

UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Mathematisches Institut

Sicherheitsaspekte kryptographischer Verfahren beim Homebanking

Diplomarbeit

Leipzig, 29. April 2002

vorgelegt von

Lars Nöbel
geb. am: 02.08.1976

Studiengang Diplom-Mathematik

Abstract

In der vorliegenden Arbeit werden kryptographische Verfahren und Protokolle vorgestellt, die im HBCI-Standard zum Einsatz kommen. Das Hauptaugenmerk liegt hierbei auf den derzeit verwendeten Algorithmen DES und RSA sowie deren möglichen Nachfolgern Rijndael und ElGamal mit elliptischen Kurven. Die dafür notwendigen mathematischen Grundlagen werden ebenso wie die grundlegenden Begriffe der Kryptographie eingeführt. Es wird auf Sicherheitsaspekte der untersuchten Algorithmen und auf die zukünftige Entwicklung eingegangen. Dabei stellt sich heraus, daß mit den benutzten Verfahren die Sicherheit der Kommunikationspartner nur unwesentlich bis gar nicht beeinträchtigt werden kann. Beim praktischen Einsatz existieren aber noch Lücken, die für einen Angriff ausgenutzt werden können.

Inhaltsverzeichnis

1	Einleitung	1
2	Mathematische Grundlagen	3
2.1	Hilfsmittel aus der Zahlentheorie	3
2.1.1	Komplexität von Algorithmen	3
2.1.2	Der Euklidische Algorithmus	5
2.1.3	Der Chinesische Restsatz	7
2.1.4	Der Satz von Euler-Fermat	8
2.1.5	Galoisfelder	9
2.2	Einwegfunktionen	9
2.2.1	Faktorisierung natürlicher Zahlen	9
2.2.2	Der diskrete Logarithmus	10
2.2.3	Nichtlineare Transformationen	11
2.3	Erzeugung von Zufallszahlen	11
2.3.1	Zufallszahlengeneratoren	11
2.3.2	Kongruenzgeneratoren	12
2.3.3	Schieberegister	14
2.3.4	Weitere Generatoren	14
2.4	Primzahltests und Faktorisierung	15
2.4.1	Probedivision und Fermat-Test	15
2.4.2	Der Miller-Rabin-Test	16
2.4.3	Pollards Methode	17
2.4.4	Das Quadratische Sieb	18

3	Kryptographische Grundlagen	19
3.1	Grundbegriffe	19
3.1.1	Kryptosysteme	19
3.1.2	Block- und Stromchiffren	20
3.2	Symmetrische Kryptosysteme	23
3.2.1	Cäsar-Chiffre und One-Time-Pad	23
3.2.2	Der DES-Algorithmus	24
3.2.3	Weitere Algorithmen	27
3.3	Asymmetrische Kryptosysteme	29
3.3.1	Einführende Bemerkungen	29
3.3.2	Der RSA-Algorithmus	30
3.3.3	Weitere Verfahren	31
3.4	Hashfunktionen	32
3.4.1	SHA	32
3.4.2	MD4 und seine Varianten	32
3.4.3	RIPEMD-160	33
3.4.4	MDC-2	33
3.4.5	Message Authentication Codes	34
3.5	Digitale Signaturen	34
3.5.1	RSA-Signaturen	34
3.5.2	ElGamal-Signaturen und DSA	34
3.6	Kryptographische Protokolle	35
3.6.1	Festcodes und Wechselcodes	35
3.6.2	Bidirektionale Protokolle	36
3.6.3	Weitere Protokolle	37
4	Angriffe auf Kryptosysteme	39
4.1	Angriffe auf Kryptosysteme	39
4.1.1	Angriffsklassen	39
4.1.2	Brute-Force-Angriff	40
4.1.3	Kryptanalyse	41
4.2	Angriffe auf Protokolle	42
4.2.1	Einfache Angriffe	42
4.2.2	Arglistige Täuschung	42
4.3	Schwachstelle Benutzer	43
4.3.1	Sicherheit der Benutzerdaten	43
4.3.2	Schutz der Benutzerdaten	44

5	Homebanking	45
5.1	Der HBCI-Standard	45
5.1.1	Beschreibung	45
5.1.2	Ablauf des HBCI-Dialogs	46
5.1.3	Sicherheitsmechanismen	47
5.1.4	Signieren von Nachrichten	48
5.1.5	Verschlüsselung von Nachrichten	49
5.1.6	Erstinitialisierung DDV	50
5.1.7	Erstinitialisierung RDH	51
5.1.8	Sicherheitsmedien	53
5.2	Schwachstellen des Protokolls	53
5.2.1	Kommunikation	53
5.2.2	Speicherung der Daten	54
5.3	Analyse der Angriffsmöglichkeiten	55
5.3.1	Allgemeine Angriffe	55
5.3.2	Angriffe auf das Protokoll	57
5.3.3	Angriffe auf Sicherheitsmedien	58
5.3.4	Angriffe auf Benutzer	59
6	Weiterführende Betrachtungen	61
6.1	Neue kryptographische Algorithmen	61
6.1.1	Advanced Encryption Standard	61
6.1.2	Spezifikation von Rijndael (AES)	62
6.2	Weitere mathematische Methoden	66
6.2.1	Elliptische Kurven	66
6.2.2	EC-Kryptographie	69
7	Zusammenfassung	71

Abkürzungsverzeichnis

\oplus	bitweise Addition modulo 2
AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
BBS	Bitgenerator nach Blum, Blum und Shub
BdB	Bundesverband deutscher Banken
CBC	Cipher-Block-Chaining-Modus
CFB	Cipher-Feedback-Modus
CID	Cardholders Information Data
DDV	DES-DES-Verfahren
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
ECB	Electronic-Codebook-Modus
ECM	Faktorisierungsmethode mit elliptischen Kurven (engl. elliptic curve factoring method)
EXPTIME	Komplexitätsklasse (exponentieller Zeitaufwand)
$\gcd(a, b)$	größter gemeinsamer Teiler von a und b (engl. greatest common divisor)
$\text{GF}(p^n)$	Galoisfeld der Ordnung p^n
HBCI	Homebanking Computer Interface
IDEA	International Data Encryption Algorithm
IFX	International Financial Exchange
IP	Internet Protocol
IPSEC	IP Security Protocol
ISO	International Standardization Organization
IV	Initialisierungsvektor
MAC	Message Authentication Code
MDC	Modification Detection Code
MDx	Message Digest x
MOV	Algorithmus von Menezes, Okamoto und Vanstone

N	Menge der natürlichen Zahlen
NFS	Zahlkörpersieb (engl. number field sieve)
NIST	National Institute of Standards and Technology
NP	Komplexitätsklasse (nichtdeterministisch polynomial)
NSA	National Security Agency
$O()$	Ordnung (Landau'sches O-Symbol)
OFB	Output-Feedback-Modus
OFX	Open Financial Exchange
P	Komplexitätsklasse (polynomialer Zeitaufwand)
PIN	Persönliche Identifizierungsnummer
PSPACE	Komplexitätsklasse (polynomialer Speicherbedarf)
QS	Quadratisches Sieb
RACE	Research and Development in Advanced Communication Technologies
RCx	Rivest Cipher x
RDH	RSA-DES-Hybridverfahren
RIPE	RACE Integrity Primitives Evaluation
RSA	Kryptosystem von Rivest, Shamir und Adleman
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
SSSA	Algorithmus von Smart, Semaev, Satoh und Araki
TAN	Transaktionsnummer
TCP	Transmission Control Protocol
\mathbb{Z}	Menge der ganzen Zahlen
ZKA	Zentraler Kreditausschuß

Kapitel 1

Einleitung

„In its simplest and most ancient form, cryptography is the problem of secure message exchange over insecure channels.“

Shafi Goldwasser

„Die besten Traktate über Kryptographie sind Werke ungläubiger Gelehrter. [...] Es gibt keine Geheimschrift, die sich nicht mit ein wenig Geduld entziffern ließe!“

Umberto Eco in „Der Name der Rose“

Kryptographie ist die Lehre von den Problemen der Datensicherheit und den Techniken, die zur Realisierung der Datensicherheit verwendet werden. Die Grundlagen der heutigen Kryptographie sind schon seit längerem erforscht, eine große Anzahl von Algorithmen zur Anwendung in der Praxis wurde bereits entwickelt und wird heutzutage in sehr vielen Bereichen eingesetzt. Trotzdem erfreut sich dieses Fachgebiet in der Forschung immer noch großer Beliebtheit, denn alle bisherigen Ergebnisse können nur eine zeitweilige Sicherheit garantieren. Schon in naher Zukunft kann es passieren, daß ein Verfahren entdeckt wird, welches die bestehenden Verschlüsselungstechniken unsicher macht. Auch die in der Praxis eingesetzten Protokolle schützen die Benutzer nicht zwangsläufig vor Angriffen, die Fahrlässigkeit eines Einzelnen kann zu Sicherheitslücken im gesamten Kommunikationssystem führen. Die Analyse von Angriffen auf vielfach eingesetzte Algorithmen und Protokolle ist Ziel dieser Arbeit.

Die Grundlagen für diese Arbeit lieferten drei Standardwerke. Das Buch „Einführung in die Kryptographie“ von Johannes Buchmann [2] schildert die mathematischen Grundlagen, beginnend bei den natürlichen und ganzen Zahlen bis zu den neuesten kryptographischen Ansätzen. Das Werk „Angewandte Kryptographie“ von Bruce Schneier [9] liefert eine detaillierte Beschreibung der kryptographischen Algorithmen und deren Implementierung. Ein allgemeiner Überblick über die gesamte Landschaft der heutigen Kryptographie ist im Buch „Moderne Verfahren der Kryptographie“ von Albrecht Beutelspacher et al. [1] zu finden. Für weitergehende Informationen sei hier auf diese drei Werke verwiesen.

Die vorliegende Arbeit ist wie folgt gegliedert: In Kapitel zwei werden zunächst einige Hilfsmittel aus der Mathematik vorgestellt. Kapitel drei beschäftigt sich mit den grundlegenden Begriffen der Kryptographie, während Kapitel vier auf Angriffe gegen Kryptosysteme eingeht. Im fünften Kapitel wird die Analyse der Angriffsmöglichkeiten anhand eines Beispiels aus der Praxis dargelegt. Schließlich wird in Kapitel sechs ein Blick auf neue kryptographische Verfahren geworfen.

Kapitel 2

Mathematische Grundlagen

2.1 Hilfsmittel aus der Zahlentheorie

2.1.1 Komplexität von Algorithmen

Die Komplexität von Algorithmen ist der Aufwand zur Lösung eines Problems. Sie ist eine Funktion der Größe der Eingangsdaten. Zu ihrer Messung kann die Anzahl der durchgeführten Operationen, der benötigte Speicherplatz oder die Rechenzeit benutzt werden. Für die weiteren Betrachtungen ist nur ihre Größenordnung entscheidend, daher wird die Komplexität von Algorithmen mit dem folgenden Ordnungssymbol¹ angegeben:

Definition 2.1 Das **Ordnungssymbol** $O(g(x))$ beschreibt die Größenordnung einer Funktion. Es ist $f(x) \sim O(g(x))$ für $x \rightarrow a$, wenn es eine Umgebung $U(a)$ und eine reelle Zahl C gibt, so daß gilt:

$$\left| \frac{f(x)}{g(x)} \right| \leq C \text{ für alle } x \in U(a) \text{ mit } x \neq a$$

Insbesondere gilt $f(x) \sim O(g(x))$ für $x \rightarrow a$, falls $\lim_{x \rightarrow a} \frac{f(x)}{g(x)}$ existiert.

Es wird stets der Fall betrachtet, der den größten Aufwand erfordert. Konstante Faktoren und Terme niederer Ordnung werden vernachlässigt.

¹Siehe Teubner-Taschenbuch der Mathematik [16, Seite 253].

Für einige Verschlüsselungstechniken werden Algorithmen benötigt, die nur unter Kenntnis zusätzlicher Eingangsdaten effizient arbeiten, sonst aber einen ungleich höheren Zeitaufwand erfordern. Effiziente Algorithmen besitzen eine Komplexität, welche höchstens so groß ist wie ein Polynom endlichen Grades, sie haben eine sogenannte **polynomiale Laufzeit**. Das heißt, wenn z_1, z_2, \dots, z_n die Größen der n Eingangsdaten sind, dann müssen nichtnegative ganze Zahlen a_1, a_2, \dots, a_n existieren, so daß die Komplexität f des Algorithmus wie folgt dargestellt werden kann:

$$f(z_1, z_2, \dots, z_n) \sim O(z_1^{a_1} z_2^{a_2} \dots z_n^{a_n})$$

In der Komplexitätstheorie werden Probleme anhand ihrer Komplexität in verschiedene Klassen eingeteilt. In die Klasse P gehören alle Probleme, die mit polynomialem Zeitaufwand lösbar sind. Ein Problem gehört zur Klasse NP, wenn die Verifizierung einer Lösung mit polynomialem Zeitaufwand möglich ist. Das Finden der Lösung kann dabei auch einen höheren Aufwand haben. Alle Probleme der Klasse P sind in der Klasse NP enthalten.

Ungeklärt ist bisher, ob es ein Problem gibt, das zur Klasse NP, aber nicht zur Klasse P gehört. Ein NP-Problem heißt NP-vollständig, wenn es mindestens so schwer ist wie jedes beliebige Problem der Klasse NP. Ein solches Problem repräsentiert sozusagen die gesamte Klasse.

Eine weitere Klasse ist PSPACE, die Klasse aller Probleme, welche mit polynomialem Speicherplatz gelöst werden können. Die Klasse NP ist in PSPACE enthalten, auch hier ist die Gleichheit noch ungeklärt. Im gleichen Sinne wie oben gibt es PSPACE-vollständige Probleme, die die gesamte Klasse repräsentieren.

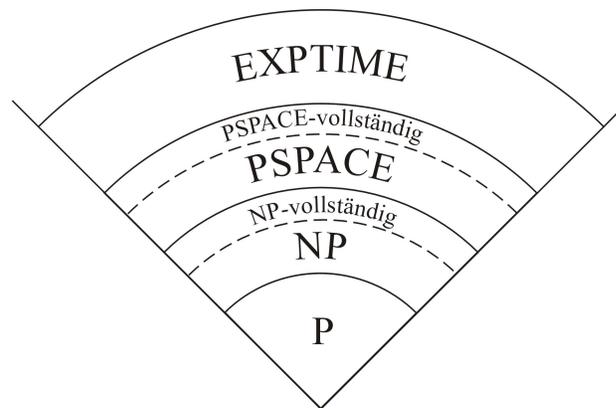


Abbildung 2.1: Komplexitätsklassen

Probleme, die mit exponentiellem Zeitaufwand lösbar sind, bilden die Klasse EXPTIME. Alle bisherigen Klassen sind in ihr enthalten, allerdings wurde bereits bewiesen, daß es Probleme in EXPTIME gibt, die nicht einmal in PSPACE und damit in keiner der anderen Klassen liegen.

Bei den meisten kryptographischen Algorithmen ist die Sicherheit dadurch gegeben, daß die Berechnungen der beteiligten Parteien nur einen polynomialen Zeitaufwand haben, während jedoch ein Angreifer mindestens ein NP-vollständiges Problem lösen muß.

2.1.2 Der Euklidische Algorithmus

Es existieren sehr effiziente Algorithmen für die Grundrechenarten, die nur einen geringen polynomialen Zeitaufwand benötigen. Der Euklidische Algorithmus ist ein gleichermaßen effizienter Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen. Der Ablauf gestaltet sich folgendermaßen²:

Euklidischer Algorithmus

```
procedure euklid(a,b;g);
if a = 0 then begin
  g := b;
end
else if b = 0 then begin
  g := a;
end
else begin
  repeat
    r := a mod b;
    a := b;
    b := r;
  until r = 0;
  g := a;
end
return (g);
```

²Siehe Beutelspacher et al. [1, Seite 116].

Alle Algorithmen sind in der Programmiersprache PASCAL notiert.

Häufig wird auch die Vielfachsummendarstellung des größten gemeinsamen Teilers gesucht. Diese wird durch das folgende Lemma geliefert³:

Lemma 2.1 (Lemma von Bézout) *Seien a und b ganze Zahlen, die nicht beide Null sind. Weiter sei $g = \gcd(a, b)$. Dann gibt es ganze Zahlen s und t mit $g = sa + tb$. Letzteres heißt **Vielfachsummendarstellung** des größten gemeinsamen Teilers.*

Zur Berechnung von s und t wird eine erweiterte Version des Euklidischen Algorithmus eingesetzt³:

Erweiterter Euklidischer Algorithmus

```

procedure xeuklid(a,b;g,s,t);
if a = 0 then begin
  g := b; s := 0; t := 1;
end
else if b = 0 then begin
  g := a; s := 1; t := 0;
end
else begin
  s1 := 1; s2 := 0; s := 0;
  t1 := 0; t2 := 1; t := 1;
  while (a mod b) <> 0 do begin
    q := a div b; r := a mod b;
    s := s1 + q * s2;
    t := t1 + q * t2;
    s1 := s2; s2 := s;
    t1 := t2; t2 := t;
    a := b; b := r;
  end;
  g := b;
end;
return (g,s,t);

```

³Siehe Beutelspacher et al. [1, Seite 117].

2.1.3 Der Chinesische Restsatz

Für die Erläuterung der Funktionsweise einiger Kryptosysteme wird der Chinesische Restsatz benötigt. Er lautet⁴:

Satz 2.2 (Chinesischer Restsatz) *Seien m_1, m_2, \dots, m_n paarweise teilerfremde natürliche Zahlen und seien a_1, a_2, \dots, a_n ganze Zahlen. Dann hat das Gleichungssystem*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_n \pmod{m_n} \end{aligned}$$

eine Lösung x , die eindeutig modulo $m = \prod_{i=1}^n m_i$ ist.

Normalerweise wird dieser Satz in leicht veränderter Form benutzt. Mit Hilfe des Lemmas von Bézout aus dem vorherigen Abschnitt kann folgender Spezialfall formuliert werden⁵:

Folgerung 2.3 *Seien m_1 und m_2 zwei teilerfremde natürliche Zahlen und sei $s \cdot m_1 + t \cdot m_2 = 1$ die zugehörige Vielfachsummendarstellung. Dann hat das Gleichungssystem*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \end{aligned}$$

die Lösung $x = a_2 \cdot s \cdot m_1 + a_1 \cdot t \cdot m_2$, die eindeutig modulo $m = m_1 m_2$ ist.

Die Lösung des Gleichungssystems kann berechnet werden, indem der erweiterte Euklidische Algorithmus auf m_1 und m_2 angewendet wird und die so erhaltenen Werte s und t in die Lösungsformel eingesetzt werden.

⁴Siehe Buchmann [2, Seite 43].

⁵Siehe Beutelspacher et al. [1, Seite 117].

2.1.4 Der Satz von Euler-Fermat

Definition 2.2 Die positiven natürlichen Zahlen a und b heißen **relativ prim** zueinander, falls $\gcd(a, b) = 1$ gilt.

Definition 2.3 Die **Eulersche Funktion** $\varphi(a)$ einer natürlichen Zahl a ist die Anzahl aller positiven natürlichen Zahlen n mit $n \leq a$, die relativ prim zu a sind. Insbesondere gilt für alle Primzahlen p : $\varphi(p) = p - 1$.

Damit kann ein weiteres Hilfsmittel aus der Zahlentheorie beschrieben werden, der Satz von Euler-Fermat. Dieser spielt bei einigen Primzahltests eine wichtige Rolle:

Satz 2.4 (Satz von Euler-Fermat) Für alle positiven natürlichen Zahlen a und b , die relativ prim zueinander sind, gilt:

$$a^{\varphi(b)} \equiv 1 \pmod{b}$$

Für den Fall, daß b eine Primzahl ist, gilt $\varphi(b) = b - 1$. Dadurch ergibt sich der kleine Satz von Fermat als Spezialfall des obigen Satzes⁶:

Folgerung 2.5 (Kleiner Satz von Fermat) Für jede Primzahl b und jede natürliche Zahl a mit $1 \leq a < b$ gilt:

$$a^{b-1} \equiv 1 \pmod{b}$$

Beweis: Der Satz von Euler-Fermat wird hier etwas allgemeiner bewiesen. Dazu sei G eine endliche abelsche Gruppe mit einem neutralen Element e . Die Ordnung von G sei \tilde{G} . Ist nun g ein Element von G , so ist die Abbildung $G \rightarrow G, h \mapsto gh$ eine Bijektion. Daher gilt $\prod_{h \in G} h = \prod_{h \in G} (gh) = g^{\tilde{G}} \prod_{h \in G} h$. Somit gilt für alle $g \in G$: $g^{\tilde{G}} = e$. Aus dem Restklassenring $\mathbb{Z}/m\mathbb{Z}$ entsteht durch folgende Definition eine Gruppe bezüglich der Multiplikation:

$$(\mathbb{Z}/m\mathbb{Z})^* = \{a \in \mathbb{Z}/m\mathbb{Z} \mid \gcd(a, m) = 1\}$$

Dies ist eine endliche abelsche Gruppe, welche die Ordnung $\varphi(m)$ hat⁷. Daraus folgt die Behauptung. □

⁶Historisch gesehen wurde der Satz in dieser Form zuerst 1640 von Fermat formuliert, ehe Euler ihn 1760 verallgemeinerte.

⁷Die Eulersche Funktion $\varphi(m)$ kann auch als Anzahl der Elemente dieser Gruppe definiert werden. Diese Definition ist äquivalent zu der vorher erwähnten.

2.1.5 Galoisfelder

Eine Menge K , auf der eine Addition und eine Multiplikation erklärt ist, heißt Körper, falls K bezüglich der Addition und $K \setminus \{0\}$ bezüglich der Multiplikation abelsche Gruppen sind und falls für alle $a, b, c \in K$ das Distributivgesetz $(a + b) \cdot c = a \cdot c + b \cdot c$ gilt. Besitzt ein Körper nur eine endliche Anzahl von Elementen, dann wird er endlicher Körper bzw. **Galoisfeld** genannt.

Eine (echte) Teilmenge eines Körpers heißt (echter) Unterkörper, wenn sie bezüglich derselben Rechenoperationen ebenfalls ein Körper ist. Ein Primkörper ist ein Körper, der keine echten Unterkörper besitzt. Endliche Primkörper sind zum Gaußschen Restklassenring $\mathbb{Z}/p\mathbb{Z}$ isomorph, wobei die Primzahl p die Anzahl der Elemente des entsprechenden Primkörpers ist.

Zu jeder Primzahl p und zu $n = 1, 2, \dots$ gibt es bis auf Isomorphie genau einen Körper mit p^n Elementen, welcher mit $GF(p^n)$ bezeichnet wird. Auf diese Weise können alle möglichen Galoisfelder erzeugt werden. Die speziellen Galoisfelder $GF(p)$ sind zu $\mathbb{Z}/p\mathbb{Z}$ isomorph.

2.2 Einwegfunktionen

2.2.1 Faktorisierung natürlicher Zahlen

Wichtige Grundbausteine der Kryptographie sind Funktionen, die leicht berechnet, aber schwer invertiert werden können. Durch sie kann zum Beispiel verhindert werden, daß ein Angreifer den Schlüssel berechnen kann, wenn er eine Nachricht und den dazugehörigen Geheimtext vorliegen hat. Eine Funktion $g : S \rightarrow T$ wird **Einwegfunktion** genannt, wenn sie folgende Anforderungen erfüllt:

1. $g(s)$ ist für alle $s \in S$ leicht berechenbar, das heißt, g kann als Algorithmus mit polynomialer Laufzeit dargestellt werden.
2. Für fast alle Elemente $t \in T$ ist es praktisch unmöglich, ein $s \in S$ mit $g(s) = t$ zu berechnen. Insbesondere darf kein Algorithmus mit polynomialer Laufzeit für g^{-1} bekannt sein⁸.
3. $g(s)$ und s müssen ähnliche Größenverhältnisse aufweisen⁹.

⁸Die Forderung der Nichtexistenz eines solchen Algorithmus ist aus heutiger Sicht viel zu hart, da selbst NP-vollständige Funktionen g^{-1} dies nicht garantieren können.

⁹Siehe Apel [17, Seite 52]. Diese eher praktische Anforderung schließt Funktionen des Typs $g(x) = O(\log_2 \log_2 x)$ aus.

Es existieren mehrere Funktionen, welche die Anforderungen an Einwegfunktionen erfüllen. Das RSA-Kryptosystem (siehe Kapitel 3.3.2) basiert auf der Multiplikation zweier Primzahlen, also $g(p, q) = pq$. Diese Operation ist sehr effizient durchführbar. Die Umkehrung, also die Faktorisierung des Produkts, ist dagegen nur schwer durchführbar, es sei denn, einer der Faktoren ist trivial. Bisher ist noch kein Algorithmus mit polynomialer Laufzeit für g^{-1} bekannt. Einige Faktorisierungsalgorithmen mit nichtpolynomialem Zeitaufwand werden später noch erwähnt.

2.2.2 Der diskrete Logarithmus

Definition 2.4 Sei a eine zu n teilerfremde natürliche Zahl mit $1 \leq a < n$. Sind sämtliche Potenzen a^d mit $1 \leq d < \varphi(n)$ modulo n von 1 verschieden, so heißt a **primitive Wurzel** von n .

Sei nun $n \in \mathbb{N}$ eine Zahl, welche primitive Wurzeln besitzt und a eine dieser Wurzeln. Weiter sei $y \in \mathbb{N}$ eine zu n teilerfremde Zahl mit $1 \leq y < n$. Dann existiert genau eine Zahl $x \in \mathbb{N}$ mit $0 \leq x < \varphi(n)$, so daß gilt:

$$y \equiv a^x \pmod{n}$$

Die Zahl x heißt **diskreter Logarithmus** von y zur Basis a modulo n . Die Berechnung von y bei gegebenem x ist in polynomialer Laufzeit möglich, für die Umkehrfunktion jedoch ist derzeit kein Algorithmus bekannt, der dies leistet. So besitzen die effizientesten heute existierenden Verfahren, der Babystep-Giantstep-Algorithmus, der Pollard- ρ -Algorithmus und das Pohlig-Hellman-Verfahren, nur einen nichtpolynomialen Zeitaufwand¹⁰.

Der diskrete Logarithmus kann für alle endlichen abelschen Gruppen (multiplikativ geschrieben) definiert werden. Dazu sei n die Anzahl der Elemente der endlichen abelschen Gruppe G . Weiterhin sei $P \in G$ und $Q \in \{P^z \mid z \in \mathbb{Z}\}$. Die Zahl $k \bmod n$ aus $\mathbb{Z}/n\mathbb{Z}$, für die $Q = P^k$ gilt, wird diskreter Logarithmus von Q zur Basis P genannt.

¹⁰Siehe Buchmann [2, 151ff.].

2.2.3 Nichtlineare Transformationen

Es seien p_1, \dots, p_m nichtlineare Polynome in n Variablen über dem zweielementigen Körper $GF(2)$. Alle in den Polynomen auftretenden Potenzprodukte sind wegen $a^2 = a$ für alle $a \in GF(2)$ quadratfrei. Durch

$$g(a_1, \dots, a_n) = (p_1(a_1, \dots, a_n), \dots, p_m(a_1, \dots, a_n))$$

wird eine Funktion $g : GF(2)^n \rightarrow GF(2)^m$ beschrieben, welche als **nicht-lineare Transformation** bezeichnet wird. Der Funktionswert $g(a_1, \dots, a_n)$ ist für beliebige $a_1, \dots, a_n \in GF(2)^n$ mit nur polynomialem Zeitaufwand berechenbar. Ist jedoch ein m -Tupel $(b_1, \dots, b_m) \in GF(2)^m$ gegeben, so ist es sehr schwer, ein n -Tupel $(a_1, \dots, a_n) \in GF(2)^n$ zu finden, welches die Gleichung $g(a_1, \dots, a_n) = (b_1, \dots, b_m)$ erfüllt. Dazu müsste für das nicht-lineare algebraische Gleichungssystem

$$\begin{aligned} p_1(x_1, \dots, x_n) &= b_1 \\ &\vdots \\ p_m(x_1, \dots, x_n) &= b_m \end{aligned}$$

eine Lösung modulo 2 gefunden werden. Dies ist ein NP-vollständiges Problem und damit praktisch nicht durchführbar.

2.3 Erzeugung von Zufallszahlen

2.3.1 Zufallszahlengeneratoren

Für viele kryptographische Algorithmen werden Zufallszahlen benötigt. Es gibt mehrere Möglichkeiten, zufällige Bitfolgen bereitzustellen. Ein Ansatz dafür ist die Verwendung von zufälligen Phänomenen der realen Welt. Dazu gehören physikalische Prozesse, etwa Hintergrundstrahlung, radioaktiver Zerfall oder Diodenrauschen. Eine weitere Möglichkeit ist der Einsatz von Softwarealgorithmen, die zum Beispiel die Zeit zwischen Anschlägen der Tastatur oder Mausbewegungen messen.

Jede dieser Möglichkeiten ist aufwendig und kann nur für eine kleine Menge von Zufallszahlen verwendet werden. Daher werden diese Ansätze eher zur Erzeugung eines zufälligen Startwertes für einen anderen Typ von Algorithmen benutzt, den Pseudozufallszahlengeneratoren. Diese erzeugen aus einem Eingabewert eine Folge von Werten mit einer festgelegten Generatorfunktion. Dabei sind aus statistischer Sicht zwei Gesichtspunkte wichtig:

1. Die Ausgabewerte sollten gemäß der erwarteten statistischen Verteilung der Werte auftreten.
2. Die Folge sollte keine Perioden aufweisen.

Für kryptographische Zwecke ist es weiterhin wichtig, daß es unmöglich ist, die Ausgabe des Generators vorhersehen zu können. Inzwischen wird sogar der Standpunkt eingenommen, daß es selbst mit einem speziell konstruierten Rechner nicht möglich sein darf, echt zufällige Folgen und erzeugte Folgen in Polynomialzeit unterscheiden zu können.

2.3.2 Kongruenzgeneratoren

Einen einfachen Generator von Pseudozufallszahlen liefert die Methode der linearen Kongruenzen. Ausgehend von einem Startwert x_0 werden dabei die Folgenglieder durch eine lineare Rekursion berechnet:

$$x_{i+1} = x_i \cdot a + b \pmod{m}$$

Die Qualität dieses sogenannten **multiplikativen Kongruenzgenerators** hängt allerdings stark von der Wahl der Parameter a , b und m ab. Eine zufällige Auswahl dieser Parameter hat fast immer einen schlechten Generator zur Folge. Außerdem ist die so erhaltene Folge periodisch, denn ein Folgenglied kann nur einen von m verschiedenen Werten annehmen. Spätestens also beim $(m+1)$ -ten Folgenglied tritt einer dieser m Werte erneut auf. Ab diesem Augenblick werden dieselben Berechnungen durchgeführt wie beim ersten Auftreten des Wertes, es ergibt sich ein periodisches Verhalten. Für eine optimale Ausbeute wird versucht, diese Periode zu maximieren. Das wird durch folgende Zusatzbedingungen erreicht:

1. $\gcd(b, m) = 1$
2. Für alle Primzahlen p gilt: $p|m \Rightarrow p|(a - 1)$
3. $4|m \Rightarrow 4|(a - 1)$

Mehrere Parameterkonstellationen wurden mit Spektraltests untersucht¹¹. Dabei wurden auch multiplikative Kongruenzgeneratoren entdeckt, die sehr gute statistische Eigenschaften aufweisen. Ein Beispiel dafür ist der folgende Generator:

$$x_{i+1} = x_i \cdot 6364136223846793005 + 1 \pmod{2^{64}}$$

Ein weiterer Generator mit guten Eigenschaften wird im ISO-C-Standard für die Funktion `rand()` vorgeschlagen:

$$x_{i+1} = x_i \cdot 1103515245 + 12345 \pmod{2^{32}}$$

Leider sind die multiplikativen Kongruenzgeneratoren für kryptographische Anwendungen unbrauchbar, denn durch die Kenntnis weniger Werte sind alle künftigen mit einer linearen Approximation vorhersagbar. Deswegen wurden weitere Untersuchungen zu Kongruenzgeneratoren mit dem Ziel durchgeführt, einen Generator ohne diese Vorhersage-Eigenschaft konstruieren zu können. 1986 veröffentlichten L. Blum, M. Blum und M. Shub¹² einen solchen Generator, den nach ihnen benannten **BBS-Bitgenerator**. Dieser benötigt zunächst zwei Primzahlen $p, q \equiv 3 \pmod{4}$ und einen zu $n = pq$ teilerfremden Wert x . Der Startwert wird mittels $x_0 = x^2 \pmod{n}$ berechnet, alle weiteren Folgenglieder werden durch folgende Rekursion bestimmt:

$$x_{i+1} = x_i^2 \pmod{n}$$

Als Zufallswert wird jedem Folgenglied x_i das niedrigstwertige Bit entnommen. Eine aus diesen Zufallsbits gewonnene Folge hat statistisch gute Eigenschaften und kann auch für kryptographische Zwecke verwendet werden. Durch die Auswertung einzelner Bits steigt allerdings der Rechenaufwand.

¹¹Siehe Knuth [5].

¹²Siehe Pomerance et al. [7, Seite 120].

2.3.3 Schieberegister

Als hardwaremäßig leicht zu realisierender Algorithmus wird das sogenannte **binäre lineare Schieberegister** eingesetzt. Dazu werden n Bit-Register in einer Reihe benötigt, welche n einzelne Bits speichern. Diese Register werden mit je einem Startbit zufällig initialisiert. Danach werden einige Taktzyklen abgearbeitet, wobei in einem Taktzyklus folgende Operationen durchgeführt werden:

- Die Inhalte einiger, vorher festgelegter Zellen werden mittels \oplus verknüpft.
- Der Inhalt jedes Registers wird um eine Stelle nach links verschoben.
- Der Überlauf am linken Rand bildet das Ausgabe-Bit dieses Taktes.
- Das nun leere Register rechts außen wird mit dem in Schritt 1 berechneten Wert besetzt.

Die Folgen, die durch ein Schieberegister erzeugt werden, haben gute statistische Eigenschaften, sind aber wegen der Linearität durch die Kenntnis weniger Bits vollständig vorhersehbar. Deswegen werden stattdessen nicht-lineare Schieberegister benutzt, das heißt, die Funktion \oplus wird durch eine nichtlineare Verknüpfungsfunktion ersetzt. Eine weitere Möglichkeit ist die Kopplung mehrerer linearer Schieberegister auf nichtlineare Weise.

2.3.4 Weitere Generatoren

Die Verallgemeinerung des BBS-Bitgenerators liefert die Klasse der Potenzierungsgeneratoren, die durch die folgende Rekursion beschrieben werden:

$$x_{i+1} = x_i^d \pmod{n}$$

Mit Hilfe der Parameter (d, n) werden die einzelnen Generatoren unterschieden. So ergibt sich für die Konstellation $(2, pq)$ der BBS-Bitgenerator. Falls n das Produkt von zwei verschiedenen ungeraden Primzahlen p und q sowie außerdem d zu $\varphi(n) = (p-1)(q-1)$ teilerfremd ist, so entspricht diese Operation der Entschlüsselungsfunktion des RSA-Kryptosystems. Deswegen wird dieser Generator auch als RSA-Bit-Generator bezeichnet.

Ein weiterer Generator, der eine Einwegfunktion nutzt, ist der diskrete Exponentialgenerator. Die Rekursion wird hier folgendermaßen beschrieben:

$$x_{i+1} = g^{x_i} \pmod{n}$$

Ist n eine ungerade Primzahl und g eine natürliche Zahl $< p$, so führt dies zu der diskreten Exponentialfunktion, deren Umkehrfunktion, wie oben beschrieben, sehr schwierig zu berechnen ist.

Auch der DES-Algorithmus (siehe Kapitel 3.2.2) kann zur Erzeugung von Zufallszahlen herangezogen werden. Dazu wird ein zufälliger Startwert mittels eines geheimen Schlüssels chiffriert. Das Ergebnis wird ausgegeben und gleichzeitig als neuer Eingabewert benutzt. Die Sicherheit wird hier durch die Geheimhaltung des Schlüssels gewährleistet, so daß die Zufallszahlenfolge ohne seine Kenntnis nicht vorhersehbar ist. Bei einigen Banken wird dieses Verfahren noch heute eingesetzt.

2.4 Primzahltests und Faktorisierung

2.4.1 Probedivision und Fermat-Test

Für asymmetrische Verschlüsselungsalgorithmen werden große zufällige Primzahlen benötigt. Um festzustellen, ob eine Zufallszahl prim ist, werden verschiedene Primzahltests angewendet. Außerdem existieren Algorithmen, welche die Teiler einer großen Zahl ermitteln. Da dies jedoch die Umkehrung einer Einwegfunktion ist, sind solche Faktorisierungsalgorithmen nach heutigem Erkenntnisstand nicht mit polynomialem Zeitaufwand durchführbar.

Die einfachste Methode, Primteiler einer Zahl n festzustellen, ist die Probedivision. Dazu kann das Sieb des Eratosthenes¹³ mit allen Primzahlen kleiner als \sqrt{n} benutzt werden. Allerdings werden mindestens $\sqrt{n}/\log \sqrt{n}$ Probedivisionen¹⁴ benötigt, ehe nachgewiesen ist, daß n eine Primzahl ist. Daher werden meist nur Primzahlen kleiner als 10^6 verwendet und zusätzlich andere Tests durchgeführt.

¹³Siehe Teubner-Taschenbuch der Mathematik [16, Seite 719].

¹⁴Siehe Buchmann [2, Seite 106].

Der kleine Satz von Fermat ist die Grundlage für den Fermat-Test. Dazu wird eine natürliche Zahl $a < n$ gewählt und der Wert $y = a^{n-1} \pmod{n}$ berechnet. Gilt $y \neq 1$, so muß nach dem Satz von Fermat die Zahl n zusammengesetzt sein. Der Fermat-Test kann leider nicht beweisen, ob eine Zahl prim ist, denn im Falle $y = 1$ kann n sowohl prim als auch zusammengesetzt sein. Selbst mehrere Wiederholungen mit verschiedenen a sind nicht ausreichend, da zusammengesetzte natürliche Zahlen existieren, die der Fermat-Test nicht erkennen kann. Diese Zahlen heißen Carmichael-Zahlen. Es gibt unendlich viele dieser Zahlen, die kleinste ist $561 = 3 \cdot 11 \cdot 17$.

2.4.2 Der Miller-Rabin-Test

Um Carmichael-Zahlen mit einem Test zu entlarven, wird eine stärkere Aussage als der kleine Satz von Fermat benötigt. Dazu ist der folgende Satz hilfreich¹⁵:

Satz 2.6 *Ist n eine Primzahl und gilt $n - 1 = d \cdot 2^k$ mit ungeradem d , so gilt für jede positive natürliche Zahl $a \leq n - 1$ eine der beiden folgenden Aussagen:*

1. $a^d \equiv 1 \pmod{n}$
2. $a^{2^i d} \equiv -1 \pmod{n}$ für ein i mit $0 \leq i < k$

Dieses Resultat ist die Grundlage für den Miller-Rabin-Test. Für diesen Test muß eine zum Primzahlkandidaten n teilerfremde Zahl a gefunden werden, die keine der beiden Aussagen erfüllt. Mehr als $\frac{3}{4}$ der Zahlen $\{2, 3, \dots, n - 1\}$ gewährleisten dies. Das bedeutet, daß eine zusammengesetzte Zahl einen Testdurchlauf nur mit einer Wahrscheinlichkeit von weniger als $\frac{1}{4}$ übersteht. Werden nun z Tests mit verschiedenen a durchgeführt, so beträgt diese Wahrscheinlichkeit nur noch maximal $(\frac{1}{4})^z$. Damit kann durch eine ausreichende Anzahl von Tests die Primzahleigenschaft mit beliebig kleiner Fehlerwahrscheinlichkeit festgestellt werden.

Für Algorithmen werden fünf Testläufe empfohlen, 25 Durchgänge werden als praktisch sicher eingestuft. Von den ungeraden Zahlen kleiner als $25 \cdot 10^9$ bestehen nur Primzahlen und die Zahl $3125031751 = 151 \cdot 751 \cdot 28351$ den Miller-Rabin-Test mit den Zahlen $a = 2, 3, 5$ und 7 . Primzahltests können also im Gegensatz zur Faktorisierung zusammengesetzter Zahlen sehr effizient durchgeführt werden.

¹⁵Siehe Pomerance et al. [7, Seite 14f.].

2.4.3 Pollards Methode

Der Fermat-Test und der Miller-Rabin-Test liefern zwar eine Aussage, ob die untersuchte Zahl prim oder zusammengesetzt ist, aber sie können nicht benutzt werden, um Primfaktoren zu berechnen. Dafür gibt es verschiedene Faktorisierungsalgorithmen, die je nach Beschaffenheit der zu untersuchenden Zahl verschieden gut arbeiten.

Die $(p - 1)$ -Methode von J. Pollard ist vor allem für zusammengesetzte Zahlen n geeignet, welche einen Primfaktor p besitzen, für den $p - 1$ nur kleine Primfaktoren hat. Für alle Vielfachen k von $p - 1$ gilt nach dem kleinen Satz von Fermat $a^k \equiv 1 \pmod{p}$ für alle Zahlen a , die nicht durch p teilbar sind. Das heißt, daß p ein Teiler von $a^k - 1$ ist. Nun ist $\gcd(a^k - 1, n)$ ein echter Teiler von n , falls $a^k - 1$ nicht durch n teilbar ist.

Der Algorithmus benutzt für k das Produkt aller Primzahlpotenzen, die nicht größer als eine Schranke B sind, das heißt:

$$k = \prod_{p \leq B} p^{e_p} \quad \text{mit } p \text{ Primzahl, } p^{e_p} \leq B, p^{e_p+1} > B$$

Sind alle Primzahlpotenzen, welche $p - 1$ teilen, kleiner als B , so ist k ein Vielfaches von $p - 1$. Die Berechnung von $\gcd(a^k - 1, n)$ mit einem geeigneten a sollte nun einen Teiler von n liefern. Mißlingt dies, so wird eine neue Schranke B gewählt und das Verfahren damit wiederholt.

Wenn allerdings $p - 1$ auch größere Primfaktoren besitzt, so ist Pollards Methode genauso wenig effektiv wie die Probedivision. In diesem Fall sind andere Verfahren besser geeignet.

2.4.4 Das Quadratische Sieb

Ein anderer Ansatz führt zu weiteren Faktorisierungsalgorithmen. Das Prinzip besteht darin, ganze Zahlen x und y zu bestimmen, für welche die folgenden Kongruenzen gelten:

$$\begin{aligned}x^2 &\equiv y^2 \pmod{n} \\x &\not\equiv \pm y \pmod{n}\end{aligned}$$

Dann ist n zwar Teiler von $x^2 - y^2 = (x - y)(x + y)$, aber weder ein Teiler von $(x - y)$ noch von $(x + y)$. Somit muß sowohl $\gcd(x - y, n)$ als auch $\gcd(x + y, n)$ ein echter Teiler von n sein.

Beim **Quadratischen Sieb** (QS) wird die Kongruenz $x^2 \equiv y^2 \pmod{n}$ durch Kombination von Relationen erzeugt. Eine Relation r_i ist eine Kongruenz der Form $x_i^2 \equiv y_i \pmod{n}$. Für den Algorithmus werden viele solcher Relationen gesammelt und eine Teilmenge S davon bestimmt, so daß für die Elemente folgendes gilt:

$$\prod_{r_i \in S} y_i \equiv y^2 \pmod{n}$$

Das ergibt dann die gewünschte Kongruenz:

$$x^2 \equiv \prod_{r_i \in S} x_i^2 \equiv \prod_{r_i \in S} y_i \equiv y^2 \pmod{n}$$

Das Verfahren ist dementsprechend auch in zwei Phasen aufgebaut. Zuerst werden geeignete Relationen gesammelt und danach wird eine passende Kombination dieser Relationen gesucht.

Das bisher schnellste Verfahren, welches den obigen Ansatz benutzt, ist das **Zahlkörpersieb** (Number Field Sieve, NFS), das einige Ergebnisse aus der Theorie der algebraischen Zahlkörper benutzt¹⁶.

¹⁶Siehe Lenstra et al. [6].

Kapitel 3

Kryptographische Grundlagen

3.1 Grundbegriffe

3.1.1 Kryptosysteme

Die Zielsetzung der Kryptographie ist, einen Text zum Zweck der Geheimhaltung in eine sinnfreie Zeichenkette zu überführen, so daß nur die Zielperson diese wieder in den Originaltext umzuwandeln vermag. Dazu bedarf es einiger Vereinbarungen:

Definition 3.1 (Alphabet) *Ein **Alphabet** ist eine endliche nichtleere Menge Σ . Die Elemente von Σ heißen Zeichen, Buchstaben oder Symbole von Σ . Als Länge von Σ wird die Anzahl der Elemente von Σ bezeichnet. Weiterhin müssen die Zeichen eines Alphabets so beschaffen sein, daß sich eine beliebige Folge von diesen Zeichen auf eine eindeutige Weise in ihre einzelnen Bestandteile zerlegen läßt.*

Ein Text ist eine endliche Folge von Zeichen eines festgelegten Alphabets. Für binäre Daten wie Bilder oder Musik reicht das Alphabet $\Sigma = \{0, 1\}$ aus. Einfache Sachverhalte können durch das Alphabet der 26 lateinischen Buchstaben $\Sigma = \{A, \dots, Z\}$ dargestellt werden. In den meisten Fällen wird allerdings der ASCII-Zeichensatz als Alphabet benutzt. Die Menge aller möglichen Texte $T \subseteq \Sigma^*$ heißt Textraum.

Definition 3.2 (Kryptosystem) Ein Tupel $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ wird **Kryptosystem** genannt, falls die folgenden Eigenschaften erfüllt sind:

1. $\mathcal{M} \subseteq \Sigma_1^*$ ist der **Klartextraum**, also die Menge aller möglichen **Klartexte** (messages).
2. $\mathcal{C} \subseteq \Sigma_2^*$ ist der **Chiffretextraum**, also die Menge aller möglichen **Chiffretexte** (ciphers).
3. $\mathcal{K} \subseteq \Sigma_3^*$ heißt **Schlüsselraum**. Die Elemente dieser Menge heißen **Schlüssel** (keys).
4. $\mathcal{E} = \{E_k \mid k \in \mathcal{K}\}$ ist eine Familie von Funktionen $E_k : \mathcal{M} \rightarrow \mathcal{C}$. Die Elemente heißen **Verschlüsselungsfunktionen** (encryption).
5. $\mathcal{D} = \{D_k \mid k \in \mathcal{K}\}$ ist eine Familie von Funktionen $D_k : \mathcal{C} \rightarrow \mathcal{M}$. Die Elemente heißen **Entschlüsselungsfunktionen** (decryption).
6. Für jedes $e \in \mathcal{K}$ existiert ein $d \in \mathcal{K}$, so daß für alle $m \in \mathcal{M}$ die Gleichung $D_d(E_e(m)) = m$ gilt.

3.1.2 Block- und Stromchiffren

Es gibt zwei Typen von kryptographischen Algorithmen, **Blockchiffren** und **Stromchiffren**. Blockchiffren codieren Klartextblöcke, die eine feste Länge haben, zu Chiffretextblöcken derselben Länge. Sie lassen sich durch Software gut implementieren, daher sind diese Blöcke meist 64 Bit lang. Stromchiffren dagegen codieren stets Zeichen für Zeichen mit einem gleichzeitig generierten Schlüsselstrom und lassen sich gut mittels Hardware implementieren.

Wenn eine Nachricht in Blöcke aufgeteilt wird, so ist der letzte Block meist zu kurz. In diesem Fall muß der Block vervollständigt werden. Dies wird **Padding** genannt. Eine einfache Methode dafür ist, die fehlenden Bytes bis auf das letzte mit Nullen zu füllen und die Anzahl der Nullen in das letzte Byte zu schreiben. Alternativ dazu kann auch in jedes Byte die Anzahl der hinzugefügten Bytes geschrieben werden. Entscheidend ist nur, daß der Empfänger die zusätzlichen Bytes wieder korrekt entfernen kann.

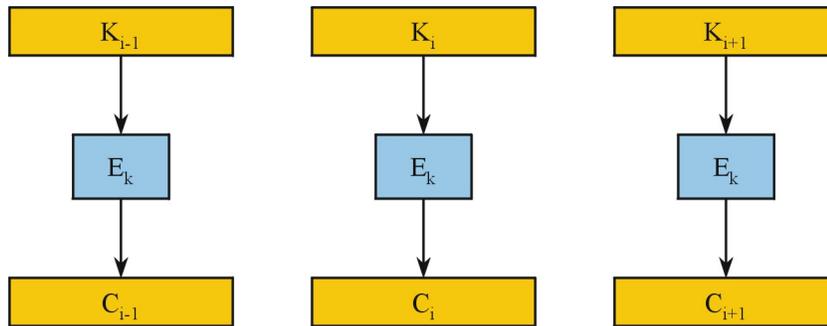


Abbildung 3.1: Electronic-Codebook-Modus

Für Blockchiffren existieren mehrere verschiedene *Betriebsmodi*. Der einfachste Betriebsmodus ist der **Electronic-Codebook-Modus** (ECB). Hier wird jeder Klartextblock einfach in einen Chiffretext überführt. Dies ermöglicht allerdings das Anfertigen eines Wörterbuches mit Klartext-Chiffretext-Paaren, denn gleiche Klartextblöcke werden in gleiche Chiffretextblöcke überführt. Weiterhin ist eine Modifikation der Nachricht leicht möglich, da nur die modifizierten Blöcke sich ändern und der Rest der Nachricht gleichbleibt. Zum Beispiel kann bei einer Überweisung der Empfänger ausgetauscht werden, vorausgesetzt, der Angreifer kennt die Position der Daten in der Nachricht und den korrekten Chiffretext seiner eigenen Bankverbindung. Dies kann durch eine Rückkopplung verhindert werden.

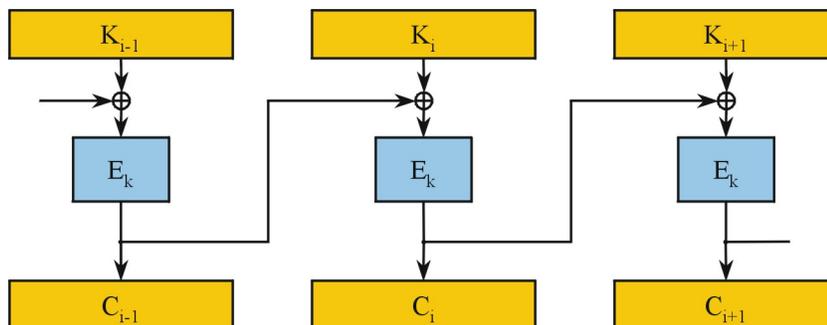


Abbildung 3.2: Cipher-Block-Chaining-Modus

Beim **Cipher-Block-Chaining-Modus** (CBC) fließt das Ergebnis der Verschlüsselung vorheriger Blöcke in die aktuelle Operation ein, indem der erhaltene Chiffretext mit dem Klartext des nächsten Blocks vor der Verschlüsselung mittels \oplus verknüpft wird.

Um auch Muster am Anfang einer Nachricht zu verschleiern, wird der erste Block mit einem sogenannten *Initialisierungsvektor* (IV) verknüpft. Dieser braucht nicht geheimgehalten zu werden, er soll die Nachricht nur eindeutig machen, so daß keine Wiederholungen möglich sind. Gut dafür geeignet sind Zeitstempel.

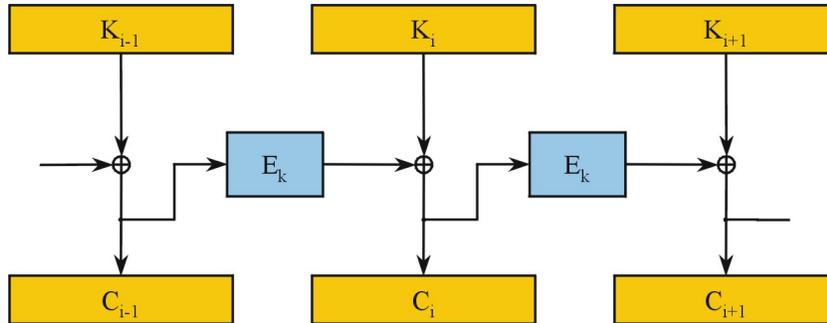


Abbildung 3.3: Cipher-Feedback-Modus

Der **Cipher-Feedback-Modus** (CFB) ist dem CBC-Modus ähnlich. Er benutzt allerdings den Chiffretext des vorherigen Blocks zur Erzeugung eines temporären Schlüssels, welcher dann mittels \oplus mit dem aktuellen Klartextblock verknüpft wird. Um den temporären Schlüssel zu erzeugen, wird der vorherige Chiffretext mit dem Originalschlüssel codiert.

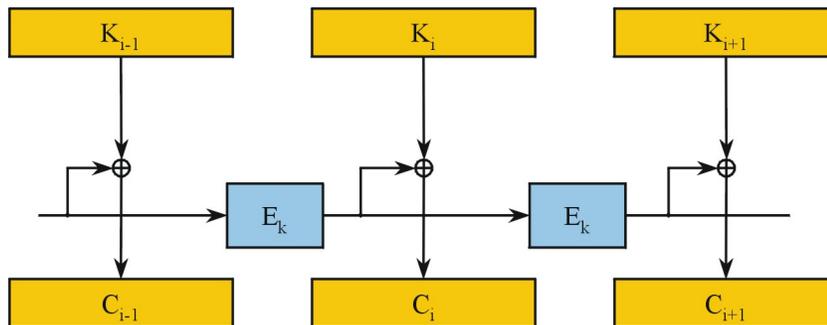


Abbildung 3.4: Output-Feedback-Modus

Der **Output-Feedback-Modus** (OFB) ist etwas einfacher als der CFB-Modus. Der Unterschied besteht darin, daß durch Verschlüsselung des gerade benutzten temporären Schlüssels mit dem Originalschlüssel ein weiterer temporärer Schlüssel erzeugt wird. Dieser wird dann genau wie im CFB-Modus mittels \oplus mit dem aktuellen Klartextblock verknüpft. Sowohl beim CFB-Modus als auch beim OFB-Modus wird für die erste Operation ein Initialisierungsvektor benötigt.

Es existieren weitere Betriebsmodi für Blockchiffren, wie etwa der Plaintext-Block-Chaining-Modus oder der Plaintext-Feedback-Modus, allerdings sind diese meist komplexer und weniger robust als die vier oben erwähnten Modi. Für die meisten Anwendungen sind diese Standardmodi vollkommen ausreichend. Da beim CFB-Modus und beim OFB-Modus der Klartext mit dem temporären Schlüssel nur mittels \oplus verknüpft wird, eignen sich diese beiden Modi auch für Stromchiffre. Bei einer **selbstsynchronisierenden Stromchiffre** ist jedes Bit des Schlüsselstroms von einer festen Anzahl vorhergehender Chiffretextbits abhängig. Dies entspricht dem CFB-Modus bei Blockchiffren. Bei der **synchronen Stromchiffre** wird der Schlüsselstrom unabhängig generiert. Dies kann wie beim OFB-Modus durch fortwährende Verschlüsselung geschehen. Auch ein komplett zufälliger Schlüsselstrom kann benutzt werden. Das hat den Vorteil, daß das Verfahren absolut sicher ist, allerdings auch den Nachteil, daß Schlüssel und Nachricht gleich lang sind.

3.2 Symmetrische Kryptosysteme

3.2.1 Cäsar-Chiffre und One-Time-Pad

Ist in einem Kryptosystem der Schlüssel der Verschlüsselungsfunktion mit dem der entsprechenden Entschlüsselungsfunktion identisch oder leicht zu berechnen, so wird von einem **symmetrischen Kryptosystem** gesprochen.

Ein einfaches Beispiel dafür ist das **One-Time-Pad**. Der Schlüssel ist in diesem System sehr lang, und zwar genauso lang wie die zu verschlüsselnde Nachricht. Zum Chiffrieren muß nur jedes Zeichen der Nachricht mit dem entsprechenden Zeichen des Schlüssels mittels \oplus verknüpft werden. Ist der Schlüssel echt zufällig gewählt worden, so widersteht die Verschlüsselung allen Angriffen. Der Empfänger der Nachricht muß zur Entschlüsselung nur die Umkehrfunktion mit demselben Schlüssel anwenden.

Es ist bewiesen worden, daß die Kommunikation absolut sicher ist, solange der gemeinsame Schlüssel geheim bleibt. Wie der Name schon sagt, kann der Schlüssel nur einmal benutzt werden, denn ab dem zweiten Mal sind statistische Tests möglich.

Eine sehr einfache Variante dieses Systems hat auch schon Julius Cäsar benutzt. Beim sogenannten **Cäsar-Chiffre** wird als Schlüssel eine Zahl zwischen 0 und 25 gewählt. Die Buchstaben werden der Reihe nach numeriert, beginnend mit 0 für den Buchstaben A. Die Verschlüsselung eines Textes verläuft folgendermaßen: Im Chiffretext wird jeweils der Buchstabe geschrieben, dessen Nummer sich ergibt, wenn die Nummer des zugehörigen Buchstabens aus dem Klartext mit dem Schlüssel modulo 26 addiert wird.

Ist der Schlüssel also zum Beispiel 7, so werden folgende Ersetzungen durchgeführt: $A \rightarrow H$, $B \rightarrow I$, \dots , $Z \rightarrow G$. Aus „KRYPTOGRAPHIE“ wird dann „RYFWAVNYHWOPL“. Zur Entschlüsselung muß nun diese Ersetzung mit demselben Schlüssel einfach umgedreht werden. Der Cäsar-Chiffre ist allerdings im Gegensatz zum One-Time-Pad nicht sehr sicher, denn das Geheimnis ist sehr einfach zu erraten. Schon durch einfaches Ausprobieren aller 26 möglichen Schlüssel läßt sich der Originaltext wiederherstellen.

3.2.2 Der DES-Algorithmus

Das wohl bekannteste symmetrische Verfahren ist der **Data Encryption Standard**, kurz DES. Es arbeitet mit einer Blocklänge von 64 Bit über dem Binäralphabet, für Klar- und Chiffretextraum gilt $\mathcal{M} = \mathcal{C} = \{0, 1\}^{64}$. Der dazugehörige Schlüsselraum ist eine Teilmenge von $\{0, 1\}^{64}$, da hier einige Zeichen nicht frei wählbar sind. Wird der Schlüssel in Blöcke zu je acht Bits aufgeteilt, dann wird das jeweils letzte Bit so gesetzt, daß die Quersumme des Blockes ungerade ist. Insgesamt gibt es somit 2^{56} verschiedene Schlüssel zur Auswahl.

Um einen Text zu verschlüsseln, muß er vorher in Binärdarstellung umgewandelt werden. Dies kann zum Beispiel bewerkstelligt werden, indem die Nummer des Zeichens im ASCII-Zeichensatz (eine Zahl zwischen 0 und 255) als Binärzahl mit 8 Stellen geschrieben wird. Danach wird diese Folge von Binärzahlen in Blöcke zu je 64 Bit aufgeteilt.

Die Verschlüsselung des DES-Verfahrens funktioniert wie folgt¹⁷: Aus dem 56-Bit-Schlüssel werden 16 Rundenschlüssel k_1, \dots, k_{16} mit je 48 Bit Länge erzeugt. Auf den Klartext wird zunächst eine *initiale Permutation IP* angewendet. Danach folgen 16 Runden SR_i interner Operationen.

¹⁷Hier wird nur ein allgemeiner Überblick gegeben. Die genaue Implementierung ist bei Buchmann [2, Seite 95ff.] bzw. bei Schneier [9, Seite 315ff.] beschrieben.

Eine solche Runde besteht aus mehreren Schritten:

1. Die 64-Bit Eingabe a wird in zwei 32-Bit Blöcke L_{i-1} und R_{i-1} geteilt, also $a = (L_{i-1}, R_{i-1})$.
2. Der rechte Eingabeblock R_{i-1} wird direkt als linker Ausgabeblock L_i ausgegeben.
3. Mittels einer *Expansionsfunktion* E wird der rechte Eingabeblock außerdem auf 48 Bit verlängert.
4. Diese 48 Bit werden mittels \oplus mit dem Rundenschlüssel k_i verknüpft.
5. Das Ergebnis wird in 8 Blöcke zu je 6 Bit aufgeteilt.
6. Mit acht sogenannten *S-Boxen* werden nun daraus acht 4-Bit-Blöcke berechnet.
7. Die 4-Bit-Blöcke werden wieder zu einem 32-Bit-Block zusammengefügt, auf welchen noch eine *Permutationsfunktion* P angewendet wird.
8. Der so erhaltene Block wird mittels \oplus mit dem linken Eingabeblock L_{i-1} verknüpft. Das Ergebnis wird als rechter Ausgabeblock R_i ausgegeben.

Wird die Operation mit den S-Boxen als Funktion S bezeichnet, so ergibt sich letztendlich:

$$(L_i, R_i) = SR_i((L_{i-1}, R_{i-1}), k_i) = (R_{i-1}, L_{i-1} \oplus P(S(E(R_{i-1}) \oplus k_i)))$$

Nach diesen 16 Runden schließlich werden durch eine Funktion EX die linken 32 Bit mit den rechten 32 Bit vertauscht und die *inverse initiale Permutation* IP^{-1} darauf angewendet. Insgesamt läßt sich die DES-Verschlüsselung also folgendermaßen ausdrücken:

$$E_k(a) = IP^{-1}(EX(SR_{16}(SR_{15}(\dots(SR_1(IP(a), k_1) \dots), k_{15}), k_{16})))$$

Zur Entschlüsselung muß jede dieser Funktionen invertiert werden. Die Funktionen IP und IP^{-1} sind zueinander invers, die Funktion EX zu sich selbst. Für die einzelnen Runden gilt $SR_i^{-1}(a, k_i) = EX(SR_i(EX(a), k_i))$. Auf die Gesamtfunktion angewendet ergibt sich für die Entschlüsselung demnach:

$$D_k(b) = IP^{-1}(EX(SR_1(SR_2(\dots(SR_{16}(IP(b), k_{16}) \dots), k_2), k_1)))$$

Durch Einsetzen läßt sich feststellen, daß $D_k(E_k(a)) = a$ gilt. Für die Entschlüsselung können also dieselben Funktionen mit denselben Schlüsseln wie für die Verschlüsselung verwendet werden. Der einzige Unterschied ist die Reihenfolge der Runden, welche genau umgekehrt abgearbeitet werden.

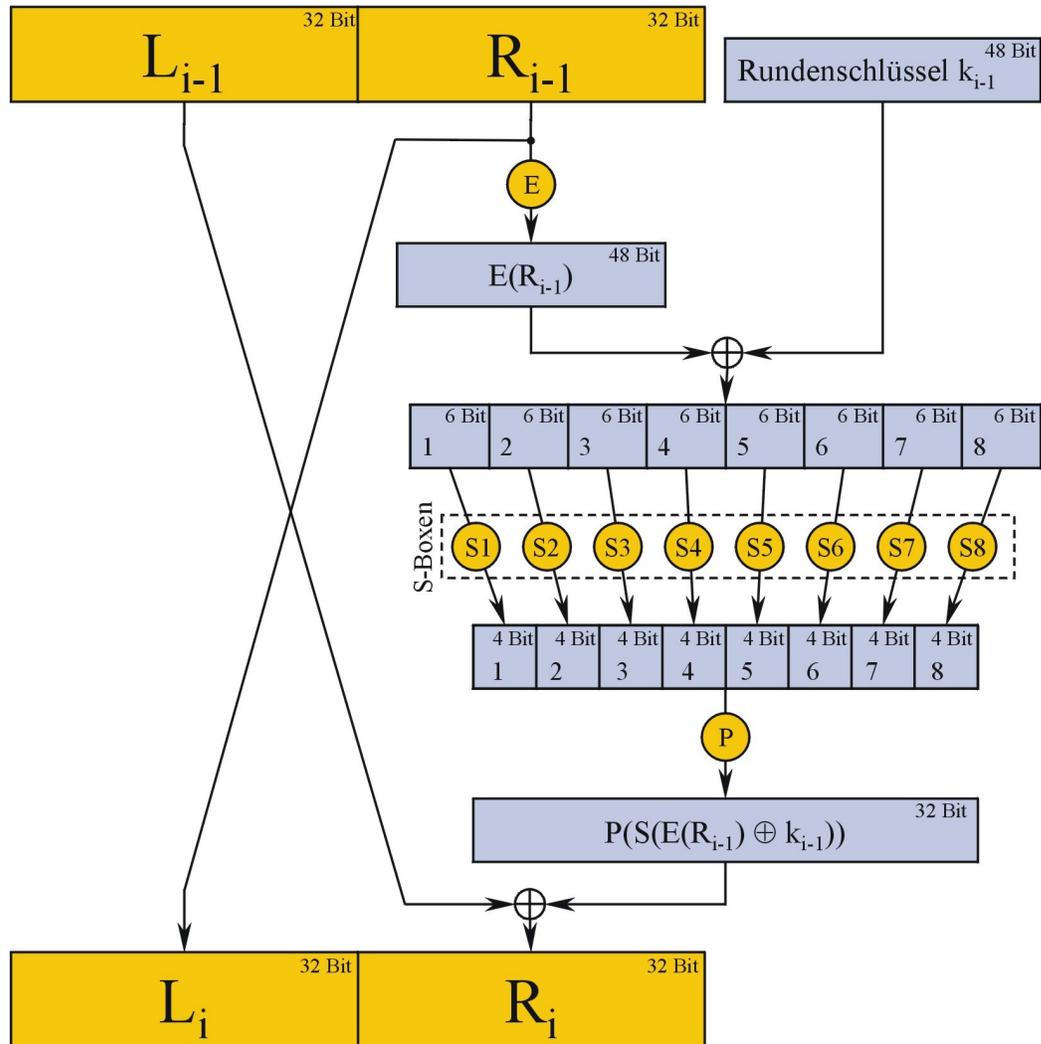


Abbildung 3.5: Schema einer Runde beim DES-Algorithmus

Der DES-Algorithmus bietet eine bessere Sicherheit gegen Angriffe als Algorithmen, welche auf linearen Transformationen basieren. Der Grund dafür sind die S-Boxen. Diese bilden die 48 Bit Eingangsdaten nichtlinear auf 32 Bit Ausgangsdaten ab. Die restlichen Operationen P , E , IP , IP^{-1} und EX lassen sich als lineare Transformationen deuten. Insgesamt kann also der DES-Algorithmus als nichtlineare Transformation von 120 Bit Eingangsdaten (Schlüssel + Eingabetext) in 64 Bit Ausgangsdaten aufgefaßt werden.

Ein Angreifer, der sowohl den Klartext als auch den zugehörigen Chiffretext kennt, muß die fehlenden 56 Bit berechnen, um an den geheimen Schlüssel zu kommen. Dazu muß er ein nichtlineares Gleichungssystem in 56 Variablen modulo 2 lösen. Dies ist ein NP-vollständiges Problem (siehe Kapitel 2.2.3), dementsprechend ist die Berechnung des Schlüssels aus heutiger Sicht nicht in polynomialer Laufzeit möglich.

Es gibt beim DES-Algorithmus vier sogenannte schwache Schlüssel mit der Eigenschaft $D_k(D_k(a)) = a$, das heißt, eine doppelte Verschlüsselung liefert wieder den Originaltext. Weiterhin gibt es zwölf sogenannte semi-schwache Schlüssel, die paarweise die Eigenschaft $D_{k_1}(D_{k_2}(a)) = a$ erfüllen. Die Wahrscheinlichkeit, einen solchen Schlüssel zufällig zu erzeugen, ist sehr gering, nämlich $(\frac{1}{2})^{54}$. Bei der Erzeugung neuer Schlüssel sollten diese trotzdem auf obige Eigenschaften geprüft werden.

3.2.3 Weitere Algorithmen

Der DES-Algorithmus wurde 1976 in den USA als Bundesstandard anerkannt und 1980 veröffentlicht. Wegen seiner kurzen Schlüssellänge von nur 56 Bit und seiner großen Verbreitung wird DES derzeit als nicht mehr sicher angesehen. Es wird vermutet, daß die National Security Agency (NSA) bereits in der Lage ist, mittels spezieller Rechner DES-Verschlüsselungen innerhalb von 15 Minuten zu knacken.

Aus diesem Grund sind viele Variationen des Algorithmus erarbeitet und erprobt worden. Einige Ansätze haben sich dabei als besonders effektiv erwiesen. Die S-Boxen können von einem zusätzlichen Schlüssel abhängig gemacht werden, allerdings müssen sie dabei geschickt erzeugt werden, sonst wird eher das Gegenteil erreicht. Auch das sogenannte *Whitening* erhöht die Sicherheit. Dabei wird der Klartext vor der Verschlüsselung blockweise mit einem weiteren 64-Bit-Schlüssel mittels \oplus verknüpft. Am Ende wird der Chiffretext noch mit einer Bitfolge verknüpft, die aus sämtlichen 120 Schlüsselbits besteht.

Die am häufigsten eingesetzte Modifikation ist die Mehrfachverschlüsselung. Das **Triple-DES-Verfahren**, welches dies leistet, erhöht die Länge des Schlüssels auf 112 Bit, auch eine Länge von 168 Bit ist möglich. Besonders interessant ist der Umstand, daß für Triple-DES die einfache DES-Implementierung ausreicht.

Die Funktionsweise von Triple-DES ist einfach: Zuerst wird der Klartext mittels DES mit den ersten 56 Bit des Schlüssels chiffriert. Danach wird der erhaltene Text mit den zweiten 56 Bit des Schlüssels dechiffriert. In der 168-Bit-Variante wird das Ergebnis zum Schluß mit den letzten 56 Bit verschlüsselt, in der 112-Bit-Variante nochmals mit den ersten 56 Bit.

Es sind noch viele weitere symmetrische Blockalgorithmen entwickelt worden. Der in Zürich entwickelte Algorithmus **IDEA** hat eine Schlüssellänge von 128 Bit, der von der NSA noch geheimgehaltene Algorithmus **Skipjack** benutzt Schlüssel der Länge 80 Bit. Beide chiffrieren 64-Bit-Blöcke und gelten als sicherer als DES. Da aber beide patentrechtlich geschützt sind, ist die Verbreitung eher gering.

Weitere bekannte Algorithmen sind die von Ron Rivest entwickelten **RC2** und **RC5**, **Blowfish** von Bruce Schneier¹⁸ sowie der aus Australien stammende **LOKI91**. Die beiden letzten Verfahren haben den Vorteil, daß sie frei verfügbar und nicht patentiert sind.

Die Suche nach einem Nachfolger für DES, dem Advanced Encryption Standard (AES), hat die Entwicklung neuer Algorithmen noch forciert. So entstanden **LOKI97**, **RC6** und **Twofish** als Nachfolger der oben genannten Verfahren. Weitere Neuentwicklungen sind in Kapitel 6.1.1 erwähnt.

Außer den Blockchiffren sind auch einige Stromchiffren entwickelt worden, die bekanntesten sind **RC4** von Ron Rivest und **SEAL** von IBM. Sie sind schnell und lassen sich leicht implementieren. Trotzdem sind sie kaum in kommerziellen Anwendungen zu finden. Immerhin benutzt das IP Security Protocol IPSEC, das zum Schutz der TCP/IP-Kommunikation eingesetzt wird, RC4 als Alternative zu DES und Triple-DES.

¹⁸Siehe Schneier [9, Seite 363ff.].

3.3 Asymmetrische Kryptosysteme

3.3.1 Einführende Bemerkungen

Ein Kryptosystem heißt **asymmetrisch**, wenn sich der Schlüssel der Verschlüsselungsfunktion von dem der zugehörigen Entschlüsselungsfunktion unterscheidet und sich keiner in polynomialer Laufzeit aus dem anderem berechnen läßt. Solche Verfahren werden auch als Public-Key-Verfahren bezeichnet. Symmetrische Verfahren haben den Nachteil, daß für jeden Kommunikationspartner ein gemeinsamer Schlüssel erzeugt werden muß. Dieser muß geheimgehalten werden und darf nicht für die Kommunikation mit anderen Parteien benutzt werden. Deswegen müssen mehrere Schlüssel sicher verwahrt werden.

Das ist bei asymmetrischen Kryptosystemen anders, hier muß nur der eigene private Schlüssel geheim bleiben. Mit dem öffentlichen Schlüssel kann jeder eine Nachricht verschlüsseln, die nur mit dem privaten Schlüssel wieder entschlüsselt werden kann. Dadurch wird auch nur ein Schlüsselpaar pro Teilnehmer benötigt. Zum Vergleich: Wollen 100 Leute mittels DES miteinander kommunizieren, so sind 4950 Schlüsselpaare notwendig, bei Public-Key-Verfahren nur 100. Andererseits sind die Schlüssel länger und die Algorithmen um ein Vielfaches langsamer.

Der erste Algorithmus dieser Art, der **Rucksack-Algorithmus** von Merkle und Hellman, wurde 1976 vorgestellt. Trotz sehr vieler Variationen und Verbesserungen wird dieser Algorithmus als unsicher angesehen, seit Shamir zeigen konnte, daß er unter gewissen Umständen geknackt werden kann. Später fand er noch weitere Schwachpunkte, so daß alle Verfahren, die darauf beruhen, kaum noch eingesetzt werden. Trotzdem wird auch in dieser Richtung weiter geforscht, um eine Alternative zu den anderen Verfahren zu haben, sollten diese ebenfalls Schwachstellen aufweisen. Als Beispiel läßt sich der **Chor-Rivest-Rucksack** anführen, der zwar kompliziert ist, aber bisher noch allen Angriffe standhalten konnte¹⁹.

¹⁹Siehe Pomerance et al. [7, Seite 86f.].

3.3.2 Der RSA-Algorithmus

Kurze Zeit nach dem Rucksack-Algorithmus veröffentlichten Rivest, Shamir und Adelman den nach ihnen benannten RSA-Algorithmus. Die Sicherheit wird in diesem Kryptosystem dadurch gewährleistet, daß es schwierig ist, große Zahlen zu faktorisieren (siehe Kapitel 2.2.1). Im folgenden wird die Funktionsweise des RSA-Verfahrens beschrieben²⁰.

Zuerst muß ein Schlüsselpaar erzeugt werden. Dazu werden zwei ausreichend große Primzahlen p und q zufällig gewählt. Dies kann zum Beispiel mit einem Zufallszahlengenerator und dem Miller-Rabin-Test geschehen. Danach werden die Produkte $n = pq$ und $\varphi(n) = (p - 1)(q - 1)$ berechnet. Anschließend wird eine natürliche Zahl e gewählt, die zu $\varphi(n)$ relativ prim ist. Die gebräuchlichsten Werte für e sind 17 und 65537. Mit diesen Zahlen erfolgt die Verschlüsselung wesentlich schneller, da Computer die Berechnung damit besonders effizient durchführen können.

Früher wurde auch häufig der Wert 3 verwendet, aber er gilt als unsicher, seit es einen Angriff gibt, der lineare Abhängigkeiten zwischen Nachrichten ausnutzt. Diese spezielle Attacke kann erfolgreich durchgeführt werden, falls mindestens e identische Nachrichten vorliegen. Die Sicherheit kann aber gewährleistet bleiben, indem die Nachricht mit zufälligen Werten aufgefüllt wird. Das verhindert das Auftauchen identischer Klartexte.

Mit Hilfe des erweiterten Euklidischen Algorithmus wird nun die Zahl d berechnet, welche

$$ed \equiv 1 \pmod{\varphi(n)}$$

erfüllt. Diese Zahl ist eindeutig bestimmt. Die Zahl d bildet den privaten Schlüssel, die Zahlen e und n den öffentlichen Schlüssel. Die Primzahlen p und q werden nun nicht mehr benötigt, sie sollten aber trotzdem nicht bekanntgegeben werden, denn durch sie ist die Rekonstruktion der Schlüssel möglich. Die Verschlüsselungsformel lautet:

$$c = E(m) = m^e \pmod{n}$$

Die Zahl m muß dabei kleiner als n sein, daher wird die gesamte Nachricht in numerische Blöcke so aufgeteilt, daß jeder Block einzeln verschlüsselt werden kann. RSA kann also als Blockchiffre aufgefaßt werden. Es kann sowohl der ECB-Modus als auch der CBC-Modus darauf angewendet werden.

²⁰Eine ausführliche Beschreibung von RSA ist bei Buchmann [2, Seite 115ff.] bzw. bei Schneier [9, Seite 531ff.] zu finden.

Auch das Entschlüsseln ist einfach, dazu wird der verschlüsselte Block c genommen und der Originalblock mit der folgenden Formel berechnet:

$$m = D(c) = c^d \pmod{n}$$

Der Satz von Euler-Fermat bestätigt die Korrektheit des Algorithmus, denn e und d wurden so gewählt, daß folgende Gleichung für ein $k \in \mathbb{N}$ gilt:

$$ed = k\varphi(n) + 1$$

Damit gilt für die Kombination von Ver- und Entschlüsselung:

$$\begin{aligned} D(E(m)) &= (m^e)^d \pmod{n} \\ &= m^{k\varphi(n)+1} \pmod{n} \\ &= m \cdot m^{k\varphi(n)} \pmod{n} \\ &= m \end{aligned}$$

Die Entschlüsselung ist nur demjenigen möglich, der den Schlüssel d kennt. Solange d , $\varphi(n)$, p und q geheim bleiben, ist nach heutigem Erkenntnisstand kein anderer als der rechtmäßige Empfänger in der Lage, den Originaltext in polynomialer Zeit wiederherzustellen.

3.3.3 Weitere Verfahren

Im Grunde liegt jedem asymmetrischen Kryptosystem eine Einwegfunktion zugrunde. Das **Rabin-Verfahren** etwa benutzt ein zur Faktorisierung äquivalentes Problem, die Berechnung von Quadratwurzeln modulo einer zusammengesetzten Zahl.

Die Sicherheit des **ElGamal-Verfahrens** beruht auf der Schwierigkeit, diskrete Logarithmen über einem endlichen Körper zu berechnen (siehe Kapitel 2.2.2). Allerdings ist ElGamal etwas langsamer als RSA²¹, da der Chiffretext hier doppelt so lang wie der Klartext ist und außerdem zwei modulare Exponentationen benötigt werden. Andererseits lassen sich einige Werte vorberechnen und die Verschlüsselung wird dadurch etwas schneller.

Ein weiterer Vorteil von ElGamal ist, daß hier ein zufälliges Element dazukommt, denn der Sender einer Nachricht kann eine Zufallszahl wählen, die in die Verschlüsselung mit einbezogen wird²². ElGamal kann auch in Verbindung mit elliptischen Kurven benutzt werden (siehe Kapitel 6.2.1), daher ist dieses Verfahren eine wichtige Alternative zu RSA.

²¹Siehe auch Schneier [9, Seite 541ff.] und Buchmann [2, Seite 125ff.].

²²Für Details siehe Buchmann [2, Seite 133ff.].

3.4 Hashfunktionen

3.4.1 SHA

Eine kryptographische **Hashfunktion** ist eine Funktion, die einen variablen Eingabewert in einen kurzen Wert fester Länge, den Hashwert, umwandelt. Diese Berechnung darf nicht in polynomialer Zeit umgekehrt werden können. Idealerweise ist eine Hashfunktion weitgehend kollisionsfrei, das heißt, es gibt kaum zwei Originaltexte, die denselben Hashwert besitzen. Weiterhin beeinflussen geringe Änderungen der Eingabe bei einer guten Hashfunktion die Ausgabe sehr stark.

Ein sehr häufig eingesetzter Algorithmus ist der vom National Institute of Standards and Technology (NIST) und der NSA entwickelte **Secure Hash Algorithm** (SHA). SHA benutzt nichtlineare Transformationen, um die Eigenschaften einer Hashfunktion möglichst gut zu erfüllen²³. Die Nachricht wird erst auf ein Vielfaches von 512 Bit aufgefüllt. Danach wird eine Schleife für jeden 512-Bit-Block der Nachricht durchlaufen. In jeder solchen Runde werden fünf 32-Bit-Variablen anhand der Daten verändert. Nach der letzten Schleife bilden diese Variablen hintereinander geschrieben den Hashwert. Dieser ist $5 \cdot 32 = 160$ Bit groß.

3.4.2 MD4 und seine Varianten

Ron Rivest entwarf eine Hashfunktion namens Message Digest (MD) und entwickelte diese anhand auftretender Schwachstellen weiter. Die vierte Version **MD4**, welche die Basis für SHA bildet, entwarf er 1991. Bei diesem Algorithmus verändern sich Variablen mittels nichtlinearer Transformationen anhand des Eingabetextes und bilden dadurch einen eindeutigen Hashwert. MD4 benutzt vier Variablen, die in drei Runden in die Berechnung eingehen, daher ist der Hashwert nur 128 Bit lang.

Schon bald nach der Veröffentlichung gab es einige erfolgreiche Attacken gegen MD4, so daß recht kurz darauf **MD5** entwickelt wurde, der diesen Angriffen besser standhält²³. Verbesserungen waren zum Beispiel der Austausch der schwachen zweiten Runde der Berechnung und das Hinzufügen einer vierten Runde. Es sind zwar noch kleinere Angriffspunkte entdeckt worden, sie sollen aber keine relevanten praktischen Auswirkungen haben.

²³SHA und MD5 sind bei Schneier [9, Seite 498ff.] ausführlich beschrieben.

3.4.3 RIPEMD-160

Für das Projekt RIPE (RACE Integrity Primitives Evaluation²⁴) der Europäischen Union wurde **RIPEMD**, ebenfalls eine Variante von MD4, entwickelt. Die größte Veränderung gegenüber MD4 ist, daß hier zwei Versionen des Algorithmus parallel laufen, welche sich nur durch die Wahl einiger Konstanten unterscheiden. Beide Ergebnisse werden zu den vier Verkettungsvariablen addiert. Dadurch scheint RIPEMD resistent gegen differentielle und lineare Kryptanalyse zu werden.

Als die Angriffspunkte gegen MD5 bekannt wurden und einige Experten meinten, daß 128 Bit zuwenig seien, wurde **RIPEMD-160** von Dobbertin, Bosselaers und Preneel als Nachfolger von RIPEMD entwickelt²⁵. Diese Hashfunktion benutzt fünf Verkettungsvariablen statt der bisherigen vier und insgesamt fünf Runden für die Berechnung. Aus Kompatibilitätsgründen wurde mit RIPEMD-128 auch eine Hashfunktion mit vier Verkettungsvariablen und vier Runden bereitgestellt²⁶.

3.4.4 MDC-2

Im Jahre 1988 entwickelte IBM Hashfunktionen auf Basis von Blockchiffren, welche Modification Detection Codes (MDC) genannt wurden. Nach der Anzahl der verwendeten Blockchiffre-Operationen werden sie mit **MDC-2** und **MDC-4** bezeichnet. Theoretisch funktionieren diese Hashfunktionen mit beliebigen Blockchiffren, sie werden aber meistens nur in Verbindung mit DES angewendet.

MDC-2 benutzt zwei Verkettungsvariablen, die in jedem Durchlauf als Schlüssel für die Blockchiffrierung dienen²⁶. Anhand der zu verarbeitenden Daten werden diese Variablen ständig verändert. Nach der Abarbeitung des letzten Blocks bilden die Verkettungsvariablen hintereinandergeschrieben den Hashwert. MDC-4 unterscheidet sich von MDC-2 nur dadurch, daß auf das Ergebnis die gesamte Operation ein weiteres Mal mit vertauschten Rollen angewendet wird.

²⁴Die Abkürzung RACE steht für Research and Development in Advanced Communication Technologies.

²⁵Siehe <http://www.esat.kuleuven.ac.be/~bosselae> [32].

²⁶Die Hashfunktionen RIPEMD-160 und MDC-2 sind bei Menezes [21, Seite 349ff.] und im ISO-Standard 10118-2 [29] beschrieben.

3.4.5 Message Authentication Codes

Ein Message Authentication Code (MAC) ist eine kryptographische Hashfunktion, die von einem Schlüssel abhängig ist. Dadurch ist eine Verifikation des Hashwertes nur mit Kenntnis des Schlüssels möglich. Dies kann genutzt werden, um eigene Dateien vor Manipulationen zu schützen.

Eine häufig benutzte Konstruktion dazu ist der **CBC-MAC**. Hier wird die Nachricht mit einem symmetrischen Verschlüsselungsverfahren im CBC-Modus chiffriert. Der letzte Ausgabeblock bildet den Hashwert.

3.5 Digitale Signaturen

3.5.1 RSA-Signaturen

Eine digitale Signatur ist eine elektronische Unterschrift. Sie dient zum Beweis, daß die Nachricht wirklich vom Signierenden stammt. Sie muß fälschungssicher und verifizierbar sein. Nach dem Signieren darf die Nachricht nicht mehr geändert werden können.

Mit dem **RSA-Verfahren** ist es möglich, digitale Signaturen zu erstellen. Dazu werden die Rollen der Schlüssel einfach vertauscht. Ein Dokument wird mit dem privaten Schlüssel verschlüsselt, so daß jeder mit dem öffentlichen Schlüssel die Originalnachricht wieder herstellen kann. Da nur der Ersteller der Nachricht den privaten Schlüssel kennt, ist sichergestellt, daß das Dokument auch von ihm stammt.

Es gibt noch einige Angriffsmöglichkeiten auf diese Signaturen. Das kann verhindert werden, indem zuerst mit einer kryptographischen Hashfunktion der Hashwert des Dokuments berechnet wird. Dieser wird signiert und zusammen mit dem Originaldokument versandt. Der Empfänger kann den Hashwert berechnen. Stimmt dieser mit dem Wert der entschlüsselten Signatur überein, so sieht er die Nachricht als echt an.

3.5.2 ElGamal-Signaturen und DSA

Auch das **ElGamal-Verfahren** kann zum Signieren einer Nachricht verwendet werden. Da die Verschlüsselung hier mit der Entschlüsselung nicht einfach vertauschbar ist, muß das Signatur-Verfahren entsprechend anders konstruiert werden. Auch hier ist der Einsatz einer Hashfunktion erforderlich, um die nötige Sicherheit herzustellen. Eine effiziente Variante des ElGamal-Signaturverfahrens ist der **Digital Signature Algorithm** (DSA).

3.6 Kryptographische Protokolle

3.6.1 Festcodes und Wechselcodes

Ein **kryptographisches Protokoll** ist eine Folge von Aktionen, an denen mehrere Personen oder Organisationen beteiligt sind. Für viele Zwecke sind verschiedene Protokolle entwickelt worden. Ein Beispiel: Die Identifikation ist die eindeutige Bestimmung der Identität einer Person, die Authentisierung dient dem Beweis der Identität. Beides kann mit Protokollen geregelt werden.

Von Geldautomaten sind Paßwortverfahren bekannt, die sogenannten **Festcodes**. Das ist ein sehr einfaches Protokoll. Es beruht darauf, daß jeder Teilnehmer ein Geheimnis besitzt, welches niemand außer einer zentralen Überprüfungsstelle wissen kann. Derjenige, der sich authentisieren soll, gibt zuerst seine Identität an. Beim Geldautomaten wird dies durch das Einstecken der ec-Karte geregelt. Danach wird das persönliche Geheimnis, in unserem Beispiel die PIN, abgefragt. Der Zentralrechner überprüft, ob in seiner Liste unter der Identität dasselbe Geheimnis steht. Erst wenn das der Fall ist, erlaubt er weitere Zugriffe.

Die Sicherheit der Liste auf dem Zentralrechner muß stets gewährleistet sein. Kann jemand Zugriff auf sie bekommen, so erlangt derjenige auch Kenntnis über die Geheimnisse aller Teilnehmer. Deswegen wird oft nur eine Liste von Hashwerten der Paßwörter gespeichert. Bei der Verifikation wird der Hashwert des eingegebenen Paßwortes mit dem gespeicherten Wert verglichen. Ein Angreifer kann aber aus der Liste der Werte kein Paßwort rekonstruieren.

Ein weiteres einfaches Protokoll ist bei einigen Varianten des Homebankings zu finden. **Wechselcodes** funktionieren genauso wie Festcodes, nur daß das Geheimnis sich ständig ändert. Das entspricht zum Beispiel den Transaktionsnummern (TAN) beim Homebanking. Für jeden Vorgang (jede Transaktion) wird hier eine neue TAN benutzt. Eine Liste der gültigen TANs muß der Teilnehmer schon vorher von seiner Bank erhalten haben. Diese TAN wird mit einem festen Geheimnis (der PIN) zusammen übermittelt, damit die Authentisierung erfolgen kann. Danach ist die TAN nicht mehr gültig.

Beide Protokolle sind unidirektional, daß heißt, die relevanten Informationen werden nur in eine Richtung übertragen. Außerdem muß das Geheimnis vorher festgelegt werden. Wenn ein Angreifer dieses Gespräch belauschen kann, so ist die Sicherheit natürlich auch nicht mehr gewährleistet.

3.6.2 Bidirektionale Protokolle

Die bidirektionale Variante der Festcodes heißt **Challenge-and-Response**. Die Idee hierbei ist, daß nicht ein festes Geheimnis abgefragt wird, sondern bei jeder Authentisierung ein neues. Dazu muß eine Partei eine Frage erfinden, die nur die andere Partei beantworten kann. Das kann zum Beispiel etwas aus dem persönlichen Umfeld sein.

Eine Variante davon ist das folgende Verfahren: Beide Seiten vereinbaren vorher einen gemeinsamen Schlüssel und eine Einwegfunktion, die von diesem Schlüssel abhängig ist. Ist einmal eine Authentisierung notwendig, so wählt der Fragende eine Zufallszahl (die Challenge) und sendet diese an den Partner. Dieser wendet die Einwegfunktion auf die Zufallszahl an und schickt das Ergebnis (die Response) zurück. Der Fragende berechnet den Wert genauso und überprüft sein Ergebnis mit dem erhaltenen Wert. Sind sie gleich, so war die Authentisierung erfolgreich.

Beim Challenge-and-Response Verfahren besteht immer noch das Problem, daß ein gemeinsamer Schlüssel auf einem unsicheren Wege vorher ausgetauscht werden muß. Dies ist beim **Diffie-Hellman-Schlüsselaustausch** nicht mehr der Fall. Wollen zwei Teilnehmer A und B einen gemeinsamen Schlüssel austauschen, so arbeiten sie das folgende Verfahren ab:

1. Beide Parteien wählen gemeinsam eine Primzahl p und eine Zahl g mit $2 \leq g \leq p - 2$. Dabei muß g eine primitive Wurzel von p sein. Die Wahl dieser beiden Zahlen kann öffentlich geschehen, Geheimhaltung ist nicht notwendig.
2. A wählt eine geheime Zahl x , berechnet $X = g^x \pmod{p}$ und schickt die Zahl X an B.
3. B wählt eine geheime Zahl y , berechnet $Y = g^y \pmod{p}$ und schickt die Zahl Y an A.
4. A berechnet $k_1 = Y^x \pmod{p}$.
5. B berechnet $k_2 = X^y \pmod{p}$.

k_1 und k_2 sind gleich $g^{xy} \pmod{p}$ und können als gemeinsamer Schlüssel verwendet werden. Ein Angreifer, der weder x noch y kennt, muß den diskreten Logarithmus berechnen. Das ist derzeit nicht in polynomialer Zeit möglich, es sollte ihm sehr schwerfallen.

3.6.3 Weitere Protokolle

Das **Interlock-Protokoll** ist ein Protokoll, welches die Kommunikation zwischen zwei Parteien vor dem Man-In-The-Middle-Angriff (siehe Kapitel 4.2.2) schützt. Der wesentliche Punkt dabei ist, daß es dem Angreifer erschwert wird, die Nachricht manipulieren zu können. Damit das Protokoll funktioniert, müssen beide Parteien eine Nachricht vorliegen haben, welche sie verschicken wollen.

Diese Nachricht wird mit dem öffentlichen Schlüssel des anderen chiffriert. Nun sendet eine Partei die Hälfte ihrer verschlüsselten Nachricht. Damit kann der Angreifer nichts anfangen, denn zum Entschlüsseln benötigt er die gesamte Nachricht. Die andere Partei verschickt nach Erhalt der Nachrichtenhälfte seine erste Hälfte. Auch hier hat der Angreifer keine Chance, Veränderungen vorzunehmen. Nun kann die erste Partei den Rest ihrer Nachricht schicken und erhält als Antwort den Rest der anderen Nachricht. Nun erst können alle Beteiligten die Nachrichten entschlüsseln. Für den Angreifer ist es aber nun zu spät, er kann nichts mehr ausrichten.

Eine Möglichkeit, eine geheime Nachricht auszutauschen, ohne vorher einen Schlüssel zu vereinbaren, liefert das **No-Key-Protokoll** von Shamir. Dabei wird die Nachricht von der ersten Partei verschlüsselt und versandt. Die andere Partei kann die Nachricht zwar nicht lesen, aber ein weiteres Mal verschlüsseln und zurücksenden. Die erste Partei entfernt ihre eigene Verschlüsselung und sendet die Nachricht ein zweites Mal. Der Empfänger kann die Nachricht nun lesen, indem er seine Verschlüsselung rückgängig macht. Die Parteien sollten sich dabei allerdings sicher sein, daß wirklich der Partner die Nachricht verschlüsselt hat und nicht etwa ein Angreifer.

Bei der Implementierung stellt sich das Problem, daß die benutzte Verschlüsselungsfunktion symmetrisch sein muß. Das heißt, es muß die Gleichung $E_{k_1}(E_{k_2}(m)) = E_{k_2}(E_{k_1}(m))$ gelten. Dazu kann zum Beispiel die diskrete Exponentialfunktion benutzt werden.

Kapitel 4

Angriffe auf Kryptosysteme

4.1 Angriffe auf Kryptosysteme

4.1.1 Angriffsklassen

Neben den Parteien, die vertrauliche Daten austauschen, gibt es noch weitere Parteien, die diese Daten gerne erfahren oder manipulieren wollen. Diese Angreifer können dazu mögliche Sicherheitslücken des verwendeten Kryptosystems oder des verwendeten Protokolls ausnutzen. Sie können diese aber auch umgehen, wenn der Benutzer seine Daten nicht ausreichend schützt. Das beste Kryptosystem nutzt nichts, wenn ein Trojanisches Pferd (siehe Kapitel 4.3.1) das Paßwort oder den Schlüssel abfangen und einem Angreifer übermitteln kann.

Ein Kryptosystem basiert auf Algorithmen, deren Sicherheit allgemein anerkannt wird. Insbesondere müssen diese Algorithmen folgende Eigenschaften besitzen:

1. Der Algorithmus ist nicht geheim.
2. Der Algorithmus steht für Analysen zur Verfügung.
3. Der Algorithmus wurde bereits von Fachleuten analysiert.
4. Der Algorithmus zeigt keine praktischen Schwächen.

Das bedeutet, daß ein Algorithmus erst eine längere Testphase durchlaufen muß, ehe seine Sicherheit anerkannt wird.

Um ein Kryptosystem zu analysieren, sind verschiedene Ansätze denkbar. Der verwendete Algorithmus ist dem Angreifer sicher bekannt. Anhand weiterer ihm bekannter Informationen werden vier Arten von Angriffen unterschieden:

1. **Ciphertext-Only-Attacke:** Der Angreifer kennt eine begrenzte Anzahl von Geheimtexten und versucht daraus die zugehörigen Klartexte oder den verwendeten Schlüssel zu ermitteln.
2. **Known-Plaintext-Attacke:** Der Angreifer kennt eine begrenzte Anzahl von Geheimtexten mit den zugehörigen Klartexten und möchte den Klartext eines weiteren Geheimtextes oder den verwendeten Schlüssel ermitteln.
3. **Chosen-Plaintext-Attacke:** Der Angreifer kann die Geheimtexte zu selbst gewählten Klartexten erzeugen und möchte den Klartext eines anderen Geheimtextes oder den verwendeten Schlüssel ermitteln.
4. **Chosen-Ciphertext-Attacke:** Der Angreifer kann selbst gewählte Geheimtexte entschlüsseln und sucht den verwendeten Schlüssel.

Ein Kryptosystem heißt sicher, wenn keiner der Angriffe erfolgversprechend ist. Die Mindestanforderung heutzutage ist die Resistenz gegen die ersten beiden Attacken. Es ist ebenfalls empfehlenswert, daß das Kryptosystem dem Chosen-Plaintext-Angriff standhält.

4.1.2 Brute-Force-Angriff

Fast jedes Kryptosystem kann mit einem speziellen Ciphertext-Only-Angriff gebrochen werden, dem **Brute-Force-Angriff**. Dahinter verbirgt sich nichts anderes als das Ausprobieren aller denkbaren Schlüssel auf einen Geheimtext. Ergibt die Entschlüsselung einen sinnvollen Text, so ist der verwendete Schlüssel gefunden.

Es gibt einige wenige Ausnahmen: Beim One-Time-Pad existiert für jeden sinnvollen Text der entsprechenden Länge ein Schlüssel. Hier ist dieser Angriff wirkungslos, denn niemand kann entscheiden, welcher Text korrekt ist.

Da dieser Angriff also auf fast jeden Algorithmus wirkt, muß er zumindest praktisch undurchführbar gemacht werden. Das geschieht dadurch, daß der Schlüsselraum so groß gewählt wird, daß das Ausprobieren aller Schlüssel selbst mit Hilfe modernster Technik eine Ewigkeit dauern würde. Es gibt auch die Forderung, daß selbst theoretisch denkbare Computer die Tests nicht in ausreichender Zeit bewältigen können. Allerdings würde dann sogar die Verschlüsselung zu lange dauern. Daher wird meist ein guter Kompromiß bei der Schlüssellänge gefunden. Die Empfehlungen der Experten sind daher auch nur für einen kurzen Zeitraum relevant, manche geben sogar jährlich neue Empfehlungen.

4.1.3 Kryptanalyse

Erfolgversprechender als Brute-Force-Angriffe sind Analysen, die Muster und Eigenschaften der Systeme untersuchen. So sind vor allem bei den einfacheren älteren Verschlüsselungstechniken statistische Tests anwendbar. Die Buchstaben eines Textes sind nicht alle gleich wahrscheinlich, das läßt sich unter Umständen auch im Chiffretext feststellen. Selbst Blöcke von Buchstaben kommen mit unterschiedlichen Häufigkeiten vor. Mit Hilfe der Statistiken können Rückschlüsse vom Chiffretext auf den Klartext gezogen werden. Daher haben alle aktuellen Algorithmen einen Mechanismus, welcher dafür sorgt, daß die Häufigkeiten der ausgegebenen Zeichen sich nicht wesentlich unterscheiden.

Bei der **differentiellen Kryptanalyse** werden Paare von Chiffretexten gezielt betrachtet. Die zugehörigen Klartexte weisen bestimmte Differenzen auf, deren Entwicklung bei der Chiffrierung mit dem gleichen Schlüssel untersucht wird. Beim DES-Algorithmus ist mit dieser Methode ein Angriff gefunden worden, der etwas effizienter als der Brute-Force-Angriff ist.

Noch erfolgreicher bei DES ist die **lineare Kryptanalyse**. Dabei werden lineare Approximationen benutzt, welche die Blockchiffren möglichst genau beschreiben. Diese Approximation stimmt mit einer gewissen Wahrscheinlichkeit p mit der Originalfunktion überein. Ist p ungleich $\frac{1}{2}$, dann läßt sich diese Asymmetrie ausnutzen. Aus Klartext und zugehörigem Chiffretext können einzelne Schlüsselbits geraten werden. Je mehr Daten zur Verfügung stehen, desto zuverlässiger ist die Schätzung. Daß die lineare Kryptanalyse den derzeit besten Angriff gegen DES darstellt, liegt vermutlich daran, daß beim Entwurf von DES die S-Boxen zwar gegen differentielle Kryptanalyse optimiert wurden, aber nicht gegen die lineare Kryptanalyse.

4.2 Angriffe auf Protokolle

4.2.1 Einfache Angriffe

Selbst wenn sichere Kryptosysteme benutzt werden und alle Daten sicher verwahrt werden, kann doch ein Angreifer Einfluß auf die Kommunikation der Parteien haben. Dazu kann er bestimmte Schwachstellen im verwendeten Protokoll ausnutzen.

Häufig wird zum Beispiel mit Festcodes gearbeitet. Dabei werden oft solche Paßwörter gewählt, welche leicht zu merken sind. Das ermöglicht einen **Wörterbuchangriff**. Ist das Paßwort ein lexikalischer Begriff, so muß nur jeder Eintrag eines Wörterbuches geprüft werden. Auch andere beliebte Paßwörter, wie Geburtsdaten oder Ersetzung von speziellen Buchstaben durch Zahlen, können damit gefunden werden. Insgesamt verkürzt dies die Zeit einer Suche nach dem Paßwort erheblich.

Nicht jeder Angriff richtet sich gegen die geheimen Daten eines Benutzers. Mit dem **Denial-Of-Services-Angriff** wird zum Beispiel versucht, eine Partei außer Gefecht zu setzen. Dazu werden einfach riesige Mengen von Daten gesendet, welche dann die Gegenseite überlasten. Das ist etwa genauso, wie wenn jemandem in einer geselligen Runde mehrere Fragen gleichzeitig gestellt werden. Die Beantwortung aller Fragen ist hier schlicht unmöglich.

4.2.2 Arglistige Täuschung

Auch wenn ein Protokoll strikt eingehalten wird, so gibt es für einen Außenstehenden Möglichkeiten, die Kommunikation der Parteien zu manipulieren. Der **Man-In-The-Middle-Angriff** ist bei Fest- und Wechselcodes, aber auch beim Diffie-Hellman-Schlüsselaustausch anwendbar. Er eignet sich insbesondere für asymmetrische Verfahren, wenn der öffentliche Schlüssel der Parteien nicht allgemein bekannt ist.

Der Angreifer sitzt hier zwischen den Parteien und gibt sich gegenüber einer Partei als die jeweils andere Partei aus. Dabei kann er die Nachrichten nicht nur abhören, sondern auch manipulieren. So kann er zum Beispiel bei asymmetrischen Verfahren seinen eigenen öffentlichen Schlüssel statt dem öffentlichen Schlüssel der beteiligten Personen weitergeben. Danach ist er in der Lage, die Nachrichten zu seinem Vorteil zu ändern oder zu löschen. Er kann sogar neue Nachrichten generieren. Einen Schutz dagegen liefert das Interlock-Protokoll, auch digitale Signaturen können helfen.

Manchmal reicht es einem Angreifer schon, wenn er eine Nachricht, zum Beispiel eine Überweisung zu seinen Gunsten, erneut senden kann. Dieser Angriff heißt **Replay-Angriff**. Wenn jede Nachricht ein Unikat ist, so ist diese Attacke wirkungslos. Eine Nachricht kann mit einem Zeitstempel oder einer laufenden Nummer versehen werden. Der Empfänger erkennt dadurch, daß eine Nachricht schon einmal versandt wurde.

4.3 Schwachstelle Benutzer

4.3.1 Sicherheit der Benutzerdaten

Selbst wenn die sichersten kryptographischen Algorithmen zusammen mit ausgereiften Protokollen benutzt werden, so sind Geheimnisse nicht sicher, solange die Benutzer ihre Daten nicht ausreichend schützen. Wenn ein geheimer Schlüssel auf einem Zettel auf dem Schreibtisch notiert ist, so braucht kein Angreifer umständlich die Chiffretexte zu analysieren. Im einfachsten Falle reicht schon ein Fernglas und ein Richtmikrofon.

Mittels spezieller Software lassen sich Daten ausspionieren. Das beste Beispiel hierfür sind Trojanische Pferde. Das sind Programme, die neben ihrer offiziellen Funktionalität zusätzliche Informationen sammeln und weiterleiten. So kann ein Trojanisches Pferd zum Beispiel einen Login-Vorgang simulieren, Benutzername und Paßwort ausspionieren, danach eine Fehlermeldung anzeigen und die Kontrolle dem Originalprogramm zurückgeben. Der Benutzer merkt eventuell gar nicht, daß er zwei verschiedene Programme vor sich hat und denkt, er habe sich nur vertippt.

Auch Viren können dazu mißbraucht werden, geheime Daten zu sammeln. Viren sind Programme, die sich vervielfältigen und verbreiten können. Sie haben daher meist Zugriff auf viele Rechner, solange sie deren Schwachstellen ausnutzen können. Als Beispiel seien einige Mailprogramme genannt, welche mit einer Mail geschickte Basic-Skripte automatisch ausführen. Das liefert auch einen wirkungsvollen Denial-of-Services-Angriff.

4.3.2 Schutz der Benutzerdaten

Benutzer müssen sich vor solch einfachen Methoden schützen. Ein Paßwort sollte daher nie aufgeschrieben werden. Im Zweifelsfalle ist eine Eselsbrücke statt dem eigentlichen Codewort besser. Geheime Schlüssel gehören nicht im Klartext auf den Rechner. Der Zugang zum Rechner sollte nur wenigen vorbehalten sein. Ein Computer im Internet ist eine leichte Beute, wenn nicht einige Sicherheitsvorkehrungen getroffen werden. Ein Firewall verhindert, daß fremde Rechner unbemerkt über nicht genutzte Ports auf den eigenen Rechner zugreifen können. Antivirenprogramme erkennen Viren und Trojanische Pferde und machen diese unschädlich. Die erfolgreichsten Angriffe sind diejenigen, die auf der Unwissenheit oder der fehlenden Sorgfalt der Benutzer basieren.

Kapitel 5

Homebanking

5.1 Der HBCI-Standard

5.1.1 Beschreibung

Das **Homebanking-Computer-Interface** (HBCI)²⁷ ist ein deutscher Standard zur Kommunikation zwischen dem Kundensystem (Rechner mit entsprechender Software) und den Bankrechnern zur Durchführung von Transaktionen beim Homebanking.

1994 legte der Bundesverband deutscher Banken (BdB) dem Zentralen Kreditausschuß (ZKA) ein erstes Konzept für einen Homebanking-Standard vor. Etwa ein Jahr später erhielt der erste HBCI-Entwurf des ZKA die Zustimmung der anderen Verbände in Deutschland. Die erste HBCI-Version wurde im November 1996 veröffentlicht. Die Version 2.2 ist seit Mai 2000 im Einsatz. Weitere Versionen existieren bereits, werden aber in der Praxis derzeit noch nicht eingesetzt.

HBCI wurde inzwischen als Vorschlag in mehreren Gremien der EU eingereicht. Es gibt noch weitere Standards, die ähnliche Ziele verfolgen. Zu nennen wären dabei vor allem die US-amerikanischen Entwicklungen *Open Financial Exchange* (OFX) und der darauf aufbauende *International Financial Exchange* (IFX)²⁸.

²⁷Siehe <http://www.hbci.de> [26].

²⁸Siehe <http://www.ofx.net> [30] und <http://www.ifxforum.org> [28].

5.1.2 Ablauf des HBCI-Dialogs

Nachrichten können nur innerhalb eines HBCI-Dialogs gesendet werden. Ein HBCI-Dialog ist folgendermaßen aufgebaut:

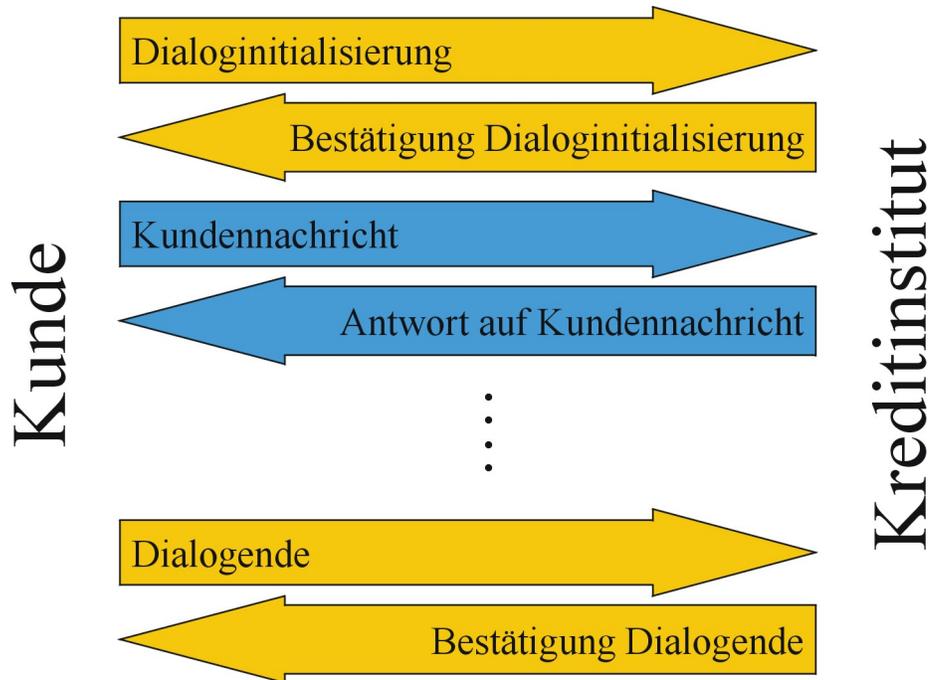


Abbildung 5.1: Ablauf des HBCI-Dialogs

Ein solcher Dialog läuft stets synchron ab, das heißt, jede Kundennachricht muß vom Kreditinstitut beantwortet werden, ehe die nächste Kundennachricht gesendet werden kann.

Die Dialoginitialisierung dient der gegenseitigen Authentisierung und dem Abgleich einiger Kommunikationsparameter. Die eigentlichen Kundennachrichten werden vor der Übertragung verschlüsselt und signiert, ebenso die Antworten vom Kreditinstitut. Durch eine Dialogbeendigungsnachricht wird sichergestellt, daß alle Nachrichten komplett und korrekt übertragen wurden.

Es gibt auch die Möglichkeit der anonymen Kommunikation, um nicht signaturpflichtige Daten abzurufen. Dazu gehören zum Beispiel Börsenkurse und allgemeine Informationen, aber auch die öffentlichen Bankschlüssel. Diese Daten werden unverschlüsselt übertragen.

5.1.3 Sicherheitsmechanismen

Im HBCI-Standard werden mehrere Sicherheitsmechanismen eingesetzt. Die Nachrichten werden verschlüsselt und signiert, wobei für jede Nachricht ein neuer Nachrichtenschlüssel benutzt wird. Ein Signaturzähler verhindert die wiederholte Übertragung derselben Nachricht. Die geheimen Daten der Benutzer werden nicht auf dem Rechner gespeichert, sondern auf einem Sicherheitsmedium. Das ist im Normalfall eine Chipkarte, aber es können auch Disketten dafür benutzt werden.

Im Rahmen von HBCI werden zwei verschiedene Verfahren benutzt, das DES-DES-Verfahren (DDV) und das RSA-DES-Hybridverfahren (RDH). Bei DDV wird das Triple-DES-Verfahren sowohl zum Signieren der Nachricht als auch zum Chiffrieren der Nachrichtenschlüssel verwendet. Die Schlüssel sind dabei 128 Bit lang²⁹. Bei RDH wird zum Verschlüsseln das RSA-Verfahren benutzt, die Schlüssel haben hier eine Länge von 768 Bit.

Jeder Benutzer benötigt Signierschlüssel und Chiffrierschlüssel. Bei DDV, das mit symmetrischen Verfahren arbeitet, sind dies geheime Schlüssel, die nur dem Benutzer und dem Kreditinstitut bekannt sind. Dagegen besitzt der Benutzer bei RDH, wo auch asymmetrische Verfahren zum Einsatz kommen, einen privaten Signierschlüssel und einen privaten Chiffrierschlüssel, die nur er kennt, sowie die entsprechenden öffentlichen Schlüssel, die jeder kennen darf.

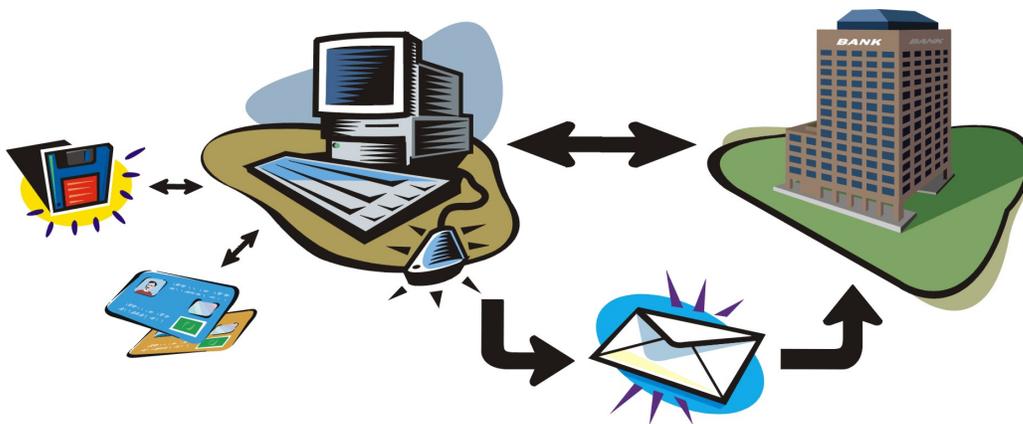


Abbildung 5.2: Kommunikationswege bei HBCI

²⁹Die effektive Schlüssellänge beträgt nur 112 Bit. Bei den weiteren Betrachtungen werden die 16 Prüfbits aber mit einbezogen.

5.1.4 Signieren von Nachrichten

Die Signaturberechnung einer Nachricht bei DDV funktioniert wie folgt:

1. **Bildung des Hashwerts der Nachricht:**

Der Hashwert wird mit der Hashfunktion RIPEMD-160 (siehe Kapitel 3.4.3) berechnet. Der erzeugte Hashwert hat eine Länge von 20 Byte (160 Bit).

2. **Formatierung des Hashwerts:**

Dem Hashwert wird „00 00 00 00“ vorangestellt, so daß er nun eine Länge von 24 Byte hat.

3. **Berechnung der Signatur:**

Als Signatur wird ein sogenannter *Retail CBC-MAC* gebildet. Dazu wird der formatierte Hashwert in drei Blöcke zu je 8 Byte aufgeteilt. Mit dem Initialisierungsvektor „00 00 00 00 00 00 00 00“ wird zunächst ein einfacher CBC-MAC über die ersten beiden Blöcke berechnet. Hier kommen als Schlüssel die ersten 64 Bit des Signierschlüssels des Benutzers zum Einsatz. Das Ergebnis wird mit dem verbleibenden Nachrichtenblock mittels \oplus verknüpft und danach mit dem kompletten Signierschlüssel mit Triple-DES verschlüsselt. Der so erhaltene Ausgabeblock der Länge 8 Byte (64 Bit) bildet den Retail CBC-MAC.

Bei RDH wird die Signatur folgendermaßen berechnet:

1. **Bildung des Hashwerts der Nachricht:**

Der Hashwert wird genauso gebildet wie bei DDV.

2. **Formatierung des Hashwerts:**

Danach durchläuft der Hashwert einen mehrstufigen Formatierungsprozeß³⁰. Das Ergebnis dieses Prozesses hat eine Länge von 64 Byte.

3. **Berechnung der Signatur:**

Der erhaltene Wert wird nun signiert. Dabei kommt eine Variante des RSA-Signaturverfahrens aus Kapitel 3.5.1 zum Einsatz³¹.

³⁰Der gesamte Prozeß ist im ISO-Standard 9796:1991 [29] beschrieben.

³¹Siehe Anhang A im ISO-Standard 9796:1991 [29].

Der Formatierungsprozeß bei RDH liefert einen zusätzlichen Vorteil: Beim Überprüfen der Signatur kann der Empfänger schon nach der Entschlüsselung erkennen, ob das Format korrekt ist, ohne die dazugehörige Nachricht überhaupt zu kennen. Wurde die Signatur bei der Übertragung ausgetauscht, so ist das Format der gefälschten Signatur mit großer Wahrscheinlichkeit fehlerhaft. Das liegt daran, daß nach der Verschlüsselung des formatierten Hashwertes die Struktur nicht mehr zu erkennen ist und daher ein Angreifer nur dann eine Signatur mit korrektem Format erzeugen kann, wenn er den geheimen Signierschlüssel des Benutzers kennt.

5.1.5 Verschlüsselung von Nachrichten

Die Nutzdaten einer HBCI-Nachricht werden generell mittels Triple-DES chiffriert. Für diese Verschlüsselung wird bei beiden Verfahren für jede einzelne Nachricht ein separater 128 Bit langer Nachrichtenschlüssel verwendet. Dieser wiederum wird mit dem Chiffrierschlüssel des Empfängers verschlüsselt und mit der Nachricht mitgeliefert.

Das heißt, bei beiden Verfahren müssen zunächst zwei DES-Schlüssel zufällig erzeugt werden. Es muß dabei sichergestellt werden, daß die Prüfbits korrekt gesetzt sind, daß die beiden Schlüssel nicht identisch sind und daß es sich um keine schwachen oder semi-schwachen Schlüssel handelt. Zusammen bilden diese beiden DES-Schlüssel den Nachrichtenschlüssel, mit welchem die gesamte Nachricht mit dem Triple-DES-Verfahren im CBC-Modus chiffriert wird. Der Initialisierungsvektor dafür lautet „00 00 00 00 00 00 00 00“.

Die Verschlüsselung des Nachrichtenschlüssels erfolgt bei DDV mit dem Triple-DES-Verfahren im ECB-Modus, bei RDH dagegen mit dem RSA-Verfahren. Bei letzterem muß der Nachrichtenschlüssel allerdings vorher noch mit „00“ auf 768 Bit aufgefüllt werden. Als Schlüssel wird jeweils der Chiffrierschlüssel des Empfängers verwendet. Dieser ist dem Benutzer in beiden Fällen bekannt, denn beim RSA-Verfahren wird der öffentliche Schlüssel des Empfängers eingesetzt, während beim DES-Verfahren der eigene Chiffrierschlüssel mit dem des Empfängers identisch ist (natürlich darf er in diesem Falle nicht öffentlich sein).

5.1.6 Erstinitialisierung DDV

Die Benutzerschlüssel müssen zuerst einmal erzeugt und verteilt werden, bevor sie beim Homebanking benutzt werden können. Beim symmetrischen Verfahren (DDV) sind zwei Aspekte dabei zu beachten:

1. Jedes Kreditinstitut besitzt einen 128 Bit langen Masterkey für jede Schlüsselart. Diese Schlüssel müssen eindeutig sein, kein anderes Kreditinstitut darf dieselben Masterkeys benutzen. Die Masterkeys dienen zum Erzeugen der Kundenschlüssel, daher müssen sie in einer sicheren Umgebung gespeichert sein.
2. Die Kundenschlüssel werden auf einer Chipkarte gespeichert, wo die Sicherheit durch einige Hardwareschutzmechanismen gewährleistet wird (siehe Kapitel 5.1.8). Diese fehlen bei der Diskettenvariante, daher ist eine Speicherung der Schlüssel darauf im HBCI-Standard bei DDV nicht vorgesehen. Außerdem sind auf der Chipkarte einige Kundendaten unter *Cardholders Information Data* (CID) gespeichert, die zur Herleitung der Kundenschlüssel benutzt werden.

Die Erzeugung eines Kundenschlüssels erfolgt folgendermaßen:

1. Von der Chipkarte wird das Feld CID ausgelesen und mit „00“ auf das nächste Vielfache von 8 Byte aufgefüllt.
2. Darauf wird die Hashfunktion MDC-2 (siehe Kapitel 3.4.4) angewendet.
3. Der so erhaltene Hashwert wird mit dem entsprechenden Masterkey mittels Triple-DES im ECB-Modus verschlüsselt.
4. Im Ergebnis der Verschlüsselung müssen die Prüfbits korrekt gesetzt werden. Das erledigt eine *Parity-Adjustment-Funktion*.
5. Die 128 Ausgabebits dieser Funktion bilden den jeweiligen Kundenschlüssel.

Die berechneten Schlüssel werden auf der Chipkarte gespeichert und der Schreibschutz der Chipkarte irreversibel aktiviert. Dieser Vorgang heißt Personalisierung der Chipkarte. Die Verteilung der Schlüssel erfolgt implizit mit der Aushändigung der Chipkarte.

Schlüsseländerungen sind hier auf elektronische Weise nicht möglich. Sollte ein Schlüssel kompromittiert werden, so muß zwangsläufig eine neue Chipkarte mit einem veränderten CID-Feld erstellt werden. Ist gar der Masterkey des Kreditinstituts bekannt, so müssen alle Chipkarten neu erstellt werden, da dann jeder die Kundenschlüssel selbst herleiten kann.

5.1.7 Erstinitialisierung RDH

Die Erzeugung der Kundenschlüssel erledigt beim asymmetrischen Verfahren (RDH) jede Partei selbständig. Dabei kommt das Verfahren aus Kapitel 3.3.2 zum Einsatz, allerdings mit einigen Einschränkungen:

1. Der öffentliche Exponent e hat den festen Wert $2^{16} + 1$.
2. Die Länge des Modulus n soll 768 Bit betragen. Dieser Wert darf maximal um 60 Bit unterschritten werden.
3. An die Faktoren p und q von n werden weitere Anforderungen gestellt:
 - $p - 1$ hat einen großen Primteiler r .
 - $q - 1$ hat einen großen Primteiler s .
 - $p + 1$, $q + 1$, $r - 1$ und $s - 1$ haben je einen großen Primteiler.
 - Die Längen von p und q unterscheiden sich um maximal 12 Bit.

Diese Bedingungen sind in der mitgelieferten Banking-Software, welche auch die Schlüsselerzeugung durchführt, bereits implementiert, der Benutzer muß dies nicht erst noch nachprüfen.

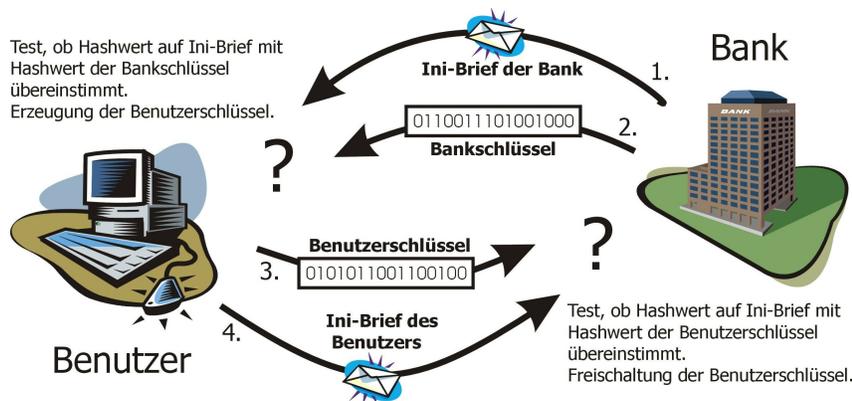


Abbildung 5.3: Schlüsselaustausch bei Erstinitialisierung RDH

Werden die Schlüssel bei dem Kreditinstitut generiert, so erfolgt die Verteilung implizit mit der Ausgabe der Chipkarte bzw. der Schlüsseldiskette. Wird die Schlüsselerzeugung dagegen beim Kunden durchgeführt, so erfolgt die Verteilung der Schlüssel in mehreren Schritten:

1. Der Benutzer holt sich auf elektronischem Wege die öffentlichen Schlüssel der Bank, berechnet den Hashwert des Signierschlüssels und vergleicht diesen mit dem Hashwert auf dem Ini-Brief des Kreditinstituts. Der Ini-Brief wird dem Kunden bei der Anmeldung zum Onlinebanking ausgehändigt und enthält neben einigen Kundendaten auch den Modulus und den Hashwert des öffentlichen Signierschlüssels der Bank.
2. Mit dem öffentlichen Signierschlüssel der Bank überprüft der Benutzer nun die Signatur des öffentlichen Chiffrierschlüssels des Kreditinstituts.
3. Nach erfolgreicher Überprüfung kann der Benutzer seine eigenen öffentlichen Schlüssel an das Kreditinstitut schicken. Diese Nachricht wird sowohl chiffriert als auch signiert. Außerdem schickt der Benutzer per Post einen unterschriebenen Ini-Brief an das Kreditinstitut oder gibt diesen persönlich dort ab. Auch hier sind wieder Modulus und Hashwert des eigenen öffentlichen Signierschlüssels aufgeführt.
4. Das Kreditinstitut berechnet jetzt den Hashwert des erhaltenen Signierschlüssels und vergleicht diesen mit dem Hashwert auf dem Ini-Brief. Stimmen diese überein, so wird der Signierschlüssel des Benutzers freigeschaltet. Außerdem kann die Bank nun auch die Signatur des Chiffrierschlüssels überprüfen und diesen ebenfalls freischalten.

Bei diesem Verfahren sind auch Schlüsseländerungen auf elektronischem Wege möglich. Dabei wird ein neuer Schlüssel generiert und mit dem alten signiert. Die Bank tauscht dann die alten gegen die neuen Schlüssel aus. Allerdings darf bei einer Kompromittierung der Schlüssel die Schlüsseländerung nicht auf diese Weise erfolgen. In diesem Falle ist wieder eine komplette Erstinitialisierung erforderlich. Sollten gar die geheimen Bankschlüssel bekannt werden, so muß die Bank neue Schlüssel generieren und jeden Benutzer schnellstmöglich darüber informieren, daß sich die Schlüssel geändert haben. Dies geschieht in der Regel beim Login zum Onlinebanking.

5.1.8 Sicherheitsmedien

Zum Speichern der Benutzerschlüssel kommen verschiedene Medien zum Einsatz, bei der Entwicklung von HBCI ist vor allem der Einsatz von Chipkarten vorgesehen. Die üblichen ec-Karten sind allerdings nicht in der Lage, kryptographische Operationen durchzuführen, daher werden hier die Schlüssel nur abgespeichert und mit einer vierstelligen PIN geschützt. Das Kundensystem liest die Schlüssel aus und erledigt damit die Verschlüsselung sowie das Signieren selbständig.

RSA-Chipkarten dagegen besitzen einen Koprozessor, der die wichtigsten RSA-Operationen, Zufallszahlenerzeugung und Primzahltests durchführen kann. Diese Karten besitzen Hardwareschutzmechanismen, die den Inhalt vor dem Auslesen schützen soll. Das Betriebssystem der Karte hält ebenfalls einen Schutz vor unberechtigtem Zugriff auf die Daten bereit. Der geheime Schlüssel verläßt diese Chipkarte nie, er kann auch nur direkt vom Koprozessor erzeugt werden. Die Operationen des Koprozessors können erst nach der Verifikation der fünfstelligen PIN der Chipkarte abgearbeitet werden.

Als Alternative können die Schlüssel auch auf einer Diskette gespeichert werden. Der einzige Schutz hier ist eine paßwortbasierte Verschlüsselung. Der Kunde wählt dazu einen Benutzernamen und ein Paßwort, aus welchen ein Schlüssel generiert wird. Mit diesem Schlüssel werden die geheimen Benutzerschlüssel chiffriert und danach wird über die gesamten Daten ein MAC berechnet, der die Daten vor Manipulation schützen soll. Dieser MAC ist ebenfalls von dem aus Paßwort und Benutzername erzeugtem Schlüssel abhängig.

5.2 Schwachstellen des Protokolls

5.2.1 Kommunikation

Der HBCI-Standard regelt den Austausch von Nachrichten über eine unsichere Verbindung. Wer diese Kommunikation belauschen möchte, kann sich nur wenige Eigenschaften zunutze machen.

Die Schlüssellänge des Nachrichtenschlüssels ist seit der Version 2.1 auf 128 Bit festgelegt, eine Reaktion auf die allgemeine Expertenmeinung, daß 64 Bit zuwenig seien. Die RSA-Schlüssel haben eine Länge von 768 Bit, dies wird sich in der nächsten Version aller Wahrscheinlichkeit nach auf 1024 Bit erhöhen.

Die einzelnen Nachrichten folgen einer festen Syntax, der Aufbau ist klar definiert. Jede Nachricht hat einige feste Bestandteile, wie zum Beispiel den Nachrichtenkopf. Es lassen sich unter Umständen sogar Teile von Nachrichten anhand der Struktur erraten. Basiert das Kundensystem auf einem Java-Applet, so werden die HBCI-Nachrichten meist mittels SSL übertragen. SSL steht für Secure Sockets Layer und ist der im Webbrowser integrierte Sicherheitsmechanismus³². Der Angreifer muß zunächst diese Hürde überwinden, ehe er die Struktur einer HBCI-Nachricht für Analysen ausnutzen kann.

Die Kommunikation zwischen Chipkartenleser und Kundensystem wird von Experten als sicher angesehen. Die Hersteller der Chipkarten bzw. der Chipkartenleser haben sehr viel investiert, um dies zu erreichen. Einige haben sogar eine längere Erfahrung mit Sicherheitstechniken vorzuweisen, wie bei der Entwicklung von Sicherheitsmerkmalen für Geldscheine.

Als schwächstes Glied der Kommunikationskette bleibt der Computer des Kunden. Auf diesem wird die Banking-Software ausgeführt, er bildet also die Schnittstelle zwischen der Chipkarte und dem Bankrechner. Das heißt insbesondere, daß Homebanking nur dann sicher genug ist, wenn der eigene Rechner gegen Angriffe ausreichend geschützt ist³³.

5.2.2 Speicherung der Daten

Die Daten auf Chipkarten sind gegen physisches Auslesen gut geschützt. Kennt ein Angreifer jedoch die PIN der Chipkarte, so kann er jede Aktion damit durchführen, die auch der rechtmäßige Besitzer durchführen kann. Die vierstellige PIN der ec-Karten ist dafür zu unsicher³⁴, denn einige PINs wie 4711 oder 3333 werden nie vergeben und einige Ziffern treten öfter auf als andere. Da ein Angreifer drei Versuche zur Verfügung hat, kann er die PIN mit einer Wahrscheinlichkeit von etwa 1/150 erraten. Deswegen haben die RSA-Chipkarten eine fünfstellige PIN, wobei jede Ziffernfolge möglich und gleichwahrscheinlich ist. Trotzdem bleibt die PIN das einfachste Angriffsziel.

Der Schutz der Daten auf einer Schlüsseldiskette erfolgt durch die Chiffrierung der Daten mit Benutzername und Paßwort. Das ermöglicht unter Umständen einen Wörterbuchangriff. Aus diesem Grund ist in der Spezifikation festgelegt, daß das Paßwort mindestens acht Zeichen lang sein muß und mindestens ein Sonderzeichen beinhalten muß.

³²Details zu SSL sind bei Smith [12, Seite 231ff.] zu finden.

³³Der Computer Chaos Club hat im Hessischen Rundfunk einen solchen Angriff erfolgreich demonstriert [24].

³⁴Siehe ec-Karten-Urteil vom Oberlandesgericht Hamm [25].

5.3 Analyse der Angriffsmöglichkeiten

5.3.1 Allgemeine Angriffe

Die Kreditinstitute werben häufig für die vielen Vorteile des Homebankings. Insbesondere wird oft erwähnt, daß die Server rund um die Uhr erreichbar seien. Das ist aber gleichzeitig auch ein Nachteil, denn dadurch sind die Erfolgsaussichten eines Denial-of-Services-Angriff höher. Einerseits muß der Server jede anonyme Abfrage der Bankschlüssel beantworten, andererseits kommunizieren im Normalfalle nicht so viele Kunden mit dem Server, daß dieser ausgelastet wäre.

Für den Angriff werden daher eigentlich nur möglichst viele Rechner mit verschiedenen IP-Adressen benötigt, die entweder eine Abfrage der Bankschlüssel an den Server senden oder aber eine Kommunikation mit einer fiktiven oder einer abgefangenen Kunden-ID aufbauen möchten. Der Server muß auf jeden Fall antworten, auch wenn die Daten falsch sind. Er sendet dann eine Fehlermeldung.

Daß der Angriff erfolgreich sein kann, liegt daran, daß der Server nicht zwischen Kunde und Angreifer unterscheiden kann. Die beste Abwehr liegt daher in der Beschränkung der gleichzeitig zugelassenen Verbindungen. Aber auch hier wäre dann ein Kunde wahrscheinlich nicht imstande, seine Überweisungen zu tätigen, da alle Verbindungen schon besetzt wären. Das ist zwar lästig, aber die Folgen dieses Eingriffs sind nicht schwerwiegend, in dringenden Fällen schaut der Kunde eben persönlich bei seiner Bank vorbei.

Ein Brute-Force-Angriff auf den Nachrichtenschlüssel ist unsinnig, denn dieser gilt nur für eine einzige HBCI-Nachricht, also für Sekundenbruchteile. Im Gegensatz dazu hat ein Angreifer unter Umständen ein Jahr Zeit für einen Brute-Force-Angriff auf die Benutzerschlüssel. Voraussetzung dafür ist allerdings, daß er einen Nachrichtenschlüssel und dessen chiffrierte Entsprechung ermittelt hat oder dem Benutzer einen eigenen Nachrichtenschlüssel untergeschoben hat. Das ist schon schwierig genug, doch selbst wenn er dies geschafft hat, so sind die Schlüssellängen von 128 Bit (DDV) bzw. 768 Bit (RDH) groß genug, damit der Angreifer keinen Erfolg verbuchen kann.

Hier muß allerdings beachtet werden, daß sich der Aufwand durch neue Technologien immer mehr verringert, daher sollten die Schlüssellängen den Anforderungen und den Empfehlungen der Experten angepaßt werden. Es ist zu erwarten, daß in den folgenden HBCI-Versionen die Schlüssellängen 192 Bit (DDV) bzw. 1024 Bit (RDH) betragen werden. Bis dahin können letzte Zweifel ausgeräumt werden, indem die Benutzerschlüssel regelmäßig geändert werden.

Ein Angreifer wird aber auch wissen, daß es bessere Angriffe als den Brute-Force-Angriff gibt, speziell für Triple-DES mit zwei Schlüsseln. Der von Oorschot und Wiener entwickelte Angriff³⁵ ist ein Known-Plaintext-Angriff, der eine große Menge von Klartexten benötigt. Ist p die Anzahl der bekannten Klartexte, so benötigt dieser Angriff Speicherplatz für p Blöcke und $2^{120}/p$ Schritte³⁶. Sobald p größer als 256 ist, ist dieser Angriff schneller als der normale Brute-Force-Angriff.

Mit Triple-DES mit drei Schlüsseln ist der Aufwand des Angreifers noch größer. Der beste bisher bekannte Angriff benötigt hier 2^{112} Schritte und Speicherplatz für 2^{56} Blöcke.

Bei RSA ist der Versuch, den Modulus n zu faktorisieren, effizienter als ein Brute-Force-Angriff, auch wenn er ebensowenig Erfolg verspricht. Es gibt auch Angriffe gegen RSA, die unter speziellen Voraussetzungen erfolgreich sind. Ist zum Beispiel in einem Public-Key-Netzwerk der Verschlüsselungsexponent e sehr klein, dann werden nur e identische Nachrichten benötigt, die mit verschiedenen öffentlichen Schlüsseln chiffriert werden, um wenigstens einen der zugehörigen privaten Schlüssel herauszufinden. Um dem entgegenzuwirken, füllen RSA-Implementierungen, die kleine Exponenten erlauben, die Nachrichten mit Zufallswerten auf. Auch gegen RSA-Implementierungen mit gemeinsamen Modulus, die mit variablen Exponenten arbeiten, gibt es gute Angriffsmöglichkeiten³⁷.

Weitere Schwachstellen sind die Wahl eines kleinen Entschlüsselungsexponenten d und das Signieren einer bereits verschlüsselten Nachricht. Diese Faktoren wurden im HBCI-Standard beachtet und geeignete Festlegungen getroffen. So wird jede Nachricht erst signiert und dann verschlüsselt, d darf keine kleinen Werte annehmen, der Verschlüsselungsexponent e hat den festen Wert $2^{16} + 1$ und der Modulus n bleibt dafür variabel.

³⁵Siehe Schneier [9, Seite 413f.].

³⁶Ein Schritt bezeichnet hier den Test eines Schlüssels.

³⁷Siehe Schneier [9, Seite 538f.].

5.3.2 Angriffe auf das Protokoll

Der HBCI-Standard soll eine sichere Kommunikation zwischen Kunde und Bank gewährleisten. Ein Angreifer, der sich zwischen den beiden Systemen befindet, kann im Normalfall die Nachricht weder lesen noch modifizieren, da bis auf wenige Ausnahmen jede Nachricht signiert und verschlüsselt ist. Der Man-in-the-middle-Angriff ist also nur auf zwei Typen der Kommunikation anwendbar, auf anonyme Verbindungen und auf die Schlüsselverteilung. Bei ersteren erhält der Angreifer keine Daten, die er nicht auch legal erhalten könnte, also bleibt ihm nur die initiale Schlüsselverteilung als Angriffsziel.

Der Dialog, in welchem der Kunde die Bankschlüssel abfragt, ist anonym und nicht verschlüsselt. Der Angreifer weiß also sofort, daß ein Benutzer neue Schlüssel erzeugen möchte. Er kann die Nachricht der Bank abfangen und dem Benutzer stattdessen seine eigenen Schlüssel senden³⁸.

HBCI verwendet hier kein spezielles Protokoll wie zum Beispiel das Interlock-Protokoll, die Korrektheit der Schlüssel muß daher auf anderem Weg geregelt werden. Bei der Anmeldung zum Homebanking erhält der Kunde einen Ini-Brief, auf dem der Hashwert eines öffentlichen Schlüssels der Bank vermerkt ist. Dieser Brief wird entweder per Post verschickt oder dem Kunden direkt ausgehändigt. Alternativ dazu ist dieser Hashwert meist auf der Homepage des Kreditinstitutes zu finden.

Der Benutzer kennt also den korrekten Hashwert, er braucht nur den Hashwert des erhaltenen Schlüssels zu berechnen und beide zu vergleichen. Stimmen sie nicht überein, so weiß er, daß jemand die Verbindung belauscht und ihm falsche Daten unterschieben möchte.

Andererseits kann jeder Benutzer öffentliche Schlüssel an die Bank übersenden. Nun möchte die Bank ebenfalls kontrollieren, ob diese Schlüssel von dem Kunden stammen, für den sie gelten sollen. Das wird gleichfalls mit einem Ini-Brief geregelt. Das Kreditinstitut schaltet die erhaltenen Schlüssel erst frei, wenn die Hashwerte übereinstimmen.

Der Übertragungsweg per Post scheint zwar nicht gerade sicher zu sein, aber für den Angreifer ist es ungleich schwerer, gleichzeitig die elektronische Kommunikation, den Briefkasten des Benutzers und teilweise sogar die Homepage der Bank zu kontrollieren. Falls der Kunde trotzdem Zweifel hat, kann er außerdem noch direkt mit seinem Kundenberater reden und sich den Ini-Brief der Bank aushändigen lassen bzw. den eigenen persönlich abgeben.

³⁸Das trifft natürlich nur auf RDH zu. Bei DDV werden keine Schlüssel elektronisch ausgetauscht.

5.3.3 Angriffe auf Sicherheitsmedien

Wenn der Angreifer Zugriff auf die Schlüsseldiskette erlangt, so kann er versuchen, selbst Transaktionen durchzuführen. Die darauf abgelegten Schlüssel sind chiffriert, es ist für ihn leichter, das Paßwort zu ermitteln, denn damit kann er auch die Schlüssel auslesen und benutzen. Dafür bietet sich ein Wörterbuchangriff an, da das Paßwort üblicherweise so gewählt wird, daß es sich leicht merken läßt.

Der HBCI-Standard erschwert einen solchen Angriff dadurch, daß das Paßwort Sonderzeichen enthalten muß und aus mindestens acht Zeichen bestehen muß. Der Benutzername kann zwar frei gewählt werden, er bietet aber praktisch keinen zusätzlichen Schutz, denn auf der Diskette werden die Bestandteile des Namens, welche zur Chiffrierung benutzt werden, unverschlüsselt abgespeichert.

Auf der Schlüsseldiskette sind nur die geheimen Schlüssel chiffriert, die öffentlichen Schlüssel und alle weiteren Daten sind im Klartext gespeichert und gegen Manipulation geschützt. Dies wird durch einen Message Authentication Code (siehe Kapitel 3.4.5) gewährleistet. Der Schlüssel für die Chiffrierung wird aus Benutzername und Paßwort generiert. Das heißt insbesondere, daß der Test eines ermittelten geheimen Schlüssels kein Problem mehr darstellt, da der zugehörige öffentliche Schlüssel im Klartext gespeichert ist.

Ein kurze Rechnung zeigt, daß der Aufwand im Verhältnis zum Nutzen eines Angriffs relativ hoch ist: Angenommen, der Benutzer wählt ein Paßwort mit bis zu sieben Buchstaben aus einem Lexikon mit 50000 solchen Wörtern und fügt an einer beliebigen Stelle eines von zehn Sonderzeichen ein. Das heißt, für jedes Wort aus dem Lexikon stehen ihm bis zu 80 Möglichkeiten (zehn Sonderzeichen auf maximal acht Positionen) offen, ein Sonderzeichen einzufügen. Das bedeutet, der Angreifer muß insgesamt vier Millionen Wörter durchprobieren, den Schlüssel berechnen und diesen testen. Selbst wenn er für jeden Test nur eine Zehntelsekunde³⁹ benötigt, so dauert dies etwa 46 Tage. Bis dahin sollte der Benutzer den Schlüssel gesperrt haben.

Das obige Beispiel geht davon aus, daß das Paßwort eine spezielle Struktur besitzt. Wenn der Benutzer ein besseres Paßwort wählt, so ist die Chance eines erfolgreichen Angriffes bedeutend geringer.

³⁹Schneier [9, Seite 535] ging 1996 von einer halben Sekunde je RSA-Entschlüsselung mit 768 Bit aus. Mit den derzeit verfügbaren Computern dürfte dies kaum noch eine Zehntelsekunde dauern.

Die Daten auf der Chipkarte sind sogar nur durch eine PIN geschützt. Dadurch scheint ein Angriff darauf zunächst erfolgreicher zu sein. Ist der Angreifer in Besitz der Chipkarte, so hat er mehrere Möglichkeiten. Einerseits kann er versuchen, die PIN zu raten, dafür stehen ihm drei Versuche pro Karte zur Verfügung. Das kann bei DDV-Chipkarten schon nach kurzer Zeit zu einem Erfolg führen, denn die Wahrscheinlichkeit für ein erfolgreiches Ermitteln der PIN liegt hier bei nur $1/150$. Das heißt, sobald der Angreifer ausreichend viele Chipkarten getestet hat, wird eine dabei sein, deren PIN er ermitteln kann⁴⁰.

Die Nachteile für ihn sind, daß er sich das Opfer nicht gezielt aussuchen kann und daß ihm nur wenig Zeit zur Verfügung steht, da die Karte bzw. die Schlüssel sofort gesperrt werden, sobald der Kunde die Bank über den Diebstahl informiert. Außerdem ist es leichter, mit der Chipkarte direkt an den Geldautomaten zu gehen, denn üblicherweise sind die DDV-Chipkarten gleichzeitig die ec-Karten der Kunden.

Bei den RSA-Chipkarten ist die Wahrscheinlichkeit niedriger, sie liegt bei $3/100000$, da die PIN hier fünfstellig ist und jede PIN gleichwahrscheinlich ist. Außerdem werden RSA-Chipkarten derzeit nicht für Transaktionen an Geldautomaten verwendet.

Alternativ dazu kann der Angreifer versuchen, die Schlüssel auszulesen. Dies ist allerdings mit viel Aufwand verbunden und selbst im Falle eines Erfolges ist die Karte danach nicht mehr zu gebrauchen, denn die Chipkarte bietet mehrere Hardwareschutzmechanismen, die erst überwunden werden müssen. Dabei werden Komponenten der Karte jedoch unweigerlich physikalisch zerstört. Das heißt, der Angreifer benötigt ein Tool, welches die Chipkarte simuliert. Bei RSA-Chipkarten wäre stattdessen auch der Einsatz einer Schlüsseldiskette denkbar.

5.3.4 Angriffe auf Benutzer

Chipkarten und Kartenleser stellen eine sichere Verwahrung der Daten bereit, die Kommunikation zwischen Bank und Kunde ist ebenfalls ausreichend geschützt. Bei dem gesamten Prozeß, der bei einer Transaktion abläuft, gibt es jedoch noch eine Lücke. Die Daten müssen von der Chipkarte über den Rechner des Benutzers an den Bankrechner übertragen werden. Angriffe auf den Rechner des Benutzers oder den Benutzer selbst haben größere Erfolgsaussichten als die oben genannten Attacks, da HBCI nicht vorschreibt, wie diese Lücke zu schließen ist.

⁴⁰Siehe ec-Karten-Urteil vom Oberlandesgericht Hamm [25].

Das heißt, jeder Benutzer muß selbst für die Sicherheit seines Rechners sorgen, sei es durch Virenschutzprogramme oder einen Firewall. Genau diese Lücke nutzt auch der bereits erwähnte Angriff des Computer Chaos Clubs aus⁴¹. Dabei wird ein Trojanisches Pferd auf dem Rechner des Kunden installiert, welches ein paar Daten wie Paßwort, Benutzerkennung oder IP-Adresse des Bankservers ausspioniert. Danach übernimmt ein Rechner, auf dem ein weiteres Kundensystem läuft, via Internet die Kontrolle über das Programm und benutzt es, um auf die Chipkarte des Kunden zuzugreifen. Dabei ist der Rechner des Benutzers nur eine Art Schnittstelle zwischen Chipkarte und dem fremden Rechner, das heißt, im besten Falle bemerkt der Kunde gar nichts von dem Vorgang.

Noch dreister erscheint der folgende Vorgang: Ein Angreifer gibt sich am Telefon als Kundenberater der Bank, oder besser, im Falle eines (eventuell sogar fingierten) Rechnerproblems als Mitarbeiter der Hotline aus. Im Laufe des Gesprächs fragt er den Kunden nebenbei nach der PIN, angeblich um etwas auszutesten. Es gibt sicherlich einige unbedarfte Benutzer, die selbst jetzt noch keinen Verdacht schöpfen und bereitwillig Auskunft geben. Danach ist es einfach, ein paar lukrative Transaktionen zu tätigen.

Auf eine kreative Variante dieses Vorgangs weisen derzeit einige Filialen der HypoVereinsbank mit Aushängen hin. Sie warnen vor über Nacht an den Türen angebrachten Kartenlesern, die angeblich zum Öffnen der Tür zum Vorraum mit den Geldautomaten dienen, aber eine Eingabe der PIN erfordern. Die HypoVereinsbank weist ausdrücklich darauf hin, daß dies dafür nicht notwendig ist und daß mit diesem Trick nur die PIN in Verbindung mit den Kartendaten ausspioniert werden soll.

Die Banken müssen also viel Aufklärungsarbeit leisten, damit solche Angriffe nicht erfolgreich sind. Auch der Verbraucherschutz und andere Institutionen weisen ständig auf Sicherheitsprobleme hin und bieten Lösungsvorschläge an. Aber letztendlich ist es immer der Benutzer selbst, der seinen Rechner und seine Daten schützen muß. Nun ist die Zahl derer, die mit einem Computer im Internet surfen können, aber wenig Ahnung von den dabei existierenden Fallen haben, größer als die Zahl derjenigen, die sich damit auskennen. Es ist aber zu erwarten, daß sich dieses Verhältnis in der Zukunft verbessert.

⁴¹Die Beschreibung des Angriffs ist auf der Homepage des Computer Chaos Clubs zu finden [24].

Kapitel 6

Weiterführende Betrachtungen

6.1 Neue kryptographische Algorithmen

6.1.1 Advanced Encryption Standard

Im Jahre 1997 schrieb das amerikanische National Institute of Standards and Technology (NIST) einen öffentlichen Wettbewerb für einen Nachfolger des in die Jahre gekommenen Data Encryption Standards (DES) aus⁴². Der **Advanced Encryption Standard** (AES) genannte neue Algorithmus sollte eine symmetrische Blockchiffre mit einer Blocklänge von 128 Bit sein, der Schlüssellängen von 128, 192 und 256 Bit unterstützt.

Dabei gab es mehrere Kriterien, die alle Vorschläge erfüllen mußten. Sie sollten sicherer, aber auch schneller als Triple-DES, frei verfügbar und öffentlich getestet sein. Kryptanalyse sollte möglichst nicht darauf anwendbar sein und die Ausgabe sich nicht von Zufallsbits unterscheiden lassen. Nach Möglichkeit sollte der zugrunde liegende Algorithmus einfach und schnell, aber mathematisch fundiert sein. Auch die Flexibilität des Verfahrens sollte Freiraum bieten, zum Beispiel für weitere Schlüssel- und Blocklängen, auch die Implementierung auf Chipkarten sollte einfach machbar sein.

Im Juni 1998 wurden 15 Verfahren zur ersten Runde zugelassen⁴³. Schon im August des gleichen Jahres wurden kryptanalytische Angriffe gegen **DEAL**, **Frog**, **Loki97** und **Magenta** bekannt. Auch **CAST-256**, **Crypton**, **DFC**, **E2**, **HPC** und **Safer+** wurden in der zweiten Runde nicht mehr berücksichtigt.

⁴²Siehe <http://csrc.nist.gov/encryption/aes> [23].

⁴³Eine Aufstellung aller 15 Verfahren mit Beschreibung ist bei Selke [10, Seite 60f.] zu finden.

Drei US-amerikanische Verfahren, **RC6** von Ron Rivest, **Mars** von IBM und **Twofish** von Bruce Schneier und zwei europäische Verfahren, **Rijndael** (Belgien) und **Serpent** (Großbritannien) wurden in dieser zweiten Runde, die von 1999 bis Ende 2001 dauerte, ausführlich geprüft und getestet.

Dabei gab es nur geringe Unterschiede, so daß die Entscheidung relativ knapp ausfiel. Am Ende durften sich Joan Daemen und Vincent Rijmen über den Sieg ihres Verfahrens Rijndael freuen.

6.1.2 Spezifikation von Rijndael (AES)

Rijndael ist ein Blockchiffre mit variablen Block- und Schlüssellängen⁴⁴. Diese können jeweils 128, 192 oder 256 Bits betragen. Das Verfahren benutzt wie DES mehrere Ver- und Entschlüsselungsrunden, in welchen Permutationen und Substitutionen mittels S-Boxen durchgeführt werden. Je nach Block- und Schlüssellänge werden verschieden viele Runden absolviert:

$$\text{Anzahl der Runden} = 6 + \frac{\max(\text{Blocklänge}, \text{Schlüssellänge})}{32}$$

Für jede Runde und für die Initialisierung wird ein Rundenschlüssel benötigt. Aus diesem Grund wird der Schlüssel mittels einer Expansionsfunktion auf $(\text{Blocklänge}) \cdot (\text{Anzahl der Runden} + 1)$ Bits vergrößert. Die Rundenschlüssel werden dann der Reihe nach aus dieser Bitfolge entnommen.

Der Eingabetext und die Rundenschlüssel werden je als zweidimensionales Feld aufgefaßt, welches aus vier Zeilen und $(\text{Blocklänge}/32)$ Spalten besteht. Jedes Element des Feldes repräsentiert dabei ein Byte. Die Bytes des Eingabetextes werden ebenso wie die Bytes der Rundenschlüssel in der Reihenfolge $a_{(0,0)}, a_{(1,0)}, a_{(2,0)}, a_{(3,0)}, a_{(0,1)}, a_{(1,1)}, \dots$ in diese Felder eingetragen.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

Abbildung 6.1: Repräsentation der Eingabe als zweidimensionales Feld

⁴⁴Hier wird nur ein allgemeiner Überblick gegeben. Die genaue Implementierung ist beim NIST [23], bei Rijmen [31] und bei Hunger [19] beschrieben.

Die Verschlüsselung erfolgt in mehreren Schritten. Zuerst wird bei der Initialisierung der erste Rundenschlüssel mit dem Eingabetext mittels \oplus verknüpft. Danach werden die einzelnen Verschlüsselungsrunden durchgeführt, wobei sich die letzte Runde geringfügig von den vorhergehenden unterscheidet.

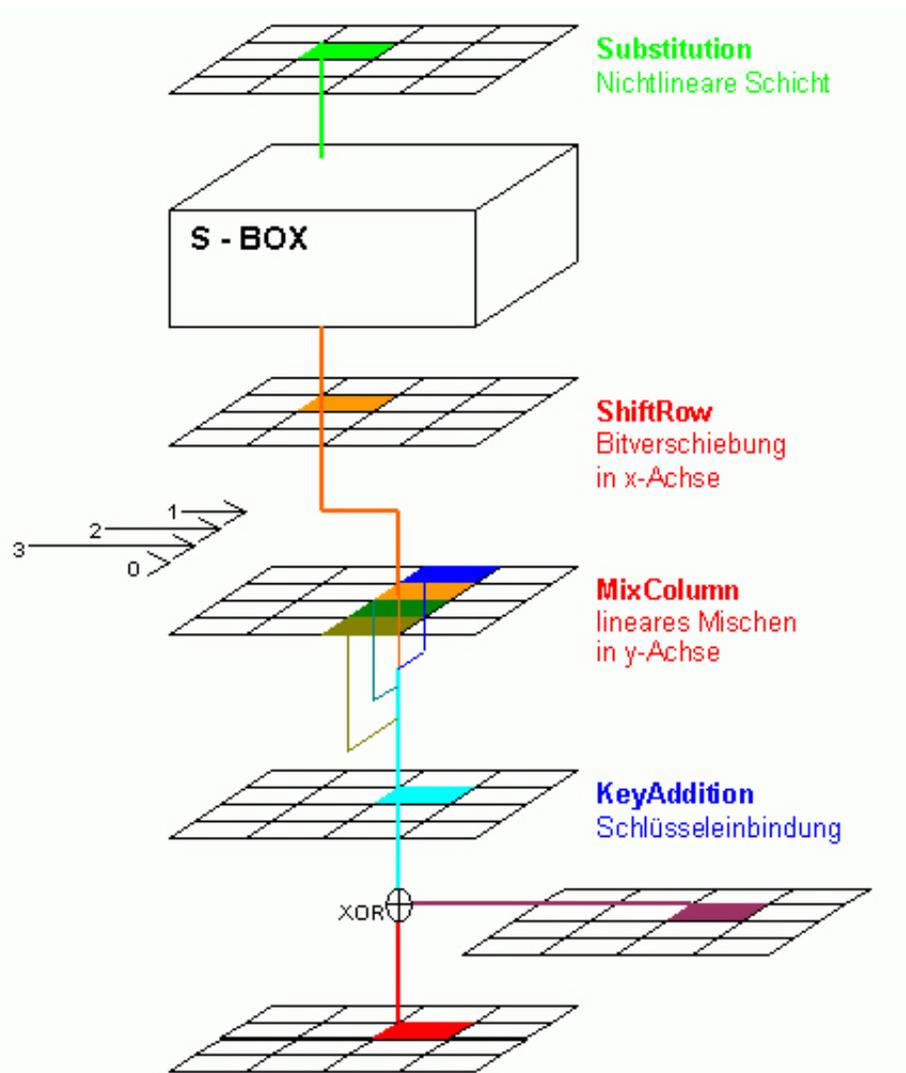


Abbildung 6.2: Ablauf einer Verschlüsselungsrunde nach Hunger [19]

Eine Verschlüsselungsrunde besteht aus vier Operationen, die in dem beschriebenen zweidimensionalen Feld durchgeführt werden:

1. **Substitution:** Jedes Byte des Feldes wird mit einer S-Box nichtlinear umgewandelt.
2. **ShiftRow:** Die Zeilen des Feldes werden je nach Zeile um eine unterschiedliche Anzahl von Stellen zyklisch verschoben.
3. **MixColumn:** Die Spalten des Feldes werden mittels einer Matrixmultiplikation permutiert.
4. **KeyAddition:** Der aktuelle Rundenschlüssel wird Byte für Byte mittels \oplus mit dem Feld verknüpft.

In der letzten Runde wird die MixColumn-Operation weggelassen. Für die Entschlüsselung müssen diese Operationen invertiert und in umgekehrter Reihenfolge abgearbeitet werden.

Rijndael ist relativ einfach gehalten, selbst DES ist komplizierter. Die einzelnen Operationen sind klar getrennt und sind für verschiedene Aufgaben zuständig. So liefert die Substitution die gewünschten Eigenschaften der nichtlinearen Transformationen, MixColumns und ShiftRow sind für eine hohe Diffusion zuständig und KeyAddition bezieht den Schlüssel in die Berechnung mit ein. Somit wird eine hohe Resistenz gegen lineare und differentielle Kryptanalyse und gegen weitere Attacken garantiert. Die einfachen und schnellen Operationen sind auf verschiedenen Systemen und sogar auf Smartcards implementierbar.

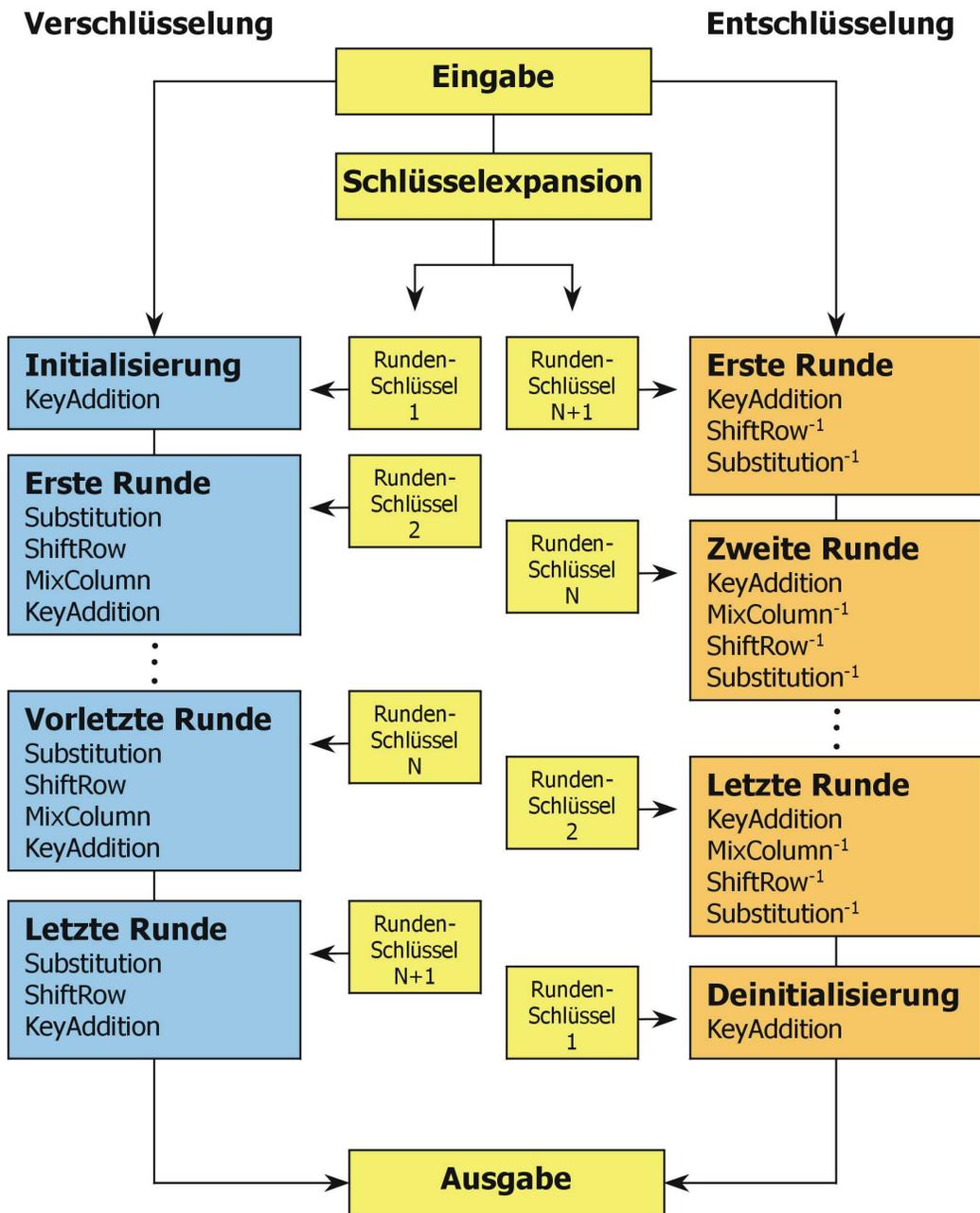


Abbildung 6.3: Ablauf von Ver- und Entschlüsselung bei Rijndael

6.2 Weitere mathematische Methoden

6.2.1 Elliptische Kurven

Der Begriff der elliptischen Kurve wurde im Zusammenhang mit der Bestimmung des Umfangs einer Ellipse eingeführt. In der Kryptographie wird nicht die gesamte Theorie gebraucht, denn nur die elliptischen Kurven über endlichen Körpern finden hier Verwendung⁴⁵. Im folgenden werden nur elliptische Kurven über den endlichen Primkörpern $GF(p)$, $p > 3$ betrachtet. Dazu wird zunächst folgende Gleichung benötigt:

$$y^2z = x^3 + axz^2 + bz^3 \quad \text{mit } a, b \in GF(p). \quad (6.1)$$

Die Diskriminante dieser Gleichung lautet:

$$\Delta = -16(4a^3 + 27b^2).$$

Diese Diskriminante darf nicht Null sein, das ist eine Nebenbedingung für die Wahl der Koeffizienten. Die Gleichung 6.1 hat die Eigenschaft, daß für jede Lösung $(x, y, z) \in GF(p)^3$ auch $c(x, y, z)$ für alle $c \in GF(p)$ eine Lösung dieser Gleichung ist. Diese Lösungen heißen *äquivalent*, falls $c \neq 0$ ist. Die zu (x, y, z) gehörige Äquivalenzklasse wird mit $(x : y : z)$ bezeichnet. Damit läßt sich folgende Definition formulieren:

Definition 6.1 Eine *elliptische Kurve* $E(p; a, b)$ ist die Menge aller Äquivalenzklassen von Lösungen der Gleichung 6.1 mit Ausnahme der Klasse $(0 : 0 : 0)$. Ein Element dieser Menge heißt **Punkt** auf der Kurve.

Ist nun $(\tilde{x}, \tilde{y}, \tilde{z})$ mit $\tilde{z} \neq 0$ eine Lösung der obengenannten Gleichung, dann ist $(x, y, 1) = \tilde{z}^{-1}(\tilde{x}, \tilde{y}, \tilde{z})$ eine dazu äquivalente Lösung, die Äquivalenzklasse dafür ist $(x : y : 1)$. Ist dagegen $\tilde{z} = 0$, dann ist auch $\tilde{x} = 0$ und die Lösungen gehören in diesem Falle alle zur Äquivalenzklasse $(0 : 1 : 0)$. Damit ergibt sich folgende Kurzform zur Beschreibung einer elliptischen Kurve:

$$E(p; a, b) = \{(x : y : 1) \text{ mit } y^2 = x^3 + ax + b\} \cup \{(0 : 1 : 0)\}.$$

Die Äquivalenzklasse $(x : y : 1)$ wird oft nur mit (x, y) bezeichnet, während für $(0 : 1 : 0)$ einfach \mathcal{O} geschrieben wird:

$$E(p; a, b) = \{(x, y) \text{ mit } y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}.$$

⁴⁵Details zu elliptischen Kurven und deren Graphen sind bei Laubrock [20] zu finden.

Durch die geeignete Definition einer Addition von Punkten auf dieser Kurve wird $E(p; a, b)$ zu einer abelschen Gruppe. Der Punkt \mathcal{O} wird dabei zum neutralen Element, das heißt, für jeden Punkt P auf der Kurve gilt:

$$P + \mathcal{O} = \mathcal{O} + P = P$$

Ist nun $P = (x, y)$ ein von \mathcal{O} verschiedener Punkt auf der Kurve, so ist auch $-P = (x, -y)$ ein Punkt auf der Kurve und es wird $P + (-P) = \mathcal{O}$ gesetzt. Sind $P = (x_1, y_1)$ und $Q = (x_2, y_2)$ zwei Punkte auf der Kurve, für die $P \neq \mathcal{O}$, $Q \neq \mathcal{O}$ und $P \neq -Q$ gilt, so wird der Punkt $P + Q = (x_3, y_3)$ wie folgt berechnet:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= -y_1 + \lambda(x_1 - x_3), \end{aligned}$$

wobei λ folgendermaßen definiert ist:

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{falls } P \neq Q, \\ \frac{3x_1^2 + a}{2y_1} & \text{falls } P = Q. \end{cases}$$

Daß diese Addition assoziativ ist, erscheint auf den ersten Blick nicht selbstverständlich. Der rechnerische Vergleich von $P + (Q + R)$ und $(P + Q) + R$ ist sehr aufwendig. Es muß auch beachtet werden, daß P , Q und R Punkte auf der elliptischen Kurve sein müssen. Aus diesem Grund sind nur spezielle Werte für die beiden Koordinaten x_i und y_i erlaubt.

Werden die elliptischen Kurven als Graphen eingeführt, so läßt sich geometrisch eine Addition zweier Punkte dieses Graphen definieren. Das Ergebnis dieser Addition wird dabei durch eine Konstruktion gewonnen. Dies liefert die obigen Rechenregeln für die Berechnung der Koordinaten des Ergebnispunktes. Anhand dieser Konstruktion läßt sich die Gültigkeit von Assoziativität und Kommutativität der so definierten Addition⁴⁶ besser verdeutlichen.

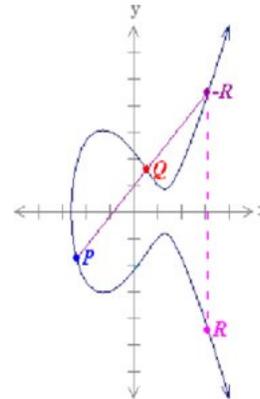


Abbildung 6.4: geometrische Addition von Punkten auf elliptischen Kurven

⁴⁶Ein einfacher geometrischer Beweis ist bei Husemüller [4, Seite 63f.] oder auch bei Laubrock [20] zu finden, die ausführliche Version liefert Silverman [11, Seite 66ff.].

Da $E(p; a, b)$ als endliche abelsche Gruppe aufgefaßt werden kann, läßt sich darin auch ein diskreter Logarithmus definieren. Am einfachsten geht dies, indem die Gruppe multiplikativ geschrieben wird und die in Kapitel 2.2.2 erwähnte Definition benutzt wird.

Elliptische Kurven lassen sich gut klassifizieren. Von Bedeutung sind vor allem die folgenden Klassen:

Definition 6.2 *Gegeben sei eine elliptische Kurve $E(p; a, b)$. N sei die Anzahl der Punkte dieser Kurve.*

- Falls $N = p$ gilt, so heißt $E(p; a, b)$ **anomal**.
- Falls $N = p + 1$ gilt, so heißt $E(p; a, b)$ **supersingulär**.

Für diese beiden Klassen von elliptischen Kurven existieren effiziente Algorithmen zur Berechnung des diskreten Logarithmus. Bei anomalen Kurven wird dafür der nach Smart, Semaev, Satoh und Araki benannte SSSA-Algorithmus benutzt, bei supersingulären Kurven leistet dies der nach Menezes, Okamoto und Vanstone benannte MOV-Algorithmus⁴⁷. Daher dürfen solche Kurven in Kryptosystemen nicht eingesetzt werden. Einfach erkennen lassen sie sich, indem Punkte $P \in E(p; a, b)$ gewählt werden und auf $(p + 1)P = \mathcal{O}$ bzw. $pP = \mathcal{O}$ getestet werden. Die Anzahl der Punkte einer Kurve kann mit dem Theorem von Hasse abgeschätzt werden⁴⁸:

Satz 6.1 (Theorem von Hasse) *Für die Ordnung von $E(p; a, b)$ gilt:*

$$|E(p; a, b)| = p + 1 - t \quad \text{mit} \quad |t| \leq \sqrt{4p}$$

Das heißt, die Kurve $E(p; a, b)$ hat ungefähr p Punkte. Es wird davon ausgegangen, daß eine elliptische Kurve mit 2^{163} Punkten etwa die gleiche Sicherheit wie ein RSA-System mit 1024 Bit bietet. Um eine solche Kurve zu erhalten, wird eine Primzahl p dieser Größenordnung festgelegt und zwei Koeffizienten a und b zufällig gewählt. Danach wird die Anzahl der Punkte auf der Kurve berechnet und auf Primfaktoren getestet. Existiert kein Primfaktor $q \geq 2^{160}$, so wird die Kurve verworfen und es werden neue Koeffizienten a und b gewählt. Das gleiche geschieht auch, wenn die Kurve supersingulär oder anomal ist.

⁴⁷Beide Algorithmen sind bei Werner [15, Seite 82ff.] ausführlich beschrieben.

⁴⁸Siehe Buchmann [2, Seite 196].

6.2.2 EC-Kryptographie

Der diskrete Logarithmus (siehe Kapitel 2.2.2) kann in beliebigen endlichen abelschen Gruppen definiert werden. Die elliptische Kurve $E(p; a, b)$ bildet mit der oben beschriebenen Addition eine solche Gruppe. Aus diesem Grund kann sowohl das ElGamal-Verschlüsselungsverfahren als auch das ElGamal-Signaturverfahren problemlos auf elliptische Kurven umgestellt werden. Dazu muß nur die Punktgruppe über $E(p; a, b)$ statt der multiplikativen Gruppe eines endlichen Körpers ($GF(p)$, $GF(2^n)$, ...) benutzt werden. Es gibt sogar zu RSA analoge Verfahren, welche mit elliptischen Kurven arbeiten.

Kryptosysteme mit elliptischen Kurven sind eine wichtige Alternative zu den anderen asymmetrischen Verfahren. Außerdem sind sie effizienter, da sie mit 163-Bit Schlüsseln auskommen und sich daher auch hardwaremäßig leichter implementieren lassen. Andererseits ist die benötigte Arithmetik etwas komplizierter.

Die in Kapitel 2.4.3 vorgestellte $(p - 1)$ -Methode profitiert ebenfalls von den Eigenschaften der elliptischen Kurven. Die Faktorisierungsmethode mit elliptischen Kurven (ECM) ist eine Verbesserung der Methode von Pollard, die mit beliebig zusammengesetzten Zahlen n funktioniert und am effizientesten ist, wenn n einen kleinen Primfaktor hat.

Kapitel 7

Zusammenfassung

Einige ausgewählte kryptographische Verfahren haben sich im Laufe der Zeit bewährt und werden daher auch in sehr vielen Bereichen benutzt. Auch beim HBCI-Standard wurde dieser Weg eingeschlagen, mit RSA, Triple-DES und weiteren gängigen Verfahren sind die bekanntesten und am besten untersuchten Algorithmen verwendet worden. Die Wahrscheinlichkeit eines erfolgreichen Angriffs gegen diese Verfahren ist sehr gering.

Die bei HBCI verwendeten Protokolle zum Schlüsselaustausch sind weniger ausgefeilt, aber es wurde der bestmögliche Kompromiß zwischen Sicherheit und Wirtschaftlichkeit gewählt. Durch die Einführung neuer Sicherheitsmedien haben es Angreifer auch hier schwer, Erfolge zu verbuchen.

Für den Fall, daß die bewährten Methoden nicht mehr ausreichen oder gar Sicherheitslücken entdeckt werden, stehen mit Rijndael (AES) und ElGamal mit elliptischen Kurven sehr gute effiziente Alternativen zur Verfügung. Optimistische Schätzungen gehen davon aus, daß die Sicherheit der Kommunikation für die nächsten zwanzig Jahre dadurch gesichert ist.

Ein Schwachpunkt des HBCI-Standards ist die Nichtbeachtung der Anfälligkeit des Rechners des Benutzers. Dadurch ist jeder Kunde gezwungen, selbst für die Sicherheit seiner Daten zu sorgen. Die Kreditinstitute beschränken sich hierbei auf die Lieferung von ausführlichen Informationen.

Die Anzahl der Besitzer eines Internetzugangs steigt in großem Maße. In diesem neuen Medium wird mit sehr vielen Leuten kommuniziert und es sollte nicht jeder diese Verbindungen belauschen können. Immer mehr Internetnutzer werden sich daher für die Sicherheit ihrer privaten Daten interessieren.

Dementsprechend ist die Vorgehensweise der Banken, nur Aufklärung zu betreiben, vermutlich ausreichend, vorausgesetzt, daß die Anzahl der Unerfahrenen immer kleiner wird. Alles weitere wird die Zukunft zeigen.

Abbildungsverzeichnis

2.1	Komplexitätsklassen	4
3.1	Electronic-Codebook-Modus	21
3.2	Cipher-Block-Chaining-Modus	21
3.3	Cipher-Feedback-Modus	22
3.4	Output-Feedback-Modus	22
3.5	Schema einer Runde beim DES-Algorithmus	26
5.1	Ablauf des HBCI-Dialogs	46
5.2	Kommunikationswege bei HBCI	47
5.3	Schlüsselaustausch bei Erstinitialisierung RDH	51
6.1	Repräsentation der Eingabe als zweidimensionales Feld	62
6.2	Ablauf einer Verschlüsselungsrunde bei Rijndael	63
6.3	Ablauf von Ver- und Entschlüsselung bei Rijndael	65
6.4	geometrische Addition von Punkten auf elliptischen Kurven	67

Literaturverzeichnis

- [1] A. Beutelspacher, J. Schwenk, K.-D. Wolfenstetter. *Moderne Verfahren der Kryptographie*. Vieweg Verlag, 3. Auflage, Wiesbaden, 1999.
- [2] J. Buchmann. *Einführung in die Kryptographie*. Springer Verlag, 2. Auflage, Berlin, 2001.
- [3] National Bureau of Standards, U.S. Department of Commerce. *Data Encryption Standard*. Federal Information Processing Standards Publication 46, Washington DC, USA, 1977.
- [4] D. Husemöller. *Elliptic Curves*. Springer Verlag, New York, 1987.
- [5] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison Wesley, 2nd Edition, Reading, USA, 1981.
- [6] A. K. Lenstra, H. W. Lenstra Jr., Editors. *Lecture Notes in Mathematics 1554: The Development of the Number Field Sieve*. Springer Verlag, New York, 1993.
- [7] C. Pomerance, Editor. *Cryptology and computational number theory*, AMS Short Course Lecture Notes, Volume 42. American Mathematical Society, Providence, USA, 1990.
- [8] W. Rankl, W. Effing. *Handbuch der Chipkarten*. Hanser, 3. Auflage, München, 1999.
- [9] B. Schneier. *Angewandte Kryptographie*. Addison-Wesley, Bonn, 1996.
- [10] G. Selke. *Kryptographie*. O'Reilly, Köln, 2000.
- [11] J. H. Silverman. *The arithmetic of elliptic curves*. Springer Verlag, New York, 1986.
- [12] R. E. Smith. *Internet-Kryptographie*. Addison-Wesley, Bonn, 1998.

- [13] M. Welschenbach. Kryptographie in C und C++. Springer Verlag, Berlin, 1998.
- [14] D. Welsh. Codes und Kryptographie. VCH Verlagsgesellschaft mbH, Weinheim, 1991.
- [15] A. Werner. Elliptische Kurven in der Kryptographie. Springer Verlag, Berlin, 2002.
- [16] E. Zeidler, Herausgeber. I. N. Bronstein, K. A. Semendjajew, G. Grosche, V. Ziegler, D. Ziegler. Teubner-Taschenbuch der Mathematik. B. G. Teubner Verlagsgesellschaft, Leipzig, 1996.

Online-Literatur

- [17] J. Apel. Kryptographie.
Vorlesung an der Universität Leipzig, 1998.
<http://www.mathematik.uni-leipzig.de/~apel>
- [18] J. Buchmann. Einführung in die Kryptographie.
Vorlesung an der TU Darmstadt, 1997.
<http://www.informatik.tu-darmstadt.de/TI>
- [19] T. Hunger. Der Rijndael (AES) Algorithmus: Eine Visualisierung.
Diplomarbeit, FH Basel, 2001.
<http://www.fhbb.ch/inform/Projekte/AES>
- [20] T. Laubrock. Krypto-Verfahren basierend auf elliptischen Kurven.
Diplomarbeit, FH Dortmund, 1999.
<http://www.laubrock.de>
- [21] A. Menezes, P. van Oorschot, S. Vanstone.
Handbook of Applied Cryptography. CRC Press, 1996.
<http://www.cacr.math.uwaterloo.ca/hac>
- [22] S. Paulus. Kryptographie II.
Vorlesung an der TU Darmstadt, 1998.
<http://www.informatik.tu-darmstadt.de/TI>
- [23] Advanced Encryption Standard (AES)
<http://csrc.nist.gov/encryption/aes>

- [24] Erfolgreicher Hackerangriff des CCC auf HBCI-Homebanking.
<http://pamphlet.ffm.ccc.de/b0t0m/ebanking.html>
- [25] ec-Karten-Urteil des OLG Hamm, 1997
http://userpage.fu-berlin.de/~dittbern/archiv/OLG_Hamm_1997.html
- [26] Homebanking-Client-Interface (HBCI)
<http://www.hbci.de>
- [27] HBCI-Kompendium
<http://www.sixsigma.de/hbci/hbci.html>
- [28] International Financial Exchange (IFX)
<http://www.ifxforum.org>
- [29] International Standardization Organization (ISO)
<http://www.iso.ch>
- [30] Open Financial Exchange (OFX)
<http://www.ofx.net>
- [31] Beschreibung des Rijndael-Verfahrens
<http://www.esat.kuleuven.ac.be/~rijmen/rijndael>
- [32] Beschreibung der Hashfunktion RIPEMD-160.
<http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html>
- [33] RSA Laboratories, RSA Security Inc.
<http://www.rsasecurity.com/rsalabs>

Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, 29. April 2002

Danksagung

Ich danke meinem Betreuer, Herrn Dr. rer. nat. habil. Joachim Apel, für die Überlassung des interessanten Themas sowie für die angeregte Diskussion in der Vorbereitungsphase und während der Ausarbeitung der Diplomarbeit.

Für Korrekturlesen und hilfreiche Tips möchte ich mich bei Pavel Pavlow und Thomas Cherek bedanken.

Ein ganz besonderer Dank gebührt auch meiner Freundin, Katja Nowick, für ihre moralische Unterstützung und für die Hilfe bei der Formulierung komplizierter Zusammenhänge.

Außerdem danke ich meinen Eltern, Ursula und Gerhard Nöbel, daß sie mich stets nach besten Kräften unterstützt haben.