

Vorgelegt an der Universität Karlsruhe (TH)  
Institut für Programmstrukturen und Datenorganisation (IPD)

## **Grundlagen und Einsatz von Kryptographie**

Seminar „Sicherheit und technischer Datenschutz in Informationssystemen“  
Lehrstuhl Prof. Böhm

von  
Marc Mültin  
8. Semester Informatik  
Karlsruhe  
26. Juni 2006



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>5</b>
<b>2</b>	<b>Was ist Kryptographie</b>	<b>7</b>
2.1	Begriffsbildungen . . . . .	7
2.2	Funktionsweise der Kryptographie . . . . .	8
<b>3</b>	<b>Symmetrische Verschlüsselungssysteme</b>	<b>9</b>
3.1	Transpositionschiffren . . . . .	10
3.2	Substitutionschiffren . . . . .	10
3.3	One-Time-Pads . . . . .	14
<b>4</b>	<b>Block- und Stromchiffren</b>	<b>17</b>
4.1	Blockchiffren . . . . .	17
4.2	Stromchiffren . . . . .	19
4.3	Betriebsmodi von symmetrischen Blockchiffren . . . . .	21
4.4	DES . . . . .	24
4.5	AES . . . . .	27
4.5.1	Die Entstehungsgeschichte . . . . .	27
4.5.2	Die Arbeitsweise . . . . .	28
<b>5</b>	<b>Asymmetrische Kryptographie</b>	<b>35</b>
5.1	Ablauf einer asymmetrischen Verschlüsselung . . . . .	36
5.2	Was ist besser? . . . . .	37
5.3	Diffie-Hellman . . . . .	38
5.3.1	Sicherheit . . . . .	39
5.4	RSA . . . . .	40
5.4.1	Vorbereitung . . . . .	41
5.4.2	Ablauf einer Verschlüsselung . . . . .	42
5.4.3	Sicherheit . . . . .	45
<b>6</b>	<b>Hybride Verschlüsselungssysteme</b>	<b>47</b>
6.1	PGP - Pretty Good Privacy . . . . .	48
6.1.1	Verwendete Algorithmen . . . . .	49
6.1.2	PGP-Datenkomprimierungsroutinen und verwendete Zufallszahlen	50
6.1.3	Der Nachrichtenkernel . . . . .	51
6.1.4	Weiteres Wissenswertes . . . . .	51



# 1 Einleitung

Schon seit Jahrtausenden verlassen sich Herrscher und Generäle auf schnelle und sichere Nachrichtenwege, um ihre Länder und Armeen zu führen. Und seit jeher wissen sie, welche schwer wiegenden Folgen es haben könnte, sollten ihre Botschaften in die falschen Hände geraten. Dann wären den rivalisierenden Staaten oder gegnerischen Streitkräften wohlgehütete Geheimnisse und entscheidende Informationen preisgegeben. Die Gefahr, dass ein Gegner solche wichtige Nachrichten abfangen könnte, war und ist Ansporn für die Entwicklung der Verschlüsselungsverfahren.

Aber auch mit dem Beginn des Informationszeitalters, das durch die Erfindung des Telefons eingeläutet wurde, besteht in unserer Gesellschaft ein wachsendes Bedürfnis, den Schutz der Privatsphäre in die eigenen Hände zu nehmen. Sei es, weil eine Firma sich vor ausländischer (oder sogar inländischer) Spionage schützen will, um keine Interna und innovativen Ideen preiszugeben, oder um den Kontakt zu einem politischen Dissidenten in einem autoritären Staat zu verschleiern, oder einfach nur, weil man den Kontakt zu einer geheimen Liebschaft vor seiner Frau vertuschen will, indem man seine E-Mails verschlüsselt: Die Gründe für die Bedürfnisse einer verschlüsselten Kommunikation sind unterschiedlichster Natur.

In meiner Ausarbeitung werde ich auf die Entwicklung und die verschiedenen Arten der kryptographischen Verfahren eingehen, den ein oder anderen Algorithmus näher beleuchten, und versuchen, dem Leser insgesamt einen guten Überblick über die wichtigsten Aspekte der Kryptographie zu geben. Dort, wo aufgrund der Beschränkung des Umfangs einer Seminararbeit nicht zu sehr ins Detail gegangen werden kann, wird der Leser auf die entsprechende weiterführende Literatur verwiesen.

Abschnitt 2, „Was ist Kryptographie“, erläutert die grundlegenden Begriffe Kryptographie, Kryptoanalyse und Kryptologie.

Abschnitt 3, „Symmetrische Verschlüsselungssysteme“, gibt einen kurzen geschichtlichen Abriss über (einen Ausschnitt der) Entwicklung der Kryptographie in den letzten Jahrhunderten, behandelt Transpositionschiffren anhand des rail-fence-Algorithmus, Substitutionsschiffren anhand der Caesar- und Vigenère-Verschlüsselung und geht auf den wichtigen Aspekt des One-Time-Pads im Zusammenhang mit sicheren Schlüsseln ein.

Abschnitt 4, „Block- und Stromchiffren“, macht einen Sprung ins heutige Informationszeitalter und zeigt die Verwendung kryptographischer Verfahren unter Zuhilfenahme von Computern. Nachdem auf die generelle Funktionsweise von Block- und Stromchiffren eingegangen wird, werden noch die diversen Betriebsmodi symmetrischer Blockchiffren

erläutert sowie deren Anwendung in aktuell verwendeten Verschlüsselungsalgorithmen wie DES (bzw. Triple-DES) und AES.

Abschnitt 5, „Asymmetrische Verschlüsselungssysteme“, stellt eine weitere kryptographische Technologie vor, die keine Konkurrenz zur bisher vorgestellten symmetrischen Kryptographie darstellen soll, sondern vielmehr dazu konzipiert wurde, die logistische Schwachstelle des Schlüsselverteilsproblems zu beheben.

Abschnitt 6, „Hybride Verschlüsselungsverfahren“, zeigt schließlich, wie sich die symmetrische und asymmetrische Kryptographie auf elegante Weise miteinander kombinieren lässt, um verschlüsselte Nachrichten auszutauschen. Als aktuelles Anwendungsbeispiel wird auf die Funktionsweise von PGP näher eingegangen.

# 2 Was ist Kryptographie

## 2.1 Begriffsbildungen

Der Begriff *Kryptographie* stammt aus dem Griechischen und setzt sich aus den Wörtern *Kryptos* (griech. *verborgen*) und *Graphein* (griech. *schreiben*) zusammen. Kryptographie ist demnach die Kunst des verborgenen Schreibens, und das Ziel besteht darin, den Sinn einer Botschaft durch Verschlüsselung zu verbergen, nicht aber deren Existenz.

Die *Steganographie* (*Steganos* ist griechisch für *bedeckt*), welche oft mit kryptographischen Verfahren kombiniert wird, hat hingegen das Verbergen der *Existenz* einer Nachricht zum Ziel. Ein klassischer Vertreter hierfür ist die Geheimtinte, die sich durch Erhitzen erst verfärbt und sichtbar wird. Aktuellere Anwendungen sind beispielsweise Wasserzeichen in Bildern, Banknoten oder Musikdateien.

Das Ziel der *Kryptoanalyse* ist komplementär, d.h. es geht darum, den Sinn einer verschlüsselten Nachricht zu erkennen. Zur klassischen Kryptoanalyse gehören analytisches Denken, die Anwendung mathematischer Verfahren und das Auffinden von Strukturen.

*Kryptologie* ist der Oberbegriff, welcher die Begriffe Kryptographie und Kryptoanalyse zusammenfasst.

### Ziele der Informationssicherheit

Bei der Übertragung jedweder Information über unsichere Kommunikationskanäle - wie beispielsweise das Internet -, gilt es diverse Sicherheitsrichtlinien zu erfüllen. Die Kryptographie verfolgt insbesondere folgende vier Sicherheitsziele:

- *Vertraulichkeit*: (Confidentiality, Secrecy, Privacy)  
Nur berechtigte Personen dürfen Zugriff auf die übertragene Information haben.
- *Authentizität*: (Authenticity) Die Herkunft (Autor) einer übertragenen Information kann korrekt identifiziert werden.
- *Integrität*: (Integrity) Unberechtigte Personen dürfen eine übertragene Information nicht verändern können. Sofern dies doch geschieht, muss dies dem Empfänger sofort ersichtlich sein.
- *Nicht-Abstreitbarkeit*: (Non-repudiation) Weder dem Sender noch dem Empfänger

einer übertragenen Information darf es möglich sein, die Übertragung abzustreiten. Ist insbesondere für Vertragsfragen von Interesse.

## 2.2 Funktionsweise der Kryptographie

Ein *Verschlüsselungsalgorithmus* oder Chiffriercode ist eine mathematische Funktion zur Ver- und Entschlüsselung. Dieser Algorithmus wirkt in Kombination mit einem *Schlüssel*, beispielsweise einem Wort, einer Zahl oder Wortgruppe zur Verschlüsselung des Klartexts. Die Sicherheit der verschlüsselten Daten ist von den folgenden zwei Größen abhängig: der Stärke des Verschlüsselungsalgorithmus und der Geheimhaltung des Schlüssels.

Der Holländer Auguste Kerckhoff (1835 - 1903) formulierte ein Prinzip, welches ein Designkriterium für moderne kryptographische Verfahren darstellt. Es besagt, dass die *Sicherheit eines kryptographischen Algorithmus nur auf der Geheimhaltung des Schlüssels* beruhen soll und *nicht auf der Geheimhaltung des Algorithmus* selbst. Der Vorteil der Anwendung des Kerckhoffs-Prinzips besteht in erster Linie darin, dass sich viele Experten eine Meinung über die Qualität eines Verfahrens bilden können und es zu einer fundierten Expertenmeinung über die Qualität eines Verfahrens führt.

Der Verschlüsselungsalgorithmus mit allen verfügbaren Schlüsseln und allen Protokollen, durch die er funktioniert, bilden ein *Verschlüsselungssystem*, beispielsweise das Verschlüsselungssystem PGP (Pretty Good Privacy).

Die formale Definition eines Verschlüsselungssystems lautet:

Ein Verschlüsselungssystem ist ein 5-Tupel  $(\mathcal{E}, \mathcal{D}, \mathcal{M}, \mathcal{K}, \mathcal{C})$ , wobei

- $\mathcal{M}$  eine Menge von Klartexten,
- $\mathcal{K}$  eine Menge von Schlüsseln,
- $\mathcal{C}$  die Menge von chiffrierten Texten,
- $\{\mathcal{E}_e : e \in \mathcal{K}\} : \mathcal{M} \times \mathcal{K} \mapsto \mathcal{C}$  die Menge der Verschlüsselungsfunktionen und
- $\{\mathcal{D}_d : d \in \mathcal{K}\} : \mathcal{C} \times \mathcal{K} \mapsto \mathcal{M}$  die Menge der Entschlüsselungsfunktionen

bezeichnet.

Die Entschlüsselungsfunktion  $\mathcal{D}$  ist die Umkehrfunktion der Verschlüsselungsfunktion. Es muss also gelten:  $\mathcal{D}(d, \mathcal{E}(e, \mathcal{M})) = \mathcal{M}$

### 3 Symmetrische Verschlüsselungssysteme

Die Beziehung zwischen den Schlüsseln  $e$  und  $d$  ist das entscheidende Kriterium, aufgrund dessen zwischen symmetrischer und asymmetrischer Verschlüsselung unterschieden wird. Ein Verschlüsselungssystem wird *symmetrisch* genannt, wenn es für jedes zulässige Schlüsselpaar  $(e, d)$  „einfach“ ist,  $d$  aus  $e$  und  $e$  aus  $d$  abzuleiten. In einem solchen Fall steckt in beiden Schlüsseln  $e$  und  $d$  im wesentlichen die gleiche Information, es besteht also eine Symmetrie zwischen Ver- und Entschlüsselung. Im einfachsten Fall eines symmetrischen Verschlüsselungssystems haben wir  $e = d$ , wobei ein solcher Schlüssel dann als  $k$  bezeichnet wird. Im Folgenden wird immer von einer solchen Situation ausgegangen.

Die gesamte klassische Kryptographie beruht auf diesem Grundsatz. Damit ein übertragener Chiffretext<sup>1</sup> von einem Angreifer nicht verstanden werden kann, muss der Schlüssel  $k$  (oder im allgemeinen Fall beide Schlüssel  $e$  und  $d$ ) geheim gehalten werden. Man bezeichnet  $k$  deshalb auch als geheimen Schlüssel (Secret Key, Symmetric Key, Single Key). Da sowohl der Sender als auch der Empfänger im Besitz dieses geheimen Schlüssels sein müssen, muss dieser im Vorfeld der eigentlichen Kommunikation vereinbart werden. Dies nennt man das *Schlüsselverteilproblem*, auf das im Abschnitt zu Asymmetrischer Kryptographie genauer eingegangen wird.

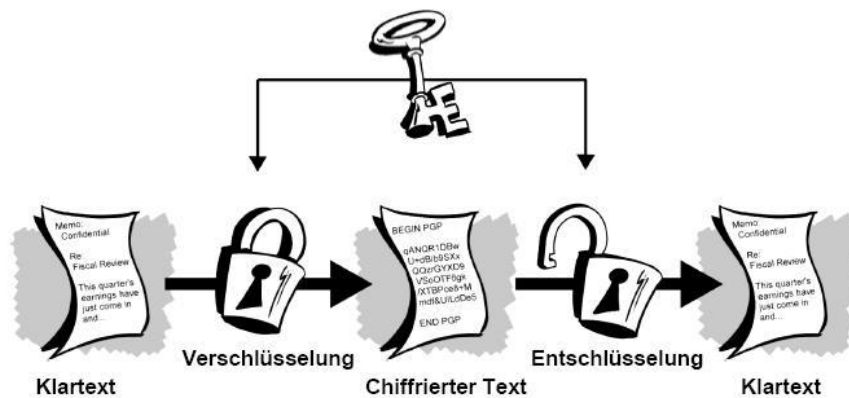


Abbildung 3.1: Klassische (konventionelle) Verschlüsselung

Es existieren zwei Grundtypen von Verschlüsselungsalgorithmen:

- Transpositionschiffren

<sup>1</sup>Im Folgenden werden „verschlüsselter Text“, „Chiffretext“ und „Chiffre“ als Synonyme verwendet.

– Substitutionschiffren

### 3.1 Transpositionschiffren

Eine *Transpositionschiffre* ordnet die Buchstaben des Klartextes neu an, um somit den verschlüsselten Text zu formen. Die Buchstaben werden jedoch nicht verändert.

Ein Beispiel hierfür ist der *rail fence* Algorithmus. Die Chiffre wird gebildet, indem der Klartext in zwei Reihen geschrieben wird, erst nach unten, dann quer laufend. Zusammengesetzt wird die Chiffre dann durch das Lesen zuerst der ersten, dann der zweiten Reihe von links nach rechts.

Beispiel: Der Klartext „HELLO WORLD“ würde geschrieben als:

HLOOL  
ELWRD

Dies resultiert dann in der Chiffre „HLOOLELWRD“.

Mathematisch ist der Schlüssel zu einer Transpositionschiffre eine Permutationsfunktion. Eine Entschlüsselungsattacke greift auf die Anagramm-Technik zurück, welche Tabellen bestehend aus n-gram Häufigkeiten benutzt, um übliche n-grams zu identifizieren. Ein n-gram ist eine Zusammensetzung aus n-Buchstaben, so wie die Menge der 2-gram von Straße aus st, tr, ra, aß, ße besteht. Die Tabellen aus n-grams müssen natürlich zur entsprechenden Sprache des Klartextes passen.

Der Kryptoanalytiker ordnet die Buchstaben in solcher Weise um, dass die Buchstaben im verschlüsselten Text n-grams mit der höchsten Häufigkeit bilden. Dieser Prozess wird wiederholt, wobei verschiedene n-grams benutzt werden, bis das Transpositionsmuster gefunden wurde.

### 3.2 Substitutionschiffren

Bei *Substitutionschiffren* muss wieder zwischen *mono-alphabetischen* und *poly-alphabetischen* unterschieden werden.

Eine Substitutionschiffre tauscht Informationsbestandteile gegeneinander aus. Dies geschieht häufig durch Vertauschen einzelner Buchstaben im Alphabet. Der wohl bekannteste Vertreter hierfür ist die *Caesar-Verschiebung*.

Für den geheimen Verkehr von Botschaften innerhalb des römischen Reiches wurde am Hof von Julius Caesar die erste (mono-alphabetische) Substitutions-Chiffrierung entwickelt und angewandt. Der „Caesar“ beruht auf einem Geheimtextalphabet, das um eine bestimmte Stellenzahl gegenüber dem Klartextalphabet verschoben ist, in diesem Falle um drei Stellen (modulo 26).

Klartextalphabet:    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  
 Geheimentalphabet: DEF GHI JKLMNOP QRSTUVWXYZABC

Klartext:        v e n i    v i d i    v i c i  
 Geheimentext: YHQL    YLGL    YLFL

Der Schlüssel bei der Caesar-Verschiebung ist offensichtlich die Zahl der Buchstaben, um das das Geheimentalphabet gegenüber das Klartextalphabet verschoben wurde. Beschränkt man sich auf das Alphabet, so können 25 mögliche Geheimschriften erzeugt werden. Die Chiffre wäre somit relativ einfach zu knacken, man müsste einfach nur maximal 25 Möglichkeiten ausprobieren und wäre nach verhältnismäßigem Aufwand am Ziel. Beschränkt man sich nicht darauf, das Alphabet zu verschieben, sondern als Geheimentalphabet beliebige Umstellungen des Klartextalphabets zulässt, dann kann eine sehr viel größere Zahl unterschiedlicher Geheimentexte erzeugt werden, nämlich  $26! = 400\,000\,000\,000\,000\,000\,000\,000\,000$ .

Aber auch diese scheinbar gewaltige Menge an Möglichkeiten fällt einer wohlbekanntesten Angriffstechnik zum Opfer: der *statistischen Häufigkeitsanalyse*.

Hierbei werden die einzelnen Buchstaben gezählt und ihre Häufigkeit notiert, meist in Prozent, also relativ zur Gesamtzahl der Buchstaben (Buchstabenhäufigkeit). Nun kann aufgrund der spezifischen Häufigkeit spezieller Buchstaben in einer Sprache – das E beispielsweise kommt in der deutschen Sprache mit rund 17% mit Abstand am häufigsten vor – auf das verwendete Alphabet geschlossen werden. Kommt in einer Nachricht also beispielsweise der ansonsten recht seltene Buchstabe Q mit etwa 17% vor, so liegt der Schluss nahe, dass Q in dieser Verschlüsselung für das E steht.

Generell lassen sich fünf verschiedene Arten von kryptoanalytischen Angriffen unterscheiden:

- „*Ciphertext-Only*“: Der Kryptoanalytiker verfügt über den Chiffretext mehrerer Nachrichten, die mit demselben Verschlüsselungsalgorithmus chiffriert wurden. Ziel ist es, den Klartext möglichst vieler Nachrichten wiederzuerstellen oder - besser noch - den oder die zur Chiffrierung der Nachrichten verwendeten Schlüssel abzuleiten, um damit weitere mit denselben Schlüsseln chiffrierte Nachrichten zu entschlüsseln.
- „*Known-Plaintext*“: Der Kryptoanalytiker besitzt Zugang zum Chiffretext diverser Nachrichten wie auch die dazugehörigen Klartexte. Ziel ist es, den oder die zur Verschlüsselung der Nachrichten verwendeten Schlüssel herauszubekommen oder einen Algorithmus zu finden, mit dem weitere Nachrichten entschlüsselt werden können, die mit denselben Schlüsseln chiffriert wurden.
- „*Chosen-Plaintext*“: Der Kryptoanalytiker verfügt nicht nur über den Chiffre- und Klartext verschiedener Nachrichten, sondern kann darüber hinaus den zu verschlüs-

selnden Klartext selbst festlegen. Damit bieten sich ihm bessere Voraussetzungen als beim known-plaintext-Angriff, da er gezielt spezielle Klartextblöcke zur Verschlüsselung auswählen kann, die unter Umständen zu weitergehenden Informationen über den Schlüssel führen. Ziel ist es, den oder die zur Verschlüsselung der Nachrichten verwendeten Schlüssel herauszubekommen oder einen Algorithmus zu finden, mit dem weitere Nachrichten entschlüsselt werden können, die mit denselben Schlüsseln chiffriert wurden.

- „*Chosen-Ciphertext*“: Der Kryptoanalytiker kann verschiedene Chiffretexte zur Entschlüsselung auswählen und hat Zugriff auf den entschlüsselten Klartext. Beispielsweise könnte er Zugang zu einem einbruchssicheren Apparat besitzen, der automatische Entschlüsselung durchführt. Ziel ist es, den Schlüssel herauszufinden.
- „*Brute-Force-Angriff*“: Sämtliche mögliche Schlüssel werden einzeln ausprobiert. Um sich davor zu schützen, sollte der Schlüsselraum möglichst groß sein.

Ein gutes Verschlüsselungssystem schützt gegen alle fünf Arten von Angriffen. Es existieren noch weitere Varianten wie z.B. *Adaptive-Chosen-Plaintext* und *Kryptoanalyse mit Gewalt*, auf die hier aber nicht näher eingegangen wird.

Nachdem nach der Entdeckung der Häufigkeitsanalyse die Kryptoanalytiker lange Zeit die besseren Karten in der Hand hatten als die Kryptographen, gelang es im 16. Jahrhundert dem französischen Diplomat Blaise de Vigenère ein nach ihm benanntes Verschlüsselungsverfahren zu entwickeln, gegen welches die klassische Häufigkeitsanalyse nicht mehr funktionierte: die *Vigenère-Verschlüsselung*. Das Verfahren beruht auf einer *poly-alphabetischen Substitution*. Polyalphabetische Ersetzungschiffren bezeichnen in der Kryptographie Formen der Textverschlüsselung, bei der einem Buchstaben/Zeichen jeweils ein anderer Buchstabe/Zeichen zugeordnet wird. Im Gegensatz zur monoalphabetischen Substitution werden für die Zeichen eines Klartextes *mehrere Geheimentalphabeten* verwendet.

Die Stärke der Vigenère-Verschlüsselung beruht darauf, dass nicht nur ein, sondern 26 verschiedene Geheimalphabete benutzt werden, um eine Botschaft zu verschlüsseln. Die Idee ist, dass je nach Position eines Zeichens im Klartext ein anderer Substitutionsschlüssel (Anzahl Positionsverschiebungen im Alphabet) verwendet wird. Für die Chiffrierung und Dechiffrierung wird ein sogenanntes Vigenère-Quadrat benötigt. Um die Botschaft zu entschlüsseln, muss der Empfänger wissen, welche Zeile des Vigenère-Quadrats für den jeweiligen Buchstaben benutzt wurde. Deshalb müssen Sender und Empfänger zuvor abstimmen, nach welcher Regel zwischen den Zeilen hin und her gewechselt wird. Diese Übereinkunft legen sie anhand eines Schlüsselwortes fest, das wiederholt angewandt wird. Je länger das Schlüsselwort, desto sicherer wird das Verfahren. Bei einem Schlüsselwort, welches aus 20 einbuchstabigen Schlüsseln besteht, würde jeder 20. Buchstabe mit demselben Schlüssel chiffriert. Dies wird als *Periode* der Chiffrierung bezeichnet.

Konkret wird ein Text zusammen mit dem Schlüsselwort mit Hilfe der in Abbildung 3.2 gezeigten Verschlüsselungstabelle (Vigenère-Quadrat) verschlüsselt, indem das aktuelle Zeichen im Klartext als Zeilenindex und das aktuelle Zeichen im Schlüsselwort als Spal-

tenindex dient, und so in der Tabelle das entsprechende Zeichen abgelesen werden kann.

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

Abbildung 3.2: Verschlüsselungstabelle von Blaise de Vigenère

Als Beispiel verschlüsseln wir die Meldung **Truppenabzug nach Osten** anhand des Schlüsselworts LICHT.

Schlüsselwort: LICHTLICHTLICHTLICHTLICHTL  
 Klartext: truppenabzug nach ostent  
 Geheimtext: EZWWIPVCISFOEHVSWUAXY

Wie bereits erwähnt, lässt sich gegen die Vigenère-Verschlüsselung mit der Häufigkeitsanalyse nichts ausrichten, weil mehrere Geheimentalphabete für eine Nachricht verwendet werden und somit ein und derselbe Buchstabe durch verschiedene Buchstaben verschlüsselt werden kann. Hinzu kommt, dass sich jedes x-beliebige Wort als Schlüssel verwenden lässt und das Verfahren daher eine enorme Zahl an Schlüsseln bietet. Ein Brute-Force-Angriff würde hier auch nicht zum Erfolg führen.

Vigenère veröffentlichte sein Werk 1586 mit dem *Traicté de Chiffres*, einer Abhandlung über die Geheimschriften.

Die polyalphabetische Verschlüsselung war während fast drei Jahrhunderten nicht zu brechen und galt als absolut sicher. Erst im 19. Jahrhundert gelang dem englischen Erfinder Charles Babbage die Kryptoanalyse eines so verschlüsselten Textes.

Das Verfahren hat nämlich eine Schwäche: Schlüsselwörter, die zum Text relativ kurz

sind, bieten keine Sicherheit. Eine Korrelation des Chiffretextes mit sich selbst, jeweils um eine Stelle verschoben, liefert die Länge des Schlüssels. Ist die Schlüssellänge (Periode)  $n$  bekannt, reduziert sich die Kryptoanalyse der Vigenère-Verschlüsselung auf die der Caesar-Verschlüsselung: alle ersten, zweiten,...,  $n$ -ten Buchstaben einer Periode gehören jeweils zur selben Caesar-Verschlüsselung und eine Häufigkeitsanalyse verrät die Buchstabenzuordnung.

Bei einem Text, der nur aus der Wiederholung eines Zeichens besteht, zeigt sich die Periode unmittelbar im Geheimtext. Ein normaler Text weist ausreichend Redundanzen auf, so dass ab einer gewissen Länge des Textes im Vergleich zum Schlüssel auch hier die Periode abgeleitet werden kann (Kasiski-Test<sup>2</sup>). Einzig ein Text aus statistisch gleich verteilten Buchstaben wäre einem Ciphertext-Only-Angriff nicht ohne Weiteres zugänglich.

Beispiel:

Text:	e e e e e e e e e e e e e e	
Schlüssel:	AKEYAKEYAKEYA	
Geheimtext:	EOICEOICEOICE	
Korrelation 1:	EOICEOICEOICE	Keine Korrelation mit Geheimtext
Korrelation 2:	EOICEOICEOICE	Keine Korrelation mit Geheimtext
Korrelation 3:	EOICEOICEOICE	Keine Korrelation mit Geheimtext
Korrelation 4:	EOICEOICEOICE	Korrelation mit Geheimtext!

Auf diese Weise bekommt man recht schnell die Schlüssellänge des verschlüsselten Textes heraus. Jetzt muss nur noch der Geheimtext spaltenweise zerlegt werden. Die Spalten, welche mit dem selben Buchstaben verschlüsselt wurden (für welche also ein und dasselbe Geheimalphabet genutzt wurde), werden zusammengefasst. Die entsprechende Alphabetverschiebung der einzelnen Teiltexthe löst man nun mittels Häufigkeitsanalyse.

### 3.3 One-Time-Pads

Die bisher vorgestellten Verfahren zur Verschlüsselung hatten alle zusammen eine gemeinsame Schwäche: den (zu kurzen) Schlüssel. Es existiert ein perfektes Verschlüsselungskonzept. Es nennt sich *One-Time-Pad (Einmalblock)* und wurde 1917 von Major Joseph Mauborgne und Gilbert Vernam von AT&T erfunden.

Klassischerweise besteht ein One-Time-Pad aus einer sehr langen Folge von *zufällig* gewählten Schlüsselbuchstaben, und zwar ist die Folge *genauso lange wie die Klartext- bzw.*

<sup>2</sup>Man versuchte anfangs, Babbages Erkenntnis gegenüber den Feinden Geheim zu halten, um den dadurch erlangten Vorsprung nicht unnötig preiszugeben. So kam es, dass Babbages Entdeckung erst im 20. Jh. bekannt wurde, während das gleiche Verfahren von Willhelm Kasinski wiederentdeckt und 1863 veröffentlicht wurde. Man spricht deshalb heute noch vom Kasinski-Test, obwohl dieser historisch gesehen nicht der Erfinder ist.

*Nutzdatenfolge.* Der Sender chiffriert mit jedem Schlüsselbuchstaben genau ein Klartextzeichen durch Addition modulo 26 (bzw. in der Welt der Bits und Bytes durch Verknüpfung eines One-Time-Pads mit den Nutzdaten via XOR). Dies ist die einzig *beweisbar sichere Verschlüsselungsmethode*, da für den Kryptoanalytiker alle Klartextfolgen gleich wahrscheinlich sind. Jeder One-Time-Pad darf nur ein einziges Mal verwendet und muss nach seiner Verwendung vernichtet werden.

Lautet die Nachricht beispielsweise  
 ONETIMEPAD  
 und die Schlüsselsequenz auf dem Block  
 TBFRRGFRFM  
 dann ist der Chiffretext  
 IPKLPSFHGQ  
 da

$$\begin{aligned} O + T \bmod 26 &= I \\ N + B \bmod 26 &= P \\ E + F \bmod 26 &= K \\ \text{usw.} \end{aligned}$$

Damit das Verfahren eingesetzt werden kann, müssen die Schlüssel zuvor über einen sicheren Kanal ausgetauscht werden. Dies mindert den praktischen Wert des Verfahrens. Ein weiterer Nachteil von One-Time-Pads besteht in dem Umstand, dass jeweils ein eigener Schlüssel für jede Sender-Empfänger-Beziehung verwendet werden muss. Damit steigt die Anzahl der Schlüssel quadratisch mit der Anzahl der Teilnehmer, wenn jeder mit jedem kommunizieren soll. One-Time-Pads sind in der Praxis nur aufwendig anwendbar und werden daher nur für außerordentlich vertrauliche Nachrichten in Organisationen mit straff organisierten Schlüsselverteilungsmechanismen genutzt. Ein Anwendungsfall hierfür wäre z.B. die Kommunikation, die über das Rote Telefon des US-Präsidenten abgewickelt wird.



## 4 Block- und Stromchiffren

Nachdem nun ein kurzer geschichtlicher Überblick über die Verschlüsselungsmethoden der letzten Jahrhunderte gegeben wurde, um den Grundbaustein für das Verständnis der Entwicklung in der Kryptographie zu legen, soll im Folgenden auf die in der heutigen Computertechnik angewendeten Verfahren eingegangen werden.

Für tiefergehende Informationen zur Geschichte der Kryptographie von den Anfängen bei den Hyroglyphen bis zur Neuzeit gibt [DK96] eine umfangreiche Übersicht.

Bei der computergestützten Verschlüsselung gibt es zwei Klassen von symmetrischen Chiffren, die sich durch ihre Eingabe unterscheiden: *Blockchiffren* und *Stromchiffren*. Blockchiffren verschlüsseln ganze Blöcke fester Länge in einem Durchgang, während Stromchiffren in jedem Arbeitsschritt ein einzelnes Zeichen verschlüsseln. Diese beiden Klassen werden im Folgenden vorgestellt; zusätzlich werden noch verschiedene Betriebsmodi für symmetrische Chiffren vorgestellt.

### 4.1 Blockchiffren

Symmetrische Verschlüsselungsalgorithmen sind meist Blockchiffren. Sie teilen die zu verschlüsselnden Daten in gleichgroße Blöcke (meist 64 oder 128 Bit) auf, die anschließend unabhängig voneinander verschlüsselt werden. Erst der Einsatz in einem der erwähnten Betriebsmodi erzeugt unter Umständen eine Abhängigkeit zwischen verschiedenen Blöcken, z.B. zum Integritätsschutz. Da die Länge des zu verschlüsselnden Textes meistens kein Vielfaches der Blockgröße ist, wird üblicherweise der letzte Block auf die fest vorgegebene Blockgröße aufgefüllt (sogenanntes *Padding*). Wichtige Größen einer Blockchiffre sind die *Blocklänge*  $n$  und die verwendete *Schlüssellänge*  $l$  (jeweils in Bit angegeben). Vereinfacht gesagt arbeitet eine symmetrische Blockchiffre wie folgt.

1. Es wird ein geheimer Schlüssel gewählt.
2. Der Klartext wird in viele gleichgroße Blöcke unterteilt.
3. Jeder dieser Blöcke wird nach einem Algorithmus transformiert und mit dem Schlüssel so kombiniert, dass mittels dem neu entstandenen Block ohne Kenntnis des Schlüssels nicht mehr auf den Klartext geschlossen werden kann. Die entstandenen verschlüsselten Blöcke bilden den Geheimtext, welcher ebenso lang ist wie der Klartext.

4. Mit dem inversen Algorithmus und ein und demselben Schlüssel kann aus dem Geheimtext wieder blockweise der Klartext errechnet werden.

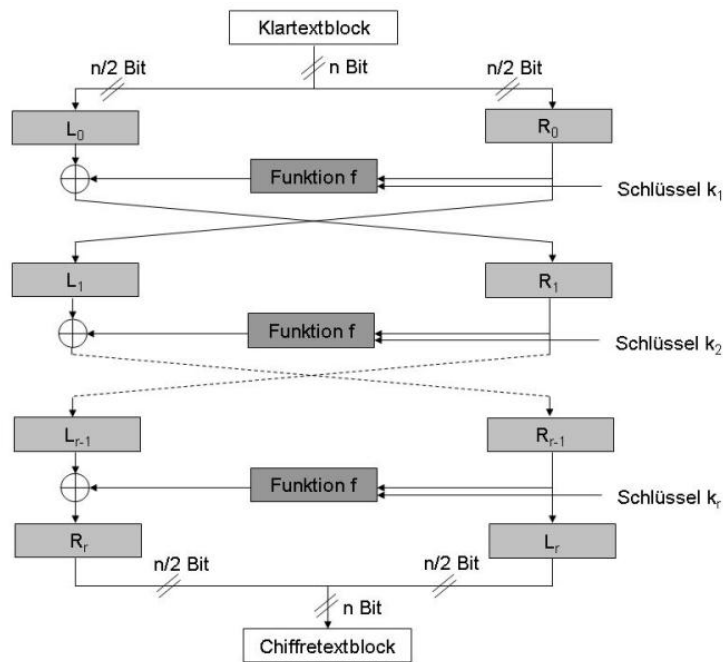


Abbildung 4.1: Feistel-Netzwerk

Abbildung 4.1 zeigt die Grundlage vieler gebräuchlicher Blockchiffren: das *Feistel-Netzwerk*, benannt nach seinem Erfinder Horst Feistel. Feistel-Netzwerke beschreiben ein Grundprinzip zur Verschlüsselung von Blöcken, denn die hierbei für Teilblöcke verwendete Transformationsfunktion ist prinzipiell austauschbar und unterscheidet sich je nach Wahl des Verschlüsselungsalgorithmus. Jeder Block wird in zwei gleichgroße Hälften  $L_0$  (linke Hälfte) und  $R_0$  (rechte Hälfte) geteilt und in mehreren *Runden* mit einer *Funktion  $f$*  bearbeitet, die pro Runde verschiedene Schlüssel verwendet. Nach jeder einzelnen Runde werden die linke Seite ( $L_i$ ) und die rechte Seite ( $R_i$ ) vertauscht. Am Ende werden die Hälften wieder zu einem Block zusammengesetzt.

Sei  $i$  die Nummer der Runde (zwischen 1 und  $r$ ). Ein Feistel-Netzwerk kann dann mit folgenden Formeln beschrieben werden:

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

Dabei ist  $f$  die sogenannte *Runden- oder Transformationsfunktion* und  $k_i$  der im  $i$ -ten Schritt verwendete Schlüssel. Meistens leitet sich der in einer Runde verwendete Schlüssel aus einem *Gesamtschlüssel* ab. Der Operator  $\oplus$  bezeichnet hier die XOR-Verknüpfung

(Exklusives Oder). Feistel-Netzwerke werden eingesetzt, weil für die Entschlüsselung die Umkehrfunktion von  $f$  nicht benötigt wird, denn zur Entschlüsselung kann man folgende Formel verwenden:

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

Allerdings ist es zur Entschlüsselung notwendig, die *Rundenschlüssel*  $k_i$  in umgekehrter Reihenfolge wie bei der Verschlüsselung zu verwenden. Feistel-Netzwerke sind also relativ effizient in Hardware zu implementieren. Ein weiterer Vorteil ist, dass bereits wenige Runden ausreichend sind, um sicherzustellen, dass jedes Bit des Chiffrats von jedem Bit des Schlüssels abhängt. Dadurch kann kein Bit des Klartextes oder des Schlüssels ohne gravierende Änderungen des Chiffrats verändert werden.

Feistel-Netzwerke kommen z.B. bei den Verschlüsselungsverfahren DES, Blowfish, CAST und Twofish zum Einsatz.

## 4.2 Stromchiffren

*Stromchiffren* wandeln Klartexte im Gegensatz zu Blockchiffren bitweise (bzw. zeichenweise) in Chiffretext um. Dadurch wird es möglich, Informationseinheiten sofort verschlüsseln zu können, noch bevor ein kompletter Block vorliegt. Dies ermöglicht ein schnelleres Arbeiten der Verschlüsselung. Informationseinheiten können z.B. ASCII-Zeichen (8 Bit lang) oder Unicode Zeichen (16 Bit lang) sein. Bei Stromchiffren kommt ein *Schlüsselstromgenerator* zum Einsatz, der zu jedem Klartextbit ( $p_i$ ) ein Schlüsselbit ( $k_i$ ) erzeugt. Das Schlüsselbit wird dann mit dem jeweiligen Klartextbit mittels XOR verknüpft ( $c_i$ ).

$$c_i = p_i \oplus k_i$$

Auf der Entschlüsselungsseite werden die Chiffretextbits mit einem identischen Schlüsselstrom XOR-verknüpft, um die Klartextbits wiederherzustellen:

$$p_i = c_i \oplus k_i$$

Das Verfahren funktioniert aus folgendem Grund:

$$p_i \oplus k_i \oplus k_i = p_i$$

Die Sicherheit des Systems beruht vollständig auf dem Schlüsselstromgenerator. Liefert dieser einen endlosen Strom von Nullen, so sind Chiffretext und Klartext identisch und die ganze Operation ist wertlos. Liefert der Schlüsselstromgenerator ein wiederholtes 16-Bit-Muster, besteht der Algorithmus aus einer einfachen XOR-Verknüpfung mit vernachlässigbar geringer Schutzwirkung. Liefert der Schlüsselstromgenerator dagegen

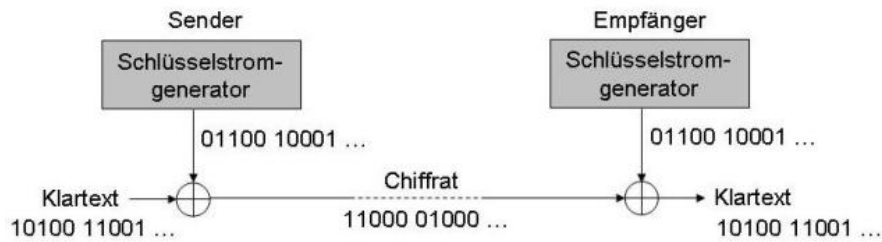


Abbildung 4.2: Verknüpfung eines Schlüsselstroms mittels XOR mit der Klartextnachricht

einen endlosen Strom von Zufallsbits (echte Zufallsbits, keine Pseudozufallsbits<sup>1</sup>), so erhält man einen One-Time-Pad und damit perfekte Sicherheit.

Da echte Zufallszahlen in Rechnern nur schwer zu erzeugen sind, liegt die Sicherheit einer Stromchiffrierung in der Realität irgendwo zwischen der einfachen XOR-Verknüpfung und One-Time-Pads. Der Schlüsselstromgenerator liefert einen Bitstrom, der zufällig aussieht, in Wirklichkeit jedoch aus einem deterministischen Strom besteht, der von einem Pseudozufallszahlengenerator erzeugt wurde.

Ein essentielles Problem bei Stromchiffren ist beispielsweise die Tatsache, dass ein Angreifer den Schlüsselstrom rekonstruieren kann, sofern er im Besitz sowohl des Klartextes als auch des Chiffretextes ist. Dies ist der Fall, da  $\forall i : p_i \oplus c_i = p_i \oplus p_i \oplus k_i = k_i$ . Weitere Nachrichten, die mit diesem Schlüsselstrom verschlüsselt werden, können also zumindest solange entschlüsselt werden, wie Bits im Schlüsselstrom vorliegen. Genau diese Lücke tritt bei der Verschlüsselung von drahtlosen Netzen mittels WEP auf.

Für weitere Informationen zu Stromchiffren allgemein bzw. deren Problemen und dem Aufbau eines Schlüsselstromgenerators sei der interessierte Leser auf [BS96] und [RBSM05] verwiesen.

Eingesetzte Stromchiffren sind A5/1 und A5/2, RC4 (Rivest Cipher Nr. 4) (bzw. dessen Ausprägung WEP) und SEAL. Über die Nutzung von RC4 bei WEP gibt z.B. [RBSM05] auf Seite 158 einen guten Überblick. Übrigens ist auch die altgediente Vigenère-Chiffre sowie der One-Time-Pad eine Stromchiffre. Der One-Time-Pad ist aber im Gegensatz zu Vigenère nicht periodisch, da sich der Schlüssel niemals wiederholt.

Anhand der Art der Erzeugung des Schlüsselstroms kann man Stromchiffren in *selbst-synchronisierende* und *synchrone* Stromchiffren unterteilen. Nähere Informationen hierzu finden sich in [BS96] und [RBSM05].

<sup>1</sup>Pseudozufallszahlen sind ein Thema für sich, welches aber im Rahmen dieser Seminararbeit nicht behandelt wird. Dies würde den Rahmen der Ausarbeitung sprengen. Für detailliertere Informationen siehe [BS96].

### 4.3 Betriebsmodi von symmetrischen Blockchiffren

Blockchiffren werden oft in Netzwerken eingesetzt, um die Übertragung von Nachrichten zu sichern. Sie weisen allerdings ein grundsätzliches Problem auf: Da Blockchiffren ein und denselben Klartextblock bei Verwendung des gleichen Schlüssels immer in den gleichen Chiffretextblock überführen, kann ein aktiver Angreifer Replay-Angriffe durchführen, indem er chiffrierte Nachrichtenblöcke aufzeichnet und diese später in eine andere Übertragung einspielt. Ein passiver Angreifer kann Wiederholungen von Blöcken erkennen und daraus eventuell Schlussfolgerungen ziehen, ohne den Chiffretext direkt entschlüsseln zu können.

Um die angesprochenen und andere Schwächen zu beseitigen und um symmetrische Chiffren zu verschiedenen Zwecken einsetzen zu können (z.B. zur Integritätssicherung), gibt es verschiedene *Betriebsmodi*, in denen die entsprechende Chiffre betrieben werden kann.

#### Electronic Codebook Mode

Beim *Electronic Codebook Mode* (ECB) wird jeder Block einzeln verschlüsselt und die einzelnen Blöcke werden nicht miteinander verknüpft. Der Name rührt daher, dass für diesen Betriebsmodus ein Codebuch angelegt werden kann, d.h. eine Tabelle, die zu jedem Block den entsprechenden verschlüsselten Block enthält. ECB ist der einfachste und zugleich unsicherste Modus, denn gleiche Klartextblöcke werden auch immer auf gleiche Chiffretextblöcke abgebildet, was einem Angreifer bereits weitreichende Rückschlüsse auf die Klartextstruktur aus dem Chiffretext erlaubt.

Wenn man eine Grafik damit verschlüsselt, die nur ein paar schwarze Linien hat, und dabei 0 (Bit) für Weiß und 1 (Bit) für Schwarz steht, wird man sehr viele Blöcke finden, die nur aus 0 bestehen. Alle Geheimtextblöcke die dann anders sind, enthalten min. eine 1 (Bit). Dadurch könnte man die Zeichnung bis auf ein paar Millimeter Abweichung rekonstruieren, ohne den Schlüssel zu kennen.

Der Vorteil liegt z.B. in der Effizienz, denn die Verarbeitung der einzelnen Klartextblöcke ist im ECB parallelisierbar.

#### Cipher Block Chaining Mode

Um das Problem von ECB zu beheben, nutzt der *Cipher Block Chaining Mode* (CBC) eine Rückkopplung, d.h. das Chiffretext des vorigen Blocks wird in die Verschlüsselung des aktuellen Blocks miteinbezogen. Damit hängt jeder Chiffretextblock von den vorigen Chiffretextblöcken ab und identische Klartextblöcke ergeben unterschiedliche Geheimtexte. Nun können Wiederholungen desselben Blocks nicht mehr erkannt werden und Replay-Attacken sind nicht mehr wirksam. Die Verknüpfung der Chiffretextblöcke miteinander wird dadurch erreicht, dass das Ergebnis des vorigen Chiffretextblocks durch XOR mit dem aktuell zu verschlüsselnden Block verknüpft wird, wie in Abbildung 4.3 dargestellt.

Bei der Entschlüsselung wird zunächst ein Chiffretextblock entschlüsselt. Anschließend wird der nächste Block entschlüsselt und mit dem Ergebnis der vorigen Entschlüsselung mittels XOR verknüpft.

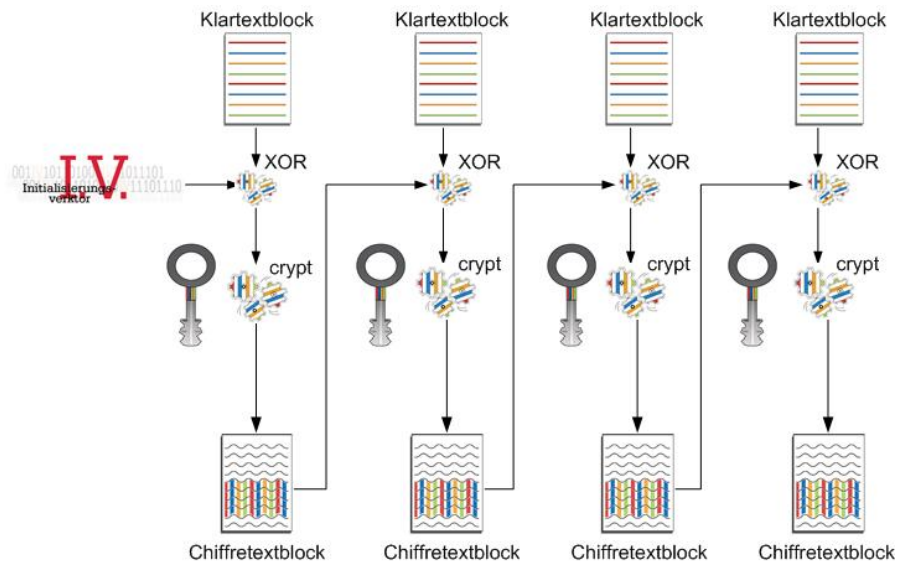


Abbildung 4.3: Funktionsweise des Cipher Block Chaining Modus

Im CBC-Modus werden unter Umständen gleiche Nachrichten immer noch auf gleiche Chiffretextblockfolgen abgebildet. Dies kann durch die Verwendung eines zufälligen *Initialisierungsvektors* (IV) vermieden werden, der mit dem ersten Block wie in der Abbildung gezeigt verknüpft wird. Nun muss allerdings zusammen mit dem ersten Block auch der Initialisierungsvektor übermittelt werden, damit es dem Empfänger möglich ist, die Nachricht zu entschlüsseln. Wird ein IV verwendet, so ist darauf zu achten, dass für jede Übertragung ein anderer IV verwendet wird, um bei jeder Verschlüsselung eine andere Blockverschlüsselung zu erreichen.

Ein Nachteil des CBC ist seine Fehleranfälligkeit: bereits 1-Bit-Fehler in einem Chiffretextblock macht die Entschlüsselung aller weiteren Blöcke unmöglich. Ein weiterer Nachteil ist die Verschlüsselung des letzten Blocks. In diesem können beliebige Bits manipuliert werden, da er nicht mehr in die weitere Verschlüsselung eingeht. Stehen im letzten Block noch wichtige Daten, sind diese somit anfällig für Angriffe. Es muss also auf Datenebene geprüft werden, ob eine Manipulation oder ein Fehler stattfand.

### Cipher Feedback Mode

Eine Blockchiffre kann durch den *Cipher Feedback Modus* (CFB) auch als selbstsynchronisierende Stromchiffre verwendet werden. Beim CBC-Modus kann die Verschlüsselung erst dann beginnen, wenn ein vollständiger Datenblock vorliegt. Diese stellt bei manchen Netzanwendungen ein Problem dar. In einem sicheren Netzwerk muss ein Terminal zum

Beispiel in der Lage sein, jedes eingetippte Zeichen sofort zum Host zu übertragen. Für Anwendungen, in denen Daten byteweise verarbeitet werden müssen, ist CBC ungeeignet.

Im CFB-Modus können Daten in Einheiten verschlüsselt werden, die kleiner als die Blockgröße sind. Hierfür kommt ein Puffer zum Einsatz, der die gleiche Größe wie ein Block der Blockchiffre hat (die Blockgröße sei hier in  $b$  Bit angegeben). Dieser Puffer wird in jedem Schritt verschlüsselt, allerdings wird zur Verschlüsselung der Klartextzeichen nur ein Teil des Puffers verwendet, der die Größe eines einzelnen Zeichens ( $z$  Bit,  $z \leq b$ ) hat. Im Beispiel der ASCII-Zeichen ist z.B.  $z = 8$ .

Zu Beginn enthält der Puffer wie im CBC einen Initialisierungsvektor. Der Puffer wird verschlüsselt und die 8 Bit, die im Ergebnis ganz links stehen, werden mit den ersten 8 Bit des Klartexts XOR verknüpft. Dies liefert das erste 8-Bit-Zeichen des Chiffretexts, das nun übertragen wird. Die gleichen acht Bit werden auf die acht Bitpositionen ganz rechts im Puffer verschoben (daher die Bezeichnung Cipher Feedback), alle anderen Bits wandern um acht Positionen nach links. Die acht Bit ganz links werden nicht mehr benötigt und somit verworfen. Anschließend wird das nächste Klartextzeichen auf die gleiche Art verschlüsselt. Zur Entschlüsselung läuft der umgekehrte Prozess ab. Abbildung 4.4 demonstriert die Vorgehensweise anhand eines 8-Bit-CFB mit einem 64-Bit-Blockalgorithmus.

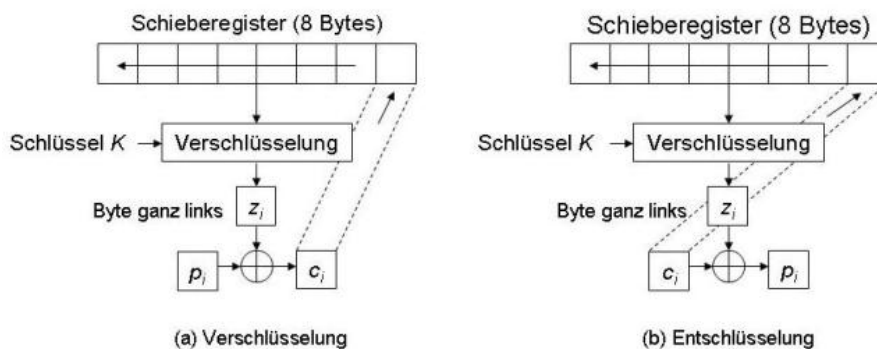


Abbildung 4.4: Funktionsweise des Cipher Feedback Modus

Ebenso wie der CBC-Modus verknüpft der CFB-Modus Klartextzeichen, so dass der Chiffretext vom vorhergehenden Klartext abhängt. Der Initialisierungsvektor ist nicht geheim. Er muss jedoch für jede Nachricht geändert werden, z.B. könnte eine Seriennummer zum Einsatz kommen, die mit jeder Nachricht erhöht wird.

### Output Feedback Mode

Der *Output-Feedback-Modus* (OFB) ist eine Methode zum Betrieb einer Blockchiffrierung als synchrone Stromchiffrierung. Er ähnelt dem CFB-Modus, nur werden hier  $z$  Bit des vorherigen *Ausgabeblocks* in die Positionen ganz rechts im Puffer gebracht. Das Verfahren heißt manchmal auch *interne Rückkopplung*, da der Rückkopplungsmechanismus

weder vom Klartextstrom noch vom Chiffretextstrom abhängt.

Es existiert noch eine Vielzahl weiterer Modi bzw. Varianten der bisher vorgestellten Modi, wie z.B. Counter Mode, Propagating Cipher Block Chaining Mode, Counter Mode mit CBC-MAC (MAC = Prüfsumme) etc. Die Vor- und Nachteile bzgl. Sicherheit, Effizienz und Ausfallsicherheit der wichtigsten vier Modi ECB, CBC, CFB, OFB fasst [BS96] sehr übersichtlich auf Seite 247 zusammen.

## 4.4 DES

Der *Data Encryption Standard (DES)* wurde 1981 vom American National Standard Institute (ANSI) als offener Standard für Privat- und Business-Anwendungen veröffentlicht.

DES ist eine symmetrische Blockchiffre mit einer Blockgröße von 64 Bit, die sowohl zur Ver- als auch Entschlüsselung den gleichen Algorithmus und den gleichen Schlüssel (mit 56 Bit Länge<sup>2</sup>) verwendet. Lediglich die Reihenfolge der einzelnen Teilschlüssel unterscheidet sich beim Ver- und Entschlüsseln. Die Sicherheit basiert auf den Prinzipien *Confusion* und *Diffusion*. Der Grundbaustein von DES ist die Anwendung einer vom Schlüssel abhängigen Kombination dieser Verfahren (eine Substitution - also Ersetzung - gefolgt von einer Permutation - einer Funktion, die keine zwei Werte auf das gleiche Ergebnis abbildet) auf den Text. Ein solcher Vorgang wird als *Runde* bezeichnet. DES arbeitet mit 16 Runden. Die gleiche Kombination von Verfahren wird sechzehnmal auf den Klartextblock angewandt.

Zu Beginn nimmt der DES-Algorithmus eine *Eingangsp permutation* vor und der Block wird in eine jeweils 32 Bit lange rechte und linke Hälfte zerlegt. Anschließend finden 16 gleiche Runden statt. Nach der sechzehnten Runde werden die rechte und linke Hälfte zusammengefügt und abschließend noch die zur Eingangsp permutation inverse Schlußpermutation angewendet. Wie Abbildung 4.5 zeigt, verwendet DES ein Feistel-Netzwerk.

In jeder Runde werden die Bits des Schlüssels verschoben und anschließend 48 Bit aus den 56 Bits des Schlüssels ausgewählt. Die rechte Hälfte der Daten wird mit Hilfe einer Expansionspermutation auf 48 Bit verbreitert, mittels XOR mit den 48 ausgewählten Schlüsselbits verknüpft und in acht so genannte *Substitutions-Boxen (S-Boxen)* gegeben, die zusammen wieder 32 Bit Ausgabe erzeugen. Abschließend findet eine weitere Permutation (die so genannte P-Box-Permutation) statt. Eine Runde - die im Folgenden als Funktion  $f$  bezeichnet wird - besteht also aus 4 Operationen: Schlüsseltransformation, Expansionspermutation, S-Box-Substitution und P-Box-Substitution. Die Ausgabe von  $f$  wird mit der linken Hälfte der Daten aus der letzten Runde mittels XOR verknüpft. Das Ergebnis dieser Operation wird anschließend die neue rechte Datenhälfte, die ehemalige

<sup>2</sup>Der Schlüssel besitzt eigentlich 64 Bits. Jedoch stehen dem Benutzer von diesen 64 Bits nur 56 Bits zur Verfügung; die übrigen 8 Bits (jeweils ein Bit aus jedem Byte) werden zum Paritäts-Check benötigt. Die wirkliche Schlüssellänge beträgt daher nur 56 Bits.

rechte Hälfte wird zur linken Hälfte. Mathematisch lässt sich eine Runde des DES folgendermaßen darstellen:

$B_i$  sei das Ergebnis der  $i$ -ten Iteration,  $L_i$  und  $R_i$  die linken und rechten Hälften von  $B_i$ ,  $K_i$  der 48 Bit lange Schlüssel der Runde  $i$  und  $f$  die Funktion, die Substitution, Permutation und XOR mit dem Schlüssel durchführt. Eine Runde sieht dann wie folgt aus:

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

## S-Boxen

Die S-Box-Substitution ist die wichtigste Stelle von DES. Die anderen Operationen des Algorithmus sind linear und lassen sich leicht analysieren. Die S-Boxen sind nichtlinear und gewährleisten mehr als alles andere die Sicherheit von DES. Aus diesem Grund lohnt es sich, einen genaueren Blick darauf zu werfen.

Eine S-Box dient als Basis für eine monoalphabetische Verschlüsselung. Sie ist meist als Array implementiert und gibt an, welches Byte wie getauscht wird. Nachdem der komprimierte Schlüssel mit dem expandierten Block XOR-verknüpft wurde, wird mit dem 48 Bit langen Ergebnis eine Substitution durchgeführt. Die 48 Bit werden in acht Teilblöcke aufgeteilt, wobei jeder Teilblock von einer eigenen der insgesamt *acht S-Boxen* bearbeitet wird. Jede S-Box verwendet eine Eingabe von 6 Bit, um eine Ausgabe von 4 Bit zu erzeugen. Die acht S-Boxen von DES benötigen insgesamt 256 Byte Speicherplatz. Tabelle 4.1 zeigt beispielhaft die sechste S-Box des DES.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Tabelle 4.1: Die sechste S-Box des DES

Die sechs Eingabebits  $b_1, b_2, b_3, b_4, b_5, b_6$  legen einen Eintrag der S-Box auf ganz spezielle Weise fest. Die Bits  $b_1$  und  $b_6$  werden zu einer zwei Bit langen Zahl zusammengesetzt, die eine Zeile der Tabelle auswählt. Die mittleren vier Bits  $b_2$  bis  $b_5$  werden zu einer vier Bit langen Zahl zwischen 0 und 15 zusammengesetzt. Sie legt eine Spalte der Tabelle fest.

Beispiel: Ist die Eingabe der Wert 51, so ist  $b_1b_2b_3b_4b_5b_6 = 110011$ . Es werden also die dritte Zeile (da  $b_1b_6 = 11$ ) und die neunte Spalte (da  $b_2b_3b_4b_5 = 1001$ ) verwendet. Dort steht der Wert 14, welcher ausgegeben wird. Somit ist für 110011 der Wert 1110 zu substituieren.

In [BS96] wird genauer auf die einzelnen Schritte von DES - namentlich die Eingangspermutation, Schlüsseltransformation, Expansionspermutation, S-Box-Substitution und

P-Box-Permutation - eingegangen. Wer einen Blick auf die durchaus interessante und kontroverse Entstehungsgeschichte (Stichwort: NSA) von DES werfen möchte, sei ebenfalls auf [BS96] verwiesen. Abbildung 4.7 gibt nochmal einen Überblick über sämtliche Schritte vom Klartext zum Chiffretext.

Der DES-Algorithmus beschreibt zunächst nur, wie ein Datenblock mit 64 Bits verarbeitet wird. Zur Verarbeitung einer Nachricht beliebiger Länge werden die bereits vorgestellten Betriebsmodi (CB, CBC, CFB, OFB) angewandt. Diese Verfahren lassen sich auf jede Blockchiffre anwenden. Die Nachricht wird in 64-Bit-Blöcke zerlegt. Beim ECB und CBC wird der letzte Block auf volle 64 Bits ergänzt bzw. ein weiterer Block angefügt.

Der Algorithmus ist zwar etwas kompliziert, doch geradlinig. Er arbeitet nur mit einfachen logischen Operationen auf kleinen Bitgruppen und kann somit ziemlich effizient in Hardware implementiert werden. Breite Anwendung findet er z.B. bei Geldautomaten: Mithilfe des DES-Algorithmus und einem geheimen Schlüssel errechnet dieser aus den Daten des Magnetstreifens (Kontonummer, Bankleitzahl, Gültigkeitszeitraum, ...) die PIN und vergleicht das Ergebnis mit der Eingabe des Benutzers.

### Sicherheit des DES

Lange Zeit war der DES der am meisten eingesetzte symmetrische Algorithmus. Allerdings besitzt er ein paar Schwächen, auf die im Folgenden eingegangen wird:

- *Schwache Schlüssel*: Gewisse Schlüssel können bewirken, dass alle Runden des DES denselben Rundenschlüssel verwenden und daher alle vermieden werden müssen. Es handelt sich u.a. um die Schlüssel (hexadezimal) 0000000 0000000, 0000000 FFFFFFFF, FFFFFFFF 0000000 und FFFFFFFF FFFFFFFF.
- *Schlüssellänge*: Die Schlüssellänge von 56 Bit gewährleistet aufgrund der gestiegenen Leistungsfähigkeit der Rechner für heutige Anwendungen keine ausreichende Sicherheit mehr. Dies zeigen die drei „DES Contests“ von 1997, 1998 und 1999. Bei allen drei Wettbewerben wurde der Schlüssel durch einen Brute-Force-Angriff gebrochen. 1999 wurden nur noch 22 Stunden dafür benötigt.
- *Design der S-Boxen*: Die S-Boxen des DES wurden von der NSA im endgültigen Entwurf noch geändert, die Design-Kriterien jedoch nicht offengelegt. Deshalb besteht die Gefahr, dass die NSA eine Hintertür installiert hat. Allerdings wurde bis heute kein Beweis für diese These gefunden.

Diese sind nur ein paar der Risiken bzw. Schwächen von DES, welche in [BS96] ausführlicher diskutiert werden.

## 4.5 AES

Um DES weiterhin nutzen zu können, wurden diverse Erweiterungen entwickelt, wovon die bekannteste wahrscheinlich *3DES* - oder auch *Triple-DES*, *Multiple DES* - sein dürfte. Hierbei werden hintereinander drei DES-Operationen ausgeführt. Der Aufwand zum Brechen durch Brute-Force steigt von  $2^{56}$  auf  $2^{168}$  (bzw.  $2^{112}$ ), da zur Ver- bzw. Entschlüsselung drei (bzw. nur zwei) verschiedene Schlüssel verwendet werden. 3DES arbeitet bei Verwendung von zwei Schlüsseln von jeweils 56 Bit mit einer Gesamt-Schlüssellänge von 112 Bit. Zur Chiffrierung des Ausgangstextes wird dieser zuerst mit Schlüssel 1 verschlüsselt. In der zweiten Stufe findet dann eine Entschlüsselung statt, normalerweise mit Schlüssel 2, worauf anschließend in Stufe 3 wiederum Schlüssel 1 verwendet wird.

Die Formel hierzu lautet:

$$\begin{aligned} C &= E_{k_1}(D_{k_2}(E_{k_3}(P))) \\ P &= D_{k_1}(E_{k_2}(D_{k_3}(C))) \end{aligned}$$

Triple-DES hat jedoch gegenüber dem DES den dreifachen Aufwand. Bis jetzt sind keine Schwächen von Triple-DES bekannt.

### 4.5.1 Die Entstehungsgeschichte

Das US-amerikanische National Institute of Standards and Technology (NIST) hatte Anfang 1997 auf der Suche nach einem neuen (sicheren) Algorithmus einen offenen Wettbewerb ausgeschrieben, dessen Sieger als *Advanced Encryption Standard* (AES) festgelegt werden sollte. Dabei wurden folgende Kriterien aufgestellt, die von den Algorithmen zu erfüllen sind:

- AES soll eine symmetrische Blockchiffre sein
- Die Blocklänge soll mindestens 128 Bit betragen.
- Schlüssel der Länge 128 Bit, 192 Bit und 256 Bit sollen möglich sein.
- Die Implementierung von AES soll in Hardware und Software leistungsfähig sein.
- Resistenz gegen alle bekannten Methoden der Kryptoanalyse
- Geringer Ressourcen-Verbrauch, um AES z.B. auch auf Smartcards einsetzen zu können
- Der Algorithmus soll von jedermann unentgeltlich genutzt werden können und insbesondere frei von patentrechtlichen Ansprüchen sein

Nachdem 5 Algorithmen die genannten Kriterien allesamt erfüllten - namentlich MARS, RC6, Rijndael, Serpent und Twofish -, wurden zwei weitere Kriterien hinzugezogen:

- Untersuchung auf theoretische Schwachstellen, die den Algorithmus in der Zukunft möglicherweise unsicher machen könnten. (Dies ist notwendig, da heute nicht gesagt werden, wie sich der Technologie-Fortschritt in den nächsten Jahren entwickelt. Vorgehensweisen, die heute als unmöglich erklärt sind, können in einigen Jahren schon alltäglich sein. Ein solches Risiko darf nicht eingegangen werden.)
- Rangfolge nach Ressourcenverbrauch und Leistung.

Im Oktober 2000 stand dann schließlich der Algorithmus *Rijndael* (gesprochen wie dt. „Rheindahl“), benannt nach den belgischen Erfindern Vincent Rijmen und Joan Daemen, als Gewinner fest. Rijndael konnte insbesondere überzeugen, da bezüglich der Leistungsfähigkeit Hardware- und Softwareimplementierungen effizient realisiert werden können und der geringe Aufbereitungsaufwand für Schlüssel schnelle Schlüsselwechsel erlaubt, was beispielsweise für Server-Implementierungen wichtig ist, die viele verschlüsselte Kommunikationsverbindungen unterhalten. Im Folgenden wird Rijndael nur noch als AES bezeichnet.

Es existiert sogar eine Rijndael Fan-Page (<http://rijndael.info/>), auf der weitere Informationen zum Algorithmus - Spezifikation, Implementierungen, Entstehungsgeschichte - gefunden werden können. Die offizielle Homepage zu AES findet sich natürlich auf den Seiten der NIST (<http://csrc.nist.gov/CryptoToolkit/aes/rijndael/>).

#### 4.5.2 Die Arbeitsweise

Nun ist das Ziel dieser Seminararbeit zwar nicht, jeden möglichen Chiffrieralgorithmus vorzustellen, jedoch handelt es sich bei DES und vor allem AES um zwei der wichtigsten Vertreter symmetrischer Chiffren. Aus diesem Grunde sollte in einer Ausarbeitung über die Grundlagen der Kryptographie ein Blick auf die Struktur und Funktionsweise dieser beiden Algorithmen geworfen werden.

Die verwendete Blocklänge und Schlüssellänge bei Rijndael können unabhängig voneinander auf 128, 192 oder 256 Bit eingestellt werden, während bei AES die Einschränkung der festgelegten Blockgröße von 128 Bit gilt. Dies ist der einzige Unterschied zwischen AES und Rijndael. AES basiert im Gegensatz zu seinem Vorgänger DES nicht auf einem Feistel-Netzwerk, sondern hat einen geschichteten Aufbau, bei dem sich jede Schicht auf einen kompletten Block auswirkt und ein Block sämtliche Schichten durchlaufen muss. Ähnlich wie DES verwendet AES verschiedene Runden zur Verschlüsselung. Die Anzahl der Runden  $r$  ist von der Schlüssellänge  $k$  und der Blockgröße  $b$  abhängig, wie aus Tabelle 4.2 hervorgeht.

Abbildung 4.8 zeigt den AES-Algorithmus in seinem schematischen Aufbau am Beispiel von zehn zu durchlaufenden Runden.

Pro Runde müssen vier Schichten durchlaufen werden.

#### Schicht 1 (Substitution)

	b = 128	b = 192	b = 256
k = 128	r = 10	r = 12	r = 14
k = 192	r = 12	r = 12	r = 14
k = 256	r = 14	r = 14	r = 14

Tabelle 4.2: Rundenanzahl  $r$  des AES bei verschiedenen Block- und Schlüssellängen

Schicht 1 besteht aus 16 *Substitutions-Boxen*. Die S-Boxen sind der nichtlineare Bestandteil des AES, die jeweils 8 Bit Eingabe verwenden, um aus einer statischen Tabelle 8 Bit Ausgabe auszuwählen (Erinnerung: 16 Boxen mit je 8 Bit  $\hat{=}$  der Beschränkung von AES auf 128 Bit Blockgröße). Die S-Boxen wurden so konstruiert, dass sie Schutz gegen verschiedene Methoden der Kryptoanalyse bieten (insbesondere lineare und differentielle Kryptoanalyse<sup>3</sup>).

### Schicht 2 (ShiftRows)

Schicht zwei permutiert die Ausgaben der S-Boxen. Ziel ist es, charakteristische Häufigkeitsverteilungen eines Textes bzw. ganze Phrasen zu beseitigen, um lineare Angriffe durch Häufigkeitsanalysen zu verhindern. Die Permutation wird bei AES auch als *Shift Rows* bezeichnet. Der Name rührt daher, dass bei AES der Eingabeblock als Matrix (zweidimensionale Tabelle mit vier Zeilen) interpretiert wird und alle Operationen auf dieser so genannten *Zustandsmatrix* ausgeführt werden. Die Permutation wird in dieser vierzeiligen Tabelle durch zyklisches Verschieben der letzten drei Reihen erreicht. Je nach Blocklänge  $b$  und Zeile in der Matrix wird die Zeile um 1 bis 4 Spalten verschoben.

### Schicht 3 (MixColumn)

Schicht 3 vermischt die Daten innerhalb der einzelnen Spalten in einer Zustandsmatrix (Diffusion). Ebenso wie die Permutation dient Diffusion dazu, charakteristische Häufigkeitsverteilungen zu verbergen. Diffusion in AES wird auch als *MixColumns* bezeichnet. Wie in Schicht 2 rührt auch dieser Name von der Zustandsmatrix her.

### Schicht 4 (KeyAddition)

Schicht 4 verknüpft den aktuellen Rundenschlüssel mit der Ausgabe des dritten Schritts. Dies ist die einzige Funktion in AES, die den Algorithmus vom *Anwenderschlüssel* abhängig macht. Die Rundenschlüssel werden nach einem *Key Schedule* durch Rekursion aus dem Anwenderschlüssel abgeleitet. Offensichtlich muss jeder Rundenschlüssel die Länge eines Blocks haben (sprich: 128 Bit). Sei im Folgenden  $N_k = 4$  bei Verwendung eines Schlüssels der Länge 128 Bit,  $N_k = 6$  bei 192 Bit und  $N_k = 8$  für Schlüssel der Länge 256 Bit. AES unterteilt den Schlüssel zunächst in  $N_k$  4-Byte Wörter  $k_0, \dots, k_{N_k-1}$  (d.h.

<sup>3</sup>[BS96] bietet hierzu einen guten Einstieg zur Erläuterung der Funktionsweise dieser beiden Kryptoanalysen.

bei  $k = 4$  sind das 16 Byte  $\equiv$  128 Bit Blocklänge). Weitere Wörter werden jeweils durch XOR-Verknüpfung des vorhergehenden Wortes  $k_{i-1}$  mit  $k_{i-N_k}$  erzeugt. Es existieren noch diverse Besonderheiten bei der Erstellung der Rundenschlüssel, die in [RBSM05] genauer nachgelesen werden können.

Im Groben kann der Verschlüsselungsvorgang mittels AES in folgenden drei Schritten kurz beschrieben werden:

- Zu Beginn wird die Schlüsselexpansion nach oben beschriebenem Key Schedule durchgeführt.
- Anschließend wird eine Anzahl von Runden (abhängig von der Block- und Schlüssellänge) durchlaufen.
- Die Verschlüsselung endet mit einer finalen Runde, die nur Schicht 1, 2 und 4 ausführt, aber auf Diffusion verzichtet.

Entschlüsselung ist mit dem gleichen Algorithmus möglich. Es müssen lediglich die Runden in umgekehrter Reihenfolge mit den jeweils inversen Transformationen durchlaufen werden.

Bis heute sind keine praktisch umsetzbaren Angriffe auf AES bekannt.

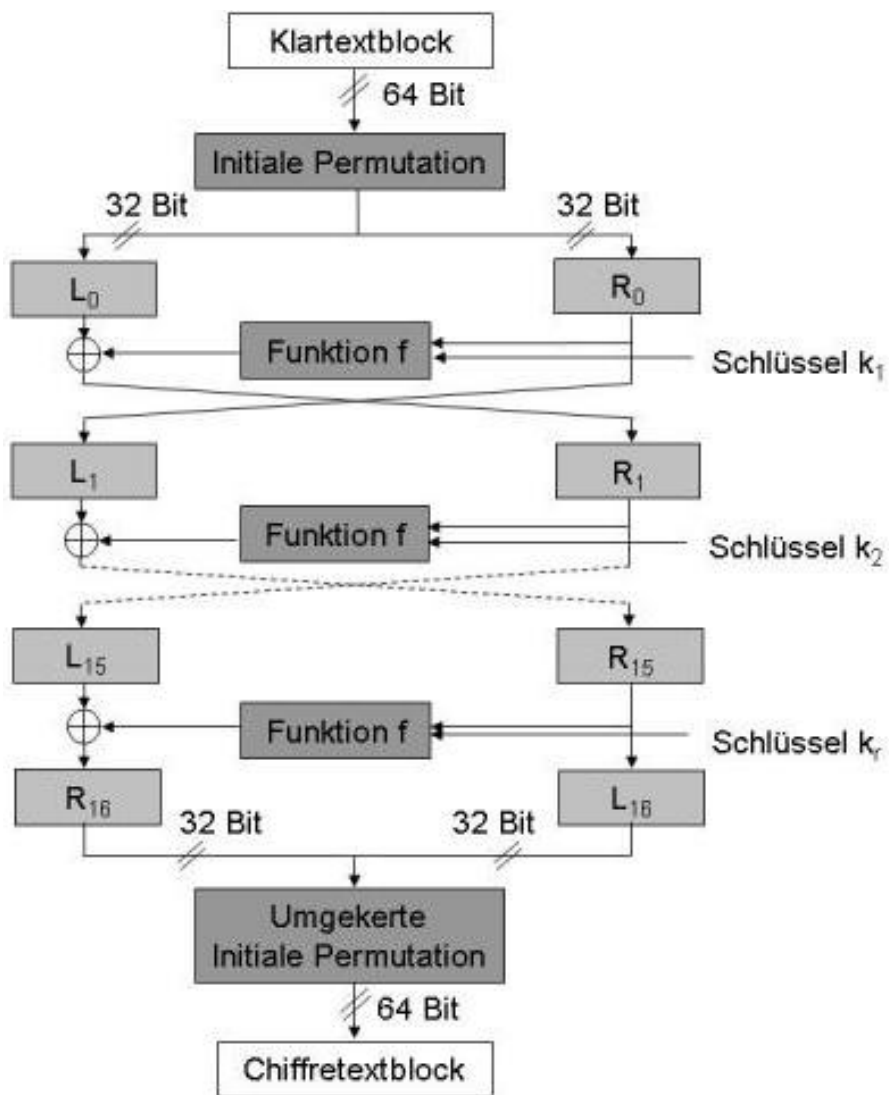


Abbildung 4.5: Struktur des DES-Algorithmus

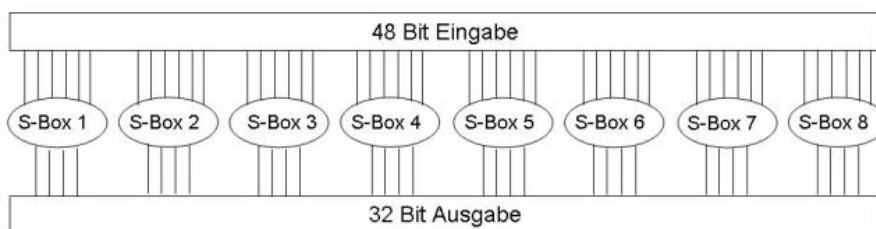


Abbildung 4.6: Substitution durch die S-Boxen

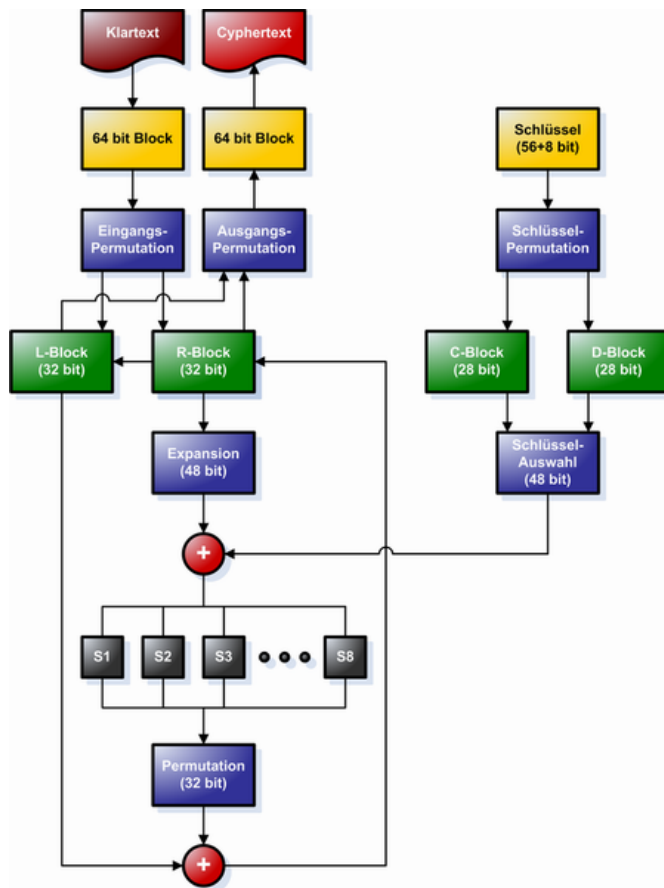


Abbildung 4.7: Die Funktionsweise von DES

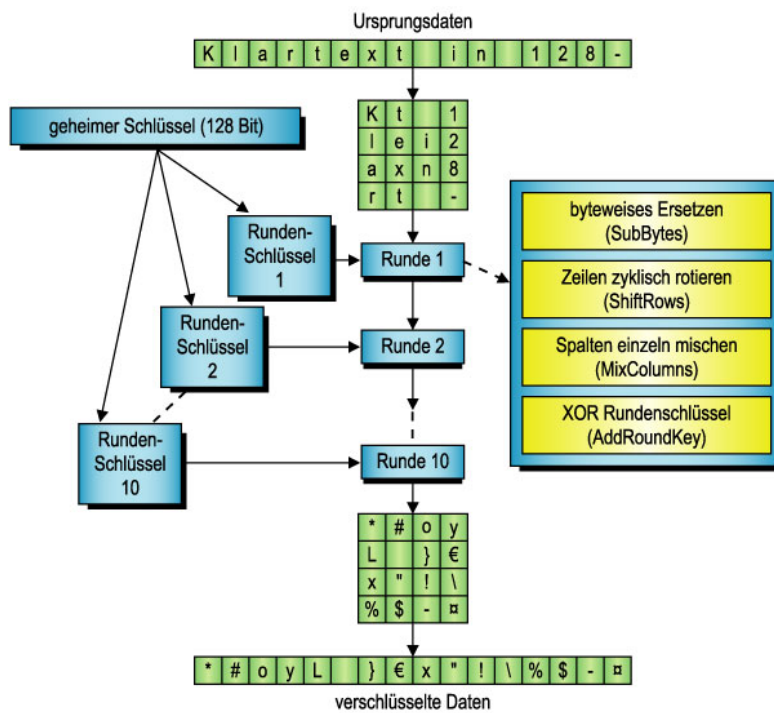


Abbildung 4.8: Schematischer Aufbau des AES-Algorithmus (10 Runden)

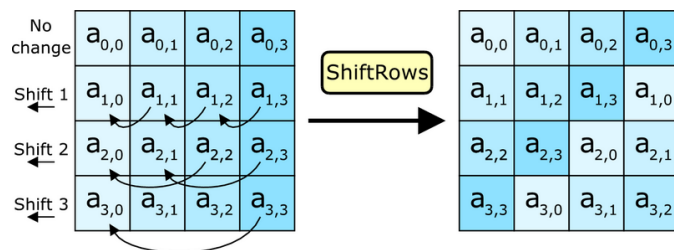


Abbildung 4.9: Verschiebung der Zeilen um eine bestimmte Anzahl von Spalten nach links



## 5 Asymmetrische Kryptographie

Die symmetrische Kryptographie hat ein entscheidendes Problem, welches immer wieder auftritt, sobald zwei Kommunikationspartner (die im weiteren Alice und Bob genannt werden) über einen (unsicheren) Kommunikationskanal Informationen miteinander austauschen wollen: das *Schlüsselverteilproblem*. Wie gestaltet man einen geheimen und vor allem sicheren Austausch der Schlüssel zwischen Sender und Empfänger? Wie kann man sicherstellen, dass nicht einem Dritten der Schlüssel zur beidseitigen chiffrierten Kommunikation in die Hände fällt? Sobald es nämlich einem Dritten gelingt, in den Besitz des geheimen Schlüssels zu gelangen, ist er automatisch in der Lage, von Alice oder Bob empfangene Daten zu entschlüsseln und sich auch als einer der beiden Teilnehmer auszugeben, Daten eigenhändig zu verschlüsseln und in deren Namen zu versenden.

Da je zwei Kommunikationspartner einen geheimen Schlüssel für ihren geheimen Informationsaustausch benötigen, stellt sich noch ein ganz anderes Problem, nämlich das der Skalierung. Jeder Teilnehmer an einem Kommunikationsnetzwerk muss mit jedem anderen Teilnehmer einen eigenen geheimen Schlüssel austauschen, dies bedeutet, dass bei  $N$  Teilnehmern  $N(N-1)$  Schlüssel ausgetauscht werden. Bei einer sehr großen Anzahl an Teilnehmern ist es offensichtlich, dass die Anzahl der symmetrischen Schlüssel sehr schlecht skaliert.

Dieser logistischen Schwachstelle versucht die *Asymmetrische Kryptographie* (auch *Public Key Kryptographie*) entgegenzutreten. Bei diesem Konzept wird folgender Ansatz verfolgt:

Anstatt eines einzelnen gemeinsamen geheimen Schlüssels  $K_{AB}$  (Schlüssel zwischen Alice und Bob) und den Funktionen bzw. Mechanismen  $E$  und  $D$  zum Ver- bzw. Entschlüsseln eines Klartextes kommen bei asymmetrischer Kryptographie *zwei verschiedene Schlüssel*  $e$  und  $d$  zum Einsatz. Dabei parametrisiert  $e$  die Verschlüsselungsfunktion  $E$  und  $d$  die Entschlüsselungsfunktion  $D$ . Somit gilt formal für alle möglichen Klartexte in der asymmetrischen Kryptographie:

$$D_d(E_e(P)) = P$$

Der Schlüssel  $e$  wird als *öffentlicher* Schlüssel bezeichnet und bildet mit dem *genau zu ihm passenden - komplementären -* Schlüssel  $d$  (der *geheime* Schlüssel) ein *Schlüsselpaar*. Das besondere an diesem Konzept ist, dass der öffentliche Schlüssel - wie es die Bezeichnung vermuten lässt - nicht geheim gehalten werden muss und an jede beliebige Person bzw. jedes beliebige System weitergegeben werden kann. Lediglich der private Schlüssel  $d$  muss geheim gehalten werden.

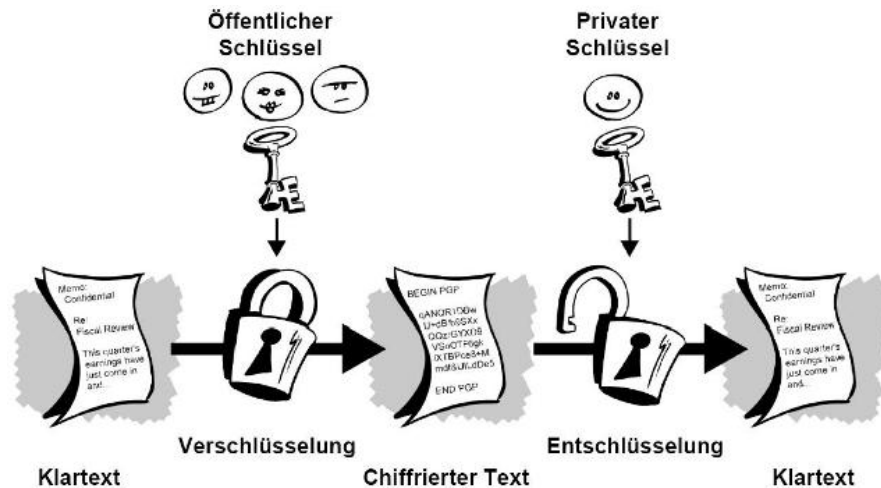


Abbildung 5.1: Verschlüsselung mit öffentlichen Schlüsseln

Ein Public-Key-Verschlüsselungssystem muss die folgenden drei Bedingungen erfüllen:

1. Es muss rechnerisch einfach sein, eine Nachricht zu ver- bzw. entschlüsseln, wenn der geeignete Schlüssel vorhanden ist.
2. Es muss rechnerisch unmöglich bzw. sehr aufwendig sein, den geheimen Schlüssel aus dem öffentlichen Schlüssel abzuleiten.
3. Es muss rechnerisch unmöglich bzw. sehr aufwendig sein, den öffentlichen Schlüssel mit Hilfe eines Chosen-Plaintext-Angriffs herauszufinden.

## 5.1 Ablauf einer asymmetrischen Verschlüsselung

Damit Alice eine verschlüsselte Nachricht an Bob schicken kann, müssen sich beide im Vorfeld auf ein gemeinsames Verschlüsselungssystem, d.h. auf die Mechanismen  $E$  und  $D$  zum Ver- und Entschlüsseln, einigen. Der Ablauf des Nachrichtenaustauschs ist dann folgender:

1. Bob generiert sich ein *Schlüsselpaar* bestehend aus einem öffentlichen Schlüssel  $e$  und einem dazu passenden geheimen Schlüssel  $d$ .
2. Bob veröffentlicht  $e$  geeignet, so dass Alice ihn abrufen kann. So könnte Bob den Schlüssel z.B. per E-Mail an Alice schicken, ihn auf Diskette speichern und diese an Alice schicken, den Schlüssel in ein weltweit einsehbares Verzeichnis mit öffentlichen Schlüsseln ablegen (wie dies z.B. im Falle von PGP-Schlüsseln mit den sogenannten Schlüsselserversn geschieht) usw. Es gibt eine Reihe einfacher Möglichkeiten, einen Schlüssel so zu veröffentlichen, dass praktisch jeder und insbesondere Alice in dessen

Besitz kommt. Der Nachweis, dass der Schlüssel auch wirklich zu Bob gehört, ist allgemein jedoch problematisch<sup>1</sup>.

3. Alice ist nun im Besitz von  $e$  und kann damit und mit Hilfe der Verschlüsselungsfunktion  $E$  das Chifftrat  $C = E_e(P)$  ihres Klartextes  $P$  erzeugen.
4. Alice schickt dieses Chifftrat  $C$  an Bob.
5. Da Bob ein zu  $e$  passendes  $d$  gehört, kann er erfolgreich den Klartext  $P = D_d(C)$  aus  $C$  berechnen.

Ein großer Vorteil asymmetrischer Kryptographie gegenüber symmetrischer liegt nun in der *Wiederverwendbarkeit des öffentlichen Schlüssels*. Die Generierung eines Schlüsselpaars in Schritt 1 muss von Bob nur ein einziges mal durchgeführt werden. Bei weiteren Kommunikationsvorgängen - auch mit anderen Kommunikationspartnern - entfällt die Generierung. Somit skaliert auch die Anzahl an Schlüsseln im Netz besser als bei einfachen symmetrischen Schlüsseln.

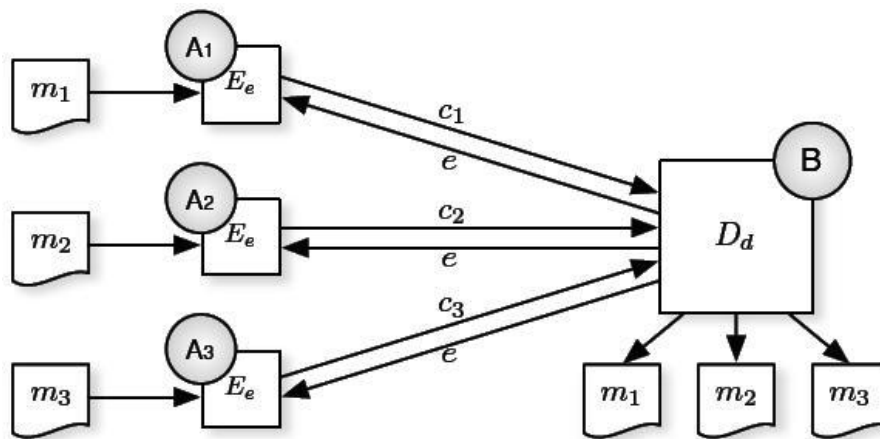


Abbildung 5.2: Schema einer Mehrparteien-Kommunikation unter Verwendung desselben öffentlichen Schlüssels  $e$

## 5.2 Was ist besser?

Nun mag sich dem ein oder anderen Leser die Frage stellen: was ist besser? Public-Key-Kryptographie oder symmetrische Kryptographie? Eine pauschale Antwort auf diese Frage kann nicht gegeben werden, denn diese beiden Arten von Kryptographie sind nicht direkt vergleichbar. Die Anzahl und Länge der Nachrichten bei Public-Key-Verfahren sind wesentlich größer als bei symmetrischen Algorithmen, auch die Schlüssellängen sind um ein Vielfaches höher. Daraus könnte man folgern, dass ein symmetrischer Algorithmus

<sup>1</sup>Eine nähere Erläuterung zum Problem der Authentizität von Schlüsseln würde hier zu weit führen, siehe dazu [BSCE00]

effizienter ist als ein Public-Key-Algorithmus. Das ist zwar korrekt, doch was bei diesem Blickwinkel nicht berücksichtigt wurde sind die erheblichen Sicherheitsvorteile der Public-Key-Kryptographie. Symmetrische und Public-Key-Kryptographie sind zwei verschiedene Ansätze zur Lösung zweier verschiedener Arten von Problemen.

Symmetrische Kryptographie eignet sich am besten zur Verschlüsselung von Daten. Sie ist um Größenordnungen schneller und nicht anfällig für Chosen-Ciphertext-Angriffe.

Public-Key-Kryptographie ist für die sichere Schlüsselverwaltung, also außerhalb des Einsatzbereichs symmetrischer Kryptographie, geeignet. Ihr Einsatz ist also zwingend auf den Austausch von Schlüsseln für konventionelle (symmetrische) Kryptographie zu beschränken (und auch für die digitale Signatur - also elektronische Unterschrift - von Schlüsseln, aber das ist ein Thema für sich). In diesem Zusammenhang muss auch auf *hybride* Systeme eingegangen werden, welchen im Kapitel 6 Rechnung getragen wird.

Beispiele für Verschlüsselungssysteme mit öffentlichen Schlüsseln sind *Elgamal* (nach dem Erfinder Taher Elgamal aus dem Jahr 1985), *RSA* (nach den Erfindern Ron L. Rivest, Adi Shamir und Leonard Adleman aus dem Jahr 1977) und *DSA*, der Digital Signature Algorithm, im Jahr 1991 von NIST vorgeschlagen. Der *Diffie-Hellman-Schlüsselaustausch* (ebenfalls nach seinen Erfindern benannt, entwickelt 1976) ist hingegen kein Verschlüsselungsverfahren, sondern eher eine theoretische Grundlage für die spätere Entwicklung von RSA.

Die nächsten beiden Abschnitte werden sich zum einen mit Diffie-Hellman und zum anderen mit RSA - als einer der heute wohl prominentesten Vertreter für asymmetrische Kryptographie - beschäftigen, um dem Leser einen etwas genaueren Einblick in die Arbeitsweise dieser Verfahren zu gewähren. Die Verfahren basieren auf teilweise recht komplizierten mathematischen Zusammenhängen, auf die diese Ausarbeitung aber nicht detailliert sondern nur grob zusammenfassend und rein übersichtlich eingehen wird.

### 5.3 Diffie-Hellman

Das erste auf der Public-Key-Idee basierende kryptographische Protokoll ist das 1976 von Whitfield Diffie und Martin Hellman vorgeschlagene Verfahren. Das Verfahren dient nicht direkt zum Verschlüsseln der Daten über einen öffentlichen Schlüssel und auch nicht zum Austausch eines vorher erzeugten Schlüssels, sondern ermöglicht den Kommunikationspartnern, auf Basis der ausgetauschten Informationen denselben Schlüssel zu generieren.

Die zugrundeliegende Mathematik ist einfach. Alice und Bob einigen sich zunächst auf eine große Primzahl  $p$  und eine Zahl  $g$  ( $2 \leq g \leq p - 2$ ), die modulo  $p$  primitiv<sup>2</sup> ist. Diese beiden Zahlen müssen nicht geheim bleiben - Alice und Bob können sich mit Hilfe eines unsicheren Kanals auf die Zahlen einigen. Es kann sogar eine ganze Benutzergruppe die gleichen Zahlen verwenden, das spielt keine Rolle.

Das Protokoll durchläuft nun folgende 4 Schritte:

<sup>2</sup>Zur Bedeutung von primitiven Elementen siehe Anhang A - Primitive Elemente.

1. Alice wählt geheim eine große zufällige Zahl  $a$  ( $2 \leq a \leq p - 2$ ) und sendet Bob

$$A = g^a \bmod p$$

2. Bob wählt geheim eine große zufällige Zahl  $b$  ( $2 \leq b \leq p - 2$ ) und sendet Alice

$$B = g^b \bmod p$$

3. Alice berechnet

$$k = B^a \bmod p = (g^b)^a \bmod p = g^{ab} \bmod p$$

4. Bob berechnet

$$k' = A^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p$$

$A$  und  $B$  bezeichnen hier die öffentlichen Schlüssel von Alice und Bob - zusammen mit  $p$  und  $g$  -, während  $a$  und  $b$  die jeweils geheimen Schlüssel darstellen sollen. Sowohl  $k$  als auch  $k'$  sind gleich  $g^{ab} \bmod p$ . Jemand, der den Kommunikationskanal abhört, kann diesen Wert nicht berechnen, sondern erfährt nur die Werte von  $p$ ,  $g$ ,  $A$  und  $B$ . Das Problem lässt sich nur lösen, indem man den *diskreten Logarithmus* berechnet und  $a$  oder  $b$  ermittelt.  $k$  stellt damit einen geheimen Schlüssel dar, den Alice und Bob *unabhängig voneinander berechnet* haben und mit dessen Hilfe sie nun sicher kommunizieren können. Abbildung 5.3 veranschaulicht noch einmal die einzelnen Schritte.

Tabelle 5.1 gibt ein Beispiel anhand konkreter Werte und zeigt noch einmal in übersichtlicher Weise, was die einzelnen Kommunikationsteilnehmer wissen. Eve (Evesdropper = Lauscher) sei hier eine dritte Person, die versucht, unbefugt den geheimen Schlüssel zu errechnen. Die Primzahl  $p$  sei hier 23, die öffentliche Basis  $g = 5$ , Alices privater Schlüssel  $a = 6$  sowie Bobs privater Schlüssel  $b = 16$ . Der geheime gemeinsame Schlüssel sei mit  $k$  notiert.

### 5.3.1 Sicherheit

Die Sicherheit des Verfahrens basiert auf der Verwendung einer sogenannten Einwegfunktion. Hierbei macht man sich die Tatsache zunutze, dass es zwar sehr einfach ist, eine Zahl zu potenzieren (sprich  $g^a \bmod p$  bzw.  $g^b \bmod p$ ), dass es jedoch nur mit sehr großem Aufwand möglich ist, den *diskreten Logarithmus* einer Zahl zu berechnen (sprich  $a$  aus  $A = g^a \bmod p$ ). In der Praxis werden sehr große Primzahlen verwendet, in Abhängigkeit der Bitkomplexität der gewählten Primzahl  $p$  steigt der Aufwand des Findens eines diskreten Logarithmus in etwa exponentiell an.

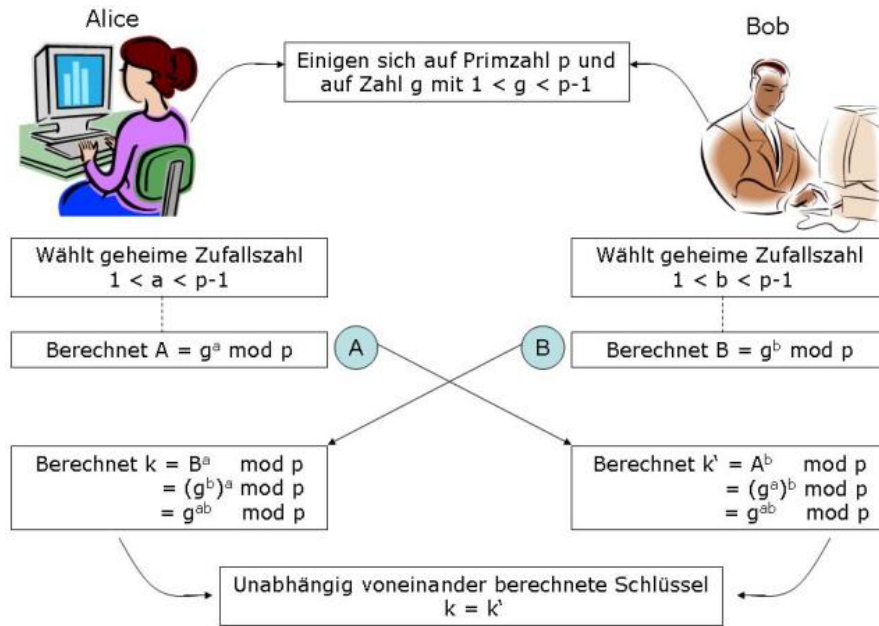


Abbildung 5.3: Protokollablauf nach Diffie-Hellman

Man könnte natürlich auch einfach ausprobieren, die Zahl  $a$  zu finden, aber die Größenordnung der Zahlen, die in der Praxis eingesetzt werden, verbietet dies: Die Zahlen (also  $p, g, a, b, A, B$ ) haben zwischen 100 und 200 Dezimalstellen. Um  $a$  durch systematisches Probieren zu finden, müsste man also mindestens  $10^{100}$  Versuche durchgehen. Diese Zahl ist um ein Vielfaches größer als die Zahl der Nanosekunden seit der Entstehung des Universums.

Bis heute ist jedoch nicht bekannt, ob es auch eine effiziente Möglichkeit gibt, d.h. eine Möglichkeit mit geringerem als exponentiell steigendem Aufwand. Wird ein Schlüsselaustausch mit dem beschriebenen Diffie-Hellman-Verfahren durchgeführt, so ist sehr leicht eine *Man-in-the-Middle*-Attacke durchführbar. Um sich dagegen zu schützen, gibt es Erweiterungen von Diffie-Hellman wie z.B. das *Station-To-Station-Protokoll*, auf welches hier aber nicht näher eingegangen werden soll.<sup>3</sup>

## 5.4 RSA

Das bekannteste und am weitesten verbreitete Public-Key-Verfahren *RSA* ist nach seinen Erfindern Ron Rivest, Adi Shamir und Leonard Adleman benannt, die es am MIT entwickelten und 1977 veröffentlichten. Es gilt als das erste Verfahren, mit dem nicht nur asymmetrische Kryptographie, sondern auch *digitale Signaturen* durchgeführt werden können. Es ist auch der populärste Public-Key-Algorithmus, nicht nur, weil er von all den im Laufe der Jahre entwickelten Public-Key-Algorithmen am einfachsten zu ver-

<sup>3</sup>Siehe dazu [BS96] bzw. [RBSM05].

Alice		Bob		Eve	
kennt	kennt nicht	kennt	kennt nicht	kennt	kennt nicht
$p = 23$	$b = 15$	$p = 23$	$a = 6$	$p = 23$	$a = 6$
Basis $g = 5$		Basis $g = 5$		Basis $g = 5$	$b = 15$
$a = 6$		$b = 15$			$k = 2$
$5^6 \bmod 23 = 8$		$5^{15} \bmod 23 = 19$		$5^a \bmod 23 = 8$	
$5^b \bmod 23 = 19$		$5^a \bmod 23 = 8$		$5^b \bmod 23 = 19$	
$19^6 \bmod 23 = 2$		$8^{15} \bmod 23 = 2$		$19^a \bmod 23 = k$	
$8^b \bmod 23 = 2$		$19^a \bmod 23 = 2$		$8^b \bmod 23 = k$	
$19^6 \bmod 23 = 8^b \bmod 23$		$8^{15} \bmod 23 = 19^a \bmod 23$		$19^a \bmod 23 = 8^b \bmod 23$	
$k = 2$		$k = 2$			

Tabelle 5.1: Kommunikationsteilnehmer und jeweiliger Wissensstand

stehen ist, sondern auch, weil er jahrelange gründliche Kryptoanalyse überstanden hat. Die Kryptoanalyse konnte die Sicherheit von RSA weder beweisen noch widerlegen, aber sie stärkt das Vertrauen in den Algorithmus.

### 5.4.1 Vorbereitung

Zunächst wird ein öffentlicher Schlüssel generiert: Dazu erzeugt Alice zwei unterschiedliche, sehr große Primzahlen  $p$  und  $q$ . Beide Primzahlen sollten in etwa von der gleichen Größenordnung sein. Das Auffinden großer Primzahlen stellt an sich ein schwieriges Problem dar, da die Komplexität der Untersuchung einer zufälligen Zahl auf die Primeigenschaft mit zunehmender Bitkomplexität exponentiell steigt. Aus diesem Grund kommen üblicherweise effiziente probabilistische Primtests wie etwa der von Rabin-Miller zum Einsatz.

Aus  $p$  und  $q$  kann Alice  $n$  berechnen:

$$n = pq$$

Alice wählt nun eine Zahl  $e$  zufällig, mit den Eigenschaften

$$2 < e < (p-1)(q-1)$$

und

$$\text{ggT}(e, (p-1)(q-1)) = 1.$$

Die letzte Eigenschaft besagt, dass  $e$  und  $(p-1)(q-1)$  keinen gemeinsamen Teiler haben, also *zueinander prim* sind. Alice kann nun schon den öffentlichen Teil ihres Schlüssel-paars publizieren:  $(n, e)$ .

Bob hat jetzt eine *Einwegfunktion*, die auf Alices Schlüssel zugeschnitten ist und mit der er seine Nachricht verschlüsselt an Alice schicken kann. An diesem Punkt ist die

verschlüsselte Nachricht sicher, da es naturgemäß sehr schwierig ist, die Umkehrung der Einwegfunktion zu berechnen<sup>4</sup>. Allerdings bleibt die Frage: Wie kann Alice die Mitteilung entschlüsseln?

Das ist nun das Geniale an Rivests Einwegfunktion: Sie ist für jedermann, der die Werte von  $p$  und  $q$  kennt, umkehrbar. Daher muss Alice ihre Werte  $p$  und  $q$  streng geheim halten. Dem Leser stellt sich vielleicht die Frage, warum die Nachricht so sicher sein soll, wenn doch alle Welt  $n$  - den öffentlichen Schlüssel - kennt, denn es wäre doch sicher möglich,  $p$  und  $q$  - den privaten Schlüssel - herausfinden und Alices Post zu lesen? Schließlich wurde  $n$  aus  $p$  und  $q$  erzeugt. Wenn  $n$  groß genug ist, so stellt sich jedoch heraus, ist es praktisch unmöglich,  $p$  und  $q$  aus  $n$  zu deduzieren, und dies ist der vielleicht schönste und eleganteste Zug an der asymmetrischen RSA-Chiffrierung.

Der geheime Teil  $d$  des Schlüsselpaars berechnet sich nun durch den erweiterten Euklidischen Algorithmus  $\text{eggt}(e, (p-1)(q-1))$ . Dieser liefert effizient (in diesem Fall mit Aufwand  $O(\log n)$ ) ein  $d$ , den geheimen Schlüssel, mit der die verschlüsselte Nachricht entschlüsselt werden kann.  $d$  hat die Eigenschaft

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

bzw. anders ausgedrückt:

$$d = e^{-1} \pmod{(p-1)(q-1)}$$

Hierbei ist zu beachten, dass  $d$  und  $e$  ebenfalls prim zueinander sind. Die beiden Primzahlen  $p$  und  $q$  werden nach der Erstellung von  $d$  nicht mehr benötigt. Sie können verworfen werden, dürfen aber niemals bekanntgegeben werden.

## 5.4.2 Ablauf einer Verschlüsselung

Alice hat ihren öffentlichen Schlüssel  $(n, e)$  für Bob zugänglich gemacht. Dieser kann nun den Klartext einer Nachricht  $P$ , den er an Alice übertragen möchte, verschlüsseln. Zur Verschlüsselung der Nachricht  $P$  zerlegt er (bzw. der Algorithmus) diese erst in numerische Blöcke<sup>5</sup>, die kleiner als  $n$  sind (bei Binärdateien wählt man die größte Zweierpotenz, die kleiner ist als  $n$ ). Sind  $p$  und  $q$  Primzahlen mit je 100 Ziffern so hat  $n$  knapp 200 Ziffern und jeder Nachrichtenblock  $m_i$  (um einen Namenskonflikt mit der Primzahl  $p$  zu vermeiden, wird hier für einen Nachrichtenblock  $m_i$  anstatt  $p_i$  verwendet) wird nicht ganz 200 Ziffern lang sein. Die verschlüsselte Nachricht  $C$  besteht aus Nachrichtenblöcken  $c_i$  ähnlicher Größe, die alle etwa gleich lang sind.

1. Bob berechnet die Chiffren

$$c_i \equiv m_i^e \pmod{n}$$

<sup>4</sup>Siehe hierzu die Erläuterung im Abschnitt Sicherheit.

<sup>5</sup>Damit eine Nachricht verschlüsselt werden kann, muss sie zunächst in eine Zahl  $M$  verwandelt werden. Beispielsweise kann ein Wort gemäß ASCII in eine binäre Zahl verwandelt werden, die dann zu Verschlüsselungszwecken als Dezimalzahl  $M$  betrachtet werden kann.

2. Bob sendet seine  $c_i$  an Alice
3. Alice kann die  $c_i$  zu  $m_i$  dechiffrieren mittels

$$m_i \equiv c_i^d \pmod{n}$$

Die Dechiffrierung funktioniert, da

$$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i m_i^{k(p-1)(q-1)} = m_i * 1 = m_i$$

(alles modulo  $n$ )

Die Begründung der Dechiffrierung, d.h. von  $P \equiv (P^e)^d \pmod{n}$ , ist auf den Satz von Fermat zurückzuführen:

$$\begin{aligned} P^{p-1} &\equiv 1 \pmod{p} \\ P^{(p-1)(q-1)} &\equiv 1 \pmod{p} \\ P^{k(p-1)(q-1)} &\equiv 1 \pmod{p} \\ P^{1+k(p-1)(q-1)} &\equiv P \pmod{p} \end{aligned}$$

Und da  $ed \equiv 1 \pmod{(p-1)(q-1)}$ , folgt

$$P^{ed} \equiv P \pmod{p}$$

Dementsprechend gilt ebenso

$$P^{ed} \equiv P \pmod{q}$$

Da  $p$  und  $q$  prim zueinander sind, folgt aus dem Chinesischen Restesatz  $P^{ed} \equiv P \pmod{n}$ . Dies beweist, dass Dechiffrieren eines aus einem Klartext  $P$  erzeugten Chiffrats  $C$  wieder zu  $P$  führt.

Tabelle 5.2 listet zur Übersicht nochmals kurz die wichtigen Variablen und deren Bedeutung im RSA-Verfahren auf.

Variable	Bedeutung	Beschreibung
$n$	Öffentlicher Schlüssel	Produkt zweier Primzahlen $p$ und $q$ (diese müssen geheim bleiben)
$e$	Öffentlicher Schlüssel	relativ prim zu $(p-1)(q-1)$
$d$	Privater Schlüssel	$e^{-1} \pmod{(p-1)(q-1)}$
$c$	Verschlüsselung	$c = m^e \pmod{n}$
$m$	Entschlüsselung	$m = c^d \pmod{n}$

Tabelle 5.2: RSA-Verschlüsselung

## Beispiel

An einem Beispiel wird der Vorgang noch einmal verdeutlicht.

Mit  $p = 47$  und  $q = 71$  gilt

$$n = pq = 3337$$

Der Chiffrierschlüssel  $e$  darf keine gemeinsame Faktoren mit

$$(p-1)(q-1) = 46 * 70 = 3220$$

haben. Wir wählen als Wert für  $e$  (zufällig) 79. Damit gilt

$$d = 79^{-1} \bmod 3320 = 1019$$

Diese Zahl wurde mit dem erweiterten Euklidischen Algorithmus berechnet.  $e$  und  $n$  werden veröffentlicht,  $d$  bleibt geheim.  $p$  und  $q$  werden vernichtet. Zur Verschlüsselung der Nachricht

$$P = 6882326879666683$$

wird diese zuerst in kleine Blöcke zerlegt. Blöcke mit je drei Ziffern funktionieren hier ganz gut. Die Nachricht wird also in sechs Blöcke  $m_i$ <sup>6</sup> zerlegt:

$$m_1 = 688$$

$$m_2 = 232$$

$$m_3 = 687$$

$$m_4 = 966$$

$$m_5 = 668$$

$$m_6 = 003$$

Der erste Block wird wie folgt verschlüsselt:

$$688^{79} \bmod 3337 = 1570 = c_1$$

Führt man diese Operation auch für die folgenden Blöcke aus, erhält man die verschlüsselte Nachricht:

$$C = 1570\ 2756\ 2091\ 2276\ 2423\ 158$$

Zur Entschlüsselung der Nachricht muss man eine ähnliche Potenzierung mit dem Dechiffrierschlüssel, also 1019 durchführen:

$$1570^{1019} \bmod 3337 = 688 = m_1$$

Der Rest der Nachricht wird analog wiederhergestellt.

RSA findet seine Anwendung in diversen Bereichen, wie z.B. der Internet- und Telefonie-Infrastruktur (X.509-Zertifikate), in Übertragungs-Protokollen (IPSec, TLS, SSH, WASTE), bei der E-Mail-Verschlüsselung (PGP, S/MIME) und der Kartenzahlung (EMV<sup>7</sup>).

<sup>6</sup>Sollte der restliche Teil einer Nachricht nicht der Blocklänge entsprechen, so kann mittels Padding der Block mit Nullen von links aufgefüllt werden.

<sup>7</sup>Die Abkürzung EMV bezeichnet eine Spezifikation für Zahlungskarten, die mit einem Prozessorchip ausgestattet sind, und für die zugehörigen Chipkartengeräte (POS-Terminals und Geldautomaten). Die Buchstaben EMV stehen für die drei Gesellschaften, die den Standard entwickelten: Europay, MasterCard und VISA.

### 5.4.3 Sicherheit

Die Sicherheit dieses Verfahrens basiert hauptsächlich auf dem Problem der Faktorisierung großer Zahlen. Natürlich kann ein Kryptoanalytiker jedes mögliche  $d$  probieren, bis er über den korrekten Wert stolpert. Dieser Brute-Force-Angriff ist jedoch noch ineffizienter als der Versuch,  $n$  zu faktorisieren.

Bei der Faktorisierung versucht ein Angreifer, die zusammengesetzte Zahl  $n$  in ihre Primteiler  $p$  und  $q$  zu faktorisieren. Schafft er dies, so ist er ebenso wie Alice in der Lage, mit Hilfe des Erweiterten Euklidischen Algorithmus den geheimen Schlüssel  $d$  und damit auch  $P$  zu bestimmen. Das *Faktorisierungsproblem*, wie es in der Mathematik genannt wird, ist jedoch für große Zahlen  $n$  nicht effizient<sup>8</sup> durchführbar. Der Aufwand zur Faktorisierung steigt in Abhängigkeit der Bit-Breite bzw. Bit-Komplexität von  $n$  in etwa exponentiell an. Die Bit-Breite von  $n = pq$  entspricht hierbei der Summe der Bit-Breiten der beiden gewählten Primzahlen: Da beide Primzahlen gleich groß sein sollten, wählt man für einen 1024-Bit-Schlüssel etwa zwei jeweils  $\frac{1024}{2} = 512$  Bit große Primzahlen. Eine Schlüssellänge von 2048 Bit wird auch für recht lange Zeit noch sicher sein, sofern keine wissenschaftlichen Durchbrüche erzielt werden.

Die Firma RSA Security veranstaltet einen Wettbewerb, die so genannte *RSA Challenge* (<http://www.rsasecurity.com/rsalabs/node.asp?id=2092>), der die Sicherheit von RSA gegenüber Faktorisierung demonstrieren soll. Der Erste, der zu einer von RSA Security veröffentlichten Zahl  $n$  deren Primfaktoren  $p$  und  $q$  angeben kann, gewinnt einen Geldpreis. Am 2. November 2005 ist die bisher größte Zahl  $n$  in ihre beiden Primfaktoren zerlegt worden. Dabei hatte  $n$  eine Größe von 640 Bit; mehr als 5 Monate Arbeit und die Hilfe von ca. 30 2.2GHz-Opteron Prozessoren sind für das Faktorisieren notwendig gewesen - für ein Preisgeld von 20.000 US Dollar.

Die Faktorisierung der Schlüssellängen mit 704 Bit, 768 Bit, 896 Bit, 1024 Bit, 1536 Bit und 2048 Bit stehen noch aus.

---

<sup>8</sup>Das Wort *effizient* meint in diesem Kontext, dass der Angriff weniger Rechenaufwand für einen Angreifer bedeutet als der Aufwand, der für einen Brute-Force-Angriff - das simple Durchprobieren aller möglichen Schlüsselkombinationen - notwendig ist.



## 6 Hybride Verschlüsselungssysteme

Im Allgemeinen kommen zur Verschlüsselung größerer Nutzdatenmengen keine asymmetrischen Verfahren zum Einsatz. Der Aufwand, d.h. die Rechenzeit, die zur Chiffrierung der Daten notwendig ist, unterscheidet sich bei symmetrischen und asymmetrischen Algorithmen enorm. Asymmetrische Verfahren sind bei Schlüssellängen, die ein vergleichbares Sicherheitsniveau (siehe hierzu [RBSM05] Seite 89) bieten, um ein Vielfaches langsamer als symmetrische.

Ebenso ist bei gleicher Sicherheit der Speicherbedarf zum Ablegen einzelner Schlüssel bei asymmetrischen Verfahren wesentlich höher im Gegensatz zu symmetrischen Verfahren.

Sichere Kommunikation mit einem bisher unbekanntem Partner ist jedoch ohne asymmetrische Kryptographie und öffentliche bzw. private Schlüssel nur schwer möglich, da keine gemeinsamen symmetrischen Schlüssel mit dem unbekanntem Gegenüber im Vorfeld ausgetauscht werden konnten. In diesen Situationen hilft die Lösung mit frei herausgebbaren öffentlichen Schlüsseln sehr elegant. In der Praxis werden daher zur Datenverschlüsselung meist so genannte *Hybrid*-Verfahren eingesetzt, eine Mischung aus asymmetrischen und symmetrischen Verfahren. Die Idee solcher Verfahren lässt sich wie folgt beschreiben:

1. Alice möchte an Bob Daten schicken. Ihre erste Aufgabe besteht darin, an den öffentlichen Schlüssel  $e_B$  von Bob zu gelangen. Dies ist z.B. über ein öffentliches Verzeichnis möglich.
2. Alice generiert zufällig einen so genannten *Sitzungsschlüssel*  $k_{AB}$  für ein symmetrisches Verfahren. Dies wird der Schlüssel sein, mit dem später die Nutzdaten zwischen Alice und Bob ausgetauscht werden.
3. Alice verschlüsselt  $C = E_{e_B}(k_{AB})$  (d.h. der Sitzungsschlüssel wird mit Hilfe des öffentlichen Schlüssels von Bob verschlüsselt) und schickt  $C$  an Bob. Bob kann  $C$  jetzt mit seinem privaten Schlüssel  $d_A$  entschlüsseln,  $k_{AB} = D_{d_B}(C)$ , und erhält so  $k_{AB}$ .
4. Sowohl Alice als auch Bob befinden sich nun im Besitz eines gemeinsamen Sitzungsschlüssels  $k_{AB}$ , mit dem sie die Daten chiffrieren und austauschen können.

Ausgetauscht werden also insgesamt der mit dem öffentlichen Schlüssel chiffrierte Sitzungsschlüssel sowie die Daten der Nachricht, die mit dem Sitzungsschlüssel chiffriert

wurden. Dieses Vorgehen kann und sollte bei länger andauernder Kommunikation zwischen Alice und Bob wiederholt werden, d.h. der Sitzungsschlüssel sollte häufiger gewechselt werden.

Abbildung 6.1 zeigt den schematischen Aufbau eines hybriden Verschlüsselungsverfahrens.

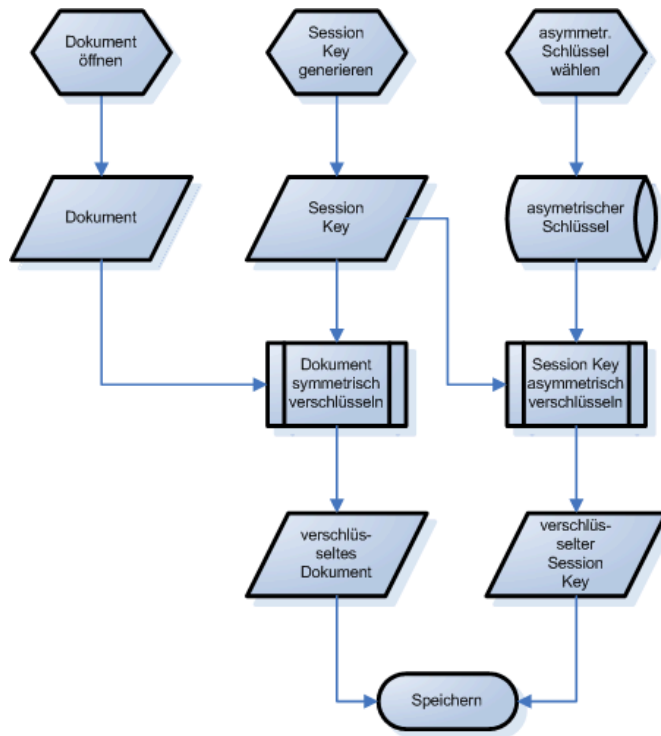


Abbildung 6.1: Beispiel für ein hybrides Verschlüsselungsverfahren

## 6.1 PGP - Pretty Good Privacy

„Es ist persönlich. Es ist vertraulich. Und außer Sie geht es niemanden etwas an. Unter Umständen bereiten Sie derzeit eine politische Kampagne vor, sprechen über Ihre Steuerangelegenheiten oder haben eine geheime Liebesaffäre. Oder aber Sie stehen mit einem politischen Dissidenten in einem autoritären Staat in Kontakt. Worum es sich auch handeln mag, Sie möchten sicherlich nicht, dass eine dritte Person Ihre elektronische Post oder Ihre vertraulichen Dokumente liest. Es ist völlig normal, dass Sie ihre Privatsphäre bewahren möchten. Die Wahrung der Privatsphäre ist genauso selbstverständlich wie die amerikanische Verfassung. [...] PGP ist ein Instrument, mit dem Menschen den Schutz ihrer Privatsphäre in die eigene Hand nehmen können. In unserer Gesellschaft besteht dafür ein wachsendes Bedürfnis. Deshalb habe ich PGP entwickelt.“

Das sind die Eröffnungsworte bzw. die letzten Worte aus einem ergreifenden Plädoyer für die allgemeine Verwendung von Kryptographie von Phil Zimmermann, nachzulesen auf <http://www.philzimmermann.com/DE/essays/index.html> bzw. in [NAI98].

PGP steht für *Pretty Good Privacy*, ein Programm zur Verschlüsselung und Signierung von E-Mail-Nachrichten und Dateien auf dem Computer, welches 1991 von Phillip R. Zimmermann entwickelt wurde. Es ermöglicht symmetrische und asymmetrische Verschlüsselung im oben erwähnten Hybridverfahren zu kombinieren und stellt die Basis für eine Web-of-Trust-PKI<sup>1</sup> dar.

Die Entstehungsgeschichte von PGP ist eine sehr interessante und kann z.B. in [SS02] oder auf Phil Zimmermanns Seite direkt nachgelesen werden. Im Jahre 1996 wurde RFC<sup>2</sup> 1991 publiziert, in dem das binäre Datenformat von PGP beschrieben wird. Seit 1997 bemüht sich die OpenPGP Workgroup der IETF (Internet Engineering Task Force) um die Standardisierung des PGP-Datenformats. Ende 1998 wurde RFC 2440 im Standards Track veröffentlicht.

Als freie PGP-Implementierung für viele Betriebssysteme ist der GNU-Privacy Guard (GPG) verfügbar.

### 6.1.1 Verwendete Algorithmen

PGP verfügt über drei verschiedene symmetrische Blockchiffren, die zur Verschlüsselung der eigentlichen Nachricht eingesetzt werden. Hierbei handelt es sich um CAST, Triple-DES und IDEA. Alle drei Chiffren arbeiten auf der Basis von 64-Bit-Blöcken von Klar- und chiffriertem Text. CAST und IDEA verfügen über Schlüsselgrößen von 128 Bit, während Triple-DES einen 168-Bit-Schlüssel verwendet. Wie der Data Encryption Standard (DES), können alle drei Verschlüsselungsarten in den Modi Cipher Feedback (CFB) oder Cipher Block Chaining (CBC) verwendet werden. Sie werden von PGP im 64-Bit-CFB-Modus verwendet.

Der CAST-Verschlüsselungsalgorithmus wurde in PGP aufgenommen, da es sich dabei um einen sehr schnellen und kostenfreien 128-Bit-Blockchiffrierer handelt. Der Name dieses Algorithmus wurde aus den Anfangsbuchstaben seiner Entwickler abgeleitet, Carlisle Adams und Stafford Tavares. Es sprechen viele Argumente dafür, dass CAST immun gegen lineare und differentielle Kryptoanalyse, die in der Fachliteratur allgemein als die leistungsfähigsten Kryptoanalyseformen gelten, ist.

Die Blockchiffre IDEA (International Data Encryption Algorithm) basiert auf dem Entwicklungskonzept, Vorgänge von verschiedenen algebraischen Gruppen zu mischen. Der Algorithmus wurde von James L. Massey und Xuejia Lai an der ETH in Zürich entwickelt und 1990 veröffentlicht. Auch IDEA widersteht den höchst erfolgreichen differential-

---

<sup>1</sup>PKI steht für *Public Key Infrastructure* und bezeichnet eine Infrastruktur zum Management von ID-Zertifikaten. Sie ermöglicht die Authentifikation der Bindung eines öffentlichen Schlüssels an einen Namen. Für weitere Details zu PKI ist die Lektüre von [RBSM05] und [BS96] zu empfehlen. Weiterhin setzt sich [BSCE00] kritisch mit den Sicherheitsrisiken einer PKI auseinander.

<sup>2</sup>RFC steht für Request For Comments (zu deutsch Aufforderung zu Kommentaren). Dies sind eine Reihe von technischen und organisatorischen Dokumenten des RFC-Editor zum Internet.

kryptoanalytischen Angriffen von Biham und Shamir<sup>3</sup> sowie Attacken durch lineare Kryptoanalyse (besser als DES).

Als Schutz enthält PGP im Repertoire seiner Blockchiffren Triple-DES. Triple-DES bedeutet, dass DES dreimal auf den gleichen Datenblock angewandt wird. Dabei werden drei verschiedene Schlüssel verwendet. Triple-DES ist zwar deutlich langsamer als CAST oder IDEA, jedoch ist die Geschwindigkeit für E-Mail-Anwendungen normalerweise nicht von großer Bedeutung.

Die öffentlichen Schlüssel, die mit PGP erzeugt werden, enthalten eingebettete Daten, die dem Absender mitteilen, welche Blockchiffren von der Empfängersoftware verstanden werden, so dass die Software des Absenders „weiß“, welche Chiffriercodes zum Verschlüsseln verwendet werden können. Die öffentlichen Diffie-Hellman/DSS-Schlüssel<sup>4</sup> akzeptieren CAST, IDEA oder Triple-DES als Blockchiffre, mit CAST als Standardeinstellung. Zum Senden von Nachrichten an RSA-Schlüssel wird von PGP nur die IDEA-Verschlüsselung verwendet, da ältere PGP-Versionen nur RSA und IDEA unterstützen.

### 6.1.2 PGP-Datenkomprimierungsroutinen und verwendete Zufallszahlen

PGP komprimiert normalerweise den Klartext vor der Verschlüsselung, da der Klartext nach der Verschlüsselung nicht mehr komprimiert werden kann. Durch Datenkomprimierung wird die Übertragungszeit verringert sowie Platz auf der Festplatte gespart und, was weitaus wichtiger ist, die kryptographische Sicherheit gesteigert. Die meisten kryptoanalytischen Verfahren nutzen im Klartext gefundene Wiederholungen zum Decodieren der Chiffre. Durch Datenkomprimierung wird diese Redundanz im Klartext reduziert, wodurch der Schutz vor kryptoanalytischen Angriffen deutlich vergrößert wird. Die Komprimierung des Klartextes bedeutet einen zusätzlichen Zeitaufwand, doch vom Sicherheitsstandpunkt aus gesehen lohnt sich der Aufwand. Dateien, die zu kurz sind oder die nicht gut komprimiert werden können, werden von PGP nicht komprimiert. Das Programm verwendet die Freeware-ZIP-Komprimierungsroutinen.

PGP verwendet zur Erstellung von temporären Sitzungsschlüsseln einen kryptographisch leistungsfähigen Generator für Pseudo-Zufallswerte. Diese Zufallswerte werden durch das PGP-Programm aus Zufallsereignissen auf der Grundlage der zeitlichen Koordination von Tastaturbetätigungen und Mausbewegungen abgeleitet. Hier wird also kein Pseudozufallszahlen-Generator verwendet, welcher eine Sicherheitslücke darstellen würde, sondern es kommen echte Zufallszahlen zum Einsatz.

---

<sup>3</sup>„Differential Cryptanalysis of the Data Encryption Standard“, Eli Biham und Adi Shamir. Springer-Verlag. ISBN: 0-387-97930-1. In diesem Buch wird die Differentialkryptoanalyse auf DES angewandt und erläutert. Das Buch eignet sich besonders zum Kennenlernen dieses Verfahrens.

<sup>4</sup>Der Digital Signature Algorithm ist ein Standard der US-Regierung für Digitale Signaturen. Er wurde vom National Institute of Standards and Technology (NIST) im August 1991 für die Verwendung in deren Digital Signature Standard (DSS) empfohlen.

### 6.1.3 Der Nachrichtenkernel

Der Nachrichtenkernel ist die kompakte „Essenz“ von 160 oder 128 Bit Größe einer Nachricht oder eine Dateiprüfsumme. Man kann ihn sich als „Fingerabdruck“ der Nachricht oder der Datei vorstellen. Der Nachrichtenkernel „repräsentiert“ eine E-Mail-Nachricht. Wenn die Nachricht in irgendeiner Form verändert wird, ändert sich auch der aus ihr berechnete Nachrichtenkernel. Dadurch können alle von einem Fälscher an der Nachricht vorgenommenen Änderungen aufgedeckt werden. Der Nachrichtenkernel wird mit Hilfe einer kryptographisch leistungsfähigen, einseitigen Hash-Funktion der Nachricht berechnet. Für einen Angreifer sollte es rechnerisch unmöglich sein, eine Ersatznachricht zu erstellen, die einen identischen Nachrichtenkernel erzeugen würde. In dieser Hinsicht ist ein Nachrichtenkernel sehr viel besser als eine Prüfsumme, da es einfach ist, eine Nachricht mit anderem Inhalt zu erstellen, die dieselbe Prüfsumme erzeugt. Wie bei einer Prüfsumme kann man die Originalnachricht jedoch nicht von ihrem Nachrichtenkernel herleiten.

Der Nachrichtenkernel wird mit dem privaten Schlüssel des Absenders - nennen wir ihn (bzw. sie) Alice - verschlüsselt. Der Empfänger - hier Bob - der Nachricht kann mit dem öffentlichen Schlüssel von Alice den Nachrichtenkernel entschlüsseln und mit dem Hash-Wert, den er selbst für die empfangene Nachricht errechnet hat, vergleichen. Stimmen diese beiden Werte überein, so kann der Empfänger sicher sein, dass die Nachricht nicht von einem unbefugten Dritten verändert wurde. Zugleich kann bei diesem Schritt aber auch noch eine weitere wünschenswerte Eigenschaft erfüllt werden: Da zu einem öffentlichen Schlüssel immer nur genau ein privater Schlüssel gehört, kann Bob beim Entschlüsseln des Nachrichtenkernels auch noch gleich die Authentizität von Alice feststellen - denn wäre der öffentliche Schlüssel nicht zum privaten Schlüssel von Alice gehörig, so könnte der Nachrichtenkernel gar nicht erst entschlüsselt werden. Ein Mechanismus zur Erzeugung einer fälschungssicheren Unterschrift hat z. B. verschiedene Vorteile für die Strafverfolgung und das Sammeln von Beweisen.

Ob nun aber der öffentliche Schlüssel von Alice auch wirklich Alice gehört, oder ob jemand Drittes - Eve die Lauscherin - sich für Alice ausgibt, ist wiederum ein anderes Problem, welches hier aber nicht näher erläutert wird. [BSCE00] geht auf die Schwachstellen einer PKI genauer ein. Auch [BS96] erläutert das Konzept und die möglichen Probleme der Web-of-Trust-PKI sehr anschaulich.

Abbildung 6.2 veranschaulicht den Vorgang der Signatur mit Hilfe eines Hash-Algorithmus.

Der derzeit in PGP<sup>5</sup> verwendete Nachrichtenkernelalgorithmus wird SHA (Secure Hash Algorithm) genannt und erzeugt eine 160-Bit-Ausgabe.

### 6.1.4 Weiteres Wissenswertes

PGP sieht kein spezielles Verfahren vor, um Vertrauen herzustellen: die Benutzer entscheiden selbst, wen sie für zuverlässig halten und wen nicht. PGP stellt Mechanismen

---

<sup>5</sup>Die derzeit aktuellste Version der PGP Corp. ist die Version 9.0. Eine Alternative zu PGP von PGP Corp. ist die Open-Source Software Gnu Privacy Guard (GnuPG), aktuell in der Version 1.4.3. Siehe hierzu <http://www.gnupg.org/>

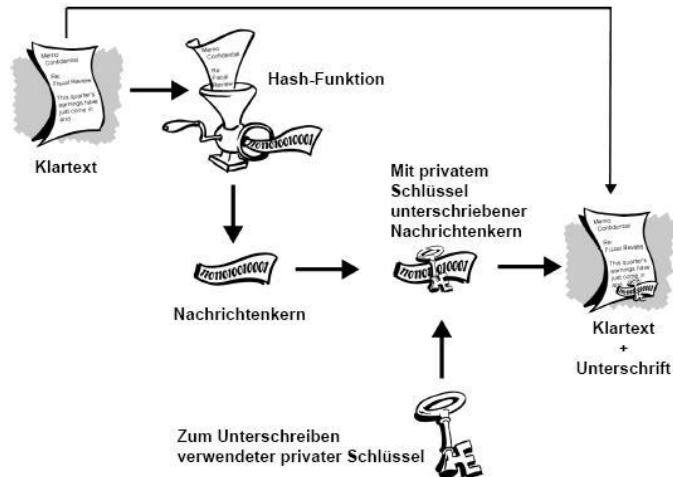


Abbildung 6.2: Sichere digitale Unterschriften

bereit, um Vertrauenswürdigkeit an öffentliche Schlüssel zu koppeln und mit Vertrauen zu arbeiten. Die Funktionsweise des Netz des Vertrauens bzw. Web of Trust ist in der Kryptologie die Idee, die Echtheit von digitalen Schlüsseln durch ein Netz von gegenseitigen Bestätigungen (Signaturen) zu sichern. Es stellt eine dezentrale Alternative zum hierarchischen PKI-System dar. Dies ist eine sehr interessante Thematik und könnte hier noch sehr erschöpfend erörtert werden. Dennoch würde dies den Rahmen dieser Ausarbeitung bei weitem sprengen, und so sei der interessierte Leser auf die Lektüre von [RBSM05] bzw. [BS96] verwiesen.

## 7 Anhang A - Primitive Elemente

Primzahlen haben nur die „1“ und sich selbst als Teiler. Ist  $p$  eine Primzahl, so bildet  $M = \{1, 2, \dots, p-1\}$  die Menge der „teilerfremden“ Elemente von  $p$ .

Am Beispiel der Primzahl  $p = 5$  werden ihr die Elemente  $\{1, 2, 3, 4\}$  zugeordnet. Der Primzahl  $p = 11$  die Elemente  $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ .

Unter diesen Elementen gibt es nun eine Zahl  $a$ , welche durch die Rechnung

$$a^i \bmod p \quad (0 \leq i \leq p-2)$$

alle Elemente aus  $M$  darstellen kann. Dieses  $a$  nennt sich dann *primitives Element*. Dies bedeutet also, dass kein Element in dieser Rechnung doppelt vorkommen darf.

Da nun  $a^0 = 1$  (für jedes  $a$ ) ist, so ist auch  $a^0 \bmod p = 1$ . Kommt die „1“ bei obiger Rechnung noch einmal vor, so handelt es sich *nicht* mehr um ein primitives Element. Anders ausgedrückt: die Reihe  $a^i \bmod p$  ( $0 \leq i \leq p-2$ ) muss stets verschiedene Elemente ergeben, wobei alle „teilerfremden Elemente“ von  $p$  dargestellt werden müssen.

Man kann diese Reihe auch folgendermaßen aufschreiben:

$$\{a^0 \bmod p, a^1 \bmod p, a^2 \bmod p, \dots, a^{p-2} \bmod p\}$$

### Ein Beispiel:

Nehmen wir die Primzahl  $p = 5$ . Die Menge der teilerfremden Elemente ist  $\{1, 2, 3, 4\}$ . Nun müssen wir für unser  $a$  zuerst die „1“, dann die „2“, dann die „3“ und zuletzt die „4“ nehmen. Dann rechnen wir  $a^0, a^1, a^2, a^3$  stets  $\bmod p$  aus. Wird dabei die Menge  $\{1, 2, 3, 4\}$  dargestellt, so haben wir ein primitives Element gefunden.

### Also:

$a = 1$ :

$$\begin{aligned} a^0 \bmod p &= 1^0 \bmod 5 = 1 \\ a^1 \bmod p &= 1^1 \bmod 5 = 1 \end{aligned}$$

Die „1“ kommt zweimal vor  $\Rightarrow a = 1$  ist kein primitives Element.

$a = 2$ :

$$\begin{aligned} a^0 \bmod p &= 2^0 \bmod 5 = 1 \\ a^1 \bmod p &= 2^1 \bmod 5 = 2 \\ a^2 \bmod p &= 2^2 \bmod 5 = 4 \\ a^3 \bmod p &= 2^3 \bmod 5 = 3 \end{aligned}$$

Man sieht nun, dass alle teilerfremden Elemente dargestellt werden. Somit ist „2“ ein primitives Element von  $p = 5$ .

$a = 3$  :

$$\begin{aligned}a^0 \bmod p &= 3^0 \bmod 5 = 1 \\a^1 \bmod p &= 3^1 \bmod 5 = 3 \\a^2 \bmod p &= 3^2 \bmod 5 = 4 \\a^3 \bmod p &= 3^3 \bmod 5 = 2\end{aligned}$$

Auch hier wird die Menge  $\{1, 2, 3, 4\}$  abgebildet. Somit ist auch „3“ ein primitives Element von  $p = 5$ .

$a = 4$  :

$$\begin{aligned}a^0 \bmod p &= 4^0 \bmod 5 = 1 \\a^1 \bmod p &= 4^1 \bmod 5 = 4 \\a^2 \bmod p &= 4^2 \bmod 5 = 1\end{aligned}$$

Die „1“ kommt hier abermals zweimal vor  $\Rightarrow a = 4$  ist ebenfalls kein primitives Element.

Fassen wir alles zusammen, so stellen wir fest, dass die Primzahl  $p = 5$  die Menge der primitiven Elemente  $\{2, 3\}$  besitzt.

# Literaturverzeichnis

- [BS96] B. Schneier. *Applied Cryptography*. Addison-Wesley, 1996.
- [BSCE00] B. Schneier. *Ten Risks of PKI*. Computer Security Journal, Volume XVI, Number 1, 2000.
- [DK96] D. Kahn. *The Codebreakers*. Scribner Book Company, 1996.
- [MB04] M. Bishop. *Computer Security – Art and Science*. Addison-Wesley, 2004.
- [NAI98] Network Associates International BV *Einführung in die Kryptographie* Network Associates International, 1998. Heruntergeladen von <http://gd.tuwien.ac.at/privacy/crypto/kryptografie.pdf>
- [PZPGP] <http://www.philzimmermann.com>
- [RBSM05] R. Bless, S. Mink, et al. *Sichere Netzwerkkommunikation*. Springer-Verlag, 2005
- [RHUB05] Prof. Rolf Haenni *Einführung in die Kryptographie - Vorlesungsskript*. Universität Bern, Juli 2005.
- [SS02] S. Singh. *Codes - Die Kunst der Verschlüsselung*. dtv-Verlag, 2002.
- [WikiDE] <http://de.wikipedia.org>
- [WikiEN] <http://en.wikipedia.org>