Jörg Schwenk

Sicherheit und Kryptographie im Internet

Theorie und Praxis

4., überarbeitete und erweiterte Auflage

Springer Vieweg

1 Kryptographie und das Internet

Übersicht

1.1 Was ist das Internet 2
1.2 Bedrohungen im Internet 5
1.3 Kryptographie 7
1.4 Symmetrische Kryptographie 8
1.5 Public-Key Kryptographie 15
1.6 Kryptographische Protokolle 23
1.7 Angriffe auf Kryptographische Verfahren 26
1.8 Zertifikate 30

In diesem Kapitel soll etwas eigentlich Unmögliches versucht werden, nämlich auf engstem Raum einen Überblick zu den Themen Internet, Kryptographie und Zertifikate zu bieten. Dazu bräuchte man mindestens zwei Bücher, und deshalb werden wir uns hier auf Grundzüge beschränken.

Der Abschnitt über das Internet bietet daher nur einige Fakten zu den Themen IP und TCP (bzw. UDP) und zur Einordnung der verschiedenen Internet-Technologien in das OSI-Schichtenmodell. Dieses Modell wird uns dann als „roter Faden“ durch das Buch begleiten, um die Fülle der Sicherheitstechnologien sinnvoll ordnen zu können: Wir beginnen bei den „versteckten“ Netzwerkschichten und arbeiten uns dann Stufe um Stufe hinauf zu der „sichtbaren“ Anwendungsschicht. Auf die verschiedenen Internet-Protokolle, die auf diesen Ebenen angesiedelt sind, gehen wir jeweils am Anfang der Kapitel näher ein.

Das Thema „Bedrohungen im Internet“ kann nur kurz angerissen werden, da es aufgrund der Vielfalt der mittlerweile bekannten Angriffstechniken ebenfalls ein Buch füllen würde. Wir werden jedoch die wichtigsten Angriffe vorstellen, da sie es ja sind, gegen die uns die Kryptographie schützen soll. Angriffe auf die besonders wichtige Klasse der Webanwendungen werden dann in Kapitel 11 eingeführt.

In den Abschnitten zur Kryptographie sind die wichtigsten Fakten zu den im Internet verwendeten Algorithmen und Protokollen zusammengetragen. Die Algorithmen werden nur als „Black Box“ verwendet, ihr innerer Aufbau wird nicht dargestellt. Wir verweisen an den entsprechenden Stellen auf die Spezialliteratur. Themen, die dort zu knapp behandelt werden, wie z.B. X.509-Zertifikate, behandeln wir wegen ihrer zentralen Bedeutung für das Internet ausführlicher.

J. Schwenk, *Sicherheit und Kryptographie im Internet,*
DOI 10.1007/978-3-658-06544-7_1, © Springer Fachmedien Wiesbaden 2014

	OSI-Modell	TCP/IP-Modell	Beispielprotokolle
7	Anwendungsschicht	Anwendungsschicht	Telnet, FTP, SMTP, HTTP, DNS, IMAP
6	Darstellungsschicht		
5	Sitzungsschicht		
4	Transportschicht	Transportschicht	TCP, UDP
3	Vermittlungsschicht	IP - Schicht	IP
2	Sicherungsschicht	Netzzugangsschicht	Ethernet, Token Ring, PPP, FDDI, IEEE 802.3/802.11
1	Bitübertragungsschicht		

Abb. 1.1 Das TCP/IP-Schichtenmodell und seine Einordnung in das 7-Schichtenmodell der OSI. Bestimmend für das Internet sind die Schichten 3 und 4.

1.1 Was ist das Internet

Unter dem Begriff *Internet* versteht man die Gesamtheit aller öffentlichen Netzwerke, die das Netzwerkprotokoll IP und die beiden Transportprotokolle TCP oder UDP verwenden. Neben TCP/IP gibt es im Internet noch weitere Protokolle auf anderen Ebenen des OSI-Schichtenmodells, und es gibt natürlich auch Netzwerke, die nicht TCP/IP verwenden (und deshalb nicht zum Internet gehören). Die „Intranets“ der Firmen zählen, auch wenn sie TCP/IP verwenden, nicht zum Internet, da sie nicht öffentlich zugänglich sind.

Abbildung 1.1 gibt das *OSI-Schichtenmodell* und seine Anpassung an die Internet-Welt wieder. Von den ursprünglich sieben Schichten (für die die ISO jeweils eigene Protokolle spezifizierte) blieben im Lauf der Entwicklung des Internet nur vier übrig, da es sich in der Praxis als schwierig erwies, die Schichten 5 bis 7 sauber zu trennen. Auch die Schichten 1 und 2 sind in der Praxis oft zusammengefasst (z.B. im Ethernet-Standard). Bei einer typischen Verbindung im Internet zwischen einem Client-PC und einem Server kommen verschiedenste Technologien zum Einsatz. Abbildung 1.2 zeigt ein typisches Beispiel für eine Internet-Verbindung:

- Ein privater Nutzer ist mit seinem PC über das Telefonnetz (DSL) mit seinem Internet Service Provider (ISP) verbunden. Die Software des ISP baut darüber eine PPP-over-Ethernet-Verbindung (Point-to-Point Protocol, vgl. Kapitel 2) auf.
- Der Einwahlrechner stellt eine Verbindung mit einem Router her. Er nutzt dazu ein gängiges Weitverkehrsprotokoll eines Telekommunikationsanbieters, z.B. ATM.
- Der Router steht in unserem Beispiel (das in diesem Punkt nicht sehr typisch ist, denn in der Regel wird eine Verbindung über mehrere Router hinweg aufgebaut) schon im Local Area Network (LAN) des Serverbetreibers, und ist daher über das LAN-Protokoll Ethernet mit dem Server verbunden.

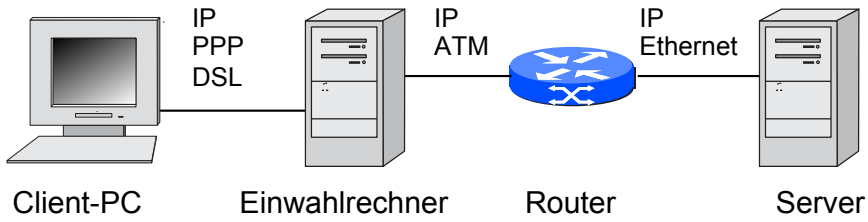

Abb. 1.2 Typisches Beispiel für die bei einer Client-Server-Verbindung verwendeten Technologien im Internet.

Der Erfolg des Internet rührt zu einem beträchtlichen Teil daher, dass der Nutzer über diesen ganzen Zoo von Protokollen nichts wissen muss: Die gesamte Datenkommunikation im Internet läuft über das IP-Protokoll, und es reicht, dieses Protokoll verstanden zu haben, um den Weg von Daten durch das Internet nachvollziehen zu können.

Das *Internet Protocol IP* [Pos81a] ist ein zustandsloses, paketorientiertes Protokoll, d.h. der Absender der Datenpakete erfährt nicht, ob sein Paket angekommen ist. (Ein IP-Paket ist also eher mit einer Postkarte zu vergleichen als mit einem Postpaket.) Auch kann man über die IP-Adresse, die 32 (IPv4) bzw. 128 (IPv6) Bit lang ist, nur einen Rechner als Ganzes adressieren, und nicht einzelne Anwendungen auf diesen Rechnern. Daher wird noch mindestens ein weiteres Protokoll benötigt, und zwar eines der beiden Transportprotokolle TCP oder UDP.

Das *User Datagram Protocol UDP* [Pos80] ist ebenfalls ein zustandsloses Protokoll, d.h. auch hier erhält der Absender keine Information darüber, ob sein Paket angekommen ist oder nicht. Das will er in vielen Fällen auch gar nicht: Insbesondere bei Multimedia-Anwendungen, wie z.B. dem Abruf eines Videoclips von einem Server, möchte der Server nicht mit vielen Rückmeldungen belästigt werden, und wenn einmal ein Paket verloren geht, so wird das Bild beim Empfänger nur kurz gestört.

Das *Transmission Control Protocol TCP* [Pos81b] dagegen bietet den Service „Einschreiben mit Rückschein“: Für jedes gesendete Datenpaket erhält der Absender eine Quittung, ein so genanntes *Acknowledgement*. Bleibt dieses Acknowledgement zu lange aus, so nimmt der Sender an, das Paket sei verloren gegangen, und überträgt es erneut. Dadurch dauert die Übertragung in manchen Fällen deutlich länger als mit UDP, aber jedes Paket kommt an. Damit diese Bestätigungen gesendet werden können, müssen sich Sender und Empfänger zunächst einmal auf die Form dieser Quittungen einigen (genauer auf die Startnummer, mit der sie beginnen, die übertragenen Bytes zu zählen). Sie müssen daher Kontakt miteinander aufnehmen, also eine Verbindung aufbauen. Deshalb wird TCP auch als verbindungsorientiertes Protokoll bezeichnet.

Abbildung 1.3 zeigt, in vereinfachter Form, den Aufbau eines TCP/IP-Pakets. Diesem Paket können während seines Transports durch das Internet weitere Header vorangestellt und wieder entfernt werden. In unserem Beispiel aus Abbildung 1.2 wären dies z.B. für ein IP-Paket, das vom Client zum Server gesendet wird, zunächst ein

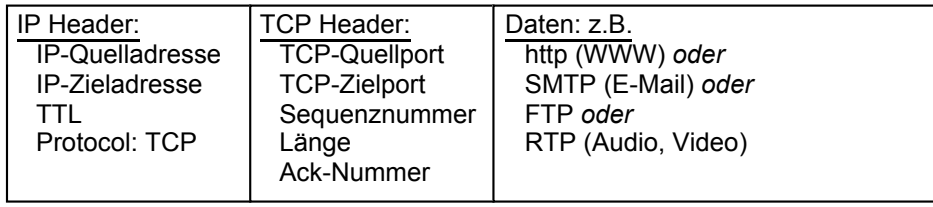

Abb. 1.3 Schematischer Aufbau eines TCP/IP-Pakets.

DSL- und PPP-Header, dann ein ATM-Header, und schließlich ein Ethernet-Header. Uns interessiert hier lediglich das eigentliche TCP/IP-Paket.

Der IP-Header enthält zunächst einmal die wichtigen Adressangaben: Jeder Computer im Internet besitzt eine (für jeden Zeitpunkt) eindeutige IP-Adresse. Spezielle Rechner im Internet, die Router genannt werden, ermitteln anhand der IP-Zieladresse den Weg, den das Paket durch das Internet zum Zielrechner nehmen muss. Diese „Routing-Verfahren“ sind dezentral ausgelegt, d.h. es gibt keine zentrale Instanz, die Wege durch das Internet festlegt. Dies war auch erklärtes Ziel bei der Entwicklung des „Arpanet“, des Vorläufers des Internet.

Die IP-Quelladresse wird vom Zielrechner benötigt, um dem Sender antworten zu können, und sie wird von den Routern benötigt, um ggf. eine Fehlermeldung an den Absender zu schicken. Eine mögliche Fehlermeldung wäre z.B. „TTL expired“, d.h. die Lebensdauer („Time-To-Live“) des IP-Pakets ist abgelaufen, ohne dass das Paket den Zielrechner erreicht hat. Der TTL-Wert gibt dabei die maximale Anzahl von Routern an, die ein IP-Paket durchlaufen darf. Dieser Wert sollte so klein wie möglich gewählt werden, um eine Überlastung des Internet mit ziellos umherwandernden IP-Paketen zu verhindern. Schließlich wird im IP-Header noch angegeben, ob TCP, UDP oder ein anderes Protokoll folgt.

Der TCP-Header dient dazu, dem Zielrechner mitzuteilen, für welches Programm das IP-Paket bestimmt ist. So weiß z.B. ein Server, dass ein IP-Paket mit Zielport 21 für das FTP-Programm, eines mit Zielport 80 aber für das HTTP-Programm (den eigentlichen Webserver) bestimmt ist. Analog definiert das Paar (IP-Quelladresse, TCP-Quellport) eindeutig das Programm auf dem Client-Rechner, an das die Antwort geliefert werden soll.

Bei TCP werden Bytes übertragen, die fortlaufend (beginnend mit der bei der Kontaktaufnahme ausgehandelten Nummer) durchnummeriert werden. Die Sequenznummer gibt dabei die Nummer des ersten in diesem Paket enthaltenen Datenbytes an; zusammen mit der Längenangabe kann der Empfänger so die Nummern aller Bytes im Paket berechnen. Die Ack-Nummer quittiert das letzte empfangene Byte, gibt also die Nummer dieses Bytes an.

Tipp: Mit so genannten Sniffer-Programmen wie z.B. Wireshark[1] kann man den gesamten Datenverkehr an der Netzwerkkarte des PCs mitschneiden. Das Programm stellt diese Daten dann auch sehr übersichtlich dar (IP-Header, TCP-Header, ...).

Es muss hier betont werden, dass alle diese Einträge verändert werden können, da sie nicht gegen Manipulation geschützt sind. Zusammen mit der Tatsache, dass der Weg eines solchen TCP/IP-Pakets durch das Internet nicht vorhersagbar ist, ergibt sich ein großes Potenzial für Angriffe. Einige davon werden wir im nächsten Abschnitt behandeln.

Dies soll als kurze Einführung in die Welt des Internet genügen. Wo es erforderlich ist, werden wir in den einzelnen Kapiteln noch einmal näher auf die verschiedenen Internet-Protokolle eingehen.

1.2 Bedrohungen im Internet

Das Internet entstand zunächst als Forschungsnetz. Die Wissenschaftler waren froh, ein gut funktionierendes Kommunikationsmedium nutzen zu können. Keiner dachte zunächst an das enorme Gefahrenpotenzial, das heute mit den Abermillionen weitgehend anonymer Nutzer gegeben ist.

Die Bedrohungen lassen sich grob in zwei Kategorien aufteilen, in passive und aktive Bedrohungen.

1.2.1 Passive Angriffe

Bei einem *passiven Angriff* liest ein Angreifer übertragene Daten mit, er verändert diese aber nicht und schleust auch keine eigenen Daten in die Übertragung ein. Der Angreifer versucht hier, die Vertraulichkeit der Datenübertragung zu brechen. Verschlüsselung schützt vor passiven Angriffen.

Viele Daten werden im Internet im Klartext übertragen. Das ist nicht anders als bei einer Sprachverbindung über eine Telefonleitung. Es gibt aber einen wichtigen Unterschied zwischen beiden Übertragungsarten: Bei einer Telefonleitung ist es relativ schwierig, an die Daten heranzukommen. Man muss im Keller des Hauses den Telefonanschluß anzapfen, einen gesicherten Verteilerkasten auf der Straße öffnen, oder man muss sich Zugang zur Vermittlungsstelle des Telekommunikationsanbieters verschaffen. All das ist nicht unmöglich, aber für einen Angreifer mit dem hohen Risiko verbunden, ertappt zu werden.

Im Internet dagegen ist das Mithören relativ einfach: Die Router des Internet stehen, im Gegensatz zu den Vermittlungsstellen der Telefonfirmen, auch in öffentlich

[1] `http://www.wireshark.org`

zugänglichen Räumen, z.B. in Universitäten. Jeder, der Zugang zu diesen Räumen hat, kann den IP-Verkehr mitlesen. Durch Manipulation der Routing-Tabellen kann zudem der IP-Verkehr gezielt umgeleitet werden.

Wie man sieht, ist der Zugriff auf IP-Daten relativ leicht und weitgehend risikolos. Die wichtigste Aufgabe der in diesem Buch besprochenen Verfahren ist es daher, diesen passiven Zugriff durch Verschlüsselung zu unterbinden.

1.2.2 Aktive Angriffe

Bei einem *aktiven Angriff* werden übertragene Daten verändert oder völlig neue Daten eingeschleust. Der Angreifer versucht, die Integrität der Datenübertragung zu brechen, oder unter fremder Identität im Internet zu agieren. Ein aktiver Angriff kann auch dazu dienen, eine vorhandene Verschlüsselung zu brechen. Message Authentication Codes und digitale Signaturen sind Hilfsmittel zum Schutz gegen aktive Angriffe.

IP Spoofing. Hier wird die IP-Quelladresse geändert. Dadurch ist es für einen Angreifer möglich, IP-Pakete an ein Opfer zu senden, die nicht zu ihm zurückverfolgt werden können. IP Spoofing ist die Basis für viele weitere Angriffe.

Port Scans dienen dazu, die „offenen Türen" eines an das Internet angeschlossenen Rechners zu ermitteln. Dies geschieht durch automatisierte Tools, die systematisch Nachrichten an bestimmte IP-Adressen und Portnummern senden und die Antworten auswerten.

Port Scans sind nicht illegal, sie dienen aber meist der Vorbereitung von Angriffen. Nahezu jeder Rechner, der längere Zeit im Internet sichtbar ist, ist davon betroffen.

DNS Cache Poisoning. Domain Name Server (DNS) liefern auf Anfrage zu einem „Domain Name" wie z.B. `www.nds.rub.de` die passende IP-Adresse (vgl. Kapitel 10). Sie sind mit einem Telefonbuch vergleichbar, in dem man zu einem Namen die Telefonnummer finden kann. Erst mit der Telefonnummer kann man dann die gewünschte Person anrufen.

DNS Cache Poisoning verändert die Einträge in bestimmten DNS-Servern, oder fälscht DNS-Antworten (z.B. mittels IP Spoofing). Dadurch können Daten, die eigentlich für `www.nds.rub.de` bestimmt waren, an die IP-Adresse des Angreifers umgeleitet werden.

Denial-of-Service (DoS). Diese Angriffe dienen dazu, bestimmte Rechner im Internet (z.B. den WWW-Server der Konkurrenzfirma) gezielt zu überlasten, damit diese ihre Aufgabe nicht mehr erfüllen können.

Ein einfaches Beispiel (TCP Half Open Connection) für einen solchen Angriff nutzt den TCP-Verbindungsaufbau aus, der drei Nachrichten beinhaltet: In der ersten Nachricht sendet der Client eine Nachricht, die seinen Vorschlag für den Startwert der Sequenznummer enthält. In der zweiten Nachricht bestätigt der Server diese Sequenznummer, indem er den um 1 erhöhten Wert als Acknowledgement zurücksendet, und

einen Vorschlag für seine eigene Sequenznummer macht. Wenn der Client dies mit einem Acknowledgement der (um 1 erhöhten) Sequenznummer des Servers beantwortet, ist der TCP-Verbindungsaufbau beendet.

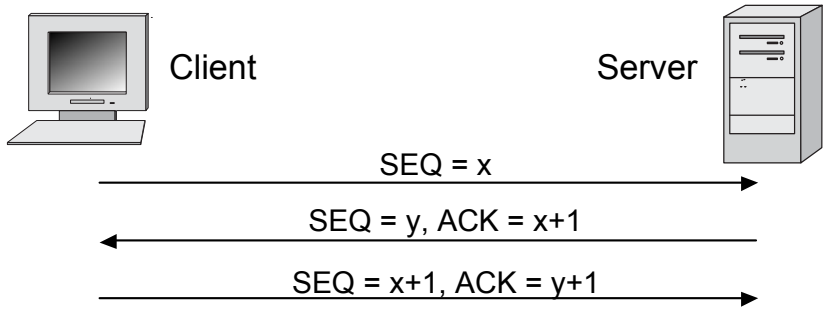

Abb. 1.4 TCP-Verbindungsaufbau. Der Server muss die Zahlen y und x+1 speichern.

Ein Angreifer kann hier die Tatsache ausnutzen, dass der Server sich für jeden Verbindungsaufbau zwei Zahlen merken muss, nämlich die Sequenznummer des Clients und den eigenen Vorschlag für die Sequenznummer.

Er generiert nun (mit Hilfe von IP Spoofing) sehr viele „erste Nachrichten“ mit zufällig gewählten Sequenznummern und verschiedenen gefälschten IP-Quelladressen. Der Server beantwortet alle diese Anfragen, und merkt sich jeweils zwei Zahlen. Irgendwann ist der verfügbare Speicher des Servers voll, und er kann keine weiteren TCP-Verbindungswünsche mehr annehmen. Er ist somit nicht mehr erreichbar.

Die Antworten des Servers an die gefälschten IP-Adressen gehen entweder ins Leere und werden von einem Router gelöscht, oder sie landen bei einem fremden Rechner und werden ignoriert.

1.3 Kryptographie

Die im vorigen Abschnitt angeführten (schon klassischen) Beispiele für Bedrohungen im Internet illustrieren auch die Gründe für die Sicherheitsprobleme: Daten sind öffentlich zugänglich und können von Unbefugten gelesen werden, sie sind nicht gegen Veränderungen geschützt und man weiß eigentlich nie so genau, woher sie kommen.

Möchte man die Gründe für diese Bedrohungen beseitigen, so kommt man nicht umhin, Methoden der Kryptographie einzusetzen. Diese (heute mathematisch fundierte) Wissenschaft stellt Verfahren bereit, um folgende Ziele zu erreichen:

- Vertraulichkeit. Mit Verschlüsselungsalgorithmen ist es möglich, Daten so zu verändern, dass nur noch der Besitzer eines bestimmten kryptographischen Schlüssels sie lesen kann.
- Authentizität. Nachrichten, die mit einem gültigen Message Authentication Code (MAC) oder einer gültigen digitalen Signatur gesichert sind, können nur von einem

Absender stammen, der die erforderlichen Schlüssel besitzt (Authentizität des Teilnehmers). Sie wurden auf ihrem Weg durch das Internet garantiert nicht verändert (Authentizität der Nachricht).

- Anonymität. Das Internet bietet neben der Anonymität für Angreifer auch die Möglichkeit, enorme Datenmengen über einzelne Nutzer zu sammeln. Die meisten normalen Nutzer des Internet sind überrascht zu erfahren, wie viele Informationen ein Webbrowser unbemerkt an den Server überträgt. In berechtigten Fällen muss es daher auch möglich sein, die Anonymität eines Internet-Nutzers zu garantieren (z.B. bei der Kontaktaufnahme mit einer Beratungsstelle für Suchtprobleme). Auch hierfür bietet die Kryptographie Verfahren.

Die Kryptographie kann grob in zwei Bereiche unterteilt werden: Die symmetrische Kryptographie, deren Methoden nur dann funktionieren, wenn die zwei oder mehr Teilnehmer über den gleichen Schlüssel verfügen, und die asymmetrische oder Public-Key Kryptographie, bei der zwischen privaten und öffentlichen Schlüsseln unterschieden wird. Beide Bereiche unterscheiden sich auch grundlegend in den eingesetzten mathematischen Hilfsmitteln.

Als dritter Bereich seien hier kryptographische Protokolle aufgeführt, bei denen eine festgelegte Abfolge von kryptographisch gesicherten Nachrichten ausgetauscht werden muss, um ein bestimmtes Ziel zu erreichen. Je nach Protokoll werden hier Methoden der symmetrischen oder der Public-Key Kryptographie (oder beides) eingesetzt.

Hier noch einige Literaturempfehlungen zum Thema Kryptographie: Beutelspacher „Kryptologie“ [Beu09] als leicht lesbare Einführung in das Gebiet und Paar/Pelz „Understanding Cryptography“ [PP10] als didaktisch gut aufgebaute Vertiefung des Themas. Wer sich noch tiefer in die moderne Kryptographie einarbeiten will, dem seien „Introduction to Modern Cryptography“ von Katz und Lindell [KL14], und das exzellente „Handbook of Applied Cryptography“ von Menezes, van Oorschot und Vanstone [MvOV96]. Speziell auf moderne kryptographische Protokolle ausgerichtet ist das Buch „Moderne Verfahren der Kryptographie“ [BSW06] von Beutelspacher, Schwenk und Wolfenstetter.

1.4 Symmetrische Kryptographie

Charakteristisch für die symmetrische Kryptographie ist, wie schon oben erwähnt, dass alle Beteiligten den gleichen kryptographischen Schlüssel besitzen müssen. Dieser Schlüssel muss vorher auf eine sichere Art und Weise ausgetauscht werden, was in der Praxis ein Problem darstellt. In größeren IT-Systemen ist symmetrische Kryptographie nur dort erfolgreich einsetzbar, wo eine sternförmige Kommunikationsstruktur existiert, z.B. beim Mobilfunk (zwischen dem Betreiber und seinen Kunden) und im Pay-TV (zwischen dem Sender und seinen Abonnenten). Im Internet spielen symmetrische Verfahren auch eine wichtige Rolle, sie müssen aber durch Public-Key-Verfahren ergänzt werden.

1.4.1 Symmetrische Verschlüsselung

Die symmetrische Verschlüsselung ist eine sehr alte Disziplin, die nach alten Quellen schon in Sparta, und später systematisch durch Julius Cäsar, eingesetzt wurde.

Sueton beschreibt dies wie folgt (De Vita Caesarum: Divus Julius LVI, zitiert nach `http://de.wikipedia.org/wiki/Verschiebechiffre`): „*... si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset: quae si qui investigare et persequi velit, quartam elementorum litteram, id est D pro A et perinde reliquas commutet.*“ Übersetzt: „... wenn etwas Geheimes zu überbringen war, schrieb er in Zeichen, das heißt, er ordnete die Buchstaben so, dass kein Wort gelesen werden konnte: Um diese zu lesen, tausche man den vierten Buchstaben, also D, gegen A aus und ebenso mit den restlichen.“

Sie setzt voraus, dass Sender und Empfänger einer Nachricht, und nur diese beiden, ein gemeinsames Geheimnis, den sogenannten Schlüssel, besitzen. Bei der von Cäsar verwendeten Chiffre ist dies die Information, dass die Buchstaben des Alphabets um 4 Stellen verschoben wurden. (Um den Vergleich mit modernen Chiffren zu ermöglichen sei hier erwähnt, dass die Anzahl der möglichen Schlüssel hier 26 ist, und die Schlüssellänge somit weniger als 5 Bit beträgt.)

Man kann die Verschlüsselung einer Nachricht, wie in Abbildung 1.5 dargestellt, als Versenden einer in einen Tresor eingeschlossenen Nachricht visualisieren.

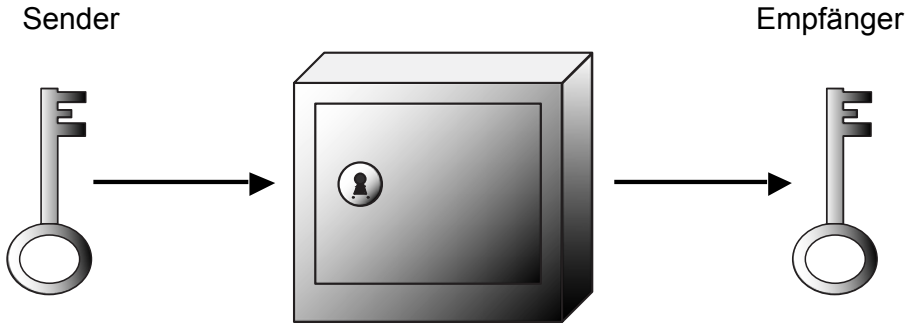

Abb. 1.5 Visualisierung der symmetrischen Verschlüsselung einer Nachricht.

Die Veranschaulichung der symmetrischen Verschlüsselung in Abbildung 1.5 stellt eine Merkhilfe für die Eigenschaften der symmetrischen Verschlüsselung dar: Es ist klar, dass man zum Öffnen des Tresors den gleichen Schlüssel benötigt, mit dem er auch verschlossen wurde. Ein Aspekt soll hier aber noch hervorgehoben werden: Die Sicherheit eines Verschlüsselungsverfahrens steckt nicht, wie die Abbildung vielleicht suggerieren könnte, in den dicken Stahlwänden des Tresors, sondern im Schloss, also im Verschlüsselungsalgorithmus selbst. Die Funktionsweise eines solchen Schlosses zu erläutern würde den Rahmen dieser Einführung sprengen. Hier sei auf die oben angeführte Literatur zum Thema Kryptographie verwiesen. Wir wollen im Folgenden die Notation

$$c = E_k(m)$$

verwenden um zu beschreiben, dass der Chiffretext c aus der Nachricht m durch Verschlüsselung mit dem Schlüssel k entsteht. Die Verschlüsselungsalgorithmen zerfallen in zwei große Gruppen, Blockchiffren und Stromchiffren.

1.4.2 Blockchiffren

Bei einer Blockchiffre wird die zu verschlüsselnde Nachricht vorher in Blöcke fester Länge eingeteilt. Typische Werte sind hier 64 oder 128 Bit (8 oder 16 Byte). Falls die Länge der Nachricht kein Vielfaches der Blocklänge ist, müssen nach einem festgelegten Verfahren noch entsprechend viele Bits aufgefüllt werden („Padding“). Die bekanntesten Blockchiffren sind der „Data Encryption Standard (DES)“ [Des77] und sein Nachfolger, der „Advanced Encryption Standard (AES)“ [AES01].

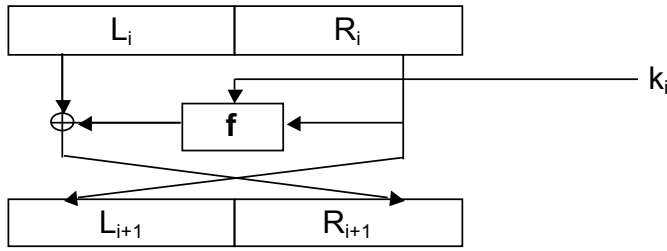

Abb. 1.6 Feistel-Grundstruktur einer DES-Runde.

Data Encryption Standard. Der *Data Encryption Standard (DES)* wurde 1977 vom amerikanischen „National Institute of Standards and Technologies (NIST)“ veröffentlicht. Er verwendet eine Blocklänge von 64 Bit und eine Schlüssellänge von 56+8 Bit, d.h. ein DES-Schlüssel besteht aus 56 zufälligen Bits und 8 „Parity Check Bits“. DES basiert auf der in Abbildung 1.6 dargestellten Feistel-Struktur: In jeder Runde werden rechte und linke Hälfte der Daten vertauscht, und auf die linke Hälfte wird das Ergebnis $f(k_i, R_i)$ bitweise XOR-verknüpft. k_i ist dabei der Rundenschlüssel, und eine „gute“ Funktion f ist ausschlaggebend für die Stärke des Verschlüsselungsalgorithmus.

Die Schlüssellänge von 56 Bit stellt das größte Manko des DES dar: Am 19. Januar 1999 wurde ein DES-Schlüssel auf einem nicht allzu teuren Spezialrechner (auf dem der DES parallel in Hardware implementiert war) durch vollständige Schlüsselsuche in nur 22 Stunden und 15 Minuten berechnet [Fou].

Advanced Encryption Standard. Die Zeit war mehr als reif für einen Nachfolger, und in einem öffentlichen Wettbewerb wurde dieser ermittelt: Für das Design des *Advanced Encryption Standard (AES)* wurde der Entwurf von J. Daemen und V. Rijmen ausgewählt. Der AES erlaubt Block- und Schlüssellängen von 128, 192 und 256 Bit. Er basiert nicht mehr auf einer Feistel-Struktur, sondern auf Matrix-Arithmetik [AES01].

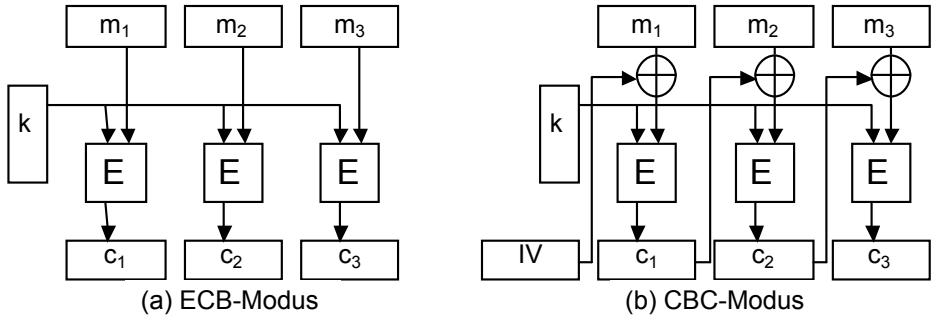

Abb. 1.7 Der (a) Electronic Codebook-Modus (ECB) und der (b) Cipher Block Chaining-Modus von Blockchiffren.

Einsatzmodi von Blockchiffren. Blockchiffren können in verschiedenen Modi eingesetzt werden.

Im *Electronic Codebook Mode (ECM)* wird jeder Block der Nachricht einzeln verschlüsselt. Diese zunächst völlig natürlich erscheinende Lösung birgt aber viele Risiken. So kann ein Angreifer einzelne Chiffretextblöcke entfernen oder umordnen, ohne dass dies bei der Entschlüsselung auffallen würde.

Ein Beispiel eines (tatsächlich realisierten) Angriffs auf ein ECB-verschlüsseltes Passwort sah wie folgt aus: Eine durch Passwort geschützte PC-Anwendung hatte sowohl den Username als auch das Passwort in einer Datei ECB-verschlüsselt abgespeichert. Beim Aufruf des Programms wurde immer der Username entschlüsselt und im entsprechenden Feld der Eingabemaske dargestellt, während das eingegebene Passwort nur mit dem verschlüsselten Wert verglichen wurde. Ein Hacker kam auf die Idee, die Reihenfolge der Blöcke in der verschlüsselten Datei zu vertauschen. Dies hatte zur Folge, dass jetzt das Passwort in der Eingabemaske unverschlüsselt angezeigt wurde, während die Anwendung auf die Eingabe des Username wartete.

Um solche Angriffe zu verhindern, wurden verschiedene Modi eingeführt, um die einzelnen Chiffretextblöcke zu verketten. Neben dem CBC-Modus sind dies der *Cipher Feedback Mode (CFB)*, der *Output Feedback Mode (OFB)* und der *Galois/Counter Mode (GCM)*.

Auf den wichtigen *Cipher Block Chaining Mode (CBC)* soll hier noch näher eingegangen werden. Bei diesem Modus wird jeweils der Chiffretextblock, der gerade verschlüsselt wurde, mit dem nächsten Klartextblock XOR-verknüpft, und dieses Ergebnis wird dann verschlüsselt. Für den ersten Klartextblock wird dazu ein Initialisierungsvektor IV verwendet. Bei Verwendung des CBC-Modus wird durch Veränderung der Reihenfolge der Chiffretextblöcke das Ergebnis der Entschlüsselung ab dem ersten falschen Block verändert. Die Entfernung eines oder mehrerer Chiffretextblöcke zerstört den darauf folgenden Klartextblock, so dass eine solche Manipulation erkannt werden kann. Negativ schlägt zu Buche, dass ein falsch übertragenes Bit gleich zwei Klartextblöcke beeinträchtigt.

Ein CBC-verschlüsselter Klartext ist *verformbar („malleable“)*: Man kann einzelne Bits des Klartextes invertieren, indem man die entspechenden Bits des IV bzw. der vorangehenden Chiffretextblocks invertiert. Dies kann in bestimmten Einsatzszenarien gravierende Konsequenzen haben (vgl. Unterabschnitt 1.7.2).

Diese Angriffe machen noch einmal klar, dass Verschlüsselung nur die Vertraulichkeit der Daten schützt, nicht ihre Integrität. In Szenarien, in denen auch die Integrität der Daten wichtig ist, muss daher der Chiffretext (und nicht der Klartext!) durch einen Message Authentication Code geschützt werden, oder es muss ein Modus gewählt werden, der *authentische Verschlüsselung* garantiert, wie z.B. *Galois/Counter Mode* [GCM07].

1.4.3 Stromchiffren

One-Time Pad. Der Prototyp aller Stromchiffren ist der *One-Time Pad* , der im militärischen und diplomatischen Bereich für die vertrauliche Übertragung von Dokumenten der höchsten Geheimhaltungsstufe eingesetzt wurde. Der One-Time Pad ist der sicherste Algorithmus überhaupt, denn man kann mathematisch beweisen, dass er nicht gebrochen werden kann. Die Idee des One-Time Pad ist einfach: Will man eine Bitfolge verschlüsseln, so wählt man eine zufällige Bitfolge der gleichen Länge aus, und XOR-verknüpft die beiden Folgen; das Ergebnis ist der Chiffretext. Jede Schlüsselfolge darf dabei nur einmal, für einen einzigen Klartext, verwendet werden.

Warum ist dieses Verschlüsselungsverfahren nicht zu brechen? Bei einem „normalen“ Verschlüsselungsverfahren erkennen wir, dass wir es gebrochen haben, weil beim richtigen Schlüssel ein sinnvoller Klartext entschlüsselt wird, beim falschen Schlüssel aber ein sinnloser. Beim One-Time Pad ist es nun aber so, dass es zu jedem gegebenen Chiffretext C und zu jedem sinnvollen Klartext M_i einer bestimmten Länge auch genau einen Schlüssel K_i gibt, der den gegebenen Chiffretext in genau diesen Klartext transformiert: $K_i = C \oplus M_i$. Da wir so zu jedem Klartext einen Schlüssel finden können, ist jeder Klartext gleich wahrscheinlich, und wir können nicht sagen, welches der richtige ist. Wir können den Klartext nur raten, aber das können wir auch, wenn C nicht gegeben ist. Also hilft uns C nicht beider Entschlüsselung, und daher ist der One-Time-Pad unknackbar.

Der One-Time Pad hat allerdings einen gravierenden Nachteil: Das Schlüsselmanagement ist extrem aufwändig. Für jede Nachricht, die Sender und Empfänger austauschen möchten, muss vorher auf sicherem Wege (z.B. in einem Diplomatenkoffer) eine Bitfolge gleicher Länge ausgetauscht werden. Dies macht den Einsatz des One-Time Pad teuer und unpraktisch.

Pseudozufallsfolgen. In der Praxis löst man das Problem des Schlüsselmanagements dadurch, dass man aus einem „normalen“ kryptographischen Schlüssel (und ggf. einem jeweils anderen Startwert) eine „pseudozufällige“ Bitfolge berechnet und diese als Schlüssel für das One-Time Pad benutzt.

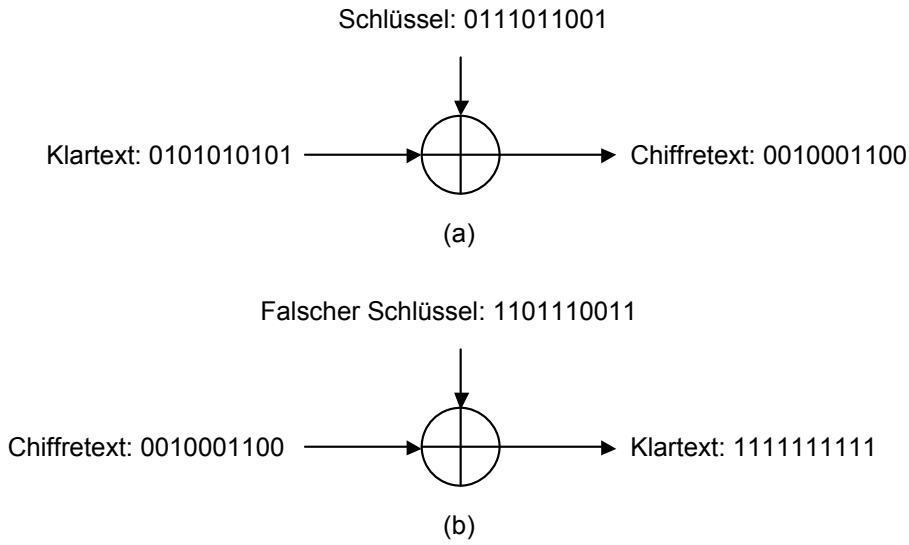

Abb. 1.8 (a) Korrekte Verschlüsselung mit dem One-Time Pad. (b) Inkorrekte Entschlüsselung mit dem falschen Schlüssel.

Die Sicherheit dieser *Stromchiffren* beruht jetzt auf der Qualität dieser Pseudozufallsfolgen: Auch wenn man bereits ein sehr langes Stück dieser Folge kennt, darf es nicht möglich sein, daraus auf den kryptographischen Schlüssel oder auf die nächsten Bits der Folge zu schließen.

Pseudozufallsfolgen kann man mit Hilfe von Blockchiffren, oder auch mit zahlentheoretischen Mitteln erzeugen (beweisbare Qualität). In der Praxis werden oft Schieberegister (die sehr einfach in Hardware zu implementieren sind) mit einer geeigneten Rückkopplungsfunktion eingesetzt.

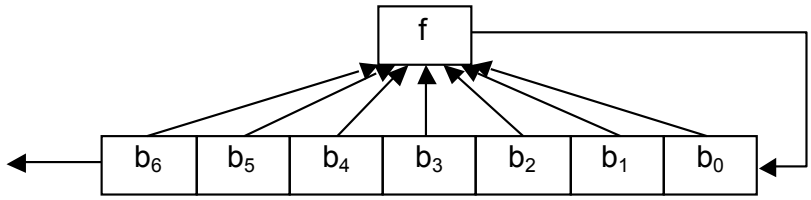

Abb. 1.9 Schematische Darstellung eines Schieberegisters. In jedem Takt werden die Bits $b_0, \ldots, b_6$ um eine Stelle nach Links verschoben. Das Bit b_6 wird so ausgegeben, und ein neues Bit b_0 wird über die Rückkopplungsfunktion f aus den zuletzt gespeicherten Bits berechnet.

Im Bereich der Internetsicherheit, in dem die Funktion zur Erzeugung der Pseudozufallsfolge in Software implementiert werden muss, ist die Stromchiffre RC4 von Ron Rivest (vgl. Abschnitt 3.3.2) wichtig.

1.4.4 Hashfunktionen

Zur symmetrischen Kryptographie gehören auch die Hashfunktionen. Der von einer *Hashfunktion* aus einem beliebig langen Datensatz berechnete Hashwert, der eine feste Länge (in der Praxis 128 oder 160 Bit) besitzt, ist eine kryptographische Prüfsumme, ähnlich den Prüfsummen aus der Codierungstheorie. Da der Hashwert aber gegen *bewusste* Veränderungen des Datensatzes schützen soll (im Gegensatz zu den *zufälligen* Änderungen, die in der Codierungstheorie abgefangen werden), muss er gewisse Eigenschaften besitzen. Zu einem gegebenen Hashwert gibt es zwar theoretisch viele Datensätze, die diesen Hashwert besitzen, aber es muss praktisch unmöglich sein,

- aus dem Hashwert y einen solchen Datensatz x mit $hash(x) = y$ zu berechnen (Einwegeigenschaft),
- zu einem Datensatz x mit $hash(x) = y$ einen zweiten Datensatz x' mit $hash(x') = y$ zu finden (schwache Kollisionsresistenz, *second preimage resistance*), oder
- zwei Datensätze x und x' zu berechnen, die den gleichen Hashwert besitzen (starke Kollisionsresistenz, *collision resistance*).

„Praktisch unmöglich“ bedeutet dabei, dass es weder mit heutigen Computern, noch mit Rechnern der nahen Zukunft möglich sein soll, dies in einem sinnvollen Zeitrahmen zu berechnen. Man kann dies auch in der Sprache der Komplexitätstheorie, eines Teilbereichs der theoretischen Informatik, formalisieren [BDG90a, BDG90b].

Eine gute Hashfunktion zu konstruieren ist eine der schwierigsten Aufgaben der Kryptographie. Daher gibt es nur eine knappe Handvoll guter Hashfunktionen, die in der Praxis aber äußerst wichtig sind: *MD5* [Riv92], *SHA-1* [rJ01] und eine Reihe von Hashfunktionen, die unter dem Schlagwort *SHA-2* zusammengefasst sind [rH06].

Durch den Nachweis eklatanter Schwächen im MD4 durch Hans Dobbertin [Dob98] wurde der Kreis der einsetzbaren Hashfunktionen verkleinert. Auch MD5 weist deutliche Schwächen auf [SSA$^+$09], so dass heute allgemein der Einsatz von SHA-1 (oder noch besser seines Nachfolgers SHA-2) empfohlen wird. Darüber hinaus hat das amerikanische National Institute of Standards and Technology (NIST) am 2.10.2012 den Gewinner des SHA-3-Wettbewerbs bekannt gegeben [oSN].

1.4.5 Message Authentication Codes (MAC) und Pseudozufallsfunktionen

Ein *Message Authentication Code* (MAC) ist eine kryptographische Prüfsumme, in die neben dem Datensatz auch noch der geheime (symmetrische) Schlüssel von Sender und Empfänger einfließt. Ein MAC kann daher im Gegensatz zu einem Hashwert nur von Sender oder Empfänger berechnet und auch nur von diesen beiden verifiziert werden.

Man kann einen MAC auf zwei verschiedene Arten berechnen: Durch iterierte Berechnung einer (symmetrischen) Verschlüsselungsfunktion, oder durch Anwendung einer Hashfunktion auf Schlüssel und Daten in einer festgelegten Reihenfolge.

Als Beispiel für einen MAC der ersten Art sei der CBC-MAC angeführt. Er entsteht dadurch, dass man den gesamten Datensatz mit einer Blockchiffre im CBC-Modus verschlüsselt, und nur den letzten Chiffretextblock abspeichert: Dies ist der MAC.

Die wichtigste Methode, einen MAC mit Hilfe einer Hashfunktion zu bilden („schlüssel-gesteuerte Hashfunktion“), ist die HMAC-Konstruktion [KBC97]. Ihre Sicherheit wurde kryptographisch nachgewiesen. Da alle einfacheren Kombinationen von Daten und Schlüssel kryptographisch angreifbar sind, ist die Verwendung der HMAC-Konstruktion dringend zu empfehlen.

Der Wert $HMAC_H$ (wobei H hier für eine beliebige Hashfunktion steht, also z.B. $HMAC_{MD5}$ oder $HMAC_{SHA1}$) für den Datensatz *text* wird wie folgt berechnet:

$$HMAC_H(text) := H(K \oplus opad || H(K \oplus ipad || text))$$

Dabei wird zunächst der Schlüssel K durch Anfügen von Nullen am Ende auf die Input-Blocklänge der verwendeten Hashfunktion verlängert. Dann wird dieser Wert bitweise mit *ipad* XOR-verknüpft, wobei *ipad* aus hinreichend vielen Bytes 0x36 besteht[2]. An das Ergebnis wird nun der Datensatz *text* angefügt, und das Ganze wird zum ersten Mal gehasht.

Anschließend wird der verlängerte Schlüssel bitweise XOR-verknüpft mit *opad*, einer hinreichend langen Wiederholung des Bytes 0x5C. Das Ergebnis der ersten Anwendung der Hashfunktion wird angefügt, und beides zusammen ein zweites Mal gehasht.

Die HMAC-Konstruktion wird im Bereich der Internetsicherheit sehr häufig eingesetzt, z.B. auch zur Ableitung neuer Schlüssel.

Pseudozufallsfunktionen werden in der Praxis ähnlich zu MACs konstruiert, sie haben aber andere Sicherheitsanforderungen: Von einem MAC fordert man, dass er nicht gefälscht werden kann, wenn der geheime MAC-Schlüssel unbekannt ist („Computational Security“). Von einer Pseudozufallsfunktion PRF verlangt man, dass ihre Ausgabe nicht von einem Zufallswert unterschieden werden kann, wenn der geheime PRF-Schlüssel zufällig gewählt wurde („Decisional Security“). Pseudozufallsfunktionen sind ein wichtiges theoretisches Konstrukt in der modernen Kryptographie; exakte Definitionen findet man z.B. in [KL14].

1.5 Public-Key Kryptographie

Bis zum Jahr 1976 ging man, zumindest in der Öffentlichkeit[3], davon aus, dass Sender und Empfänger immer einen gemeinsamen geheimen Schlüssel benötigen, um vertraulich miteinander kommunizieren zu können. Dann erschien der Artikel „New Directions

[2] Die Notation 0x36 bedeutet hier und im Rest des Buches, dass 0x36 eine Hexadezimalzahl ist, also den Dezimalwert $3 \cdot 16 + 6 = 54$ besitzt.

[3] `http://www.bristol.ac.uk/pace/graduation/honorary-degrees/hondeg08/cocks.html`

in Cryptography“ von W. Diffie und M. Hellman [DH76], und die Welt der Kryptographie hatte sich verändert.

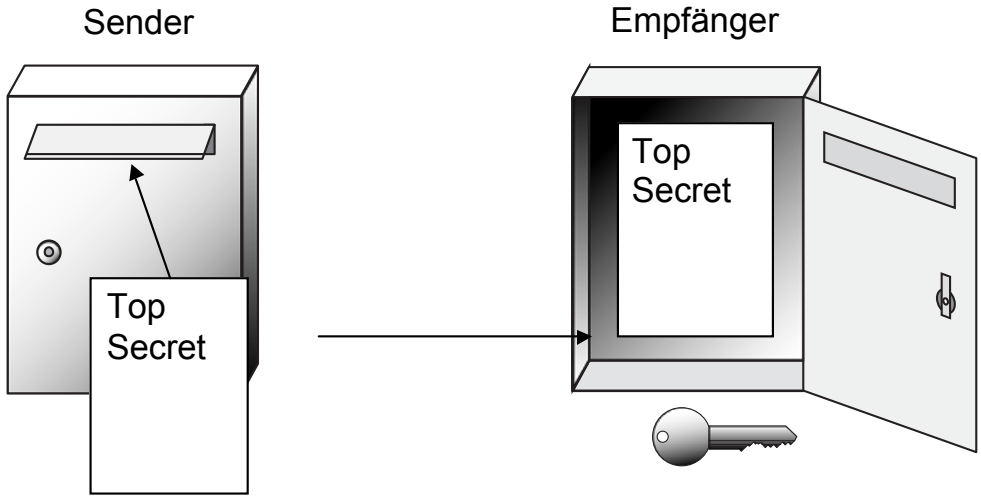

Abb. 1.10 Visualisierung der asymmetrischen (oder Public Key-) Verschlüsselung als Einwurf einer Nachricht in einen Briefkasten.

Dieser Artikel war wegweisend. Er stellte nicht nur das erste *Public-Key-Verfahren* vor (die Diffie-Hellman-Schlüsselvereinbarung, vgl. Unterabschnitt 1.5.3), sondern beschrieb auch schon weitere Möglichkeiten dieser neuen Disziplin, wie die Public-Key-Verschlüsselung und die digitale Signatur. Diese Entdeckung lag damals in der Luft, da auch R. Merkle fast zur gleichen Zeit auf eine ähnliche Idee kam [Mer78].

Die Begriffe „Public-Key-Verfahren“ und „asymmetrische Verfahren“ beschreiben dabei das Gleiche: Kryptographische Verfahren, bei denen die Schlüssel nicht symmetrisch auf Sender und Empfänger verteilt sind, sondern „asymmetrisch“ ein *öffentlicher Schlüssel* („public key“) potenziell vielen Teilnehmern zugänglich ist, während der *private Schlüssel* („private key“) nur einer einzigen Person oder Instanz bekannt ist.

Es war trotzdem nicht einfach, diese neuen Ideen zu akzeptieren, und so wurde 1978 aus dem Versuch heraus, die Spekulationen von Diffie und Hellman zu widerlegen, der wohl berühmteste Public-Key-Algorithmus geboren: Das zunächst als „MIT-Algorithmus“ bezeichnete Verfahren von R. Rivest, A. Shamir und S. Adleman, der RSA-Algorithmus [RSA78] (vgl. Unterabschnitt 1.5.4).

1.5.1 Asymmetrische Verschlüsselung

Die *asymmetrische Verschlüsselung* kann man wie in Abbildung 1.10 dargestellt visualisieren: Ein Sender verschlüsselt eine Nachricht, indem er sie in einen öffentlich zugänglichen Briefkasten (der dem öffentlichen Schlüssel entspricht) wirft. Nur der Eigentümer des Briefkastens, der den dazu passenden privaten Schlüssel besitzt, kann

diesen öffnen und die Nachricht lesen. Wichtige Verschlüsselungsverfahren wie RSA und ElGamal werden in den Abschnitten 1.5.4 und 1.5.5 vorgestellt.

1.5.2 Digitale Signatur

Um eine *digitale Signatur* eines Datensatzes zu erstellen, wird zunächst sein Hashwert gebildet. Dieser Hashwert wird dann mit dem privaten Schlüssel signiert. Überprüft werden kann die digitale Signatur dann von quasi jedem mit Hilfe des öffentlichen Schlüssels. Abbildung 1.11 visualisiert dies als gläsernen Tresor.

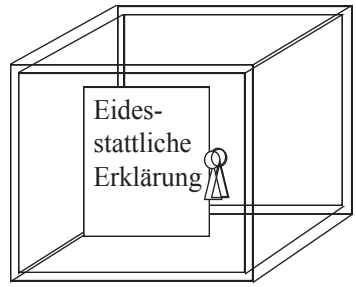

Abb. 1.11 Visualisierung der digitalen Signatur als gläserner Tresor. Nur der Besitzer des privaten Schlüssels kann eine Nachricht hinein legen und damit signieren.

Wichtige Eigenschaften der handschriftlichen Unterschrift besitzt die digitale Signatur ebenfalls. Hierzu gehört z.B. die Tatsache, dass eine Unterschrift nur von einer einzigen Person erzeugt, aber von vielen verifiziert werden kann.

Es gibt aber auch Unterschiede: Während die handschriftliche Unterschrift in der Regel immer ungefähr gleich aussieht, und nur dadurch mit einem Dokument verknüpft wird, dass sie auf dem gleichen Blatt Papier steht wie dieses, hängt der Wert der digitalen Unterschrift direkt vom Wert des zu signierenden Dokuments ab.

Ein wichtiger juristischer Unterschied ist der, dass die handschriftliche Unterschrift direkt von einer Person stammt, die damit eine Willenserklärung abgibt. Die digitale Signatur wird dagegen von einer Maschine berechnet, und kann so im Extremfall auch gegen den Willen einer Person geleistet werden, z.B. durch eine Fehlfunktion der Software oder Schadsoftware wie Viren oder Trojanische Pferde.

1.5.3 Diffie-Hellman Schlüsselvereinbarung

Das erste Problem, das mit Mitteln der Public-Key Kryptographie gelöst wurde, war das Problem der Schlüsselvereinbarung: Mussten wirklich an einer zentralen Stelle zwei Kopien desselben Schlüssels generiert werden, um dann mittels Geheimkurier an Sender und Empfänger verteilt zu werden?

Die Antwort, die Diffie und Hellman in ihrem berühmten Artikel [DH76] auf diese Frage gaben, war ein klares „Nein“. Man muss nur bereit sein, gewisse Ergebnisse zur Komplexität von zahlentheoretischen Problemen zu akzeptieren.

Das zahlentheoretische Problem, das bei der *Diffie-Hellman-Schlüsselvereinbarung* zum Einsatz kommt, ist das Problem des diskreten Logarithmus modulo einer großen Primzahl p: Es ist (durch Einsatz der bekannten effizienten Algorithmen) sehr einfach, aus den Zahlen g und x den Wert $y = g^x \pmod p$ zu berechnen. Dagegen ist es „praktisch unmöglich“, aus den Zahlen y und g einen Wert x mit $y = g^x \pmod p$ zu berechnen.

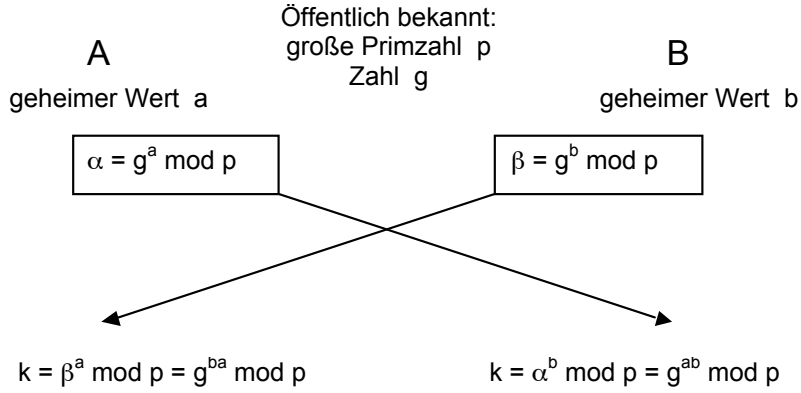

Abb. 1.12 Das Diffie-Hellman-Schlüsselvereinbarungsverfahren zur Berechnung eines gemeinsamen Schlüssels k für die Teilnehmer A und B.

Mit anderen Worten: Die diskrete Exponentialfunktion ist leicht, der diskrete Logarithmus praktisch unmöglich zu berechnen. Funktionen, die die Eigenschaft haben, in der einen Richtung leicht und in der anderen Richtung praktisch nicht berechenbar zu sein, bezeichnet man in der Kryptographie auch als Einwegfunktionen. (Ein anderes Beispiel für Einwegfunktionen sind die Hashfunktionen.)

Anmerkung: Für diese wie für alle anderen nachfolgenden zahlentheoretischen Probleme ist die Berechnung der Umkehrfunktion natürlich nur dann „praktisch unmöglich“, wenn die eingesetzten Zahlen hinreichend groß sind. In der Praxis kommen hier Zahlen zum Einsatz, die binär dargestellt eine Länge zwischen 1024 und 2048 Bit haben.

Die Sicherheit des in Abbildung 1.12 beschriebenen Diffie-Hellman-Verfahrens beruht somit auf der Tatsache, dass die diskrete Exponentialfunktion eine Einwegfunktion ist. Das Verfahren liefert das gewünschte Ergebnis, nämlich einen gemeinsamen Schlüssel für A und B, weil diese spezielle Einwegfunktion noch die zusätzliche Eigenschaft hat, dass die Reihenfolge der beiden Exponenten vertauscht werden darf.

1.5.4 RSA

Der bekannteste Public-Key-Algorithmus beruht auf einem anderen zahlentheoretischen Problem, auf dem Problem der Faktorisierung von großen Zahlen. Auch hier liegt wieder eine Einwegfunktion vor: Es ist einfach, zwei große Zahlen zu multiplizieren, aber praktisch unmöglich, eine solche große Zahl wieder in ihre Primfaktoren zu zerlegen.

Dieses Problem wird aber nicht direkt eingesetzt, sondern in einer von dem Mathematiker Leonhard Euler (1707-1783) stammenden modifizierten Form (Satz von Euler): Ist eine Zahl n das Produkt von zwei Primzahlen p und q, so gilt für jede Zahl x

$$x^{(p-1)(q-1)} \mod n = 1.$$

Der *RSA-Algorithmus* macht sich diesen Satz wie folgt zunutze: In der Schlüsselerzeugungsphase werden zunächst zwei große Primzahlen p und q berechnet (dazu gibt es schnelle Algorithmen), und ihr Produkt $n = pq$ wird gebildet. Danach wird eine Zahl e, die teilerfremd zu $\phi(n) := (p-1)(q-1)$ sein muss, ausgewählt. Zusammen mit dem Produkt n bildet sie den öffentlichen Schlüssel (e, n).

Dann berechnet man mit Hilfe des Euklidischen Algorithmus eine weitere Zahl d, für die

$$e \cdot d \mod (p-1)(q-1) = 1$$

gilt. Dies bedeutet mit anderen Worten, dass $e \cdot d = k(p-1)(q-1) + 1$ ist, für eine uns unbekannte ganzzahlige Konstante k. Die Zahl d ist der private Schlüssel.

RSA-Verschlüsselung. Verschlüsseln kann man eine Nachricht m nun, indem man einfach

$$c = m^e \pmod n$$

berechnet. Die Entschlüsselung ist sehr ähnlich, man muss hier

$$m' = c^d \pmod n$$

berechnen. Von einer erfolgreichen Entschlüsselung können wir natürlich nur dann sprechen, wenn $m = m'$ gilt. Warum diese Gleichheit gilt, ist in Abbildung 1.13 kurz erläutert.

RSA-Signatur. Mit Hilfe des RSA-Verfahrens kann man auch leicht eine digitale Signatur realisieren: Man bildet zunächst den Hashwert $h = hash(m)$ der zu signierenden Nachricht m, und berechnet die Signatur, indem man diesen Hashwert „entschlüsselt“:

$$sig = h^d \pmod n.$$

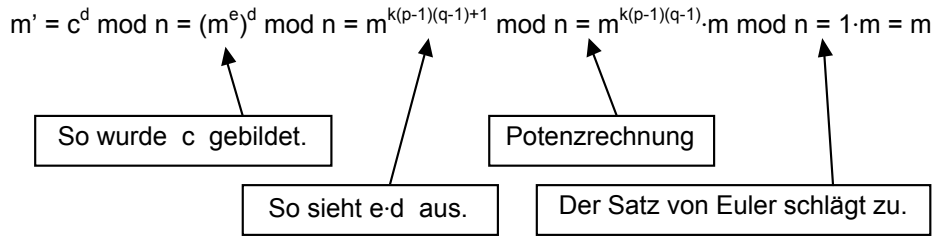

Abb. 1.13 Beweisskizze zur Korrektheit der Entschlüsselung im RSA-Verfahren

Die Signatur wird überprüft, indem zunächst erneut der Hashwert des Dokuments gebildet wird, und dann der Wert *sig* durch Potenzierung mit dem öffentlichen Schlüssel e „entschlüsselt“ wird. Die Signatur war gültig, wenn diese beiden Werte übereinstimmen. Der Beweis der Korrektheit dieser Vorgehensweise erfolgt analog zu Abbildung 1.13, nur mit vertauschten Rollen von e und d.

An dieser Stelle sei direkt darauf hingewiesen, dass bei anderen Signaturverfahren das Signieren keineswegs als „Entschlüsseln mit dem privaten Schlüssel“ erklärt werden kann. Wegen der enormen Popularität von RSA ist dieses Missverständnis aber immer noch weit verbreitet.

1.5.5 ElGamal

Um das nötige Gegengewicht zur Eleganz, Einfachheit und Popularität von RSA zu schaffen, sollen hier die Verschlüsselungs- und Signaturverfahren von *ElGamal* angeführt werden [Gam85], die wie das Diffie-Hellman-Verfahren auf dem Problem des diskreten Logarithmus beruhen.

ElGamal-Verschlüsselung. Das Public-Key-Verschlüsselungsverfahren ist dabei eine direkte Weiterbildung des Diffie-Hellman-Verfahrens: Statt die Werte $y = g^x \pmod p$ nur einmal zu verwenden, werden sie als öffentliche Schlüssel publiziert, und x wird als privater Schlüssel geheim gehalten.

Um eine Nachricht zu verschlüsseln, muss man nur die zweite Hälfte des Diffie-Hellman-Verfahrens nachholen: Aus einer geheimen Zufallszahl r bildet man zum einen $z = g^r \pmod p$, und zum anderen den symmetrischen Schlüssel $k = y^r \pmod p$. Die Nachricht wird dann mit k (symmetrisch) verschlüsselt, und dieser verschlüsselten Nachricht c wird z vorangestellt.

Aus dem Paar (z, c) kann der legitime Empfänger die Nachricht entschlüsseln, indem er zunächst ebenfalls den Schlüssel k als $k = z^x \pmod p$ berechnet, und dann damit c entschlüsselt.

ElGamal-Signatur. Die Idee, mit Hilfe des diskreten Logarithmus ein Signaturverfahren zu bauen, ist dann schon wesentlich trickreicher: Im ElGamal-Signaturverfahren [Gam85] wird eine Nachricht nicht, wie beim RSA-Verfahren, durch Vertauschen der

Reihenfolge von Ver- und Entschlüsselung unterschrieben, sondern durch eine komplexere Operation. Dies hat zur Folge, dass man aus der Signatur der Nachricht nicht auf diese zurück schließen kann. Zur Erzeugung und Verifikation einer digitalen Unterschrift werden der gleiche private Schlüssel x und der gleiche öffentliche Schlüssel $y = g^x \pmod p$ verwendet wie beim ElGamal-Verschlüsselungsverfahren.

Zur Erzeugung einer digitalen Unterschrift für eine Nachricht m geht ein Teilnehmer T dabei wie folgt vor: Zunächst bildet er $h(m)$, dann wählt er eine zu $p-1$ teilerfremde Zufallszahl r und bildet

$$k = g^r \pmod p.$$

Danach berechnet er

$$s = r^{-1}(h(m) - xk) \pmod{p-1}.$$

Die digitale Unterschrift der Nachricht m besteht aus dem Paar (k, s), und es gilt

$$(rs + xk) \pmod{p-1} = h(m).$$

Der Empfänger der signierten Nachricht $(m, (k, s))$ kann die Unterschrift prüfen, indem er die beiden Werte $g^{h(m)} \pmod p$ und $y^k \cdot k^s \pmod p$ bildet und vergleicht, ob diese Zahlen identisch sind. Bei einer gültigen Signatur funktioniert das, weil

$$y^k \cdot k^s \pmod p = g^{xk} \cdot g^{rs} \pmod p = g^{xk+rs} \pmod p = g^{h(m)} \pmod p$$

gilt. (Zum Verständnis des letzten Schrittes muss man sich in Erinnerung rufen, dass eine Reduktion der Basis modulo p im Exponenten einer Reduktion modulo $p-1$ entspricht.)

Der Trick bei diesem Signaturverfahren besteht darin, dass nur derjenige die Zahl s berechnen kann, der den privaten Schlüssel x kennt. Um die Sicherheit dieses privaten Schlüssels auch bei mehrmaliger Verwendung zu garantieren, muss zusätzlich noch eine Zufallszahl r (die jedes Mal verschieden ist) in die Berechnung von s mit einfließen, um zu verhindern, dass aus zwei verschiedenen Signaturwerten s der private Schlüssel berechnet werden kann.

Es gibt eine große Anzahl von Varianten des ElGamal-Signaturverfahrens [HMP95]. Im Gegensatz zum RSA-Verfahren kann man in der Grundform des ElGamal-Verfahrens aus der Signatur die Nachricht (bzw. ihren Hashwert) nicht zurückgewinnen. Varianten, die diese „message recovery"-Eigenschaft besitzen, werden in [NR96] beschrieben.

1.5.6 DSS und DSA

Eine besonders effiziente Variante des ElGamal-Verfahrens, die auf eine Idee von C. Schnorr zurückgeht [Sch89], wurde in den USA als *Digital Signature Algorithm* (DSA) zur Verwendung im *Digital Signature Standard* [GFD09] empfohlen.

Der öffentliche Schlüssel bei diesem Verfahren besteht aus den vier Werten (p, q, g, y):

- p ist eine große Primzahl, $|p| > 1024$.
- q ist eine weitere Primzahl $|q| = 160$, mit der zusätzlichen Eigenschaft, dass sie ein Teiler von $p - 1$ ist.
- g ist ein Element der Ordnung q in Z_p^*, d.h. es gilt $g^q \pmod p = 1$ (und $g^c \pmod p \neq 1$ für alle $c \leq q$).
- Der private Schlüssel x ist eine zufällig gewählte Zahl aus der Menge $\{1, \ldots, q-1\}$, und $y = g^x \pmod p$ ist der öffentliche Schlüssel.

Beim DSA spielen also zwei Primzahlen eine Rolle: Eine große Primzahl q der Länge 160 Bit, und eine noch wesentlich größere Primzahl p. Die *Verifikation* der digitalen Signatur erfolgt auch in der multiplikativen Gruppe $\mathbb{Z}_p^*$, die $p-1$ Elemente enthält. Das ausgewählte Element g erzeugt darin eine Untergruppe, die genau q Elemente enthält. Daher können alle Berechnungen zu *Erzeugung* einer digitalen Signatur ausschließlich modulo q durchgeführt werden, und dies ist letztendlich der Grund für die gegenüber RSA oder ElGamal deutlich kleineren Signaturen. Die Länge von q ist nicht zufällig gewählt, sondern sie entspricht genau der Länge der Ausgabe eines anderen FIPS-Standards, nämlich der Hashfunktion SHA-1.

Die Erzeugung der Signatur ist ähnlich zum ElGamal-Signaturverfahren, nur werden alle Berechnungen modulo q durchgeführt, und eine Subtraktion wird durch eine Addition ersetzt.

1. Wähle eine geheime, zufällige ganze Zahl r, $0 \leq r < q$.
2. Berechne $k = (g^r \pmod p) \pmod q$.
3. Berechne $r^{-1} \pmod q$.
4. Berechne $s = r^{-1}(h(m) + x \cdot k) \pmod q$.
5. Die Signatur der Nachricht m ist das Paar (k, s).

Die Überprüfung der Signatur ist kniffliger, da hier genau auf die Reihenfolge der Reduktionen modulo p oder q geachtet werden muss.

1. Verwende den öffentlichen Schlüssel (p, q, g, y).
2. Überprüfe dass $0 \leq k < q$ und $0 \leq s < q$; wenn nicht, ist die Signatur ungültig.
3. Berechne $w = s^{-1} \pmod q$ und $h(m)$.
4. Berechne $u_1 = w \cdot h(m) \pmod q$ und $u_2 = k \cdot w \pmod q$.
5. Berechne $v = (g^{u_1} y^{u_2} \pmod p) \pmod q$.
6. Die Signatur ist genau dann gültig, wenn $v = k$.

Für alle hier aufgeführten Berechnungen gibt es effiziente Algorithmen. Diese sind z.B. in [MvOV96] angegeben. Der große Vorteil des DSA liegt in der Kürze der erzeugten Signaturen: Das Paar (k, s) ist nur 320 Bit groß, bei einem beliebig vergrößerbaren „Sicherheitsparameter“ p, der nur in den öffentlichen Schlüssel einfließt. Signaturen vergleichbarer Sicherheit wären bei RSA 1024 Bit, und bei ElGamal 2048 Bit lang.

1.5.7 Hybride Verschlüsselung von Nachrichten

Nachdem die wichtigsten Begriffe aus der Kryptologie eingeführt sind, können wir das allgemeine Prinzip beschreiben, nach dem längere Datensätze verschlüsselt werden.

In Abbildung 1.14 ist dieses Prinzip der *hybriden Verschlüsselung* illustriert. Der Sender einer Nachricht wählt zufällig einen symmetrischen Schlüssel und verschlüsselt mit ihm (und einem passenden Algorithmus) den Nachrichtentext (Verschließen des Tresors). Anschließend verschlüsselt er diesen symmetrischen Schlüssel mit dem öffentlichen Schlüssel des Empfängers (Einwerfen des Schlüssels in den Briefkasten) und fügt dieses Kryptogramm an die Nachricht an. Soll eine Nachricht an mehrere Empfänger gesendet werden, so muss der symmetrische Schlüssel jeweils separat mit dem öffentlichen Schlüssel jedes Empfängers verschlüsselt und dieses Kryptogramm ebenfalls mit angefügt werden. (Bildlich gesprochen würden dann mehrere Briefkästen mit dem Tresor zusammen versandt werden.)

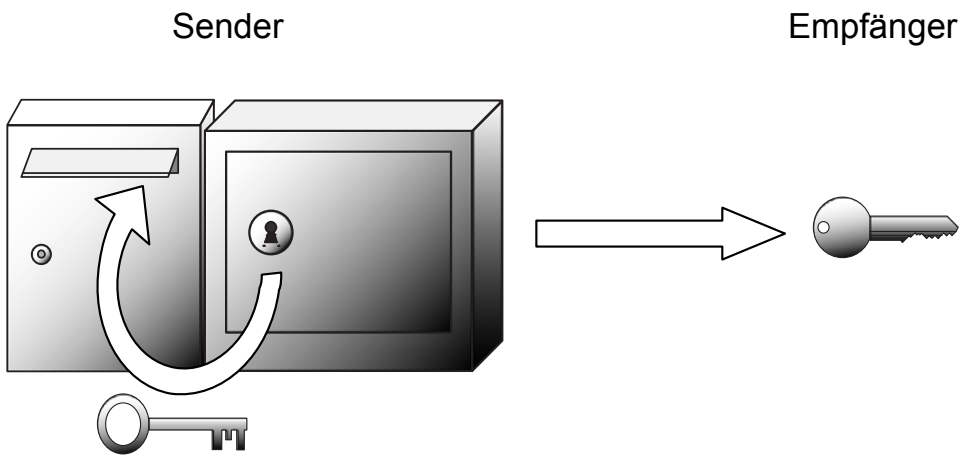

Abb. 1.14 Hybride Verschlüsselung einer Nachricht.

Dieses Verfahren wendet man an, weil symmetrische Verschlüsselungsverfahren wesentlich schneller und effizienter sind als asymmetrische. Man spart also eine Menge Zeit, weil nur der (relativ kurze) symmetrische Schlüssel asymmetrisch verschlüsselt werden muss.

1.6 Kryptographische Protokolle

In diesem Abschnitt wollen wir näher auf kryptographische Verfahren eingehen, die eine starke interaktive Komponente besitzen. Diese Verfahren werden auch *kryptographische Protokolle* genannt [BSW06], weil es hier wie bei einem diplomatischen Protokoll auf die korrekte Einhaltung von Regeln ankommt, damit sie funktionieren.

In der Praxis werden kryptographische Protokolle meist zur Authentifizierung von Teilnehmern eingesetzt (z.B. Mobilfunk, PPP-Einwahl beim Internet Service Provider). Wir werden später noch näher auf komplexere Protokolle wie SSL oder IPSec IKE eingehen. Hier sollen nur die Protokolle zusammengefasst werden, die in verschiedenen Einsatzgebieten im Internet verwendet werden.

1.6.1 Username/Password

Das einfachste Protokoll, das schon lange aus der UNIX-Welt bekannt ist, ist das Username/Password-Protokoll. Nach dem Starten eines Computers, oder zur Authentifizierung an einem geschützten Webserver, wird der Nutzer aufgefordert, seinen Namen („Username“) und sein Passwort („Password“) einzugeben. Der Computer vergleicht dann die angegebenen Werte mit den in einer speziellen Datei oder Datenbank gespeicherten Paaren, und gibt bei Übereinstimmung den Zugriff frei.

Das Passwort sollte dabei aus Sicherheitsgründen nicht im Klartext abgespeichert sein, sondern in verschlüsselter oder gehashter Form, und der Computer muss eine zusätzliche Operation auf dem eingegebenen Passwort durchführen, bevor er die Werte vergleichen kann. Eine Kompromittierung der Passwortdatei führt damit nicht automatisch zum Verlust der Sicherheit für das System: Der Dieb kann ja die Verschlüsselung oder Hashwertbildung nicht rückgängig machen.

Bei schwachen Passwörtern (z.B. Vornamen, kurze Zeichenkombinationen) ist es aber möglich, einen so genannten „Wörterbuchangriff“ durchzuführen: Man stellt sich ein Wörterbuch der gängigsten Passwörter zusammen, und hasht alle Einträge dieser Liste. Nachdem die so erhaltenen Paare $(word, H(word))$ nach dem Hashwert $H(word)$ sortiert wurden, kann man bequem jeden in der gestohlenen Passwortdatei gefundenen Hashwert H in der Liste suchen, und bei einem Treffer $H = H(word)$ das Passwort $word$ ermitteln. Wir werden im Abschnitt zur PPTP-Sicherheit noch näher auf diese Angriffe eingehen.

Die größte Schwäche des Username/Password-Verfahrens im Internet ist es allerdings, dass das Passwort *eingegeben* und *übertragen* werden muss. Geschieht die *Übertragung* unverschlüsselt, so kann das Passwort im Internet aufgezeichnet werden.

Die heute wichtigste Klasse von Angriffen auf Passwörter sind aber Angriffe auf die *Eingabe*:

- *Phishing*-Angriffe („Password Fishing“): Das Passwort-Eingabeformular einer Webseite wird vom Angreifer so täuschend echt nachgebildet, dass der Nutzer garnicht merkt, dass es sein Passwort direkt an den Server des Angreifers sendet.
- *Cross-Site Scripting (XSS)*: Ein vom Angreifer in die Login-Seite eingeschleustes Javascript-Programm liest das Passwort im Eingabefeld aus und sendet es an den Angreifer (Unterabschnitt 11.2.1).

Die Aufklärung von Nutzers über die Risiken von Passwörtern darf daher nicht allein in der Empfehlung münden, „sichere“ Passwörter zu benutzen: Diese schützen lediglich

von Wörterbuchangriffen. Phishing-Angriffe sind schwer zu erkennen, und einem XSS-Angriff (oder sogar einem Angriff mittels Schadsoftware) stehen Nutzer absolut hilflos gegenüber. Es ist daher an der Zeit, sichere Alternativen zu Nutzername/Passwort zu entwickeln und einzuführen (Abschnitt 11.4).

1.6.2 One Time Password

Um Gefahren, die mit dem Diebstahl eines Passworts verbunden sind, zu entgehen, kann man festlegen, dass jedes Passwort nur genau einmal verwendet werden darf. Diese „One Time Password“ (OTP)-Verfahren werden z.B. beim Onlinebanking eingesetzt, wo für jeden Buchungsvorgang eine nur einmalig verwendbare Transaktionsnummer (TAN) eingegeben werden muss, oder z.B. auch zum Aufladen von Handys mit vorausbezahlten Karten. Automatisiert wurden diese Verfahren z.B. in der Produktlinie SecureID der Firma RSA Inc.

1.6.3 Challenge-and-Response

Die nächste Verbesserung, und die nächste Stufe der Interaktivität, stellen Frage- und Antwortprotokolle dar, die als *Challenge-and-Response-Protokolle* bezeichnet werden. Möchte sich ein Client (Handy, Nutzer) gegenüber einem Server (Netzbetreiber) authentisieren, so sendet der Server eine Zufallszahl, die dann vom Client mit einem (symmetrischen) Schlüssel verschlüsselt werden muss, den nur Client und Server kennen. Der Server überprüft dann diesen Wert und gibt ggf. den Zugriff frei.

Auch von diesem Protokoll gibt es natürlich viele Varianten: Der Server kann eine verschlüsselte Zufallszahl senden, die der Client entschlüsseln muss, oder es kann eine HMAC-Konstruktion oder Public-Key Kryptographie eingesetzt werden.

Challenge-and-Response-Protokolle sind weit verbreiten, von der Identifizierung des Handys in Mobilfunknetzen bis hin zum CHAP-Protokoll bei der Einwahl ins Internet.

1.6.4 Certificate/Verify

Stehen die Mittel der Public-Key Kryptographie zur Verfügung, so kann man verschiedene Varianten von Challenge-and-Response anwenden, um die Authentizität eines Teilnehmers zu überprüfen.

Der öffentliche Schlüssel wird meist in Form eines Zertifikats (s.u.) mit dem Namen (der Identität) eines Teilnehmers verknüpft und steht in dieser Form der Öffentlichkeit zur Verfügung. Beim Certificate/Verify-Protokoll kann man z.B. verlangen, dass der Teilnehmer eine Zufallszahl signieren soll, oder man kann ihm eine mit dem öffentlichen Schlüssel verschlüsselte Zufallszahl zusenden, die er dann entschlüsseln muss. Die Antworten können dann jeweils mit dem öffentlichen Schlüssel verifiziert werden.

Certificate/Verify taucht oft als Baustein in komplexeren Protokollen (z.B. SSL oder IPSec IKE) auf.

1.7 Angriffe auf Kryptographische Verfahren

In den einzelnen Kapiteln dieses Buches werden wir nicht nur die kryptographischen Sicherheitsstandards des Internet vorstellen, sondern auch (sofern bekannt) Angriffe auf diese.

1.7.1 Angriffe auf Verschlüsselung

Man klassifiziert Angriffe auf Verschlüsselungsverfahren nach der Menge und Qualität der Informationen, die einem Angreifer zur Verfügung stehen.

Bei einem *Ciphertext Only*-Angriff steht nur der Chiffretext selbst zur Verfügung. Dies ist die klassische Situation, wenn eine verschlüsselte Nchricht abgefangen wurde.

Sehr häufig sind aber auch *Known Plaintext*-Angriffe möglich, etwa wenn der Angreifer selbst verschlüsseln kann (Public Key-Verschlüsselung), wenn er eine Nachricht von einem Netzwerkgerät verschlüsselt bekommt (z.B. Senden einer bekannten Nachricht an einen Host im WLAN, der WLAN Access Point verschlüsselt hier die Nachricht), oder wenn er Teile der Nachricht kennt (z.B. die IP-Adressen eines IP-Pakets).

Um einen *Chosen Ciphertext*-Angriff durchführen zu können, benötigt der Angreifer ein Entschlüsselungsorakel, also eine Instanz, die auf Anfrage beliebig viele, vom Angreifer gewählte Chiffretexte entschlüsselt und den Klartext zurückgibt. Dieses Szenario klingt ziemlich exotisch, und wenn man das Wort „Adaptive“ noch hinzufügt, bekommt man die stärkste kryptographische Angriffsklasse. Trotzdem ist dieses Szenario realistisch: Daniel Bleichebacher hat 1998 einen solchen Angriff auf SSL vorgestellt (siehe Kapitel 7)!

1.7.2 Padding Oracle-Angriffe auf den CBC-Modus von Blockchiffren

Auf einen Angriff sollhier wegen seiner praktischen Bedeutung gesondert eingegangen werden: Auf *Padding Oracle*-Angriffe auf Blockchiffren im CBC-Modus (Unterabschnitt 1.4.2). Der Name rühr daher, dass dieser Angriff nur funktioniert, wenn das Paddingverfahren aus RFC 2040 [BR96] eingesetzt wird.

Padding wird bei Blockchiffren immer dann eingesetzt, wenn die Länge der Nachricht kein Vielfaches der Blocklänge der Chiffre ist. So wird z.B. vor Verschlüsselung mit AES die Nachricht in Blöcke der Länge 16 Byte (128 Bit) zerlegt. Ist der letzte Block der Nachricht dann nur 12 Byte lang, so müssen noch 4 Byte Padding angefügt werden.

Damit der Empfänger nach Entschlüsselung erkennen kann, welche Bytes zur Nachricht und welche zum Padding gehören, muss das Padding klar erkennbar sein.

RFC 2040 spezifiziert eine sehr elegante Methode, dies zu tun: Fehlen noch n Byte, um den Block zu vervollständigen, so wird n mal das Byte 0x0n angefügt. (Im obigen AES-Beispiel also die Bytes 0x04 0x04 0x04 0x04.) Nach Entschlüsselung muss der Empfänger also nur das letzte Byte nehmen, dessen Wert die Anzahl der zu entfernenden Bytes angibt.

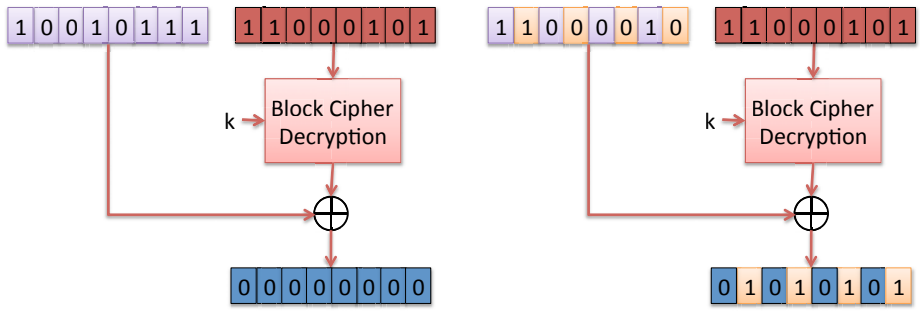

Abb. 1.15 Beim CBC-Modus können einzelne Bits des Klartexts durch Veränderung des IV gezielt verändert werden. Dies ist hier für einen Blocklänge von einem Byte (8 Bit) dargestellt.

Die Verschlüsselung im CBC-Modus garantiert Vertraulichkeit, aber keine Integrität: Durch Veränderung des IV kann auch der erste Klartextblock verändert werden („Malleability“, siehe Abbildung 1.15). Dies kann ein Angreifer nutzen, wenn der Chiffretext von einem Server entschlüsselt wird, das das RFC 2040-Padding überprüft und im Fehlerfall eine Meldung ausgibt. Wir beschreiben dies für eine Blocklänge von 8 Byte.

Zur Entschlüsselung des letzten Bytes cb_8 des ersten Chiffretextblocks $c_1 = (cb_1, cb_2, \ldots, cb_8)$ geht der Angreifer wie folgt vor:

1. Er wählt einen zufälligen Initiaisierungsvektor IV' und sendet (IV', c_1) an den Server. Mit hoher Wahrscheinlichkeit erhält er eine Fehlermeldung, da nach Entschlüsselung das Padding nicht korrekt ist.
2. Er probiert nun für IV'_8, das letzte Byte des falschen Initialisierungsvektors, alle 256 Möglichkeiten durch, und sendet so lange das Paar (IV', c_1) (mit jeweils anderen Werten für das letzte IV-Byte) an den Server, bis er *keine* Fehlermeldung mehr erhält. Wir bezeichnen den so gefundenen Initialisierungsvektor mit IV''.
3. Das Padding ist jetzt also korrekt, und dies kann mehrere Ursachen haben: Das letzte Byte des neuen Klartextes kann 0x01 sein, oder die beiden letzten Bytes können gleich 0x02 0x02 sein, ...
4. Am Wahrscheinlichsten ist es, dass das letzte Byte 0x01 ist, und der Angreifer kann dies relativ leicht verifizieren: Er modifiziert nur das vorletzte Byte IV''_7, und wenn der Server hier ebenfalls keine Fehlermeldung zurückgibt, so war das letzte Byte des Klartexts gleich 0x01.

5. Somit haben wir ein IV'' gefunden, für den gilt $IV''_8 \oplus X_8 = 0x01$, wobei $X = (X_1, X_2, \ldots, X_8)$ die Ausgabe der Blockchiffre ist (vgl. Abbildung 1.16). Daraus können wir $X_8 = IV''_8 \oplus 0x01$ berechnen.
6. Mit Hilfe der Werte X_8 und IV_8 (des 8. Byte des Original-IV) können wir cb_8 leicht berechnen: $cb_8 = X_8 \oplus IV_8$, denn dies ist genau die Berechnung, die bei Verwendung der Originalwerte (IV, c) an dieser Stelle durchgeführt wird.

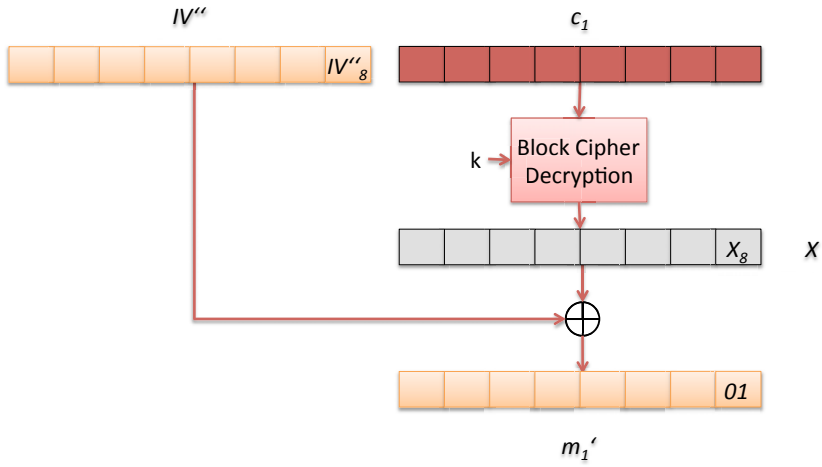

Abb. 1.16 Padding Oracle-Angriff auf den CBC-Modus.

Wir haben jetzt cb_8 berechnet. Um das nächste Byte cb_7 zu ermitteln, verändern wir IV''_8 durch Invertieren der letzten beiden Bits so ab, dass sich mit dem so veränderten neuen IV''' ergibt $IV'''_8 \oplus X_8 = 0x02$. Nun führen wir die Schritte 1 bis 6 sinngemäß für das vorletzte Byte durch: Wir verändern das vorletzte Byte des IV so lange, bis wir ein Padding 0x02 0x02 gefunden haben, berechnen X_7, und daraus dann (mit dem Original-IV) cb_7. Für das drittletzte Byte suchen wir das Padding 0x03 0x03 0x03, und so weiter.

Haben wir so alle Bytes des ersten Klartextblocks ermittelt, so hilft uns die Symmetrie des CBC-Modus weiter: Wir verwenden jetzt c_1 als (Original-)Initialisierungsvektor, c_2 als Chiffretextblock, und variieren die einzelnen Bytes des neuen IV immer so lange, bis wir ein gültiges Padding gefunden haben, und bestimmen daraus zunächst die X-Bytes, und dann mit Hilfe von c_1 die Klartextbytes. Dies funktioniert für alle Blöcke des Chiffretexts.

Fazit: Wird das RFC 2040-Padding verwendet, und gibt ein Server Fehlermeldungen bei Padding-Fehlern zurück, so können wor den Klartext Byte für Byte berechnen. Die Schlüssellänge spielt dabei keine Rolle (DES ist genauso unsicher wie Triple-DES, AES-128 genauso unsicher wie AES-512), wir benötigen im Schnitt nur 128 Serveranfragen zur Entschlüsselung eines Bytes.

Padding Oracle-Angriffe wurden zuerst (theoretisch) von Serge Vaudenay [Vau02] beschrieben, Rizzo und Duong [RD10] haben dann 8 Jahre später gezeigt, dass diese

Angriffe in der Praxis angenadt werden können. Seit 2010 wurde der CBC-Modus mindestens einmal pro Jahr in einer neuen Orakel-Variante praktisch (vgl. Kapitel 7 und Kapitel 12) gebrochen, so dass beim Einsatz von CBC heute Vorsicht geraten ist.

1.7.3 Angriffe auf Hashfunktionen

Eine Hashfunktion gilt als gebrochen, wenn es möglich ist, zwei Urbilder mit gleichem Hashwert zu finden (Brechen der collision resistance); dies ist z.B. für MD5 möglich. Schwieriger ist es, zu einem gegebenen Urbild ein zweites zu finden (Brechen der second preimage resistance). Kann schließlich die Einwegeigenschaft gebrochen werden, so sind automatisch auch die beiden anderen Eigenschaften gebrochen.

1.7.4 Angriffe auf MAC und digitale Signaturen

Die wichtigste Sicherheitseigenschaft für digitale Signaturen ist *Existential Unforgeability (EUF)*. Diese Eigenschaft besagt, dass ein Angreifer (der den privaten Schlüssel natürlich nicht kennt) für eine beliebige Nachricht, zu er der keine Signatur kennt (bzw. angefordert hat, s.u.), auch keine berechnen kann. Dies soll selbst dann gelten, wenn der Angreifer Siganuren zu beliebigen Nachrichten anfordern kann.

Diese Sicherheitseigenschaft gilt analog auch für Message Authentication Codes.

1.7.5 Angriffe auf Protokolle

Bei kryptographischen Protokollen gibt es neben den „normalen“ kryptographischen Attacken auf die verwendeten Algorithmen noch weitere Angriffe, die man berücksichtigen muss. An dieser Stelle sollen nur zwei Beispiele für solche Angriffe herausgegriffen und erläutert werden, weitere Beispiele sind in den nachfolgenden Kapiteln zu finden.

Bei einem *Replay-Angriff* werden alte, aufgezeichnete Nachrichten erneut gesendet. Würden dagegen keine Vorkehrungen getroffen (z.B. durch Verwendung einer TAN), so könnte man sich z.B. bei einem Homebanking-Verfahren einmal 50 € überweisen lassen, die entsprechende (ggf. verschlüsselte und/oder signierte) Nachricht an den Server der Bank aufzeichnen, und dann noch weitere 99 mal an den Server senden, um insgesamt 5000 € überwiesen zu bekommen.

Beim *Man-in-the-middle* -Angriff schaltet sich der Angreifer einfach in die Leitung zwischen zwei Teilnehmer A und B. Er gibt sich A gegenüber als B aus, und B gegenüber als A. Ein prominentes Opfer dieses Angriffs ist das Diffie-Hellman-Protokoll. Ein Angreifer X kann, wie in Abbildung 1.17 dargestellt, so einen Schlüssel k_1 mit A, und einen Schlüssel k_2 mit B aushandeln, und dann den gesamten Datenverkehr zwischen A und B mithören, indem er ihn umverschlüsselt. Wir werden im Kapitel zu IPSec IKE sehen, wie man diesen Angriff abwehren kann.

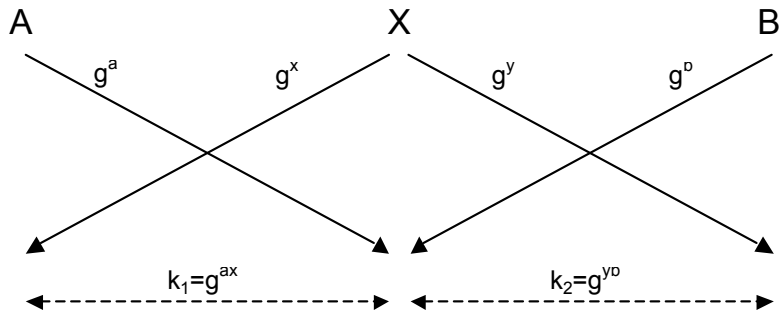

Abb. 1.17 Der „Man-in-the-middle"-Angriff auf das Diffie-Hellman-Protokoll.

1.8 Zertifikate

Im täglichen Leben gibt es zwei Methoden, eine Person zu identifizieren: Über persönliche Bekanntschaft oder durch Vorlage eines amtlichen Ausweises. Wenn man z.B. ein neues Sparkonto bei der Sparkasse im eigenen Dorf eröffnet, wird der Bankmitarbeiter unter „Ausgewiesen durch" wahrscheinlich ein „Persönlich bekannt" eintragen, während in einer Großstadt dazu in der Regel die Vorlage des Personalausweises erforderlich ist.

Auch im Internet wurden beide Verfahren praktiziert: So gab es eine Zeitlang immer wieder PGP-Schlüsselaustauschparties, bei denen man jeweils den eigenen öffentlichen Schlüssel mit anderen austauschte. Danach wusste man genau, an wen man eine verschlüsselte Nachricht sendete, oder wer ein Dokument signiert hatte. Die Möglichkeit der persönlichen Bekanntschaft haben im globalen Internet aber nur wenige Personen. Daher verwundert es nicht, dass die zweite Methode immer größere Bedeutung gewinnt.

Zertifikate sind im Internet das Äquivalent zu amtlichen Dokumenten im normalen Leben. Sie enthalten Angaben zum Aussteller („ausstellende Behörde"), zum Inhaber und zu seinen persönlichen Daten, die insbesondere seine eindeutige Adresse im Internet, z.B. die E-Mail-Adresse, und seinen öffentlichen Schlüssel umfassen. Alle diese Daten werden durch eine digitale Signatur der ausstellenden Stelle verknüpft und dadurch gegen Veränderungen geschützt.

1.8.1 X.509

Als Standard für Zertifikate hat sich der ITU-Standard X.509 [CSF+08] durchgesetzt, der noch zusätzliche Angaben im Zertifikat erlaubt. Der Aufbau eines X.509-Zertifikats ist in Abbildung 1.18 wiedergegeben.

Die aktuelle Versionsnummer des X.509-Standards ist 3. Alle Zertifikate eines Herausgebers erhalten eine unterschiedliche Seriennummer, so dass das Paar (Herausgeber, Seriennummer) ein Zertifikat eindeutig bestimmt.

Um die Signatur des Zertifikats verifizieren zu können, muss man natürlich wissen, welcher Hash- und welcher Signaturalgorithmus zur Erzeugung der Signatur verwendet wurden.

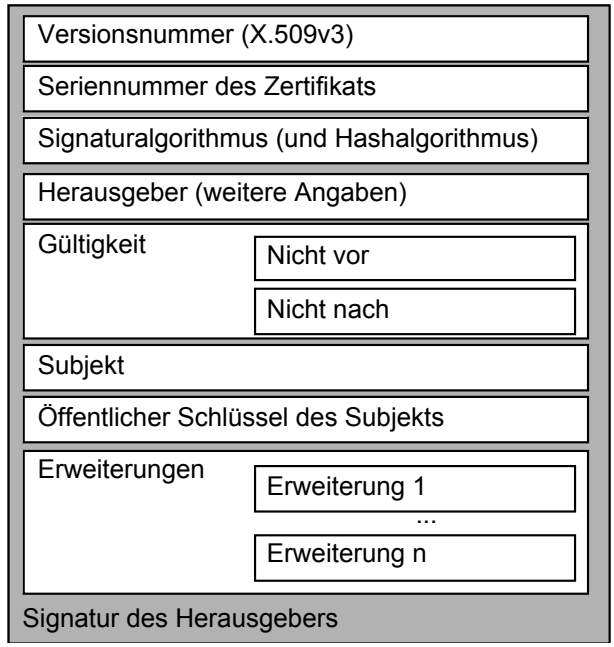

Abb. 1.18 Aufbau eines X.509-Zertifikats.

Die Namen (Identitäten) von Herausgeber und Subjekt sollen nach Möglichkeit in Form eines „Distinguished Name“ angegeben werden. Als Beispiel für einen solchen Namen sei hier der Distinguished Name der IETF angegeben:

- CN = www.ietf.org
- OU = IETF
- O = Foretec Seminars Inc.
- L = Reston
- S = Virginia
- C = US

„C=“ gibt dabei das Land an, „S=“ den (amerikanischen) Bundesstaat, „L=“ die „Location“, „O=“ die Organisation, „OU=“ den Bereich innerhalb der Organisation, und „CN=“ den Namen der Instanz, der hier ein Domain-Name ist, da dieser Name aus dem SSL-Zertifikat des IETF-Webservers entnommen ist. Die strenge geographische Namensgebung passt allerdings nicht immer zu den im Internet üblichen Namen. Daher tauchen in Zertifikaten auch immer wieder andere Namensformen wie Domain-Namen

(SSL-Zertifikate) oder E-Mail-Adressen auf. Diese sind dann bei der Auswertung des Zertifikats oft sogar wichtiger.

Jedes X.509-Zertifikat ist nur für eine gewisse Zeit gültig: Für Endnutzer-Zertifikate ist dies in der Regel ein Jahr, CA- und Wurzelzertifikate (vgl. Unterabschnitt 1.8.2) haben eine deutlich längere Gültigkeit. Abbildung 1.19 zeigt die Ansicht auf ein Zertifikat in der Darstellung des Microsoft-Zertifikatsanzeige.

Wichtig ist natürlich der öffentliche Schlüssel des Subjekts, der ja gerade zertifiziert werden soll. In den Erweiterungsfeldern, die erst ab Version 3 hinzukamen, werden Einschränkungen für den Verwendungszweck des Zertifikats angeführt.

1.8.2 Public Key Infrastrukturen (PKI)

Um solche Zertifikate von Endbenutzern und Ausstellern einfach auf ihre Gültigkeit hin überprüfen zu können, werden sie in der Regel in hierarchische Public Key Infrastrukturen (PKI) eingebettet. Diese Struktur besteht aus einer sog. „Wurzelinstanz“ an der Spitze und untergeordneten Instanzen, den sog. „Certification Authorities (CA)“ und schließlich den Benutzern.

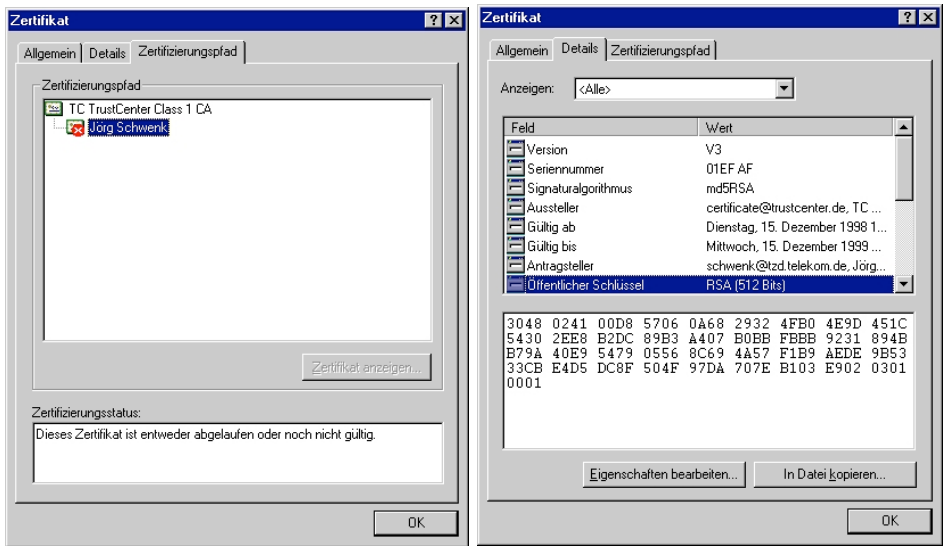

Abb. 1.19 PKI und öffentlicher Schlüssel eines abgelaufenen E-Mail-Zertifikats des Autors.

Die Wurzelinstanz („Root“) erzeugt dabei ein spezielles Zertifikat, das sie selbst signiert: das Wurzel-Zertifikat („Root Certificate“). Mit dem privaten Schlüssel, der zu diesem Wurzel-Zertifikat gehört, werden alle Zertifikate der untergeordneten Stellen signiert. Diese wiederum signieren mit ihrem jeweiligen privaten Schlüssel die ausgestellten Zertifikate der Benutzer. Da in den Zertifikaten jeweils ein Verweis auf die ausstellende Stelle und deren Zertifikat enthalten ist, kann so eine vollständige Kette vom Benutzer-Zertifikat bis zum Wurzel-Zertifikat durchlaufen werden, in der jeweils

die Gültigkeit des untergeordneten Zertifikats mit Hilfe des öffentlichen Schlüssels der übergeordneten Zertifizierungsstelle verifiziert werden kann.

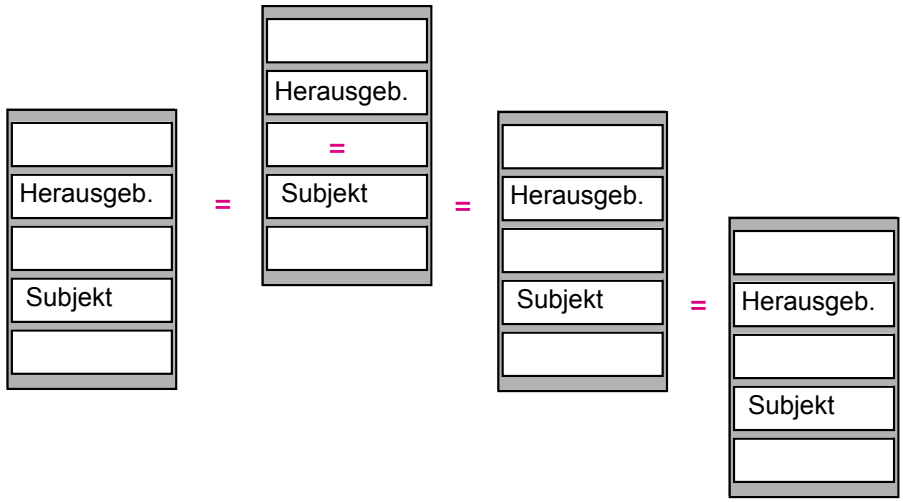

Abb. 1.20 Komplexere PKI mit mehr als einer CA.

Die Instanz, die unter „Subjekt“ in einer Ebene dieser Hierarchie auftritt, wird damit in der folgenden Ebene zum „Herausgeber“. So entstehen die komplexeren PKIs aus Abbildung 1.20. Diese bieten zahlreiche Vorteile:

- Unter nur einem Wurzelzertifikat können durch Anfügen verschiedener CAs Zertifikate unterschiedlichster Art ausgestellt werden: SSL-Zertifikate für Webserver von CA1, Zertifikate für E-Mail von CA2, oder Zertifikate für S/MIME von CA3, um nur einige Beispiele zu nennen.
- Weiter kann man für jede CA verschiedene Sicherheitsrichtlinien („Security Policies“) aufstellen: Um ein E-Mail-Zertifikat der Sicherheitsstufe 1 zu erhalten, muss man nur eine gültige E-Mail-Adresse nachweisen, für ein Zertifikat der Stufe 3 muss bei einer Registrierungsstelle der Personalausweis vorgelegt werden.
- Im firmeninternen Einsatz kann die Struktur der PKI auch die Struktur der Firma widerspiegeln: Das Wurzelzertifikat gehört zur Firmenleitung, und die untergeordneten CAs zu den einzelnen Bereichen und Abteilungen.

Mit dieser flexiblen Struktur sind natürlich noch viele weitere Anwendungen realisierbar. Einige dieser Anwendungen könnte man auch mit verschiedenen Wurzelzertifikaten realisieren, aber dies vervielfacht die Probleme mit Root-Zertifikaten, auf die wir jetzt näher eingehen wollen.

Die Wurzelinstanz ist der Anker des Vertrauens. Wie bereits erwähnt, ist dies ein selbst signiertes Zertifikat, das natürlich zunächst wieder authentisch verteilt werden muss. Viele große Zertifizierer lassen deshalb ihre Wurzel-Zertifikate in die Standardbrowser/E-Mail-Clients/Betriebssysteme integrieren, so dass diese nach der Installation der Software automatisch dem Benutzer zur Verfügung stehen. Damit

wird dem Nutzer vorgeschrieben, welchen Zertifikaten er zu trauen hat, und welchen zunächst einmal nicht.

In bestimmten Einsatzbereichen ist diese Vorgehensweise bereits zum de facto-Standard geworden, weil sie so einfach ist und beim Nutzer keinerlei Kenntnisse zu Zertifikaten oder PKIs voraussetzt. Wir werden im SSL-Kapitel näher darauf eingehen.

Die korrekte, aber leider auch schwierigere Variante wurde für die PKI zum deutschen Signaturgesetz [sig09] gewählt: Vertrauen in das Wurzelzertifikat, das von der Regulierungsbehörde für Telekommunikation und Post (RegTP) verwaltet wird, wurde dadurch geschaffen, dass der öffentliche Schlüssel in gedruckter Form im Bundesgesetzblatt veröffentlicht wurde.

Wie wir im Abschnitt zu den Namensfeldern in X.509-Zertifikaten bereits gesehen haben, sind die Festlegungen des X.509-Standards, der aus der Welt der großen nationalen Telekommunikationsunternehmen stammt, nicht immer direkt auf das Internet übertragbar. Sinnvolle Festlegungen für das Internet wurden daher in [CSF+08] getroffen.

1.8.3 Gültigkeit von Zertifikaten

Ein wichtiger praktischer Aspekt ist die Gültigkeit von Zertifikaten. Wir haben ja bereits gesehen, dass die meisten Zertifikate nur ein Jahr lang gültig sind. Was passiert aber, wenn sie innerhalb dieser Jahresfrist ungültig werden? Dies kann z.B. dann passieren, wenn

- die E-Mail-Adresse geändert wird: Dadurch wird das alte E-Mail-Zertifikat ungültig.
- der Nutzer sein Passwort für seinen privaten Schlüssel vergessen hat. Er kann dann keine mit dem öffentlichen Schlüssel aus dem Zertifikat verschlüsselten Nachrichten mehr entschlüsseln, oder Dokumente signieren.
- der private Schlüssel gestohlen oder berechnet wurde.
- ein Zertifikat versehentlich falsch ausgestellt wurde.
- die Server der CA gehackt wurden[4].

Der X.509-Standard sieht für solche Fälle Sperrlisten, so genannte „Certificate Revocation Lists (CRL)“ vor. Dies sind einfach Listen der Seriennummern derjenigen Zertifikate, die eigentlich noch gültig wären, aber aus einem bestimmten Grund nicht mehr gültig sind. (Dieser Grund kann optional in der CRL mit angegeben werden.) Die ganze Liste wird dann von der Certification Authority, die diese Zertifikate ausgegeben hat, signiert.

[4] Vgl. `http://en.wikipedia.org/wiki/DigiNotar`

Die Idee ist eigentlich einfach, die Umsetzung in der Praxis dafür umso schwieriger. Dies fängt damit an, dass diese CRLs zunächst einmal geladen werden müssen. Doch von wo? Und wie? Wie oft?

Die einzige Möglichkeit, einem Client zuverlässig mitzuteilen, wo er die CRL zu einem Zertifikat laden kann, ist in dem Zertifikat selbst. Bei allen anderen Lösungen muss der Pfad zum Laden der CRL manuell konfiguriert werden.

Das „Wo“ wäre somit geklärt. Das „Wie“ kann auf die gleiche Weise mit angegeben werden, indem das zu verwendende Protokoll (z.B. LDAP oder HTTP) zusammen mit dem Pfad abgespeichert wird. Natürlich muss dann jeder Client dieses Protokoll auch beherrschen, was z.B. bei einem E-Mail-Client nicht unbedingt selbstverständlich ist.

Auf die Frage „Wie oft?“ könnte man jetzt einfach „So oft wie möglich“ antworten, wenn die CRLs nicht so groß werden könnten. Diese Vorgehensweise stellt aber nicht nur die Geduld des Nutzers auf eine harte Probe, der jedes Mal warten muss, bis die CRL geladen ist, sondern bereitet auch enorme Lastprobleme auf den Web- oder LDAP-Servern der CA. Hier muss also ein heuristischer Kompromiss gefunden werden (z.B. einmal pro Woche), und dieser muss dann auch in den entsprechenden Clients konfigurierbar sein.

Eine konsequente Weiterentwicklung des „So oft wie möglich“-Ansatzes ist das Online Certificate Status Protocol (OCSP) [SMA+13]. Mit dem OCSP-Protokoll kann man die Gültigkeit einzelner Zertifikate online überprüfen. Neben den schon oben erwähnten Lastproblemen stellt sich hier noch die Frage, welche Vorteile hier noch die Public-Key Kryptographie bietet: Wenn man vor jeder Verschlüsselung oder Überprüfung der Authentizität einer Nachricht eine Online-Verbindung zu einem vertrauenswürdigen Server aufbauen muss, dann kann man auch gleich ein Protokoll wie Kerberos verwenden.

Die Diskussion darüber, wie gesperrte Zertifikate zu behandeln sind, ist also noch nicht beendet.