97] Ei Biham, Adi Shamir. *Differential fault analysis of secret key cryptosystems*. CRYPTO, 1997.

[BiShWa] Alex Biryukov, Adi Shamir, David Wagner: *Real Time Cryptanalysis of A5/1 on a PC*. Fast Software Encryption, 2000

[BKLPPR] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. Robshaw, Yanick Seurin, Charlotte Vikkelsoe: *PRESENT: An Ultra-Lightweight Block Cipher.* (im Internet erhältlich) 2010

[BlaFun] Simon Blake-Wilson, Paul Funk: *Tunneled TLS Authentication Protocol*. Internet Draft, 2002

[Bleich] Daniel Bleichenbacher: *Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1*. CRYPTO, 1998

[Blueto] *Bluetooth Specification Version 2.0*. (im Internet erhältlich) 2004

[BoKhRe] Andrey Bogdanov, Dmitry Khovratovich, Christian Rechberger: *Biclique Cryptanalysis of the Full AES*. (im Internet erhältlich) 2011

[BoDeLi] Dan Boneh, Richard A. DeMillo, Richard J. Lipton: *On the Importance of Checking Cryptographic Protocols for Faults*. Lecture Notes in Computer Science, 1997

[BoDoPr] Antoon Bosselaers, Hans Dobbertin, Bart Preneel: *The hash function RIPEMD-160*. (im Internet erhältlich) 2004

[BoGoWa] Nikita Borisov, Ian Goldberg, David Wagner: *Analysis of 802.11 Security or Wired Equivalent Privacy Isn't*. (im Internet erhältlich) 2001

[BoHoHN] K. Boman, G. Horn, P. Howard, V. Niemi: *UMTS Security*. Electronics & Communication Engineering Journal 10/2002

[BonBru] Dan Boneh, David Brumley: *Remote timing attacks are practical*. USENIX Security Symposium, 2003.

[BonFra] Dan Boneh, Matthew Franklin: *Identity-Based Encryption from the Weil Pairing*. (im Internet erhältlich) 2001

[Borche] Detlef Borchert: *Der Diginotar-SSL-Gau und seine Folgen*. Heise Newsticker vom 27.01.2012

[BoVeCZ] Martin Boesgaard, Mette Vesterager, Thomas Christensen, Erik Zenner: *The Stream Cipher Rabbit*. (im Internet erhältlich) undatiert

[BreWel] Kathrin Bremer, Günther Welsch: *Die europäische Signaturrichtlinie in der Praxis*. Datenschutz und Datensicherheit 2/2000

[BrGoWa] Marc Briceno, Ian Goldberg, David Wagner: *A pedagogical implementation of A5/1*. (im Internet erhältlich) 1999

[BroPip] Lawrie Brown, Josef Pieprzyk: *Introducing the new LOKI97 block cipher*. (im Internet erhältlich) 1998

[BSI] *Systembeschreibung ElcroDat 6-2*. Bundesamt für Sicherheit in der Informationstechnik, 2012

[BSI-03110] *Advanced Security Mechanisms for Machine Readable Travel Documents (Technical Guideline TR-03110)*. Bundesamt für Sicherheit in der Informationstechnik, 2012

[BuCaDM] L. Burnett, G. Carter, E. Dawson, W. Millan: *Efficient Methods for Generating MARS-Like S-Boxes*. Fast Software Encryption, 2000

[Buch96] Johannes Buchmann: *Faktorisierung großer Zahlen*. Spektrum der Wissenschaft 9/1996

[Buch03] Johannes Buchmann: *Einführung in die Kryptographie*. Springer Verlag, Berlin, 2003

[BuCoDA] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr., Luke O'Connor, Mohammad Peyravian, David Safford, Nevenko Zunic: *MARS – a candidate cipher for AES*. (im Internet erhältlich) 1999

[BurCon] Bryan Burrough, John Connolly: *Inside Hollywood's Big Wiretap Scandal*. Vanity Fair 6/2006

[CalKir] Jon Callas, Christian Kirsch: *Eintagsfliegen. Kurzzeitzertifikate als Alternative zum Widerruf*. KES 1/2006

[CamWie] K. W. Campbell, M. J. Wiener: *Proof that DES Is Not a Group*. CRYPTO, 1992

[CCC] *Microsoft-Betriebssysteme mit eingebauter Hintertür: Sichere Verschlüsselung mit Windows infrage gestellt*. Presseinformation des Chaos Computer Club vom 03.09.1999

[Chandr] Praphul Chandra: *Bulletproof Wireless Security. GSM, UMTS, 802.11, and Ad Hoc Security*. Newnes, Burlington (USA), 2005

[ChLeBu] Christy Chatmon, Tri van Le, Mike Burmester: *Secure Anonymous RFID Authentication Protocols*. (im Internet erhältlich) undatiert

[CNG] *Cryptography API: Next Generation*. Microsoft Developer Network, 2008

[CoFiSe] Nicolas Courtois, Matthieu Finiasz, Nicolas Sendrier: *How to Achieve a McEliece-based Digital Signature Scheme*. (im Internet erhältlich) 2001

[CoGoPa] Nicolas Courtois, Louis Goubin, Jacques Patarin: *SFLASHv3, a fast asymmetric signature scheme*. (im Internet erhältlich) 2003

[CoFrAv] Henri Cohen, Gerhard Frey, Roberto Avanzi: *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Taylor & Francis, London, 2005

[Coleri] Robert Coleridge: *The Cryptography API, or How to Keep a Secret*. Microsoft Developer Network Technology Group, 1996

[CoLeSt] Scott Contini, Arjen K. Lenstra, Ron Steinfeld: *VSH, an Efficient and Provable Collision-Resistant Hash Function*. (im Internet erhältlich) 2005

[Common] Hans-Joachim Bickenbach, Jürgen Brauckmann, Alfred Giessler, Tamás Horváth, Hans-Joachim Knobloch: *Common PKI Specifications for Interoperable Applications Version 2.0*. T7 & Teletrust, 2009

[CoNoON] Nicolas T. Courtois, Karsten Nohl, Sean O'Neil: *Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards*. (im Internet erhältlich) undatiert

[CouPie] Nicolas Courtois, Josef Pieprzyk: *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*. (im Internet erhältlich) 2002

[Courto] Nicolas T. Courtois: *An Improved Differential Attack on Full GOST*. (im Internet erhältlich) 2012

[CoWo] *Deutsche Geheimdienste sammeln Millionen Daten*. Computerwoche Online vom 27.02.2012

[CRYPTR] *CRYPTREC Report 2002*. (im Internet erhältlich) 2002

[CSA] *CSA – known facts and speculations*. (im Internet erhältlich) 2004

[DaeRij97] Joan Daemen, Vincent Rijmen: *AES Proposal: Rijndael*. (im Internet erhältlich) 1997

[DaeRij01] J*The Design of Rijndael. AES. The Advanced Encryption Standard*. Springer Verlag, Berlin, 2001

[Davies] Joshua Davies: *Implementing SSL/TLS Using Cryptography and PKI*. John Wiley & Sons, New York, 2011

[DelKne] Hans Delfs, Helmut Knebl: *Introduction to Cryptography. Principles and Applications*. Springer Verlag, Berlin, 2007

[DifHel] Whitfield Diffie, Martin E. Hellman: *New Directions in Cryptography.* IEEE Transactions on Information Theory 6/1976

[DifLan] Whitfield Diffie, Susan Landau: *Privacy on the Line: Politics of Wiretapping and Encryption*. MIT Press, Cambridge, 2007

[Dingfe] Marc Dingfelder: *COMP128 – Kollisionsattacke*. (im Internet erhältlich) 2003

[Dobber] Hans Dobbertin: *Digitale Fingerabdrücke*. Datenschutz und Datensicherheit 2/1997

[Donway] Roger Donway: *The Case For Frank Quattrone*. Navigator Juli/August 2004

[DriZeg] Benjamin Drisch, Thomas Zeggel: *Seitenkanalangriffe. Eine kurze Einführung*. (im Internet erhältlich) undatiert

[DuKiLP] Dang Nguyen Duc, Jaemin Park, Hyunrok Lee, Kwangjo Kim: *Enhancing Security of EPCglobal Gen-2 RFID Tag against Traceability and Cloning*. SCIS, 2006

[DuKeSh] Orr Dunkelman, Nathan Keller, Adi Shamir: *A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony.* (im Internet erhältlich) 2010

[Eckert] Eckert, Claudia: *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Oldenbourg, München, 2012

[EffRan] Wolfgang Effing, Wolfgang Rankl: *Handbuch der Chipkarten*. Carl Hanser Verlag, München, 2008

[Eichle] Jens Eichler: *Trau, schau, wem. Social Engineering – Paranoia oder reale Gefahr?* KES 1/2007

[ElGama] Taher ElGamal: *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. IEEE Transactions on Information Theory 4/1985

[EllSch] Carl Ellison, Bruce Schneier: *Ten risks of PKI*. Computer Security Journal 1/2000

[Engele] Urs Paul Engeler: *Was sagen Sie jetzt?* Die Weltwoche 12/97

[Ertel] Wolfgang Ertel: *Angewandte Kryptographie*. Carl Hanser Verlag, München, 2003

[Esslin] Bernhard Esslinger: *Das CrypTool-Skript: Kryptographie, Mathematik und mehr*, 11. Auflage. (im Internet erhältlich) 2013

[eSTREA] Steve Babbage, Christophe De Cannière, Anne Canteaut, Carlos Cid, Henri Gilbert, Thomas Johansson, Matthew Parker, Bart Preneel, Vincent Rijmen, Matthew Robshaw: *The eSTREAM Portfolio*. (im Internet erhältlich) 2008

[EUSIG] *Richtlinie 1999/93/EG des Europäischen Parlaments und des Rates vom 13. Dezember 1999 über gemeinschaftliche Rahmenbedingungen für elektronische Signaturen*. Europäische Union, 1999

[Faber] Eberhard von Faber: *Sicherheitseigenschaften von Standleitungstechnologien*. Bundesamt für Sicherheit in der Informationstechnik, 2007

[FeSc00] Niels Ferguson, Bruce Schneier: *A Cryptographic Evaluation of IPsec*. (im Internet erhältlich) 2000

[FeScKo] Niels Ferguson, Bruce Schneier, Tadayoshi Kohno: *Cryptography Engineering*. John Wiley & Sons, New York, 2010

[FeScWh] Niels Ferguson, Richard Schroeppel, Doug Whiting: *A simple algebraic representation of Rijndael*. Proceedings of Selected Areas in Cryptography, 2001

[Finken] Klaus Finkenzeller: *RFID-Handbuch. Grundlagen und praktische Anwendungen von Transpondern, kontaktlosen Chipkarten und NFC*. Carl Hanser Verlag, München, 2008

[FinTS] *FinTS – Standard für sicheres Online-Banking*. (im Internet erhältlich) 2006

[Fips-46] *Federal Information Processing Standards Publication 46-2. Data Encryption Standard (DES)*. NIST, 1993

[FIPS-140] *Federal Information Processing Standards Publication 140-2. Security Requirements for cryptographic modules*. NIST, 1994

[FIPS-180] *Federal Information Processing Standards Publication 180-2. Secure Hash Standard*. NIST, 2002

[FIPS-186] *Federal Information Processing Standards Publication 186-2. Digital Signature Standard (DSS)*. NIST, 2000

[FIPS-198] *Federal Information Processing Standards Publication 198. The Keyed-Hash Message Authentication Code (HMAC)*. NIST, 2002

[Flanne] Sarah Flannery: *An Investigation of a New Algorithm vs. the RSA*. (im Internet erhältlich) 1998

[FLSWBK] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, Jesse Walker: *The Skein Hash Function Family.* (im Internet erhältlich) 2008

[FlMaSh] Scott Fluhrer, Itsik Mantin, Adi Shamir: *Weaknesses in the Key Scheduling Algorithm of RC4*. (im Internet erhältlich) 2004

[FluMcG] Scott Fluhrer, David McGrew: *Statistical Analysis of the Alleged RC4 Key stream Generator.* (im Internet erhältlich) 2000

[Focus] *Tod im Handgepäck*. Focus 33/2006

[FR] *Aberglaube lohnt sich nicht*. FR-online vom 31.03.2007

[Freibu] Friederike Freiburg: *Tausende private E-Mails stellen Assad-Familie bloß*. Spiegel Online vom 14.03.2012

[FTD] *Abgehörte Gespräche belasten Kapitän Schettino*. FTD Online vom 25.01.2012

[Gagné] Martin Gagné: *Identity-Based Encryption: a Survey.* Cryptobytes 1/2003

[GaJuPa] Simson Garfinkel, Ari Juels, Ravi Pappu: *RFID Privacy: An Overview of Problems and Proposed Solutions*. (im Internet erhältlich) 2005

[GarRos] Simson Garfinkel, Beth Rosenberg: *RFID. Applications, Security, and Privacy.* Addison-Wesley Longman, Amsterdam, 2005

[Gatz] Andreas Gatz: *Camouflage für Krypto-Keys*. KES 6/2001

[GeLeCh] Dianelos Georgoudis, Damian Leroux, Billy Simón Chaves: *The FROG Encryption Algorithm*. (im Internet erhältlich) 1998

[GePiQS] Neil Gershenfeld, Gilles Piret, Jean-Jacques Quisquater, François-Xavier Standaert: *SEA, a Scalable Encryption Algorithm for Small Embedded Applications*. (im Internet erhältlich) 2006

[GGHNPP] Henri Gilbert, Marc Girault, Philippe Hoogvorst, Fabrice Noilhan, Thomas Pornin, Guillaume Poupard, Jacques Stern, Serge Vaudenay: *Decorrelated Fast Cipher: an AES candidate*. (im Internet erhältlich) 1998

[GGMRVW] Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs: *Dismantling MIFARE Classic*. ESORICS 2008

[GilHan] Frank Gillert, Wolf R. Hansen: *RFID für die Optimierung von Geschäftsprozessen: Prozess-Strukturen, IT-Architekturen, RFID-Infrastruktur.* Carl Hanser Verlag, München, 2009

[GilHar] James Gillogly, Larry Harnisch: *Cryptograms from the Crypt*. Cryptologia 4/1996

[Glass] Eric Glass: *The NTLM Authentication Protocol and Security Support Provider.* (im Internet erhältlich) 2006

[Golic] Jovan Dj. Golic: *Linear Statistical Weakness of Alleged RC4 Key stream Generator.* EUROCRYPT, 1997

[Grover] Lov Grover: *A fast quantum mechanical algorithm for database search*. ACM Symposium on the Theory of Computing, 1996

[GuSaBS] J. Guerra-Casanova, C. Sánchez-Ávila, G. Bailador, A. de Santos Sierra: *Authentication in mobile devices through hand gesture recognition*. International Journal of Information Security 2/2012

[Gutman] Peter Gutmann: *Secure Deletion of Data from Magnetic and Solid-State Memory.* (im Internet erhältlich) 1996

[HaAnDa] Feng Hao, Ross Anderson, John Daugman: *Combining cryptography with biometrics effectively.* (im Internet erhältlich) 2005

[HaMeVa] Darrel Hankerson, Alfred J. Menezes, Scott A. Vanstone: *Guide to Elliptic Curve Cryptography.* Springer Verlag, Berlin, 2004

[Heys] Howard M. Heys: *A Tutorial on Linear and Differential Cryptanalysis*. (im Internet erhältlich) undatiert

[HeJoMe] Martin Hell, Thomas Johansson, Willi Meier: *Grain – A Stream Cipher for Constrained Environments*. (im Internet erhältlich) undatiert

[HeScSe] Erwin Hess, Marcus Schafheutle, Pascale Serf: *The Digital Signature Scheme ECGDSA*. (im Internet erhältlich) 2006

[Hoffme] Andreas Hoffmeister: *Verfahren zur sicheren Sicherheitsüberprüfung verschlüsselter Daten in einem Firewall-System*. Patentschrift, Anmeldenummer 10133184, 2003

[Hook] David Hook: *Beginning Cryptography with Java*. Wiley Publishing, Indianapolis, 2005

[HoPiSi] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman: *NTRU: A Ring-Based Public Key Cryptosystem*. (im Internet erhältlich) 1998

[Horste] Patrick Horster: *Kryptologie*. Bibliographisches Institut & Brockhaus, Mannheim, 1985

[HoSHLM] Deukjo Hong, Jaechul Sung, Seokhie Hong, Sangjin Lee, Dukjae Moon: *A New Dedicated 256-bit Hash Function: FORK-256*. (im Internet erhältlich) 2005

[HrFrKS] Juraj Hromkovic, Karin Freiermuth, Lucia Keller, Björn Steffen: *Einführung in die Kryptologie*. Vieweg, Wiesbaden, 2010

[HubJac] Klaus Huber, Michael Jacobson Jr.: *The MAGENTA Block Cipher Algorithm*. (im Internet erhältlich) 1998

[Hughes] J. Hughes: *Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1*. OASIS, 2004

[IoRuSt] John Ioannidis, Aviel D. Rubin, Adam Stubblefield: *Using the Fluhrer, Mantin, and Shamir Attack to Break WEP*. (im Internet erhältlich) 2001

[ISO24727-1] *Identification cards – Integrated circuit card programming interfaces – Part 1: Architecture*. ISO/IEC 24727-1:2007

[ISO24727-2] *Identification cards – Integrated circuit card programming interfaces – Part 2: Generic card interface*. ISO/IEC 24727-2:2008

[ISO24727-3] *Identification cards – Integrated circuit card programming interfaces – Part 3: Application interface*. ISO/IEC 24727-3:2008

[ISO24727-4] *Identification cards – Integrated circuit card programming interfaces – Part 4: Application programming interface (API) administration*. ISO/IEC 24727-4:2008

[ISO29192-2] *Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers*. ISO/IEC 29192-2:2012

[ISO7816-8] *Identification cards – Integrated circuit card – Part 8: Commands for security operations*. ISO/IEC 7816-8:2004

[ISO7816-11] *Identification cards – Integrated circuit cards – Part 11: Personal verification through biometric methods*. ISO/IEC 7816-11:2004

[ISO7816-15] *Identification cards – Integrated circuit cards – Part 15: Cryptographic information application*. ISO/IEC 7816-15:2004

[ISO9796] *Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms*. ISO/IEC 9796-2:2002

[JaJuKo] Joshua Jaffe, Benjamin Jun, Paul Kocher: *Introduction to Differential Power Analysis and Related Attacks*. (im Internet erhältlich) 1998

[Juel05/1] Ari Juels: *RFID Security and Privacy: A Research Survey.* (im Internet erhältlich) 2005

[Juel05/2] Ari Juels: *RFID Privacy: A technical primer for the non-technical reader.* (im Internet erhältlich) 2005

[Juel05/3] Ari Juels: *Minimalist Cryptography for Low-Cost RFID Tags*. (im Internet erhältlich) 2005

[Kahn96] David Kahn: *The Codebreakers*. Scribner, New York, 1996

[Kahn04] David Kahn: *The Reader of Gentlemen's Mail*. Yale University Press, New Haven, 2004

[Kahn12] David Kahn: *Seizing the Enigma*. Houghton Mifflin, Boston, 1991

[KarKie] Christian Karpfinger, Hubert Kiechle: *Kryptologie: Algebraische Methoden und Algorithmen*. Vieweg, Wiesbaden, 2010

[Karkow] Josef Karkowsky: *Wirtschaftsspionage, Deutschland als Selbstbedienungsladen*. KES 4/1997

[KASUMI] *KASUMI specification*. (im Internet erhältlich) 1999

[KaTeDe] Andre Karamanian, Srinivas Tenneti, Francois Dessart: *PKI Uncovered*. Cisco Press, Indianapolis, 2011

[KatLin] Jonathan Katz, Yehuda Lindell: *Introduction to Modern Cryptography*. Chapman & Hall, Boca Raton, 2007

[KeKoSc] Tadayoshi Kohno, John Kelsey, Bruce Schneier: *Preliminary Cryptanalysis of Reduced-Round Serpent*. (im Internet erhältlich) 2000

[KelLuc] John Kelsey, Stefan Lucks: *Collisions and Near-Collisions for Reduced-Round Tiger*. (im Internet erhältlich) 2006

[KelSch] John Kelsey, Bruce Schneier: *MARS Attacks! Preliminary Cryptanalysis of Reduced-Round MARS Variants*. (im Internet erhältlich) 2000

[KeSW97] John Kelsey, Bruce Schneier, David Wagner: *Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA*. ICICS, 1997

[KeSW99] John Kelsey, Bruce Schneier, David Wagner: *Key Schedule Weaknesses in SAFER+*. (im Internet erhältlich) 1999

[KeScWH] John Kelsey, Bruce Schneier, David Wagner, Chris Hall: *Side Channel Cryptanalysis of Product Ciphers*. Journal of Computer Security 2-3/2000

[KhKuMa] Gurgen Khachatrian, Melsik Kuregian, James Massey: *Nomination of SAFER+ as Candidate Algorithm for the Advanced Encryption Standard (AES)*. (im Internet erhältlich) 1998

[KilRog] Joe Kilian, Phillip Rogaway: *How to protect DES against exhaustive key search*. CRYPTO, 1996

[Klein] Andreas Klein: *Attacks on the RC4 stream cipher*. Designs, Codes and Cryptography, 2007

[Klima] Vlastimil Klima: *Tunnels in Hash Functions: MD5 Collisions Within a Minute*. (im Internet erhältlich) 2006

[KLSSWW] John Kelsey, Stefan Lucks, Bruce Schneier, Mike Stay, David Wagner, Doug Whiting: *Improved Cryptanalysis of Rijndael*. Fast Software Encryption 2000

[KMAUOT] Masayuki Kanda, Shiho Moriai, Kazumaro Aoki, Hiroki Ueda, Miyako Ohkubo, Youichi Takashima, Kazuo Ohta, Tsutomu Matsumoto: *E2 – A Candidate Cipher for AES*. (im Internet erhältlich) 1998

[Knudse] Lars Knudsen: *DEAL – A 128-bit Block Cipher.* (im Internet erhältlich) 1998

[Koch95] Paul Kocher: *Cryptanalysis of Diffie-Hellman, RSA, DSS and Other Systems Using Timing Attacks*. (im Internet erhältlich) 1995

[Koch98] Paul Kocher: *A Quick Introduction to Certificate Revocation Trees*. (im Internet erhältlich) 1998

[KolKol] Nora Koldehoff, Stefan Koldehoff: *Aktenzeichen Kunst. Die spektakulärsten Kunstdiebstähle der Welt*. Dumont, Köln, 2004

[KPPPRS] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, A. Rupp, M. Schimmler: *How to Break DES for Euro 8,980*. SHARCS 2006

[Krawcz] Hugo Krawczyk: *HMQV: A High-Performance Secure Diffie-Hellman Protocol*. (im Internet erhältlich) 2005

[Kuhn] Nico Kuhn: *Das Buch der geheimen Verschlüsselungstechnik*. Data Becker, Düsseldorf, 2009

[LaiMas] Xuejia Lai, James L. Massey: *A Proposal for a New Block Encryption Standard*. EUROCRYPT, 1990

[LanHel] Suzan Langford, Martin Hellman: *Differential-Linear Cryptanalysis*. CRYPTO, 1994

[Lardsc] Michael Lardschneider: *Security Awareness – Grundlage aller Sicherheitsinvestitionen*. Datenschutz und Datensicherheit 7/2007

[LCKLHS] Jesang Lee, Donghoon Chang, Hyun Kim, Eunjin Lee, Deukjo Hong, Jaechul Sung, Seokhie Hong, Sangjin Lee: *A New 256-bit Hash Function DHA-256: Enhancing the Security of SHA-256*. (im Internet erhältlich) 2005

[Leiber] Otto Leiberich: *Vom diplomatischen Code zur Falltürfunktion*. Spektrum der Wissenschaft 6/1999

[LenVer] Arjen K. Lenstra, Eric R. Verheul: *An overview of the XTR public key system*. (im Internet erhältlich) 2000

[LiMaMN] Xiaoyi Liu, Cheryl Madson, David McGrew, Andrew Nourse: *Cisco Systems' Simple Certificate Enrollment Protocol (SCEP)*. Internet Draft 2006

[LimHwa] Chae Hoon Lim, Hyo Sun Hwang: *CRYPTON: A New 128-bit Block Cipher – Specification and Analysis (Version 1.0)*. (im Internet erhältlich) 1999

[LinTho] Pete Lindstrom, Frank Thornton: *RFID Security. Protect the Supply Chain*. Syngress Media, Rockland (USA), 2006

[Lipp00] Peter Lipp: *Sicherheit und Kryptographie in Java*. Addison-Wesley, München, 2000

[Lipp06] Manfred Lipp: *VPN – Virtuelle Private Netzwerke. Aufbau und Sicherheit*. Addison-Wesley, München, 2006

[LiRiWa] Moses Liskov, Ronald Rivest, David Wagner: *Tweakable Block Ciphers*. (im Internet erhältlich) 2002

[LMQSV] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, Scott Vanstone: *An efficient Protocol for Authenticated Key Agreement*. (im Internet erhältlich) 1998

[Luck99 Norbert Luckhardt: *Schlüsselknacker aus Leuchtdioden*. c't 11/1999

[LuWeZe] Stefan Lucks, Rüdiger Weis, Erik Zenner: *Sicherheit des GSM-Verschlüsselungsstandards A5*. Datenschutz und Datensicherheit 7/2000

[ManSha] Itsik Mantin, Adi Shamir: *A Practical Attack on Broadcast RC4*. FSE, 2001

[MaPüSc] Tobias Martin, Stefan Pütz, Roland Schmitz: *Security Mechanisms in UMTS*. Datenschutz und Datensicherheit 10/2001

[Mats93] Mitsuru Matsui: *Linear Cryptanalysis Method for DES Cipher.* EUROCRYPT, 1993

[Mats97] Mitsuru Matsui: *New Block Encryption Algorithm MISTY.* (im Internet erhältlich) 1997

[McElie] Robert McEliece: *A public-key cryptosystem based on algebraic coding theory.* JPL DSN Progress Report 4244, 1978

[MenSch] Thomas Menzel, Erich Schweighofer: *Das österreichische Signaturgesetz*. Datenschutz und Datensicherheit 9/1999

[MeOoVa] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone: *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1997

[MeQuVa] Alfred J. Menezes, Minghua Qu, Scott A. Vanstone: *Some new Key agreement protocols providing mutual implicit authentication*. Workshop on Selected Areas of Cryptograpphy, 1995

[Mollin] Richard A. Mollin: *An Introduction to Cryptography.* Chapman & Hall, Boca Raton, 2006

[MorYin] Shiho Moriai, Yiqun Lisa Yin: *Cryptanalysis of Twofish (II)*. (im Internet erhältlich) 2000

[MRTD] *Doc 9303, Machine Readable Travel Documents, Parts 1-3*. ICAO, 2005-2008

[MULTI] *MULTI-S011. An Integrity-Aware Block Encryption Based on Cryptographic Pseudorandom Number Generator.* (im Internet erhältlich) 2001

[NeeSch] Roger M. Needham, Michael D. Schroeder: *Using encryption for authentication in large networks of computers*. Communications of the ACM 12/1978

[NESSIE] *Final report of European project number IST-1999-12324, named New European Schemes for Signatures, Integrity, and Encryption.* (im Internet erhältlich) 2004

[Nichol] Randall K. Nichols: *ICSA Guide to Cryptography.* McGraw-Hill, New York, 1999

[NIST800-90] Elaine Barker, John Kelsey: *Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)*. NIST Special Publication 800-90, 2007

[NohPlö] Karsten Nohl, Henryk Plötz: *Mifare – Little Security, Despite Obscurity.* 24th Chaos Communication Congress, 2007

[NybRue] Kaisa Nyberg, Rainier Rueppel: *A new signature scheme based on the DSA giving message recovery.* ACM Conference on Computer and Communications Security, 1993

[Oechsl] Philippe Oechslin: *Making a Faster Cryptanalytic Time-Memory Trade-Off*. (im Internet erhältlich) 2003

[Opplig] Rolf Oppliger: *Contemporary Cryptography.* Artech House, Norwood (USA), 2005

[P1619] *IEEE P1619: Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices*. IEEE, 2007

[P1363] *IEEE P1363: Standard Specifications For Public Key Cryptography.* IEEE, 2000

[P1363a] *IEEE P1363a: Standard Specifications For Public Key Cryptography: Additional Techniques Version D7*. IEEE, 2004

[P1363.1] *IEEE P1383.1: Standard Specifications For Public Key Cryptography: Lattice-Based Public-Key Cryptography (Draft)*. IEEE, 2007

[P1363.2] *IEEE P1383.2: Standard Specifications For Public Key Cryptography: Password-Based Public Key Cryptography (Draft)*. IEEE, 2007

[P1363.3] *IEEE P1383.3: Standard Specifications For Public Key Cryptography: Identity-Based Public Key Cryptography using Pairings (Draft)*. IEEE, 2007

[PaaPel] Christof Paar, Jan Pelzl: *Understanding Cryptography.* Springer Verlag, Berlin, 2012

[PKCS] *Public-Key Cryptography Standards (PKCS)*. RSA Laboratories, 2007

[PKCS#1] *PKCS#1 v2.1: RSA Cryptography Standard*. RSA Laboratories, 1999

[PKCS#3] *PKCS#3: Diffie-Hellman Key-Agreement Standard – Version 1.4*. RSA Laboratories, 1993

[PKCS#5] *PKCS#5: Password-Based Encryption Standard – Version 1.5*. RSA Laboratories, 1993

[PKCS#6] *PKCS#6: Extended-Certificate Syntax Standard – Version 1.5*. RSA Laboratories, 1993

[PKCS#7] *PKCS#7: Cryptographic Message Syntax Standard – Version 1.5*. RSA Laboratories, 1993

[PKCS#8] *PKCS#8: Private-Key Information Syntax Standard 1.2*. RSA Laboratories, 1993

[PKCS#9] *PKCS#9: Selected Attribute Types – Version 1.1*. RSA Laboratories, 1993

[PKCS#10] *PKCS#10 v1.7: Certification Request Syntax Standard*. RSA Laboratories, 2000

[PKCS#11] *PKCS#11 v2.10: Cryptographic Token Interface Standard*. RSA Laboratories, 1999

[PKCS#12] *PKCS#12 v1.0: Personal Information Exchange Syntax*. RSA Laboratories, 1999

[PKCS#13] Burt Kaliski: *PKCS #13: Elliptic Curve Cryptography Standard*. RSA Laboratories PKCS Workshop, 1998

[PKCS#14] Robert W. Baldwin, James W. Gray: *PKCS #14: Pseudo-Random Number Generation*. RSA Laboratories PKCS Workshop, 1998

[PKCS#15] *PKCS #15 v1.1: Cryptographic Token Information Syntax Standard*. RSA Laboratories, 2000

[PreCan] Bart Preneel, Christophe De Cannière: *Trivium Specifications*. (im Internet erhältlich) undatiert

[PreOor] Bart Preneel, Paul van Oorschot: *On the security of two MAC algorithms*. EUROCRYPT, 1996

[Pröhl] Mark Pröhl: *Kerberos*. dpunkt.verlag, Heidelberg, 2011

[PyTeWe] Erik Tews, Ralf-Philipp Weinmann, Andrei Pyshkin: *Breaking 104 bit WEP in less than 60 seconds*. (im Internet erhältlich) 2007

[RAND] *A Million Random Digits with 100,000 Normal Deviates*. RAND Corporation, 1955

[Reimer] Helmut Reimer: *Digitale Signatur: Update fürs Signaturgesetz*. KES 4/2000

[RFC822] D. Crocker: *Standard for the format of ARPA Internet text messages*. RFC 822, 1982

[RFC1319] B. Kaliski: *The MD2 Message-Digest Algorithm*. RFC 1319, 1992

[RFC1320] R. Rivest: *The MD4 Message-Digest Algorithm*. RFC 1320, 1992

[RFC1321] R. Rivest: *The MD5 Message-Digest Algorithm*. RFC 1321, 1992

[RFC1421] J. Linn: *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*. RFC 1421, 1993

[RFC1422] S. Kent: *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management*. RFC 1422, 1993

[RFC1423] D. Balenson: *Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers*. RFC 1423, 1993

[RFC1424] B. Kaliski: *Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services*. RFC 1424, 1993

[RFC1492] C. Finseth: *An Access Control Protocol, Sometimes Called TACACS*. RFC 1492, 1993

[RFC1521] N. Borenstein, N. Freed: *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*. RFC 1521, 1993

[RFC1661] W. Simpson: *The Point to Point Protocol*. RFC 1661, 1994

[RFC1731] J. Myers: *IMAP4 Authentication Mechanisms*. RFC 1731, 1994

[RFC1734] J. Myers: *POP3 AUTHentication command*. RFC 1734, 1994

[RFC1750] D. Eastlake, S. Crocker, J. Schiller: *Randomness Recommendations for Security*. RFC 1750, 1994

[RFC1968] G. Meyer: *The PPP Encryption Control Protocol (ECP)*. RFC 1968, 1996

[RFC1994] W. Simpson: *PPP Challenge Handshake Authentication Protocol (CHAP)*. RFC 1994, 1996

[RFC2104] H. Krawczyk, M. Bellare, R. Canetti: *HMAC: Keyed-Hashing for Message Authentication*. RFC 2103, 1997

[RFC2144] C. Adams: *The CAST-128 Encryption Algorithm*. RFC 2144, 1997

[RFC2195] J. Klensin, R. Catoe, P. Krumviede: *IMAP/POP AUTHorize Extension for Simple Challenge/Response*. RFC 2195, 1997

[RFC2202] P. Cheng, R. Glenn: *Test Cases for HMAC-MD5 and HMAC-SHA-1*. RFC 2202, 1997

[RFC2268] R. Rivest: *A Description of the RC2(r) Encryption Algorithm*. RFC 2268, 1998

[**RFC2289**] N. Haller, C. Metz, P. Nesser, M. Straw: *A One-Time Password System*. RFC 2289, 1998

[**RFC2403**] C. Madson, R. Glenn: *The Use of HMAC-MD5-96 within ESP and AH*. RFC 2403, 1998

[**RFC2404**] C. Madson, R. Glenn: *The Use of HMAC-SHA-1-96 within ESP and AH*. RFC 2404, 1998

[**RFC2405**] C. Madson, N. Doraswamy: *The ESP DES-CBC Cipher Algorithm With Explicit IV*. RFC 2405, 1998

[**RFC2408**] D. Maughan, M. Schertler, M. Schneider, J. Turner: *Internet Security Association and Key Management Protocol (ISAKMP)*. RFC 2408, 1998

[**RFC2419**] K. Sklower, G. Meyer: *The PPP DES Encryption Protocol, Version 2 (DESE-bis)*. RFC 2419, 1998

[**RFC2420**] H. Kummert: *The PPP Triple-DES Encryption Protocol (3DESE)*. RFC 2420, 1998

[**RFC2433**] G. Zorn, S. Cobb: *Microsoft PPP CHAP Extensions*. RFC 2433, 1998

[**RFC2440**] J. Callas, L. Donnerhacke, H. Finney, R. Thayer: *OpenPGP Message Format*. RFC 2440, 1998

[**RFC2451**] R. Pereira, R. Adams: *The ESP CBC-Mode Cipher Algorithms*. RFC 2451, 1998

[**RFC2535**] D. Eastlake: *Domain Name System Security Extensions*. RFC 2535, 1999

[**RFC2560**] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams: *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP*. RFC 2560, 1999

[**RFC2595**] C. Newman: *Using TLS with IMAP, POP3 and ACAP*. RFC 2595, 1999

[**RFC2612**] C. Adams, J. Gilchrist: *The CAST-256 Encryption Algorithm*. RFC 2612, 1999

[**RFC2617**] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart: *HTTP Authentication: Basic and Digest Access Authentication*. RFC 2617, 1999

[**RFC2637**] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, G. Zorn: *Point-to-Point Tunneling Protocol (PPTP)*. RFC 2637, 1999

[**RFC2661**] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, B. Palter: *Layer Two Tunneling Protocol »L2TP«*. RFC 2661, 1999

[**RFC2712**] A. Medvinsky, M. Hur: *Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)*. RFC 2712, 1999

[**RFC2716**] B. Aboba, D. Simon: *PPP EAP TLS Authentication Protocol*. RFC 2716, 1999

[RFC2743] J. Linn: *Generic Security Service Application Program Interface Version 2, Update 1*. RFC 2743, 2000

[RFC2759] G. Zorn: *Microsoft PPP CHAP Extensions, Version 2*. RFC 2759, 2000

[RFC2808] M. Nystrom: *The SecurID(r) SASL Mechanism*. RFC 2808, 2000

[RFC2831] P. Leach, C. Newman: *Using Digest Authentication as a SASL Mechanism*. RFC 2831, 2000

[RFC2845] P. Vixie, O. Gudmundsson, D. Eastlake, B. Wellington: *Secret Key Transaction Authentication for DNS (TSIG)*. RFC 2845, 2000

[RFC2857] A. Keromytis, N. Provos: *The Use of HMAC-RIPEMD-160-96 within ESP and AH*. RFC 2857, 2000

[RFC2865] C. Rigney, S. Willens, A. Rubens, W. Simpson: *Remote Authentication Dial In User Service (RADIUS)*. RFC 2865, 2000

[RFC2875] H. Prafullchandra: *Diffie-Hellman Proof-of-Possession Algorithms*. RFC 2875, 2000

[RFC2898] B. Kaliski: *PKCS #5: Password-Based Cryptography Specification. Version 2.0*. RFC 2898, 2000

[RFC2945] T. Wu: *The SRP Authentication and Key Exchange System*. RFC 2945, 2000

[RFC2994] H. Ohta, M. Matsui: *A Description of the MISTY1 Encryption Algorithm*. RFC 2994, 2000

[RFC3078] G. Pall, G. Zorn: *Microsoft Point-To-Point Encryption (MPPE) Protocol*. RFC 3078, 2001

[RFC3161] C. Adams, P. Cain, D. Pinkas, R. Zuccherato: *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. RFC 3161, 2001

[RFC3174] D. Eastlake, P. Jones: *US Secure Hash Algorithm 1 (SHA1)*. RFC 3174, 2001

[RFC3217] R. Housley: *Triple-DES and RC2 Key Wrapping*. RFC 3217, 2001

[RFC3394] J. Schaad, R. Housley: *Advanced Encryption Standard (AES) Key Wrap Algorithm*. RFC 3394, 2002

[RFC3414] U. Blumenthal, B. Wijnen: *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*. RFC 3414, 2002

[RFC3588] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko: *Diameter Base Protocol*. RFC 3588, 2003

[RFC3645] S. Kwan, P. Garg, J. Gilroy, L. Esibov, J. Westhead, R. Hall: *Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG)*. RFC 3645, 2003

[**RFC3647**] S. Chokhani, W. Ford, R. Sabett, C. Merrill, S. Wu: *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*. RFC 3647, 2003

[**RFC3658**] O. Gudmundsson: *Delegation Signer (DS) Resource Record (RR)*. RFC 3658, 2003

[**RFC3711**] M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman: *The Secure Real-time Transport Protocol (SRTP)*. RFC 3711, 2004

[**RFC3739**] S. Santesson, M. Nystrom, T. Polk: *Internet X.509 Public Key Infrastructure: Qualified Certificates Profile*. RFC 3739, 2004

[**RFC3748**] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz: *Extensible Authentication Protocol (EAP)*. RFC 3748, 2004

[**RFC3830**] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, K. Norrman: *MIKEY: Multimedia Internet KEYing*. RFC 3830, 2004

[**RFC4033**] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose: *DNS Security Introduction and Requirements*. RFC 4033, 2005

[**RFC4034**] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose: *Resource Records for the DNS Security Extensions*. RFC 4034, 2005

[**RFC4035**] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose: *Protocol Modifications for the DNS Security Extensions*. RFC 4035, 2005

[**RFC4120**] C. Neuman, T. Yu, S. Hartman, K. Raeburn: *The Kerberos Network Authentication Service (V5)*. RFC 4120, 2005

[**RFC4132**] S. Moriai, A. Kato, M. Kanda: *Addition of Camellia Cipher Suites to Transport Layer Security (TLS)*. RFC 4132, 2005

[**RFC4210**] C. Adams, S. Farrell, T. Kause, T. Mononen: *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*. RFC 4210, 2005

[**RFC4211**] J. Schaad: *Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)*. RFC 4211, 2005

[**RFC4251**] T. Ylonen, C. Lonvick: *The Secure Shell (SSH) Protocol Architecture*. RFC 4251, 2006

[**RFC4252**] T. Ylonen, C. Lonvick: *The Secure Shell (SSH) Authentication Protocol*. RFC 4252, 2006

[**RFC4253**] T. Ylonen, C. Lonvick: *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253, 2006

[**RFC4254**] T. Ylonen, C. Lonvick: *The Secure Shell (SSH) Connection Protocol*. RFC 4254, 2006

[RFC4255] J. Schlyter, W. Griffin: *Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints*. RFC 4255, 2006

[RFC4279] P. Eronen, H. Tschofenig: *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. RFC 4279, 2005

[RFC4302] S. Kent: *IP Authentication Header*. RFC 4302, 2005

[RFC4303] S. Kent: *IP Encapsulating Security Payload (ESP)*. RFC 4303, 2005

[RFC4308] P. Hoffman: *Cryptographic Suites for IPsec*. RFC 4308, 2005

[RFC4309] R. Housley: *Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)*. RFC 4309, 2005

[RFC4418] T. Krovetz: Request for Comments: *UMAC: Message Authentication Code using Universal Hashing*. RFC 4418, 2006

[RFC4422] A. Melnikov, K. Zeilenga: *Simple Authentication and Security Layer (SASL)*. RFC 4422, 2006

[RFC4432] B. Harris: *RSA Key Exchange for the Secure Shell (SSH) Transport Layer Protocol*. RFC 4432, 2006

[RFC4492] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, B. Moeller: *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*. RFC 4492, 2006

[RFC4634] D. Eastlake, T. Hansen: *US Secure Hash Algorithms (SHA and HMAC-SHA)*. RFC 4634, 2006

[RFC4650] M. Euchner: *HMAC-Authenticated Diffie-Hellman for Multimedia Internet KEYing (MIKEY)*. RFC 4650, 2006

[RFC4738] D. Ignjatic, L. Dondeti, F. Audet, P. Lin: *MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)*. RFC 4738, 2006

[RFC4869] L. Law, J. Solinas: *Suite B Cryptographic Suites for IPsec*. RFC 4869, 2007

[RFC5008] R. Housley, J. Solinas: *Suite B in Secure/Multipurpose Internet Mail Extensions (S/MIME)*. RFC 5008, 2007

[RFC5055] T. Freeman, R. Housley, A. Malpani, D. Cooper, W. Polk: *Server-Based Certificate Validation Protocol (SCVP)*. RFC 5055, 2007

[RFC5246] T. Dierks, E. Rescorla: *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246, 2008

[RFC5280] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280, 2008

[**RFC5751**] B. Ramsdell, S. Turner: *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification*. RFC 5751, 2010

[**RFC5755**] S. Farrell, R. Housley, S. Turner: *An Internet Attribute Certificate Profile for Authorization*. RFC 5755, 2010

[**RFC5905**] D. Mills, J. Martin, J. Burbank, W. Kasch: *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905, 2010

[**RFC5906**] B. Haberman, D. Mills: *Network Time Protocol Version 4: Autokey Specification*. RFC 5906, 2010

[**RFC5996**] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen: *Internet Key Exchange Protocol Version 2 (IKEv2)*. RFC 5996, 2010

[**RFC6124**] Y. Sheffer, G. Zorn, H. Tschofenig, S. Fluhrer: *An EAP Authentication Method Based on the Encrypted Key Exchange (EKE) Protocol*. RFC 6124, 2011

[**RFC6189**] P. Zimmermann, A. Johnston, J. Callas: *ZRTP: Media Path Key Agreement for Unicast Secure RTP*. RFC 6189, 2011

[**RiRoSY**] Ronald Rivest, Matt Robshaw, Ray Sidney, Yiqun Yin: *The RC6 Block Cipher*. (im Internet erhältlich) 1998

[**Rispen**] Sybe Rispens: *Die elektronische Gesundheitskarte. Whitepaper Sicherheit*. (im Internet erhältlich) 2008

[**Rivest**] Ronald Rivest: *The RC5 Encryption Algorithm*. (im Internet erhältlich) 1994

[**Roos**] Ute Roos: *Entblößt*. iX 7/2001

[**RUB**] *Vom sorglosen Umgang mit Daten. RUB-Forscher decken Sicherheitslücken auf Sensible Daten in Internetverbindungen via Satellit*. Presseinformation der Ruhr-Universität Bochum vom 29.11.2004

[**SaReHo**] M. Salter, E. Rescorla, R. Housley: *Suite B Cipher Suites for TLS*. Internet Draft 2008

[**Schm98**] Klaus Schmeh: *Safer Net – Kryptografie im Internet und Intranet*. dpunkt.verlag, Heidelberg, 1998

[**Schm01**] Klaus Schmeh: *Kryptografie und Public-Key-Infrastrukturen im Internet*. dpunkt.verlag, Heidelberg, 2001

[**Schm02**] Klaus Schmeh: *Die 55 größten Flops der Wirtschaftsgeschichte. Krimis, Krisen, Kuriositäten*. Redline Wirtschaft, Frankfurt, 2002

[**Schm04**] Klaus Schmeh: *Als deutscher Code-Knacker im Zweiten Weltkrieg*. Telepolis vom 23.9.2004

[**Schm06**] Klaus Schmeh: *The East German Encryption Machine T-310 and the Algorithm It Used*. Cryptologia 3/2006

[Schm07/1] Klaus Schmeh: *Codeknacker gegen Codemacher. Die faszinierende Geschichte der Verschlüsselung*. W3L Verlag, Bochum, 2007

[Schm07/2] Klaus Schmeh: *Kryptografie. Verfahren, Protokolle, Infrastrukturen*. dpunkt.verlag, Heidelberg, 2001

[Schm08/1] Klaus Schmeh: *Wettrennen der Codeknacker.* Telepolis vom 09.01.2008

[Schm08/2] Klaus Schmeh: *Versteckte Botschaften. Die faszinierende Geschichte der Steganografie*. dpunkt.verlag, Heidelberg, 2008

[Schm08/3] Klaus Schmeh: *NSA Suite B and its significance for non-USA organisations*. ISSE, 2008

[Schm08/4] Klaus Schmeh: *E-Mail-Krypto-Gateways*. DACH Security, 2008

[Schm09/1] Klaus Schmeh: *Kryptografie – Verfahren, Protokolle, Infrastrukturen.* dpunkt.verlag, Heidelberg, 2009

[Schm09/2] Klaus Schmeh: *Elektronische Ausweisdokumente*. Carl Hanser Verlag, München, 2009

[Schm10] Klaus Schmeh: *Alexander von Kryha and His Encryption Machines*. Cryptologia 4/2010

[Schm11] Klaus Schmeh: *Die Erben der Enigma*. Secunet, Essen, 2011

[Schm12/1] Klaus Schmeh: *Nicht zu knacken*. Carl Hanser Verlag, München, 2012

[Schm12/2] Klaus Schmeh: *Bitschließfach*. iX 12/2011

[Schmid] Jürgen Schmidt: *Zwei weitere Comodo-SSL-Registrare gehackt*. Heise Newsticker vom 31.03.2011

[Schn96] Bruce Schneier: *Applied Cryptography.* John Wiley & Sons, New York, 1996

[Schn05] Bruce Schneier: *SHA-1 is broken*. Crypto-Gram 3/2005

[Schn07/1] Bruce Schneier: *The Blowfish Encryption Algorithm*. http://www.schneier.com/blowfish.html

[Schn07/2] Bruce Schneier: *The Twofish Encryption Algorithm*. http://www.schneier.com/twofish.html

[Schroe] Richard Schroeppel: *The Hasty Pudding Cipher – A Tasty Morsel.* (im Internet erhältlich) 1998

[Schwen] Jörg Schwenk: *Sicherheit und Kryptographie im Internet*. Vieweg, Wiesbaden, 2002

[SciEng] *Break DES in less than a single day.* Presseinformation der Firma SciEngine, undatiert

[Selke] Gisbert Selke: *Kryptographie – Verfahren Ziele Einsatzmöglichkeiten.* O'Reilly, Köln, 2000

[Sender] Ralf Senderek: *Key-Experiments – How PGP Deals With Manipulated Keys*. (im Internet erhältlich) 2000

[ShaDin] Adi Shamir, Itai Dinur: *Cube Attacks on Tweakable Black Box Polynomials*. (im Internet erhältlich) 2008

[Sham79] Adi Shamir: *How to share a secret*. Communications of the ACM 11/1979

[Sham99] Adi Shamir: *Factoring Large Numbers with the TWINKLE Device*. (im Internet erhältlich) 1999

[ShaTro] Adi Shamir, Eran Tromer: *Factoring Large Number with the TWIRL Device*. CRYPTO, 2003

[ShaWoo] Yaniv Shaked and Avishai Wool: *Cracking the Bluetooth PIN*. (im Internet erhältlich) 2005

[Shibbo] *About Shibboleth*. http://shibboleth.internet2.edu

[ShKiHa] Takeshi Shimoyama, Kiyofumi Takeuchiy, Juri Hayakaway: *Correlation Attack to the Block Cipher RC5 and the Simplied Variants of RC6*. (im Internet erhältlich) 2000

[Shor] Peter W. Shor: *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer.* SIAM Journal on Computing, 26/1997

[Short] *Hacker-Gruppe AntiSec stellt abgefangene E-Mails der texanischen Polizei online*. ShortNews vom 02.09.2011

[SigB] *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung. Übersicht über geeignete Algorithmen*. Bundesnetzagentur, 2006

[SigG] *Gesetz über Rahmenbedingungen für elektronische Signaturen und zur Änderung weiterer Vorschriften*. 2001

[SigGCH] *Bundesgesetz vom 19. Dezember 2003 über Zertifizierungsdienste im Bereich der elektronischen Signatur (Bundesgesetz über die elektronische Signatur, ZertES)*. 2005

[SigGÖ] *Bundesgesetz über elektronische Signaturen (Signaturgesetz – SigG)*. 1999

[SigGÖÄ] *Änderung des Signaturgesetzes*. 2008

[SigR] *Einheitliche Spezifizierung der Einsatzbedingungen für Signaturanwendungskomponenten*. Bundesnetzagentur, 2006

[SigV] *Verordnung zur elektronischen Signatur.* 2001

[SigVCH] *Verordnung vom 3. Dezember 2004 über Zertifizierungsdienste im Bereich der elektronischen Signatur (Verordnung über die elektronische Signatur, VZertES)*. 2005

[SigVÖ] *Verordnung des Bundeskanzlers über elektronische Signaturen (Signaturverordnung – SigV)*. 2000

[SigVÖÄ] *Verordnung des Bundeskanzlers über elektronische Signaturen (Signaturverordnung – SigV)*. 2008

[SikStr] Axel Sikora, Tobias Straub: *Cryptography in Embedded Systems*. Embedded World, 2005

[Simmon] Gustavus Simmons: *Subliminal Channels: Past and Present*. European Transactions on Telecommunications 4/1994

[Sin] Susana Sin: *COMP128. An Algorithm for Authentication and Cipher Key Generation in GSM Networks*. (im Internet erhältlich) undatiert

[Skipja] *SKIPJACK and KEA Algorithm Sepcification*. NSA, 1998

[SKWHFW] Bruce Schneier, John Kelsey, David Wagner, Chris Hall, Niels Ferguson, Doug Whiting: *The Twofish Encryption Algorithm*. John Wiley & Sons, New York, 1999

[SNCSSF] *Secure Network Communications and Secure Store & Forward Mechanisms with the SAP R/3 System*. (im Internet erhältlich) 1998

[Sony] *The 128-bit Blockcipher CLEFIA*. Sony, 2007

[Spie97] *Schlüssel für den Staat*. Der Spiegel 12/1997

[Spie06] *»Nudelröllchen« und die große Erpressung*. Der Spiegel 39/2006

[Spitra] Helfried Spitra (Hg.): *Die großen Kriminalfälle*. Campus, Frankfurt, 2004

[Stal99] William Stallings: *SNMPv3: Simple & Secure*. Information Security 1/1999

[Stal06] William Stallings: *Cryptography and Network Security*. Pearson, Upper Saddle River, 2006

[Starts] Alla Startseva: *Online-Wahlen*. GRIN Verlag, Santa Cruz, 2007

[StDeni] Tom St Denis: *Cryptography for Developers*. Syngress, Rockland (USA), 2007

[Stinso] Douglas R. Stinson: *Cryptography Theory and Practice*. Taylor & Francis, Boca Raton, 2005

[STOA] *Development of Surveillance Technology and Risk of Abuse of Economic Information*. Europäische Union, Scientific and Technological Options Assessment (STOA), 1999

[Südd10] *Gouverneur tritt wegen Sex-Skandals zurück*. Süddeutsche.de vom 17.05.2010

[Südd11] *»In meiner Freizeit gebe ich den Regierungschef«*. Süddeutsche.de vom 17.09.2011

[SwSpPr] Jochim Swoboda, Michael Pramateftakis, Stephan Spitz: *Kryptographie und IT-Sicherheit: Grundlagen und Anwendungen*. Vieweg, Wiesbaden, 2008

[Tilb99] Henk van Tilborg: *Fundamentals of Cryptology. A Professional Reference and Interactive Tutorial*. Kluwer Academic Publishers, Norwell, 1999

[Tilb05] Henk van Tilborg: *Encyclopedia of Cryptography and Security.* Springer Verlag, Berlin, 2005

[TraWas] Wade Trappe, Lawrence C. Washington: *Introduction to Cryptography with Coding Theory.* Prentice Hall, Upper Saddle River 2005

[Ulfkot] Udo Ulfkotte: *Marktplatz der Diebe*. C. Bertelsmann, München, 1999

[Veiel] Axel Veiel: *Vorwurf der Zuhälterei*. Frankfurter Rundschau Online vom 28.03.2012

[Verste] Gerhard Versteegen: *Knackpunkte – Zertifizierung nach der ITSEC*. iX 2/1997

[V-PKI] *Allgemeine Informationen zur Verwaltungs-PKI*. (im Internet erhältlich) 2007

[VSITR] *Verwaltungsvorschrift des Innenministeriums zum Geheimschutz von Verschlusssachen beim Einsatz von Informationstechnik (VSITR)*. Bundesamt für Sicherheit in der Informationstechnik, 2004

[WaFeLY] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu: *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD.* (im Internet erhältlich) 2004

[WaFuTP] Dai Watanabe, Soichi Furuya, Kazuo Takaragi, Bart Preneel: *A New Keystream Generator MUGI*. 9th International Workshop on Fast Software Encryption, 2002

[WaYiYu] Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu: *Finding Collisions in the Full SHA-1*. CRYPTO, 2005

[WaYaYa] Xiaoyun Wang, Andrew C Yao, Frances Yao: *Cryptanalysis on SHA-1*. (im Internet erhältlich) 2005

[Wätjen] Dietmar Wätjen: *Kryptographie. Grundlagen, Algorithmen, Protokolle*. Spektrum Akademischer Verlag, Heidelberg, 2003

[Welt] *US-Geheimdienst soll BND-Agenten während des Irak-Krieges belauscht haben*. Welt Online vom 21.02.2006

[Werner] Annette Werner: *Elliptische Kurven in der Kryptographie*. Springer Verlag, Berlin, 2002

[WheNee] David Wheeler, Roger Needham: *TEA, a tiny encryption algorithm*. Fast Software Encryption, 1994

[Witte] Jens Witte: *Polizei nimmt dritten Täter fest*. Spiegel Online vom 11.02.2010

[WMDRM] *Windows Media DRM FAQ*. (im Internet erhältlich) 2005

[Wobst] Reinhard Wobst: *Abenteuer Kryptologie*. Addison-Wesley, München, 2001

[Wu] Hongjun Wu: *The Stream Cipher HC-128*. (im Internet erhältlich) undatiert

[Wu98] Tom Wu: *The Secure Remote Password Protocol*. (im Internet erhältlich) 1998

[X.509] *Information Technology – Open Systems Interconnection – The Directory: Authentication Framework*. 1993

[X9.42] *Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.* American National Standard for Financial Services X9.42–2003

[X9.44] *Public Key Cryptography for the Financial Services Industry: Key Establishment Using Integer Factorization Cryptography.* American National Standard for Financial Services X9.44-2007

[X9.62] *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).* American National Standard for Financial Services X9.62:2005

[X9.63] *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.* American National Standard for Financial Services X9.63-2001

[XAdES] *ETSI TS 101 903 XAdES Version 1.3.2*. ETSI, 2006

[XMLE] *XML Encryption Syntax and Processing.* W3C Recommendation, 2002

[XMLS] *XML Signature Syntax and Processing.* W3C Recommendation, 2002

[Yardle] Herbert Yardley: *The American Black Chamber.* Blue Ribbon, New York, 1931

[zTrace] *Facts Page*. zTrace, 2004

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

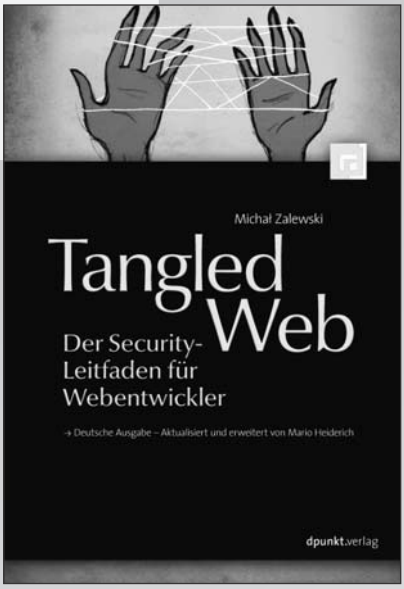
W

X

Y

Z

Ziffern



2013, 392 Seiten, Broschur
€ 36,90 (D)
ISBN 978-3-86490-002-0

Michał Zalewski

Tangled Web – Der Security-Leitfaden für Webentwickler

Deutsche Ausgabe – Aktualisiert und erweitert von Mario Heiderich

Viele der Technologien, die dem Web zugrunde liegen – HTTP, HTML, CSS, JavaScript usw. –, sind unabhängig voneinander entstanden. Sie bilden kein harmonisches Ganzes, sondern eher ein »Tangled Web«, einen Wirrwarr im Web. Was das in puncto Sicherheit für Webentwickler bedeutet, beschreibt Michał Zalewski, einer der international wichtigsten Browser-Security-Experten, anschaulich in diesem Buch. Er zeigt, wie Browser funktionieren, deckt Schwachstellen auf und erklärt, wie Webentwickler ihre Applikationen besser schützen können.

»Ich empfehle, dieses Buch zu lesen, wenn Sie viel – und ich meine in der Tat: viel – darüber wissen wollen, wie vermurkst Webbrowser, Protokolle und verwandte Technologien wirklich sind.«
(Richard Bejtlich, TaoSecurity)

»Pflichtlektüre für jeden Webentwickler«
(packetstormsecurity.org)

dpunkt.verlag

Ringstraße 19 B · 69115 Heidelberg
fon 0 62 21/14 83 40
fax 0 62 21/14 83 99
e-mail hallo@dpunkt.de
http://www.dpunkt.de