Kryptologie

Einführung und Überblick

— Folien —

Andreas de Vries

heise Newsticker vom 16.2.2005

Was bedeutet diese Meldung für die Sicherheit im Internet?

heise Newsticker vom 16.2.2005

Was bedeutet diese Meldung für die Sicherheit im Internet?

Sind jetzt alle Verschlüsselungen gebrochen?

- Können wir nun überhaupt noch Kreditkartennummern übers Web weitergeben oder Online-Banking durchführen?
- Was bedeuten Begriffe wie „Kryptoverfahren“, „Hash-Funktion“, „geknackt“?

Inhaltsverzeichnis

Kapitel 1

Grundbegriffe der Kryptologie

Informationssicherheit in Kommunikationssystemen – Ziele

- *Vertraulichkeit (privacy):* Alle Informationen sind nur den Befugten zugänglich. Eine Nachricht kann nur von dem Empfänger gelesen werden.
- *Integrität (integrity):* Informationen können nicht verfälscht werden. Eine Nachricht kann nicht verfälscht werden.
- *Authentizität (authenticity):* Der Empfänger kann sicher sein, das die Nachricht von dem ausgewiesenen Sender stammt.
- *Verbindlichkeit (non-repudiation):* Weder Sender noch Empfänger einer Nachricht können leugnen, die Nachricht verschickt bzw. bekommen zu haben.
- *Verfügbarkeit (availability):* Jede gewünschte Information oder jeder gewünschte Dienst des Kommunikationssystems ist stets verfügbar.

Angriffe auf Kommunikationssysteme

Jede dieser Eigenschaften eines Kommunikationssystems kann Angriffen ausgesetzt sein.

- *passive Angriffe:* Nachrichten oder Informationen werden abgehört, aber nicht verändert
- *aktive Angriffe:* Nachrichten oder gar das Kommunikationssystem werden verändert
  - z.B. Angriffe auf die Verfügbarkeit des Systems (*DoS*-Angriffe),

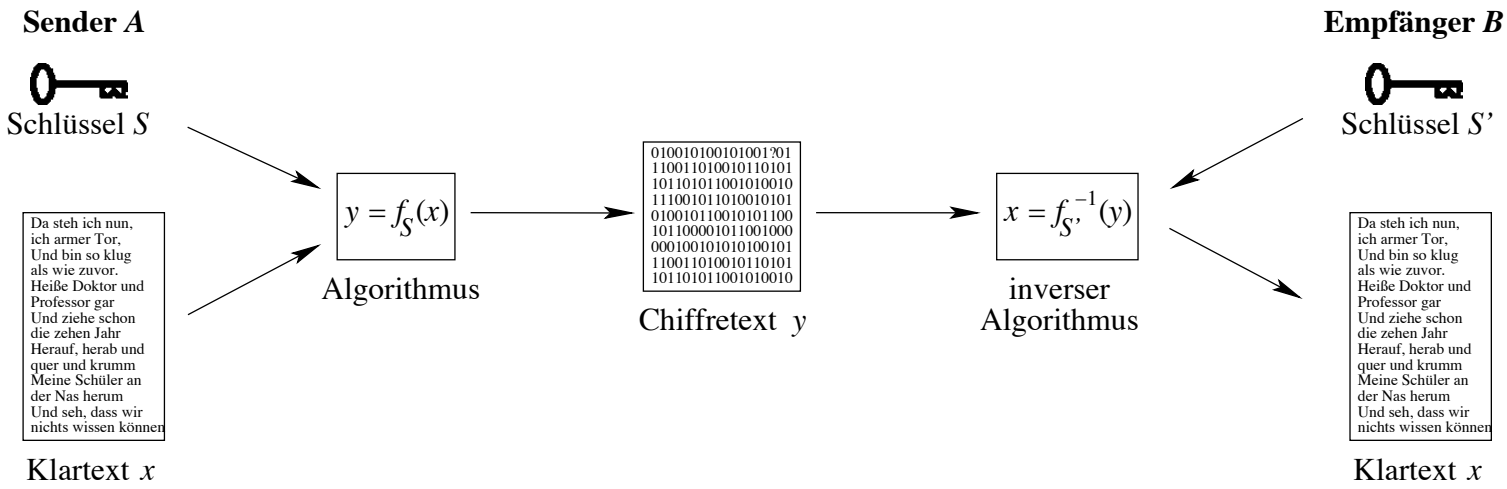
Kryptologie = Kryptographie + Kryptanalyse

- ***Kryptographie:*** Verschlüsselung von Informationen

  Ziel: Angriffe auf Vertraulichkeit, Integrität, Authentizität und Verbindlichkeit von Nachrichten zu verhindern
- ***Krypt(o)analyse***: Analyse von Verschlüsselungen.

  Ziel: das Brechen kryptographischer Verfahren ist

1.1 Kryptosysteme



Der Verschlüsselungsalgorithmus f ist eine umkehrbare Abbildung

$$f_S: \text{ Klartext } \rightarrow \text{ Chiffretext }, \qquad f_S \text{ und } f_{S'}^{-1} \text{ eindeutig.} \tag{1.1}$$

f_S: Chiffrierschlüssel S als Parameter,
$f_{S'}^{-1}$: Umkehrabbildung von f_S, Dechiffrierschlüssel S' als Parameter
Zu einem S gehört stets ein S' und umgekehrt.

***(Kerckhoffs Prinzip)* Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Algorithmus abhängen, sondern nur von der Geheimhaltung seiner Schlüssel.**

„Leute, die behaupten, dass sie einen unknackbaren Code besitzen, bloß weil sie ihn selbst nicht knacken können, sind entweder Genies oder Dummköpfe. Letztere sind leider weitaus häufiger vertreten.“ (Bruce Schneier)

Die 3 Hauptprobleme der Kryptographie

1. **Schlüsselaustausch:** Ein sicherer Kanal zum Schlüsselaustausch muss gewährleistet sein.
2. **Authentifizierung:** Der Sender muss sicher feststellen können, dass er tatsächlich mit dem Empfänger und nicht mit einer dritten Person kommuniziert.
3. **Abhörerkennung *(Intrusion Detection)*:** Sender oder Empfänger muss feststellen können, ob die Kommunikation abgehört wird oder Nachrichten sogar verändert wurden.

Vollkommen sichere Kryptosysteme: OTP

- Ein Kryptosystem heißt ***vollkommen sicher***, wenn x und y *„stochastisch unabhängig“* sind:

$$P(x|y) = P(x) \quad \Longleftrightarrow \quad P(x \wedge y) = P(x)P(y) \tag{1.2}$$

- Die einzige vollkommen sichere Chiffre ist das **One-Time-Pad**
- Nachricht x (in binärer Form) und genauso lange Zufallsfolge S wird XOR-verknüpft:

$$y = x \oplus S.$$

- Der Empfänger muss denselben Schlüssel S kennen und kann aus dem Chiffretext y den Klartext x ermitteln, denn $x = y \oplus S$.

  Beipiel:

Alice verschlüsselt:	Bob entschlüsselt:
$x = 0110\ 0101\ 1101$	$y = 1100\ 1011\ 1001$
$S = 1010\ 1110\ 0100$	$S = 1010\ 1110\ 0100$
$y = x \oplus S = 1100\ 1011\ 1001$	$x = y \oplus S = 0110\ 0101\ 1101$

- Bedingungen für die vollkommene Sicherheit: der Schlüssel S ...
  - ist echt zufällig erzeugt
  - ist nur Sender und Empfänger bekannt
  - wird nur einmal verwendet

Probleme des One-Time-Pads & Dilemma der klassischen Kryptographie

- Schlüssel muss echt zufällig sein,
- Schlüsselaustausch über einen sicheren Kanal

Dilemma der klassischen Kryptographie *(Catch-22)* Es gibt vollkommen sichere Arten der Kommunikation, vorausgesetzt man kann vollkommen sicher kommunizieren . . .

In der Praxis sind daher *praktisch sichere* Kryptosysteme wichtiger, . . .

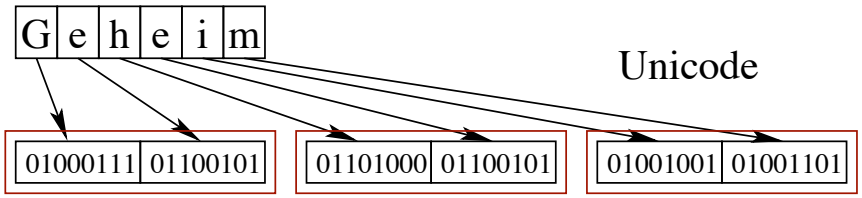
- die mit dem jeweils aktuellen Stand der Technik in n Jahren gebrochen werden,
- wobei die Größe von n von den Sicherheitsbedürfnissen der Kommunikationsteilnehmer abhängt.

  Beispiele:

  - Cäsar-Verschlüsselung, die das römische Heer während der Gallischen Kriege erfolgreich anwandte (*heute* trivial zu brechen)
  - DES, für ein Vierteljahrhundert weltweiter Verschlüsselungsstandard, aber Ende der 1990er Jahre von einem Spezialrechner in 4,5 Tagen gebrochen

1.2 Block- und Stromchiffre

- In allen modernen kryptographischen Verfahren wird die zu verschlüsselnde Nachricht in Zahlen (Bitfolgen) codiert, in der Regel mit einem der gängigen Codes[1] ASCII oder Unicode.
- Diese Zahlen werden durch den Verschlüsselungsalgorithmus mathematisch verändert und ergeben damit eine andere Zahlenfolge, die den Chiffretext darstellt.
- Eine ***Blockchiffre (block cipher)*** unterteilt den Klartext in Blöcke fester Länge, die unabhängig voneinander mit demselben Schlüssel verschlüsselt werden.



- Eine ***Stromchiffre (stream cipher)*** dagegen verschlüsselt die gesamte Nachricht Zeichen für Zeichen.

[1] Ein Code hat nichts mit Verschlüsselung zu tun, sondern ist nur eine Vorschrift (Tabelle), die jedem Buchstaben eine eindeutige Zahl zuordnet. Siehe dazu auch mein Skript „Grundlagen der Informatik“.

1.3 Kryptanalyse: Wann ist eine Chiffre geknackt?

- ***Brechen*** oder Knacken eines Kryptosystems: bei Unkenntnis des Schlüssels aus einem Chiffretext den Klartext oder gar den Schlüssel zu entdecken.
- Eine Methode, alle kombinatorisch möglichen Verschlüsselungen durchzuprobieren, heißt ***Brute-Force-Attacke***.
- Fast alle Kryptosysteme lassen sich mit Brute-Force brechen (bei genügend sicheren Schlüsseln abe Aufwand von hunderttausend oder Millionen Jahren)
- Wichtige kryptanalytische Angriffe:
  - *Ciphertext-Only-Angriff:* Der Angreifer verfügt nur über eine bestimmte Menge an Chiffretexten.
  - *Known-Plaintext-Angriff:* Der Angreifer kennt neben dem Chiffretext den zugehörigen Klartext.
  - *Chosen-Plaintext-Angriff:* Der Angreifer kann einen beliebigen Klartext vorgeben und hat die Möglichkeit, an seinen Chiffretext zu gelangen.
  - *Chosen-Ciphertext-Angriff:* Der Angreifer kann einen beliebigen Chiffretext vorgeben und an den zugehörigen Klartext gelangen.
  - *Soziale Angriffe:* Der Angreifer bestiehlt, besticht, bedroht oder foltert eine Person, die den Schlüssel kennt.

  Gelingt ein Angriff mit deutlich weniger Versuchen als beim Brute-Force-Ansatz, gilt das Verfahren als *gebrochen* oder *geknackt*.

Differentielle Kryptanalyse

- eine statistische Methode zur Erlangung des Geheimschlüssels.
- üblicherweise ein Chosen-Plaintext-Angriff, (es gibt aber auch Erweiterungen als Known-Plaintext- und sogar Ciphertext-Only-Angriffe.)
- Paare von Klartexten, die sich jeweils durch eine irgend definierte konstante *Differenz* unterscheiden, (in der Regel durch die XOR-Operation definiert) und die Differenz der entsprechenden Chiffretexte berechnet.
- Ist der Angriff erfolgreich, so ergeben sich statistische Muster in der Verteilung der Chiffretext-Differenzen, die den geheimen Schlüssel erkennen lassen.
- Biham und Shamir führten 1990 die Methode der bis dahin nicht öffentlich bekannte differentiellen Kryptanalyse als einen Chosen-Plaintext-Angriff gegen DES ein.

Kapitel 2

Symmetrische Verschlüsselungen

- Bei ***symmetrischen Verschlüsselungsalgorithmen*** wird ein einziger geheimer Schlüssel zur Ver- und Entschlüsselung verwendet
- In dem Kryptosystem gilt also $S = S'$.
- Da dieser Schlüssel unter allen Umständen von Sender und Empfänger geheim gehalten werden muss, heißen solche Verfahren auch *Secret-Key-Verfahren*.

2.1 Cäsar-Verschiebung

- Verschlüsselungsalgorithmus: *Ersetze jeden Buchstabe des Klartextes zyklisch durch denjenigen Buchstaben, der s Stellen im Alphabet weiter steht.*
- Mathematisch formuliert: $f_s : \{0,1,\ldots,25\} \to \{0,1,\ldots,25\}$,

$$f_s(n) = n + s \bmod 26. \tag{2.1}$$

  wo n die Stelle im Alphabet von x:

x	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
$n = \langle x \rangle$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

- f bezeichnet die Stelle des im Alphabet chiffrierten Buchstabens. Der Schlüssel ist dabei also die Verschiebungskonstante s.
- Die Umkehrfunktion von f_s lautet einfach $f_s^{-1}(y) = y - s \bmod 26$. (Oder einfach: $f_s^{-1} = f_{-s}$, was aber nur für die Cäsar-Verschiebung gilt.)

Beispiel 2.1 *Cäsar-Verschiebung mit $s = 3$.* Mit dem Schlüssel $s = 3$ kann man sich leicht die folgende Chiffriertabelle erstellen (kleine Buchstaben verwendet man dabei üblicherweise für den Klartext, große für den Chiffretext):

x	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
y	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

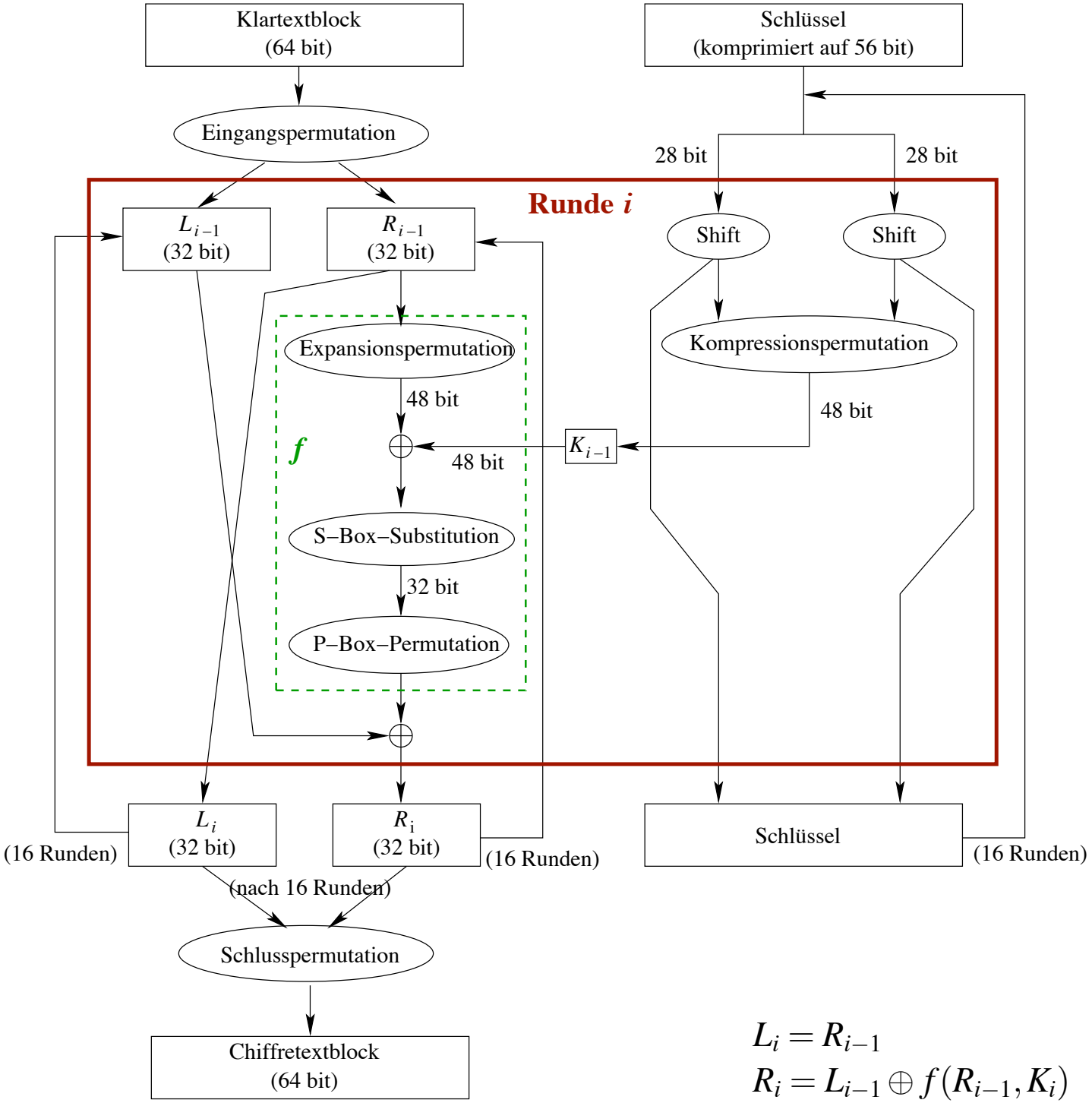
Der Chiffretext „YHQL, YLGL, YLFL“ lautet damit im Klartext „veni, vidi, vici“ („ich kam, sah, siegte“). □

Cäsar-Verschiebung — Untersuchung

- Wieviel mögliche verschiedene Schlüssel gibt es bei der Cäsar-Verschiebung? Natürlich genau 26 (wobei einer, nämlich $s = 0$, nicht wirklich zu gebrauchen ist ...).
- Um die Cäsar-Verschlüsselung zu knacken und ein Chiffrat zu entschlüsseln, könnte man also versuchen, alle möglichen Schlüssel durchzuprobieren. Eine der 26 berechneten Texte ist dann der Klartext. Mit Papier und Bleistift ist ein solcher Brute-Force-Angriff auf die Cäsar-Verschiebung zwar ein gewisser Aufwand, ein Computer dagegen würde die 26 Texte in Bruchteilen einer Sekunde berechnen.
- Merke: Im Computerzeitalter ist ein Schlüsselraum der Größe 26 (also etwa 5 Bit) unbrauchbar.
- Man kann die Cäsar-Verschiebung drastisch verbessern, wenn man eine allgemeine Permutation (d.h. eine 1:1-Zuordnung) der 26 Buchstaben verwendet. Dann hätte man plötzlich immerhin $26! \approx 4 \cdot 10^{26}$ verschiedene Schlüssel, ($26!$ entspricht etwa 89 Bit.)
- Solche Verfahren nennt man ***„Monoalphabetische Verschlüsselungen“***.

2.2 DES

- Symmetrischer Verschlüsselungsstandard der NIST von Juni 1977 bis Februar 2001.
- eine symmetrische Blockchiffre, die den Klartext in Blöcken von 64 bit verschlüsselt.
- Der Algorithmus erhält einen Block von 64 bit und liefert einen ebenso langen Chiffretext.
- Die Schlüssellänge beträgt 56 bit, als Schlüssel kommt jede 56 bit lange Zahl in Frage, auch wenn es wenige, leicht vermeidbare „schwache“ Zahlen gibt.
- Die gesamte Sicherheit des Verfahrens beruht auf dem Schlüssel.



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

Der Kern von DES: die S-Box-Substitution

- Die Sicherheit von DES basiert im Wesentlichen auf der Funktion f, speziell auf der S-Box-Substitution.
- Sie besteht aus 8 S-Boxen, die jeweils 6 bit der eingehenden 48 bit transformieren und eine Ausgabe von je 4 bit liefern [3, §4.1].
- Die S-Box-Transformation ist die einzige nichtlineare Transformation des DES.
- Informationstheoretisch betrachtet erhält DES 120 bit Information (64 bit des Klartextblock und 56 bit des Schlüssels), gibt aber nur 64 bit aus. Netto werden also 56 bit an Information verborgen (= „Entropie erzeugt“).

Sicherheit von DES

- DES wurde mehrfach Ende der 1990er Jahre durch Brute-Force gebrochen.
- DES war also am Ende des letzten Jahrtausends als Verschlüsselungsstandard nicht mehr haltbar. Ist der Algorithmus also schlecht?
- Problem: die Schlüssellänge 56 bit — der DES zugrunde liegende Algorithmus LUCIFER von Horst Feistel verwendete einen 128-bit-Schlüssel, der auch heute noch unbrechbar wäre.

2.3 Triple-DES

- Eine der recht oft eingesetzten Verbesserungen von DES ist Triple-DES.
- Es wendet DES dreimal hintereinander mit zwei Schlüsseln K_1, K_2 an, nach dem Schema

  $$y = E_{K_1}(D_{K_2}(E_{K_1}(x))),$$

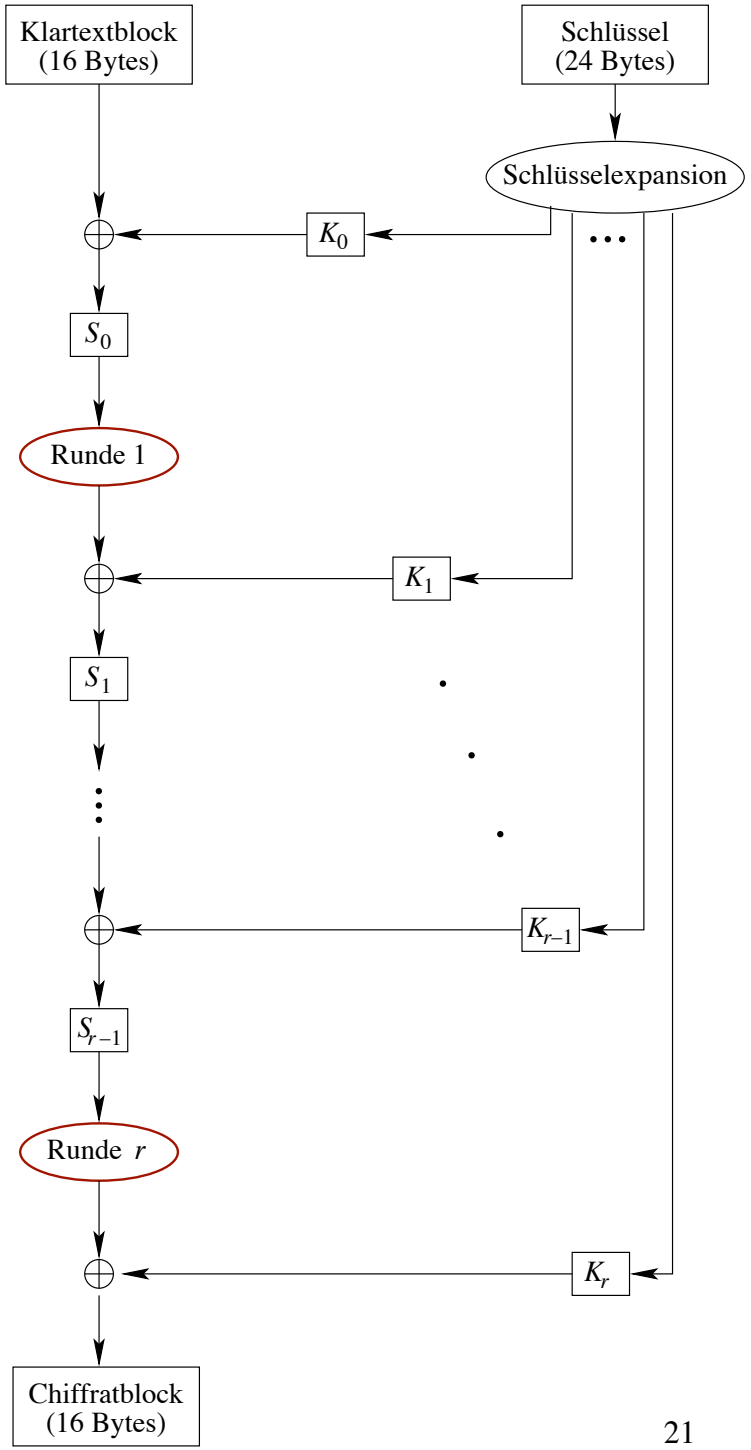
  wo E_{K_1} Verschlüsselung *(encoding)* mit dem Schlüssel K_1 bedeutet und D_{K_2} Entschlüsselung *(decoding)* mit dem Schlüssel K_2.
- Mit zwei 56-bit-Schlüsseln ist damit die effektive Schlüssellänge 112 bit.
- Warum die Entschlüsselung zwischendurch, und nicht einfach $E_{K_2}E_{K_1}$ hintereinander?
- Es gibt einen Known-Plaintext-Angriff, den ***Meet-in-the-Middle*-Angriff**, der auf alle Blockchiffren anwendbar ist, der zum Auffinden des Schlüssls nur doppelt so aufwendig wie auf DES ist — die Schlüssellänge ist also effektiv nicht 112 bit, sondern nur 57 bit!

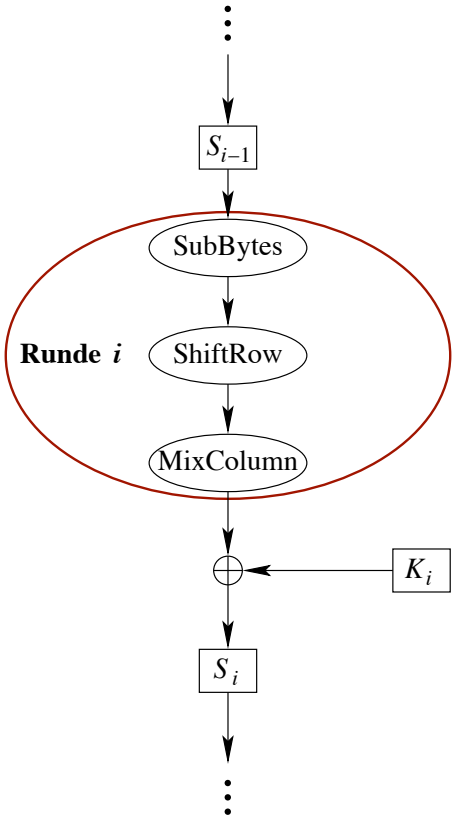
2.4 AES *(Advanced Encryption Standard)*

- seit Februar 2001 der offizielle symmetrische Verschlüsselungsalgorithmus der NIST (FIPS 197).
- Blockchiffre, deren Block- und Schlüssellänge unabhängig voneinander auf einen der Werte 16, 24 oder 32 Byte (128/192/256 Bit) gesetzt werden können.
- Ist n_B die Anzahl der 4-Byte-Worte eines Textblocks, so ist die Blocklänge $4n_B$; ist entsprechend n_K die Anzahl der 4-Byte-Worte des Schlüssels K, so ist die Schlüssellänge $4 \cdot n_K$.
- Der zu verschlüsselnde Klartext durchläuft r Runden,

r	$n_B = 4$	$n_B = 6$	$n_B = 8$
$n_K = 4$	10	12	14
$n_K = 6$	12	12	14
$n_K = 8$	14	14	14

- $r+1$ Rundenschlüssel $K_0, K_1, \ldots, K_r$, der Länge $4 \cdot n_B$ Byte werden mit der *Schlüsselexpansion* aus dem $4 \cdot n_K$ Bytes langem Schlüssel erzeugt.
- Das Zwischenergebnis der Verschlüsselung nach der i-ten Runde heißt **Zustand** *(state)* und wird mit S_i bezeichnet.


Klartextblock
(16 Bytes)
Schlüssel
(24 Bytes)
Schlüsselexpansion
K_0
S_0
Runde 1
K_1
S_1
K_{r-1}
S_{r-1}
Runde r
K_r
Chiffratblock
(16 Bytes)


S_{i-1}
SubBytes
Runde i
ShiftRow
MixColumn
K_i
S_i

- Jeder Zustand S_i, jeder expandierte Rundenschlüssel K_i und der Ursprungsschlüssel K hat jeweils die Form einer Matrix:

$$S_i = \begin{pmatrix} s_{0,0} \cdots s_{0,n_B-1} \\ \vdots \quad \ddots \quad \vdots \\ s_{3,0} \cdots s_{3,n_B-1} \end{pmatrix}, \quad K_i = \begin{pmatrix} k_{0,0} \cdots k_{0,n_B-1} \\ \vdots \quad \ddots \quad \vdots \\ k_{3,0} \cdots k_{3,n_B-1} \end{pmatrix}, \quad K = \begin{pmatrix} K_{0,0} \cdots K_{0,n_K-1} \\ \vdots \quad \ddots \quad \vdots \\ K_{3,0} \cdots K_{3,n_K-1} \end{pmatrix}.$$

- Jede Spalte der Zustands- sowie der Schlüsselmatrizen entspricht einem 4-Byte-Wort.

- In jeder Runde i wird jedes Byte des Zustands S_i von den Teilalgorithmen `subBytes`, `shiftRow` und `mixColumn` verändert.

- Bezeichnet `s[i]` den Zustand S_i und `k[i]` den Rundenschlüssel K_i so lautet der Pseudocode von AES:

```
aes( klartext, schlüssel ) {
   k = schlüsselExpansion( schlüssel);
   s[0] = xor( klartext, k[0] );
   for ( i = 0; i < r; i++ ) {
      s[i] = subBytes( s[i] );
      s[i] = shiftRow( s[i] );
      if ( i < r - 1 )
         s[i] = mixColumn( s[i] );
      s[i+1] = xor( s[i], k[i] );
   }
   return s[r];   // das ist der Chiffretextblock
}
```

- Die Eingabe des Algorithmus `aes` ist also der Klartextblock und der Schlüssel, Rückgabe ist der Chiffretextblock.
- Alle Transformationen der Zustände sind unabhängig von dem Schlüssel, bis auf die XOR-Verknüpfungen.
- Die Transformationen ShiftRow und MixColumn sind Vertauschungen von Zeilen und Spalten der Zustandsmatrix S_i („lineare Transformationen“)
- subBytes ist eine kompliziertere („nichtlineare“) Transformation. Sie und ihre Umkehrabbildung sind gemäß der Wertetabelle gegeben:

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

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

Bewertung von AES

- AES ist schnell. Auf einem 5-GHz-Rechner sind mit gängigen C-Compilern bis zu 1 Gbit/s verarbeitbar; damit ist er grob zehnmal schneller als sein Vorgänger DES
- Ähnlich wie DES ist AES effizient in Hardware implementierbar, da nur einfache Bit-Operationen wie XOR und zyklische Verschiebungen verwendet werden.
- Da AES zudem auf Bytes beruht, lässt er sich sehr effizient auf den 8-Bit-Prozessoren von Chipkarten implementieren.
- Auf den üblichen Prozessoren benötigt der Programmcode nur etwa 1 KB Speicherplatz, das für die Daten benötigte RAM beschränkt sich bei Blocklänge 16 Byte auf 36 bis 52 Byte, je nach verwendeter Schlüssellänge.
- Die Sicherheit von AES wird als hoch angesehen. Es ist kein erfolgreicher Angriff bekannt.
- Weitere Informationen zu AES: *http://csrc.nist.gov/CryptoToolkit/aes/rijndael/* Ein interaktives Applet, das AES visualisiert, finden Sie unter *http://haegar.fh-swf.de/Rijndael/*

Checkliste der Kryptoprobleme

Checkliste für symmetrische Chiffren
Schlüsselaustausch gelöst?
Authentifizierung?
Abhörerkennung?

Checkliste der Kryptoprobleme

Checkliste für symmetrische Chiffren	
Schlüsselaustausch gelöst?	nein
Authentifizierung?	
Abhörerkennung?	

Checkliste der Kryptoprobleme

Checkliste für symmetrische Chiffren	
Schlüsselaustausch gelöst?	nein
Authentifizierung?	nein
Abhörerkennung?	

Checkliste der Kryptoprobleme

Checkliste für symmetrische Chiffren	
Schlüsselaustausch gelöst?	nein
Authentifizierung?	nein
Abhörerkennung?	nein

Aufgaben

Übung 2.4.1 *(S-Box von AES)* Die SubBytes-Transformation des AES, auch "S-Box" genannt, ist eine nichtlineare Transformation, die man durch die obige Wertetabelle darstellen kann. Entsprechend ergibt sich die inverse S-Box aus der der obigen rechten Wertetabelle
(a) Müssen die Transformationen der S-Box bei festem y eindeutig sein? (Begründung)
(b) Wie wird das Wort `0x00460057` durch die S-Box transformiert, und wie lautet es zurück transformiert?

Kapitel 3

Unsymmetrische Verschlüsselungen

- Ein unsymmetrisches Verschlüssungsverfahren *(public key system)* ist ein Kryptosystem, in dem jeder Teilnehmer einen eigenen öffentlichen *(public key)* sowie einen privaten Schlüssel *(private key)* hat, der stets geheim bleibt.
- D.h.: $S \neq S'$, Ver- und Entschlüsselung einer Nachricht funktionieren mit jeweils einem eigenen Schlüssel.
- Alices hat öffentlichen Schlüssel mit P_A und geheimen Schlüssel S_A, Bob hat P_B und S_B.
- Jeder Teilnehmer des Kryptosystems erzeugt sein eigenes Schlüsselpaar P_X und S_X.
- Der private Schlüssel S_X wird unter allen Umständen geheim gehalten, der öffentliche Schlüssel P_X jedoch kann jedem bekannt gemacht werden. Er kann sogar in ein öffentliches Verzeichnis eingetragen werden, ähnlich einem Telefonbuch, in dem jeder den Schlüssel jedes anderen nachschlagen kann.

3.1 Falltürfunktionen

Der Kern jedes unsymmetrischen Kryptosystems ist eine „Falltürfunktion“, die gewährleistet, dass der öffentlich bekannte Schlüssel nicht die effiziente Berechnung des ihm eindeutig zugeordneten geheimen Schlüssels ermöglicht.

Definition 3.1 Sei $f : X \to Y$ eine umkehrbare Funktion zwischen zwei gleichmächtigen (d.h. „gleich großen“) Mengen X und Y. Dann ist f eine ***Falltürfunktion (trap-door function)***, wenn die Bedingungen erfüllt sind:

1. $y = f(x)$ ist leicht berechenbar, also in polynomialer Rechenzeit abhängig von der Länge von x.
2. Die Umkehrung $x = f^{-1}(y)$ ist ohne zusätzliche Information (den Geheimschlüssel) schwer berechenbar, also nicht in polynomialer Rechenzeit abhängig von der Länge von y.

□

- Für eine Einwegfunktion gibt es keinen Schlüssel, der die Umkehrung leicht berechenbar macht.
- „schwer berechenbar“ ist ein empirischer Begriff und hängt ab von dem aktuellen Stand der Rechnertechnologie (Rechenschieber, Quantenrechner) und neuen Algorithmen. *(Was 2005 eine Falltürfunktion war, wird es vielleicht 2015 oder 2105 nicht mehr sein!)*

Beispiele für Falltürfunktionen

(a) ***(Differentiation und Integration)*** Sei F die Abbildung der differenzierbaren Funktionen $f : \mathbb{R} \to \mathbb{R}$ in die Menge der integrierbaren Funktionen mit einer Stammfunktion, also

$$F(f) = f'.$$

Dann ist die Umkehrung $F^{-1}(f) = \int f(x)\,\mathrm{d}x$. F ist eine Falltürfunktion. („Differenzieren ist Technik, Integrieren ist Kunst.“)

(b) ***(Modulare Exponenzierung und diskreter Logarithmus)*** Für $m, n \in \mathbb{N}$ (m und n teilerfremd) sei $f : \mathbb{Z}_n \to \mathbb{Z}_n$ die Funktion auf der Menge $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$, die gegeben ist durch

$$f(x) = m^x \bmod n. \tag{3.1}$$

Das ist leicht zu berechnen. Was ist aber die Umkehrfunktion? Betrachten wir z.B. $n = 9$, $m = 2$. Dann gilt

x	0	1	2	3	4	5	6	7	8	9
$f(x) = 2^x \bmod 9$	1	2	4	8	7	5	1	2	4	8

Dann ist f: $\{0, 1, 2, 3, 4, 5\} \to \{1, 2, 4, 5, 7, 8\}$ umkehrbar — aber dass $f^{-1}(7) = 4$ ist, kann man nur aus unserer Wertetabelle ablesen, es ist „schwer berechenbar“ (natürlich nicht für so kleine Zahlen ...)

3.2 RSA

- benannt nach ihren Erfindern Ron Rivest, Adi Shamir und Len Adleman (1978)
- das erste unsymmetrische Verschlüsselungsverfahren und ist immer noch das wichtigste.
- basiert auf zwei Falltürfunktionen:

  1. Multiplikation großer Zahlen $\leftrightarrow$ Umkehrung: Faktorisierung
  2. modulare Exponenzierung $\leftrightarrow$ Umkehrung: diskreter Logarithmus

  Beide Umkehrungen sind schwer berechenbar.

- Anders ausgedrückt: Es ist einerseits sehr leicht, große Primzahlen zu finden und Potenzen modular zu berechnen, aber sehr schwer, die Primfaktorzerlegung großer Zahlen zu finden und die modulare Exponenzierung zu invertieren.

RSA — Schlüsselerzeugung

In einem *RSA Kryptosystem* erzeugt jeder Teilnehmer seine eigenen Schlüssel:

1. Wähle zwei zufällige große Primzahlen p und q, $p \neq q$. (Die Primzahlen sollten mehr als 200 Stellen haben, d.h. größer als 660 bit sein.)
2. Berechne $n = pq$ und die *Carmichael-Funktion* $\lambda(n) = \mathrm{kgV}(p-1, q-1)$.[1]
3. Wähle zufällig eine natürliche Zahl d, die teilerfremd zu $\lambda(n)$ ist. (d sollte von der Größenordnung von n sein, d.h., $d \lessapprox \lambda(n)$.)
4. Berechne die Zahl e, so dass $ed = 1 \bmod \lambda(n)$, d.h.: $ed \ \% \ n = 1$, s.o. (Dies geht effizient mit dem Erweiterten Euklid'schen Algorithmus.[2])
5. Veröffentliche das Zahlenpaar $P = (e, n)$ als deinen *öffentlichen Schlüssel*.
6. Halte das Zahlenpaar $S = (d, n)$ als deinen *privaten Schlüssel* geheim.

(Die Zahlen p und q werden nun nicht mehr benötigt, sie sollten vernichtet werden.)

- Der Schlüsselparameter e heißt auch Verschlüsselungsexponent *(encryption exponent)*, d der Entschlüsselungsexponent *(decryption exponent)*, und n der RSA-Modul *(RSA modulus)*.
- Die Veröffentlichung kann über ein öffentliche zugängliches Schlüsselverzeichnis geschehen („Telefonbuch"), oder über eine Datei auf einem Server.

[1] Das kgV berechnet man effizient, indem man den ggT verwendet: es gilt $\mathrm{kgV}(a,b) = \frac{ab}{\mathrm{ggT}(a,b)}$ für alle $a, b \in \mathbb{N}$.

[2] Fragen Sie dazu einen Wirtschaftsinformatiker der FH Südwestfalen; der kennt den Algorithmus spätestens seit seinem zweiten Semester...

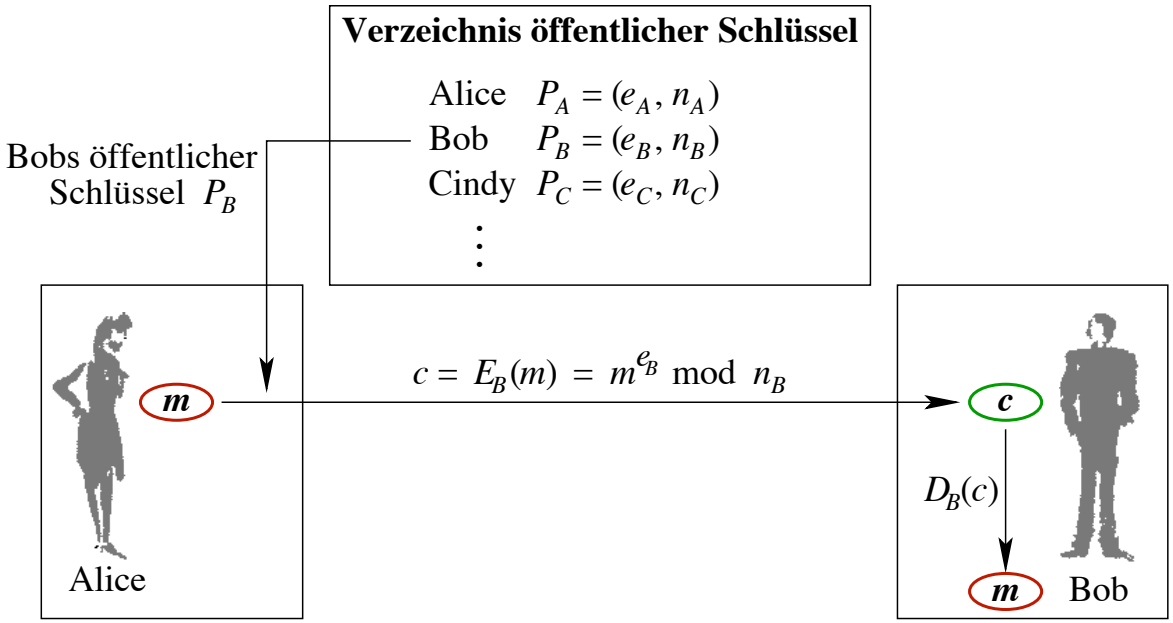
RSA — Ver- und Entschlüsselung

- Die Länge einer Nachricht darf höchstens n sein. D.h., für jede Nachricht m gilt $m \in \mathbb{Z}_n$, wobei $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$.
- Der RSA-Verschlüsselungsalgorithmus ist die Funktion $E : \mathbb{Z}_n \to \mathbb{Z}_n$,

$$E(m) = m^e \bmod n. \tag{3.2}$$

  Die Verschlüsselung der Nachricht m geschieht durch Einsetzen von m in E.
- Die Entschlüsselung eines Chiffretexts $c \in \mathbb{Z}_n$ geschieht mit Hilfe des Privaten Schlüssels $S = (d, n)$ durch die Abbildung $D : \mathbb{Z}_n \to \mathbb{Z}_n$,

$$D(c) = c^d \bmod n. \tag{3.3}$$



Beispiel 3.2 Alice hat sich zufällig die beiden Primzahlen $p = 3$ und $q = 5$ gewählt. Dann gilt $n = 15$ und $\lambda(n) = \text{kgV}(2,4) = 4$. Eine zu 4 teilerfremde Zahl ist z.B. $d = 7$. Man findet e durch Raten (oder effizienter durch den Erweiterten Euklidischen Algorithmus) z.B. als $e = 3$, denn ed % 4 = 21 % 4 = 1. Damit erhält sie den öffentlichen Schlüssel P_A und den privaten Schlüssel S_A, gegeben durch

$$P_A = (3,15), \qquad S_A = (7,15). \tag{3.4}$$

Will nun Bob die Nachricht $m = 8$ verschlüsselt an Alice schicken, so nimmt er ihren öffentlichen Schlüssel P_A und bildet daraus das Chiffrat

$$c = E(8) = 8^3 \,\%\, 15 = 512 \,\%\, 15 = 2$$

(denn $512 \div 15 = 34$ Rest 2). Alice bekommt diesen Wert $c = 2$ und entschlüsselt ihn mit ihrem privaten Schlüssel S_A:

$$m = D(2) = 2^7 \,\%\, 15 = 128 \,\%\, 15 = 8$$

(denn $128 \div 15 = 8$ Rest 8). Also sind E und D tatsächlich Umkehrfunktionen voneinader. □

Beispiel 3.3 Sei $p = 11$ und $q = 13$. Dann gilt $n = 143$ und $\lambda(n) = \text{kgV}(10,12) = 60$. Eine zu 60 teilerfremde Zahl ist z.B. $d = 23$. Ein möglicher Wert für e ist dann z.B. $e = 47$, denn $ed \ \% \ 60 = 1081 \ \% \ 60 = 1$. Damit erhalten wir

$$P_A = (47, 143), \qquad S_A = (23, 143). \tag{3.5}$$

Aus der Nachricht $m = 8$ wird dann das Chiffrat

$$\begin{aligned} c &= E(8) = 8^{47} \ \% \ 143 = 8^{3+2^2+2^3+2^5} \ \% \ 143 \\ &= \underbrace{(8^3 \ \% \ 143)}_{512\%143=83} \cdot \underbrace{(8^4 \ \% \ 143)}_{4096\%143=92} \cdot \underbrace{(8^8 \ \% \ 143)}_{27} \cdot \underbrace{(8^8 \ \% \ 143)^4}_{27^4\%143=53} \ \% \ 143 \\ &= 83 \cdot 92 \cdot 27 \cdot 53 \ \% \ 143 = 10927116 \ \% \ 143 = 57. \end{aligned}$$

Alice bekommt diesen Wert $c = 2$ und entschlüsselt ihn mit ihrem privaten Schlüssel S_A:

$$m = D(57) = 57^{23} \ \% \ 143 = (\underbrace{57^3 \ \% \ 143}_{8}) \cdot (\underbrace{57^5 \ \% \ 143)^4}_{109^4\%143=1} \ \% \ 143 = 8.$$

□

Sicherheit von RSA — Faktorisierung großer Zahlen

- Die effizienteste (bislang bekannte) Methode zur Ermittlung geht über den „kleinen Umweg“ der Primfaktorzerlegung des RSA-Moduls n:
  - Gelingt es einem Angreifer, die Zahl n in ihre Primfaktoren p und q zu zerlegen, kann er λ bestimmen und d berechnen — damit hat er Alices Geheimschlüssel „geknackt“
- Allerdings ist genau die Schwierigkeit, große Zahlen zu Faktorisieren: Uraltes Problem, das schnellste zur Zeit bekannte Verfahren ist das „Zahlenkörper-Sieb“ von Pollard (1988), $T(n) = O(\exp(\frac{64}{9}\sqrt{\ln n} \cdot \ln\ln n)^{2/3})$.

Größenordnung	**bits**	**erforderliche Operationen**	**CPU-Zeit**
$n \approx 10^{50}$	167	$1.4 \cdot 10^{10}$	14 Sekunden
$n \approx 10^{75}$	250	$9 \cdot 10^{12}$	2.5 Stunden
$n \approx 10^{100}$	330	$2.3 \cdot 10^{15}$	26.6 Tage
$n \approx 10^{200}$	665	$1.2 \cdot 10^{23}$	3.8 Mio Jahre
$n \approx 10^{300}$	1000	$1.5 \cdot 10^{29}$	$4.9 \cdot 10^{12}$ Jahre

Tabelle 3.1: Laufzeiten der Primfaktorisierung verschiedener Zahlen n auf einem 10-GHz-Rechner (mit *Quadratic Sieve*, $T(n) = O(n^{\sqrt{\ln\ln n / \ln n}})$) [5, S. 165].

- Die Firma RSA Laboratories schreibt Schlüsselbrecher-Wettbewerbe aus, zu finden unter dem URL *www.rsasecurity.com/rsalabs/challenges/*
- Letzte faktorisierte Zahl (1999): eine 512-Bit-Zahl (155 Dezimalstellen), Rechenzeit 35,7 CPU-Jahre, parallel auf 292 Computern
- Ein 1024-Bit-RSA-Modul (308 Dezimalstellen) wird ca. 2037 faktorisiert sein.

Sicherheit von RSA — Diskreter Logarithmus

- Wäre die modulare Exponenzierung *keine* Falltürfunktion, so könnte ein Angreifer eine beliebige Nachricht x mit dem öffentlichen Schlüssel modulo n exponieren,
$$y = x^e \bmod n.$$
- Da $x = x^{ed} \bmod n$, könnte er effizient den diskreten Logarithmus bilden, so dass
$$ed = \text{“}\log_{x \bmod n} 1\text{”},$$
und damit schnell den geheimen Schlüssel d erlangen.

3.3 Elliptische Kurven

Die Lösungsmenge einer Gleichung der Form

$$y^2 = x^3 + ax + b \tag{3.6}$$

mit gegebenen Konstanten a, b und zwei Unbekannten x und y wird *elliptische Kurve* genannt (Abb. 3.1).

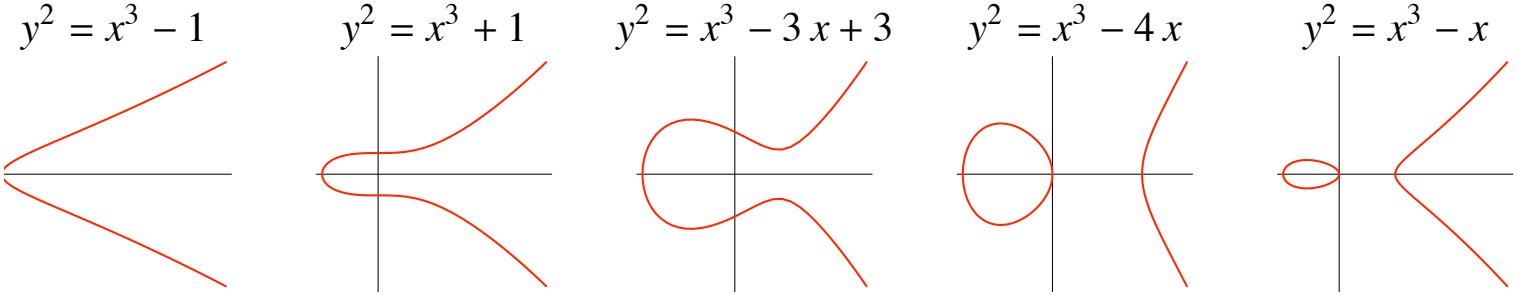


Abbildung 3.1: Verschiedene elliptische Kurven (waagerecht: x-Achse, senkrecht: y-Achse)

Jede Gerade, die nicht parallel zur y-Achse ist, hat entweder einen oder drei Schnittpunkte mit der Kurve (wenn man Tangentenpunkte doppelt zählt).

Arithmetik auf elliptischen Kurven

- Man kann auf elliptischen Kurven „geometrisch rechnen“
- Sind P und Q zwei Punkte auf einer gegebenen elliptischen Kurve, so definieren sie eine Gerade, deren dritter Schnittpunkt $-P-Q$ ist und gespiegelt an der x-Achse den Punkt $P+Q$ ergibt (Abb. 3.2).

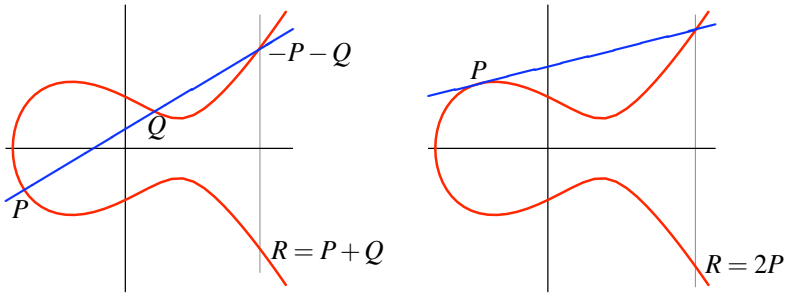


Abbildung 3.2: Addition zweier Punkte auf der elliptischen Kurve $y^2 = x^3 - 3x + 3$

- Auf diese Weise *definiert* man eine Addition auf der elliptischen Kurve. Zur Addition eines Punktes P mit sich selbst nimmt man die Tangente in P (doppelter Schnittpunkt!), sucht deren dritten Schnittpunkt und spiegelt ihn, so dass $2P = R$.
- Wenn man nun noch den ***„Punkt im Unendlichen“*** (also alle Punkte des ‚Horizonts’ der Ebene zusammengefasst zu einem Punkt) als Null betrachtet, so gilt $P+0 = P$ und $P-P = 0$, denn eine senkrechte Gerade hat immer einen Schnittpunkt mit der Kurve im Unendlichen.

Formeln für Addition auf elliptischen Kurven

Wenn $P = (p_x, p_y)$, so gilt für $R = 2P$, mit $R = (r_x, r_y)$

$$r_x = \left(\frac{3p_x^2 + a}{2p_y}\right)^2 - 2p_x, \qquad r_y = -p_y + \left(\frac{3p_x^2 + a}{2p_y}\right)(p_x - r_x); \tag{3.7}$$

für $P = (p_x, p_y)$, $Q = (q_x, q_y)$ und $S = P + Q$ gilt mit $S = (s_x, s_y)$,

$$s_x = \left(\frac{q_y - p_y}{q_x - p_x}\right)^2 - p_x - q_x, \qquad s_y = -p_y + \left(\frac{q_y - p_y}{q_x - p_x}\right)(p_x - s_x). \tag{3.8}$$

Diskrete Arithmetik auf elliptischen Kurven

- Kryptologisch wichtig sind nun ganzzahlige Werte für x und y, die auf einer gegebenen elliptischen Kurve liegen.

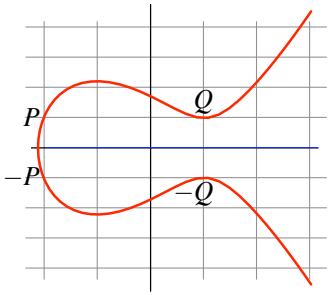


Abbildung 3.3: Ganzzahliges Gitter auf der elliptischen Kurve $y^2 = x^3 - 3x + 3$. In dem Bildausschnitt schneidet das Gitter die Kurve nur in $P = (-2,1)$, $-P = (-2,-1)$, $Q = (1,1)$ und $-Q = (1,-1)$. Die nächsten Punkte sind erst $\pm R = (16289, \pm 35327)$. Es gilt übrigens $P + Q = -Q$ und $2Q = -P$.

- Da solche ganzzahligen Werte durch ein Gitter ($\mathbb{Z} \times \mathbb{Z}$) gegeben sind, betrachtet man also nur dessen Schnittpunkte mit der Kurve (Abb. 3.3).

- Wenn man nun nur noch modulo q rechnet, wo q eine Primzahlpotenz ist ($q = p^k$ für p prim und $k \in \mathbb{N}$), so funktioniert die Addition immer noch [4, S. 174ff]. In den Formeln (3.7) und (3.8) müssen die Nenner jeweils durch Multiplikation mit $(2p_y)^{-1}$ bzw. $(q_x - p_x)^{-1}$ ersetzt werden, wobei $k = n^{-1}$ das Inverse modulo q von n ist, d.h. $nk \bmod q = 1$.

Beispiel 3.4 Betrachten wir die elliptische Kurve $y^2 = x^3 + 2x + 3 \bmod 5$. (Beachte: $2 = -3 \bmod 5$). Dann schneidet die Kurve das Gitter $\mathbb{Z}_5 \times \mathbb{Z}_5$ in den vier Punkten $P = (x,y) = (3,1)$, $-P = (3,4)$, $Q = (1,1)$ und $-Q = (1,4)$. Dann gilt:

n	1	2	3	4
nP	$(3,1)$	$(3,4)$	0	$(3,1)$

□

- In der Kryptologie werden große Zahlen für a, b und q verwendet und ein Punkt P auf der elliptischen Kurve festgelegt.
- Mit Hilfe der Formeln (3.7) und (3.8) kann man nun effizient den Punkt nP für ein sehr große $n < q$ berechnen:
  - Entsprechend der binären Darstellung von n werden nur die 2-er-Potenzen P, $2P$, 2^2P, 2^3P, ..., addiert, deren Faktor für n nicht 0 ist
  - beispielsweise wird $n = 13$ binär durch $n = 1101_2$ dargestellt, d.h.,
    $$13P = P + 4P + 8P.$$
  - Man benötigt so höchstens $\log_2 n$, also $O(\log n)$ Additionen.
- Tatsächlich ist kein Algorithmus bekannt, der effizient n aus $Q = nP$ berechnet.
- Versucht man z.B. sukzessive alle Vielfachen von P durch, also P, $2P$, $3P$, ..., und vergleicht sie mit Q, so kommt man natürlich zum Erfolg, aber benötigt n, also $O(n)$, Additionen

Sicherheit von Verfahren mit elliptischen Kurven

- Die Falltürfunktion ist die ganzzahlige Multiplikation nP des Punktes P für große n.
  - Sind die Parameter hinreichend groß (ist zum Beispiel q prim und mehr als 160 bit lang), ist der Computer ohne weiteres in der Lage, sehr schnell (in wenigen Bruchteilen einer Sekunden) den Punkt $\mathfrak{m}(n) = nP$ zu bestimmen,
  $$T_{\mathfrak{m}}(n) = O(\log n).$$
  - Das inverse Problem, n aus $Q = nP$ und P zu erhalten, ist jedoch nicht in akzeptabler Zeit möglich,
  $$T_{\mathfrak{m}^{-1}}(n) = O(n).$$
- Dies wird als das „Diskrete Logarithmus Problem über Elliptischen Kurven“ bezeichnet, kurz ***ECDL-Problem*** *(Elliptic Curve Discrete Logarithm Problem)*.
- Mit einer Länge von zum Beispiel 200 Bit für q ist eine gute elliptische Kurve genau so sicher wie ein RSA-Modulus von über 1024 bit Länge (zumindest nach dem heutigen Forschungstand).
- Eine Implementierung eines auf elliptischen Kurven und der Java Cryptography Architecture (JCA/JCE) basierenden Algorithmus ist OpenSource herunterladbar unter

  http://www.flexiprovider.de/

3.4 Kryptanalyse: Man-in-the-Middle-Angriff

- Der *Man-in-the-Middle-Angriff* besteht darin, dass ein Angreifer jeglichen Informationsaustausch zwischen Alice, Bob und dem öffentlichen Schlüsselverzeichnis abfängt und in seinem Sinne manipuliert.
- Möchte beispielsweise Alice eine Nachricht an Bob verschlüsseln und versucht, seinen öffentlichen Schlüssel abzurufen, so übermittelt der Angreifer ihr seinen eigenen.
- Daraufhin verschlüsselt Alice die Nachricht mit dem Angreiferschlüssel, im Glauben nur Bob kann die Nachricht jetzt lesen.
- Der Angreifer entschlüsselt die Nachricht jedoch mit seinem eigenen Schlüssel, verändert sie gegebenfalls, und sendet sie mit Bobs öffentlichen Schlüssel an Bob.
- Weder Alice noch Bob merken, dass ihre Kommunikation abgehört wird.

Checkliste der Kryptoprobleme

Checkliste für unsymmetrische Chiffren
Schlüsselaustausch gelöst?
Authentifizierung?
Abhörerkennung?

Checkliste der Kryptoprobleme

Checkliste für unsymmetrische Chiffren	
Schlüsselaustausch gelöst?	je nachdem
Authentifizierung?	
Abhörerkennung?	

Falls, wie es in der Praxis oft geschieht, die öffentlichen und privaten Schlüssel von einer zentralen Schlüsselverwaltung erzeugt und verteilt werden, bleibt das Problem des sicheren Kanals.

Checkliste der Kryptoprobleme

Checkliste für unsymmetrische Chiffren	
Schlüsselaustausch gelöst?	je nachdem
Authentifizierung?	ja
Abhörerkennung?	

Falls, wie es in der Praxis oft geschieht, die öffentlichen und privaten Schlüssel von einer zentralen Schlüsselverwaltung erzeugt und verteilt werden, bleibt das Problem des sicheren Kanals.

Checkliste der Kryptoprobleme

Checkliste für unsymmetrische Chiffren	
Schlüsselaustausch gelöst?	je nachdem
Authentifizierung?	ja
Abhörerkennung?	nein

Falls, wie es in der Praxis oft geschieht, die öffentlichen und privaten Schlüssel von einer zentralen Schlüsselverwaltung erzeugt und verteilt werden, bleibt das Problem des sicheren Kanals.

3.6 Vergleich mit symmetrischen Chiffren — Sicherheit

- Falltürfunktionen benötigen in der Regel sehr große Zahlen (z.B.: RSA-Modul $\geqq$ 1 kbit, um als derzeit sicher zu gelten)
- Der tatsächliche Schlüsselraum ist viel kleiner als der benötigte Speicherplatz: denn nicht alle Zahlen $\leqq$ 1 kbit können mögliche RSA-Schlüssel sein, Zahlen mit zwei Primfaktoren sind „dünn“ gesät
- Ein symmetrisches Verfahren kann den gesamten Schlüsselraum ausschöpfen

symmetrisch	**unsymmetrisch**	**Verhältnis**
40 bit	–	–
56 bit	(400 bit)	1:7,1
64 bit	512 bit	1:8
80 bit	768 bit	1:9,6
90 bit	1024 bit	1:11,4
112 bit	1792 bit	1:16
120 bit	2048 bit	1:17,1
128 bit	2304 bit	1:18

Tabelle 3.2: Vergleich der erforderlichen Schlüssellängen symmetrischer und unsymmetrischer Chiffren bei gleicher Sicherheit. Als aktuell sichere Schlüssellängen gelten die Zeilen unter dem Strich.

- Bei den gängigen Protokollen wie TSL (SSL) wird ein symmetrischer Schlüssel nur einmal pro Sitzung verwendet wird, ein öffentlicher aber für hunderte bis tausende von Nachrichten.
- Da zudem ein öffentlicher Schlüssel juristische Gültigkeit hat (haben kann), wiegt das Brechen eines unsymmetrischen Schlüssels schwerer als eines symmetrischen.

Vergleich mit symmetrischen Chiffren

Schlüsselaustausch

- Eine wesentliche Schwachstelle aller symmetrischen Chiffren ist der Austausch bzw. die Vereinbarung des Schlüssels.
- Dazu benötigen die Kommunikationspartner einen sicheren Kanal (Kurier!)

Performanz

- Software-Implementierungen von RSA sind derzeit um mehr als den Faktor 1000 langsamer als AES (vgl. [3, §5.2.4] und S. 24).
- Ursache: erheblicher Rechenaufwand (modulare Exponentiation) der Ver- und Entschlüsselung in RSA, im Vgl. zu den schnellen Byte-Operationen von AES
- Während RSA-Verschlüsselungen nicht wesentlich mehr als einige hundert kbit/s bewältigen, gelingt AES ein Datendurchsatz von 1 Gbit/s auf einem 5-GHz-Rechner.
- Bei Hardware-Implementierungen ist das Geschwindigkeitsverhältnis noch ungünstiger für unsymmetrische Verfahren, man kann von einem Faktor $>$ 10 000 für das Verhältnis RSA/AES ausgehen. Zur Zeit erreichen Hardware-Implementierungen von RSA maximal 64 kbit/s.

Vergleich mit symmetrischen Chiffren — Fazit

- Symmetrische Verschlüsselungsverfahren sind schneller als unsymmetrische,
- besitzen aber das grundsätzliche Problem des Schlüsselaustauschs
- Unsymmetrische Verfahren eröffnen ganz neue Funktionen in Kommunikationsystemen, wie
  - Authentifizierung und
  - digitale Unterschriften,
- In der Realität werden beide Verfahren eingesetzt:

  In TSL (früher: SSL) werden unsymmetrische Verfahren zur Authentifizierung und zum Austausch eines Geheimschlüssels verwendet, der eigentliche verschlüsselte Nachrichtenaustausch geschieht mit einer schnelleren symmetrischen Chiffre.
- Man nennt solche Verfahren *hybrid*.[3]

[3] *hybrid:* gemischt, (aus Verschiedenem) zusammengesetzt

3.7 Implementierung in Java

Die Erzeugung und Ausgabe eines RSA-Schlüsselpaares ist in Java sehr einfach. Es werden lediglich die Klassen `KeyPairGenerator` und `KeyPair` aus dem Paket `java.security` benötigt:

```
import java.security.*;

public class RSATest {
   public static void main(String[] args) {
      try {
         KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
         keyGen.initialize(1024);
         KeyPair pair = keyGen.generateKeyPair();

         String ausgabe = "Geheimschlüssel: " + pair.getPrivate();
         ausgabe += "\nÖffentlicher Schlüssel: " + pair.getPublic();
         javax.swing.JOptionPane.showMessageDialog(null, ausgabe, "PKS", -1);
      } catch ( NoSuchAlgorithmException nsae ) {
         System.err.println("Algorithmus zur Schlüsselerzeugung existiert nicht!");
         //nsae.printStackTrace();
      } finally {
         System.exit(0);
      }
   }
}
```

Da die statische `getInstance`-Methode, die eine Instanz des `KeyPairGenerators` erzeugt, einen Ausnahmefehler wirft, wenn der genannte Algorithmus nicht existiert, muss sie in einen `try`/`catch`-Block.

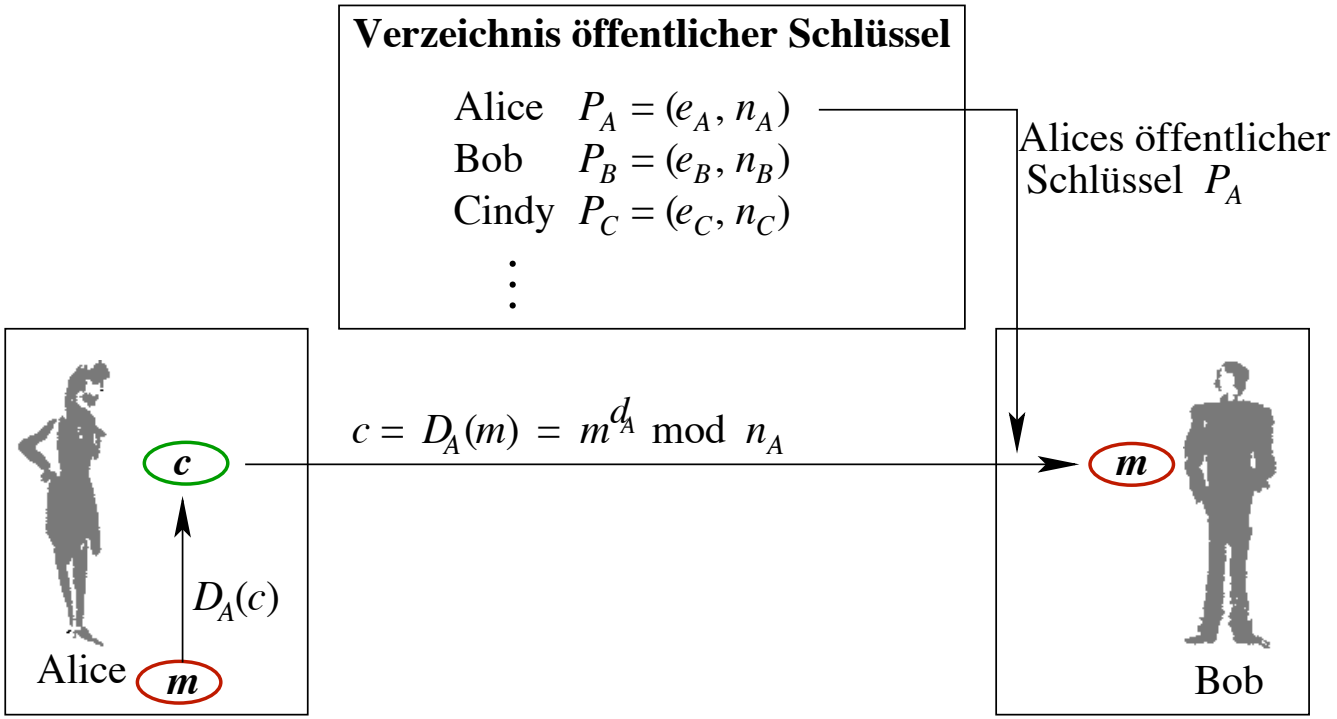
Digitale Signatur — Eigenschaften

1. Sie ist *authentisch:* Das Dokument wurde vom Absender unterzeichnet. (z.B. handschriftliche Unterschrift im Beisein eines Notars)
2. Sie ist *fälschungssicher:* Kein anderer hat das Dokument unterzeichnet. (Fälschungssicherheit durch die individuelle Handschrift)
3. Sie ist *nicht wiederverwendbar:* Die Unterschrift ist Bestandteil des unterzeichneten Dokuments und in kein anderes Dokument übertragbar. (Handschriftliche Unterschriften sind ungültig, wenn sie kopiert oder gefaxt sind.)
4. Sie ist *integer mit dem Inhalt des Dokuments:* Nachdem das Dokument unterschrieben ist, kann es nicht mehr geändert werden. (Verträge werden doppelt ausgefertigt, Veränderungen müssen identisch bei beiden Versionen sein.)
5. Sie ist *verbindlich*. Der Unterzeichner kann später nicht behaupten, er habe das Dokument nicht unterschrieben oder abgesendet.

4.8 Digitale Unterschriften mit RSA

Grundidee: der geheime Schlüssel des Senders dient als seine digitale Unterschrift. Folgendes Protokoll ist möglich:

1. Alice verschlüsselt die Nachricht m mit ihrem geheimen Schlüssel S_A: $y = D_A(m)$;
2. Diese verschlüsselte Nachricht verschickt sie an Bob;
3. Bob wendet Alices öffentlichen Schlüssel P_A an und erhält die ursprüngliche Nachricht m.



Digitale Unterschriften mit RSA

- Das signierte Dokument kann *von jedem* entschlüsselt werden kann, der Zugriff auf Alices öffentlichen Schlüssel hat.
- Will Alice ein Dokument signieren, was nur Bob lesen darf, so verschlüsselt Alice das signierte Dokument mit Bobs öffentlichem Schlüssel.
- Eigenschaften einer Unterschrift sind erfüllt, solange der öffentliche Schlüssel authentisch ist und der geheime Schlüssel niemand Anderem bekannt ist:
  1. *authentisch:* Bob weiß, dass die Nachricht von Alice unterzeichnet ist, da er ihren öffentlichen Schlüssel erfolgreich angewendet hat.
  2. *fälschungssicher:* Nur Alice kennt ihren geheimen Schlüssel
  3. *nicht wiederverwendbar:* Die Unterschrift ist eine spezifische Eigenschaft des Dokuments, sie kann nicht in ein anderes übertragen werden.
  4. *integer:* Würde das Dokument unterwegs geändert, kann es mit Alices öffentlichen Schlüssel nicht mehr entziffert werden.
  5. *verbindlich:* Bob (und ggf. jeder Richter) kann ohne Alices Unterstützung ihre Unterschrift mit ihrem öffentlichen Schlüssel verifizieren.
- Für die rechtliche Gültigkeit der Unterschrift muss die Authentizität von Alices öffentlichem Schlüssel P_A garantiert sein. Nur dann kann Bob annehmen, dass es tatsächlich ihre Unterschrift ist. $\Longrightarrow$ *Trust Center*

4.9 DSA *(Digital Signature Algorithm)*

- seit 1994 der Standard für digitale Signaturen von Dokumenten der US-Regierung und wurde von der NIST und dem NSA entwickelt. (Spezifikation FIPS-186 *http://www.itl.nist.gov/fipspubs/fip186.htm*)
- Er geht auf Signierungalgorithmen von Schnorr und ElGamal zurück [6, S. 564].

Schlüsselerzeugung

1. Sie wählt eine Primzahl p der Länge n bit, wobei $512 \leqq n \leqq 1024$ und ein Vielfaches von 64 ist.
2. Sie wählt einen 160 bit langen Primfaktor q von $p-1$, d.h. $q \mid p$ und $2^{159} < q < 2^{160}$.
3. Sie wählt eine beliebige Zahl z mit $0 < z < p-1$ und $z^{(p-1)/q} \bmod p \neq 1$.
4. Sie berechnet $g = z^{(p-1)/q} \bmod p$.
5. Sie wählt eine beliebige Zahl $S_A < q$.
6. Sie berechnet $P_A = g^x \bmod p$.

Die drei Parameter p, q und g sind öffentlich bekannt und können innerhalb des Kommunikationsnetzes fest vorgegeben sein. S_A ist Alices öffentlicher Schlüssel, S_A ihr geheimer.

Signierung in DSA

Alice signiert eine Nachricht m mit Hilfe der Hash-Funktion SHA (s.u.) in drei Schritten:

1. Sie erzeugt eine Zufallszahl $k < q$.

2. Sie berechnet

$$\begin{aligned} r &= (g^k \bmod p) \bmod q, \\ s &= k^{-1}(\mathrm{SHA}(m) + rS_A) \bmod q. \end{aligned} \tag{4.9}$$

   Hierbei ist SHA eine allen Kommunikationsteilnehmern bekannte Hash-Funktion und S_A Alice geheimer Schlüssel. Die Inverse k^{-1} modulo q errechnet Alice effizient mit Hilfe des Erweiterten Euklidischen Algorithmus.

3. Sie sendet die Zahlen (m, r, s) als Signatur an Bob.

Verifikation der Signatur in DSA

- Bob muss die Parameter p, q und g kennen. Gegebenenfalls sind sie in dem Kommunikationsnetz vorgegeben.
- Außerdem benötigt er Alices öffentlichen Schlüssel P_A, dessen Echtheit er voraussetzen kann.

Mit den folgenden Schritten verifiziert er dann Alices Signatur (m, r, s).

1. Er berechnet sukzessive:

$$\begin{aligned} w &= s^{-1} \bmod q, \\ u_1 &= w \cdot \mathrm{SHA}(m) \bmod q, \\ u_2 &= rw \bmod q, \\ v &= (g^{u_1} P_A^{u_2} \bmod p) \bmod q, \end{aligned} \tag{4.10}$$

2. Wenn $v = r$, so ist die Signatur verifiziert.

4.10 Implementierung in Java

```
import java.security.*;

public class DSASignatureTest {
   public static void main(String[] args) {
      String doc1 = "Vertrag über den Kauf der Schlossallee.\nDer Preis beträgt 1 Mio \u20A
      String doc2 = "\nZusatzvereinbarung: Gehe nicht über Los!";
      String doc3 = "\nZiehe keine 2000 \u20AC ein!";
      byte[] i1 = doc1.getBytes();
      byte[] i2 = doc2.getBytes();
      byte[] i3 = doc3.getBytes();

      try{
         // Erzeuge DSA-Schlüssel:
         KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
         keyGen.initialize(1024);
         KeyPair pair = keyGen.generateKeyPair();

         // Signiere Dokument:
         Signature signatur = Signature.getInstance("SHA1withDSA");
         signatur.initSign( pair.getPrivate() );

         signatur.update(i1);
         signatur.update(i2);
         signatur.update(i3);
         byte[] unterschrift = signatur.sign();

         // Verifizierung:
         Signature pruef = Signature.getInstance("SHA1withDSA");
         pruef.initVerify( pair.getPublic() );
         pruef.update(i1);
         pruef.update(i2);
         pruef.update(i3);
         boolean verifiziert = pruef.verify( unterschrift );

         // ----------- Ausgabe: --------------------
```

```
            String ausgabe = "Nachricht: ------------\n";
            ausgabe += doc1 + doc2 + doc3;
            ausgabe += "\n... als byte-Arrays: --------\n";
            for ( int i = 0; i < i1.length; i++ ) {
               ausgabe += i1[i] + " ";
            }
            ausgabe += "\n";
            for ( int i = 0; i < i2.length; i++ ) {
               ausgabe += i2[i] + " ";
            }
            ausgabe += "\n";
            for ( int i = 0; i < i3.length; i++ ) {
               ausgabe += i3[i] + " ";
            }
            ausgabe += "\n... und als Signatur in " + unterschrift.length + " bytes: --------\
            for ( int i = 0; i < unterschrift.length; i++ ) {
               ausgabe += unterschrift[i] + " ";
            }

            ausgabe += "\nSignatur ";
            ausgabe += verifiziert ? "OK!" : "nicht OK!";

            System.out.println(ausgabe);
            //javax.swing.JOptionPane.showMessageDialog(null, ausgabe, "Hash-Wert", -1);
         } catch ( NoSuchAlgorithmException nsae ) {
            System.err.println("Algorithmus existiert nicht!");
            //nsae.printStackTrace();
         } catch ( InvalidKeyException ike ) {
            System.err.println("Schlüsselfehler!");
            //ike.printStackTrace();
         } catch ( SignatureException se ) {
            System.err.println("Konnte nicht signieren!");
            //se.printStackTrace();
         }
      }
   }
```

Kapitel 5

Hash-Funktionen

- In der Praxis wird man kein komplettes Dokument verschlüsseln, sondern nur seinen Hash-Wert.
- Ein *Hash-Wert* ist eine Art Prüfziffer von fester Länge, berechnet aus einer beliebig langen Nachricht mit Hilfe einer öffentlich bekannten *Hash-Funktion*.

Definition 5.1 Eine ***Hash-Funktion*** (manchmal auch: *Kompressionsfunktion*) ist eine Funktion $h : W \to H$ einer (auch unendlich großen) Menge W von „Worten“ oder „Nachrichten“ auf eine endliche Menge $H \subset \mathbb{Z}$ von Hash-Werten, für die gilt:

- $h(m)$ ist leicht berechenbar;
- zu einem gegebenem Hash-Wert y ist es schwer, ein Wort m mit $h(m) = y$ zu finden;
- zu einem gegebenem Wort m ist es schwer, ein zweites Wort m' mit $h(m) = h(m')$, also gleichem Hash-Wert, zu finden.

□

Hash-Funktionen

- Zweck eines Hash-Wertes: Veränderungen der Nachricht während des Übermittlungsweges sind überprüfbar, seien es Störungen oder Angriffe.
- Der Sender übermittelt die Nachricht m und ihren Hash-Wert $h(m)$,

  $$(m, h(m)).$$

  Der Empfänger berechnet einfach den Hash-Wert der Nachricht m, die er empfängt, und vergleicht ihn mit dem Hash-Wert in der Sendung. Stimmen beide überein, so ist die Nachricht unverändert.
- Eine Hash-Funktion kann nicht umkehrbar sein, denn sie bildet eine riesige Menge auf eine vergleichsweise kleine Menge von Hash-Werten ab.
- Es *muss* also mehrere Worte geben, die denselben Hash-Wert haben.
- Findet man zwei Worte $w^{(1)}$ und $w^{(2)}$ mit $h(w^{(1)}) = h(w^{(2)})$, so spricht man von einer ***Kollision***.
- Alle Worte, die zu einem Hash-Wert gehören, bilden sein ***Urbild*** *(preimage)*.

Beispiel 5.2 Sei $h : \{0,1\}^* \to \{0,1\}$,

$$h(w) = w_n \oplus \ldots \oplus w_1$$

die XOR-Verknüpfung eines beliebig langen Bit-Strings. Z.B. ist $h(101) = 1 \oplus 0 \oplus 1 = 0$. h ist eine (sehr einfache) Hash-Funktion, und 0 ist der Hash-Wert von 101. Die Länge der Eingabe ist beliebig, die Ausgabe ist entweder 0 oder 1, also ein Bit. Da $h(1001) = 0$, gibt es für die beiden verschiedenen Worte $w^{(1)} = 101$ und $w^{(2)} = 1001$ eine Kollision. □

- Verschiedene Nachrichten liefern – bei einer „guten“ Hash-Funktion – mit hoher Wahrscheinlichkeit verschiedene Hash-Werte, so dass sie als Fingerabdruck bezeichnet werden können.
- Eine „gute“ Hash-Funktion berechnet effizient den Hashwert ordnet und mit hoher Wahrscheinlichkeit verschiedenen Nachrichten verschiedene Hashwerte zu, ohne dass sich umgekehrt die Nachricht aus ihrem Hashwert effizient berechnen lässt.
- Die einfache Hash-Funktion aus Beispiel 5.2 ist in diesem Sinne keine gute Hash-Funktion, denn die Hälfte von zufällig ausgewählten Bit-Strings („Worten“) hat den Hash-Wert 0, die andere 1.

Überblick über gängige Hash-Funktionen

Hash-Funktion	Blocklänge	relative Geschwindigkeit
MD4	128 bit	1,00
MD5	128 bit	0,68
RIPEMD-128	128 bit	0,39
SHA-1	160 bit	0,28
RIPEMD-160	160 bit	0,24

- Interessanterweise bauen alle diese Hash-Funktionen auf dem von Ron Rivest Ende der 1980er entwickelten MD4 auf.
- RIPEMD-160 gilt als sehr sicher.
- SHA-1 ist der aktuelle internationale Standard.

5.1 SHA *(Secure Hash Algorithm)* — Ablauf

1. *Aufteilen der Nachricht in 512-bit-Blöcke:* Die Nachricht m wird so aufgefüllt, dass ihre Länge ein Vielfaches von 512 bit beträgt.

2. *Bildung von 80 Wörtern à 32 bit:* Jeder 512-bit-Block wird in 16 Blöcke M_0, …, M_{15} à 32 bit geteilt, daraus 80 Wörter $W_0, \ldots, W_{79}$ gebildet:

$$W_t = \begin{cases} M_t & \text{wenn } 0 \leqq t \leqq 15, \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1 & \text{sonst.} \end{cases}$$

   ($\lll$: *Bit-Rotation*, z.B. $\texttt{10100} \lll \texttt{1} = \texttt{01001}$).

3. *Initialisierung der Variablen und Konstanten:*

$$K_t = \begin{cases} \texttt{0x5A827999} = \lfloor \sqrt{2} \cdot 2^{30} \rfloor & \text{wenn } 0 \leqq t \leqq 19, \\ \texttt{0x6ED9EBA1} = \lfloor \sqrt{3} \cdot 2^{30} \rfloor & \text{wenn } 20 \leqq t \leqq 39, \\ \texttt{0x8F1BBCDC} = \lfloor \sqrt{5} \cdot 2^{30} \rfloor & \text{wenn } 40 \leqq t \leqq 59, \\ \texttt{0xCA62C1D6} = \lfloor \sqrt{10} \cdot 2^{30} \rfloor & \text{wenn } 60 \leqq t \leqq 79, \end{cases}$$

$$A = \texttt{0x67452301}, \qquad B = \texttt{0xEFCDAB89}, \qquad C = \texttt{0x98BADCFE},$$

$$D = \texttt{0x10325476}, \qquad E = \texttt{0xC3D2E1F0}.$$

   und fünf Variablen $A, \ldots, E$ und $a, \ldots, e$, verwendet, die wie folgt initialisiert werden:

$$a = A, \qquad b = B, \qquad c = C, \qquad d = D, \qquad e = E.$$

   Alle Konstanten und Variablen haben 32 bit = 8 Bytes.

4. *Die Hauptschleife:*

for ($t = 0; t \leqq 79; t{+}{+}$) {

$\text{tmp} = (a \lll 5) + f_t(b,c,d) + e + W_t + K_t;$

$e = d;$

$d = c;$

$c = b \lll 30;$

$b = a;$

$a = \text{tmp};$

}

Hierbei ist die nichtlineare Funktionenschar f_t definiert durch

$$f_t(x,y,z) = \begin{cases} (x \wedge y) \vee (\neg x \wedge z) & \text{wenn } 0 \leqq t \leqq 19, \\ (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) & \text{wenn } 40 \leqq t \leqq 59, \\ x \oplus y \oplus z & \text{sonst.} \end{cases}$$

5.2 Angriffe auf Hash-Funktionen

1. *Urbild-Angriff (pre-image attack):* Wie schwer ist es, zu einem vorgegebenen Hash-Wert eine Nachricht zu erzeugen, die denselben Hash-Wert ergibt? Hat der Hash-Wert eine Länge von n bits, so benötigt man erwartungsgemäß 2^n Versuche.

2. *Kollisionsangriff („Geburtstagsangriff"):* Wie schwer ist es, zwei verschiedenen Nachrichten mit gleicher Prüfsumme zu finden? Hat der Hash-Wert eine Länge von n bits, so benötigt man erwartungsgemäß dazu $\sqrt{2^n}$, also $2^{n/2}$ Versuche.

- Beide Angriffe lassen sich theoretisch mit *Brute Force* realisieren.
- SHA-1 bildet Hash-Werte mit 160 Bit, d.h. $2^{160} \approx 1,5 \cdot 10^{48}$ Hash-Werte
- Probiert man bei vorgegebenem Hash-Wert 2^{160} zufällige Nachrichten durch, erhält man mit hoher Wahrscheinlichkeit eine mit dem gleichen Hash-Wert (dauert mit reeller Hardware weit über 100 Millionen Jahre)
- Will man nur eine Kollision finden, also zwei Nachrichten mit dem gleichen aber beliebigen Hash-Wert, muss man ‚nur' $2^{80} \approx 1,2 \cdot 10^{24}$ zufällig ausgewählte Nachrichten durchprobieren.
- Urbild-Angriffe auf SHA-1 erfordern damit *nicht doppelt* soviele Versuche, sondern 2^{80} mal so viele Hash-Operationen wie ein Kollisionsangriff.

Das Geburtstagsparadoxon

- Den enormen Unterschied zwischen dem Aufwand für Urbild- und Kollisionsattacken veranschaulicht das Geburtstagsparadoxon.
- Wenn Sie jemanden suchen, der am selben Tag Geburtstag hat wie Sie, müssen sie für eine Trefferwahrscheinlichkeit von 50 Prozent **253** Leute fragen.
- Ist Ihnen der konkrete Geburtstag egal und Sie suchen nur zwei Personen mit dem gleichen, genügen viel weniger. Bereits bei **23** Menschen ist die Chance fünfzig Prozent, dass zwei davon am selben Tag Geburtstag haben.

5.3 MAC *(message authentication code)*

- Hash-Wert unter Verwendung einer Hash-Funktion, die von einem geheimen Schlüssel abhängt.
- Wenn m die Nachricht ist und E_S die Verschlüsselungsfunktion mit dem Geheimschlüssel S, so gilt für den MAC der Nachricht m

$$\mathrm{MAC}(m) = (m, E_S[h(m)]). \tag{5.1}$$

- Damit wird verhindert, dass ein Angreifer die Nachricht verändert ($m \mapsto m'$) und den Hash-Wert ($h(m')$) einfach neu berechnet, denn für die Berechnung benötigt er den Geheimschlüssel S.
- Der MAC kann in zwei Varianten erstellt werden, mit einem unsymmetrischen Verschlüsselungsverfahren oder einer symmetrischen Chiffre:
  - ***Signierter MAC:*** Will Alice die Nachricht m signieren, so berechnet sie den Hash-Wert $h(m)$ und signiert ihn: $E_{S_A}[h(m)]$.
  - ***Symmetrisch verschlüsselter MAC:*** Alice verschlüsselt den Hash-Wert $h(m)$ ihrer Nachricht m einfach mit einem mit Bob vereinbarten symmetrischen Verfahren, $E_S[h(m)]$. Nur Bob kennt neben Alice den geheimen Schlüssel S und kann den Hash-Wert überprüfen.

5.4 TLS (bzw. SSL)

- Die Firma Netscape stellte 1994 das SSL-Protokoll *(Secure Sockets Layer)* vor, das für verschlüsselte HTTP-Verbindungen gedacht war.
- Es ist die Grundlage für das Standardprotokoll TLS *Transport Layer Protocol*. TLS.
- TLS basiert auf *Zertifikaten* von ***Trust-Centern*** oder CA *(Certification Authority)*, also Institutionen, die öffentliche Schlüssel verwalten und deren Authentizität garantieren.
- Ein Zertifikat eines Netzteilnehmers besteht im Wesentlichen aus seinem Namen und seinem öffentlichen Schlüssel und wird von dem Trust-Center signiert. Damit garantiert es die Verbindung von Namen und öffentlichem Schlüssel.

TSL — Protokoll

1. *Authentifizierung:* Alice (A) überprüft die Identität ihres Gesprächspartners Bob (B) durch den folgenden kurzen Dialog:

$A \rightarrow B$:	Hallo!
$B \rightarrow A$:	Hallo, ich bin Bob, *Zertifikat(Bob)*.
$A \rightarrow B$:	Beweise es!
$B \rightarrow A$:	$D_B[m, h(m)]$, mit $m =$ „Alice, hier ist Bob“.

   Alice kann nun Bobs signierte Nachricht durch seinen öffentlichen Schlüssel aus seinem Zertifikat verifizieren.

2. *Austausch eines symmetrischen Geheimschlüssels:* Alice erzeugt einen zufälligen Geheimschlüssel K und übermittelt ihn verschlüsselt an Bob:

$A \rightarrow B$:	$E_B(K)$

   K heißt auch Sitzungsschlüssel *(session key)*.

3. *Nachrichtenaustausch:* Nun können die Nachrichten mit einem vereinbarten symmetrischen Verschlüsselungsverfahren (z.B. AES) und dem A und B bekannten symmetrischen Schlüssel K ausgetauscht werden.

$A \leftrightarrow B$:	$E_K[m, h(m)]$

   Die Nachricht m wird mit ihrem Hash-Wert verschickt, um durch Angreifer veränderte Daten zu erkennen.

5.5 Angriffe auf signierte Dokumente — Fälschungen

- Um einen digital signierten Vertrag nachträglich zu fälschen, müsste der Angreifer einen Urbild-Angriff durchführen.
- Er müsste also zum vorgegebenen Hash-Wert des echten Vertrags einen zweiten, gefälschten Vertrag finden, der denselben Hash-Wert ergibt.
- Das erfordert schon prinzipiell sehr viel mehr Operationen (2^{160} bei SHA-1).
- Alle bekannten Angriffe beziehen sich nur auf Kollisionen.
- Verfahren, die die Anzahl der benötigten Operationen für Urbild-Angriffe deutlich reduzieren, sind bisher nicht bekannt.
- Mit Kollisionen, die sich in realistischer Zeit errechnen lassen, sind durchaus Angriffe auf digitale Signaturen möglich, wenn der Angreifer den Hash-Wert des Originals frei bestimmen kann.

Beispiel 5.3 *(Fälschung eines Vertrags)*

- Ein intelligenter Angreifer erstellt zwei Verträge, einen mit dem richtigen Kaufpreis für ein Haus und einen mit einer sehr viel höheren Summe.
- Da er bei einem Hash-Wert der Größe n bits weiß, dass er mit $2^{n/2}$ Hash-Operationen eine Kollision finden kann, überlegt er sich $\frac{n}{2}$ kosmetische Änderungen wie zusätzliche Leerzeichen am Zeilenende.
- Indem er alle möglichen Kombinationen der Änderungen durchführt, erzeugt er jeweils $2^{n/2}$ Versionen der beiden Verträge, unter denen sich mit hoher Wahrscheinlichkeit ein Paar mit gleichem Hash-Wert findet.
- Lässt er sein Opfer dann das Exemplar mit dem richtigen Kaufpreis unterschreiben, kann er im Nachhinein den Text ersetzen, ohne dass sich der Hash-Wert ändert und damit vor Gericht ziehen.

□

Schlussfolgerungen

- Auch durch Urbild-Angriffe ließen sich im Übrigen Verfahren nicht knacken, die erst digital signieren und dann chiffrieren.
- Wer gar nicht erst an den Klartext herankommt, kann auch keinen Hash-Wert fälschen. Daher bleiben SSH und IPSec sicher.

Vielen Dank für Ihre Aufmerksamkeit!

Haben Sie noch Fragen?

Literaturverzeichnis

[1] BUCHMANN, Johannes A.: *Introduction to Cryptography*. New York : Springer-Verlag, 2001

[2] CORMEN, Thomas H. ; LEISERSON, Charles E. ; RIVEST, Ronald L.: *Introduction to Algorithms*. New York : McGraw-Hill, 1990

[3] ERTEL, Wolfgang: *Angewandte Kryptograhie*. München Wien : Carl Hanser Verlag, 2001

[4] KOBLITZ, Neal: *A Course in Number Theory and Cryptography*. 2nd. New York : Springer-Verlag, 1994

[5] PADBERG, Friedhelm: *Elementare Zahlentheorie*. 2nd. Heidelberg Berlin : Spektrum Akademischer Verlag, 1996

[6] SCHNEIER, Bruce: *Angewandte Kryptographie*. Bonn : Addison Wesley, 1996