Kryptologie
— Einführung und Überblick —

Andreas de Vries

FH Südwestfalen University of Applied Sciences
Haldener Straße 182, D-58095 Hagen, e-mail: de-vries@fh-swf.de

Version: 8. Mai 2005

Inhaltsverzeichnis

Kapitel 1

Grundbegriffe der Kryptologie

In jedem Kommunikationssystem gibt es die Gefahr, dass Unbefugte an für sie nicht bestimmte Informationen gelangen, sie verfälschen oder gar zerstören. Allgemein spricht man in einem solchen Fall von einem *Angriff* auf das Kommunikationssystem. Gerade in Rechnernetzen, die einen direkten oder indirekten Zugang zum Internet haben, in dem potentiell Milliarden von Angreifern existieren, ist diese Gefahr sehr groß. Insbesondere Angriffe durch Viren und Würmer über E-mails oder unsichere Server haben in den letzten Jahren für das Thema der Daten- und Informationssicherheit sensibilisiert.

Daten- und Informationssicherheit hat als Ziel die Gewährung oder Schaffung der folgenden Eigenschaften eines Kommunikationssystems.

- *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.

Die letzte dieser Eigenschaften ist eher ein Problem der technischen Infrastruktur des Kommunikationssystems. In einem Rechnernetz bedeutet sie das Funktionieren des Netzwerks und der Server.

Jede dieser Eigenschaften eines Kommunikationssystems kann Angriffen ausgesetzt sein. Man unterscheidet zwischen *passiven Angriffen*, bei denen Nachrichten oder Informationen abgehört, aber nicht verändert werden, und *aktiven Angriffen*, bei denen Nachrichten oder gar das Kommunikationssystem verändert werden. Angriffe auf die Verfügbarkeit des Systems z.B. sind aktiv; vor allem sind hier *DoS*-Angriffe *(denial-of-service attacks)* zu nennen, die gezielt versuchen, Server zum Absturz zu bringen.

Die Angriffe auf Vertraulichkeit, Integrität, Authentizität und Verbindlichkeit von Nachrichten zu verhindern ist das Ziel der *Kryptographie*. Ihr Ansatz ist die Verschlüsselung von Informationen.

Demgegenüber steht die *Krypt(o)analyse*, deren Ziel das Brechen kryptographischer Verfahren ist, also verschlüsselte Informationen Unbefugten (ohne Kenntnis des Schlüssels) zugänglich zu machen. Kryptographie und Kryptanalyse werden zusammen gefasst unter dem Begriff *Kryptologie*,

$$\text{Kryptologie} = \text{Kryptographie} + \text{Kryptanalyse}.$$

1.1 Kryptosysteme

Ein Grundbegriff der Kryptologie ist das *Kryptosystem* (oder *kryptographisches Kommunikationssystem*). Es besteht aus drei Elementen, dem *Verschlüsselungsalgorithmus* (auch die *Chiffre* genannt), den *Schlüsseln S* und S' des Senders und des Empfängers einer Nachricht, und dem *Chiffrat* oder *Chiffretext (cipher text)*, d.h. der verschlüsselten Nachricht, wie in Abb. 1.1 schematisiert. Der Klartext *(plain text)* ist dabei die unverschlüsselte Nachricht.

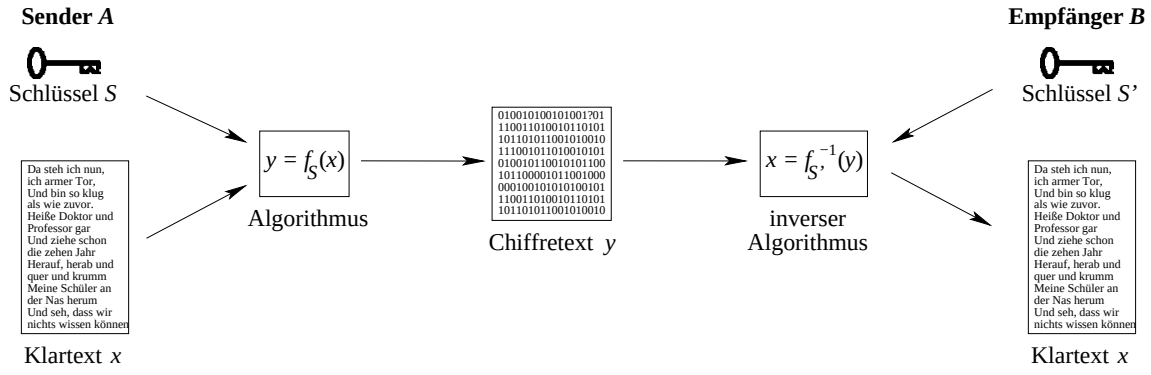


Abbildung 1.1: Ein allgemeines Kryptosystem, bestehend aus Sender- und Empfängerschlüssel S und S', dem Chiffretext y und dem Verschlüsselungsalgorithmus f.

Zentrales Element eines Kryptosystems ist der Verschlüsselungsalgorithmus. Er ist eine umkehrbare mathematische Abbildung f_S, die abhängig von dem Chiffrierschlüssel S einen Klartext in einen Chiffretext transformiert, in Symbolen

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

Hierbei bedeutet $f_{S'}^{-1}$ die Umkehrung der Abbildung f_S, die von dem Dechiffrierschlüssel S' als Parameter abhängt. Beispiele für elementare reelle Funktionen sind $f : \mathbb{R}_+ \rightarrow \mathbb{R}_+$, $f(x) = x^2$, mit der Umkehrung $f^{-1} : \mathbb{R}_+ \rightarrow \mathbb{R}_+$, $f^{-1}(y) = \sqrt{y}$, oder $f : \mathbb{R} \rightarrow \mathbb{R}_+$, $f(x) = \mathrm{e}^x$ mit der Umkehrung $f^{-1} : \mathbb{R}_+ \rightarrow \mathbb{R}$, $f^{-1}(y) = \ln y$.

Der Verschlüsselungsalgorithmus f hat einen Parameter S, den Schlüssel, in Symbolen f_S. Im Allgemeinen haben jeweils Sender und Empfänger einen Schlüssel S und S', die nicht identisch sein müssen. Dieser Parameter bestimmt die konkrete Abbildung, die zur

Verschlüsselung oder Entschlüsselung verwendet wird. Die beiden Schlüssel hängen voneinander ab, zu einem Sende- oder Chiffrierschlüssel S gehört stets ein Empfänger- oder Dechiffrierschlüssel S' und umgekehrt.

1.1.1 Darf ich vorstellen? – Alice und Bob

Zwei Personen haben sich in der Kryptologie etabliert, Alice und Bob. In der Regel ist Alice die Senderin, Bob der Empfänger. Ich werde mich an diese schöne Namensregel halten.

Da im Englischen *eavesdropping* Abhören, Lauschangriff bedeutet, bekommt der ungewollte Eindringling in das Kommunikationssystem oft den Namen Eve.

1.1.2 Kerckhoffs Prinzip

Mit diesem Vorgehen, den Schlüssel als *Parameter* in den Verschlüsselungsalgorithmus eingehen zu lassen, gewinnt man die Tatsache, dass der Algorithmus an sich nicht geheim gehalten werden muss, sondern nur die Schlüssel. Auf Neulinge im Bereich der Kryptologie wirkt das zunächst uneinsichtig, denn warum sollte man den Algorithmus nicht auch geheim halten? Hat es ein Angreifer nicht leichter, einen Chiffretext zu entschlüsseln, wenn er den Algorithmus kennt? Die Erfahrung zeigt interessanterweise das genaue Gegenteil. Die Sicherheit eines Kryptosystems wird erheblich größer, wenn der Algorithmus frühzeitig veröffentlicht wird. Er wird dann nämlich von den besten Kryptologen der Welt attackiert und geprüft, und nur wenn er diesen permanenten „Stresstest“ überlebt, kann er als sicher gelten. Die weltweite Gemeinde der Kryptologen sorgt damit für die bestmögliche Qualitätskontrolle.

Es gibt zahlreiche Beispiele, auch aus der jüngsten Vergangenheit, bei denen sich selbstausgedachte und geheimgehaltene Algorithmen als Flops herausgestellt haben [6, §2.2]. So gibt es beispielsweise ernsthafte Bedenken gegen den Verschlüsselungsstandard WEP bei WLANs, wie unter

http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html

nachzulesen ist.

Die folgende Maxime gilt daher als grundlegend für die moderne „starke“ Kryptologie, seitdem sie von Auguste Kerckhoffs[1] bereits im 19. Jahrhundert formuliert wurde.

Axiom 1.1 ***(Kerckhoffs Prinzip)*** *Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Algorithmus abhängen, sondern nur von der Geheimhaltung seiner Schlüssel.*

Das Kerckhoffs'sche Prinzip wird oft auch als „Grundsatz der Kryptologie“ bezeichnet. Es drückt in exakter Form die Erfahrung aus, die der angesehene Kryptologe Bruce Schneier so formuliert:

[1] Auguste Kerckhoffs (1835–1903), belgischer Kryptologe

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. [12, S. 8]

1.1.3 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.

1.1.4 Vollkommen sichere Kryptosysteme

Ein Kryptosystem heißt *vollkommen sicher*, wenn das Chiffrat y keine Information über den Klartext x liefert. Mathematisch drückt man es genauer aus:

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

d.h. die bedingte Wahrscheinlichkeit $P(x|y)$, den Klartext x bei Kenntnis des Chiffrats y zu erraten, entspricht der Wahrscheinlichkeit $P(x)$, den Klartext überhaupt zu erraten. Die Ereignisse x und y sind also stochastisch unabhängig voneinander.

Bislang ist nur eine Chiffre bekannt, die ein vollkommen sicheres Kryptosystem ermöglicht, die *Vernam-Chiffre*, auch bekannt als *One-Time-Pad* („Einmal-Block“). Für eine gegebene Nachricht x in binärer Form wird eine genauso lange Zufallsfolge S von Nullen und Einsen erzeugt und mit der Nachricht x XOR-verknüpft,

$$y = x \oplus S.$$

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

Diese Chiffre ist vollkommen sicher, wenn der Schlüssel S echt zufällig erzeugt wird, nur Sender und Empfänger bekannt ist und nur einmal verwendet wird. Wenn eine dieser drei Bedingungen nicht erfüllt ist, ist die Chiffre nicht mehr vollkommen sicher. Würden Alice und Bob der Versuchung erliegen, denselben Schlüssel für eine zweite Nachricht zu verwenden, so wird das vollkommen sichere System plötzlich zu einem leicht brechbaren: Sind

nämlich $y^{(1)}$ und $y^{(2)}$ die beiden Chiffretexte und $x^{(1)}$ und $x^{(2)}$ die entsprechenden Klartexte, so gilt wegen

$$y^{(1)} \oplus y^{(2)} = (x^{(1)} \oplus S) \oplus (x^{(2)} \oplus S) = x^{(1)} \oplus x^{(2)} \oplus S \oplus S = x^{(1)} \oplus x^{(2)}$$

denn $S \oplus S = 0$. Das heißt aber, dass ein Angreifer aus der Kenntnis der beiden Chiffretexte durch $y^{(1)} \oplus y^{(2)} = x^{(1)} \oplus x^{(2)}$ schon viel Information über die Klartexte hat!

Die beiden größten Problem des One-Time-Pads sind, einerseits einen echt zufälligen Schlüssel zu erzeugen, andererseits diesen zufälligen Schlüssel über einen sicheren Kanal auszutauschen, bevor er benutzt werden kann. Mit dem zweiten Problem entsteht damit das Dilemma der klassischen Kryptographie:

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

Im Kern ist also das erste Hauptproblem der Kryptographie (S. 7) nicht gelöst. Damit hat das One-Time-Pad als bislang einzig bekannte vollkommen sichere Chiffre vorwiegend akademischen Wert, praktisch ist es nicht realisierbar. (Und selbst wenn es gelänge, einen Schlüssel über einen sicheren Kanal zu übermitteln, so könnte man doch gleich den Klartext senden...)

Seit jeher spielen denn auch nicht vollkommen sichere Systeme die Hauptrolle in der Kryptologie, sondern *praktisch sichere* Kryptosysteme, die mit dem jeweils aktuellen Stand der Technik in n Jahren gebrochen werden, wobei die Größe von n von den Sicherheitsbedürfnissen der Kommunikationsteilnehmer abhängt. Ein Beispiel für ein praktisch sicheres Verfahren sei die Cäsar-Verschlüsselung, die das römische Heer während der Gallischen Kriege anwandte. Für die damaligen Feinde Cäsars war diese Chiffre schwer zu brechen, für heutige Standards ist es ein triviales Verfahren. Ein modernes Beispiel ist DES, der ein Vierteljahrhundert der weltweite Verschlüsselungsstandard war, aber Ende der 1990er Jahre von einem Spezialrechner in 4,5 Tagen gebrochen werden konnte. Er gilt heute nicht mehr als sicher.

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[2] ASCII oder Unicode. Diese Zahlen werden durch den Verschlüsselungsalgorithmus mathematisch verändert und ergeben damit eine andere Zahlenfolge, die das Chiffrat darstellt. Wir werden im Folgenden unter Klartext die codierte Zahlenfolge verstehen, und entsprechend mit Chiffrat die chiffrierte Zahlenfolge.

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.

[2]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“.

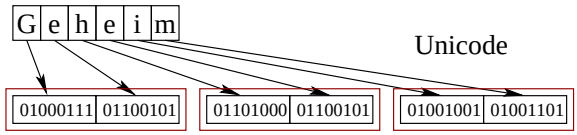


Abbildung 1.2: Prinzipielles Vorgehen beim Blockchiffre. Ein zu verschlüsselnder Text wird codiert, z.B. mit Unicode à 2 Byte je Zeichen (es sind die führenden 8 Nullen weggelassen); daraus werden 2-Byte-Blöcke gebildet.

1.3 Kryptanalyse: Wann ist eine Chiffre geknackt?

Das Brechen oder Knacken eines Kryptosystems besteht darin, bei Unkenntnis des geheimen Schlüssels aus einem Chiffrat den Klartext zu entschlüsseln oder gar den geheimen Schlüssel zu entdecken. Eine Methode, alle kombinatorisch möglichen Verschlüsselungen durchzuprobieren, heißt *Brute-Force-Attacke*.[3] Sie ist eine wichtige kryptanalytische Methode. Fast alle Kryptosysteme lassen sich mit Brute-Force brechen, aber bei genügend sicheren Schlüsseln beträgt der Aufwand hunderttausende oder Millionen von Jahren.

Es gibt eine ganze Reihe kryptanalytischer *Angriffe* auf ein Kryptosystem, die wichtigsten sind:

- *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. Dieser Angriff spielt vor allem für Kryptosysteme mit öffentlichen Schlüsseln eine Rolle, denn mit ihnen kann jeder beliebige Klartext verschlüsselt werden.
- *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*.

1.3.1 Differentielle Kryptanalyse

Die *differentiellen Kryptanalyse* ist eine statistische Methode zur Erlangung des Geheimschlüssels. Sie ist üblicherweise ein Chosen-Plaintext-Angriff, es gibt aber auch Erweiterungen als Known-Plaintext- und sogar Ciphertext-Only-Angriffe. Es werden bei dieser Methode Paare von Klartexten verwendet, die sich jeweils durch eine irgend definierte konstante

[3] *brute force* - Brachialgewalt

Differenz unterscheiden, die in der Regel durch die XOR-Operation definiert ist (also die unterschiedlichen Bits zählt), 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 Verschlüsselungsverfahren

Bei *symmetrischen Verschlüsselungsalgorithmen* wird ein einziger geheimer Schlüssel zur Ver- und Entschlüsselung verwendet. In dem Kryptosystem in Abb. 1.1 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*. Ein Beispiel ist die Cäsar-Verschlüsselung, eines der historisch ersten Verfahren, das das Römische Militär[1] einsetzte.

2.1 Cäsar-Verschiebung

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

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

Die Variable n bezeichnet dabei die Stelle im Alphabet des zu verschlüsselnden Buchstabens x (man schreibt auch $n = \langle x \rangle$):

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 das Chiffrat):

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

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

[1] Wir verwenden im Folgenden unser heutiges Alphabet mit 26 Buchstaben, nicht das Lateinische, das zu Cäsars Zeiten 25 Buchstaben enthielt (mit U=V).

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 statt einer einfachen Alphabetverschiebung 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, und das schafft keiner mit Papier und Bleistift zu seinen Lebzeiten. Solche Verfahren nennt man „Monoalphabetische Verschlüsselungen“. Übrigens: 26! entspricht etwa 89 Bit.

2.2 DES

DES *Data Encryption Standard* war der symmetrische Verschlüsselungsstandard der NIST von Juni 1977 bis Februar 2001. Es ist 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.

Der Algorithmus ist im Überblick in Abbildung 2.1 dargestellt. Zunächst wird der 64 bit lange Schlüssel K durch eine Schlüsselpermutation auf 56 bit verkürzt. Der Klartextblock wird zuerst durch die Eingangspermutation gemischt, und in zwei 32-bit-Blöcke L_0 und R_0 geteilt, durchläuft dann 16 Runden und erzeugt nach der Schlusspermutation, die genau invers zur Eingangspermutation ist, den Chiffretextblock.

In Runde i wird der 56 bit lange Schlüssel K durch eine Schlüsselpermutation in zwei gleich lange Hälften à 28 bit geteilt, die beide bitweise verschoben und durch die Kompressionspermutation auf 48 bit verkürzt werden. Der jeweils rechte Teilblock R_{i-1} wird unverändert als linker Teilblock L_i der nächsten Runde übernommen, während eine Funktion $f(R_{i-1}, K_i)$ mit den Parametern R_{i_1} und dem Teilschlüssel K_i ausgeführt wird, das Ergebnis mit L_{i-1} XOR-verknüpft und als rechter Teilblock R_i der nächsten Runde verwendet wird, also kurz:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i). \end{aligned} \tag{2.2}$$

Der Chiffretext wird entschlüsselt, indem derselbe Schlüssel verwendet wird und die Operationen in umgekehrter Reihenfolge ausgeführt werden.

2.2.1 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 transformie-

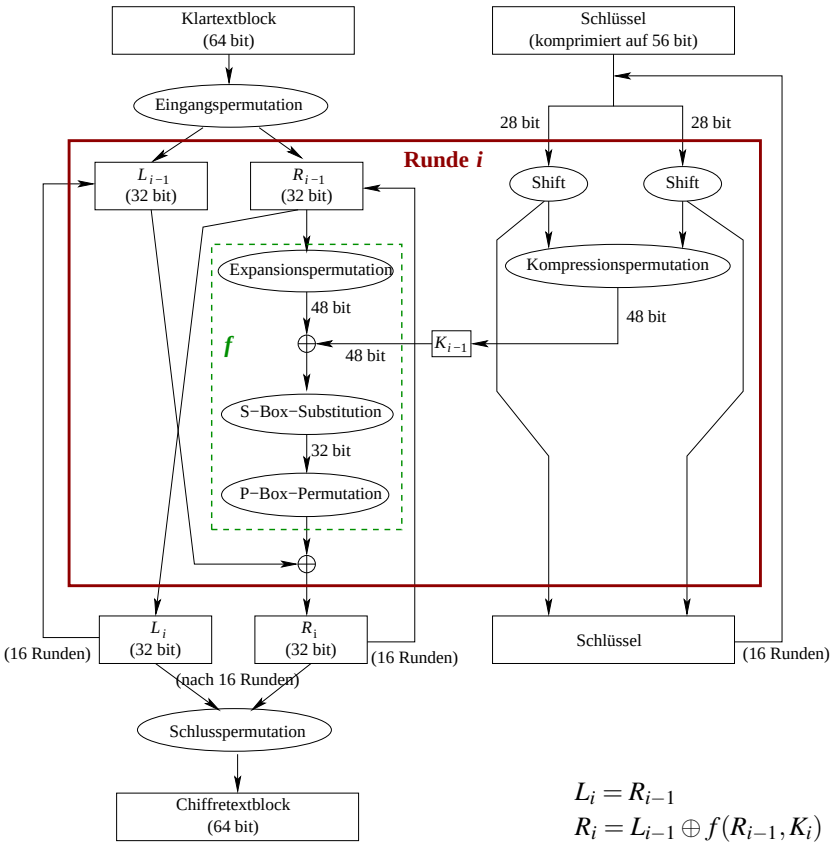


Abbildung 2.1: Ablaufdiagramm von DES, mit den Rundenschlüsseln K_i ($i = 1, 2, \ldots, 16$.

ren und eine Ausgabe von je 4 bit liefern [6, §4.1]. Die S-Box-Transformation ist die einzige nichtlineare Transformation des DES.

Informationstheoretisch betrachtet erhält DES 120 bit Information (64 bit des Klartextblocks und 56 bit des Schlüssels), gibt aber nur 64 bit aus. Effektiv werden also netto 56 bit an Information verborgen oder Entropie erzeugt.

2.2.2 Sicherheit von DES

DES wurde mehrfach Ende der 1990er Jahre durch Brute-Force gebrochen. Einer der schnellsten Angriffe wurde am 19.4.1999 mit einem Netzwerk von 100 000 Rechnern und einem Spezialrechner der *Electronic Frontier Foundation* durchgeführt, bei dem ein 88 Byte langer vorgegebener Chiffretext in 22 Stunden dechiffriert wurde.

DES war also am Ende des letzten Jahrtausends als Verschlüsselungsstandard nicht mehr haltbar. Ist der Algorithmus also schlecht? Seit DES Standard ist, hat es die Diskussion um die Länge des Schlüssels gegeben, denn es gibt keinen wirklichen Grund für die Komprimierung von 64 auf 56 bit — ganz im Gegenteil, der DES zugrunde liegende Algorithmus namens LUCIFER von Horst Feistel[2] verwendete einen 128-bit-Schlüssel, der auch heute noch unbrechbar wäre.

[2]Horst Feistel (1915–1990), deutsch-amerikanischer Kryptologe

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? Interessanterweise gibt es 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 — die Schlüssellänge ist also effektiv nicht 112 bit, sondern nur 57 bit! (Siehe dazu [6, §4.1.6] und [12, §15.1].)

2.4 AES

AES *(Advanced Encryption Standard)* ist seit Februar 2001 der offizielle symmetrische Verschlüsselungsalgorithmus der NIST (FIPS 197). Er ist Nachfolger von DES *(Data Encryption Standard)*, der seit 1977 der Standard war. AES ging durch ein öffentliches Ausschreibungsverfahren von 1997 aus dem Algorithmus Rijndael der belgischen Kryptologen Vincent Rijmen und Joan Daemen hervor.

AES ist eine Blockchiffre, deren Blocklänge und Schlüssellänge unabhängig voneinander auf einen der Werte 16, 24 oder 32 Byte (128/192/256 Bit) gesetzt werden können.[3] 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 mehrere Runden, deren Anzahl r zwischen 10 und 14 liegt, abhängig von Block- und Schlüssellänge:

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

Insgesamt werden $r+1$ Rundenschlüssel $K_0, K_1, \ldots, K_r$, jeweils der Blocklänge $4 \cdot n_B$ Byte, mit Hilfe einer Prozedur namens *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. Nach jeder Runde i wird dieser Zustand mit einem Rundenschlüssel K_i XOR-verknüpft (Symbol: $\oplus$). Jeder Zustand S_i, jeder expandierte Rundenschlüssel K_i und der Ursprungsschlüssel K hat jeweils die Form einer Matrix,

$$S_i = \begin{pmatrix} s_{0,0} & \cdots & s_{0,n_B-1} \\ \vdots & \ddots & \vdots \\ s_{3,0} & \cdots & s_{3,n_B-1} \end{pmatrix}, \quad K_i = \begin{pmatrix} k_{0,0} & \cdots & k_{0,n_B-1} \\ \vdots & \ddots & \vdots \\ k_{3,0} & \cdots & k_{3,n_B-1} \end{pmatrix}, \quad K = \begin{pmatrix} K_{0,0} & \cdots & K_{0,n_K-1} \\ \vdots & \ddots & \vdots \\ K_{3,0} & \cdots & K_{3,n_K-1} \end{pmatrix}. \tag{2.3}$$

[3] Prinzipiell sind alle durch 4 teilbaren Bytewerte von 16 bis 32 möglich.

Jede Spalte der Zustands- sowie der Schlüsselmatrizen entspricht einem 4-Byte-Wort. Der Startzustand S_{-1} ist der Klartextblock der Länge $4 \cdot n_B$ Bytes, der Endzustand S_{r+1} ist der Chiffretextblock. Innerhalb jeder Runde i wird jedes einzelne Byte des Zustands S_i nacheinander von den Teilalgorithmen `SubBytes`, `ShiftRow` und `MixColumn` verändert. Bezeichnet `s[i]` den Zustand S_i und `k[i]` den Rundenschlüssel K_i (d.h., `k` ist ein Array aus n_K-Bytes der Länge $r+1$), so lautet der Pseudocode des AES-Algorithmus:

```
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. Das entsprechende Ablaufdiagramm ist in Abb. 2.2 dargestellt. 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“), die Transformation SubBytes ist eine kompliziertere („nichtlineare“) Transformation. Sie ist gemäß der Wertetabelle in Abb. 2.3 gegeben. Alle Operationen sind eineindeutig umkehrbar. So ist die Umkehrabbildung der SubBytes-Transformation eindeutig durch die Wertetabelle in Abb. 2.3 (rechts) dargestellt.

Informationstheoretisch betrachtet erzeugt AES insgesamt bei Eingabe von $4n_B$ Byte Klartextblock und $4n_K$ Byte Schlüssel bei Ausgabe von $4n_B$ Byte Chiffretext effektiv eine Entropie („Unwissenheit“) von $4n_K$ Byte, also bei geringstmöglicher Schlüssellänge ($n_k = 4$) von 16 Byte $=$ 128 bit.

2.4.1 Bewertung von AES

AES hat zwei herausragende Eigenschaften, die ihn gegenüber den meisten gängigen symmetrischen Algorithmen auszeichnen, hohe Geschwindigkeit und effiziente Hardware-Implementierbarkeit. 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 [6, S. 63, 69].

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.

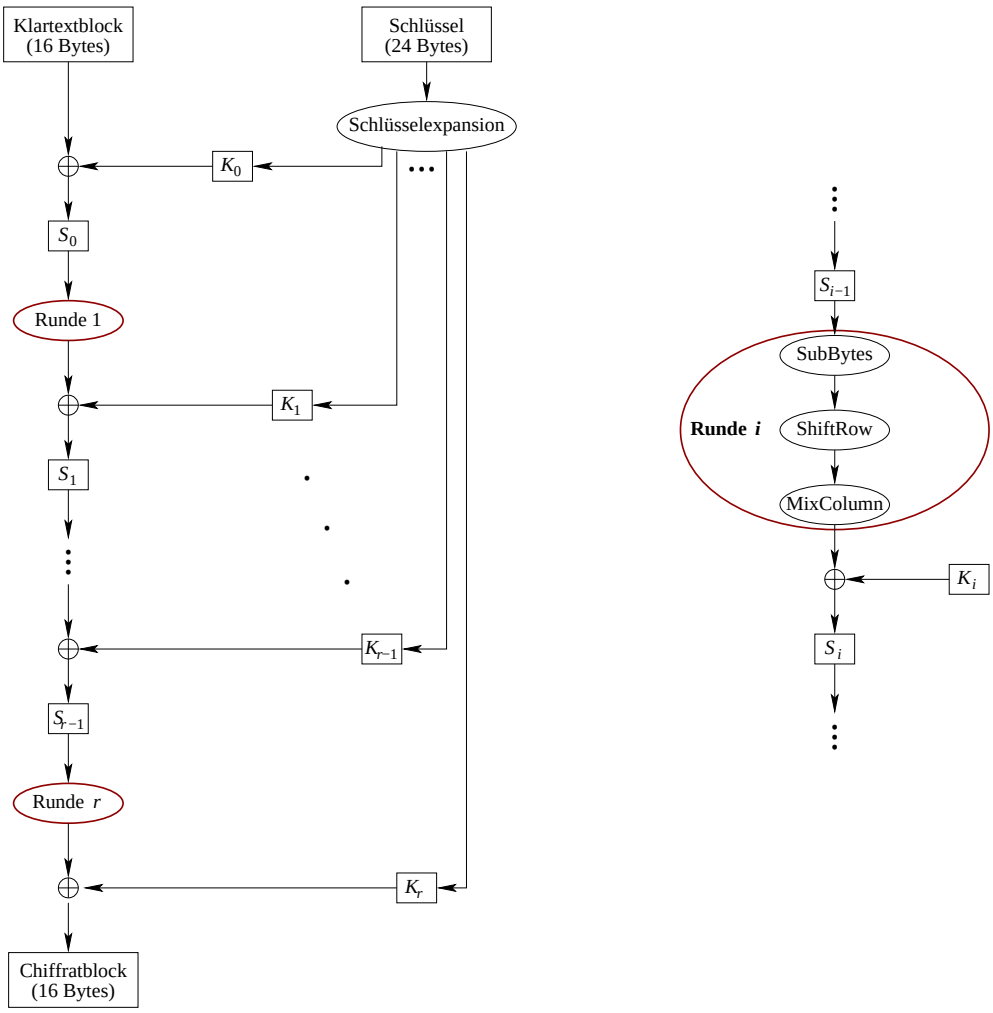


Abbildung 2.2: Ablaufdiagramm von AES für Blocklänge 16 Bytes und Schlüssellänge 24 Bytes, mit den Rundenschlüsseln K_i und den Zuständen S_i. Rechts der detailliertere Ablauf einer einzelnen Runde i.

Die Sicherheit von AES wird als hoch angesehen. Er ist resistent gegen übliche kryptanalytische Angriffe. Es ist kein erfolgreicher effizienter 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/*

2.5 Checkliste der Kryptoprobleme

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

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

Abbildung 2.3: Links: Die S-Box von AES, rechts die inverse S-Box. Es sind die Wertetabellen für das Byte `xy` (in Hexadezimaldarstellung) angegeben.

Aufgaben

Übung 2.5.1 *(S-Box von AES)* Die SubBytes-Transformation des AES, auch "S-Box" genannt, ist eine nichtlineare Transformation, die man durch die Wertetabelle in Abb. 2.3 darstellen kann. Entsprechend ergibt sich die inverse S-Box aus der Wertetabelle in Abb. 2.3.

(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 sowohl einen eigenen öffentlichen Schlüssel *(public key)* als auch einen privaten Schlüssel *(private key)* hat; letzterer wird stets geheim gehalten. In Abb. 1.1 auf S. 5 ist also $S \neq S'$, Ver- und Entschlüsselung einer Nachricht funktionieren mit jeweils einem eigenen Schlüssel. Daher die Bezeichnung „unsymmetrisch".

In dem verbreitetsten unsymmetrischen Verfahren, RSA, besteht jeder Schlüssel aus einer Gruppe von natürlichen Zahlen. Wie in der Kryptologie üblich, heißen Sender und Empfänger einer verschlüsselten Nachricht Alice und Bob. Wir werden Alices öffentlichen Schlüssel mit P_A und Bobs mit P_B bezeichnen, während Alices privater Schlüssel mit S_A und Bobs mit S_B (für *secret*) dargestellt wird. 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 beiden folgenden Bedingungen erfüllt sind:

1. $y = f(x)$ ist leicht berechenbar, also in polynomialer Rechenzeit bezüglich der Länge von x.

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

Bleibt die Umkehrfunktion f^{-1} einer Falltürfunktion auch mit zusätzlicher Information schwer berechenbar, so heißt sie *Einwegfunktion (one-way function)* [8, S. 85]. □

Im Unterschied zu Falltürfunktionen gibt es für eine Einwegfunktion also keinen Schlüssel, der die Umkehrung leicht berechenbar macht.

Die Definitionen sind nicht mathematisch exakt, denn die Konzepte „leicht berechenbar“ und „schwer berechenbar“ sind empirische Begriffe und hängen ab von dem aktuellen Stand der Rechnertechnologie (Rechenschieber, Quantenrechner) und neuen Algorithmen. Bislang weiß niemand, ob es Falltürfunktionen oder Einwegfunktionen überhaupt gibt, oder ob unser Wissen nur nicht ausreicht und doch irgendwann eine effiziente Umkehrung berechnet werden kann. Was 2005 eine Falltürfunktion war, wird es vielleicht 2015 oder 2105 nicht mehr sein!

Beispiele 3.2 (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 modularer oder 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

RSA ist benannt nach ihren Erfindern Ron Rivest, Adi Shamir und Len Adleman (1978), zwei Informatikern und einem Mathematiker. RSA war das erste unsymmetrische Verschlüsselungsverfahren und ist immer noch das wichtigste. Es basiert im Kern auf zwei Falltürfunktionen, der Multiplikation großer Zahlen und der modularen Exponenzierung, deren Umkehrungen, also die Faktorisierung großer Zahlen und der diskrete Logarithmus, nur schwer berechenbar sind. Anders ausgedrückt: Es ist einerseits sehr leicht, große Primzahlen zu finden und zu multiplizieren oder Potenzen modular zu berechnen, aber sehr schwer, große Zahlen in ihre Primfaktoren zu zerlegen oder die modulare Exponenzierung zu invertieren.

Um das RSA-Verfahren zu verstehen, benötigen wir noch eine für manche ungewohnte Notation für „modulo“, die in der Mathematik üblich ist und auf Gauß zurückgeht. Rechnet man mit natürlichen Zahlen m und n, so besagt eine Gleichung der Art

$$k = m \bmod n,$$

dass $k \,\%\, n = m \,\%\, n$ ist; man spart sich also das Schreiben der modulo-Division % auf beiden Seiten einer Gleichung, indem man einmal „mod“ *hinter* die Gleichung oder eine Gleichungskette schreibt.

In einem *RSA Kryptosystem* erzeugt jeder Teilnehmer seinen privaten und seinen öffentlichen Schlüssel durch den folgenden Algorithmus.

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 der Verschlüsselungsexponent *(encryption exponent)*, d der Entschlüsselungsexponent *(decryption exponent)*, und n der RSA-Modul *(RSA modulus)*. Die Veröffentlichung kann über ein öffentlich zugängliches Schlüsselverzeichnis geschehen, ähnlich einem Telefonbuch, oder aber über eine Datei auf einem Server.

Dieser Algorithmus der Schlüsselerzeugung ist jedoch nur die Vorarbeit, die jeder Kommunikationsteilnehmer für sich vornehmen muss. Wie wird nun ver- und entschlüsselt? 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 eigentliche RSA-Verschlüsselungsalgorithmus ist die Funktion $E : \mathbb{Z}_n \to \mathbb{Z}_n$,

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

Die Verschlüsselung einer Nachricht m geschieht einfach 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 $D : \mathbb{Z}_n \to \mathbb{Z}_n$,

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

Das Verfahren, wie Alice eine verschlüsselte Nachricht an Bob schickt, ist schematisch in Abb. 3.1 dargestellt. Natürlich kann das Verfahren nur funktionieren, wenn die Einträge in

[1] Das kgV berechnet man effizient, indem man den ggT verwendet: $\mathrm{kgV}(a,b) = \frac{ab}{\mathrm{ggT}(a,b)}$ 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...

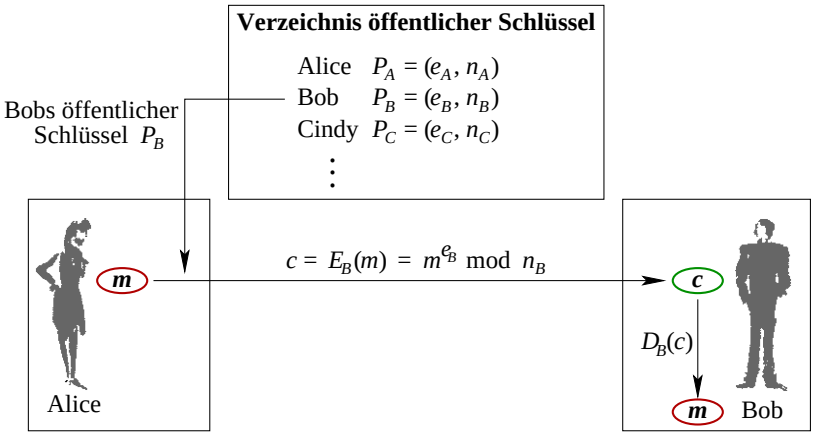


Abbildung 3.1: Alice sendet eine verschlüsselt Nachricht m an Bob, indem sie *seinen* öffentlichen RSA-Schlüssel P_B verwendet.

dem öffentlichen Verzeichnis „stimmen“, d.h. wenn öffentlicher und privater Schlüssel zueinander gehören.

Beispiel 3.3 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 sich gegenseitig Umkehrfunktionen. □

Beispiel 3.4 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:

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

□

3.2.1 Die Korrektheit von RSA

Die Korrektheit von RSA, nämlich die nicht sofort offensichtliche Tatsache, dass E und D tatsächlich einander inverse Funktionen sind, liegt an der einfachen Beziehung

$$m^{ed} = m \bmod n \qquad \text{for } m \in \mathbb{Z}_n. \tag{3.6}$$

Sie folgt aus dem „Korollar von Carmichael“, siehe [13]. Die beiden Beispiele illustrieren, wie man mit RSA die beiden Schlüssel berechnet. In realen RSA-Systemen sind die Zahlen natürlich bedeutend größer.

Fußnote: Oft findet man in der Literatur die Definition von RSA-Kryptosystemen basierend auf der so genannten Euler-Funktion φ statt auf der Carmichael-Funktion λ, cf. [4]. Da aber für zwei Primzahlen p und q gilt: $\varphi(pq) = (p-1)(q-1)$, haben die beiden Funktionswerte $\varphi(pq)$ und $\lambda(pq)$ dieselben Teiler. Daher ist ein zu $\lambda(pq)$ teilerfremder Schlüsselparameter d ebenso teilerfremd zu φ, und umgekehrt. Nur der öffentliche Schlüsselparameter e berechnet sich unterschiedlich. Um es genauer zu sagen: Jeder mit Hilfe der Euler-Funktion berechnete öffentliche Schlüssel ist auch ein öffentlicher Schlüssel gemäß der Carmichael-Funktion; die Umkehrung jedoch gilt im Allgemeinen nicht. (Beweis: Da $\lambda(n)|\varphi(n)$, impliziert die Gleichung $ed = 1 \bmod \varphi(n)$, dass $ed = 1 \bmod \lambda(n)$.) Verwendet man die Euler-Funktion, so beweist man die Korrektheit des RSA-Verfahrens mit dem „Satz von Euler“ an Stelle des Korollars von Carmichael [13].

3.2.2 Sicherheit von RSA

RSA basiert auf der Schwierigkeit, die Umkehrfunktion von E zu finden, obwohl man E kennt. Es sind eigentlich zwei Falltürfunktionen, auf denen die Sicherheit von RSA basiert, die Faktorisierung großer Zahlen und der Diskrete Logarithmus.

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 beiden Primfaktoren p und q zu zerlegen, so kann er die Carmichael-Zahl λ bestimmen und d berechnen, für das ja gilt $ed \ \% \ \lambda(n) = 1$ — damit hat er Alices Geheimschlüssel „geknackt“, er kann nun jede Nachricht an Alice lesen.

Allerdings ist genau die Schwierigkeit, große Zahlen zu Faktorisieren. Das Problem ist uralt, schon die griechischen Mathematiker haben nach schnellen Algorithmen gesucht. Das

schnellste zur Zeit bekannte Verfahren ist das „Zahlkörper-Sieb“ *(number field sieve)* von Pollard aus dem Jahr 1988. Es hat für eine Zahl n eine Laufzeit von [3, §8.5]

$$T_{\mathrm{f}}(\ell) \;=\; O\big(\mathrm{e}^{(\frac{64}{9}\sqrt{\ell}\,\cdot\,\ln\ell)^{2/3}}\big) \qquad \text{mit } \ell = \ln n. \tag{3.7}$$

Hier bezeichnet ℓ die Länge der Zahl, ihre Länge $l_2(n)$ in bit hängt mit ihr zusammen über $l_2(n) = \lceil \log_2 n \rceil = \lceil \ell / \ln 2 \rceil$. Die benötigten Rechenzeiten für verschiedene Zahlen n auf einem 10 GHz-Rechner sind in Tabelle 3.1 angegeben. Zum Vergleich benötigt das sukzessive

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 zur Primfaktorisierung verschiedener Zahlen n auf einem 10-GHz-Rechner (mit *Quadratic Sieve*, $T(n) = O(n^{\sqrt{\ln\ln n / \ln n}})$) [11, S. 165].

Teilen von n durch alle natürlichen Zahlen $\leqq \sqrt{n}$ eine Laufzeit $O(n^{1/2}) = O(\mathrm{e}^{\ell/2})$, ist also *erheblich* langsamer.

Bezogen auf die Länge $\ell = \ln n$ der Zahl n ist die Laufzeit des Zahlkörpersiebs (3.7) also nicht mehr polynomial, jedoch besser als exponentiell. Man nennt solche Laufzeiten daher *subexponentiell* [3, §8.4].

Die Firma RSA Laboratories schreibt Faktorisierungswettbewerbe aus, zu finden unter dem URL

www.rsasecurity.com/rsalabs/challenges/

Die letzte faktorisierte Zahl war 1999 eine 512-Bit-Zahl (155 Dezimalstellen); die Rechenzeit betrug 35,7 CPU-Jahre, parallel ausgeführt von 292 Computer mit etwa 400 MHz Taktfrequenz.

Trägt man die Anzahl der Dezimalstellen der größten faktorisierten Zahl über die Jahre seit 1970 auf, so entdeckt man empirisch einen linearen Zusammenhang. Extrapoliert man diese Beziehung, so wird ein 1024-Bit-RSA-Modul (308 Dezimalstellen) etwa 2037 faktorisiert sein. Heute kann es also als sicher betrachtet werden.

Diskreter Logarithmus

Wäre andererseits die diskrete 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 $ed =$ “$\log_{x \bmod n} 1$”, und damit schnell den geheimen Schlüssel d erlangen.

3.3 Elliptische Kurven

Die Lösungsmenge einer Gleichung der Form

$$y^2 = x^3 + ax + b \tag{3.8}$$

mit gegebenen Konstanten a, b und zwei Unbekannten x und y wird *elliptische Kurve* genannt (Abb. 3.2).[3] Jede Gerade, die nicht parallel zur y-Achse ist, hat entweder einen oder drei

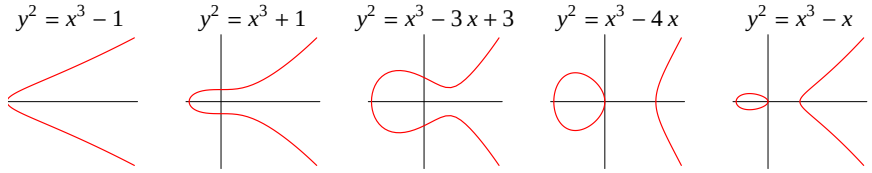


Abbildung 3.2: Verschiedene elliptische Kurven (waagerecht: x-Achse, senkrecht: y-Achse)

Schnittpunkte mit der Kurve (wenn man Tangentenpunkte doppelt zählt).[4]

Eine verblüffende Eigenschaft von elliptischer Kurven ist, dass man mit ihnen „geometrisch rechnen“ kann. 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. 3.3). Auf diese Weise *definiert* man eine Addition auf

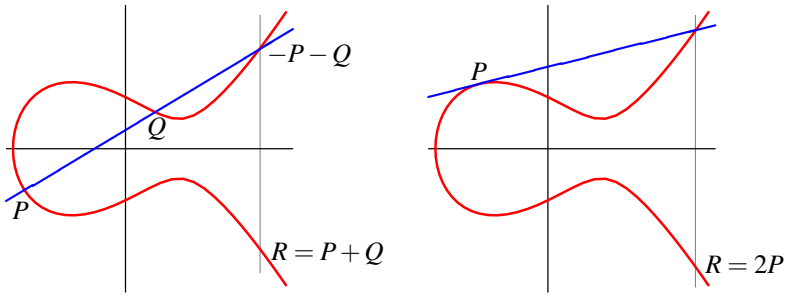


Abbildung 3.3: Addition zweier Punkte auf der elliptischen Kurve $y^2 = x^3 - 3x + 3$

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

[3] Allgemein müsste man eine elliptische Kurve definieren durch

$$y^2 + b_1xy + b_0y = x^3 + a_2x^2 + a_1x + a_0$$

mit gegebenen Konstanten a_0, a_1, a_2, b_0, b_1 und zwei Unbekannten x und y. Die Gleichung heißt *Weierstraß-Gleichung*. Erlaubt man als Werte für diese Größen reelle Zahlen, so kann sie (durch „Koordinatentransformationen“, also Drehungen und Streckungen der x- und y-Achse, ohne dass die Kurve verändert wird) auf die einfachere Form (3.8) bringen. Rechnet man jedoch modulo 2^k mit $k \in \mathbb{N}$, so ist eine elliptische Kurve von einem der beiden Typen $y^2 + cy = x^3 + ax + b$ oder $y^2 + xy = x^3 + ax^2 + b$, und modulo 3^k entsprechend $y^2 = x^3 + ax^2 + bx + c$. [8, S. 168].

[4] Das liegt daran, dass ein Polynom dritten Gerades entweder eine oder drei Nullstellen besitzt (wenn man Nullstellen wie $x^3 = 0$ oder $x^2 = 0$ – wie für das Polynom x^3 oder $x^3 - x^2$ – mit ihrer „Vielfachheit“ zählt, also dreifach bzw. doppelt.

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. Die so definierte Addition funktioniert konsistent, wie Jacobi[5] bereits 1834 bewiesen hat [15, §3.8.1.3]. Bei einer gegebenen elliptischen Kurve (3.8) kann man explizite Formeln angeben für $2P$ und $P + Q$ [8, S. 170]: wenn $P = (p_x, p_y)$, so gilt für $R = 2P$, mit $R = (r_x, r_y)$

$$r_x = \left(\frac{3p_x^2 + a}{2p_y}\right)^2 - 2p_x, \qquad r_y = -p_y + \left(\frac{3p_x^2 + a}{2p_y}\right)(p_x - r_x); \tag{3.9}$$

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

$$s_x = \left(\frac{q_y - p_y}{q_x - p_x}\right)^2 - p_x - q_x, \qquad s_y = -p_y + \left(\frac{q_y - p_y}{q_x - p_x}\right)(p_x - s_x). \tag{3.10}$$

Kryptologisch wichtig sind nun ganzzahlige Werte für x und y, die auf einer gegebenen elliptischen Kurve liegen. Da solche ganzzahligen Werte durch ein Gitter ($\mathbb{Z} \times \mathbb{Z}$) gegeben

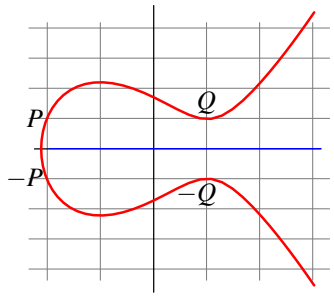


Abbildung 3.4: Ganzzahliges Gitter auf der elliptischen Kurve $y^2 = x^3 - 3x + 3$. In dem Bildausschnitt schneidet das Gitter die Kurve nur in $P = (-2, 1)$, $-P = (-2, -1)$, $Q = (1, 1)$ und $-Q = (1, -1)$. Die nächsten Punkte sind erst $\pm R = (16289, \pm 35327)$. Es gilt übrigens $P + Q = -Q$ und $2Q = -P$.

sind, betrachtet man also nur dessen Schnittpunkte mit der Kurve (Abb. 3.4). 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 [8, S. 174ff]. In den Formeln (3.9) und (3.10) 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 3.5 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 mit der „Kurvenaddition“:

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

Insbesondere ist $4P = P$, man sagt P hat die *Ordnung* 4. □

[5] Carl Jacobi (1804–1851), deutscher Mathematiker

3.3.1 Sicherheit von Verfahren mit elliptischen Kurven

In der Kryptologie werden natürlich viel größere Zahlen für a, b und q verwendet. Neben diesen Parametern wird ein Punkt P auf der elliptischen Kurve festgelegt. Mit Hilfe der Formeln (3.9) und (3.10) kann man nun effizient den Punkt nP für ein sehr große $n < q$ berechnen. Wenn man nämlich entsprechend der binären Darstellung von n nur die 2-er-Potenzen P, $2P$, 2^2P, 2^3P, ..., addiert, deren Faktor für n nicht 0 ist, so benötigt man höchstens $\log_2 n$ Additionen. Beispielsweise wird $n = 13$ binär durch $n = 1101_2$ dargestellt, d.h.,

$$13P = P + 4P + 8P.$$

Umgekehrt gibt es offenbar keine Regelmäßigkeit der Punkte nP bei variierendem n. 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 Additionen.

Damit ist also die Multiplikation auf elliptischen Kurven eine Falltürfunktion. Da Additionen und Multiplikationen von Zahlen $\leqq n$ stets Laufzeiten von $O(\ln n)$ beanspruchen [3, §1.5], ergibt sich für die Berechnung von $\mathfrak{m}(n) = nP$ eine Laufzeit

$$T_{\mathfrak{m}}(n) = O(\ln n), \qquad \text{während} \qquad T_{\mathfrak{m}^{-1}}(n) = O(n). \tag{3.11}$$

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 Sekunde) den Punkt nP zu bestimmen. Das inverse Problem, n aus $Q = nP$ und P zu erhalten, ist jedoch nicht in akzeptabler Zeit möglich. Dies wird als das „Diskrete Logarithmus Problem über Elliptischen Kurven“ bezeichnet, kurz ECDL-Problem *(Elliptic Curve Discrete Logarithm Problem)*. Die Falltürfunktion bei Verfahren mit elliptischen Kurven ist also die ganzzahlige Multiplikation nP für große n.

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). Der Grund dafür ist, dass die schnellsten Algorithmen zur Lösung des ECDL-Problems Problems eine exponentielle Laufzeit haben, im Gegensatz zu den subexponentiellen Laufzeiten der zur Zeit besten Faktorisierungsalgorithmen (Zahlkörpersieb, Quadratisches Sieb oder Faktorisieren mit Elliptischen Kurven). Dies erklärt, warum die Parameter von Kryptoverfahren, die auf dem Problem Faktorisieren ganzer Zahlen beruhen, größer sind als die Parameter von Kryptoverfahren, die auf dem ECDL-Problem basieren.

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

Asymmetrische Kryptosysteme ermöglichen erstmals in der Kryptologie Authentifizierung, und zwar über den öffentlichen Schlüssel. 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.

3.5 Checkliste der Kryptoprobleme

Unsymmetrische Verschlüsselungsverfahren haben erstmals das Problem der Authentifizierung gelöst. Abhängig von der Schlüsselverwaltung lösen sie sogar die Problematik des Schlüsselaustauschs: 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 für unsymmetrische Chiffren	
Schlüsselaustausch gelöst?	je nachdem
Authentifizierung?	ja
Abhörerkennung?	nein

3.6 Vergleich mit symmetrischen Chiffren

3.6.1 Sicherheit

Unsymmetrische Verschlüsselungsverfahren zeichnen sich dadurch aus, dass sie eine mathematische Transformation verwenden, die leicht („effizient“) durchzuführen ist, deren Umkehrung aber nur schwer gelingt. Bei RSA ist dies die Faktorisierung großer Zahlen, bei ElGamal oder Diffie-Hellman [6] ist es der „diskrete Logarithmus“.

Um die Umkehrung praktisch unmöglich zu machen, müssen die verwendeten Zahlen, also der oder die Schlüssel, sehr groß sein. Bei RSA ist dies z.B. der RSA-Modul, der aus derzeitiger Sicht größer als 1 kbit sein sollte um als sicher zu gelten. Nun ist jedoch offensichtlich, dass nicht alle Zahlen $\leqq$ 1 kbit mögliche RSA-Schlüssel sein können, denn die Zahlen mit zwei Primfaktoren sind „dünn“ gesät. Das weiß natürlich auch ein (kompetenter) Angreifer, d.h. der tatsächliche Schlüsselraum ist viel kleiner als der benötigte Speicherplatz; Ähnliches gilt für andere unsymmetrische Verfahren. Im Gegensatz dazu kann ein symmetrisches Verfahren den gesamten Schlüsselraum ausschöpfen (obwohl es auch einige wenige „schwache“ Schlüssel gibt). Das führt zu der Tabelle 3.2, in der die benötigten Schlüssellängen bei jeweils gleicher Sicherheit gegenüber gestellt sind. Man erkennt, dass die Länge eines unsymmetrischen Schlüssels bei gleicher Sicherheit etwa 15- bis 20-mal so groß ist wie die eines symmetrischen, wobei die Schere bei steigender Sicherheit immer weiter auseinander klafft.

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.

Zu beachten ist außerdem, dass bei den gängigen Protokollen wie TSL (SSL) 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.

3.6.2 Schlüsselaustausch

Eine wesentliche Schwachstelle aller symmetrischen Chiffren ist der Austausch bzw. die Vereinbarung des Schlüssels. Dazu benötigen die Kommunikationspartner einen sicheren Kanal, z.B. einen Kurier — ein schwerer, oft kaum zu realisierender Umstand.

3.6.3 Performanz

Software-Implementierungen von RSA sind derzeit um mehr als den Faktor 1000 langsamer als AES (vgl. [6, §5.2.4] und S. 15). Die Ursache ist der erhebliche Rechenaufwand (modulare Exponentiation) der Ver- und Entschlüsselung in RSA, im Gegensatz zu den schnellen Byte-Operationen von AES oder vergleichbaren Operationen anderer symmetrischer Verfahren. 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.

3.6.4 Fazit

Generell kann man festhalten, dass symmetrische Verschlüsselungsverfahren schneller sind als unsymmetrische, aber das grundsätzliche Problem des Schlüsselaustauschs besitzen.

Außerdem eröffnen unsymmetrische Verfahren ganz neue Funktionen in Kommunikationsystemen, wie Authentifizierung und digitale Unterschriften, die mit symmetrischen Ver-

fahren gar nicht denkbar wären.

In der Realität hat sich aber nicht das eine *gegen* das andere Prinzip durchgesetzt, sondern es gibt verschiedene Einsatzgebiete, in denen die jeweiligen Vorteile und Möglichkeiten ausgenutzt werden. So werden in Protokollen wie TSL (früher: SSL) 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*.[6]

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, wie die folgende einfache Applikation zeigt.

```
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 `KeyPairGenerator`s erzeugt, einen Ausnahmefehler wirft, wenn der genannte Algorithmus nicht existiert, muss sie in einen `try`/`catch`-Block.

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

Kapitel 4

Digitale Signatur

Öffentliche Verschlüsselungssysteme eröffnen somit eine qualitativ vollkommen neue Möglichkeit, die *digitale Signatur* oder *digitale Unterschrift*. Sie ist ausschließlich mit symmetrischen Verfahren nicht zu realisieren.

4.1 Eigenschaften einer Unterschrift

Eigenhändige Unterschriften sind schwer zu kopieren und gelten so als Beweis für die Urheberschaft eines Dokuments bzw. das Einverständnis mit seinem Inhalt. Auch elektronische Dokumente müssen signiert werden, z.B. Kaufverträge im Internet, elektronische Banktransaktionen oder verbindliche E-mails.

Allgemein muss eine Unterschrift folgende 5 Eigenschaften besitzen:

1. Sie ist *authentisch:* Das Dokument wurde vom Absender unterzeichnet. Das gilt z.B. für eine handschriftliche Unterschrift im Beisein eines Notars oder des Vertragspartners.

   ie ist *fälschungssicher:* Kein anderer hat das Dokument unterzeichnet. Bei handschriftlichen Unterschriften wird Fälschungssicherheit durch die individuelle Handschrift angenommen.

2. Sie ist *nicht wiederverwendbar:* Die Unterschrift ist Bestandteil des unterzeichneten Dokuments und in kein anderes Dokument übertragbar. Handschriftliche Unterschriften sind z.B. ungültig, wenn sie kopiert (oder gefaxt) sind.

3. Sie ist *integer mit dem Inhalt des Dokuments:* Nachdem das Dokument unterschrieben ist, kann es nicht mehr geändert werden. Daher werden Verträge doppelt ausgefertigt, Veränderungen müssen identisch bei beiden Versionen sein.

4. Sie ist *verbindlich.* Der Unterzeichner kann später nicht behaupten, er habe das Dokument nicht unterschrieben oder abgesendet.

4.2 Digitale Unterschriften mit RSA

Die Grundidee ist, dass der geheime Schlüssel des Senders als seine digitale Unterschrift dient. 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.

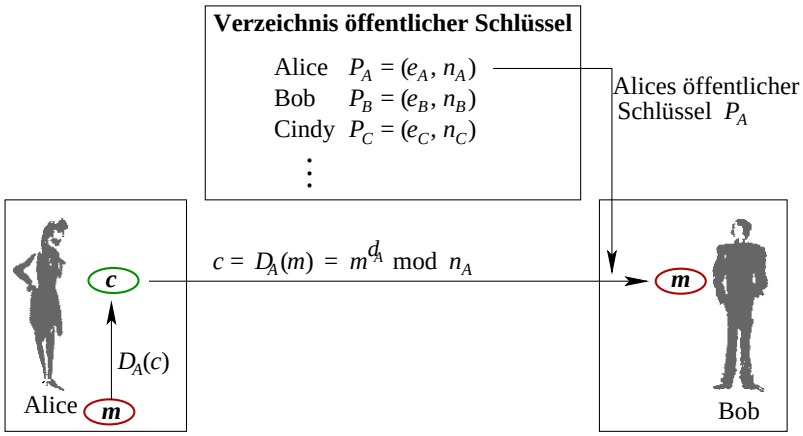


Abbildung 4.1: Alice sendet eine digital unterschriebene Nachricht an Bob; Bob verwendet Alices öffentlichen Schlüssel, um die Nachricht zu entschlüsseln und so zu verifizieren, dass Alice sie tatsächlich mit ihrem privaten Schlüssel unterschrieben hat.

Das Verfahren ist in Abb. 4.1 skizziert. Es ist „invers“ zu der Verschlüsselung. Beachten Sie, dass das signierte Dokument *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.

Die notwendigen 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.

Insbesondere für die rechtliche Gültigkeit der Unterschrift ist es wesentlich, dass die Authentizität von Alices öffentlichem Schlüssel P_A garantiert ist. Nur dann kann Bob annehmen, dass es tatsächlich ihre Unterschrift ist und nicht die einer dritten Person, die sich als Alice ausgibt. Eine Möglichkeit, diese Garantie zu geben, sind *Trust Center*, die die öffentlichen Schlüssel verwalten und für die Authentizität sorgen.

4.3 DSA

Der *Digital Signature Algorithm (DSA)*[1] ist seit 1994 der Standard für digitale Signaturen von Dokumenten der US-Regierung und wurde von der NIST und dem NSA entwickelt. Er geht auf Signierungalgorithmen von Schnorr und ElGamal zurück [12, S. 564].

4.3.1 Schlüsselerzeugung

Zur Erzeugung ihres Schlüsselpaares (S_A, P_A) muss Alice die folgenden sechs Schritte durchführen:

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.

4.3.2 Signierung

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

$$\begin{aligned} r &= (g^k \bmod p) \bmod q, \\ s &= k^{-1}(\mathrm{SHA}(m) + rS_A) \bmod q. \end{aligned} \tag{4.1}$$

[1] Offizielle Spezifikation FIPS-186 *http://www.itl.nist.gov/fipspubs/fip186.htm*

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.

4.3.3 Verifikation der Signatur

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:

$$\begin{aligned} w &= s^{-1} \bmod q, \\ u_1 &= w \cdot \mathrm{SHA}(m) \bmod q, \\ u_2 &= rw \bmod q, \\ v &= (g^{u_1} P_A^{u_2} \bmod p) \bmod q, \end{aligned} \tag{4.2}$$

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

4.4 Implementierung in Java

Die Erzeugung und Ausgabe eines DSA-Schlüsselpaares ist in Java fast identisch mit der eines RSA-Schlüsselpaares. Für die Signierung eines Dokumentes muss dieses zunächst in Bytes umgeformt werden, um dann von einem Objekt der Klasse `Signature` mit der Methode `update` eingetragen und per `sign` signiert zu werden. Die Verifizierung verläuft analog.

```
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 \u20AC.";
      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: --------\n";
         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 i.d.R. 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.

□

Der Zweck eines solchen Hash-Wertes ist es, Veränderungen der Nachricht während des Übermittlungsweges überprüfbar zu machen, 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)*.

In der Literatur werden Hash-Funktionen oft auch als „Einweg-Hashfunktionen“ [4, 6] bezeichnet. Man muss sich nur klar darüber sein, dass sie *keine* Einwegfunktionen sind.

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 oder überhaupt 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.

Einen kleinen Überblick über wichtige Hash-Funktionen gibt Tabelle 5.1. Es ist inter-

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

Tabelle 5.1: Gängige Hash-Funktionen; nach [3].

essant, dass sie alle auf dem von Ron Rivest Ende der 1980er entwickelten MD4 aufbauen und ihn verbessern. RIPEMD-160 gilt als sehr sicher. SHA-1 ist der aktuelle internationale Standard.

5.1 SHA

Der SHA *(Secure Hash Algorithm)* wurde von dem NIST und der NSA entwickelt und ist der aktuelle Standard als Hash-Funktion. Er verläuft in den folgenden Schritten.

1. *Aufteilen der Nachricht in 512-bit-Blöcke:* Die Nachricht m wird so aufgefüllt, dass ihre Länge ein Vielfaches von 512 bit beträgt. Im Detail wird an die binäre Darstellung der Nachricht eine 1 angehängt und dann soviele Nullen, dass die Länge einem Vielfachen von 512 - 64 bit entspricht, sodann wird eine 64-bit-Darstellung der (originalen) Nachricht angehängt.

2. *Bildung von 80 Wörtern à 32 bit:* Jeder einzelne der 512-bit-Blöcke wird nun in 16 Blöcke $M_0, M_1, \ldots, M_{15}$ à 32 bit geteilt. Diese werden dann in 80 Wörter $W_0, \ldots, W_{79}$ umgewandelt:

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

Hierbei bezeichnet $\lll$ die *Bit-Rotation* oder *zirkuläre Linksverschiebung* (z.B. `10100` $\lll$ `1` $=$ `01001`).[1]

3. *Initialisierung der Variablen und Konstanten:* In SHA werden 80 Konstanten $K_0, \ldots, K_{79}$ (mit jedoch nur vier verschiedenen Werten), gegeben durch

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

fünf Konstanten $A, \ldots, B$, gegeben durch

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

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

und fünf Variablen[2] $A, \ldots, E$ 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++) {
  tmp $= (a \lll 5) + f_t(b,c,d) + e + W_t + K_t$;
  $e = d$;
  $d = c$;
  $c = b \lll 30$;
  $b = a$;
  $a =$ tmp;
}

Hierbei ist die nichtlineare Funktionenschar f_t definiert durch

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

[1] Die ursprüngliche SHA-Spezifikation der NSA enthielt die Bit-Rotation noch nicht. Sie „korrigiert ein technisches Problem, durch das der Standard weniger sicher wurde als ursprünglich gedacht“ [12, S. 506]. Die NSA hat die Natur des Problems meines Wissens nie genauer erläutert.

[2] Der fast identisch ablaufende MD5 hat nur 64 Konstanten $K_i = \lfloor 2^{32} |\sin i| \rfloor$, nur vier Konstanten $A, \ldots, D$ und vier Variablen (er produziert ja auch nur Hash-Werte von 128 bit).

5.2 Angriffe auf Hash-Funktionen

Die Qualität einer kryptographischen Hash-Funktion beurteilt man vor allem nach ihrer Widerstandsfähigkeit gegen zwei Typen von Angriffen:

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 beispielsweise bildet Hash-Werte mit 160 Bit. Es gibt somit insgesamt ‚nur' $2^{160} \approx 1,5 \cdot 10^{48}$ verschiedene Hash-Werte, und viele Worte ergeben folglich denselben Hash-Wert.

Hat man einen vorgegebenen Hash-Wert und probiert 2^{160} zufällige Nachrichten durch, ist die Wahrscheinlichkeit, eine mit dem gleichen zu erhalten, sehr hoch. Dieser Vorgang würde mit reeller Hardware allerdings weit über 100 Millionen Jahre dauern. Will man nur eine Kollision finden, also zwei Nachrichten mit dem gleichen aber beliebigen Hash-Wert, muss man für ähnliche Erfolgschancen ‚nur' $2^{80} \approx 1,2 \cdot 10^{24}$ zufällig ausgewählte Nachrichten durchprobieren. Preimage-Angriffe auf SHA-1 erfordern damit nicht doppelt soviele Versuche, sondern 2^{80} mal so viele Hash-Operationen wie ein Kollisionsangriff. Trotzdem liegt auch die einfache Suche nach SHA-1-Kollisionen noch weit außerhalb des technisch Machbaren.

5.2.1 Das Geburtstagsparadoxon

Den enormen Unterschied zwischen dem Aufwand für Preimage- 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

Der MAC *(message authentication code)* oder Authentifizierungscode ist ein 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[3] und wird von dem Trust-Center signiert. Damit garantiert es die Verbindung von Namen und öffentlichem Schlüssel.

Das Protokoll von TSL besteht aus drei Schritten, der Authentifikation, dem Schlüsselaustausch und dem eigentlichen Nachrichtenaustausch:

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

$A \to B$:	Hallo!
$B \to A$:	Hallo, ich bin Bob, *Zertifikat(Bob)*.
$A \to B$:	Beweise es!
$B \to A$:	$D_B[m, h(m)]$, 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:

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

   K heißt auch Sitzungsschlüssel *(session key)*.

[3] Daneben enthält es andere Informationen wie Gültigkeitsdauer, Austellungsdatum usw.; zur Erläuterung mögen die obigen Angaben reichen.

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.

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

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

5.5 Angriffe auf signierte Dokumente

5.5.1 Nachträgliches Fälschen

Um beispielsweise einen digital signierten Vertrag nachträglich zu fälschen, müsste der Angreifer einen Preimage-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 Preimage-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. Das illustriert das folgende, etwas vereinfachte Beispiel.

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. □

Auch durch Preimage-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.

Kapitel 6

Quantenkryptographie

Qubits sind die quantenphysikalische Erweiterung klassischer Bits. Das Besondere an einem Qubit ist, dass es im Gegensatz zu einem Bit nicht nur zwei Zustände 0 und 1 annehmen kann, sondern *unendlich viele*. Wird es jedoch gemessen, so kann man nur einen von zwei möglichen Messwerten „0“ oder „1“ erhalten. Ihr Eintreten hängt von gewissen Wahrscheinlichkeiten ab, die durch die Messapparatur (genauer: die von ihr bestimmten „Basiszustände“) und den Zustand des Qubits bestimmt sind.

6.1 Qubits

Ein *Qubit* ist ein Quantensystem, das als ein Einheitsvektor $|\psi\rangle$ in einem zweidimensionalen (komplexen) Vektorraum dargestellt wird, dessen zwei Basisvektoren mit $|0\rangle$ und $|1\rangle$ bezeichnet werden,

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle, \qquad \text{wo } \alpha_0, \alpha_1 \in \mathbb{C}, \quad \text{mit } |\alpha_0|^2 + |\alpha_1|^2 = 1. \tag{6.1}$$

Die Basiszustände werden auch *Eigenzustände* genannt. Die komplexen Konstanten α_0 und α_1 heißen *Wahrscheinlichkeitsamplituden*. Für α_0, $\alpha_1 \neq 0$ heißt der Zustand $|\psi\rangle$ eine „Superposition“ der Basiszustände (Abb. 6.1). Gilt andererseits $|\psi\rangle = |0\rangle$ oder $|1\rangle$, so ist das Qubit in einem *reinen Zustand*.

Beispiel 6.1 Ein allgemeines Qubit kann durch

$$|\psi\rangle = \mathrm{e}^{\mathrm{i}\delta} \begin{pmatrix} \mathrm{e}^{-\mathrm{i}\varphi/2} \cos\frac{\vartheta}{2} \\ \mathrm{e}^{\mathrm{i}\varphi/2} \sin\frac{\vartheta}{2} \end{pmatrix} \tag{6.2}$$

mit den drei Winkeln δ, φ, $\vartheta \in [0, 2\pi)$. dargestellt werden. Damit gilt $\alpha_0 = \mathrm{e}^{\mathrm{i}(\delta-\varphi/2)} \cos\frac{\vartheta}{2}$ und $\alpha_1 = \mathrm{e}^{\mathrm{i}(\delta+\varphi/2)} \sin\frac{\vartheta}{2}$, insbesondere

$$|\alpha_0|^2 = \cos^2\frac{\vartheta}{2}, \qquad |\alpha_1|^2 = \sin^2\frac{\vartheta}{2}. \tag{6.3}$$

Das ergibt $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Der Winkel δ heißt die *globale Phase* und φ die *relative Phase* des Qubits [10, §2.2.7]. □

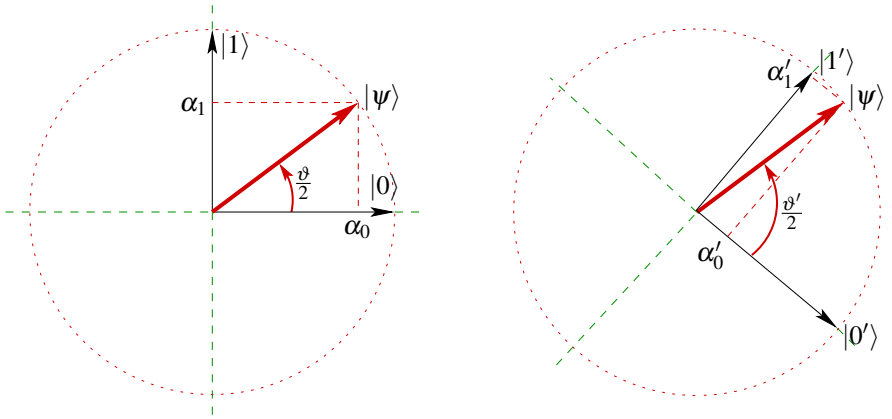


Abbildung 6.1: Ein Qubit $|\psi\rangle$ als eine Superposition von Basiszuständen $|0\rangle$ und $|1\rangle$, d.h. $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, mit den Amplituden α_0 and α_1. Ein Qubit entspricht also einem Vektor auf dem gepunkteten Kreis, hier illustriert als *reeller* Unterraum des $\mathbb{R}^2$. Rechts dasselbe Qubit mit einem um 45° gedrehten Basissystem $|0'\rangle$ und $|1'\rangle$ mit $|\psi\rangle = \alpha'_0|0'\rangle + \alpha'_1|1'\rangle$. Die Basiszustände links bilden die „+"-Basis, rechts die „×"-Basis.

Postulat 1 (Messung eines Qubits) Wenn ein Qubit $|\psi\rangle$ in (6.1) gemessen wird, ist das Messergebnis „0" mit der Wahschreinlichkeit $|\alpha_0|^2$ and „1" mit der Wahrscheinlichkeit $|\alpha_1|^2$:

$$P(j) = |\alpha_j|^2 \qquad \text{for } j = 0, 1. \tag{6.4}$$

Nach der Messung ist das Qubit entweder in Zustand $|0\rangle$ oder in Zustand $|1\rangle$:

$$|\psi'\rangle = \begin{cases} |0\rangle & \text{wenn das Messergebnis „0" war,} \\ |1\rangle & \text{wenn das Messergebnis „1" war.} \end{cases} \tag{6.5}$$

Hier bezeichnet $|\psi'\rangle$ den Zustand des Qubits nach der Messung. Oft sagt man, dass durch eine Messung des Zustands $|\psi\rangle$ das System in die Eigenzustände $|0\rangle$ oder $|1\rangle$ „kollabiert".

Die Wirkung einer Messung, d.h. der Wechsel von Zustand $|\psi\rangle$ nach Zustand $|\psi'\rangle = |0\rangle$ oder $|1\rangle$, ist in Abbildung 6.2 skizziert. Eine wesentliche Eigenschaft der Quantenmechanik ist, dass die Messung eines Quantensystems, hier eines Qubits, *den Zustand selbst verändert*. Das ist ganz im Gegensatz zu unserer Alltagserfahrung, nach der die reine Beobachtung von Objekten deren Zustand nicht ändert. Beispielsweise können Sie ein Bild anschauen, ohne dass Sie dadurch seinen Inhalt oder seine Farben ändern. Auch beim Sport ändert der Ball seine Richtung nicht allein dadurch, dass Sie ihn beobachten. Nach der Messung ist der Zustand des Qubits exakt bekannt, es ist in einem reinen Zustand.

Beispiel 6.2 Es sei das Qubit aus Beispiel 6.1 gegeben. Mit Gleichung (6.3) gilt dann

$$P(0) = \cos^2 \vartheta, \qquad P(1) = \sin^2 \vartheta. \tag{6.6}$$

Obwohl ein Qubit allgemein von drei Parametern abhängt, den Winkeln δ, φ, und ϑ, kann nur der Winkel ϑ direkt gemessen werden und ist daher eine physikalisch beobachtbare

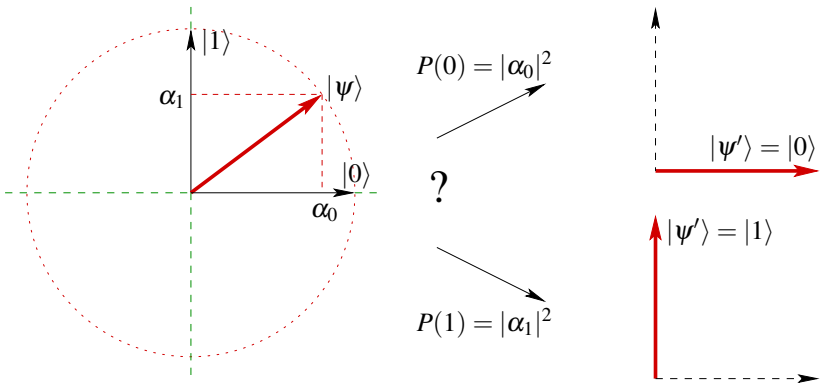


Abbildung 6.2: Die Wirkung einer Messung eines Qubits: Der Qubit-Zustand $|\psi\rangle$ kollabiert zu $|\psi'\rangle$, und zwar entweder $|0\rangle$ mit Wahrscheinlichkeit $P(0) = |\alpha_0|^2$ oder $|1\rangle$ mit Wahrscheinlichkeit $P(1) = |\alpha_1|^2$.

Größe. Die globale Phase δ und die relative Phase φ haben keine direkte physikalische Bedeutung für ein einzelnes Qubits, da sie nicht gemessen werden können. (Für mehrere Qubits jedoch bewirken sie Interefrenzen und sind daher im Allgemeinen durchaus physikalisch relevant.) Speziell für das Qubit mit $\delta = \varphi = 0$ und $\vartheta = 0$ gilt $|\psi\rangle = |0\rangle$, es ist also in einem reinen Zustand. Das Qubit mit $\vartheta = \pi$ lautet entsprechend $|\psi\rangle = |1\rangle$ und ist ebenfalls in einem reinen Zustand. Qubits in reinen Zuständen verhalten sich genau wie klassische Bits. □

Allgemein heißt der Kollaps einer Superposition in einen reinen Zustand *Dekohärenz*. Dekohärenz ist ein weit allgemeineres Phänomen, das immer dann auftritt, wenn ein Quantensytem mit seiner Umgebung wechselwirkt. Eine Messung ist ein spezieller Fall von Dekohärenz.

Das Postulat der Messung impliziert übrigens die *Heisenbergsche Unschärferelation* [14, Gl. (133)]. Sie besagt, dass man den vollständigen Zustand eines Quantensystems, also beispielsweise Ort *und* Impuls eines Elementarteilchens, prinzipiell nicht gleichzeitig exakt messen kann.

Physikalisch kann ein Qubit durch jedes Quantensystem realisiert werden, dass sich in genau zwei Basiszuständen befinden kann. In der Quantenkryptographie sind das üblicherweise Photonen, deren Polarisationsrichtung nur entweder $|0\rangle$ = „horizontal polarisiert“ oder $|1\rangle$ = „vertikal polarisiert“ sein kann. Weitere Beispiele sind Ammoniakmoleküle NH_3 in einem Ammoniak-Maser, die nur zwei Basiszustände (entsprechend den geometrischen Positionen des N-Atoms entweder über oder unter der Ebene der drei H-Atome) annehmen können, oder „magnetische Resonanz“, d.h. rotierende Teilchen in einem magnetischen Feld, z.B. ein Proton *(nuclear magentic resonance NMR)* oder ein Elektron in einem Kristallgitter [7, §16].

6.1.1 Messung in unterschiedlichen Basissystemen

Ein gegebenes Qubit kann auch bezüglich eines anderen Systems von Basiszuständen $|0'\rangle$ und $|1'\rangle$ gemessen werden. Aus der linearen Algebra ist bekannt, dass zwei Basissysteme

durch eine orthogonale Transformation (also eine Spiegelung oder eine Rotation) ineinander überführt werden können. Für das Beispiel aus Abb. 6.1 gilt insbesondere

$$|0'\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \quad |1'\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \tag{6.7}$$

bzw.

$$\begin{pmatrix} |0'\rangle \\ |1'\rangle \end{pmatrix} = A \cdot \begin{pmatrix} |0\rangle \\ |1\rangle \end{pmatrix} \qquad \text{mit} \qquad A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{pmatrix}. \tag{6.8}$$

A ist eine Rotation um den Winkel $\frac{\pi}{4} = 45°$ im Uhrzeigersinn (... die *Komponenten* eines Vektors würden aber *gegen* den Uhrzeigersinn, also mathematisch positiv gedreht). Wir werden das Basissystem $\{|0\rangle, |1\rangle\}$ kurz als „+"-Basis bezeichnen, das System $\{|0'\rangle, |1'\rangle\}$ als „×"-Basis.

Ist nun ein Qubit $|\psi\rangle$ in der „+"-Basis mit den Wahrscheinlichkeitsamplituden α_0, α_1 gegeben, so ergeben sich die Wahrscheinlichkeitsamplituden in der „×"-Basis durch die Beziehungen (6.7) gemäß

$$\alpha_0' = \frac{\alpha_0 - \alpha_1}{\sqrt{2}}, \qquad \alpha_1' = \frac{\alpha_0 + \alpha_1}{\sqrt{2}}, \tag{6.9}$$

Das bedeutet, dass dasselbe Qubit bezüglich verschiedener Basiszustände die Messwerte „0" oder „1" mit verschiedenen Wahrscheinlichkeiten ergibt. Diese physikalische Eigenschaft wird sich als wesentlich für Quantenkryptographie herausstellen.

6.1.2 Quantenregister

Ein *Quantenregister* $|\psi\rangle$ der Größe n ist ein Quantensystem aus n Qubits,

$$|\psi\rangle = \boxed{|\psi_n\rangle} \boxed{|\psi_{n-1}\rangle} \boxed{\ldots} \boxed{|\psi_2\rangle} \boxed{|\psi_1\rangle}$$

Wir schreiben ein Quantenregister der Größe n in der Form[1] $|\psi\rangle = |\psi_n \psi_{n-1} \ldots \psi_1\rangle$. Es gibt 2^n Basiszustände, dargestellt durch Binärstrings der Länge n:

$$|\underbrace{0\ldots00}_{n \text{ places}}\rangle, \quad |0\ldots001\rangle, \quad \ldots, \quad |01\ldots11\rangle, \quad |11\ldots1\rangle, \tag{6.10}$$

(Daher ist der Zustandsraum eines Quantenregisters $\mathbb{C}^{2^n}$.) Aus Bequemlichkeit wird ein solcher Basiszustand oft einfach durch seine Dezimaldarstellung ausgedrückt, beispielsweise für $n = 6$,

$$|1\rangle = |000001\rangle, \quad |2\rangle = |000010\rangle, \quad |5\rangle = |000101\rangle, \quad \ldots \tag{6.11}$$

Eine Dezimalzahl j entspricht dem Binärstring $s = (s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$ durch die Formel

$$j = \sum_{k=0}^{n-1} s_k \cdot 2^k. \tag{6.12}$$

[1] Oft findet man auch die Notation $|\psi\rangle = |\psi_n\rangle|\psi_{n-1}\rangle \ldots |\psi_1\rangle$, oder auch $|\psi\rangle = |\psi_n\rangle \otimes |\psi_{n-1}\rangle \otimes \ldots \otimes |\psi_1\rangle$, das „Tensorprodukt".

Ein allgemeines Quantenregister $|\psi\rangle$ der Größe n ist durch

$$|\psi\rangle = \sum_{s\in\{0,1\}^n} \alpha_s |s\rangle = \sum_{j=0}^{2^n-1} \alpha_j |j\rangle \qquad \text{mit} \sum_{j=0}^{2^n-1} |\alpha_j|^2 = 1, \tag{6.13}$$

gegeben. Die zweite Gleichung ist lediglich eine Umformulierung der Binärstrings $s \in \{0,1\}^n$ als Dezimalzahlen. Tatsächlich besagt Gl. (6.13), dass ein Quantenregister eine Superposition einiger, wenn nicht gar aller, der 2^n Basiszustände

$$|0\cdots00\rangle, \quad |0\cdots01\rangle, \quad \ldots, \quad |1\cdots11\rangle.$$

sein kann.

6.1.3 Verschränkung

Was bedeutet nun jedoch eine Superposition in einem Quantenregister? Z.B. kann ein Quantenregister der Größe $n = 3$ individuelle Zahlen wie 3 oder 7 speichern, $|011\rangle = |3\rangle$, $|111\rangle = |7\rangle$, jedoch auch zwei oder mehrere gleichzeitig. Anstatt nämlich das erste Qubit auf $|0\rangle$ oder $|1\rangle$ zu setzen, können wir es in eine Superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle$ bringen, und wir erhalten

$$\frac{1}{\sqrt{2}}\Big(|0\rangle + |1\rangle\Big)|1\rangle|1\rangle = \frac{1}{\sqrt{2}}\Big(|011\rangle + |111\rangle\Big) = \frac{1}{\sqrt{2}}\Big(|3\rangle + |7\rangle\Big) \tag{6.14}$$

Bei einer Superposition in einem Quantenregister werden also *gleichzeitig mehrere* Zustände gespeichert, im Höchstfall 2^n Zustände. Diese Eigenschaft ist so bemerkenswert, dass man für einen solchen Zustand einen eigenen Begriff geprägt hat.

Definition 6.3 Ein Quantenregister der Größe $n \geqq 2$ in einer Superposition von mindestens zweien der 2^n Basiszustände $|00\cdots0\rangle$, $|01\cdots0\rangle$, …, $|11\cdots1\rangle$, ist in einem *verschränkten Zustand.*[2] Die beteiligten Basiszustände heißen *(miteinander) verschränkt.* □

Mit anderen Worten kann man bei einem Quantenregister in verschränktem Zustand nicht sagen „das erste Qubit ist in Zustand α_0, das zweite in Zustand α_1, …“. Bemerkenswerterweise ist in der Quantenmechanik Verschränkung nicht die Ausnahme, sondern die Regel.

Die speziellen Zustände zweier verschränkter Qubits

$$|\Phi^\pm\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle \pm |11\rangle\right), \qquad |\Psi^\pm\rangle = \frac{1}{\sqrt{2}}\left(|01\rangle \pm |10\rangle\right) \tag{6.15}$$

heißen *Bell-Zustände*, *Singlets* oder *EPR-Paare*, zu Ehren von Einstein, Podolsky und Rosen, die in einer Veröffentlichung [5] von 1935 als erste solche Zustände und deren Konsequenzen untersuchten. Es ist zu beachten, dass wir nun nicht das erste oder das zweite Qubit separat messen können, denn sie sind unentwirrbar miteinenader verbunden. Stattdessen lässt eine Messung eines der Qubits das gesamte System, also *beide* Qubits zu einem der vier Bell-Zustände kollabieren.

[2] Dieser Begriff geht auf den österreichischen Physiker Erwin Schrödinger (1887–1961) zurück.

Verschränkte Zustände sind ein sehr bemerkenswertes Phänomen, da sie die Eigenschaft der „Nichtlokalität“ besitzen: Es kann der Fall eintreten, dass zwei verschränkte Qubits sehr weit voneinander entfernt sind; eine Messung des einen Qubits bestimmt *im selben Augenblick* den Zustand des anderen Qubits. Das heißt, es liegt eine „instantane Fernwirkung“ vor. Wenn beispielsweise das erste Qubit zunächst im Bell-Zustand Ψ^- ist und nach seiner Messung das klassische Ergebnis „0“ liefert, so kollabiert das zweite Qubit in den Zustand $|1\rangle$ und liefert nach einer Messung sicher „1“, und umgekehrt. Die Nichtlokalität widerspricht der Relativitätstheorie.

6.2 Quantenschlüsselaustausch (QKD)

Als Voraussetzung eines Protokolls für den quantenkryptographischen Schlüsselaustausch *(quantum key distribution QKD)* müssen Alice und Bob je eine Messapparatur besitzen, mit der sie Qubits in mindestens zwei Basen „$+$“ oder „$\times$“ messen können.

Das EPR-Protokoll

1. Eine Quelle erzeugt EPR-Paare, also verschränkte Qubitpaare, die sich alle in einem vorgegebenen der vier Bell-Zustände (6.15) befinden, der Alice und Bob bekannt ist. Jeweils ein Qubit jedes Paares wird an Alice und eines an Bob verschickt.

2. Sowohl Alice als auch Bob messen die Qubits. Dabei verwenden sie jeweils die Basis „$+$“ oder „$\times$“ nach einer der beiden Varianten:

   (a) Über den unsicheren Kommunikationskanal vereinbaren sie *nach* Erhalt eines Qubits, bezüglich welcher Basis sie messen.

   (b) Beide wählen die Basis bei jedem Qubit per Zufall aus und teilen sich über den unsicheren Kommunikationskanal gegenseitig mit, bei welchem Qubit sie welche Basis verwendet haben. Sie verwenden nur die Messwerte, bei denen die Basis übereinstimmte, die anderen vergessen sie.

   Entsprechend dem festgelegten Bell-Zustand wissen daher beide genau, was der Andere gemessen hat.

3. Über den unsicheren klassischen Kanal teilen sie sich gegenseitig ihre Ergebnisse einiger ihrer Messungen mit. Korrelieren sie entsprechend des gegebenen Bell-Zustands exakt, so wissen sie, dass diese Qubits EPR-Paare waren und der EPR-Kanal daher nicht gestört wurde. Die Messergebnisse aller Qubits können daher als Geheimschlüssel verwendet werden.

Der so ausgetauschte Geheimschlüssel ist echt zufällig erzeugt, beliebig lang und abhörsicher übertragen. Er eignet sich daher für ein One-Time-Pad. Strenggenommen wird durch QKD also nicht nur ein Schlüssel ausgetauscht, sondern auch erzeugt.

Das EPR-Protokoll ist das allgemeinste der gängigen QKD-Protokolle. Das historisch erste BB84 wurde von Bennett und Brassard 1984 vorgestellt, ein weiteres B92 von Bennett acht Jahre später.

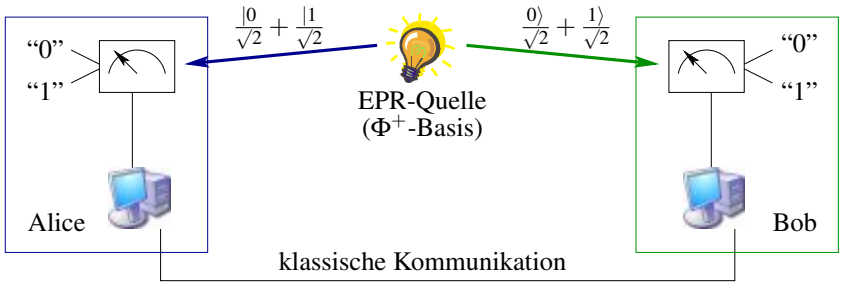


Abbildung 6.3: EPR-Protokoll zur Schlüsselerzeugung und zum Schlüsselaustausch mit Quantenteleportation. Die EPR-Quelle erzeugt hier jeweils Qubit-Paare (z.B. Photonen) im Bell-Zustand $\Phi^+ = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$, von denen das erste zu Alice und das zweite zu Bob gesendet wird. Alice und Bob erhalten dann jeweils gleiche Messwerte, wenn der EPR-Zustand nicht gestört wurde.

6.2.1 Physikalische Realisierungen

Die bisher aufgeführten Eigenschaften von Quantensystemen sind nicht mehr nur bloße Theorie. Insbesondere das Phänomen der Verschränkung war bis 1982 ein eher akademisches Paradoxon, aber seit dem bahnbrechenden Experiment von Alain Aspect [1] war es physikalische Realität, und zusehends übernahmen Experimentalphysiker die Initiative, Ein weiterer Meilenstein war die Quantenteleportation von der Arbeitsgruppe von Zeilinger 1997 [2]. Dadurch erwachsen stetig Erfahrung und auch Intuition über Quantensysteme. Man debatiert nicht mehr darüber, wieviel Engel eigentlich auf einer Nadelspitze Platz finden können, sondern entdeckt Realitäten.

So gibt es bereits nicht nur quantenkryptographische Experimente, wie die erste mit QKD verschlüsselte Banktransaktion am 21.4.2004 in Wien (*www.quantenkryptographie.at*) sondern sogar kommerzielle Produkte:

- Der QKD-Apparat *Clavis* der Schweizer Firma idQuantique (*www.idquantique.com*) von Prof. Nicolas Gisin der Universität Genf ermöglicht Schlüsselaustausch über Glasfaserkabel bis zu 60 km Länge. (Preis[3]: etwa 100 000 €)

- Die US-Firma MagiQ Technologies (*www.magiqtech.com*) bietet mit QPN Security Gateway eine Kombination aus klassischer und Quantenkryptographie an, die kompatibel mit existierenden unsymmetrischen Verschlüsselungsverfahren und PKI's ist, allerdings auf einem Glasfasernetz basiert. (Preis[3]: etwa 150 000 US $)

[3] *http://www.carnet.hr/CUC/cuc2004/program/radovi/a4_stipcevic/a4_presentation.pdf*

6.3 Checkliste der Kryptoprobleme

Quantenkryptographie, bzw. Quantenschlüsselaustausch QKD, löst erstmals das Problem der Abhörerkennung. Da die eigentliche Information des erzeugten Geheimschlüssels über verschränkte Qubits erfolgt, ist sogar das Problems des Schlüsselaustauschs gelöst. Die einzige Lücke der Quantenkryptographie ist, dass eine Authentifizierung nicht möglich ist. Wie soll Alice erkennen, dass sie tatsächlich mit Bob kommuniziert, und umgekehrt? Daher ist das Problem des Schlüsselaustauschs nur unter der Voraussetzung gelöst, dass Alice und Bob sicher sein können, dass sie miteinander kommunizieren.

Checkliste für Quantenkryptographie	
Schlüsselaustausch gelöst?	ja und nein
Authentifizierung?	nein
Abhörerkennung?	ja

Kapitel 7

Schlusswort

Wir haben die drei Zweige der modernen rechnerbasierten Kryptographie kennen gelernt, die *S-Kryptographie* (für *secret key*), die *P-Kryptographie* (für *public key*) und die *Q-Kryptographie* (für *quantum*). Jede hat ihre Stärken und Schwächen, keine kann die drei Hauptprobleme der Kryptographie lösen. Mit der Kombination aller drei Zweige jedoch gelingt dies, wenn nämlich Alice und Bob nach dem EPR-Protokoll (Q-Kryptographie) einen Geheimschlüssel (S-Kryptographie) austauschen und über eine unsymmetrische Chiffre (P-Kryptographie) kommunizieren.

Checkliste für S, P & Q-Kryptographie	
Schlüsselaustausch gelöst?	ja
Authentifizierung?	ja
Abhörerkennung?	ja

Doch das Rennen zwischen Hase (Kryptographie) und Igel (Kryptanalyse) geht weiter. Schon gibt es Ideen, dass ein Angreifer einer QKD sich „durchsichtig“ mit den EPR-Paaren verschränkt. Auch der P-Kryptographie droht große Gefahr, denn seit 1994 ist der Shorsche Quantenalgorithmus bekannt, der die Faktorisierung großer Zahlen effizient durchführt und damit RSA, eventuell sogar alle derzeitigen Chiffren bricht — jedoch nur auf Quantenrechnern läuft, die es zur Zeit (2005) noch nicht in ausreichender Größe gibt.

Wir sind mit unserem kryptologischen Wissen an nie geahnte Horizonte gestoßen. Erst vor knapp 30 Jahren wurde das Problem der Authentifizierung gelöst, die perfekte Abhörerkennung einer Kommunikation entsprechend erst vor 20 Jahren. Viele Probleme der Kryptologie werden wohl noch gestellt und gelöst werden, die wir uns heute noch gar nicht vorstellen können. Irgendwann wird man auf die Inhalte des vorliegenden kurzen Beitrags vielleicht ähnlich amüsiert blicken, wie wir mit unseren heutigen Maschinen auf die Cäsar-Verschiebung.

Das Rennen geht weiter.

Anhang

A.1 AES im Detail

A.1.1 Byte-Addition und Byte-Multiplikation

Wir stellen ein Byte dar als einen binären String („Vektor“):

$$x = (x_7, x_6, \ldots, x_1, x_0), \qquad \text{mit } x_n = 0 \text{ oder } 1.$$

Die Byte-Addition $\oplus$ definieren wir dann durch ein bitweises XOR: Für zwei Byte x und y mit $x = (x_7, ..., x_0)$ und $y = (y_7, ..., y_0)$ gilt

$$x \oplus y = (x_7 + y_7 \bmod 2, \ldots, x_0 + y_0 \bmod 2). \tag{A.1}$$

Z.B. ist $(11010111) \oplus (100000011) = (01010100)$. Man sieht, dass die Byte-Addition zweier Bytes wieder ein Byte ergibt (es gibt keinen Übertrag wie in der normalen Addition). Für AES wird auch die Multiplikation etwas verändert, wir berechnen die Byte-Multiplikation $x \odot y$ in zwei Schritten:

1. Berechne $z = x \cdot y$ mit Hilfe der $\oplus$-Addition.

2. Berechne z modulo dem Binärstring

$$m = (1|00011011) \tag{A.2}$$

   (In Hexadezimaldarstellung gilt $m = 11\text{B}$, oder in Dezimaldarstellung $m = 283_{10}$; m ist eine Primzahl.) Der Deutlichkeit halber schreiben wir einen vertikalen Strich an der Stelle, an der das letzte Byte beginnt. Für die Modulo-Rechnung verwendet man die (binäre) ganzzahlige Division mit Rest, in der statt mit „+“ mit $\oplus$ gerechnet wird. **Sonderregel:** Ist z größer als ein Byte, also $z \geqq (1|00000000)$, so wird immer einmal durch m geteilt, also auch wenn $z < m$ (dann mit „negativem“ Rest). Auf diese Weise ist der modulo-Wert immer ein Byte.

Beispiel A.1 Für $x = 111$, $y = 11$ (die führenden Nullen und die Klammern sind der Übersichtlikeit halber weggelassen):

$$\begin{array}{r} 111 \cdot 11 \\ \hline 111 \\ \oplus \quad 111 \\ \hline 1001 \end{array} \tag{A.3}$$

also $z = 111 \cdot 11 = 1001$. Dann ist $z \bmod m = 1001$, denn $z < m$ und $z < 1|00000000$, also Alles in Allem:

$$\boxed{111 \odot 11 = 1001.} \tag{A.4}$$

□

Beispiel A.2 Sei $x = 1010111$, $y = 10000011$. Dann gilt wegen

$$\begin{array}{r} 1\,010111 \cdot 10000011 \\ \hline 1010111 \\ \oplus \quad 1010111 \\ \oplus \quad 1010111 \\ \hline 10101101111001 \end{array} \tag{A.5}$$

also $z = 1010111 \cdot 1000011 = 101011|01111001$. Dann ist $z \bmod m$ nach Byte-Division mit Rest durch m:

$$\begin{array}{r} 101\ 011|01111001 \div 1|00011011 = 1001000 \text{ Rest } 11000001 \\ \oplus\ 100\ 011\ 011 \\ \hline 1|000\ 00011 \\ \oplus\ \ 1\ 000\ 11011 \\ \hline 11000001 \end{array} \tag{A.6}$$

Also ist

$$\boxed{1010111 \odot 1000011 = 11000001.} \tag{A.7}$$

□

Warum muss man so „krumme“ Operationen verwenden? Leider ist es mathematisch nicht anders möglich, konsistentes Rechnen mit einzelnen Bytes, also 256 Elementen, zu ermöglichen. Andererseits sind Operationen mit Bytes sehr effizient zu implementieren, so dass sie für Computer oder auch Smart-Cards schnell durchführbar sind – obwohl wir Menschen uns damit etwas schwerer tun.[1] Man nennt die Menge eines Bytes mit seinen $2^8 = 256$ Elementen, auf der die Byte-Addition $\oplus$ und die Byte-Multiplikation $\odot$ wie oben definiert sind, auch *Galois-Körper*[2] *(Galois field)*, kurz $GF(2^8)$ [6, § A4], [9, App. C].

[1] Mathematisch sind diese Byte-Operationen äquivalent zu den Operationen auf Polynomen in x mit Koeffizienten modulo 2. Der Koeffizient a_i der i-ten Potenz von x^i ist also entweder 0 oder 1 und befiindet sich an der i-ten Stelle des entsprechenden Bit-Strings, gezählt von hinten und beginnend mit der nullten Stelle. Z.B. ist das Polynom $x^2 + 1$ äquivalent zu 101. Der Modulo-Wert m ist in dieser Darstellung das Polynom

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Die Menge der Polynome bildet zusammen mit diesen „krummen“ Operationen einen „Körper“, also eine Menge von Objekten, mit denen man wie mit rationalen oder reellen Zahlen konsistent rechnen kann.

[2] Évariste Galois (1811–1832) , französischer Mathematiker, starb tragischerweise an den Verwundungen infolge eines Duells, dessen Gründe unklar sind aber wohl mit seiner Liebe zu einer Frau zusammen hingen, nach welchem ihn sein Gegner und seine eigenen Sekundanten sterbend zurückließen. Er gilt mit seinen revolutionären Ideen als Begründer der Gruppentheorie. Er hat einige wesentliche Gedanken noch in der Nacht vor dem Duell niedergeschrieben.

A.1.2 Die subBytes-Transformation

Die subBytes-Transformation ist der eigentliche Kern von AES. Sie wird auf jedes Byte des Zustands S_i einzeln angewendet. Kennzeichnen wir solch ein Byte durch s, d.h.

$$s = (s_7, s_6, \ldots, s_1, s_0).$$

Dann besteht subBytes aus den folgenden Schritten:

1. Berechne das Byte a, so dass gilt

$$s \odot a = 1. \tag{A.8}$$

   Das geschieht effizient mit dem Erweiterten Euklidischen Algorithmus, angewandt auf Bytes. Sonderfall: Das Null-Byte $s = (00000000)$ wird abgebildet auf sich selbst. Durch unsere oben definierte Byte-Multiplikation $\odot$ modulo m ist dieses a eindeutig.

2. Berechne das Byte $b = (b_7, \ldots, b_0)$, das durch die folgende (affine) Transformation aus a hervorgeht, wenn wir die Bytes als Spaltenvektoren schreiben:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \pmod 2 \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \tag{A.9}$$

   Die einzelnen Additionen sind modulo 2 zu rechnen.

Der erste Schritt ist die einzige nichtlineare und nichtaffine Operation des gesamten Algorithmus, der zweite „walgt“ die einzelnen Bits noch einmal kräftig durch. Dieser Schritt ist notwendig, da sonst das s-Byte der nächsten Runde genau das a-Byte der aktuellen wäre — und damit wäre nach jeder zweiten Runde wieder der alte s-Wert berechnet! Insbesondere wird nun das Null-Byte $s = (00000000)$ abgebildet auf $b = (001100011)$, also dezimal: $0 \mapsto 99_{10}$, oder hexadezimal: 0x00 $\mapsto$ 0x63.

Die subBytes-Transformation wird auch als „S-Box“ *(substitution box)* von AES bezeichnet. Man kann sie sehr effizient als eine 16×16-Tabelle (bzw. ein Array der Länge 256) implementieren, in der man die ersten 4 Bits in der Vertikalen gegen die letzten 4 Bits in der Horizontalen aufträgt (Abb. 2.3).

A.1.3 Exkurs: Nichtlinearität

Was heißt „Nichtlinearität“ genau? Mathematisch korrekt definiert man: Seien V und W zwei Mengen, auf denen jeweils eine Addition $+$ definiert ist. Dann heißt eine Abbildung $f : V \to W$ *linear*, wenn für jedes Paar $u, v \in V$ gilt

$$f(u+v) = f(u) + f(v). \tag{A.10}$$

Beispiel A.3 Ein einfacher Fall liegt vor für $V = U = \mathbb{R}$ und der üblichen Addition und $f(x) = cx$ für eine Konstante $c \in \mathbb{R}$, denn für alle $x, y \in \mathbb{R}$ gilt $f(x+y) = c(x+y) = cx+cy = f(x)+f(y)$. Aber $g(x) = x^2$ ist nicht linear, denn es gilt schon $g(2) = 4$, aber $g(1)+g(1) = 2$, also $g(1+1) \neq g(1)+g(1)$. □

Beispiel A.4 Für $V = \mathbb{R}^m$ und $W = \mathbb{R}^n$ ist die Funktion $f : \mathbb{R}^m \to \mathbb{R}^n$,

$$f(\boldsymbol{v}) = A\boldsymbol{v}, \tag{A.11}$$

mit einer $(m \times n)$-Matrix A linear, da $f(\boldsymbol{v}+\boldsymbol{w}) = A(\boldsymbol{v}+\boldsymbol{w}) = f(\boldsymbol{v}) + f(\boldsymbol{w})$. □

Beispiel A.5 Der Galois-Körper $GF(2^8)$, also die Menge aller 8-bit-Vektoren $x = (x_7, x_6, \ldots, x_0)$, ist ein 8-dimensionaler Vektorraum über $\mathbb{Z}_2 = \{0, 1\}$. Für $V = W = GF(2^8)$ und die oben definierte Byte-Addition $\oplus$ ist die Abbildung $f : GF(2^8) \to GF(2^8)$,

$$f(x) = Ax \bmod 2 \tag{A.12}$$

mit einer quadratischen (8×8)-Matrix A, deren Einträge nur aus Nullen und Einsen bestehen, linear. Insbesondere ist eine Permutation eine lineare Abbildung $f(x) = Ax$ mit einer Matrix A, in der in jeder Zeile und in jeder Spalte jeweils genau eine Eins steht. Eine Eins an der Stelle (i, j) in der Matrix bedeutet dann, dass das j-te bit an die i-te Stelle kommt.

Auch die Byte-Multiplikation $g : GF(2^8) \to GF(2^8)$, $g(x) = x \odot y$ für ein gegebenes y ist linear. Das folgt daraus, dass das „konsistente Rechnen“ (mathematisch durch die „Körperaxiome“ definiert) das Distributivgesetz vorschreibt, d.h. es gilt für alle $x^{(1)}, x^{(2)}, y \in GF(2^8)$, dass $(x^{(1)} \oplus x^{(2)}) \odot y = (x^{(1)} \odot y) \oplus (x^{(2)} \odot y)$. □

Proposition A.6 *Die subBytes-Transformation von AES und insbesondere ihr erster Teilschritt (A.8) sind nicht linear.*

Beweis. Zunächst zeigen wir die Nichtlinearität der subBytes-Transformation insgesamt (Abschnitt a), sodann die des ersten Teilschritts (Abschnitt b).

(a) Bezeichnen wir die subBytes-Transformation als g. Um die Behauptung zu zeigen, reicht es, ein einziges Beispielpaar $s^{(1)}$, $s^{(2)} \in GF(2^8)$ zu finden, so dass $g(s^{(1)} \oplus s^{(2)}) \neq g(s^{(1)}) \oplus g(s^{(2)})$. Betrachten wir dazu $s^{(1)} = 0\text{x}7\text{D}$ und $s^{(2)} = 0\text{x}03$. Nach der Tabelle 2.3 gilt $g(s^{(1)}) = 0\text{xFF}$ und $g(s^{(2)}) = 0\text{x}7\text{B}$, d.h.

$$g(s^{(1)}) \oplus g(s^{(2)}) = 11111111 \oplus 11111011 = 00000100 = 0\text{x}04 \tag{A.13}$$

Andererseits ist $s^{(1)} \oplus s^{(2)} = 11111101 \oplus 11 = 11111110 = 0\text{xFE}$, d.h.

$$g(s^{(1)} \oplus s^{(2)}) = 0\text{xF}3 = 11110011 \tag{A.14}$$

Mit Gleichung (A.13) gilt damit $g(s^{(1)} \oplus s^{(2)}) \neq g(s^{(1)}) \oplus g(s^{(2)})$.

(b) Nun der Beweis der Nichtlinearität des ersten Teilschritts (A.8). Mit dem Verschiebungsvektor $v = 0\text{x}63 = 11000110$ (A.9) gilt: $g(s^{(1)}) \oplus g(s^{(2)}) \oplus v = 11000010 = 0\text{xC}2$ und $g(s^{(1)} \oplus s^{(2)}) \oplus v = 000110100 = 0\text{x}34$. Verschieben wir das Byte $b = g(s)$ um v, so folgt, wegen $v \oplus v = 0$, $Aa = g(s) \oplus v$ mit der Matrix A aus (A.9). Das bedeutet aber, dass bereits

der erste Schritt der subBytes-Transformation in Gleichung (A.8) nicht linear gewesen sein muss. Q.E.D.

Wieso ist Nichtlinearität so wichtig für einen Verschlüsselungsalgorithmus? Am ehesten kann man es vielleicht geometrisch verstehen: Lineare Transformationen sind stets Drehungen und Spiegelungen von Objekten, eventuell mit Verzerrungen. Wesentliche Strukturen bleiben aber unter Umständen erkennbar. Im Extremfall könnte es passieren, dass ein Null-Byte nach einer linearen (oder einer „affinen“, d.i. linear + Verschiebung) Transformation regelmäßige Strukturen aufweist.

A.1.4 Die ShiftRows-Transformation

`ShiftRows` verschiebt die Zeilen 1 bis 3 der Zustandsmatrix S_i in Gleichung (2.3) zyklisch nach rechts, Zeile 0 wird nicht verändert. Zeile 1 wird um 1 Byte nach rechts verschoben, Zeile 2 wird um c_2 Bytes und Zeile 3 wird um c_3 Bytes, wobei die Konstanten c_i von der Blocklänge abhängen gemäß folgender Tabelle.

	$n_B = 4$	$n_B = 6$	$n_B = 8$
c_2	2	2	3
c_3	3	3	4

Die Umkehrung dieser Transformation ist einfach die gleiche zyklische Verschiebung nach links.

A.1.5 Die MixColumn-Transformation

Die Transformation MixColumn bildet jede Spalte $(s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j})$, $j = 0, \ldots, n_B$, eines Zustands S_i in (2.3), ein „4-Byte Wort“, auf ein neues 4-Byte-Wort ab durch die Matrixmultiplikation

$$\begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix} \leftarrow \begin{pmatrix} 10 & 11 & 01 & 01 \\ 01 & 10 & 11 & 01 \\ 01 & 01 & 10 & 11 \\ 11 & 01 & 01 & 10 \end{pmatrix} \odot \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix} \tag{A.15}$$

Die Matrixelemente sind Bytes in binärer Darstellung (die führenden Nullen sind weg gelassen). Die Multiplikationen und Additionen sind hier jeweils Byte-Multiplikationen $\odot$ und Byte-Additionen $\oplus$.

A.1.6 Die Schlüsselexpansion

Der Schlüssel der Länge $4n_K$ Bytes wird so erweitert, dass daraus $r+1$ Rundenschlüssel der Länge n_B generiert werden können, d.h. es werden insgesamt $(r+1)n_B$ 4-Byte-Worte für die Rundenschlüssel benötigt. Bei einer Blocklänge von 16 Byte und zwölf Runden benötigen wir also $16 \cdot 13 = 208$ Worte.

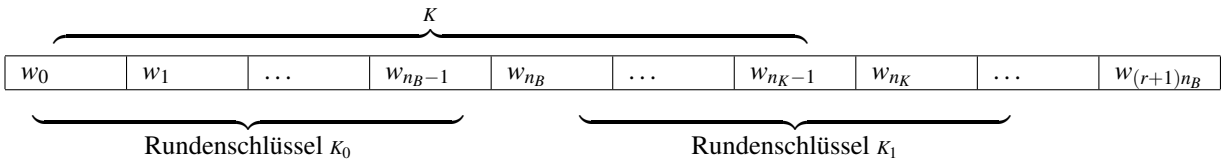
Wir bezeichnen die Worte mit w_i, wobei $i = 0, \ldots, (r+1)n_B$, und jedes Wort besteht aus 4 Bytes a_j,

$$w_i = (a_0, a_1, a_2, a_4) \qquad \text{mit } a_j \in \{0,1\}^8. \tag{A.16}$$

Die Schlüsselexpansion wird aus dem Schlüssel K die $(r+1)n_B$ benötigten Worte erzeugen. Die ersten n_K Worte sind Kopien der n_K Worte des Schlüssels K aus Gleichung (2.3),

$$w_0 = (K_{0,0}, K_{1,0}, K_{2,0}, K_{3,0}), \quad \ldots, \quad w_{n_K-1} = (K_{0,n_K-1}, K_{1,n_K-1}, K_{2,n_K-1}, K_{3,n_K-1}).$$

Jedes folgende Wort w_i ($i = n_K, \ldots, (r+1)n_B$) wird gebildet aus dem XOR des vorhergehenden Wortes w_{i-1} und des Wortes w_{i-n_K}, welches eine Schlüssellänge n_K früher beginnt. Für Worte, die bei einem Vielfachen der Schlüssellänge n_K beginnen, d.h. für w_i mit $i \,\%\, n_K = 0$, wird vor dem XOR eine nichtlineare Transformation durchgeführt, die unter anderem SubBytes aufruft. Die Zusammenstellung der Rundenschlüssel aus dem expandierten Schlüssel erfolgt wortweise sequentiell:



Die Schlüsselexpansion lautet in Pseudocode, mit dem Schlüssel K (`k`) als Eingabe dem Wort-Array `w`, in dem `w[i]` das Wort w_i darstellt:

```
schlüsselExpansion ( k ) {
   i = 0;
   while ( i < nK ) {  // kopiert die Worte des Schlüssels
      for ( j = 0; j < 4; j++ ) {
         w[i][j] = k[i][j];
      }
      i++;
   }
   while ( i < (r+1) * nB ) {
      temp = w[i - 1];   // i ist hier >= nK > 1!
      if ( i % nK == 0 )
         temp = xor( subWord( rotWord( temp ) ), rCon[i/nK];
      else if ( nK > 6 && i % nK == 4 )
         temp = subWord( temp );

      w[i] = xor( w[i-nK], temp );
      i++;
   }
   return w;
}
```

`subWord` nimmt als Eingabe ein 4-Byte-Wort, wendet auf jedes Byte die Funktion `subBytes` an und gibt das so veränderte Wort zurück; `rotWord` bekommt als Eingabe ein Wort mit den Bytes (a_0, a_1, a_2, a_3) und gibt eine zyklische Permutation (a_1, a_2, a_3, a_0) zurück; `rCon` ist die Rundenkonstante und besteht aus dem 4-Byte-Wort

$$(10_2^i \bmod 2^8, 0, 0, 0).$$

A.1.7 Eigenschaften von AES

- *Vertauschbarkeit von SubBytes und ShiftRows:* Man erhält dasselbe Ergebnis, wenn man die Reihenfolge von SubBytes und ShiftRows vertauscht und sie direkt nacheinander ablaufen (die beiden Funktionen „kommutieren“).

- *Linearität von MixColumn bzgl. XOR:* MixColumn ist linear bzgl. der Byte-Addition, d.h. insbesondere

$$\texttt{mixColumn}(S_i \oplus K_i) \;=\; \texttt{mixColumn}(S_i) \oplus \texttt{mixColumn}(K_i). \tag{A.17}$$

  Damit kann man auch die Reihenfolge von mixColumn und XOR vertauschen, wenn sie direkt nacheinander ablaufen.

A.2 Das Geburtstagsparadoxon

Das Geburtstagsparadoxon ist ein Standardproblem der Statistik. *Wie viele Menschen müssen in einem Raum sein, damit die Wahrscheinlichkeit größer als $\frac{1}{2}$ ist, dass mindestens zwei davon am selben Tag Geburtstag haben?* Die Antwort ist erstaunlich niedrig: 23.

Beweis. Sei n die Anzahl der Personen, die sich im Raum befinden. Das ist die Unbekannte, die wir bestimmen wollen. Wir nehmen zunächst an, dass ein Anwesender im Raum mit einer Wahrscheinlichkeit $\frac{1}{m}$, mit

$$m = 365,$$

an einem bestimmten Tag im Jahr Geburtstag hat. (Damit sollen statistische Tendenzen vermieden werden, z.B. wenn nur „Sommerkinder“ oder Unglücksraben mit Geburtstag 29.2.) Wir nennen den Fall, dass zwei Anwesende den gleichen Geburtstag haben, kurz „Kollision“. Dann suchen wir die Wahrscheinlichkeit

$$p(n) = \text{Wahrscheinlichkeit, dass für } n \text{ Personen mindestens eine Kollision auftritt.}$$

Um sie zu berechnen, bestimmen wir zunächst die *Gegenwahscheinlichkeit* q, dass *keine* Kollision auftritt. Es gilt

$$q = 1 - p(n). \tag{A.18}$$

Was ist nun aber q? Nummerieren wir zunächst alle Anwesenden von 1 bis n durch und bezeichnen mit q_i die Wahrscheinlichkeit, dass Person Nummer i mit keinem der vorherigen $i-1$ Personen gemeinsam Geburtstag hat. Dann gilt

$$q = q_1 \cdot q_2 \cdot \ldots \cdot q_n. \tag{A.19}$$

Offensichtlich ist $q_1 = 1$, denn die erste Person kann keinen Geburtstag mit ihrer „Vorgängerperson“ haben. Person Nummer 2 hat nun jedoch nur noch $m-1$ „freie“ Geburtstage, denn die erste hat ja bereits einen belegt. Damit ist die Wahrscheinlichkeit $q_2 = (m-1)/m$. Für Person 3 ist es damit $q_3 = (m-2)/m$, usw. Allgemein erhalten wir also

$$q_i = \frac{m-i+1}{m} \qquad \text{wenn } 1 \leqq n \leqq \min(m,n), \tag{A.20}$$

denn Person i findet schon $(i-1)$ belegte Geburtstage vor. Für $i > \min(m,n)$ ist natürlich $q_i = 0$, denn für $i > m$ gibt es keinen freien Tag mehr, und für $i > n$ keine Person mehr im Raum. Den Fall $n > m$ können wir aber ausschließen, denn bei $m = 365$ Tagen sind mehr als 365 Personen anwesend, und es *muss* mindestens eine Kollision geben (d.h. $p = 1$). Sei daher

$$n \leqq m. \tag{A.21}$$

Dann gilt wegen (A.18) und (A.19)

$$\boxed{p(n) = 1 - \frac{m(m-1)(m-2)\cdots(m-n+1)}{m^n}} \tag{A.22}$$

Für gegebenes m ist das also eine Funktion der Anzahl n der anwesenden Personen. Wir müssten nun zur Lösung der Frage $p(n) = \frac{1}{2}$ setzen und nach n umstellen. Das ist jedoch leider nicht möglich, aber man kann eine Wertetabelle für $p(n)$ erstellen und erhalten (ausschnittweise) die Tabelle

n	$p(n)$
...	...
22	0,475
23	0,507
50	0,970
...	...

Das bedeutet, bei $n = 23$ ist die Wahrscheinlichkeit einer Kollision größer als $\frac{1}{2}$. Q.E.D.

Ein ganz anderes Problem ist die folgende Frage: Wie viele Menschen müssen sich in einem Raum befinden, damit die Wahrscheinlichkeit größer als $\frac{1}{2}$ ist, dass einer davon am selben Tag Geburtstag hat wie Sie? Die Antwort ist 253.

Beweis. Sei n die Anzahl der Personen, die sich im Raum befinden. Das ist die Unbekannte, die wir bestimmen wollen. Wie im Beweis für das Geburtstagsparadoxon nehmen wir wieder an, dass ein Anwesender im Raum mit einer Wahrscheinlichkeit $\frac{1}{m}$, mit $m = 365$, an einem bestimmten Tag im Jahr Geburtstag hat. Dann suchen wir die Wahrscheinlichkeit

$$p(n) = \text{Wahrsch., dass mindestens eine der } n \text{ Personen am Tag X Geburtstag hat.}$$

Um sie zu berechnen, bestimmen wir zunächst die Gegenwahscheinlichkeit $q(n)$, dass *keiner* am Tag X Geburtstag hat. Es gilt

$$q(n) = 1 - p(n). \tag{A.23}$$

Was ist nun aber $q(n)$? Dazu überlegen wir, dass für nur eine Person $q(1) = (m-1)/m$ ist, denn ihr Geburtstag hat $m - 1 = 364$ Chancen, nicht am Tag X zu sein. Für zwei Personen gilt entsprchend $q(2) = (m-1)^2/m^2$, denn sowohl die erste als auch die zweite Person hat jeweils mit der Wahrscheinlichkeit $(m-1)/m$ nicht am Tag X Geburtstag. Damit erhalten wir

$$q(n) = \left(\frac{m-1}{m}\right)^n,$$

d.h. mit (A.23)

$$p(n) = 1 - \left(\frac{m-1}{m}\right)^n. \tag{A.24}$$

und umgestellt nach n

$$n = \frac{\ln[1-p(n)]}{\ln\frac{m-1}{m}}. \tag{A.25}$$

Für $p(n) = \frac{1}{2}$ und $m = 365$ ergibt das $n = -\ln 2/\ln(364/365) \approx 252,65$. Q.E.D.

A.3 Die Korrektheit von DSA

Die Korrektheit von DSA ist schnell bewiesen und basiert auf dem folgenden Lemma, das aus dem Kleinen Fermatschen Satz folgt.

Lemma A.7 *Seien p und q Primzahlen, so dass $q \mid (p-1)$. Sei ferner $g = z^{(p-1)/q} \bmod p$ für eine natürliche Zahl $z < p$. Dann gilt:*

$$1 = g^q \bmod p. \tag{A.26}$$

Ferner folgt für zwei Zahlen a, b mit $a = b$ mod q, dass

$$g^a = g^b \bmod p. \tag{A.27}$$

Beweis. Es gilt

$$g^q = (z^{(p-1)/q})^q = z^{p-1} = 1 \bmod p.$$

Die letzte Gleichung folgt aus dem Kleinen Satz von Fermat [6, Satz A.5]. by Fermat's Little Theorem. Das beweist Gleichung (A.26). Sei nun $a = b \bmod q$, d.h. $a = b + kq$ für eine ganze Zahl k. Dann

$$g^a = g^{b+kq} = g^b g^{kq} = g^b (g^q)^k = g^b \bmod p$$

mit Gleichung (A.26). Q.E.D.

Sei nun (m,r,s) die DSA-Signatur, die Alice an Bob gesendet hat, und sei $m',r',s')$ diejenige, die Bob empfangen hat. Dann gilt.

Satz A.8 *Sei (m,r,s) die gesendete und (m',r',s') die empfangene DSA-Signatur. Wenn $m' = m$, $r' = r$ und $s' = s$, so ergibt sich in dem DSA-Verifikation-Algorithmus $v = r'$.*

Beweis. Mit (4.2) ergeben sich w, u_1, u_2 und v. Da $P_A = g^{S_A} \bmod p$, gilt mit Lemma A.7

$$v = g^{u_1} g^{u_2} = g^{w\,\mathrm{SHA}(m)} \cdot P_A^{rw} = g^{w\,\mathrm{SHA}(m)} \cdot g^{rwS_A} = g^{w(\mathrm{SHA}(m)+rS_A)} \bmod p \bmod q, \tag{A.28}$$

Mit (4.1) ist

$$s = k^{-1}(\mathrm{SHA}(m) + rS_A) \bmod q,$$

und mit (4.2) daher $w = k(\mathrm{SHA}(m) + rS_A)^{-1} \bmod q$ oder

$$k = w(\mathrm{SHA}(m) + rS_A) \bmod q.$$

Mit Gleichung (A.27) und (A.28) folgt dann $v = (g^k \bmod p) \bmod q = r = r'$. Q.E.D.

Im Umkehrschluss bedeutet eine fehlgeschlagene Verifikation daher, dass mindestens einer der drei empfangenen Signaturparameter (m', r', s') nicht mit den gesendeten (m, r, s) übereinstimmt.

A.4 Der Erweiterte Euklidische Algorithmus

Einer der wichtigsten Algorithmen der Zahlentheorie ist der Erweiterte Euklidische Algorithmus. Er liefert eine Lösung der Gleichung

$$\mathrm{ggT}(m,n) = x_1 \cdot m + x_2 \cdot n \tag{A.29}$$

für x_0, $x_1 \in \mathbb{Z}$, m, $n \in \mathbb{N}$. Es gibt zwei Varianten des Erweiterten Euklidischen Algorithmus, eine rekursive und eine iterative.

Rekursive Variante

In Pseudocode lautet sie:

```
extendedEuclid(m, n) {
    if (n == 0) {
        return (m, 1, 0);
    } else {
        (m, x1, x2) = extendedEuclid(n, m mod n);
        return (m, x2, x1 − (m/n)x2);
    }
}
```

Hierbei ist (m, x_1, x_2) als Vektor zu betrachten. In Java lässt sich das wie folgt implementieren.

```
public static long[] extendedEuclid(long m, long n) {
   if ( n == 0 ) {
      long[] x = {m, 1, 0};
      return x;
   } else {
      long[] x = extendedEuclid(n, m % n);
      long tmp = x[1];
      x[1] = x[2];
      x[2] = tmp - (m/n) * x[1];
      return x;
   }
}
```

Iterative Variante

Die folgende iterative Version ist zwar etwas länger, kann jedoch auch mit negativen Eingaben für m und n umgehen:

```
public static long[] extendedEuclid(long m, long n) {
   long u = 0, v = 1, q, r, tmp;
   int mSign = m < 0 ? -1 : 1;
   int nSign = n < 0 ? -1 : 1;

   m = mSign * m;
   n = nSign * n;

   long x[] = {m,1,0};

   while ( n > 0 ) {
       // determine q and r such that m = qn + r:
       q = m / n; r = m % n;
       // replace:
       m = n; n = r;
       tmp = u; u = x[1] - q*u; x[1] = tmp;
       tmp = v; v = x[2] - q*v; x[2] = tmp;
   }
   x[0] = m;
   x[1] = mSign * x[1];
   x[2] = nSign * x[2];
   return x;
}
```

Für die rekursive Version müsste man die Fälle $m < 0$ und $n < 0$ *vor* dem Aufruf edes Algorithmus ntsprechend bearbeiten.

A.5 Laufzeit

Wie groß ist die Laufzeit des Algorithmus? Die Anzahl N der Iterationen ist höchstens [3, §1.8]

$$N \leqq \frac{\ln \max(m,n)}{\ln \varphi} + 1, \qquad \text{wo } \varphi = \frac{1+\sqrt{5}}{2} \approx 1,618. \tag{A.30}$$

φ ist der „Goldene Schnitt“. Er hat einige erstaunliche Eigenschaften, die zur Herleitung der Anzahl Iterationen für den Euklidischen Algorithmus ist:

$$\varphi = 1 + \frac{1}{\varphi}. \tag{A.31}$$

e Analysiert man den Aufwand etwas genauer, so ergibt sich eine Laufzeit

$$\boxed{T_{\text{Euklid}}(m,n) = O(\ln m \cdot \ln n).} \tag{A.32}$$

Das ist sehr erstaunlich, dass die Laufzeit des Euklidischen Algorithmus von derselben Größenordnung wie die Multiplikation zweier Zahlen ist [3, §1.5], obwohl er doch viel komplizierter aussieht! Es ist diese Eigenschaft des Euklidischen Algorithmus, die ihn so wertvoll insbesondere für die Kryptologie macht.

A.6 Berechnung der modularen multiplikativen Inversen

Ein wichtiges Problem der Zahlentheorie und der Kryptologie ist Berechnung der „modularen multiplikativen Inversen“. Was ist das? Gegeben seien x, $n \in \mathbb{N}$, $x < n$, die *teilerfremd* sind, d.h. $\text{ggT}(n,x) = 1$. Dann ist die *multiplikative Inverse* y von x modulo n definiert als die (eindeutige!) Lösung $y < n$ der Gleichung

$$1 = xy \bmod n. \tag{A.33}$$

Was bedeutet diese Gleichung, wenn wir sie „ausschreiben“? Allgemein besagt $1 = m \bmod n$ doch, dass die Zahl $(m-1)$ durch n teilbar sein muss, d.h. es gibt ein Vielfaches $k \in \mathbb{Z}$ von n, so dass gilt

$$1 = m \bmod n \qquad \Longleftrightarrow \qquad 1 - m = kn.$$

Auf Gleichung (A.33) angewendet bedeutet das

$$1 = xy \bmod n \qquad \Longleftrightarrow \qquad 1 = xy + kn \tag{A.34}$$

für ein $k \in \mathbb{Z}$. Auf den ersten Blick ist uns nicht viel geholfen: Wenn Sie z.B. die Gleichung $13y + 17k = 1$ lösen möchten, so könnte das auf den ersten Blick eine langwierige Aufgabe sein.

Da allerdings $\text{ggT}(x,n) = 1$ gilt, ist diese Gleichung mit $x_1 = y$ und $x_2 = k$ ein Spezialfall der Gleichung (A.29)! Der Euklidische Algorithmus löst sie uns also automatisch und in kurzer Laufzeit:

$$\boxed{y = x_1,} \quad k = x_2,, \qquad \text{wo} \quad (x_0, x_1, x_2) = \text{extendedEuclid}(m,n).$$

(x_0 ergibt 1).

Literaturverzeichnis

[1] ASPECT, Alain ; GRANGIER, Philippe ; ROGER, Gérard: 'Experimental realization of Einstein-Podolsky-Rosen-Bohm gedankenexperiment: a new violation of Bell's inequalities'. In: *Phys. Rev. Lett* 49 (1982), S. 91–94. – http://link.aps.org/abstract/PRL/v49/p91

[2] BOUWMEESTER, Dik ; PAN, Jan-Wei ; MATTLE, Klaus ; EIBL, Manfred ; WEINFURTER, Harald ; ZEILINGER, Anton: 'Experimental quantum teleportation'. In: *Nature* 390 (1997), S. 575

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

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

[5] EINSTEIN, Albert ; PODOLSKY, B. ; ROSEN, Nathan: 'Can quantum mechanical description of physics reality be considered complete?'. In: *Phys. Rev.* 47 (1935), S. 777–780

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

[7] GOSWAMI, Amit: *Quantum Mechanics*. 2nd. Dubuque, IA : Wm. C. Brown publishers, 1997

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

[9] MACKAY, David: *Information Theory, Inference, and Learning Algorithms*. Cambridge : Cambridge University Press, 2003

[10] NIELSEN, Michael A. ; CHUANG, Isaac L.: *Quantum Computation and Quantum Information*. Cambridge : Cambridge University Press, 2000

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

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

[13] VRIES, Andreas de: 'The ray attack on RSA cryptosystems'. In: MUNO, Ralf W. (Hrsg.): *Jahresschrift der Bochumer Interdisziplinären Gesellschaft eV 2002*. Stuttgart : ibidem-Verlag, 2003, S. 11–38. – http://arxiv.org/abs/cs.CR/0307029

[14] ZEIDLER, Eberhard: *Applied Functional Analysis. Applications to Mathematical Physics*. New York : Springer-Verlag, 1995

[15] ZEIDLER, Eberhard (Hrsg.): *Teubner Taschenbuch der Mathematik. Teil 1*. Leipzig : B. G. Teubner, 1996

Index