Kryptologie

Rainer Schöpf

Sommersemester 2003

1 Einführung: Was ist Kryptologie

Die Wissenschaft von den Geheimschriften
Seit es Sprache gibt, gibt es vertrauliche Mitteilungen.

$$\text{Romeo} \overset{\text{Ich liebe dich}}{\longrightarrow} \text{Julia}$$

1. Abhörsicherheit: Chiffrieren
2. Integrität: Unversehrtheit und Unveränderbarkeit
3. Authenzität: Gewissheit über den Ursprung
4. Anonymität

1.1 Sichere Übermittlung einer Nachricht

1. Persönlich oder durch einen vertrauenswürdigen Boten („Der Kurier der Kaiserin“)
2. Existenz verheimlichen, zum Beispiel durch
   - Unsichtbare Tinte (Karl May)
   - Verstecken in größerem Text oder im Bild (Steganographie)
3. Verschlüsseln!
   Übermittlung über einen an sich unsicheren Kanal, aber so „chiffriert“, dass niemand Unbefugter sie „dechiffrieren“ kann.

Kryptologie war die Domäne des Militärs. Nur dort gab es ausreichend Motivation und vor allem auch (finanzielle) Mittel, um die damaligen Chiffriermaschinen zu entwickeln („Enigma“). Dies trieb auch die Entwicklung der digitalen Computer an.

1.2 Vorteil der Kryptologie

Kryptologie ist Mathematik, und damit kann man Dinge *beweisen*, zum Beispiel, dass eine Methode eine gewisse Sicherheit bietet. (im Idealfall).

2 Historische, Einfaches und Grundlagen

2.1 Terminologie

- Klartext: a b c d e f ...
- Geheimtext (Kryptogramm): A B C D E F ...
- Chiffrieren
- Dechiffrieren
- Alphabet
  - $\{a, \ldots, z\}$
  - $\{1, \ldots, 26\}$
  - $\{0, \ldots, 25\}$
  - $\{0, 1\}$
  - $\{(a_1, \ldots, a_{64}) | a_i \in \{0, 1\}\}$

2.2 Die Skytala von Sparta

Vor 2500 Jahren benutzte die Regierung von Sparta die sogenannte Skytala: das waren zwei Zylinder mit genau gleichem Umfang. Der Absender einer Botschaft wickelte ein schmales Band aus Pergament spiralförmig darum und schrieb dann der Länge nach die Nachricht darauf. Nur wenn der Empfänger einen Zylinder mit gleichem Umfang hatte, konnte er die Nachricht lesen.

Praktisch bedeutet das, dass der Text spaltenweise in einer bestimmten Anzahl von Zeilen aufgeschrieben wird (Transpositionschiffre).

2.3 Verschiebechiffren: Caesar

Die von Cäsar benutzte Chiffre erhält man, indem man unter das Klartextalphabet das Geheimtextalphabet schreibt, aber um 23 Stellen nach rechts (oder um 3 Stellen nach links) verschoben:

Klartext:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Geheimtext:	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

2.4 Algorithmus und Schlüssel

Chiffrieralgorithmus: Methode, „wie“.
Schlüssel beschreibt wie der Algorithmus verwendet wird.

Algorithmus kann nicht geheim gehalten werden („Chiffriermaschine“), das heisst, die Sicherheit beruht auf der Geheimhaltung des Schlüssels.

Statt der sicherer Übermittlung einer oder mehrerer geheimer Nachrichten ist die sichere Übermittlung des Schlüssels notwendig. Vorteil: der Schlüssel ist kürzer und muss nur einmal übermittelt werden, zum Beispiel rigendwann vorher („Reitender Bote“).

Wichtige Alternative: Public Key Systeme

Aber: es ist nötig, den Schlüssel sicher aufzubewahren.

2.5 Kryptoanalyse der Caesar-Chiffre

Wie kann ich eine verschlüsselte Nachricht „knacken“?

Angenommen, ich weiss oder vermute, es handelt sich um eine Verschiebechiffre.

Methode 1 Systematische Schlüsselsuche (Exhaustive Key Search)
Ich probiere alle Möglichekeiten durch (bei Verschiebechiffre maximal 26): bei k verschiedenen möglichen Schlüsseln maximal k Versuche, im Durchschnit $k/2$.

Methode 2 Statistische Analyse: Häufigkeit der Buchstaben

2.6 Monoalphabetische Chiffrierungen

Eine Chiffrierung heisst *monoalphabetisch*, wenn jeder Buchstabe des Alphabets stets zu demselben Geheimtextbuchstaben chiffriert wird.

(Beispiel)

Jeder monoalphabetischen Chiffre entspricht eine Permutation der Buchstaben des Alphabets und umgekehrt. es gibt also insgesamt $26! \approx 4 \cdot 10^{26}$ davon.

2.7 Mathematischer Exkurs: Modulare Arithmetik

Rechne modulo 26: wir teilen jede Zahl durch 26 wie in der Grundschule:
$1 : 26 = 0$ Rest 1
...
$25 : 26 = 0$ Rest 25
$26 : 26 = 1$ Rest 0

Buchstabe	Häufigkeit	Buchstabe	Häufigkeit
a	6,51%	n	9,78%
b	1,89%	o	2,51%
c	3.06%	p	0,79%
d	5,08%	q	0,02%
e	17,40%	r	7,00%
f	1,66%	s	7,27%
g	3,01%	t	6,15%
h	4,76%	u	4,35%
i	7,55%	v	0,67%
j	0,27%	w	1,89%
k	1,21%	x	0,03%
l	3,44%	y	0,04%
m	2,53%	z	1,13%

Tabelle 1: Häufigkeiten der Buchstaben der deutschen Sprache

Gruppe	Gesamthäufigkeit
e,n	27,18%
i,s,r,a,t	34,48%
d,h,u,l,c,g,m,o,b,w,f,k,z	36,52%
p,v,j,y,x,q	1,82 %

Tabelle 2: Buchstabengruppen

$32 : 26 = 1$ Rest 6
und betrachten nur den Rest. Wir schreiben:

$$\begin{aligned} 32 &= 6 \pmod{26} \\ 26 &= 60 \pmod{26} \end{aligned}$$

Allgemein;

$$p = q \pmod{26} \equiv p - q \text{ ist durch } 26 \text{ teilbar}$$

Grundrechenarten gelten, Vorsicht bei Division!

2.8 Tauschchiffren

Wir ordnen jedem Buchstaben eine Zahl zu:

$$\begin{aligned} a &\rightarrow 1 \\ b &\rightarrow 2 \end{aligned}$$

$$\begin{aligned} &\dots \\ y &\to 25 \\ z &\to 0 \end{aligned}$$

Eine Verschiebechiffre entsteht durch Addition einer Zahl t modulo 26 (nämlich wenn ich um t Stellen verschiebe).

(Beispiel)

Beim Dechiffrieren ziehe ich t ab (modulo 26) oder addiere $26 - t$ (modulo 26).

Wenn ich addieren kann, kann ich auch multiplizieren, zum beispiel mal 2:

```
a b c d e f...
D F H J L...
```

Dies ist nicht eindeutig (weil 26 durch 2 teilbar ist), analog bei mal 13:

```
a b c d e f...
M Z M Z M Z...
```

Bei mal 3:

```
a b c d e f...
C F I L O R...
```

Es geht also bei Multiplikation mit Zahlen, die teiler fremd zu 26 sind:

$$1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25$$

Es gibt also 12 dieser *Multiplikativen Chiffren*.

Durch Kombination mit der Verschiebechiffre ergibt sich die Tasuschchiffre: multipliziere mit s, addiere t.

Tauschchiffre:

Schlüssel Ein paar (s, t) von natürlichen Zahlen < 26, s teilerfremd zu 26.

Chiffre Sei s' eine Zahl, für die gilt: $s \cdot s' = 1 \pmod{26}$, das heisst $s' = 1/s \pmod{26}$.

$$\begin{aligned} x &\to x \cdot s + t \pmod{26} \\ y &\to (y - t) \cdot s' \pmod{26} \end{aligned}$$

Es gibt $26 \cdot 12 = 312$ Tauschchiffren.

2.9 Schlüsselwortchiffre

- Schlüssel: Ein Wort. Doppelt vorkommende Buchstaben werden beim zweiten oder späteren Auftreten entfernt. Das Wort wird als Anfang des Geheimtextalphabets benutzt, dieses wird mit dem restlichen Buchstaben des Alphabets aufgefüllt.

- Chiffrieren/Dechiffrieren: wie gehabt

2.10 Kryptoanalyse

Prinzip von Kerkhoffs:
Die Sicherheit eines Kryptosystem darf nicht von der Geheimhaltung des Algorithmus abhängen; sie beruht *nur* auf der Geheimhaltung des Schlüssels.

Mögliche Attacken:

Known ciphertext attack Der Angreifer kennt (oder hört ab) ein Stück Geheimtext. Davon kann man immer ausgehen.

Known plaintext attack Der Angreifer kennt ein Stück Klartext und das dazugehörige Stück Geheimtext. Kommt durchaus vor.

Chosen plaintext attack Der Angreifer hat Zugang zum Verschlüsselungsalgorithmus und kann selbst gewählte Stücke Klartext verschlüsseln, zum Beispiel, wenn er Zugang zu der Verschlüsselungsmaschine hat.

Das Knacken einer monoalphabetischen Chiffrierung einer natürlichen Sprache (26 Buchstaben) einfach durch die Attacke mit bekanntem Geheimtext, und zwar mittels statistischer Analyse der Buchstabenhäufigkeit und der Bigrammhäufigkeit.

Bigramm	Häufigkeit	Bigramm	Häufigkeit
en	3,88%	nd	1,99%
er	3,75%	ei	1,88%
ch	2,75%	ie	1,79%
te	2,66%	in	1,67%
de	2,00%	es	1,52%

Tabelle 3: Die häufigsten Bigramme im Deutschen

2.11 Moderne Monoalphabetische Chiffrierungen

DES Alphabet: 64-bit Strings, Schlüssel: 56-bit Strings.
(1999: Vollständige Schlüsselsuche in 22 Stunden!).

3DES DES mit k_1, dann mit k_2^{-1}, dann wieder mit k_1. Der Schlüssel ist also 112 bit lang.

AES Schlüssel 128/256 bit.

2.12 Verschleierung der Häufigkeiten

Man sorgt dafür, dass die Geheimtextzeichen alle mit der gleichen Häufigkeit auftreten. Bei der *homophonen* Chiffre werden einem Klartextzeichen mehrere Geheimtextzeichen zugeordnet, und zwar

- kommt jedes Geheimtextzeichen dabei nur einaml vor (sonst wäre die Dechiffrierung nicht eindeutig), und
- wird die Verteilung so gewählt, dass alle Geheimtextzeichen gleich häufig vorkommen.

Beispiel: den Buchstaben a,...,z werden die Zahlen 00,...,99 zugeordnet, und zwar nach folgender Vorschrift:
Beim Chiffrieren wählt man zufällig eines der zugehörigen Zeichen aus.

Buchstabe	Zugeordnete Zeichen	Buchstabe	Zugeordnete Zeichen
a	10 21 52 59 71	n	30 35 43 62 63 67 68 72 77 79
b	20 34	o	02 05 82
c	28 06 80	p	31
d	04 19 70 81 87	q	25
e	09 18 33 38 40 42 53 54 55 60 66 75 85 86 92 93 99	r	17 36 51 69 74 78 83
f	00 41	s	15 26 45 56 6173 96
g	08 12 97	t	13 32 90 91 95 98
h	07 24 47 89	u	29 01 58
i	14 39 46 50 65 76 88 94	v	37
j	57	w	22
k	23	x	44
l	16 03 84	y	48
m	27 11 49	z	64

Tabelle 4: Eine homophone Chiffre

2.13 Die Vigenère-Chiffre

Blaise der Vigenère (1523-1596) machte die Vigenère-Verschlüsselung im Jahre 1586 bekannt. Man benutzt verschiedene monoalphabetische Chiffrierungen im Wechsel. Die Idee ist älter, ähnliche Chiffrierungen wurden schon von anderen im 16. Jhd. veröffentlicht.

Es handelt sich um eine periodische *polyalphabetische* Chiffrierung. Bei einer polyalphabetischen Chiffrierung wird ein Klartextbuchstabe *nicht* jedesmal zu demselben Geheimtextbuchstaben verschlüsselt, aber natürlich nicht wollkürlich. Die Dechiffrierung muss *eindeutig* sein.

Monoalphabetische Chiffre Jeder Buchstabe des Klartextalphabets wird durch genau einen Buchstaben des Geheimtextalphabets ersetzt.

Homophone Chiffre Jedem Buchstaben des Klartextalphabets wird zufällig ein Buchstabe aus einer entsprechenden Gruppe von Buchstaben des Geheimtextalphabets zugeordnet.

Polygram-Chiffre Ganze Buchstabenblöcke des Klartextalphabets werden gemeinsam verschlüsselt, zum Beispiel aba zu RTQ.

Polyalphabetische Chiffre besteht aus mehreren monoalphabetischen Chiffren, die nach jedem Buchstaben des Klartextes gewechselt werden (Enigma). Können heutzutage ziemlich einfach gebrochen werden (Computer).

Warum die doch schon angestaubte Vigenère-Chiffre?

- Prototyp für viele noch heute benutzte Algorithmen (WordPerfect)
- Wichtig als Beispiel
- Kryptoanalyse: Kasiski-Test und Friedman-Test

Verschlüsselung:
Klartext: fachhochschulebingen
Schlüsselwort: BLAU
`BLAUBLAUBLAUBLAUBLAU`
`fachhochschulebingen`
Für jeden Klartextbuchstaben nehmen wir den darüberstehenden Buchstaben des Schlüsselworts. Dieser bestimmt die Zeile, mit der der Klartextbuchstabe verschlüsselt wird.
f : B : Zeile die mit B beginnt, Spalte f, ergibt G
a : L : ergibt L
c : A : ergibt C
h : U : ergibt B
usw.

Schlüsselwort:	`BLAUBLAUBLAUBLAUBLAU`
Klartext:	`fachhochschulebingen`
Geheimtext:	`GLCB...`

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Tabelle 5: Vigenère-Quadrat

Dechiffrieren: Nimm den Schlüsselwortbuchstaben, der über dem Geheimtextbuchstaben steht, dann nimm die Zeile die mit diesem Schlüsselwortbuchstaben beginnt. Gehe in dieser Zeile, vom Geheimtextbuchstaben ausgehend nach oben: G : Zeile B, über G steht f, usw.

2.14 Kryptoanalyse der Vigenère-Chiffre

2.14.1 Kasiski-Test

Wilhelm Kasiski (1805-1881)
Es gibt Perioden mit der Länge des Schlüsselwortes.
Beispiel: Verschlüsselung des Wortes „ein“:

Schlüsselwort:	`BLAUBLAUBLAUBLAUBLAUBLAU`
Klartext:	`ein..ein..ein..ein..ein..ein`
Geheimtext:	`FTN..PIH..ECO..YJY..FTN..PIH`

Es gibt also nur 4 Möglichkeiten, ein Wort zu verschlüsseln, weil das Schlüsselwort 4 Buchstaben lang ist. Außerdem gilt:
Wenn zwei Klartextfolgen einen Abstand haben, der ein Vielfaches der Länge des Schlüsselworts ist, dann sind die zugehörigen Geheimtextfolgen gleich!

Kasiski-Test: Man suche Folgen aus gleichen Buchstaben im Geheimtext und bestimme deren Abstand. Dieser ist (vermutlich) ein Vielfaches der Schlüsselwortlänge.

2.14.2 Der Friedman-Test

William Friedman, der größte Kryptologe aller Zeiten (1891-1969) entwickelte diesen Test 1925.
Frage: Mit welcher Wahrscheinlichkeit besteht ein willkürlich aus dem Klartext herausgegriffenes Buchstabenpaar aus gleichen Buchstaben?
Antwort: Koinzidenzindex.
Wir nehmen eine beliebige Folge von Buchstaben, Länge n. Die Anzahl der a's sei n_1,...,die der z's sei n_{26}. Daraus folgt, dass die Summe all dieser Zahlen gleich n ist. Die Wahrscheinlichkeit, bei zufälliger Auswahl zweier Buchstaben auf gleiche zu treffen, ist

$$I = \frac{\sum_{i=1}^{26} n_i(n_i - 1)}{n(n-1)}$$

Dies ist der Friedmansche Koinzidenzindex.
Was nützt uns das? Wenn wir wissen, wie oft die verschiedenen Buchstaben vorkommen, können wir diesen Index ausrechnen. Sei p_i die Wahrscheinlichkeit für

Buchstabe i. Dann ist die der Koinzidenzindex:

$$I = \sum_{i=1}^{26} p_i^2$$

Zwei Beispiele:

1. Deutsche Sprache, wir kennen die Wahrscheinlichkeiten. Dann ist laut Tabelle 2: $I = 0{,}0762$.

2. Ein zufälliger Buchstabensalat. Dann sind alle $p_i = 1/26$, und die Summe ist

$$I = \frac{1}{26} \approx 0{,}0385$$

Allgemein kann man mit den entsprechenden mathematischen Methoden beweisen, dass der Konizidenzindex um so größer wird, je unregelmäßiger der Text ist. Der Wert $1/26$ ist das absolute Minimum.
Test für monoalphabetische Chiffre: Wenn der Konizidenzindex ungefähr $0{,}0762$, dann ist der Text monoalphabetisch verschlüsselt.
Test für Vigenère-Chiffre: Wenn der Geheimtext die Länge n hat, und die Länge des Schlüsselwortes h ist, dann ist der Konizidenzindex

$$I = \frac{0{,}0377n}{h(n-1)} + \frac{0{,}0385n - 0{,}0762}{n-1}$$

oder, nach h aufgelöst:

$$h = \frac{0{,}0377n}{I(n-1) - 0{,}0385n + 0{,}0762}$$

Mit diesen Methoden ist jede Vigenère-Chiffre leicht zu knacken. Durch diese Tests kennen wir die Länge des Schlüsselwortes. Denn die Buchstaben, die unter dem gleichen Schlüsselwortbuchstaben stehen, werden durch dieselbe monoalphabetische Chiffrierung verschlüsselt. Also schaue ich mit alle Geheimtextbuchstaben aus dieser Gruppe an und bestimme den häufigsten, der dann dem Klartextbuchstaben e entspricht. Das mache ich für alle Buchstaben des Schlüsselwortes, fertig!

2.14.3 Auswege

Erster Ausweg: ich nehme ein sehr langes Schlüsselwort, Beispiel den Text eines Buches. Problem: da das Schlüsselwort aus Sätzen der deutschen Sprache besteht, lassen sich wieder statistische Methoden auf den Geheimtext anwenden (Friedman).

Zweiter Ausweg: ich nehme ein Schlüsselwort, das wieder sehr, sehr lang ist, und aus zufällig gewürfelten Buchstaben besteht (Buchstabenwurm). Ein solches System ist beweisbar sicher (perfekte Sicherheit).

3 Ein bischen Formalismus: Chiffriersysteme

3.1 Chiffriersysteme

Bisher: Sender und Empfänger vereinbaren *einen* Schlüssel, verschlüsseln damit *einen* Klartext. Das ist unrealistisch.
Chiffriersysteme: Menge von Klartexten, Geheimtexten, Schlüsseln
Beispiele:

- Klartexte: Alle Bücher in der Bibliothek der FH Bingen
  Chiffrierungen: Alle 312 affinen Chiffrierungen
  Geheimtexte: alle, die sich aus Anwendung aller 312 Chiffrierungen auf alle Bücher ergeben

- Klartexte: alle lateinischen Ausdrücke in allen Asterix-Bänden, Chiffrierungen alle 26 Caesar-Chiffrierungen und die dazugehörigen Geheimtexte.

Wir haben also immer:

$\mathbf{M}$ Menge aller Klartexte

Menge aller Schlüssel

$\mathbf{C}$ Menge aller Geheimtexte

Wir bezeichnen mit f den Verschlüsselungsalgorithmus; die Spezialisierung auf Schlüssel k mit f_k. Die Abbildungen von $\mathbf{M}$ in $\mathbf{C}$ nennen wir Transformationen; die Menge aller Transformationen $\mathbf{F}$.

Wichtige Eigenschaften der f_k: Umkehrbarkeit: es gibt f_k^{-1} mit

$$f_k^{-1}(f_k(m)) = m, m \in \mathbf{M}$$

paragraphDefinition eines Chiffriersystems (Claude Shannon):
Ein (symmetrisches) Chiffriersystem $\mathbf{S}$ besteht aus einer Menge $\mathbf{M}$ von Klartexten, einer endlichen Menge $\mathbf{C}$ von Geheimtexten und einer Menge $\mathbf{F}$ von umkehrbaren Transformationen von $\mathbf{M}$ in $\mathbf{C}$.

Bemerkungen:

- Verschiedene f können für denselben Klartext den gleichen Geheimtext ergeben.

- Aus der Umkehrbarkeit folgt, dass die Menge $\mathbf{M}$ höchstens genauso viele Elemente enthalten kann wie $\mathbf{C}$: $|\mathbf{M}| \leq |\mathbf{C}|$.

3.2 Sicherheit eines Chiffriersystems

Was ist Sicherheit? Was ist perfekte Sicherheit?

Perfekte Sicherheit: Angreifer hat keine Chance, etwas über das Chiffriersystem zu lernen, das heisst Wahrscheinlichkeit Null.

Sei μ ein Klartext, $\mathbf{p}(\mu)$ die Wahrscheinlichkeit, dass dieser Klartext vorkommt.
Beispiele:

- Klartexte sind Buchstaben des Textes eines deutschen Buches: die Häufigkeiten $\mathbf{p}(\mu)$ sind die Buchstabenhäufigkeiten in Tabelle 1.
- Klartexte sind Buchstabenpaare der Texte eines deutschen Buches: die $\mathbf{p}(\mu)$ sind die Bigrammhäufigkeiten aus Tabelle 3

Die $\mathbf{p}(\mu)$ heissen *a-priori-Wahrscheinlichkeiten* oder *theoretische* Wahrscheinlichkeiten.

Angreifer fängt Geheimtext γ ab. Er kann (prinzipiell) alle Klartexte und alle möglichen Verschlüsselungen durchgehen und analysieren, wie hoch die Wahrscheinlichkeit ist, dass γ von μ kommt. Diese beobachteten oder a-posteriori-Wahrscheinlichkeiten seien $\mathbf{p}_\gamma(\mu)$.
Beispiele:

- Klartexte $\mathbf{M}$ Buchstaben des Textes eines deutschen Buches, Chiffierungen alle 26 Verschiebechiffren. Jeder Geheimtextbuchstabe hat die gleiche Wahrscheinlichkeit, also ist für jedes γ: $\mathbf{p}_\gamma(\mu) = \mathbf{p}(\mu)$ für alle μ.
- Die Klartexte seien die ersten 100 Buchstaben einer jeden Seite des ersten Bandes des großen Brockhaus, $|M|$ = Anzahl der Seiten des ersten Bandes. Chiffierungen alle 26 Verschiebechiffren. Für jeden Klartext ist seine Wahrscheinlichkeit gleich $1/|M|$, $p_\gamma(\mu)$ ist entweder 0 oder 1.

Wenn für ein gewisses μ gilt: $\mathbf{p}_\gamma(\mu) > \mathbf{p}(\mu)$, dann kommt γ mit hoher Wahrscheinlichkeit vom Klartext μ her. Ist umgekehrt $\mathbf{p}_\gamma(\mu) < \mathbf{p}(\mu)$, so kommt γ mit hoher Wahrscheinlichkeit nicht von μ. In beiden Fällen hat der Angreifer etwas gelernt.
Definition: Ein Chiffriersystem $\mathbf{S}$ bietet perfekte Sicherheit, wenn für jeden Geheimtext gilt: $\mathbf{p}_\gamma(\mu) = \mathbf{p}(\mu)$ für alle Klartexte μ. (a-priori gleich a-posteriori Warhscheinlichkeiten)

Verschiebechiffren sind perfekt, wenn sie nur einen Buchstaben verschlüsseln!

Kriterien für perfekte Systeme:

- 1. Kriterium: In einem perfekten System $\mathbf{S}$ kann jeder Klartext auf jeden Geheimtext abgebildet werden.
  Begründung: sonst wären manche der $p_\gamma(\mu) = 0$.

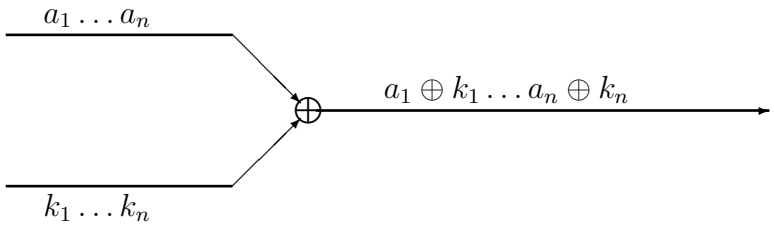
- 2. Kriterium: In einem perfekten System $\mathbf{S}$ gilt: $|\mathbf{F}| \geq |\mathbf{C}| \geq |\mathbf{M}|$.
  Begründung: Damit ein beliebiger Klartext μ in alle möglichen Geheimtexte überführt werden kann, braucht man mindestens soviele Transformationen wie mögliche Geheimtexte.

- 3. Kriterium: Sei $\mathbf{S}$ ein System mit $|\mathbf{F}| = |\mathbf{C}| = |\mathbf{M}|$. Ferner gelte: Es gibt zu jedem Klartext μ und zu jedem Geheimtext γ genau eine Transformation, die μ in γ überführt. Dann ist $\mathbf{S}$ perfekt!

3.3 Das One-Time Pad, ein perfektes System

Wir betrachten folgendes Chiffriersystem:

- Klartexte: alle Buchstabenfolgen der Länge n, $a_1 \dots a_n$.
- Schlüssel: alle Buchstabenfolgen der Länge n, $k_1 \dots k_n$.
- Geheimtext: $a_1 \oplus k_1 \dots a_n \oplus k_n$.



Was bedeutet $\oplus$? Addition gemäß Verschiebechiffre (Caesar), oder anders ausgedrückt: Vigenère-Chiffre mit Schlüsselwort $k_1 \dots k_n$.
Es gilt die erste Voraussetzung des 3. Kriteriums, denn: $|\mathbf{F}| = |\mathbf{C}| = |\mathbf{M}|$.
2.Voraussetzung kann buchstabenweise überprüft werden, also ist dieses System perfekt.
Erfunden wurde dieses *One-Time-Pad* 1917 von Gilbert S. Vernam (1890-1960). (Abreissblock!)

Heutzutage: bits statt Buchstaben, die a_i und k_i sind aus dem Alphabet $(0, 1)$. Als Regel nimmt man binäre Addition,

$$0 \oplus 0 = 0, \quad 1 \oplus 0 = 1, \quad 0 \oplus 1 = 1, \quad 1 \oplus 1 = 0$$

Oder auch: exclusive-or (XOR).

Wichtig Alle möglichen Schlüssel müssen gleich wahrscheinlich sein, deswegen werden die Bits zufällig erzeugt.

Problem Übermittlung der Schlüssel.

Lösung Erzeugung der Schlüsselbits als *pseudozufällige* Folge von Bits durch einen Algorithmus

4 Block- und Stromchiffren

4.1 Was sind Blockchiffren?

Einfache Definition: Eine Blockchiffre verschlüsselt immer ganze Wörter, die alle gleich lang sind, also Folgen der Länge $n$von Buchstaben eines Alphabets.

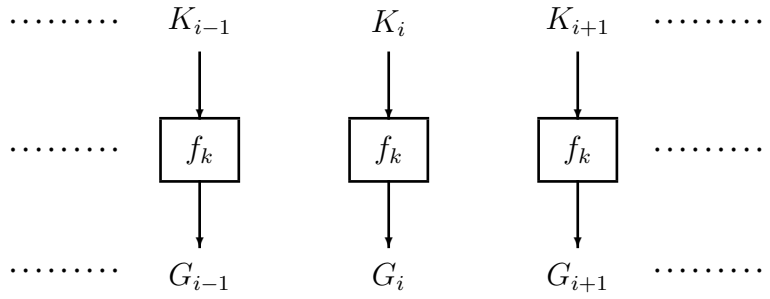
Präzise Definition: Eine Blockchiffre hat als Menge der Klartexte und als Menge der Geheimtexte die Menge aller Wörter der Länge n über einem Alphabet.

Beispiel: DES. Alphabet ist die Menge der Bits $(0, 1)$, Länge ist 64.

4.1.1 ECB-Modus (Electronic Code Book)

ECB: Der Klartext wird in Blöcke der Länge n (64) aufgeteilt. Ist der letzte Block zu kurz wird er mit zufälligen Zeichen aufgefüllt. Jeder Block wird mit einer gegebenen Verschlüsselungsfunktion f_k(mit einem Schlüssel k) verschlüsselt:
Sei K_i der i-te Klartextblock, G_i der i-te Geheimtextblock, dann gilt:

$$\begin{aligned} G_i &= f_k(K_i) \\ K_i &= f_{k^{-1}}(G_i) \end{aligned}$$



Probleme:

- Für einen gegebenen Schlüssel wird derselbe Klartext immer zu demselben Geheimtext kodiert.

- Der Angreifer kann Teile der Nachricht nochmal senden.

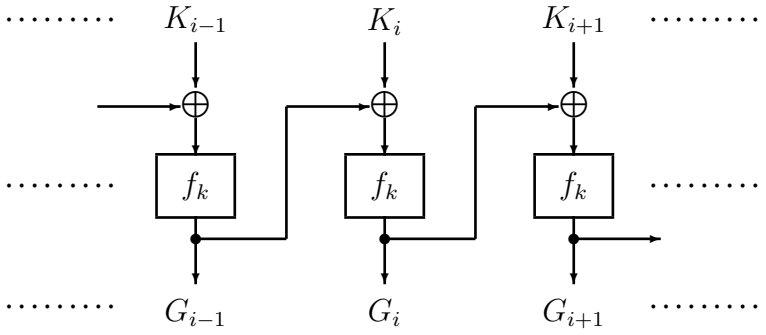
Probleme der simplen Blockchiffre:

- Wenn der Geheimtext zu einem gegebenen Klartext gekannt ist, kann dieser spezielle Block immer kodiert werden. Gefahr am größten zu Beginn und Ende der Nachricht.
- Block Replay

4.1.2 CBC-Modus (Cipher Block Chaining)

CBC: Vor dem Chiffrieren eines Blockes wird dieser mit dem vorhergehenden Geheimtextblock verküpft, üblicherweise mit bitweisem XOR. Zusätzlich wird am Anfang ein zufälliger Block eingefügt (IV, Initialization Vector):
Sei K_i der i-te Klartextblock, G_i der i-te Geheimtextblock, dann gilt:

$$\begin{aligned} G_i &= f_k(K_i \oplus G_{i-1}) \\ K_i &= G_{i-1} \oplus f_{k^{-1}}(G_i) \end{aligned}$$



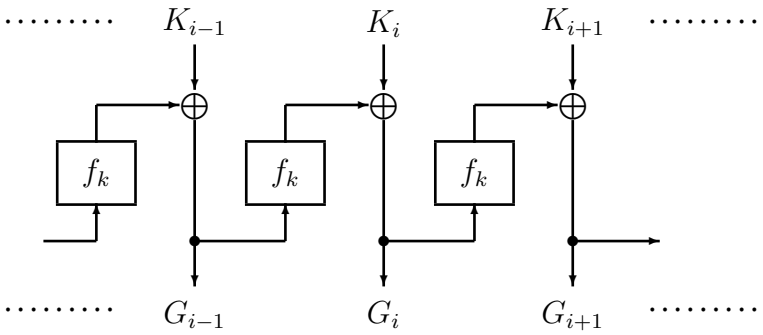
Probleme:

- Anfügen von Blocks durch Angreifer
- Modifikation von Blöcken duch Angreifer
- Sehr lange Nachrichten haben immer noch Muster

4.1.3 CFB-Modus (Cipher Feedback Mode)

CFB: Das Ergebnis der Chiffrierung wird mit dem Klartext verknüpft und ergibt so den Geheimtextblock. Auch hier wird ein IV (G_0) benötigt.

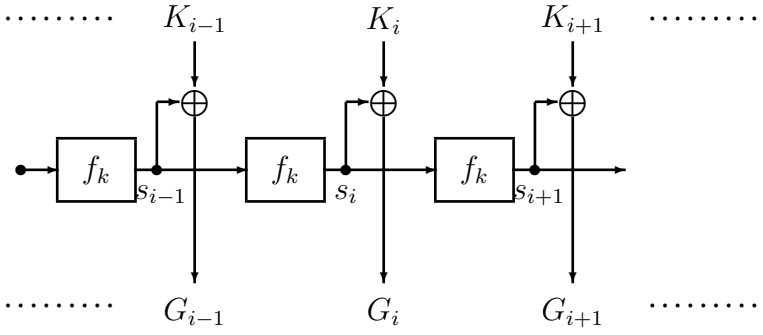
$$\begin{aligned} G_i &= f_k(G_{i-1}) \oplus K_i \\ K_i &= G_i \oplus f_{k^{-1}}(G_{i-1}) \end{aligned}$$



4.1.4 OFB-Modus (Output Feedback Mode)

CFB: Derselbe Block wird mehrfach verschlüsselt, das Ergbnis mit dem Klartext verknüpft (internal Feedback). Auch hier wird ein IV (s_0) benötigt.

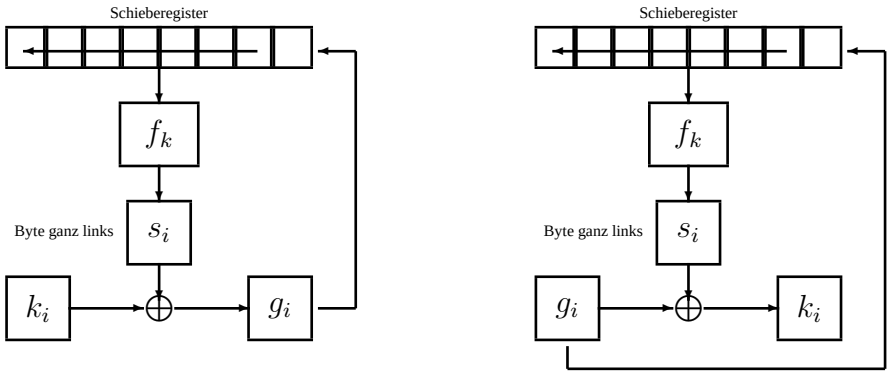
$$\begin{aligned} s_i &= f_k(s_{i-1}) \\ G_i &= s_i \oplus K_i \\ K_i &= G_i \oplus s_i \end{aligned}$$



4.2 Stromchiffren

Eine Stromchiffre codiert den Klartext zum Geheimtext Bit für Bit. Dazu wird ein sogenannter Schlüsselstrom erzeugt: eine Folge von Bits.
Zur Sicherheit hängt der Schlüsselstrom vom einem Schlüssel ab.

Blockchiffren können als Stromchiffren verwendet werden. Da die zu verschlüsselnden Texte kürzer sind als ein Block, müssen zusätzliche Verfahren angewandt werden. Beispiel: CFB wird benutzt, um ein die Daten byteweise zu kodieren (8-bit CFB). Die verwendete Blockchiffre arbeitet auf 64-bit-Blöcken.[1]



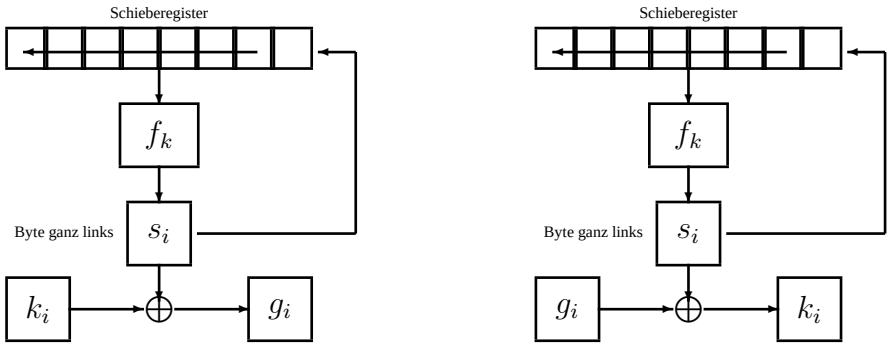
Verschlüsselung Entschlüsselung

8-bit CFB

Zu Beginn wird das Schieberegister mit einem IV gefüllt. Dieser Block wird verschlüsselt und das Byte s_i ganz links (die ersten 8 Bits des verschlüsselten Blocks) per XOR mit einem Byte k_i des Klartextes verknüpft. Das Ergebnis g_i wird gesendet und außerdem in das rechte Ende des Schieberegisters eingefügt, die vorhandenen Bits werden um 8 Postionen nach links bewegt. Das nächste Byte des Klartextes wird in derselben Art und Weise verschlüsselt. Beachte: Für jedes Byte klar- beziehungsweise Geheimtext wird ein kompletter 64-Bit-Block verschlüsselt.

Analog kann auch 8-Bit OFB zur Verschlüsselung benutzt werden. Allerdings zeigt die Kryptoanalyse, dass man OFB nur benutzen sollte, wenn die Blocklänge gleich der Blocklänge des verwendeten Verschlüsselungsalgorithmus ist.

[1] Man könnte auch 1-bit CFB benutzen, um bitweise zu kodieren, allerdings muss dann für jedes Bit ein 64-bit-Block chiffriert werden.



Verschlüsselung Entschlüsselung

8-bit OFB

5 DES und AES

5.1 Geschichte des Data Encryption Standard

- NBS 1973: Öffentlicher Aufruf für Vorschläge für einen kryptografischen Standardalgorithmus. Resonanz groß, aber wenig Expertise.
- NBS 1974: Neuer Aufruf. Ein vielversprechender Kandidat von IBM, basierend auf einem den frühen 1970ern entwickelten ein Algorithmus namens Lucifer. Mit Hilfe der NSA wurde daraus:
- 17.3.1975: Veröffentlichung der Details und Aufruf zum Kommentar: große Resonanz.
- 1976: Zwei Workshops über die Mathematik des Algorithmus und über die Möglichkeit, den Schlüssel länger zu machen. „lively“.
- 23.11.1976: Amerikanischer Standard, der für alle kritische, aber nicht als geheim klassifizierte Regierungskommunikation genutzt werden durfte.
- 15.1.1977: Veröffentlichung der offiziellen Beschreibung für DES.
- Wahrscheinlich ein Missverständnis zwischen NBS und NSA.
- In den folgenden zehn Jahren wurde DES von vielen Organisationen zum kryptografischen Standard gemacht, zum Beispiel von der American Bankers Association, ISO, Autralischer Bank-Standard.

- DES als Standard muss alle fünf Jahre neu evaluiert und bestätigt werden. 1983 ging das einfach so durch. 1987 wollte die NSA (zuständig durch ein von Ronald Reagan unterzeichnete Direktive) nicht wieder zertifizieren. Das kam nicht gut, besonders in der Finanzwelt, und so wurde nach längerer Diskussion DES bis 1992 zertifiziert, angeblich zum letzten Mal.

  1992 gab es immer noch keine Alternative. Zwar waren sich alle einig, dass DES nicht mehr den heutigen Anforderungen entsprach, aber der Algorithmus wurde für weitere fünf Jahre zertifiziert.

- Im Herbst 2000 wurde AES, der Advanced Encryption Standard von NIST veröffentlicht. Ab 26.5.2002 muss AES statt DES für kritische, nicht als geheim klassifizierte Informationen benutzt werden.

- AES ist wieder das Ergebnis eines Aufrufs, bei dem diesmal mehrere Algorithmen eingereicht wurden. Zur allgemeinen Überraschung schaffte es der europäische Vorschlag Rijndael.

- Ein guter Überblicksartikel ist hier.

5.2 Der DES-Algorithmus

DES ist ein Blockchiffre, die Blöcke von 64 Bits verschlüsselt, unter Verwendung eines 56 Bit langen Schlüssels. Der fundamentale Block von DES ist eine Substitution, gefolgt von einer Permutation, abhängig von einem Schlüssel. Dies nennt man eine *Runde* des Algorithmus. DES hat 16 Runden, jede davon arbeitet in gleicher Weise. Die arithmetischen und logischen Operationen sind so gewählt, dass mit der Hardware der 70er leicht zu implementieren waren. Zu Beginn des Algorithmus steht eine Anfangspermutation (Initial Permutation (IP)) der 64 Bits. Nach der Anfangspermutation wird der Block in zwei Hälften zu je 32 Bit aufgeteilt. Dann kommen 16 identische Operationen f, in denen die Daten mit dem Schlüssel kombiniert werden. Dies geschieht durch Bildung von 16 Rundenschlüsseln K_i aus dem Schlüssel k.

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \end{aligned}$$

Danach werden die beiden 32 Bit langen Blöcke zusammengefasst, und die Umkehrung der Anfangspermutation ausgeführt.

5.2.1 Anfangspermutation

Diese Permutation ist immer dieselbe und hat keinen Einfluss auf die Sicherheit der Verschlüsselung. Vermutlich sollte damit das Laden des 64 Bit lange Blocks

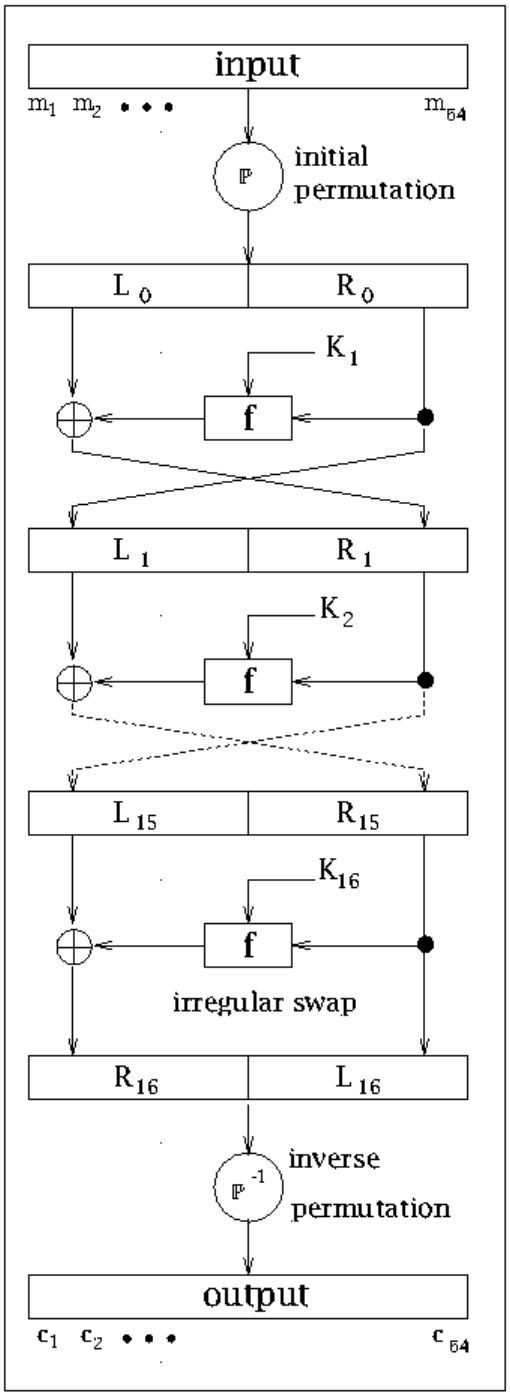


Abbildung 1: DES

in einen DES-Chip leichter gemacht werden, indem man ein Byte grosse Stücke lädt.

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Tabelle 6: Anfangspermutation (IP)

5.2.2 Erzeugen der Rundenschlüssel

Zunächst wird der 64-Bit DES-Schlüssel auf 56 Bitreduziert, indem in einer Permutation jedes achte Bit weggelassen wird (Paritätsbit). Dann wird der entstehen-

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Tabelle 7: DES: Schlüsselpermutation

de Schlüssel in zwei Hälften zu 28 Bit aufgeteilt. dann wird, je nach Runde, für jede der beiden Hälften ein zirkulärer Linksshift um ein oder zwei Bit durchgeführt. Nach dem Shift werden aus den entstandenen 56 Schlüsselbits 48 ausge-

Runde	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Shift	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tabelle 8: DES: Linkshift der Schlüsselbits

wählt. Dies nennt man eine Kompressionspermutation. Durch den Linksshift wird in jeder Runde eine andere Auswahl der Schlüsselbits verwendet. Jedes Bit wird ungefähr in 14 der 16 Runden benutzt, allerdings nicht alle Bits genau gleich oft.

5.2.3 Die Rundenfunktion

Die Rundenfunktion besteht aus folgenden Schritten:

1. Expansionsfunktion E: aus 32 Bits werden 48 Bits.
2. Kombination mit dem Schlüssel durch XOR.

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Tabelle 9: DES: Kompressionspermutation

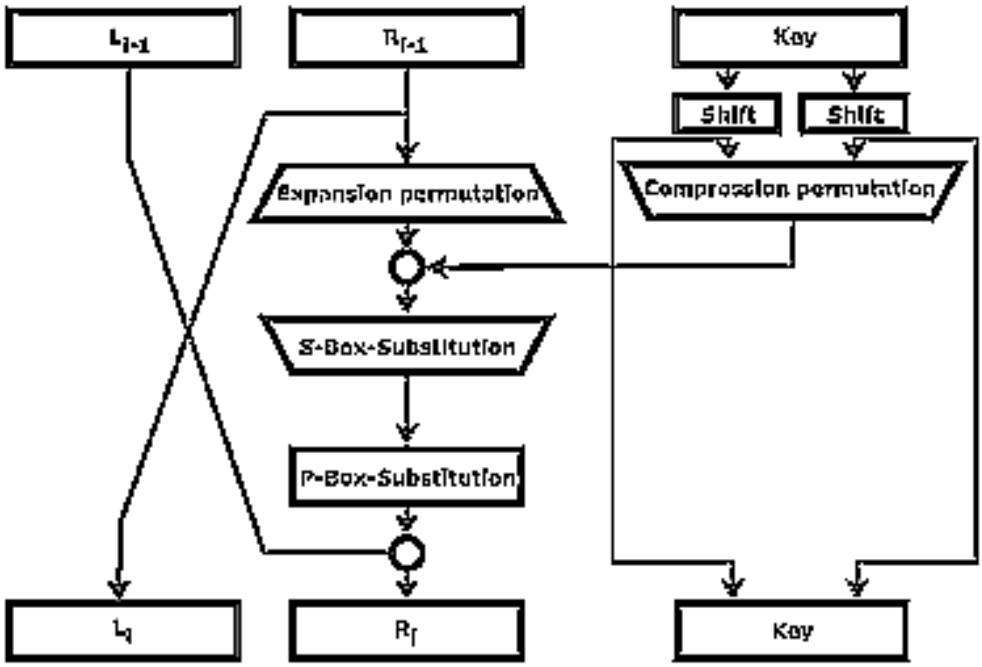


Abbildung 2: Rundenfunktion f

3. S-Box-Ersetzung

4. P-Box-Permutation

Die Expansionsfunktion E operiert auf der rechten Hälfte der Daten R_i und erweitert die 32 Bits zu 48 Bits, die dann per XOR mit dem Rundenschlüssel K_i verknüpft werden. Der Hauptzweck ist allerdings, mit einem Bit mehr als eine Substitution zu beeinflussen. Dadurch verteilt sich der Einfluss der Eingabebits schneller über alle Ausgabebits (Lawineneffekt). Die entstandenen 48 Bit werden nun durch andere 32 Bit substituiert, und zwar in 8 Gruppen von 6 Bit zu 4 Bit. Dazu dienen 8 Substitutionsboxen oder S-Boxen, alle verschieden. Die 32 Bit aus der S-Box-Operation werden nun der sogenannten P-Box permutiert. Dies ist wiederum eine reine Permutation, kein Bit geht verloren und keines wird hinzugefügt.

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

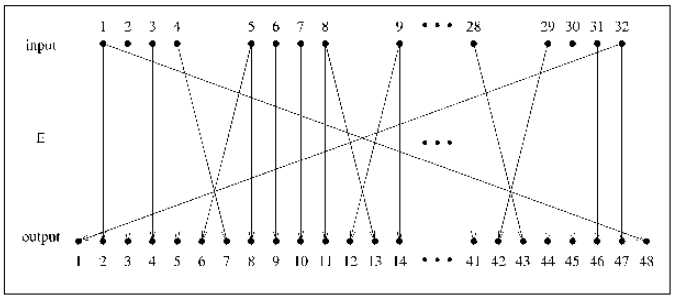


Tabelle 10: DES: Expansionsfunktion

5.2.4 Endpermutation

Die beiden Häften werden *nicht* vertauscht.

5.2.5 Entschlüsselung

Man sollte meinen, die Entschlüsselung funktioniere ganz anders. Im Gegenteil! Der Algorithmus ist genau derselbe, nur die Rundenschlüssel müssen in umgekehrter Reihnefolge angewandt werden, das heisst, statt zirkulärem Linkshift erfolge zirkulärer Rechtsshift, und die Anzahl der Shifts ist in der jeweiligen Runde $0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1$.

5.3 DES in Hardware und Software

Es gibt mittlerweile DES-Chips, die mit einer Gesschwindigkeit in der Größenordnung von GB/s verschlüsseln können!

In Software sind wir (bei üblichen Rechnern) eher noch in der Gegend von Millionen Blöcken/s.

5.4 Sicherheit von DES

- Es wurde vermutet, die NSA hätte eine Hintertür eingebaut. Wahrscheinlich ist das falsch, und der Einfluss der NSA diente dazu, zu verhindern, dass IBM eine Hintertür eingebaut hatte.
- Gewisse Schlüssel sind *schwachte Schlüssel*, weil sie nach der Schlüsselpermutation zu einfachen Bitmustern führen (z.B FFFFFFFF).

```
   Bit   |Bits 2, 3, 4, and 5 form:
 1  6 | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      |-------------------------------------------------
 0  0 |14  4 13  1  2 15 11  8  3 10  6 12  5  9  0  7
 0  1 | 0 15  7  4 14  2 13  1 10  6 12 11  9  5  3  8
 1  0 | 4  1 14  8 13  6  2 11 15 12  9  7  3 10  5  0
 1  1 |15 12  8  2  4  9  1  7  5 11  3 14 10  0  6 13

Bit   |Bits 8, 9, 10, and 11 form:
 7 12 | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      |-------------------------------------------------
 0  0 |15  1  8 14  6 11  3  4  9  7  2 13 12  0  5 10
 0  1 | 3 13  4  7 15  2  8 14 12  0  1 10  6  9 11  5
 1  0 | 0 14  7 11 10  4 13  1  5  8 12  6  9  3  2 15
 1  1 |13  8 10  1  3 15  4  2 11  6  7 12  0  5 14  9

Bit   |Bits 14, 15, 16, and 17 form:
13 18 | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      |-------------------------------------------------
 0  0 |10  0  9 14  6  3 15  5  1 13 12  7 11  4  2  8
 0  1 |13  7  0  9  3  4  6 10  2  8  5 14 12 11 15  1
 1  0 |13  6  4  9  8 15  3  0 11  1  2 12  5 10 14  7
 1  1 | 1 10 13  0  6  9  8  7  4 15 14  3 11  5  2 12

Bit   |Bits 20, 21, 22, and 23 form:
19 24 | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      |-------------------------------------------------
 0  0 | 7 13 14  3  0  6  9 10  1  2  8  5 11 12  4 15
 0  1 |13  8 11  5  6 15  0  3  4  7  2 12  1 10 14  9
 1  0 |10  6  9  0 12 11  7 13 15  1  3 14  5  2  8  4
 1  1 | 3 15  0  6 10  1 13  8  9  4  5 11 12  7  2 14

Bit   |Bits 26, 27, 28, and 29 form:
25 30 | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      |-------------------------------------------------
 0  0 | 2 12  4  1  7 10 11  6  8  5  3 15 13  0 14  9
 0  1 |14 11  2 12  4  7 13  1  5  0 15 10  3  9  8  6
 1  0 | 4  2  1 11 10 13  7  8 15  9 12  5  6  3  0 14
 1  1 |11  8 12  7  1 14  2 13  6 15  0  9 10  4  5  3

Bit   |Bits 32, 33, 34, and 35 form:
31 36 | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      |-------------------------------------------------
 0  0 |12  1 10 15  9  2  6  8  0 13  3  4 14  7  5 11
 0  1 |10 15  4  2  7 12  9  5  6  1 13 14  0 11  3  8
 1  0 | 9 14 15  5  2  8 12  3  7  0  4 10  1 13 11  6
 1  1 | 4  3  2 12  9  5 15 10 11 14  1  7  6  0  8 13

Bit   |Bits 38, 39, 40, and 41 form:
37 42 | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      |-------------------------------------------------
 0  0 | 4 11  2 14 15  0  8 13  3 12  9  7  5 10  6  1
 0  1 |13  0 11  7  4  9  1 10 14  3  5 12  2 15  8  6
 1  0 | 1  4 11 13 12  3  7 14 10 15  6  8  0  5  9  2
 1  1 | 6 11 13  8  1  4 10  7  9  5  0 15 14  2  3 12

Bit   |Bits 44, 45, 46, and 47 form:
43 48 | 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      |-------------------------------------------------
 0  0 |13  2  8  4  6 15 11  1 10  9  3 14  5  0 12  7
 0  1 | 1 15 13  8 10  3  7  4 12  5  6 11  0 14  9  2
 1  0 | 7 11  4  1  9 12 14  2  0  6 10 13 15  3  5  8
 1  1 | 2  1 14  7  4 10  8 13 15 12  9  0  3  5  6 11
```


Tabelle 11: DES: S-Boxen

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Tabelle 12: DES: P-Box Permutation

- Die DES-Verschlüsselung mit verschiedenen Schlüsseln bilden keine Gruppe, sonst wäre mehrfache Verschlüsselung mit verschiedenen Schlüsseln genauso sicher wie einfache.
- Mittlerweile sind Rechner so schnell, dass brute-force Attacken (complete key search) in Stunden möglich sind.
- Differentielle Kryptoanalyse zeigt, dass DES mit weniger als 16 Runden einfacher mit einer Known-Plaintext-Attacke als durch vollständige Schlüsselsuche (Brute-Force-Attacke) gebrochen werden kann.

5.5 Die wirklichen Designkriterien

Nachdem die differentielle Kryptoanalyse bekannt geworden war, veröffentlichte IBM die Designkriterien für die S-Box- und P-Box-Transformationen:

- Jede S-Box hat 6 Bits Eingabe und 4 Bits Ausgabe (Designlimit für Chips im Jahre 1974)
- Kein Outputbit einer S-Box sollte zu ähnlich einer linearen Transformation der Eingabebits sein.
- Hält man das erste und letzte Eingabebit einer S-Box fest und variiert die anderen 4 Bits, so ergibt sich jeder mögliche 4-Bit-Ausgabekombination genau einmal.
- Wenn sich zwei Eingabemuster einer S-Box um 1 Bit unterscheiden, so müssen mindestens 2 Outputbits unterschiedlich sein.
- Wenn sich zwei Eingabemuster einer S-Box nur in den mittleren 2 Bits unterscheiden, müssen sich die zugehörigen Outputs um mindesten 2 Bits unterscheiden.
- Wenn sich zwei Eingabemuster einer S-Box in den ersten beiden Bits unterscheiden und die letzten beiden Bits übereinstimmen, dürfen die zugehörigen Outputs nicht gleich sein.
- Für eine gegebene 6-Bit-Differenz zweier Eingabemuster dürfen nicht mehr als 8 der 32 Paare zur selben Differenz der Outputs führen.

- Ein ähnliches Kriterium, aber für drei S-Boxen.
- Die 4 Ausgabebits jeder S-Box in Runde i werden in der P-Box so verteilt, dass 2 von ihnen die mittleren Bits von S-Boxen in Runde $i+1$ beeinflussen und die anderen 2 die Bits am Ende.
- Die 4 Ausgabebits jeder S-Box beeinflussen 6 verschiedene S-Boxen, es gibt keine 2 Bits, die dieselbe S-Box beeinflussen.
- Wenn ein Outputbit einer S-Box ein mittleres Bit einer anderen S-Box beeinflusst, dann darf ein Ausgabebit dieser S-Box kein mittleres Bit der erten S-box beeinflussen.

Dies ist heutzutage einfach, damals war es hart. Die Leute bei IBM liessen den Computer monatelang nach passenden Transformationen suchen.

5.6 Varianten

- 3DES

5.7 AES/Rijndael

AES/Rijndael ist eine Blockchiffre mit Blocklänge 128 Bits, die Schlüssellänge ist 128, 192 oder 256 Bits. Die Definition ist unter diesem Link veröffentlicht.

5.7.1 Zustandsfeld (State array)

Intern arbeitet DES mit dem sogenannten State Array, das aus 128 Bit (16 Bytes)

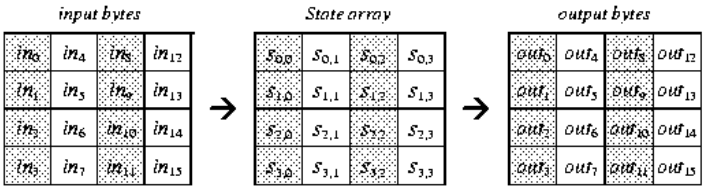


Abbildung 3: Zustandsfeld (State Array)

besteht, angeordnet als 4 Blöcke von je 4 Bytes:

$$s_{i,j}, 0 \leq i, j \leq 3$$

Wenn $e_i, i = 0, \ldots, 15$ die 16 Eingabebytes bezeichnet, so werden diese am Anfang in s kopiert via

$$s_{z,s} = e_{4z+s}$$

Entsprechend wird bei der Ausgabe zurückkopiert:

$$a_{4z+s} = s_{z,s}$$

Die 4 Bytes in jeder Spalte des Zustandsfeldes bilden Worte à 32 Bit:

$$w_0 = s_{0,0}s_{1,0}s_{2,0}s_{3,0} \qquad w_2 = s_{0,2}s_{1,2}s_{2,2}s_{3,2}$$
$$w_1 = s_{0,1}s_{1,1}s_{2,1}s_{3,1} \qquad w_3 = s_{0,3}s_{1,3}s_{2,3}s_{3,3}$$

Bei 128 Bit Schlüssellänge hat der Algorithmus 10 Runden, bei 192 Bit 12 Runden, bei 256 Bit 14 Runden.

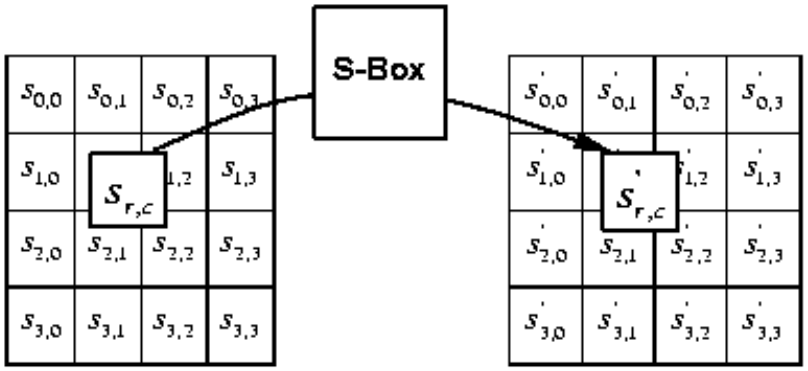
5.7.2 AES: Algorithmus

Zu Beginn wird der Block wie oben angegeben in das Zustandsfeld kopiert. Der Pseudocode für den Algorithmus ist wie folgt (`Nb` ist die Anzahl der 32-Bit-Worte, `Nr` die Anzahl der Runden):

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Die einzelnen Schritte:

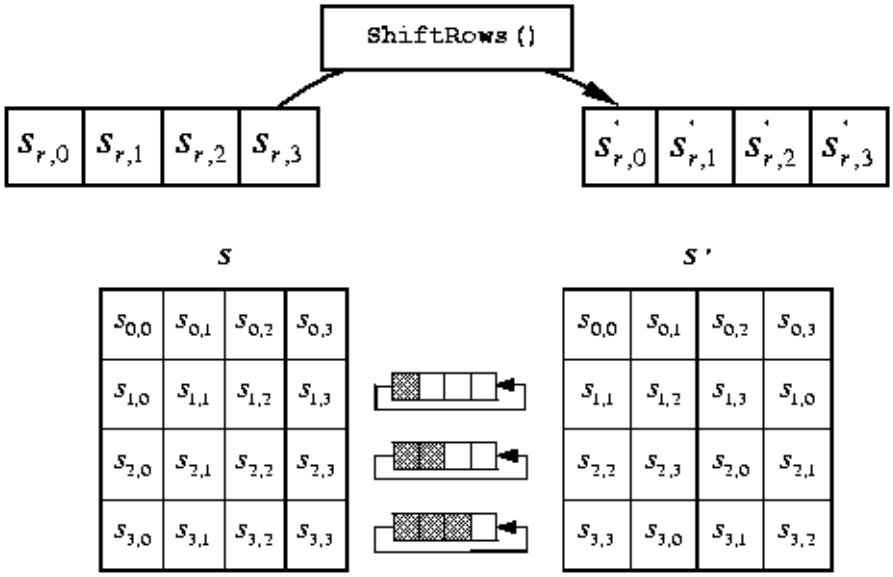
- `SubBytes()` ist eine S-Box-Transformation.



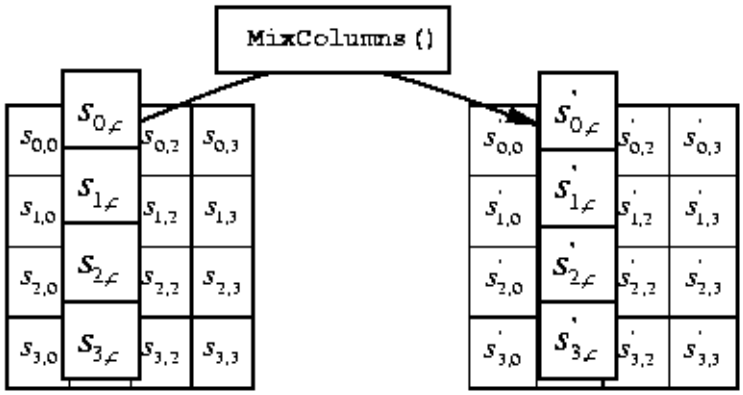
x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

- `ShiftRows()` verschiebt die letzten drei Zeilen des Zustandsfeldes zyklisch um jeweils verschieden viele Bits:

$$s'_{z,s} = s_{z,(s+shift(r,Nb) \bmod Nb} \quad \text{für } 0 < z < 4 \text{ und } 0 \leq c < Nb$$
$$\text{shift}(1,4) = 1;\ \text{shift}(2,4) = 2;\ \text{shift}(3,4) = 3.$$



- `MixColumns()` verändert das Zustandsfeld spaltenweise.



- `AddRoundKey()` verknüpft das Zustandsfeld mit dem Rundenschlüssel per XOR.

5.7.3 AES: Rundenschlüssel

Hier ist der Pseudocode zur Schlüsselexpansion:

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0
```

```
   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk
   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Bemerkung: $Nk = 4, 6$ und 8 sind alle in dem `if` zusammengefasst.

- `SubWord()` bezeichnet wieder die S-Box-Transformation.
- `RotWord()` rotiert ein 32-Bit Wort um 8 Bit nach links.
- Die Felder `Rcon[i]` sind 32-Bit-Konstanten.

Die ersten `Nk` Worte des expandierten Schlüssels stimmen mit dem Schlüssel überein. Jedes weitere Wort `w[i]` ist das Ergebnis einer XOR-Operation des vorhergehenden Wortes `w[i-1]` mit dem Wort `w[i-Nk]`, das `Nk` Worte vorher steht, `w[i-Nk]`, ausser bei Worten, deren Position ein Vielfaches von `Nk` ist, bei denen eine zusätzliche Transformation angewandt wird, gefolgt von einem XOR mit `Rcon[i/Nk]`.

Bei Schlüssellänge 256 Bits und wenn $i + 4$ ein Vielfaches von Nk ist, wird ausserdem vor dem XOR noch ein `SubWord()` auf `w[i]` angewandt.

5.8 Sicherheit von AES

Wie sicher ist AES? Kürzlich (2002) wurde eine Arbeit veröffentlicht, deren Autoren behaupten, sie könnten (zumindest theoretisch), den Schlüssel zu berechnen, wenn sie ein oder zwei Klartexte und zugehörige Geheimtexte kennen (known plaintext attack). Die Komplexität ist allerdings 2^{100}, im Gegensatz zu 2^{128} Operationen, um alle möglichen Schlüssel durch vollständige Schlüsselsuche zu finden.

Das ist noch nicht kritisch. Kritisch wird es erst, wenn diese Methode auch praktisch funktioniert und noch weiter vereinfacht wird.

6 Authentifikation

Bisher haben wir uns mit der Geheimhaltung von Nachrichten beschäftigt, dem Schutz vor einem passiven Angreifer, der Nachrichten Abhören möchte. Jetzt geht es um den Schutz vor einem *aktiven* Angreifer, der Nachrichten aktiv verändern möchte. Es geht insbesondere um drei Punkte

- Nachrichtenintegrität: Wie kann der Empfänger einer Nachricht sicher sein, dass die Nachricht nicht verändert wurde?
- Nachrichtenauthentifikation: Wie kann der Empfänger sicherstellen, dass die Nachricht wirklich vom angeblichen Absender stammt?
- Benutzerauthentifikation: Wie kann eine Person ihre Identität beweisen?

6.1 Kryptografische Prüfsummen

Damit der Empfänger die Unversehrtheit der Nachricht prüfen kann, braucht er zusätzliche Information. Dieser Informationsblock heisst *kryptografische Prüfsumme*, *kryptografischer Fingerabdruck (Fingerprint)*, *Message Authentication Code* (MAC), oder auch *parametrisierte Hashfunktion*.

Erzeugen eines MAC Aus der Nachricht m berechnet der Sender mit Hilfe eines Algorithmus F unter einem geheimen Schlüssel k den Message Authentication Code: MAC $= F_k(m)$.

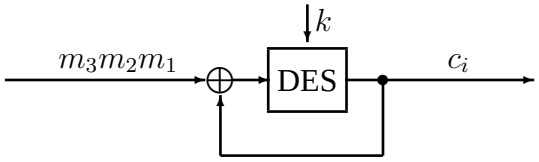
Überprüfen eines MAC Der Empfänger erhält mit der Nachricht m' einen Message Authentication Code MAC$'$. Er überprüft, ob MAC$' = F_k(m')$.
(Sender und Empfänger müssen also vorher einen geheimen Schlüssel vereinbaren.)
Bemerkungen:

- Ein Angreifer kann die Nachricht nicht fälschen, ohne dass der Empfänger es merkt, weil der Angreifer den geheimen Schlüssel k nicht kennt.
- Der Empfänger kann nur erkennen, ob die Nachricht unversehrt ist; stellt er fest, dass sie verändert wurde, kann er das Original *nicht* erzeugen.

Wie sieht das in der Praxis aus? Als Beispiel verwenden wir DES im CBC-Modus: Die Nachricht wird in Blöcke der Länge 64 Bit zerlegt (eventuell mit passenden

Bits aufgefüllt). Dann werden die Blöcke der Reihe nach mit DES-CBC verschlüsselt und der letzte Block des Geheimtextes wird als MAC verwendet.

$$\begin{aligned} c_1 &= f_k(m_1) \\ c_i &= f_k(c_{i-1} \oplus m_i) \end{aligned}$$



MAC mittels DES-CBC

Vorteile:

- MAC hängt von allen Blöcken der Nachricht ab.
- Mac hat eine feste Länge (immer 64 Bit).

Das bringt uns zu der Frage: Wann ist ein MAC-Algorithmus „gut“?

1. Es sollte praktisch unmöglich[2] sein, zu einem gegebenen MAC eine passende Nachricht zu finden („Einweg-Hashfunktion’).
2. Es ollte praktisch unmöglich sein, zwei verschiedene Nachrichten m und m' zu finden, die zum gleichen MAC führen („Kollisionsresistenz“).
   Man unterscheidet:
   (a) Schwache Kollisionsresistenz: Für ein gegebenes m is es praktisch unmöglich, ein passendes m' zu finden, so dass die MACs übereinstimmen.
   (b) Starke Kollisionsresistenz: Es ist praktisch unmöglich, irgendein Paar m, m' mit demselben MAC zu finden.

6.2 Einweg-Hashfunktionen

Eine *Hashfunktion* bildet Strings beliebiger Länge auf Strings fester Länge ab.
Eine *Kompressionsfunktion* bildet Strings fester Länge auf Strings fester, aber geringerer Länge ab.
Eine Funktion h heisst *Einwegfunktion*, wenn es praktisch unmöglich ist, sie umzukehren, das heisst, für ein gegebenes s ein x zu finden, so dass $h(x) = s$.

[2] D.h., es würde sehr sehr lange dauern.

6.3 Die Geburtstagsattacke

Frage: Wie lang muss das Ergebnis einer Einweg-Hashfunktion sein, damit der MAC sinnvoll einsetzbar ist?
Alle solchen Einweg-Hashfunktionen sind prinzipiell mit der *Geburtstagsattacke* angreifbar: Berechne und speichere möglichst viele Nachrichten und deren Hashwerte. Die Attacke erfolgt durch Vergleich eines gegebenen MAC mit den gespeicherten Werten. Damit lässt sich die Frage so formulieren: Wie lang muss das Ergebnis sein, damit es praktisch unmöglich ist, genügend viele Werte auszurechnen und zu speichern? Es muss so viele verschiedene mögliche Hashwerte geben, dass es praktisch unmöglich ist, dass zufällig zweimal der gleiche Wert herauskommt.

Diese Frage lässt sich durch eine einfache statistische Analyse beantworten. Es ist das geliche Problem, wie beim *Geburtstagsparadox*: In einem Raum halten sich k Personen auf. Wie groß ist die Wahrscheinlichkeit, dass zwei von ihnen am gleichen Tag Geburtstag haben?

Etwas allgemeiner formuliert: Es gibt n verschiedene mögliche Geburtstagstage ($n = 366$) und k Personen im Raum. Die Wahrscheinlichkeit, dass die i-te Person ($i = 1, \ldots, k$) am Tag g_i ($g_i = 1, \ldots, n$) Geburtstag hat, ist $1/n$. Für alle Personen zusammen schreiben wir:

$$(g_1, \ldots, g_k), \quad g_i \in \{1, \ldots, n\}$$

Die Wahscheinlichkeit dafür ist $1/n^k$. Ist p die Wahrscheinlichkeit, dass zwei (oder mehr) Personen am gleichen Tag Geburtstag haben, so ist $q = 1 - p$ die Wahrscheinlichkeit, dass alle Geburtstage verschieden sind. Wir rechnen q aus.

Was uns interessiert, ist die Menge E aller $(g_1, \ldots, g_k)$, in denen alle g_i verschieden sind.[3] Da jedes davon die Wahrscheinlichkeit $1/n^k$ hat, ist q gleich $1/n^k$, multipliziert mit der Anzahl der Elemente von E. Wir müssen also die Elemente von E zählen.

In der ersten Position g_1 kann jeder Tag vorkommen, g_i kann also alle n verschiedenen Werte annehmen. Hat g_1 einen bestimmten Wert, so kann dieser für g_2 nicht mehr vorkommen, deswegen gibt es für g_2 nur $n - 1$ mögliche Werte, und so weiter. Insgesamt gilt:

$$|E| = n(n-1)(n-2)\ldots(n-k+1)$$

Die gesuchte Wahrscheinlichkeit q ist

$$q = \frac{1}{n^k}|E| = \frac{1}{n^k}n(n-1)\ldots(n-k+1) = 1(1-\frac{1}{n})\ldots(1-\frac{k-1}{n}).$$

[3] Ist $k > n$, so gibt es gar keine!

Da für alle reellen Zahlen x die Ungleichung $1 + x \leq e^x$ gilt, ist

$$q \leq e^{-1/n} \dots e^{-(k-1)/n} = e^{-k(k-1)/(2n)}$$

Daraus folgt

$$k \leq \frac{1}{2}(1 + \sqrt{1 - 8n \ln q})$$

Also reicht es

$$k \geq \frac{1}{2}(1 + \sqrt{1 - 8n \ln 2})$$

zu wählen, damit $q \leq \frac{1}{2}$ oder $p \geq \frac{1}{2}$ ist. Wenn $n = 366$ und $k = 23$, dann ist die Wahrscheinlichkeit, dass zwei Personen am gleichen Tag Geburtstag haben, größer als $1/2$.

Wenn n sehr groß ist, gilt:

$$k \approx \frac{1}{2}\sqrt{8n \ln 2} = \sqrt{2 \ln 2}\sqrt{n} \approx 1{,}2\sqrt{n}$$

Zurück du unseren Einweghashfunktionen, bei denen die gleiche statistische Analyse gilt: Kann die Hashfunktion N verschiedene Werte annehmen, so reichen

$$k \geq \frac{1}{2}(1 + \sqrt{1 - 8N \ln 2})$$

Werte aus, damit die Wahrscheinlichkeit für zwei gleiche Hashwerte größer als $1/2$ ist. Nehmen wir an, dass die Hashfunktion Werte mit n Bits berechnet, so ist $N = 2^n$, also

$$k \geq f(n) = \frac{1}{2}(1 + \sqrt{(1 + (8 \ln 2)2^n} \approx 1{,}2\sqrt{2^n} \approx 2^{n/2}.$$

Heutzutage wählt man daher $n \geq 128$. Für den digitalen Signaturstandard wird sogar $n \geq 160$ verlangt.

6.4 Beispiele aus der wirklichen Welt

6.4.1 MD4

MD4 (Message Digest 4) wurde 1990 von Ron Rivest vorgestellt. Der Output ist ein 128-Bit-Hashwert.

Designkriterien

Sicherheit: Es ist praktisch unmöglich, zwei Nachrichten zu finden, die denseben Hashwert haben. Keine Attacke ist besser als Brute Force.

Direkte Sicherheit: Die Sicherheit von MD4 beruht *nicht* auf unbewiesenen Annahmen, wie: es ist schwierig, große Zahlen zu faktorisieren.

Geschwindigkeit: MD4 ist geeignet für schnelle Implementationen in Software. Es beruht auf einfachen Operationen auf 32-Bit-Operanden.

Einfachheit und Kompaktheit: MD4 ist so einfach wie möglich, ohne große Datenstrukturen oder ein kompliziertes Programm.

Favorisiert Little-Endian-Architekturen: MD4 ist optimiert für Mikroprozessorarchitekturen (insbesondere Intel).

Kurz danach wurden Attacken auf Teile des Algorithmus veröffentlicht. Obwohl sich diese nicht auf den vollen Algorithmus erweitern lassen, verbesserte Ron Rivest MD4 zu M5.

6.4.2 MD5

MD5 ist im Design ähnlich zu MD4, aber komplizierter. MD5 verarbeitet die EIngabe in Blöcken von 512 Bits, und zwar in (16) Subblöcken zu je 32 Bit.

Zunächst wird die Nachricht auf ein Vielfaches von 512 Bit verlängert. Die letzten 64 Bit sind dabei die Länge der Nachricht (als 64-Bit-Zahl); zwischen der eigentlichen Nachricht und diesem Längenfeld wird mit genau einem 1-Bit und so vielen 0-Bits wie nötig aufgefüllt.

Dann werden die einzelnen 512-Bit-Blöcke verarbeitet, und zwar in jeweils 4 Runden:

(Bild)

Diese Operation wird solange wiederholt, bis alle Eingabeblöcke abgearbeitet sind. Das Ergebnis ist dann der 128-Bit-Hashwert ABCD.

Eine Runde sieht folgendermaßen aus:

(Bild)

Dabei bedeutet `<<<s` einen zirkulären Linksshift um s Bit. Die Werte A, B, C, D werden zyklich auf a, b, c, d abgebildet.

Problem: Es wurde nachgewiesen, dass es möglich ist, Kollisionen zu erzeugen, also gleiche Hashwerte für verschiedene Nachrichten. Das hat (bisher) keine praktische Bedeutung, aber es verletzt eines der weiter oben genannten Designkriterien.

6.5 SHA (Secure Hash Algorithm)

SHA ist Teil des Sceure Hash Standard (SHS), der von NIST (mit Hilfe der NSA) zur Benutzung für den Digital Signature Standard entworfen wurde. Es heisst, die Designprinzipien seien „ähnlich wie bei MD4“; sie wurden aber nicht im Detail veröffentlicht. SHA erzeugt einen 160-Bit-Hashwert.

Wie bei MD5 wird die Nachricht zunächst auf ein Vilefaches von 512 Bit verlängert. Die Hauptschleife besteht wiederum aus 4 Runden, die allerdings komplizierter sind als bei MD5:

(Bild)

Es ist keine kryptografische Attacke gegen SHA bekannt.

6.6 Benutzerauthentifikation

Wie kann sich eine Person (eine Benutzerin) gegenüber einem Rechner (Zielsystem) ausweisen? Die naive Methode benutzt einen Benutzernamen und ein Passwort, das nur der Benutzerin bekannt ist.

Probleme:

1. Benutzername und Passwort werden im Klartext übertragen und können abgehört werden. Lösung: Verschlüsselung
2. Selbst verschlüsselter Verkehr ist protokoliier- und wiederholbar („Replay Attack“). Mögliche Lösungen: Wechsel der Schlüssel, Zeitstempel (Timestamps), Einmalpasswörter (Beispiel: TANs)
3. Ein Angreifer könnte sich den Schlüssel beschaffen, damit ist die gesamte Methode unsicher.
4. Die Passwörter sind auf dem Zielsystem abgelegt. Lösung: lege Hashwerte der Passwörter ab, damit sind ide Passwörter auf dem Zielsystem nicht entschlüsselbar.
5. Das Zielsystm ist nicht identifizierbar, ein aktiver Angreifer könnte den Benutzerinnen die Identität des Zielsystem vorspiegeln und so in den Besitz der Passwörter kommen.

Wie kann man sich vor der Replay-Attacke schützen? Eine mögliche Lösung heisst Challenge-Response-Methode und funktioniert so:

1. Das Zielsystem schickt ein zufällig gewähltes Bitmuster RAND an die Benutzerin, die sich authentifizieren will.

2. Die Benutzerin wendet einen Algortihmus auf RAND an, der von ihrem Passowrt abhängt, $f_k(\text{RAND})$, und shcickt das Ergebnis an das Zielsystem.

3. Das Zielsystem berechnet ebenfalls $f_k(\text{RAND})$ und vergleicht.

Das sieht so aus, als müsste doch das Passwort auf dem Zielsystem hinterlegt sein. Nein! Denn k kann zum Beispiel der Hashwert des Passwortes sein. Allerdings nützt auch das nichts, wenn ein Angreifer das Zielsystem einmal ausgespäht hat.

Nach wie vor ist dies nur einseitige Authentifikation, das Zielsystem könnte vorgespiegelt sein. Dies kann man durch gegenseitige Challenge-Response lösen.

Praktische Anwendungen: GSM-Handy, dort erfolgt eine Challenge-Response-Authentifizierung des Handy mit Hilfe der SIM-Karte. Erst bei UMTS authentifiziert sich auch das Netz gegenüber dem Handy.

6.7 Zero-Knowledge-Protokolle

Alle bisher vorgestellten Systeme beruhen mehr oder weniger auf dem sogenannten Prinzip des Gemeinsamen Geheimnisses (Shared Secret): es gibt eine (geheime) Information, die beiden Seiten, Benutzerin wie Zielsystem, bekannt ist.

Kann ich das auch ohne gemeinsames Geheimnis machen? Kann ich einen Kommunikationspartner davon überzeugen, dass ich etwas weiss, ohne ihm das Wissen selbst mitzuteilen. Ja! Die Antwort liefen die *Zero-Knowledge-Protokolle*.

Beispiel: Circes Geheimnis

Nach n Versuchen, ist die Wahrscheinlichkeit, das Circe das Geheimnis nicht kennt 2^{-n}.

6.7.1 Das Fiat-Shamir-Protokoll

Das Fiat-Shamir-Protokoll wurde 1986 von Adi Shamir und Amos Fiat vorgestellt. Es wurde für den Einsatz auf Chipkarten entwickelt und beruht darauf, dass es äußerst schwierig ist, die modulare Quadratwurzel einer großen Zahl v zu berechnen.

Seien n, r positive ganze Zahlen. Gesucht ist eine Zahl s mit der Eigenschaft $s^2 = v \pmod{n}$. Wenn n genügend groß ist (200–300 Dezimalstellen bzw. 1024 Binärstellen), dann ist die Kenntnis von s ein Geheimnis, dass man gut hüten sollte.

Zu Beginn wählt man in einer authentisierungszentrale n als Produkt zweier ungefährt gleich langer Primzahlen p, q: $n = p \cdot q$. p und q werden geheim gehalten![4] Diese Zahlen müssen so groß sein, dass es praktisch unmöglich ist, p und q als Primfaktoren von n zu berechnen. n wird öffentlich gemacht.

Für jeder Benutzer wird jetzt eine Zahl s bestimmt, die das Geheimnis des Benutzers ist, und daraus $v = s^2 \pmod{n}$ berechnet.

Benutzerin Alice beweist folgendermaßen, dass sie s kennt, ohne die geringste Information über s selbst preiszugeben:

1. Alice wählt zufällig eine Zahl r, die teilerfremd zu n ist, berechnet $x = r^2 \pmod{n}$, und sendet diese Zahlt x an das Zielsystem.
2. Das Zielsystem wählt zufällig eine Bit b aus[5] und sendet es an Alice (Challenge).
3. Alice berechnet:
   (a) Im Fall $b = 1$: $y = r \cdot s \pmod{n}$,
   (b) Im Fall $b = 0$: $y = r \pmod{n}$,

   und sendet y an das Zielsystem.
4. Das Zielsystem berechnet $y^2 \pmod{n}$ und prüft
   (a) Im Fall $b = 1$: $y^2 \pmod{n} = (r \cdot s)(r \cdot s) \pmod{n} = r^2 s^2 \pmod{n} = x \cdot v \pmod{n}$,
   (b) Im Fall $b = 0$: $y^2 \pmod{n} = r^2 \pmod{n} = x \pmod{n}$.

Analyse:

1. Die Challenge st die Frage nacht entweder r oder $r \cdot s$. Nur wenn Alice s kennt, kann sie beide Fragen richtig beantworten. Kennt sie s nicht, ist die Fehlerwahrscheinlichkeit pro Schritt $1/2$, nach n Schritten 2^{-n}.
2. Beide Seiten müssen nur einfache Operationen durchführen.
3. Das Zielsystem verwendet nur öffentliche Informationen, nur Alice kennt s.
4. s selbst bleibt geheim.
5. Das Verfahren beruht darauf, dass es praktisch unmögliche ist, die modulare Quadartwurzel zu berechnen.

[4]Denn: wenn man die Quadratwurzeln $pmodp$ und $\pmod{q}$ kennt, kann man die Quadratwurzel $\pmod{()pq}$ daraus sozusagen zusammensetzen.

[5]dies entspricht der Wahl Odysseus, ob er rechts ohder links ruft

6.8 Chipkarten

Eine Chipkarte ist eine Plastikkarte mit einem eingebetteten Minirechner. Damit kann zum ersten Mal Sicherheit auf der kryptografischer Basis realisiert werden, und zwar für jedermann, nicht nur für Experten! Das liegt am Zusammentreffen zweier Eigenschaften:

- Chipkarten sind ideal für Kryptografie: Speicherung geheimer Schlüssel, Ausführung kryptografischer Algorithmen
- Chipkarten sind ideal für Menschen: einfach zu benutzen (PIN)

Zugangskontrolle mit Chipkarte

Einkaufen mit der Chipkarte

6.9 Anwendungen in der realen Welt: MS-CHAP und Kerberos

6.9.1 PPTP und MS-CHAP

Das Point-to-Point Tunneling Protocol (PPTP) ist ein Protokoll von Microsoft für sichere Kommunikation über (unsichere) TCP/IP-Verbingungen.[6] Die ursprüngliche (NT-)Implementation dieses Protokolls war unsicher, sowohl bei der Authentifikation als auch bei der Verschlüsselung. Ich bschränke mich hier auf die Authentifikation mit MS-CHAP. Näheres siehe hier.

Der PPTP-Server läuft auf NT 4 oder 5 (W2K) Server. Es gab zunächst drei Authentifizierungsmethoden:

1. Passwort wird im Klartext übertragen,
2. Passwort-Hash wird übertragen,
3. Challenge-Response mittels MS-CHAP.

Nur bei der dritten Methode wurde die Verbindung verschlüsselt aufgebaut.

[6] Im Verbindungsassistenten nennt sich das einfach VPN.

Passwort-Hashes Es wurden gleichzeitig zwei verschiedene Hash-Funktionen verwendet, vermutlich der Rückwärtskompatibilität wegen:

1. Lan-Manager-Hash:

   (a) Das Passwort wird durch Abschneiden oder Auffüllen mit 0-Bytes auf 14 Byte Länge gebracht.

   (b) Alle Kleinbuchstaben werden in die entsprechenden Großbuchstaben umgewandelt.

   (c) Die zwei Hälften à 7 Byte, also 56 Bits werden als DES-Schlüssel benutzt, um festgelegte Konstanten (!) zu verschlüsseln.

   (d) Die beiden jeweils 8 Byte langen Ergebnisse ergeben zusammen den 16 Byte langen LAN-Manager-Hashwert.

   Dies ist ein sehr schwacher Hash! Zum Beispiel: ist das Passwort nicht länger als 7 Buchstaben, dann ist der besteht der zweite DES-Schlüssel aus lauter 0-Bits, und damit sind die letzten 8 Byte des LAN-Manager-Hashes konstant!

2. NT-Passwort-Hash: Wandle das Passwort (das wiederum aus maximal 14 Zeichen bestehen darf) nach Unicode um und berechen daraus den 16 Byte langen MD4-Hash.

Problem: es werden *immer* beide Hashwerte benutzt. Damit ist das Verfahren nur so gut wie der schwächere LAN-Manager-Hash.

Das MS-CHAP-Verfahren (Version 1)

1. Der Klient fordert ein Login an.

2. Der Server sender eine zufällige 8-Byte-Challenge.

3. Der Klient berechnet den Lan-Manager-Hash des Passwortes, fügt 5 0-Bytes hinzu, um drei 56 Bit lange DES-Keys zu bekommen. Die Server-Challenge wird mit jedem dieser drei DES-Keys verschlüsselt, die Ergebnisse werden zu einer 24 Byte langen Antwort zusammengefasst. Das gleiche Verfahren wird mit dem NT-Passwort-Hash durchgeführt. Die Ergebnisse gehen zurück an den Server.

4. Der Server führt dieselben Operationen durch und vergleicht, aber *nur* mit *einem* der beiden 24 Byte langen Ergebnisse, abhängig von einem Flag im Antwortpaket des Klienten. Der andere 24-Byte-Block wird ignoriert.

Probleme:

1. Es reicht aus, den schwachen LAN-Manager-Hash zu attackieren.

2. Weil drei Teile unabhängig voneinander verschlüsselt werden, kann man das Protokoll selbst attackieren:

   Ist das Passwort maximal 7 Zeichen lang, so sind die letzten 8 Byte des LAN-Manager-Hashes konstant. Daher bestehen die letzten 8 Byte des 24-Byte langen Antwortblocks aus dem Challenge, das mit dieser Konstanten verschlüsselt ist!

3. Der Server ist nicht authentisiert.

Kurz gesagt, das Protokoll selbst ist so unsicher, dass es mit einfachen Mitteln möglich ist, dem Server eine falsche Identität vorzutäuschen. Dazu kommt dann noch, dass die Verschlüsselung des Datenverkehrs auch nicht sicher war. Microsoft hat dann nachgelegt und eine neue Version entwickelt, die die Probleme vermeidet: MS-CHAPv2.

Das MS-CHAP-Verfahren (Version 2)

1. Der Klient fordert ein Login an.

2. Der Server sendet eine zufällige 16 Byte lange Challenge.

3. Der Klient erzeugt ebenfalls eine 16 Byte lange Zufallszahl (“Peer Authentication Challenge”). Aus der Server-Challenge, der Peer Authentication Challenge und dem Benutzernamen des Klienten wird mittels SHA-1 ein 8 Byte langer Hashwert erzeugt.

   Wie bei MS-CHAPv1 werden aus dem NT-Passwort-Hash drei DESSchlüssel erzeugt, die diesen Hashwert verschlüsseln. Der Klient sendet dem Server die Peer Authentication Challenge und das 24 Byte lange Ergebnis der Verschlüsselung.

4. Der Server führt dieselben Berechnungen durch und prüft die Gültigkeit. Ist das Ergebnis richtig, erzeugt er aus der Peer Authentication Challenge un dem NT-Passwort-Hash eine 20 Byte lange “Peer Authenticator Response” und sendet sie an den Klienten.

5. Der Klient führt die gleichen Operationen durch und prüft das Ergebnis. Damit wird sichergestellt, dass auch der Server das NT-Passwort-Hash kennt.

Probleme:

- Warum so kompliziert?
- NT-Passwort-Hash ist durch Brute-Force-Attacke knackbar.
- Schwache Passworte sind nach wie vor leicht knackbar.

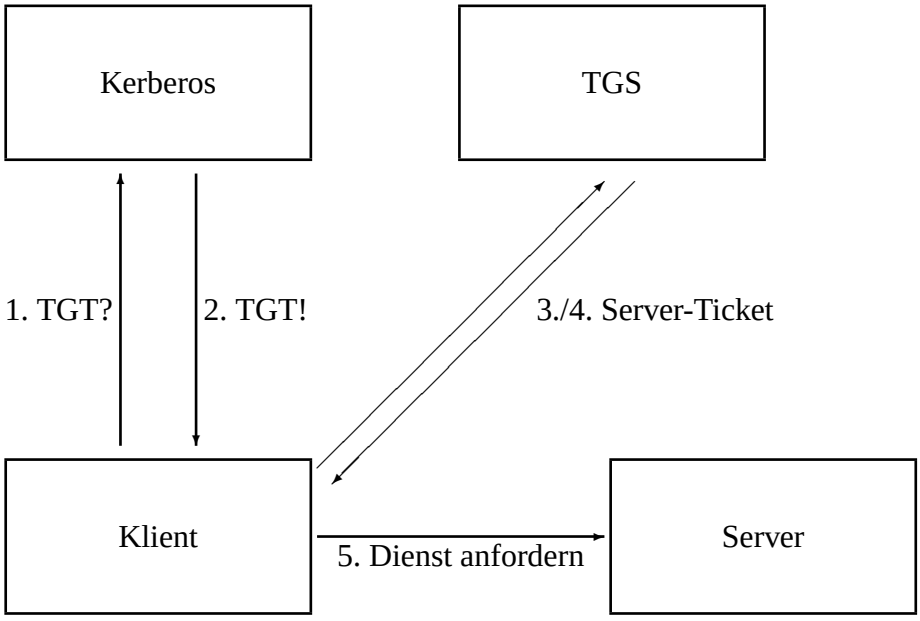
6.9.2 Kerberos

- Kerberos wurde am MIT für das Athena-Projekt entwickelt.
- Kerberos 4 war die erste öffentliche Version (1987).
- Kerberos 5 (1991) ist die Authentisierungsmethode in Windows seit NT 5 (W2K).
- Kerberos benutzt Shared Secrets.
- Dreh- und Angelpunkt ist der Kerberos Authentication Server, bei dem Benutzer und Netzwerkdienste ("Principals") ihre geheimen Schlüssel hinterlegen. Daher muss diese Rechner sicher sein (in W2K: DC). Für Benutzer ist dieser geheime Schlüssel eine verschlüsselte Version des Passwortes.

  Damit ist es möglich, Nachrichten zu verschicken, die einen Principal von der Identität eines anderen überzeugen. Außerdem werden dort Sitzungsschlüssel erzeugt, mit denen der Netzwerkverkehr zwischen zwei Parteien verschlüsselt wird. Ist die Sitzung beendet, wird der zugehörige Schlüssel zerstört, das heisst, beide Parteien löschen ihn.
- Die Verschlüsselung selbst erfolgt mit DES (oder 3DES) im CBS-Modus.
- Ein Ticket ist eine sichere Methode, einen Server von der Identität eines Klienten zu überzeugen. Es gilt immer nur für einen Server und einen Klienten und enthält:
  - Name und Netzwerkadresse des Klienten
  - Name des Servers
  - Gültigkeitsdauer (Timestamp), typisch Stunden bis maximal Tage.
  - Einen Sitzungsschlüssel

  Das Ticket ist mit dem geheimen Schlüssel des Servers verschlüsselt; damit ist der Klient sicher, dass nur der zuständige Server etwas damit anfangen kann.

- Zum Anfordern eines Dienstes wird neben dem Ticket ein Authenticator benutzt. Dieser ist (im Gegensatz zum Ticket) nur einmal gültig und ist mit dem Sitzungsschlüssel der beiden Partner verschlüsselt. Er enthält:
  - Name des Klienten
  - Zeitstempel
  - Sitzungsschlüssel (optional)
- Es gibt einen speziellen Dienst, den Ticket Granting Service (TGS). Dieser kann Tickets für alle anderen Dienste ausgeben, wenn der Klient sich mit dem initialen Ticket Granting Ticket bei ihm identifiziert.
- Authentisierungsschritte:
  1. Klient fordert Ticket-Granting-Ticket (TGT) an.
  2. Kerberos sendet TGT.
  3. Klient benutzt TGT, um ein Server-Ticket anzufordern.
  4. TGS sendet Server-Ticket.
  5. Klient benutzt Server-Ticket, um Dienst anzufordern.



Sicherheit von Kerberos

- Alle Rechner müssen die gleiche Uhrzeit haben.

- Da Tickets eine gewisse Lebensdauer haben, gibt es die Möglichkeit von Replay Attacks.
- Passworte können geraten werden!
- Die Kerberos-Software auf allen beteiligten Rechern muss sicher und vertrauenswürdig sein.

7 Public-Key-Kryptografie

Bisher: Symmetrische Kryptosysteme, das heisst:

1. Wer verschlüsseln kann, kann auch entschlüsseln.
2. Vor der Verschlüsselung muss der geheime Schlüssel ausgetauscht werden.

Geht's auch anders? Ja!
Mai 1976: Whitfield Diffie und Martin Hellman veröffentlichen ihre Arbeit: New Directions in Cryptography:

1. Versenden verschlüsselter Nachrichten ohne Schlüsselaustausch (nur theoretisch, ohne konkretes Verfahren),
2. Konzept der Digitalen Signatur
3. Schlüsselaustausch über einen unsicheren Kanal.

1977: RSA-Algorithmus bietet konkretes Verfahren.

7.1 Idee der Public-Key-Kryptografie

1. Jeder Teilnehmer hat ein Paar von Schlüsseln, einen geheimen Schlüssel d (private key) und einen öffentlichen Schlüssel e (public key).
2. Es ist praktisch unmöglich, von e (öffentlich) auf d (geheim) zu schließen.
3. Dies definiert ein *asymmetrisches Kryptosystem*.
4. e dient zum *Verschlüsseln*:

   $$e : \text{Klartext} \rightarrow \text{Geheimtext}.$$

   Wir schreiben: $g = E_B(k)$, um die Verschlüsselung des Klartextes k mit dem öffentlichen Schlüssel der Person B zu bezeichnen.

5. Nur mit d kann ich die Nachricht wieder entschlüsseln:

$$d : \text{Geheimtext} \to \text{Klartext}.$$

   Wir schreiben: $k = D_B(g)$.

6. Mathematisch gesprochen: e definiert eine Einwegfunktion, d eine „Hintertür“, um die Umkehrung doch berechnen zu können.

Wenn ich jemandem eine geheime Nachricht senden will, besorge ich mir den öffentlichen Schlüssel des Empfängers, verschlüssele damit und sende das Ergebnis. Nur der Empfänger besitzt den zugehörigen geheimen Schlüssel, um die Nachricht entschlüsseln zu können.

Ein asymmetrisches Kryptosystem heisst Public-Key-Verschlüsselungssystem (oder auch: asymmetrisches Verschlüsselungssystem), wenn für jede Nachricht m gilt:

$$c = E(m), \quad m = D(c),$$

das heisst, D macht E rückgängig.

Vorteile

- Kein Schlüsselaustausch
- Nur ein Schlüssel je Teilnehmer
- Neue Teilnehmer können jederzeit hinzugefügt werden.

Nachteile

- Wie garantiere ich, dass ich den öffentlichen Schlüssel des Empfängers bekomme (und keine Fälschung)? Lösung: PKI (Public Key Infrastructure), kommt später.
- Asymmetrische Verfahren sind langsam.

7.2 Erste Realisierung: RSA-Algorithmus

Autoren: Ronald Rivest, Adi Shamir, Leonard Adleman

Idee: Für jeden Teilnehmer nehme zwei etwa gleich große Primzahlen p, q, berechne $n = p \cdot q$ und bestmme zwei natürliche Zahlen e und d so, dass gilt:

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

Dann ist d der geheime Schlüssel, e (zusammen mit n) der öffentliche Schlüssel.
Verschlüsselung einer Zahl $m < n$:

$$g = m^e \bmod n$$

Entschlüsselung mit d:

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

Zauberei? Nein, Zahlentheorie!

7.3 Kleiner Zahlentheoretischer Exkurs

Von Leonard Euler (1707-1783) stammt ein nach ihm benannter Satz, aus dem folgende Aussage abgeleitet werden kann:
Seien p, q Primzahlen, $n = p \cdot q$. Für jede natürliche Zahl $m \leq n$ und jede natürliche Zahl s gilt:

$$m^{s(p-1)(q-1)+1} \bmod n = m.$$

Beispiel: $p = 2$, $q = 5$. Dann gilt für $s = 1$:

$$m^5 \bmod 10 = m \quad \text{für } m \leq 10.$$

Anwenden dieser Aussage auf unser Problem:

$$\begin{aligned} m' &= c^d \bmod n \\ &= (m^e \bmod n)^d \bmod n = (m^e)^d \bmod n = m^{e \cdot d} \bmod n \\ &= m^{s(p-1)(q-1)+1} \bmod n \\ &= m. \end{aligned}$$

Damit ist bewiesen, dass die Methode tatsächlich funktioniert.

Wie schwer ist es, e und d zu finden? Die Antwort liefert der Euklidische Algorithmus (um 300 v. Chr.):
Für jede natürliche Zahl n, die teilerfremd zu (p-1)(q-1) ist, kann man leicht eine natürliche Zahl d finden, so dass gilt:

$$e \cdot d = s(p-1)(q-1) + 1.$$

e wird vorgegeben, d und s ergeben sich im Laufe der Rechnung.

In unserem Fall ist die Voraussetzung leicht erfüllt, weil $n = p \cdot q$ ist, und damit teilerfremd zu $p - 1$ und $q - 1$. Anders formuliert: wenn p und q bekannt sind, sind e und d leicht zu berechnen. Umgekehrt, ist es schwer, d zu berechnen, wenn es schwer ist p und q zu bestimmen. Dieses Problem hat zwei Aspekte:

- Angreifer: Ich habe e und n, ich will d berechnen!
  In diesem Fall reduziert sich das Problem auf die Frage: wie schwer ist es, eine gegebene natürliche Zahl n in ihre Primfaktoren zu zerlegen? Damals (1977): 512 Bit sind „schwer“. Heute werden 2048 Bit als Länge für die Zahl n empfohlen.

- Benutzer: Ich brauche zwei etwa gleich grosse Primzahlen p und q, aus denen ich meine Schlüssel berechnen kann!
  Es gibt zwar sehr viele Primzahlen (zum Beispiel ist im Bereich der 512 Bit langen Zahlen jede 354te Zahl eine Primzahl), aber trotzdem muss ich prüfen, ob eine Zahl eine Primzahl ist. Zwar sind p und q nur etwa halb so lang wie n, was meinen Aufwand kleiner macht, aber im Prinzip stehe ich vor demselben Problem wie der Angreifer.

Die Lösung bieten probabilistische Primzahltests. Solche Tests können nicht beweisen, dass eine Zahl eine Primzahl ist, aber sie können diese Aussage mit einer gewissen Wahrscheinlichkeit treffen. Allerdings haben sie eine ganz entscheidende Eigenschaft: die Feststellung, dass eine Zahl *keine* Primzahl ist, ist immer richtig. Es kann also nur vorkommen (mit einer kleinen Wahrscheinlichkeit), dass der Test eine Zahl zur Primzhal erklärt, obwohl sie keine ist.

Beispiel: Miller-Rabin-Test: Input: p, ein Kandidat für eine Primzahl, und a, eine Zufallszahl, die kleienr als p ist. Liefert dieser Test die Aussage, dass eine Zahl eine Primzahl ist, so ist die Wahrscheinlichkeit höchstens $1/4$, dass sie doch zusammengesetzt ist. Wiederholt man den Test k-mal mit verschiedenen Zahlen a, so ist diese Wahrscheinlichkeit höchstens 4^{-k}, für $k = 10$ also schon etwa 10^{-6}. Dies ist allerdings eine sehr pessimistische Annahme, realistischer sind Fehlerhäufigkeiten von $1/10000$ für einen Durchgang des Tests.

Bewaffnet mit diesem Wissen können wir mit vertretbarem Aufwand Primzahlen der Länge k Bits zufällig bestimmen, und zwar so:

1. Wir wählen zufällig ein Bitmuster der Länge k, in dem das erste und letzte Bit 1 sind. (Das erste Bit, weil die Länge sonst kleiner als k wäre, und das letzte Bit, weil Primzahlen immer ungerade sind.) Damit haben wir einen Kandidaten n.

2. Wir prüfen, ob n durch kleine Primzahlen teilbar ist: 3, 5, 7, 11,… Dies ist nicht wirklich nötig, geht aber sehr schnell und schließt einen große Zahl von möglichen Kandidaten aus. Testet man zum Beispiel nur auf Division

durch 3, 5 und 7, so hat man schon über die Hälfte der Kandidaten eliminiert. Häufig testet man für alle Primzahlen unter 256 (80%); sehr effizient ist es, auf alle Primzahlen unter 2000 zu testen.

3. Wir führen den Miller-Rabin-Test für ein zufälliges $a < n$ durch. Wiederhole dies noch vier Male.

4. Fällt n durch einen der Tests, fange von vorne an.

7.4 Hybride Kryptosysteme

Der RSA-Algortihmus kann nur Bitmuster verschlüsseln, die kürzer als die Zahl $n = p \cdot q$ sind. Daher wird auch hier, wie bei DES, die Nachricht in Blöcke passender Länge zerlegt, und jeder Block einzeln verschlüsselt.

Asymmetrische Verschlüsselung ist langsam. Deswegen benutzt man meistens die asysmmetrische Verschlüsselung nur zum Austausch des Schlüssels für ein symmetrisches Kryptosystem:

1. Der Sender A wählt einen geheimen Schlüssel k und verschlüsselt seine Nachricht m damit: $g_1 ? f_k(m)$.

2. Der Sender verschlüsselt k mit dem öffentlichen Schlüssel e des Empfängers B: $g_2 = E_B(k)$ und sendet g_1 und g_2 an B.

3. B entschlüsselt zunächst g_2 mit seinem geheimen Schlüssel: $k = D_B(g_2)$, erhält damit k. Damit kann er g_1 schnell entschlüsseln.

Beispiele für hybride Systeme: SSL/TLS, PGP

Die Sicherheit eines hybriden Systems ist nur so gut wie die Sicherheit der Bestandteile: kann ich den symmetrischen Teil brechen, habe ich die Nachricht; kann ich den asymmetrischen Teil brechen, bekomme ich den geheimen Schlüssel k und kann damit die Nachricht entschlüsseln.

7.5 Besonderheiten des RSA-Verfahrens

1. Normalerweise: Verschlüsselung mit öffentlichem Schlüssel e, Entschlüsselung mit geheimem Schlüssel d. Weil aber bei RSA die Methode (modulare Exponentiation) dieselbe ist, gilt auch die Umkehrung: Was mit dem geheimen Schlüssel d verschlüsselt wird, kann mit dem öffentlichen Schlüssel e entschlüsselt werden:

$$(k^d \bmod n)^e \bmod n = k^{d \cdot e} \bmod n = k.$$

Diese Eigenschaft gilt *nicht* für jedes asymmetrische Verschlüsselungssystem.

2. Multiplikativität:
   Wenn ich die Verschlüsselung zweier Nachrichten kenne, kann ich die Verschlüsselung des Produktes berechnen:

   $$(m_1 m_2)^e \bmod n = m_1^e m_2^e \bmod n = (m_1^e \bmod n)(m_2^e \bmod n).$$

   Dies bietet eine Angriffsmöglichkeit, ich für gewisse Nachrichten (der Form $m_1 m_2$) die Verschlüsselung vorhersagen kann.
   Mögliche Lösung: Klartexte einschränken, zum Beispiel so, dass das letzte Byte immer gleich dem ersten ist. Dann ist $m_1 m_2$ so gut wie nie eine gültiger Klartext.

7.6 Schlüsselaustausch nach Diffie und Hellman

1. Die beiden Partner A und B vereinbaren

   - eine Prinzahl p
   - eine natürliche Zahl $g < p$.

   Diese brauchen *nicht* geheim zu sein.

2. A und B wählen je eine natürliche Zahl a bzw. b, die beide kleiner als $p-1$ sind und halten sie geheim.
   A berechnet: $\alpha = g^a \bmod p$ und schickt α an B,
   B berechnet: $\beta = g^b \bmod p$ und schickt β an A.

3. A berechnet: $k_a = \beta^a \bmod p$, B berechnet: $k_b = \alpha^b \bmod p$.
   Diese beiden Zahlen k_a und k_b sind gleich, denn:

   $$\begin{aligned} k_a &= \beta^a \bmod p = (g^b \bmod p)^a \bmod p = g^{ba} \bmod p, \\ k_b &= \alpha^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p. \end{aligned}$$

   $k_a = k_b = k$ ist dann ein gemeinsames Geheimnis, das A und B entweder direkt als Schlüssel nehmen, oder sie berechnen daraus den Schlüssel auf eine vorher vereinbarte Weise. (Dies kann eine öffentlich bekannte Methode sein, zum Beispiel die Vorschrift: „Nimm die ersten 56 Bit als DES-Schlüssel!“)

Diese Verfahren ist wieder praktisch unmöglich zu knacken, weil die Berechnung des diskreten logarithmus schwierig ist.

Warum macht man den DH-Schlüsselaustausch, wo es doch mit Public-Key-Kryptosystemen so einfach geht? Eng mit dem DH-Verfahren zusammen hängt das *El-Gamal-Verschlüsselungsverfahren* (Taher El Gamal 1985) (und andere, zum Beispiel Massey-Omura):

1. Alle Teilnehmer haben dieselbe Primzahl p und natürliche Zahl $g < p$.

2. B hat als geheimen Schlüssel $b < p - 1$, als öffentlichen Schlüssel $\beta = g^b \bmod p$.

3. A wählt zufällig eine Zahl $a < p - 1$ und berechnet $k = \beta^a \bmod p$. Dann wird die Nachricht m mit einem Verfahren f mit Schlüssel k verschlüsselt: $c = f_k(m)$.

4. A sendet an B: c und $\alpha = g^a \bmod p$.

5. B wendet seinen geheimen Schlüssel b auf α an: $k = \alpha^b \bmod p$. Damit entschlüsselt B den Geheimtext c: $m = f_k^{-1}(c)$.[7]

Bemerkung: El-Gamal ist ein sogenanntes *randomisiertes* Verschlüsselungsverfahren: A kann jedesmal eine andere Zahl a wählen. Dadurch entsteht aus demselben Klartext jedesmal ein anderer Geheimtext; die Geheimtexte sind zufällig und gleichverteilt, wenn a dies ist. Dies erschwert die Kryptoanalyse, zum Beispiel die Anwendung statistischer Tests.

Ein weiteres Schema, das auf dem DH-Schlüsselaustausch beruht, wurde von Jim Maseey (ETH Zürich) und Jim Omura (UC) unabhängig voneinander entwickelt. Es beruht auf Shamirs No-Key-Algorithmus. Dabei wird ohne Austausch von Schlüsseln trotzdem verschlüsselt kommuniziert! Man kann sich das Prinzip mit Hilfe von Koffern mit Vorhängeschlössern veranschaulichen:

1. A steckt die zu übermitelnde Nachricht in einen Koffer, versieht diesen mit einem Vorhängeschloss und behält den Schlüssel. Der verschlossene Koffer wird an B geschickt.

2. B verschließt den Koffer mit einem weiteren Vorhängeschloss, behält den Schlüssel und schickt den Koffer an A zurück.

3. A entfernt das erste Schloss, dessen Schlüssel sie behalten hat, lässt das zweite Schloss unangetastet und schickt den Koffer an B zurück.

4. B öffnet das verbleibende Schloss und kann die Nachricht aus dem Koffer entnehmen.

Was bedeutet das für die Kryptografie, die ja mit Zahlen und Operationen auf Zahlen arbeitet, und nicht mit Schlössern und Schlüsseln? Wir brauchen eine doppelte Verschlüsselung (für A und B), bei der es egal ist, in welcher Reihenfolge die Entschlüsselungen durchgeführt werden, denn den obigen Schritten entsprechen folgende Operationen auf der Nachricht m:

[7] Im ursprünglichen El-Gamal-Schema war $f_k(m) = k \cdot m \bmod p$, aber das ist eine unnötige Einschränkung.

1. A berechnet $f_{k_A}(m)$ und schickt dies an B.
2. B berechnet $f_{k_B}(f_{k_A}(m))$ und schickt dies an A.
3. A kehrt seine Operation um: $f_{k_A}^{-1}(f_{k_B}(f_{k_A}(m)))$ und schickt dies an B.
4. B berechnet $f_{k_B}^{-1}(f_{k_A}^{-1}(f_{k_B}(f_{k_A}(m))))$.

Damit dies wieder m ergibt, muss gelten:

$$f_{k_B}(f_{k_A}(m)) = f_{k_A}(f_{k_B}(m)).$$

Dies ist eine *kommutative* symmetrische Verschlüsselung.

Massey-Omura-Schema

1. Alle Teilnehmer vereinbaren: eine Primzahl p so, dass $p-1$ einen großen Primfaktor hat.
2. Jeder Teilnehmer T wählt einen Schlüssel e_T, der teilerfremd zu $p-1$ ist un berechnet d_T so, dass $e_T \cdot d_T = 1 \bmod p-1$ ist.[8] e_T und d_T werden geheimgehalten.
3. A berechnet: $\alpha = m^{e_A} \bmod p$ und schickt dies an B.
4. B berechnet: $\beta = \alpha^{e_B} \bmod p = m^{e_A \cdot e_B} \bmod p$ und schickt dies an A.
5. A berchnet $\gamma = \beta^{d_A} \bmod p = \alpha^{d_A \cdot e_b} \bmod p = m^{d_A e_A e_B} \bmod p$
   $= (m^{d_A e_A} \bmod p)^{e_B} \bmod p = m^{e_B} \bmod p$ und schickt dies an B.
6. B berechnet $\gamma^{d_B} \bmod p = m^{d_B e_B} \bmod p = m$.

Zum Knacken dieses Verfahrens ist wieder die Berechnung des diskreten Logarithmus notwendig.

Ein Problem dieses Schemas ist seine Anfälligkeit für eine sogenannte *Man-in-the-middle*-Attacke: Ein Angreifer C fängt die Nachrichten ab und gibt sich A gegenüber als B und B gegenüber als A aus. Das Schema hat keine Vorkehrung, einen solchen Eingriff zu entdecken. C wickelt mit beiden Seiten das Protokoll ab und ist am Ende im Besitz der Klartextnachricht m.

[8] Aus dem Satz von Euler folgt nämlich, dass $m^{e_T \cdot d_T} \bmod p = m$ ist.

8 Die digitale Signatur

Die Idee der digitalen Signatur ist abgeleitet von der handschriftlichen Unterschrift:

- Nur die richtige Person kann ihre eigene Unterschrift produzieren.
- *Jeder* kann die Unterschrit prüfen (zumindest im Prinzip).

Idee der Anwednung eines asyymetrischen Kryptosystems:
Ein asymmetrisches Kryptosystem heisst ***Signaturschema***, wenn für jede Nachricht m mit Hilfe des öffentlichen Schlüssels E überprüft werden kann, ob m und $D(m)$ zusammenpassen. Man schreibt: $D(m) = \text{sig}$.

Beispiel:

1. A unterschreibt eine Nachricht mit ihrem geheimen Schlüssel D_A und sendet m und $\text{sig} = D(m)$ an B.
2. B überprüft, ob m und $D(m)$ zusammenpassen.

Digitale Signatur mit dem RSA-Algorithmus Bei RSA gilt: $E(D(m)) = m$. Das heisst, man kann $\text{sig} = D(m)$ überprüfen, indem man $E(\text{sig})$ berechnet:

$$E(\text{sig}) = E(D(m)) = m.$$

Da dabei m wiedergewonnen wird, spricht man von einem Signaturverfahren mit Nachrichtenrückgewinnung.
Konkret:

1. Um eine Nachricht m zu signieren, berechnet A als Signatur:

$$s = m^{d_A} \bmod n.$$

2. Um s zu verifizieren berechnet B:

$$s^{e_A} = m^{e_A \cdot d_A} \bmod n = m.$$

   Dies ist nur korrekt, wenn A wirklich ihren geheimen Schlüssel d_A benutzt hat.

Ein entscheidender Unterschied zur handschriftlichen Signatur ist, dass die digitale Signatur untrennbar mit der Nachricht verbunden ist. Weder Nachricht noch Signatur kann nachträglich verändert werden, ohne dass die digitale Signatur ungültig wird.

Angriffsmöglichkeiten gegen die digitale Signatur nach dem RSA-Verfahren

1. Schlüssel unterschieben:
   Der Angreifer C überzeugt B davon, dass e_C der öffentliche Schlüssel von A ist. Da B der Sache traut, glaubt er, alle mit d_C signierten Nachrichten, seien von A unterschrieben.

2. Existenzielle Fälschung:
   C wählt irgendeine Zahl $s < n$. Dann behauptet C, s sei eine Signatur von A. Zur Verifikation berechnet B $m = s^{e_A} \pmod{n}$ und glaubt, A habe m signiert. Wenn m jetzt auch noch ein halbwegs sinnvoller Text ist, wird B dem glauben.

3. Multiplikativität:
   Sind s_1 und s_2 die Signaturen von m_1 und m_2, so ist
   $$s = s_1 s_2 \bmod n = m_1^d m_2^d \bmod n = (m_1 m_2)^d \bmod n$$
   die Signatur von $m = m_1 m_2$. Aus zwei gültigen Signaturen lässt sich so leicht eine dritte gültige Signatur berechnen, ohne den zugehörigen privaten Schlüssel zu kennen.

Auswege

1. Redundanz:
   Signiere nur Texte, deren Binärdarstellung zum Beispiel aus zwei gleichen Hälften besteht (erledigt 2,3).

2. Signatur mit Hashwert:
   Wende eine öffentliche, kollisionsresistente Einweg-Hashfunktion $h(x)$ auf den Text an, und berechne die Signatur wie folgt:
   $$s = h(m)^d \bmod n.$$
   Hier müssen m und s versendet werden, da man aus s die Nachricht m *nicht* zurückgewinnen kann.
   Der Empfänger berechnet $s^e \bmod n$ und vergleicht mit $h(m)$ (erledigt 2,3).
   In der Praxis wählt man eine gängige 160-Bit-Hashfunktion (zum Beispiel SHA-1) und wendet eine Expansionsfunktion an, zm einen langen Bitstring zu bekommen.

8.1 Andere Public-Key-Verfahren

Für die digitale Signatur lässt sich sofort jedes Public-Key-Verfahren verwenden, bei dem Verschlüsselung und Entschlüsselungsmethode vertauschbar sind, das heisst $E(D(m)) = m$. Aber es geht auch anders.

8.1.1 El-Gamal-Signatur

Hier sind Entschlüsselung und Verschlüsselung nicht vertauschbar; daher sieht das Signaturverfahren etwas anders aus als die weiter oben vorgestellte El-Gamal-Verschlüsselung:

1. A wählt einmal eine Primzahl p und eine Zahl $g < p$.
2. A wählt $a < p-1$ und berechnet $\alpha = g^a \bmod p$. Der geheime Schlüssel ist a, der öffentliche Schlüssel besteht aus (p, g, α).
3. Signiert wird eine Nachricht m mit Hilfe einer Hashfunktion $h(x)$. A wählt zufällig eine Zahl $k < p-1$, die zu $p-1$ teilerfremd ist und berechnet
$$\begin{aligned} r &= g^k \bmod p \\ s &= k^{-1}(h(m) - a \cdot r) \bmod (p-1) \end{aligned}$$
Die Signatur besteht aus (r, s).
4. Verifikation der Signatur: überprüfe zunächst, dass $1 \leq r \leq p-1$. Dann berechne
$$\alpha^r r^s = (g^a)^r (g^k)^{k^{-1}(h(m)-a\cdot r)} \bmod p = g^{h(m)} \bmod p.$$

Probleme des El-Gamal-Signatur-Verfahrens

1. Kann durch Berechenung des diskreten Logarithmus geknackt werden (sehr schwierig).
2. Unter gewissen Bedingungen ist das Knacken einfacher, nämlich wenn
   (a) $p = 3 \bmod 4$,
   (b) g ein Teiler von $p-1$ ist,
   (c) die Berchnung diskreter Logarithmen in einer Untergruppe möglich ist (das heisst, wenn g nicht zu groß ist).

   Dies vermeidet man indem man die ersten der beiden Möglichkeiten ausschließt.
3. Für jede Signatur *muss* die Zahl k neu gewählt werden, sonst lässt sich der geheime Schlüssel a berechnen!
4. Es muss eine Hashfunktion verwendet werden, sonst ist die existenzielle Fälschung möglich.
5. Dieses Verfahren ist deutlich aufwändiger als RSA: die Überprüfung erfordert drei modulare Exponentiationen.

8.1.2 Der Digital Signature Algorithm (DSA)

- 1991 vorgeschlagen von NIST; Teil des Digital Signature Standard (DSS, 1994); NSA wieder beteiligt.
- Effizientere Variante von El-Gamal, nur zwei modulare Exponentiationen, die Exponenten sind nur 160 Bit lang.

Ingredienzien:

1. Eine Primzahl q der Länge 160 Bit, das heisst,
$$2^{159} < q < 2^{160},$$
2. eine Primzahl p der Länge L, wobei L zwischen 512 und 1024 liegt und ein Vielfaches von 64 ist, also
$$2^{511+64t} < p < 2^{512+64t}, \quad t = 0, 1, 2, 3, 4, 5, 6, 7, 8,$$
(Ursprünglich sollte p genau 512 Bit lang sein, dieses wurde aber als zu kurz kritisiert.)
3. q ist ein Teiler von $p - 1$,
4. eine Zahl $x < p - 1$, sodass
$$g = x^{(p-1)/q} \bmod p > 1,$$
5. eine Zahl $a < q$.
6. Berechne $\alpha = g^a \bmod p$.
7. Der geheime Schlüssel ist die Zahl a, der öffentliche Schlüssel besteht aus des Zahlen (p, q, g, α).
8. Eine EInweg-Hashfunktion $h(m)$, DSS schreibt SHA-1 vor (160 Bit).

Erzeugung der Signatur Die Signatur der Nachricht m berechnet sich wie folgt: Absender A wählt eine Zufallszahl $k < q$ und berechnet:

$$\begin{aligned} r &= (g^k \bmod p) \bmod q, \\ s &= k^{-1}(h(m) + a \cdot r) \bmod q. \end{aligned}$$

Die Signatur besteht aus (r, s).

Überprüfung der Signatur Der Empfänger überprüft zunächst, dass $1 \leq r, s \leq q-1$ sind. Dann berechnet er

$$\begin{aligned} & ((g^{(s^{-1}h(m)) \bmod q}\alpha^{(rs^{-1}) \bmod q}) \bmod p) \bmod q \\ = \; & ((g^{(s^{-1}(h(m)+ar)) \bmod q}) \bmod p) \bmod q \\ = \; & (g^k \bmod p) \bmod q \\ = \; & r. \end{aligned}$$

Effizienz

- Beschleunigung durch Vorberechnung,
- Exponenten nur 160 Bit.

Sicherheit

- Neues k für jede Signatur wählen,
- Hashfunktion,
- Überprüfung von $1 \leq r, s \leq q-1$.

9 Das Schlüsselverteilungsproblem

Alle öffentlichen Schlüssel sind allgemein bekannt. Es muss aber sichergestellt sein, zu welcher Person ein öffentlicher Schlüssel gehört, sonst könnte ein Angreifer einen falschen öffentlichen Schlüssel unterschieben. Wie kann B garantieren, dass ein öffentlicher Schlüssel d zur Person A gehört?

1. A übergibt den eigenen öffentlichen Schlüssel d_A persönlich an B–sichere Methode aber nur in sehr kleinem Rahmen praktikabel.

2. C übergibt seinen öffentlichen Schlüssel d_C persönlich an B. A übergibt d_A persönlich an C. C erstellt eine digitale Signatur $\text{sig}_C(\text{Name}_A, d_A)$ des Paares aus Name und Schlüssel von A und schickt sie zusammen mit d_A per Email an B weiter.

   Name, öffentliche Schlüssel und Signatur zusammen bezeichnet man als *Zertifikat*.

   Wenn B dem Überbringer C vertraut, vertraut er auch Cs digitaler Signatur und glaubt daher, dass d_A der öffentliche Schlüssel von A ist.

Letzteres kann man auch über mehrere Personen fortsetzen: B traut C, C traut A and E, A traut D und F, D traut G und H, E traut F, G traut C, H traut G. Nehmen genügend Personen teil, so ergibt sich ein Netzwerk des Vertrauens ("web of trust").

Auf dieser ad-hoc Methode baut die Schlüsselverteilung bei PGP auf. Es gibt keine Regel, wie genau Schlüssel weitergegeben werden; jeder Teilnehmer entscheidet selber, ob er einer digitalen Signatur soweit traut, dass er den damit unterschriebenen Schlüssel für richtig hält oder sogar selber digital signiert. Dies erfordert keine weitere Organisation und hat sich bisher auch bewährt. Der Nachteil dabei ist, dass es nicht sicher möglich ist, an den signierten öffentlichen Schlüssel einer bestimmten Person zu kommen; es muss ja keinen Vertrauenspfad zu dieser Person geben.

Einen anderen Weg geht die hierarchische Methode: Hier gibt es eine zentrale Instanz, der alle Teilnehmer vertrauen. Diese Instanz kann entweder alle Schlüssel der Teilnehmer direkt überprüfen und signieren oder aber diese Aufgabe an Unterinstanzen delegieren. Dadurch wird garantiert, dass der öffentliche Schlüsel eines jeden Teilnehmers von einer vertrauenswürdigen Instanz digital signiert ist. Eine solche Instanz nennt man eine *Certificate Authority* (CA), die oberste Instanz *Root-CA*.

Eine Root-CA signiert ihren öffentlichen Schlüssel mit dem dazugehörigen geheimen Schlüssel. Das bedeutet: man kann zwar die Gültigkeit eines Root-CA-Zertifikats überprüfen, indem man die digitale Untersschrift prüft, aber dem Zertifikat selbst und damit der Root-CA muss man vertrauen.

Beispiel Die Firma MBI will die öffentlichen Schlüssel aller Angestellten signieren. Dazu richtig sie eine zentrale Root-CA Z am Firmensitz in den USA ein. Damit nicht alle Angestellten persönlich dorthin reisen müssen, um ihre Identität nachzuweisen, werden in allen Ländern, in denen die Firma Niederlassungen hat, länderspezifische Signaturstellen (CAs) eingerichtet. Die zentrale Signaturinstanz Z signiert die öffentlichen Schlüssel aller dieser Signaturstellen. Damit nicht alle Angestellten in Deutschland zum Sitz der Niederlassung reisen müssen, werden an allen Standorten standortspezifische Signaturinstanzen eingerichtet, deren öffentliche Schlüssel von der für Deutschland zuständigen Signaturinstanz D digital signiert werden. Jede Angestellte muss jetzt nur noch bei der Signaturinstanz S ihres Standortes persönlich vorsprechen, um ihren öffentlichen Schlüssel digital signieren zu lassen. Die so bestätigten öffentlichen Schlüssel werden im Firmennetzwerk veröffentlicht.

Wenn ein Angestellter B in den USA den öffentlichen Schlüssel einer Angestellten A in Deutschland braucht, besorgt er sich diesen im Firmennetzwerk und überprüft die Kette der Signaturen: As Schlüssel ist von S signiert. Die Gültig-

keit der Signatur von S kann er mit dem öffentlichen Schlüssel von S prüfen. Der wiederum ist von D signiert, der öffentliche Schlüssel von D von Z. Den einzigen Schlüssel, den B also wirklich auf sichere und nachweisbare Weise braucht, ist der der Root-CA Z.

Solch eine Struktur nennt man eine *Public-Key-Infrastruktur* (PKI). Allerdings gehört noch mehr dazu: es muss zum Beispiel möglich sein, einen öffentlichen Schlüssel für ungültig zu erklären. Ein Teilnehmer könnte nämlich seinen privaten Schlüssel verlieren (womit der öffentliche Schlüssel nutzlos wird, weil das Gegenstück fehlt), oder noch schlimmer, der private Schlüssel könnte gestohlen oder gar bekannt werden. In diesen Fällen muss dies der zuständigen Signaturinstanz gemeldet werden. Diese veröffentlicht eine oder mehrere Listen ungültiger Schlüssel, oder sie bietet sogar die Möglichkeit, über das Internet nachzufragen, ob ein öffentlicher Schlüssel gültig ist oder nicht.

Das Root-CA-Zertifikat müssen alle Teilnehmer einmal auf sichere Weise bekommen. Dies ist aber viel einfacher, als viele Zertifikate auf sichere Weise zu erhalten. Zum Beispiel enthalten die Web-Browser vom Hersteller schon eine die Zertifkate einer ganzen Reihe von vertrauenswürdigen Root-CAs.

Basis der Zertifikatsdienste ist der Standard X.509 (CCITT). Er regelt das Format der Zertifikate und was man alles an Attributen unterbringen kann.

Zur Demonstration zeigt Abbildung 4 den Inhalt eines Zertifkats, dessen Gültigkeitsdauer bereits abgelaufen ist (erzeugt mit openssl). Dasselbe Zertifikat, dargestellt im Internet Explorer, zeigt Abbildung 5. Ein Root-CA-Zertifikat von Verisign, Inc., dargestellt im Internet Explorer, zeigt Abbildung 6.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=DE, ST=Rhineland-Palatinate, L=Mainz, O=Proteosys,
                 OU=Certificate Issuing Authority,
                 CN=Proteosys of Mainz Issuing CA/Email=ca@proteosys.de
        Validity
            Not Before: Oct 10 15:43:10 2000 GMT
            Not After : Aug  6 15:43:10 2001 GMT
        Subject: C=DE, ST=Rhineland-Palatinate, L=Mainz, O=Proteosys,
                  CN=imap.proteosys/Email=postmaster@proteosys.de
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:af:1f:64:56:5f:58:ce:8c:df:8b:fa:98:42:73:
                    82:31:8a:71:73:6d:54:5a:da:33:9f:28:24:28:49:
                    5f:f3:33:a9:0d:f2:b7:a9:a5:51:31:bf:91:60:7f:
                    df:4c:db:7d:30:5f:13:8a:55:3d:b4:45:7a:d9:72:
                    bb:f5:f0:24:51:f7:31:0b:64:5a:c1:c1:1c:1d:e4:
                    32:25:2d:87:f9:9b:d8:c9:0b:23:45:b1:04:75:98:
                    2e:f8:a7:40:83:d5:52:7f:7d:a3:c9:96:77:27:15:
                    0d:96:15:e3:cf:3b:a9:6f:14:e9:39:e1:0e:2d:1b:
                    4e:03:0e:3b:05:fb:71:9a:23
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                email:postmaster@proteosys.de
            Netscape Comment:
                OpenSSL generated custom server certificate
            Netscape Cert Type:
                SSL Server
    Signature Algorithm: md5WithRSAEncryption
        31:93:99:e4:10:01:1f:28:9d:b8:7e:63:0e:15:44:f4:fb:9c:
        60:de:8c:07:03:bc:e0:2b:b5:13:72:9e:9a:98:e5:c5:3f:47:
        c8:8f:12:40:34:26:38:87:d4:c7:a5:b3:77:f7:85:a7:18:92:
        eb:6f:98:49:d2:b6:d1:ad:42:27:b0:45:e0:1c:9a:68:7c:07:
        eb:be:46:b4:da:ab:38:3d:b6:cc:a1:80:74:2d:aa:ca:3f:b5:
        aa:de:84:6f:ca:ad:17:94:74:77:54:0d:7b:dd:fe:d4:d7:b2:
        a8:f1:28:c7:c9:a6:1c:a0:7b:78:82:0d:05:0a:41:24:de:82:
        28:f6:8f:0a:20:d0:07:f5:c7:ae:ff:b2:1e:c5:43:1d:c0:aa:
        bb:27:3f:63:2e:71:79:bc:a8:07:64:a8:22:2e:79:d9:c9:6b:
        57:4b:15:6b:03:3c:04:2b:21:5b:4f:af:37:1c:47:3f:73:a5:
        9a:cf:30:00:b5:e5:c3:3c:76:b8:81:34:0a:30:01:c9:bb:d1:
        f4:6a:b2:18:35:7b:90:31:06:9f:9a:e8:8f:de:5b:ac:ad:c8:
        a6:b5:f7:19:99:8f:21:17:14:5e:6b:c3:bd:37:16:04:0b:3c:
        76:9b:9f:61:e8:4f:76:b5:56:90:f0:87:df:2e:21:eb:96:74:
        84:54:f4:a6
```

Abbildung 4: Abgelaufenes Zertifikat, dargestellt mit openssl

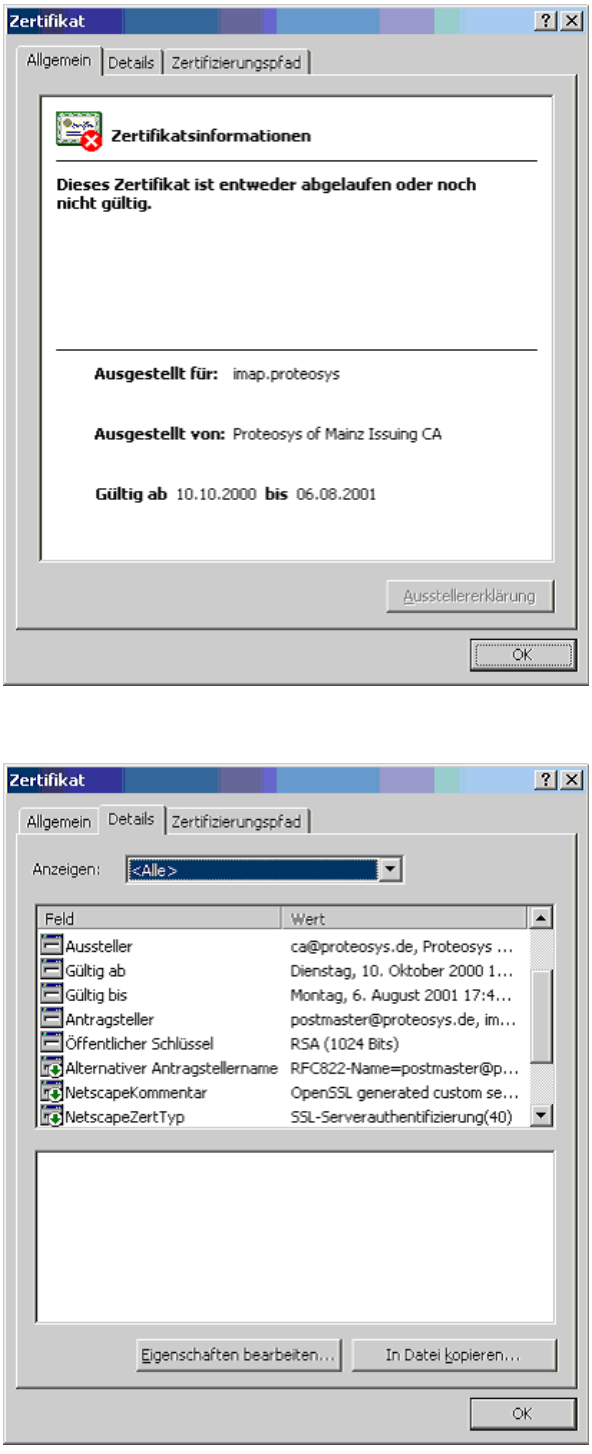


Abbildung 5: Abgelaufenes Zertifikat, dargestellt im Internet Explorer

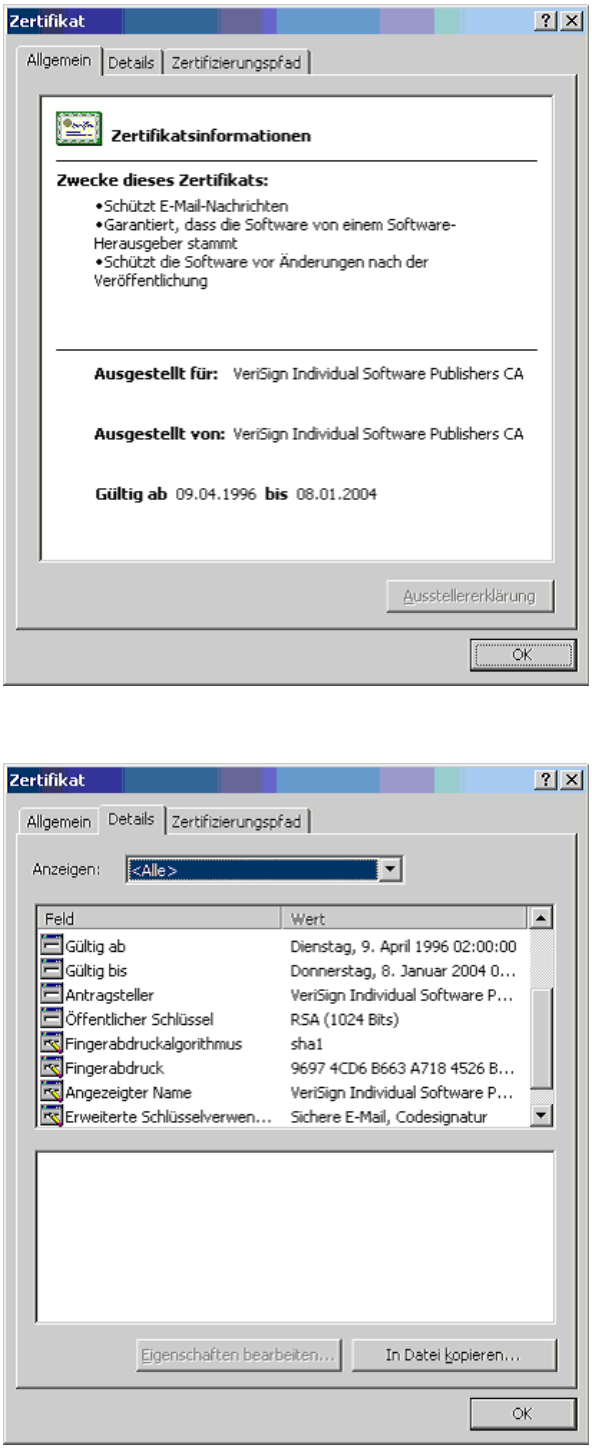


Abbildung 6: Root-CA-Zertifikat von Verisign

10 Literatur

10.1 Bücher

Albrecht Beutelspacher, Kryptologie, Eine Einführung in die Wissenschaft vom Verschlüsseln, Verbergen und Verheimlichen, Vieweg Braunschweig 2002, ISBN 3-528-58990-6.
Eine gute Einführung in das Thema; auf dieses Buch baue ich die Vorlesung auf.

Albrecht Beutelspacher, Geheimsprachen, Geschichte und Techniken, C.H Beck München, 3. Auflage 2002, ISBN 3-406-49046-8.
Eine einfache Einführung ohne viel Mathematik, die man auch dem Laien in die Hand drücken kann.

Johannes Buchmann, Einführung in die Kryptografie, Zweiter, erweiterte Auflage, Springer 2001, ISBN 3-540-41283-2. Eine sehr stark mathematisch geprägte Einführung in das Thema.

Donald E. Knuth, The Art of Computer Programming, Volume 2, Seminumerical Algorithms, 2nd Ed., Adison-Wesley 1980, ISBN 0-201-03822-6. Enthält ein langes Kapitel über die Erzeugung von (Pseudo-)Zufallszahlen.

Bruce Schneier, Applied Cryptography, Protocols, Algorithm and Source Code in C, 2nd ed., John Wiley & Sons 1996, ISBN 0-471-11709-9. (Deutsche Ausgabe bei Addison-Wesley)
Die Bibel der Kryptografie, leider schon etwas älter, sodass die letzten Entwicklungen nicht drin stehen.

Bruce Schneier, Secrets & Lies, Digital Security in a Networked World, John Wiley & Sons 2000, ISBN 0-471-25311-1.
Ein Buch über Angriffe und Sicherheit im digitalen Zeitalter

10.2 Einige Links

US National Security Agency (NSA) National Cryptologic Museum:
`http://www.nsa.gov/museum/index.html`

Eine gute Startseite mit vielen Kryptologie-Links:
`http://www.infoserversecurity.org/`

Ein Überblicksartikel zu DES:
http://www.ams.org/notices/200003/fea-landau.pdf

Definition von AES:
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Zum Knacken von AES:
http://eprint.iacr.org/2002/044/

Zur Sicherheit des Point-to-Point Tunneling Protocols und von MS-CHAP:
http://www.counterpane.com/pptp.pdf

Kerberos und Windows 2000:
http://www.securityfocus.com/infocus/1493

Public-Key Cryptography Standards:
http://www.rsasecurity.com/rsalabs/pkcs/

S/MIME:
http://www.rsasecurity.com/standards/smime/faq.html

S/MIME und OpenPGP:
http://www.imc.org/smime-pgpmime.html

Die rechtliche Stellung der digitalen Signatur in Deutschland:
http://www.seccommerce.de/de/fachwissen/politik_und_recht/deutschland/deutschland.html

10.3 Einige Romane

Dorothy L. Sayers, The Nine Tailors, Dt. Der Glocken Schlag.
Mal eine ganz andere Chiffrierung.

Dorothy L. Sayers, Have His Carcase, Dt. Zur fraglichen Stunde.
Ein ganzes Kapitel über das Knacken einer Chiffrierung.

Neal Stephenson, Cryptonomicon, Harper Collins 2000, ISBN 0-380-78862-4.
Ein Roman über die Enigma, die Kryptologie während des Zweiten Weltkrieges, einen Goldschatz und vieles mehr.