

Kryptoanalyse



Ariane Sutor
Frühjahr 2003

Inhaltsverzeichnis

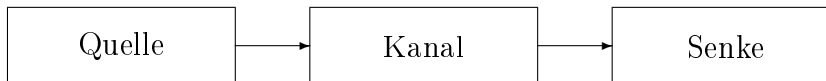
1	Kryptoanalyse	4
1.1	Kryptosysteme	4
1.2	Grundbegriffe und Angriffsarten	6
1.3	Einige grundlegende Methoden	7
2	Mathematische Grundlagen	9
2.1	Der größte gemeinsame Teiler	9
2.2	Die Gruppen $(\mathbb{Z}_n, +)$ und $(\mathbb{Z}_n^*, \cdot)$	10
3	Klassische Chiffren	12
3.1	Caesar-Addition	12
3.2	Monoalphabetische Substitutionen	12
3.3	Das Vigenère-Verfahren	16
4	DES	19
4.1	Beschreibung des Algorithmus	19
4.1.1	Klartext und Schlüsselraum	19
4.1.2	Die initiale Permutation	20
4.1.3	Die interne Blockchiffre	20
4.1.4	Die S-Boxen	21
4.1.5	Weitere Runden der internen Blockchiffre	23
4.1.6	Die Rundenschlüssel	23
4.2	Entschlüsselung	24
4.3	Differentielle Kryptoanalyse des DES	25
4.4	Betriebsmodi für Blockchiffren	27
4.4.1	ECB-Mode	27
4.4.2	CBC-Mode	28

<i>INHALTSVERZEICHNIS</i>	3
5 Public Key Kryptographie	30
5.1 Der man-in-the-middle-Angriff	32
5.2 Erzeugung großer Primzahlen	33
5.3 Das RSA-Verfahren	34
5.3.1 Schlüsselerzeugung, Verschlüsselung und Entschlüsselung .	34
5.3.2 RSA Verschlüsselung als Blockchiffre	36
5.3.3 Angriffe auf RSA	37
5.4 Diffie-Hellman-Schlüsselaustausch	39
6 Faktorisierung	41
6.1 Probedivision	41
6.2 Das quadratische Sieb	41
7 Diskrete Logarithmen	45
7.1 Enumeration	45
7.2 Der Babystep-Giantstep-Algorithmus	45
8 Komplexitätstheorie	47

1 Kryptoanalyse

1.1 Kryptosysteme

Wir betrachten ein einfaches Nachrichtenübertragungssystem:



Eine Nachricht entsteht immer in der Quelle und hat das Ziel, der Senke etwas mitzuteilen. Daher besteht eine Nachricht aus einer Darstellungs- und einer Bedeutungskomponente. Die Bedeutungskomponente repräsentiert den Inhalt der Nachricht, während die Darstellungskomponente die Darstellung der Nachricht repräsentiert.

Von der Nachricht zu unterscheiden ist der Begriff der Information. Information entsteht immer in der Senke und zwar genau dann, wenn der Inhalt der Nachricht der Senke nicht vollständig bekannt war.

Die Kryptographie beschäftigt sich mit der Darstellung einer Nachricht. Sie entwickelt **Kryptosysteme**, um Nachrichten geheimzuhalten. Ein Kryptosystem wird immer dann benötigt, wenn Nachrichten auf unsicheren Kanälen übermittelt werden. Ein unsicherer Kanal ermöglicht es unbefugten Dritten, die übermittelten Nachrichten ebenfalls zu empfangen, vielleicht sogar abzufangen und verfälscht weiterzuleiten. Ein Kanal ist zum Beispiel auch dann unsicher, wenn er es Dritten ermöglicht, dem Absender oder Empfänger eine falsche Identität vorzuspiegeln.

Die klassischen Anwendungsbereiche von Kryptosystemen sind Militär, Diplomatie und Geheimdienste. Inzwischen sind wir alle zu Anwendern von Kryptosystemen geworden, beispielsweise wenn wir im Internet einkaufen, beim Login in einem Computer-Netzwerk, bei der Nutzung von Geldautomaten, beim Online-Banking und beim Telefonieren mit Mobiltelefonen. In all diesen Fällen ist zumindest eine Feststellung der Identität des Nutzers von entscheidender Bedeutung. Soweit Daten übertragen werden, ist es auch deren Geheimhaltung.

Das folgende Diagramm beschreibt das Zusammenwirken der Komponenten eines Kryptosystems (oder einer Chiffre):



Wir geben eine formale Definition eines Kryptosystems:

Definition 1.1 Ein **Verschlüsselungsverfahren** oder ein **Kryptosystem** ist ein Quintupel $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ mit folgenden Eigenschaften:

1. $\mathcal{P}$ ist eine Menge. Sie heißt Klartextrraum. Ihre Elemente heißen Klartexte.
2. $\mathcal{C}$ ist eine Menge. Sie heißt Geheimtextraum. Ihre Elemente heißen Geheimtexte.
3. $\mathcal{K}$ ist eine Menge. Sie heißt Schlüsselraum. Ihre Elemente heißen Schlüssel.
4. $\mathcal{E} = \{E_k : k \in \mathcal{K}\}$ ist eine Familie von Funktionen $E_k : \mathcal{P} \rightarrow \mathcal{C}$. Ihre Elemente heißen Verschlüsselungsfunktionen.
5. $\mathcal{D} = \{D_k : k \in \mathcal{K}\}$ ist eine Familie von Funktionen $D_k : \mathcal{C} \rightarrow \mathcal{P}$. Ihre Elemente heißen Entschlüsselungsfunktionen.
6. Für jedes $e \in \mathcal{K}$ gibt es ein $d \in \mathcal{K}$, so daß für alle $p \in \mathcal{P}$ die Gleichung $D_d(E_e(p)) = p$ gilt.

Die gewählten Bezeichnungen entsprechen den englischen Wörtern: $\mathcal{P}$ wie plaintext, $\mathcal{C}$ wie ciphertext, $\mathcal{K}$ wie key, $\mathcal{E}$ wie enciphering, $\mathcal{D}$ wie deciphering.

Die zu verschlüsselnden Klartexte sind meistens Zeichenfolgen, die aus Zeichen eines gewissen Alphabets bestehen. Beispielsweise können bei einem Kryptosystem die Klartexte Zeichenfolgen (Wörter) sein, die aus Zeichen des deutschen Alphabets $\{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z\}$ zusammengesetzt sind. Häufig identifiziert man die Buchstaben eines Alphabets, das die Klartexte eines Kryptosystems bildet, mit gewissen natürlichen Zahlen. Als Beispiel betrachten wir den ASCII-Zeichensatz. Der erweiterte ASCII-Zeichensatz besteht aus 256 Zeichen. Jedem Zeichen wird eine Zahl zwischen 0 und 255 zugeordnet. Die folgende Tabelle enthält einen Ausschnitt aus dem ASCII-Zeichensatz:

32		33	!	34	"	35	#	36	\$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124		125	}	126	~	127	DEL

1.2 Grundbegriffe und Angriffsarten

Kryptoanalyse ist die Lehre von den Angriffen auf Kryptosysteme. Ziel der Kryptoanalyse ist es, ohne Kenntnis des geheimen Schlüssels möglichst viele Informationen über den Klartext zu gewinnen.

Kryptoanalyse ist nicht nur für den feindlichen Spion interessant. Jeder Nutzer eines Kryptosystems sollte abschätzen können, ob seine Daten tatsächlich sicher sind vor unbefugtem Zugriff. Man muss deshalb wissen, was ein geschickter Kryptoanalytiker herausfinden kann. Insbesondere muss jeder, der ein Kryptosystem entwirft, etwas von Kryptoanalyse verstehen.

Um Angriffe auf Verschlüsselungsverfahren zu erschweren, kann man das verwendete Verfahren geheimhalten. Die Sicherheit, die man daraus gewinnen kann, ist aber sehr zweifelhaft. Ein Angreifer hat nämlich viele Möglichkeiten zu erfahren, welches Kryptosystem verwendet wird: Er kann beobachten, welche technischen Hilfsmittel zur Verschlüsselung benutzt werden oder jemand, der früher mit dem Verfahren gearbeitet hat, gibt die Information preis. Es ist also völlig unklar, ob es gelingen kann, das verwendete Kryptosystem wirklich geheimzuhalten.

Aus diesen Gründen bezieht die moderne Kryptoanalyse einen anderen Standpunkt, den auch wir einnehmen. Sie geht davon aus, daß jeder weiß, welches Kryptosystem verwendet wird (**Kerkhoffsches Prinzip**). Nur der verwendete Schlüssel und die verschlüsselten Klartexte werden als geheim angenommen.

Unter der Voraussetzung des bekannten Verfahrens ist es sicherlich der größte Erfolg des Kryptoanalytikers, wenn er den Schlüssel gewinnt. Das hat den Vorteil, daß er ohne weitere Arbeit alle chiffrierten Nachrichten genauso schnell mitlesen kann wie der Empfänger selbst, solange sich der Schlüssel nicht ändert. Der "nächstkleinere" Erfolg ist die Ermittlung des Klartexts.

Im folgenden klassifizieren wir grob verschiedene Typen von Angriffen:

Geheimtext-Angriff (Ciphertext-only attack) Der Angreifer kennt eine begrenzte Anzahl von Geheimtexten und möchte daraus die zugehörigen Klartexte bzw. den verwendeten Schlüssel bestimmen. Sind keine weiteren Gesetzmäßigkeiten bekannt, so bleibt nur der **Brute-Force-Angriff** (Angriff mit Brachialgewalt), d. h. das Ausprobieren aller möglichen Schlüssel. Wenn der Geheimtext-Angriff erfolgreich sein kann, so ist ein Kryptosystem total unsicher. Viele klassische Chiffren sind durch Geheimtext-Angriffe zu brechen.

Klartext-Angriff (Known-plaintext attack) Der Angreifer kennt eine begrenzte Anzahl von Geheimtexten mit den zugehörigen Klartexten und möchte daraus den Schlüssel bzw. den Klartext zu einem weiteren Geheimtext ermitteln. Die Kenntnis von Klartext-Geheimtext Paaren ergibt sich z. B., wenn gewisse Wörter im Text vermutet werden, etwa Grußformeln am Anfang des Textes. Von

modernen Chiffrierverfahren wird erwartet, daß sie Klartext-Angriffen widerstehen.

Angriff mit ausgewähltem Klartext (Chosen-plaintext attack) Der Angreifer kann den Geheimtext zu von ihm gewählten Klartexten erzeugen. Im Falle eines symmetrischen Verfahrens braucht er dafür einen James Bond, der dem Chiffrierer den gewünschten Klartext unterschreibt. Public-Key-Verfahren sind jedoch geradezu gekennzeichnet durch diese Möglichkeit. Sie müssen natürlich gegen Angriffe mit ausgewähltem Klartext sicher sein.

1.3 Einige grundlegende Methoden

Es gibt keine allgemeine Theorie, wie man ein gutes Kryptosystem entwirft und ebensowenig eine allgemeine Theorie, wie man es bricht. Ein Brute-Force-Angriff ist natürlich immer möglich. Deshalb ist es ein Erfolg für den Kryptographen, wenn dem Kryptoanalytiker kein anderer Angriff gelingt.

Wie schwierig ein Brute-Force-Angriff ist, hängt von der Größe des Schlüsselraums und der Geschwindigkeit der Entschlüsselung ab. Wenn man die Schlüssel per Hand durchprobieren muss, können schon tausend Schlüssel recht aufwendig sein. Heute werden zum Entschlüsseln Computer eingesetzt. Da diese immer schneller werden, müssen auch die Schlüsselräume immer größer werden.

Für die Kryptoanalyse ist es wichtig, schnell zu entscheiden, ob ein probeweise entschlüsselter Text der gesuchte Klartext sein kann. Hierzu sollte man in Erfahrung bringen, in welcher Form der gesuchte Klartext gegeben ist, etwa in welcher Sprache und in welchem Datei-Format. Ein schneller Test auf möglichen Klartext kann dann zunächst nach verbotenen Zeichen suchen, anschließend Buchstabenhäufigkeiten auszählen, dann nach verbotenen Bigrammen suchen und schließlich die Wörter des Texts mit einem Wörterbuch vergleichen. Die verbleibenden Texte, die alle Testabschnitte bestanden haben, müssen schließlich persönlich darauf geprüft werden, ob sie sinnvoll sind.

Speziell bei Kryptosystemen, die zeichenweise verschlüsseln, sind statistische Methoden ein wichtiges Mittel der Kryptoanalyse. Beispielsweise kommen Buchstaben in natürlichen Sprachen nicht gleich häufig vor. Die folgende Tabelle gibt die Häufigkeit der Buchstaben in der deutschen und in der englischen Sprache an.

Buchstabe	Häufigkeit in %	
	deutsch	englisch
a	6,5	8,2
b	1,9	1,5
c	3,1	2,8
d	5,1	4,3
e	17,4	12,7
f	1,7	2,2
g	3,0	2,0
h	4,8	6,1
i	7,5	7,0
j	0,3	0,2
k	1,2	0,8
l	3,4	4,0
m	2,5	2,4

Buchstabe	Häufigkeit in %	
	deutsch	englisch
n	9,8	6,7
o	2,5	7,5
p	0,8	1,9
q	0,02	0,1
r	7,0	6,0
s	7,3	6,3
t	6,2	9,1
u	4,4	2,8
v	0,7	1,0
w	1,9	2,4
x	0,03	0,2
y	0,04	2,0
z	1,1	0,1

Chiffrierfehler bilden einen wichtigen Ansatz für die Kryptoanalyse. Ungeschickte, vorhersehbare Schlüsselwahl kann den theoretischen Schlüsselraum stark eingrenzen. Der Brute-Force-Angriff wird dann viel leichter. Beispielsweise wird beim Login in einem Computer-Netzwerk sehr oft die Aufforderung "Passwort eingeben" wörtlich genommen und ein echtes Pass-*Wort* eingegeben.

Ein Klartext-Angriff wird durch das Verwenden stereotyper Formulierungen oder das wiederholte Senden von wenig geändertem Klartext ermöglicht. Dies kann beispielsweise passieren, wenn in einem Unternehmen Dokumente nach vorgegebenen Richtlinien verfasst werden.

2 Mathematische Grundlagen

Per aspera ad astra

2.1 Der größte gemeinsame Teiler

Um kryptographische Verfahren darstellen und verstehen zu können, benötigen wir einige mathematische Grundlagen, insbesondere aus der Zahlentheorie.

Wir bezeichnen mit $\mathbb{N}$ die Menge der natürlichen Zahlen und mit $\mathbb{Z}$ die Menge der ganzen Zahlen:

$$\mathbb{N} = \{1, 2, 3, \dots\}, \quad \mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, 3, \dots\}.$$

Wir sagen $a \in \mathbb{Z}$ **teilt** $b \in \mathbb{Z}$ (und schreiben dafür $a|b$), wenn es ein $c \in \mathbb{Z}$ gibt mit $b = a \cdot c$. Dann heißt a ein Teiler von b .

Seien a, b ganze Zahlen, die nicht beide Null sind. Eine natürliche Zahl g heißt **größter gemeinsamer Teiler** von a und b , falls die folgenden Bedingungen erfüllt sind:

1. $g|a$ und $g|b$,
2. Gilt für ein $d \in \mathbb{Z}$ sowohl $d|a$ als auch $d|b$, so folgt $d|g$.

Je zwei ganze Zahlen, die nicht beide gleich Null sind, besitzen genau einen größten gemeinsamen Teiler g . Wir schreiben

$$g =: \text{ggT}(a, b).$$

Der größte gemeinsame Teiler zweier ganzer Zahlen kann effizient berechnet werden, und zwar mit dem **euklidischen Algorithmus**. Er wie funktioniert nach dem folgendem Prinzip:

Satz 2.1 Seien $a, b \in \mathbb{Z}$ mit $b > 0$. Dann gibt es eindeutig bestimmte ganze Zahlen q und r , für die gilt

- $a = b \cdot q + r$ ("a geteilt durch b ergibt q mit Rest r"),
- $0 \leq r < b$,

nämlich $q = \lfloor a/b \rfloor$ und $r = a - bq$. Mit obigen Bezeichnungen gilt dann $\text{ggT}(a, b) = \text{ggT}(b, r)$.

Das bedeutet: Wir können das Problem der Berechnung des größten gemeinsamen Teilers von a und b zurückführen auf die Berechnung des größten gemeinsamen

Teilers von b und r . Wenn wir $ggT(a, b)$ ausrechnen wollen, führen wir die obige Reduktion so lange durch, bis sich als Rest 0 ergibt:

$$ggT(a, b) = \dots = ggT(c, 0) = c.$$

In der Kryptographie ist die **Vielfachsummendarstellung** des größten gemeinsamen Teilers von besonderer Bedeutung:

Satz 2.2 *Sei $g = ggT(a, b)$. Dann gibt es ganze Zahlen s und t mit $g = sa + tb$.*

Zur Berechnung von s und t benutzt man eine erweiterte Version des euklidischen Algorithmus.

2.2 Die Gruppen $(\mathbb{Z}_n, +)$ und $(\mathbb{Z}_n^*, \cdot)$

In der Situation von Satz 2.1 nennt man q den (ganzzahligen) Quotient und r den Rest der Division von a durch b . Wir schreiben

$$r = a \mod b.$$

Mit dem Operator $\mod$ definieren wir eine modulare Arithmetik. Dazu betrachten wir die Menge der natürlichen Zahlen zusammen mit der Null, die kleiner als n sind für ein $n \in \mathbb{N}$:

$$\mathbb{Z}_n := \{0, 1, \dots, n-1\}.$$

In dieser Menge können wir "addieren", "subtrahieren" und "multiplizieren": Seien $a, b \in \mathbb{Z}_n$. Dann ist die Summe $a \oplus b$, die Differenz $a \ominus b$ und das Produkt $a \otimes b$ wie folgt definiert:

$$\begin{aligned} a \oplus b &:= (a + b) \mod n, \\ a \ominus b &:= (a - b) \mod n, \\ a \otimes b &:= (a \cdot b) \mod n. \end{aligned}$$

Die Struktur $\mathbb{Z}_n$ zusammen mit $\oplus$ hat viele Eigenschaften, die vom gewöhnlichen Rechnen in $\mathbb{Z}$ bekannt sind:

- Die Summe $a \oplus b$ ist wieder ein Element von $\mathbb{Z}_n$.
- Die Operation $\oplus$ ist assoziativ, d. h. für alle $a, b, c \in \mathbb{Z}_n$ gilt

$$(a \oplus b) \oplus c = a \oplus (b \oplus c).$$

- Es gibt ein neutrales Element, nämlich das Element 0.

- Jedes Element $a \in \mathbb{Z}_n$ hat ein Inverses, nämlich $n - a$. Es gilt

$$a \oplus (n - a) = a + (n - a) \mod n = n \mod n = 0.$$

- $\oplus$ ist kommutativ: Es gilt $a \oplus b = b \oplus a$.

In der Mathematik fasst man obige Eigenschaften in einer Definition zusammen und nennt eine Menge mit einer Rechenoperation, die sie erfüllt, eine **kommutative Gruppe**. $(\mathbb{Z}_n, \oplus)$ ist eine kommutative Gruppe.

Die Menge $\mathbb{Z}_n$ ist zusammen mit der Multiplikation $\otimes$ in der Regel keine Gruppe, selbst wenn man das Nullelement ausnimmt. Aber die Teilmenge derjenigen Elemente aus $\mathbb{Z}_n$, die teilerfremd zu n sind, bildet eine multiplikative Gruppe. Diese Menge wird mit $\mathbb{Z}_n^*$ bezeichnet:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \text{ggT}(a, n) = 1\}.$$

Das neutrale Element der kommutativen Gruppe $(\mathbb{Z}_n^*, \otimes)$ ist das Element 1. Das multiplikative Inverse eines Elements $a \in \mathbb{Z}_n^*$ kann mit Hilfe der Vielfachsummen-darstellung berechnet werden. Es gilt nämlich:

Satz 2.3 *Ist $\text{ggT}(a, n) = 1 = sa + tn$, so ist $a' := s \mod n$ das multiplikative Inverse von a in $\mathbb{Z}_n^*$.*

Die Anzahl der Elemente von $\mathbb{Z}_n^*$ wird mit $\varphi(n)$ bezeichnet (**Eulersche φ -Funktion**):

$$\varphi : \mathbb{N} \rightarrow \mathbb{N}, \quad n \mapsto |\{a \leq n \mid \text{ggT}(a, n) = 1\}|.$$

Aus Bequemlichkeitsgründen schreiben wir in Zukunft $+$ statt $\oplus$, $-$ statt $\ominus$, $\cdot$ statt $\otimes$.

3 Klassische Chiffren

3.1 Caesar-Addition

Schon in der Antike machte man sich Gedanken, wie man Nachrichten versenden kann, ohne dass sie von Unbefugten gelesen werden. Caesar fand folgende Lösung: Jedem Buchstaben der Nachricht wird als Geheimtext der Buchstabe zugeordnet, der sich drei Positionen weiter hinten im Alphabet befindet.

Klartext	VENI	VIDI	VICI
Geheimtext	YHQL	YLGL	YLFL

Ob die antiken Kryptoanalytiker Schwierigkeiten mit dem Brechen dieser Chiffre hatten, ist der Autorin unbekannt, ein heutiger Angreifer wird allerdings den Klartext mühelos finden. Trotzdem ist die Grundidee des Verfahrens nicht schlecht und wir wollen es in die Sprache der Mathematik übersetzen, um damit rechnen zu können. Dazu werden die 26 Buchstaben des deutschen Alphabets mit den Elementen von $\mathbb{Z}_{26}$ identifiziert, z. B. gemäß der folgenden Tabelle:

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

Der Verschlüsselung gemäß Caesar entspricht nach dieser Identifizierung die Addition von drei zum Wert des Klartexts. Obiges Beispiel liest sich damit so:

Klartext	V	E	N	I	V	I	D	I	V	I	C	I
Klartext in $\mathbb{Z}_{26}$	22	5	14	9	22	9	4	9	22	9	3	9
Verschlüsselung +	3	3	3	3	3	3	3	3	3	3	3	3
Geheimtext in $\mathbb{Z}_{26}$	25	8	17	12	25	12	7	12	25	12	6	12
Geheimtext	Y	H	Q	L	Y	L	G	L	Y	L	F	L

Wir addieren modulo 26, d. h. es gelten $23+3=0 \bmod 26$, $24+3=1 \bmod 26$, $25+3=2 \bmod 26$. Somit wird dem Klartext W der Geheimtext Z zugewiesen, X geht über in A, Y wird zu B.

3.2 Monoalphabetische Substitutionen

Monoalphabetische Substitution ist eine Verallgemeinerung der Caesar-Addition. Die Caesar-Addition kann als spezielle Permutation des Alphabets aufgefasst werden, nämlich als zyklische Verschiebung um drei Stellen. Der Schlüsselraum vergrößert sich gigantisch, wenn wir beliebige Permutationen des Alphabets zulassen, nämlich auf $26! \approx 4 \cdot 10^{26}$ mögliche Schlüssel.

Monoalphabetische Substitutionen haben einen entscheidenden Nachteil: Sie übertragen alle statistischen Merkmale des Klartexts auf den Geheimtext, insbesondere die Verteilung der Häufigkeit der einzelnen Buchstaben. Bei der Kryptoanalyse einer monoalphabetischen Chiffre wird man also zunächst eine Häufigkeitstabelle der einzelnen Zeichen des Geheimtexts anfertigen und diesen dann gemäß ihrer Häufigkeit Buchstaben des Geheimtexts zuweisen. Bei kurzen Texten treten natürlich Abweichungen zwischen beobachteten und erwarteten Häufigkeiten auf. Man muss dann experimentieren und nach probeweiser Zuordnung der häufigsten Buchstaben Wörter ergänzen und so weitere Zuordnungen finden. Von einem bestimmten Punkt ab tritt dann der Klartext hervor und der Rest ist sehr einfach.

Wir betrachten als Beispiel den folgenden Geheimtext, der mit einer monoalphabetischen Substitution verschlüsselt wurde:

```
ftyjlmaikktsysltsistmeqmkhzjtstysqtlstystkjqjavaik
ltmuqltrtwquutstsaikvatietmkltkftkjvyhztskypmavamuk
ltmravadykvtihzjtjisntahzjtjtystwvtysstrtvntkqsstiuk
ytwmtykjystystmtsjetmsisrgqsisrteatzmahzjislstisbyr
uyvvvyqstsutyvtstysankqvijisntltijtsltmwvtystmnvairm
itstmpvastjltkktsgquaeetskjaautsltnyqeqmutskqtmkjai
svyhzpmyuyjygykysllakkytlyryjavizmtssqhyuutmeitmt
stisfazmkhztysvyhzjqvvtmeyslirzavjtslytktmpvastjz
ajqltmntkktmrkarjzajjttyspmqnvtulytutykjtstystmnt
fqzstmfamtsseakjyuutmisrvithwvyhzbimvqtkisrlytktkpmq
nvtukfimtsgyvtvgqmkhzvatrrtrtuahzjantmlytlmtzjtskyh
zutykjtiskiulakzsislztmwvtystmntlmihwjtmapytmkhzty
shztsisllakykjtysseahzlmqvvyrfityvtkyurmqqktsislrast
sxasyhzjlytwvtyststntlmihwjtspapytmkhztyshztsfamtsly
tkyhzisrvithwvyhzeitvzjts
```

Die Auszählung der 739 Buchstaben ergibt folgende Häufigkeiten. Die Zahl hinter dem Buchstaben gibt die Häufigkeit in Prozent an, dahinter steht die absolut gezählte Häufigkeit.

T	17.9973	133	J	4.8714	36	E	1.6238	12
S	10.8254	80	L	4.4655	33	P	1.3532	10
Y	8.3897	62	Z	4.3302	32	W	1.3532	10
K	6.7659	50	H	3.3829	25	F	1.0825	8
M	6.4953	48	Q	3.1123	23	G	0.6766	5
A	5.2774	39	U	3.1123	23	B	0.4060	3
V	5.0068	37	R	2.5710	19	D	0.1353	1
I	4.8714	36	N	1.7591	13	X	0.1353	1

Wir ersetzen nun die klein geschriebenen Geheimtextbuchstaben versuchsweise durch groß geschriebene Klartextbuchstaben, wobei wir gemäß obiger Tabelle

folgende Zuordnung für die häufigsten Buchstaben treffen:

t	s	y	k	m	a
E	N	I	S	R	A

Es ergibt sich folgender Text:

```
fEIjLRaiSSENINlENiNEReqRShzjENEINqElENEINESjqjAvAiSl
ERuqlErEwquuENENaiSvAEieERSLESfESjvIhzENSpIRAvARuSLE
RrAvAdISvEihzjEjiNnEAhzjEjEINEwvEINERevNESqNNEiuSIEW
REISjINEINERENjeERNiNrgqNiNrEeAEzRAhzjiNlNEiNbIruIvv
IqNENuEiVENEINAnSqvijiNnElEijENlERwvEINERnvAirRiENER
pvANEjlESSEngquAeeENSjAuuENlEnIqeqRuENSqERSjAiNvIhzp
RIuIjIgSINllASSSIElIrIjAvizRENNqhzIuuEREiEREINEiNfAz
RShzEINvIhzjqvvEEReINliNrAvjENlIESERpvANEjzAjqlERnE
SSERrESArjzAjJEEINpRqnvEuIEuEISjENSEINERnEfqzNERfAR
ENeASjIuuERiNrviEhvwIhzbiRvqESiNrlIESESpRqnvEuSfiRLE
NgIEvEgqRShzvAErErEuAhzjAnERlIElREzjENSihzuEISjENSiu
lASzINiNlzERwvEINERnElRihwjERpApIERSHzEINhzENiNllASI
SjEINeAhzlRqvIrfeIvESiurRqSSENiNlrANbENxANihzjlIEWv
EINENnElRihwjENpApIERSHzEINhzENfARENlIESIhzinrviEhvw
IhzeiEzvjen
```

In der siebten Zeile findet sich die Buchstabenfolge SINllASSIE. Wörter eines deutschen Texts ergibt das wohl nur für die Zuordnung von l zu D. In der ersten Zeile findet sich nun die Folge DRAiSSENINDEN, das die Zuordnung von i zu U nahelegt. SqNNE läßt sich nur durch die Zuordnung von q zu O erklären. Wir ersetzen den häufigsten verbleibenden Buchstaben j durch T. Damit ergibt sich folgender Text:

```
fEITDRAUSSENINDENUNEReORShzTENEINOEDENEINESTOTAvAUSD
ERuODerEwOuuENENAUsvAEUeERSDESfESTvIhzENSpIRAvARuSDE
RrAvAdISvEUhzTETUNnEAhzTETEINEwvEINERevNESONNEUuSIEW
REISTINEINERENTeERNUNrgONUNrEeAEzRAhzTUNDNEUNbIruIvv
IONENuEiVENEINAnSOvUTUNnEDEUTENDERwvEINERnvAUrRUENER
pvANETDESSEngOuAeeENSTAuuENDEnIOeORuENSOERSTAUNvIhzp
RIuITIgSINDDASSSIEDIrITAvUzRENNOhzIuuEREUEREINEUNfAz
RShzEINvIhzTOvvEEReINDUNrzAvTENDIESERpvANETzATODERnE
SSERrESArTzATTEEINpRONvEuDIEuEISTENSEINERnEfOzNERfAR
ENeASTIuuERUNrvUEhvwIhzBURvOESUNrDIESESpRONvEuSfURDE
NgIEvEgORShzvAErErEuAhzTAnERDIEDREzTENSihzuEISTENSUu
DASzINUNDzERwvEINERnEDRUhwTERpApIERSHzEINhzENUNDASI
STEINeAhzDROvvIrfeIvESiurROSSENUNDrANbENxANihzTDIEWv
EINENnEDRUhwTENpApIERSHzEINhzENfARENDIESIhzUNrvUEhvw
IhzeUEzvTEN
```

Der Rest ist nicht mehr schwer. Wir ermitteln folgende Zuordnungstabelle:

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

und lesen den Klartext:

weitdraussenindenunerforschteneinoedeneinestotalausd
 ermodegekommenenauslaeuferdeswestlichenspiralarmsde
 rgalaxisleuchtetunbeachteteinekleinegelbesonneumsiek
 reistineinerentfernungvonungefaehrachtundneunzigmill
 ionenmeileneinabsolutunbedeutenderkleinerblaugruener
 planetdessenvomaffenstammendebioformensoerstaunlichp
 rimitivsinddasssiedigitaluhrennochimmerfuereineunwah
 rscheinlichtollerfindunghaltendieserplanethatoderbe
 ssergesagthatteeinproblemidiemeistenseinerbewohnerwar
 enfastimmerungluecklichzurloesungdiesesproblemswurde
 nvielevorschlaegegemachtaberdierehtensichmeistensum
 dashinundherkleinerbedruckterpapierscheinenunddasi
 steinfachdrolligweilesimgrossenundganzenjanichtdiekl
 einenbedrucktenpapierscheinenwarendiesichunglueckl
 ichfuehlten

Es handelt sich dabei um den Anfang des Buches "Per Anhalter durch die Galaxis", siehe [Ad]. Der Klartext wurde übrigens mit Cryptool verschlüsselt und wieder entschlüsselt, siehe [Cr].

Als Ergänzung der Häufigkeitsanalyse kann man die Methode des wahrscheinlichen Wortes heranziehen. Die Koinzidenzmuster von Wörtern bleiben unter Permutationen des Alphabets erhalten. Natürlich darf man nicht auf ein so einzigartiges Wort wie

Mississippi
 Koinzidenzmuster 12332332442

hoffen, aber auch schon "Division" hat in einem militärischen Kontext ein verräterisches Muster. Gerade in militärischen Nachrichten werden oft Dienstgrade und die Namen von Truppenteilen auftauchen, so dass es reichlich Ansatzpunkte der Kryptoanalyse gibt. Entdeckt man das Koinzidenzmuster eines wahrscheinlichen Wortes in einem Geheimtext, so wird man es probeweise zu einem Ansatz für die Entschlüsselung ausnutzen.

Die Häufigkeitsanalyse der einzelnen Buchstaben wird oft durch die Bigramm-Statistik ergänzt, bei der man die Häufigkeit von Buchstabenpaaren heranzieht.

Insbesondere bei kurzen Geheimtexten kann es nämlich schwierig sein, Buchstaben mit annähernd gleicher Häufigkeit richtig zuzuordnen.

Ein schon recht wirksamer Ansatz, die Häufigkeitsanalyse zu verhindern und Koinzidenzmuster zu brechen, besteht in der Verwendung von Homophonen und Blendern. Man verwendet für jeden Buchstaben des Klartextes so viele Homophone im Geheimentalphabet, wie es seiner Häufigkeit entspricht. Auf diese Weise erreicht man annähernd Gleichverteilung im Geheimtext. Blender sind bedeutungslose Zeichen, die wahllos in den Text eingestreut werden.

Man kann noch weitere Chiffrierregeln angeben, die man befolgen sollte, um dem Kryptoanalytiker die Arbeit so schwer wie möglich zu machen. Auf keinen Fall darf man den Wortzwischenraum anzeigen.

3.3 Das Vigenère-Verfahren

Ein wesentlicher Angriffspunkt gegen einfache Substitution ist, dass jedem Zeichen im Geheimtext immer das gleiche Klartextzeichen entspricht. Dadurch bleiben charakteristische Muster der Wörter erhalten. Eine entscheidende Verbesserung gegenüber monoalphabetischer Substitution ist die polyalphabetische Substitution. Hierbei wird die Substitutionsvorschrift von der Position im Text abhängig gemacht. Die klassische Form der polyalphabetischen Substitution ist das Vigenère-Verfahren. Es verwendet mehrere Caesar-Additionen, die periodisch, durch ein Schlüsselwort gesteuert, abgewechselt werden:

Klartext	c	o	g	i	t	o	e	r	g	o	s	u	m
Schlüsselwort	k	e	y	k	e	y	k	e	y	k	e	y	k
Geheimtext	n	t	f	t	y	n	p	w	f	z	x	t	x

Zum leichteren Rechnen identifiziert man natürlich wieder die Buchstaben des Alphabets mit den Elementen von $\mathbb{Z}_{26}$. Die Vigenère-Chiffre geht auf Vigenère zurück, der sein Verfahren im Jahr 1585 beschrieb. Sie galt einige Jahrhunderte lang als "chiffre indechiffable". Das Vigenère-Verfahren ebnet Häufigkeitsverteilungen ein und bricht Koinzidenzmuster.

Das Vigenère-Verfahren läßt sich sofort mit einem Klartext-Angriff brechen. Um einen erfolgreichen Geheimtext-Angriff zu führen, muss man zunächst die Länge n des Schlüsselworts herausfinden. Ist die Schlüssellänge bekannt, so kann man den Text in n Teile zerlegen und in jedem Teil die Buchstaben sammeln, deren Position im Geheimtext zu einer festen Restklasse nach der Schlüssellänge gehört. In jedem der n Teile führt man dann eine Häufigkeitsanalyse der Buchstaben durch und ordnet dem häufigsten Buchstaben den Klartext "e" zu.

Wie findet man die Schlüssellänge? Der kryptoanalytische Durchbruch gelang dem preußischen Offizier Kasiski 1863. In einem genügend langen Geheimtext

lassen sich Parallelstellen finden, also doppelt auftauchende Buchstabenfolgen. Diese können "unecht" sein, also zufällig durch die Chiffre erzeugt, oder "echt", wenn nämlich der Klartext Parallelstellen enthält, deren Abstand ein Vielfaches der Schlüssellänge ist. Parallelstellen von einzelnen Buchstaben sind natürlich wertlos und solche von Bigrammen meistens auch. Mit wachsender Länge der Buchstabenfolge, die mehrfach auftritt, wird die zufällige Erzeugung jedoch unwahrscheinlicher. Man wird also Parallelstellen im Geheimtext suchen und den größten gemeinsamen Teiler der Abstände der wahrscheinlich echten Parallelstellen als Schlüssellänge annehmen. Entscheidend für das Verfahren ist die Unterscheidung von echten und unechten Parallelstellen, die nicht immer zuverlässig gelingt. Ist der Text lang genug, gibt es jedoch meistens auch sehr lange Parallelstellen. Wir betrachten den folgenden (schon recht langen) Geheimtext.

```
xsquplfmtgmouhiwoivfdztjtqpuqhjaocmeqhjaosauanfdbaie
qlrgesofwiredbmomoxdbscgqlxvfefenqadvmoenjnbzisymiw
suimrnkmscdtnjljbjfmwmlfhmjzypdfwvfsyqtfgwozyzetwml
dynkuwvfuhjjfbgqlsmoudpzosyftiftlfuihcophjmonqhycqd
jcvfzgjamsvfuhfttctvfostfrmvfysvffsmqcswsptbgawmfbs
bffffhlfbemjfwcubrjzjfhinyysvfpqpriwefbapqlxlbivmuwmh
swujfcakjblemmxkjsljscysmipsqhsqgdvqnyywxvszfuhjmokii
dmhzfwvmuwmlpztflkaorcosbfdusveuyxwsdtbzyyzbhweqlgw
tgmssyxshhpbfnjwjbxsavqwnrqfyynkusvtqcswspxabswskis
qhkssthqnyymoutvqwpdjqpaglqgfgcosxnwtsaqdigdfaaxgliw
ojqfxyagsgkixujyfumnwmlbpmspcjvsspuqhxadvufumywogcn
puxzjbcopbjjlmjzywtfrzvoeywsdiqywkdvjmjzwmwoivepuxa
thmjzzfuirzpxfnysqmnmnehfwtaysmorobztjfkovjobyvjjssm
qcswoptmedohcusvqmjnwsgkiqcsuisvxmjljfewmtuwmmoutvqwpd
jqpoggymdusv
```

Die Suche nach Parallelstellen ergibt folgende Parallelstellen mit mindestens vier Buchstaben:

[w o i v] Position 16, Abstand 600,	[p u q h] Position 27, Abstand 528,
[q h j a] Position 29, Abstand 8,	[m w m l] Position 125, Abstand 416,
[h m j z] Position 130, Abstand 496,	[y n k u] Position 158, Abstand 288,
[v f u h] Position 163, Abstand 56,	[h j m o] Position 198, Abstand 160,
[y s v f] Position 238, Abstand 48,	[s m q c] Position 243, Abstand 432,
[v m u w] Position 307, Abstand 64,	[q n y y] Position 347, Abstand 128,
[d u s v] Position 392, Abstand 344,	[q c s w] Position 453, Abstand 224,
[m o u t] Position 480, Abstand 240,	[s g k i] Position 529, Abstand 168,

Der größte gemeinsame Teiler der Abstände ist 8. Wir nehmen daher 8 als Schlüssellänge an. Da jeder Buchstabe des Schlüssels eine Caesar-Addition festlegt genügt es, in jeder Restklasse nach der Schlüssellänge einen Buchstaben richtig zuzuordnen. Wir suchen in jeder Restklasse den häufigsten Buchstaben und vermuten, daß sein Klartext "e" lautet. Die häufigsten Buchstaben in den Restklassen sind:

Klasse	1	2	3	4	5	6	7	8
häufigster	f	s	m	f	q	x	j	w
Buchstabe								

Wir lösen für jede Klasse die Gleichung

$$e + x = \text{"häufigster Buchstabe"}$$

nach x auf und erhalten das Lösungswort "Anhalter". Der Klartext lautet damit:

```
weitdraussenindenunerforschteneinoedeneinestotalausd
ermodegekommenenauslaeuferdeswestlichenspiralarmsde
rgalaxis...
```

Den kennen wir ja schon.

Eine Verallgemeinerung des Verfahrens, die die Kryptoanalyse erschwert, ergibt sich, wenn man nicht nur Caesar-Verschiebungen verwendet, sondern jedem Buchstaben des Schlüsselworts eine beliebige monoalphabetische Substitution zuordnet. Diese Idee hatte auch schon Vigenère.

4 DES

Wir wollen uns nun modernen kryptographischen Verfahren zuwenden. Ein wichtiges standardisiertes und immer noch sehr gutes Kryptosystem ist der **Data Encryption Standard** (DES). DES wurde von IBM im Auftrag des amerikanischen NBS (National Bureau of Standards, heute NIST) entwickelt und 1977 veröffentlicht. Bis heute ist die Beteiligung der NSA (National Security Agency) an der Entwicklung nicht völlig geklärt. Da zwar der Algorithmus, nicht aber (zunächst) die Kriterien des Entwurfs veröffentlicht wurden, konnten sich Gerüchte halten, die NSA habe in DES geheime Hintertüren eingebaut, die die unbefugte Dechiffrierung "von Amts wegen" ermöglichen könnte. Andererseits wurde das Verfahren von vielen Kryptoanalytikern intensiv geprüft, ohne dass Schwachstellen gefunden wurden. Die einzige wesentliche Schwachstelle ist die rasante Entwicklung der Computer-Technik, mit deren Hilfe man heute einen Brute-Force-Angriff angesichts der relativ kleinen Schlüssellänge von 56 Bits in weniger als einem Tag durchführen kann.

4.1 Beschreibung des Algorithmus

4.1.1 Klartext und Schlüsselraum

Die folgende Darstellung des DES-Algorithmus ist [Bu] entnommen. Klartext- und Schlüsselraum des DES ist $\mathcal{C} = \mathcal{P} = \{0, 1\}^{64}$. Die DES-Schlüssel sind Bitstrings der Länge 64, die folgende Eigenschaft haben: Teilt man einen String der Länge 64 in acht Bytes auf, so ist jeweils das letzte Bit eines jeden Bytes so gesetzt, dass die Quersumme aller Bits im betreffenden Byte ungerade ist. Die ersten sieben Bits eines Bytes in einem DES-Schlüssel legen das achte Bit fest. Dies ermöglicht Korrektur von Speicher- und Übertragungsfehlern. In einem DES-Schlüssel sind also nur 56 Bits frei wählbar. Insgesamt gibt es 2^{56} DES-Schlüssel. Der DES-Schlüssel für Ver- und Entschlüsselung ist derselbe.

Beispiel 4.1 Ein gültiger DES-Schlüssel ist hexadezimal geschrieben

497389DC64A7B958

Seine Binärentwicklung lautet

0	1	0	0	1	0	0	1
0	1	1	1	0	0	1	1
1	0	0	0	1	0	0	1
1	1	0	1	1	1	0	0
0	1	1	0	0	1	0	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	1
0	1	0	1	1	0	0	0

4.1.2 Die initiale Permutation

Wir werden jetzt den DES-Algorithmus im Einzelnen beschreiben. Auf den zu verschlüsselnden Klartext p wird zunächst eine initiale Permutation IP angewandt. Dies ist eine für das Verfahren fest gewählte, vom Schlüssel unabhängige Bitpermutation auf Bitvektoren der Länge 64. Die Permutation IP lautet

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

Dies ist folgendermaßen zu verstehen: Ist $p \in \{0,1\}^{64}$, $p = p_1p_2 \dots p_{64}$, dann ist $IP(p) = p_{58}p_{50} \dots p_7$. Auf das Ergebnis dieser Permutation wird der eigentliche Algorithmus angewandt. Zuletzt wird die Ausgabe erzeugt, indem auf das Ergebnis des Algorithmus die zu IP inverse Permutation IP^{-1} angewandt wird.

4.1.3 Die interne Blockchiffre

Sei $p \in \{0,1\}^{64}$ der zu verschlüsselnde Klartext. p wird aufgeteilt in zwei Bitstrings L_0 und R_0 der Länge 32, wobei L_0 die linke Hälfte von p bezeichnet, R_0 die rechte. In Abhängigkeit von k wird ein erster Rundenschlüssel K_1 berechnet. Die Beschreibung der Rundenschlüssel folgt später. Auf R_0 wird die Expansionsfunktion $E : \{0,1\}^{32} \rightarrow \{0,1\}^{48}$ angewandt:

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

Anschließend wird der String $E(R_0) \oplus K_1$ gebildet und in acht Blöcke B_i , $1 \leq i \leq 8$ aufgeteilt:

$$E(R_0) \oplus K_1 = B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8$$

mit $B_i \in \{0, 1\}^6$, $1 \leq i \leq 8$. Im nächsten Schritt werden die Funktionen

$$S_i : \{0, 1\}^6 \rightarrow \{0, 1\}^4, \quad 1 \leq i \leq 8$$

verwendet, die sogenannten S-Boxen. Die S-Boxen sind von zentraler Bedeutung für die Sicherheit des DES. Sie werden später genauer beschrieben. Mit diesen Funktionen wird der String

$$C = C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8$$

berechnet, wobei $C_i = S_i(B_i)$ ist für $1 \leq i \leq 8$. Dieser Bitstring wird gemäß der Permutation P permutiert:

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

Das Ergebnis ist $f_{K_1}(R_0)$. Die erste Runde der internen Blockchiffre ist beendet.

4.1.4 Die S-Boxen

Wir beschreiben nun die Funktionen S_i , $1 \leq i \leq 8$. Diese Funktionen heißen S-Boxen. Sie werden in der folgenden Tabelle dargestellt.

Zeile	Spalte															
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
S_1																
[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
[1]	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
[2]	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
[3]	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2																
[0]	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
[1]	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
[2]	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
[3]	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3																
[0]	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
[1]	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
[2]	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
[3]	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4																
[0]	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
[1]	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
[2]	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
[3]	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5																
[0]	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
[1]	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
[2]	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
[3]	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6																
[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
S_7																
[0]	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
[1]	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
[2]	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
[3]	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8																
[0]	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
[1]	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
[2]	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
[3]	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Jede S-Box wird durch eine Tabelle mit vier Zeilen und 16 Spalten beschrieben.

Für einen String $B = b_1b_2b_3b_4b_5b_6$ wird der Funktionswert $S_i(B)$ folgendermaßen berechnet: Man interpretiert die natürliche Zahl mit Binärentwicklung b_1b_6 als Zeilenindex und die natürliche Zahl mit Binärentwicklung $b_2b_3b_4b_5$ als Spaltenindex. Den Eintrag in dieser Zeile und Spalte der S-Box stellt man binär dar und füllt diese Binärentwicklung vorne so auf, daß ihre Länge 4 wird. Das Ergebnis ist $S_i(B)$.

Bekannt sind unter Anderem folgende Design-Prinzipien der S-Boxen:

1. Keine S-Box ist eine lineare oder affine Abbildung.
2. Wird ein Bit des Inputs geändert, so ändern sich mindestens zwei Bits des Outputs.

4.1.5 Weitere Runden der internen Blockchiffre

In Abschnitt 4.1.3 wurde die erste Runde der internen Blockchiffre beschrieben. Im DES werden 16 ausgeführt, aber es ist auch denkbar, eine beliebige Rundenzahl $r \geq 1$ festzulegen. Hierzu muss aus dem Schlüssel $k \in \mathcal{K}$ eine Folge $K_1, \dots, K_r$ von Rundenschlüsseln festgelegt werden. Wie dies im DES ausgeführt wird, folgt im nächsten Abschnitt. Wir erinnern uns, dass der Klartext p in zwei Hälften L_0 und R_0 zerlegt wurde. Außerdem wurde $f_{K_1}(R_0)$ berechnet. Nun konstruieren wir eine Folge von Bitstrings $((L_i, R_i))_{1 \leq i \leq r}$ gemäß

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

Die Ergebnisse $f_{K_i}(R_{i-1})$ werden für $1 \leq i \leq r$ analog zum Vorgehen in Abschnitt 4.1.3 berechnet. Als Ergebnis der internen Blockchiffre wird der Bitstring (R_r, L_r) ausgegeben. Im DES wird auf dieses Ergebnis noch die Inverse der initialen Permutation angewandt.

4.1.6 Die Rundenschlüssel

Es bleibt noch zu erklären, wie aus dem Schlüssel $k \in \mathcal{K}$ die Rundenschlüssel K_i , $1 \leq i \leq 16$ für 16 Runden berechnet werden. Sei ein DES-Schlüssel $k \in \{0, 1\}^{64}$ gegeben. Die Rundenschlüssel haben die Länge 48. Wir definieren für $1 \leq i \leq 16$ v_i folgendermaßen:

$$v_i = \begin{cases} 1 & \text{für } i \in \{1, 2, 9, 16\}, \\ 2 & \text{sonst.} \end{cases}$$

Nun werden zwei Funktionen benutzt: Es ist $PC1 : \{0, 1\}^{64} \rightarrow \{0, 1\}^{28} \times \{0, 1\}^{28}$,

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

und $PC2 : \{0, 1\}^{28} \times \{0, 1\}^{28} \rightarrow \{0, 1\}^{48}$,

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

$PC1$ bildet einen Bitstring $k = k_1 \dots k_{64}$ der Länge 64 auf zwei Bitstrings C und D der Länge 28 ab. Dabei ist $C = k_{57}k_{49} \dots k_{36}$ und $D = k_{63}k_{55} \dots k_4$. Die Funktion $PC2$ bildet ein Paar (C, D) von Bitstrings der Länge 28 auf einen Bitstring der Länge 48 ab. Der Wert $PC2(b_1 \dots b_{56})$ ist $b_{14}b_{17} \dots b_{32}$. Mit diesen Bausteinen erhält man die Schlüssel so:

1. Setze $(C_0, D_0) = PC1(k)$.
2. Für $1 \leq i \leq 16$ berechne K_i folgendermaßen: C_i ist der String, den man durch eine zirkuläre Linksshift um v_i Stellen aus C_{i-1} erhält, D_i ist der String, den man durch eine zirkuläre Linksshift um v_i Stellen aus D_{i-1} gewinnt. Berechne dann $K_i = PC2(C_i, D_i)$.

Dies beendet die Beschreibung des DES-Algorithmus. Sie ist so kompliziert, dass man sie kaum behalten kann. Man sollte aber beachten, dass sie nur sehr einfache Verknüpfungen erfordert, die sich leicht in Hardware realisieren lassen.

4.2 Entschlüsselung

Um einen Geheimtext zu entschlüsseln, wendet man DES mit der umgekehrten Schlüsselfolge auf ihn an. Die Entschlüsselung der internen Blockchiffre beruht dabei auf folgender Gleichung:

$$(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus f_{K_i}(L_i)), \quad 1 \leq i \leq 16.$$

Daher kann man unter Verwendung der Schlüsselfolge $(K_{16}, K_{15}, \dots, K_1)$ in 16 Runden das Paar (L_0, R_0) aus dem Geheimtext (R_r, L_r) zurückgewinnen.

4.3 Differentielle Kryptoanalyse des DES

Ein Brute-Force Angriff bedeutet im Fall von DES das Durchprobieren von 2^{56} Schlüsseln. Für etwa 72.000.000.000.000.000 Schlüssel (72 Billionen) muss der Geheimtext dechiffriert und auf sinnvollen Klartext getestet werden. Bis heute ist Brute-Force der beste bekannte Angriff gegen DES.

1990 wandten die israelischen Mathematiker Biham und Shamir die von ihnen entwickelte differentielle Kryptoanalyse unter anderem auf den DES an (siehe [Bi]). Nach der Entdeckung der differentiellen Kryptoanalyse wurden die Entwurfskriterien der S-Boxen veröffentlicht. Die S-Boxen garantieren offenbar eine maximale Immunität gegenüber differentieller Kryptoanalyse. Das ist wohl kein Zufall, d. h. IBM und NSA kannten diesen Angriff schon während der Entwurfsphase des DES. Der Wissensvorsprung der NSA gegenüber der öffentlichen kryptologischen Forschung auf diesem Gebiet betrug demnach wenigstens 20 Jahre.

Differentielle Kryptoanalyse ist trotzdem eine raffinierte Angriffsart von allgemeinem Interesse. Wir stellen sie hier am Beispiel des DES kurz vor.

Die differentielle Kryptoanalyse ist ein Klartextangriff. Ausgangspunkt der differentiellen Kryptoanalyse ist die Differenz von Bitstrings. Dabei bezeichnen wir eine Folge von n Bits (für ein $n \in \mathbb{N}$) als Bitstring der Länge n . Die Differenz zweier Bitstrings einer festen Länge n erhält man, wenn man die Strings bitweise modulo 2 addiert. In der Differenz zweier Bitstrings sind also die Bits an den Stellen gesetzt, an denen die Bitstrings sich unterscheiden. Die Differenz zweier Bitstrings wird auch mit *XOR* bezeichnet (exklusives Oder):

$$XOR : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n, \quad (X, X^*) \mapsto X \oplus X^*.$$

Nun werden Paare von Bitstrings der Länge 64 mit einer fest gewählten Differenz jeweils mit dem DES-Algorithmus mit dem unbekannten Schlüssel verschlüsselt. Anschließend wird die Differenz des Outputpaares des Algorithmus betrachtet. Diese Differenz enthält Informationen über den verwendeten Schlüssel.

Wir verwenden im folgenden die Bezeichnungen aus Abschnitt 4.1.3. Gegeben seien zwei Bitstrings X und X^* der Länge 32. Es gilt für die Expansion E

$$E(X) \oplus E(X^*) = E(X \oplus X^*)$$

Für jeden Rundenschlüssel K_i , $1 \leq i \leq 16$, gilt

$$(E(X) \oplus K_i) \oplus (E(X^*) \oplus K_i) = E(X \oplus X^*).$$

Schließlich gilt für die Outputs der S-Boxen C und C^* und die Permutation P

$$P(C) \oplus P(C^*) = P(C \oplus C^*).$$

Die S-Boxen sind bekanntlich nicht linear. Die Kenntnis der Differenz eines Paares von Inputs der S-Boxen garantiert nicht die Kenntnis der Differenz des Output-Paares. Normalerweise sind mehrere Output-Differenzen möglich. Ein Spezialfall ergibt sich, wenn beide Inputs gleich sind. Dann müssen auch beide Outputs gleich sein. Eine wichtige Beobachtung ist die folgende: Für jede spezielle Input-Differenz sind nicht alle Output-Differenzen möglich. Außerdem kommen die möglichen Output-Differenzen nicht gleich häufig vor, manche treten sehr viel öfter auf als andere.

Im DES hat jede S-Box $2^6 = 64$ mögliche Inputs, folglich gibt es 64^2 mögliche Input-Paare. Es gibt 64 mögliche Differenzen der Input-Paare und $2^4 = 16$ mögliche Differenzen der Output-Paare. Wir fassen die Verteilung der Input- und Output Differenzen einer S-Box in einer Tabelle zusammen. Die folgende Tabelle enthält einen Ausschnitt der Differenzverteilung der S-Box S_1 . Dabei wurde den binären Differenzen ihre Darstellung als natürliche Zahl zur Basis 10 zugeordnet. Die vollständigen Tabellen aller S-Boxen findet man in [Bi].

Input Differenz	Output Differenz															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
48	0	4	6	0	12	6	2	2	8	2	4	4	6	2	2	4
49	4	8	2	10	2	2	2	2	6	0	0	2	2	4	10	8
50	4	2	6	4	4	2	2	4	6	6	4	8	2	2	8	0
51	4	4	6	2	10	8	4	2	4	0	2	2	4	6	2	4
52	0	8	16	6	2	0	0	12	6	0	0	0	0	8	0	6

Wir zeigen nun an einem Beispiel, wie man mit Hilfe bekannter Input-Paare und Output-Differenzen einer S-Box die Schlüsselbits finden kann.

Beispiel 4.2 Wir betrachten die S-Box S_1 . Gegeben sei ein Paar von Bitstrings X und X^* der Länge 32, ein Input-Paar der internen Blockchiffre. Da X und X^* als bekannt vorausgesetzt werden, können wir $E(X)$ und $E(X^*)$ berechnen. Wir geben hier direkt $E(X)$ und $E(X^*)$ an: Es gelte $E(X) = 000001$, $E(X^*) = 110101$. Die Differenz $E(X) \oplus E(X^*)$ lautet 110100. Wegen $(E(X) \oplus K_i) \oplus (E(X^*) \oplus K_i) = E(X \oplus X^*)$ ist damit auch die Differenz der Inputs von S_1 110100. Weiter sei bekannt, dass die 4 Bits der Output-Differenz von S_1 die folgenden sind: 1101. Wir wollen die ersten 6 Bits des Rundenschlüssels finden. In der Tabelle der Differenzenverteilung von S_1 sehen wir, dass es 4 mögliche Input-Paare mit den angegebenen Input- und Output-Differenzen gibt. Diese 4 möglichen Paare führen zu 8 möglichen Schlüsseln, denn die möglichen Schlüsselbits können wegen $E(X) \oplus K = B_1B_2B_3B_4B_5B_6B_7B_8$ als Differenz der ersten 6 Bits von $E(X)$ und B_1 bzw. als Differenz der ersten 6 Bits von $E(X^*)$

und B_1 berechnet werden. Jedes Input-Paar führt zu zwei Schlüsseln. Folgende möglichen Input-Paare führen zu folgenden möglichen Schlüsseln:

S-Box Input		Mögliche Schlüssel	
000110	110010	000111	110011
010000	100100	010001	100101
010110	100010	010111	100011
011100	101000	011101	101001

Die richtigen Schlüsselbits müssen in dieser Tabelle vorkommen.

Die Verwendung weiterer Paare von Input- und Output-Differenzen, die durch den selben Schlüssel verschlüsselt werden, liefert weitere Kandidaten für die ersten Schlüsselbits. Seien nun X und X^* zwei neue Bitstrings der Länge 32. Es gelte $E(X) = 100001$, $E(X^*) = 010101$. Die Differenz der ersten 6 Bits dieser beiden Bitstrings ist damit 110100. Die Outputdifferenz der S-Box S_1 sei 0011. Es gibt 3 mögliche Input-Paare, die zu 6 möglichen Schlüsseln führen:

S-Box Input		Mögliche Schlüssel	
000001	110101	100000	010100
000010	110110	010111	100011
010101	100001	000000	110100

Zwei Schlüssel tauchen in beiden beiden Tabellen auf: 010111 und 100011. Einer von beiden liefert die ersten 6 Bits des gesuchten Rundenschlüssels.

4.4 Betriebsmodi für Blockchiffren

Wichtige Verschlüsselungsverfahren sind Blockchiffren. Sie bilden Blöcke fester Länge auf Blöcke derselben Länge ab. Beispielsweise werden im DES Bitstrings der Länge 64 auf Bitstrings der Länge 64 abgebildet. Man kann Blockchiffren in unterschiedlicher Weise benutzen, um ganze Texte zu verschlüsseln. Wir stellen im folgenden zwei Möglichkeiten vor.

4.4.1 ECB-Mode

Gegeben sei eine Blockchiffre mit Alphabet $\{0,1\}$ und Blocklänge n für eine natürliche Zahl n . Eine naheliegende Art aus dieser Blockchiffre ein Verschlüsselungsverfahren für beliebig lange Texte zu machen, ist ihre Verwendung im Electronic Codebook Mode (ECB-Mode). Ein beliebig langer Klartext wird in Blöcke der Länge n aufgeteilt. Gegebenenfalls muss man den Klartext so ergänzen, dass seine Länge durch n teilbar ist. Dann wird jeder Block mit einem zulässigen Schlüssel der Blockchiffre verschlüsselt.

Bei der Verwendung des ECB-Modus werden gleiche Klartextblöcke in gleiche Chiffretextblöcke verschlüsselt. Regelmäßigkeiten des Klartexts führen zu Regelmäßigkeiten des Geheimtexts. Ein anderer Nachteil des ECB-Modus besteht darin, dass ein Angreifer die Nachricht ändern kann, indem er Geheimtextblöcke einfügt, die mit dem gleichen Schlüssel verschlüsselt worden sind. Genauso kann ein Angreifer unbemerkt die Reihenfolge der Geheimtextblöcke ändern.

4.4.2 CBC-Mode

Der Cipherblock Chaining Mode (CBC-Mode) beseitigt einige Nachteile des ECB-Modus. Die Verschlüsselung eines Klartextblocks hängt dabei nicht nur von dem Block selbst und dem Schlüssel ab, sondern auch von den vorhergegangenen Blöcken. Die Verschlüsselung ist also kontextabhängig. Nachträgliche Veränderung des Geheimtexts kann man daran erkennen, dass die Entschlüsselung nicht mehr funktioniert.

Gegeben sei eine Blockchiffre mit Alphabet $\{0,1\}$, Blocklänge n und Schlüsselraum $\mathcal{K}$. Zu $k \in \mathcal{K}$ sei E_k die Verschlüsselungsfunktion, D_k die zugehörige Entschlüsselungsfunktion. Wir benötigen einen festen Initialisierungsvektor $IV \in \{0,1\}^n$. Der Klartext wird in Blöcke der Länge n aufgeteilt. Will man eine Folge $m_1, \dots, m_t$ von Klartextblöcken der Länge n mit dem Schlüssel k verschlüsseln, so setzt man

$$c_0 = IV, \quad c_j = E_k(c_{j-1} \oplus m_j), \quad 1 \leq j \leq t.$$

Man erhält die Geheimtextblöcke $c_1, \dots, c_t$. Um sie zu entschlüsseln setzt man

$$c_0 = IV, \quad m_j = c_{j-1} \oplus D_k(c_j), \quad 1 \leq j \leq t.$$

Tatsächlich ist $c_0 \oplus D_k(c_1) = c_0 \oplus c_0 \oplus m_1 = m_1$. Entsprechend verifiziert man das ganze Verfahren.

Beispiel 4.3 Gegeben sei eine Blockchiffre mit Blöcken der Länge 4. Es sei $\mathcal{K} = S_4$, d. h. der Schlüsselraum sei die Menge der Permutationen der Menge $\{1, 2, 3, 4\}$. Für $\pi \in S_4$ sei

$$E_\pi : \{0,1\}^4 \rightarrow \{0,1\}^4, \quad b_1 b_2 b_3 b_4 \mapsto b_{\pi(1)} b_{\pi(2)} b_{\pi(3)} b_{\pi(4)}.$$

Der Klartext m sei

$$m = 101100010100101.$$

Er wird in Blöcke der Länge 4 aufgeteilt. Der letzte Block hat dann nur die Länge 3. Er wird auf die Länge 4 ergänzt, indem eine Null angehängt wird. Wir erhalten die Blöcke

$$m_1 = 1011, \quad m_2 = 0001, \quad m_3 = 0100, \quad m_4 = 1010.$$

Wir verwenden den Schlüssel

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}.$$

Als Initialisierungsvektor verwenden wir

$$IV = 1010.$$

Damit ergibt sich $c_0 = 1010$,

$$c_1 = E_\pi(c_0 \oplus m_1) = E_\pi(0001) = 0010,$$

$c_2 = 0110$, $c_3 = 0100$, $c_4 = 1101$. Also ist der Geheimtext

$$c = 0010011001001101.$$

Wir entschlüsseln ihn wieder und erhalten

$$m_1 = c_0 \oplus E_{\pi^{-1}}(c_1) = 1010 \oplus 0001 = 1011$$

sowie $m_2 = 0001$, $m_3 = 0100$, $m_4 = 1010$. π^{-1} bezeichnet die zu π inverse Permutation. Sie ist gegeben durch

$$\pi^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}.$$

Es gilt $D_\pi = E_{\pi^{-1}}$.

Gleiche Texte werden im CBC-Mode verschieden verschlüsselt, wenn man den Initialisierungsvektor ändert. Außerdem hängt die Verschlüsselung eines Blocks von den vorhergegangenen Blöcken ab. Gleiche Klartextblöcke in unterschiedlichem Kontext werden im allgemeinen verschieden verschlüsselt.

5 Public Key Kryptographie

Um eine verschlüsselte Nachricht lesen zu können, braucht der Empfänger einen Schlüssel zum Entschlüsseln. Damit sichergestellt ist, dass nur der Empfänger die Nachricht lesen kann, muss der Schlüssel geheim gehalten werden. Jahrhunderte lang, seit den Anfängen der Kryptographie im alten Griechenland bis in die zweite Hälfte des 20. Jahrhunderts, gingen Kryptologen davon aus, dass der geheime Schlüssel ein gemeinsames Geheimnis von Sender und Empfänger sein muss. Er wurde über einen sicheren Kanal ausgetauscht, etwa durch einen vertrauenswürdigen Kurier, der ihn im Schuhabsatz oder im Griff des Spazierstocks aufbewahrte.

Gibt es nur wenige Kommunikationspartner, die untereinander geheime Nachrichten austauschen wollen, so kann man einigen Aufwand beim Austausch des Schlüssels betreiben. Heute aber sind wir alle Nutzer kryptographischer Verfahren und wollen etwa über das Internet möglichst vertraulich miteinander kommunizieren. Wollten wir mit jedem potentiellen Kommunikationspartner einen geheimen Schlüssel austauschen, so wären wir organisatorisch völlig überfordert.

Die Frage, ob der Schlüssel eines Kryptosystems ein gemeinsames Geheimnis von Sender und Empfänger sein muss, stellten sich auch Diffie und Hellman. Ihre diesbezüglichen Überlegungen, die sie 1976 veröffentlichten, gelten als Grundlagen der Public Key Kryptographie.

Das Prinzip der Public Key Kryptographie kann man sich wie das Prinzip eines Briefkastens vorstellen. Jeder kann eine Nachricht dort einwerfen, aber nur der Besitzer eines Schlüssels kann den Briefkasten öffnen und die eingeworfenen Nachrichten lesen. Es muss sichergestellt werden, dass nur der Adressat einen Briefkastenschlüssel besitzt.

In der Kryptographie betrachtet man keine Briefkästen, sondern mathematische Verfahren. Wichtig sind dabei die Begriffe **öffentlicher Schlüssel** und **privater Schlüssel**. Dem Einwerfen in den Briefkasten entspricht dann das Verschlüsseln einer Nachricht mit einem öffentlichen Schlüssel e , d. h. ein Klartext m wird durch das Verfahren f mit dem Schlüssel e zum Geheimtext c :

$$c = f_e(m).$$

Der Briefkastenschlüssel entspricht einem geheimen privaten Schlüssel d , mit dessen Hilfe der Geheimtext entschlüsselt wird:

$$m' = f_d(c).$$

Hierbei ist darauf zu achten, dass korrekt entschlüsselt wird, d. h. dass

$$m' = f_d(c) = f_d(f_e(m)) = m$$

gilt für alle Klartexte m .

Bis heute gibt es keine beweisbar sicheren Public Key Kryptographieverfahren. Die Sicherheit der bekannten Verfahren hängt von der Schwierigkeit ab, bestimmte Berechnungsprobleme aus der Zahlentheorie zu lösen, z. B. das Faktorisierungsproblem für natürliche Zahlen. Wir rufen den Hauptsatz der elementaren Zahlentheorie in Erinnerung:

Eine natürliche Zahl $p > 1$ heißt Primzahl, wenn sie genau zwei positive Teiler hat, nämlich 1 und p .

Satz 5.1 *Jede natürliche Zahl kann als Produkt von Primzahlen geschrieben werden. Bis auf die Reihenfolge sind die Faktoren in diesem Produkt eindeutig bestimmt.*

Man geht davon aus, dass die in der Kryptographie verwendeten Berechnungsprobleme Einwegfunktionen sind. Eine **Einwegfunktion** ist eine Funktion, die einfach auszuführen, aber schwer - praktisch unmöglich - zu invertieren ist. Ein alltägliches Beispiel einer Einwegfunktion ist ein Telefonbuch. Es ist leicht, zu einem Namen die zugehörige Nummer zu finden, aber schwer, ohne weitere Hilfe, etwa eines Computers, zu einer gegebenen Nummer den Namen zu finden. Bisher hat sich auch das Faktorisierungsproblem für natürliche Zahlen als Einwegfunktion erwiesen. Es ist leicht, zu zwei ganzen Zahlen p und q ihr Produkt $n = pq$ zu berechnen, aber es ist kein Algorithmus bekannt, der p und q effizient berechnet, falls nur n gegeben ist.

In der Public Key Kryptographie werden Einwegfunktionen verwendet, um Klartexte zu verschlüsseln. Verwendet man hierzu eine echte Einwegfunktion, so ist der Klartext vor jeglichem Zugriff sicher, insbesondere erhält der Empfänger nur unlesbaren Buchstaben- oder Zahlensalat. Der Empfänger sollte aber in der Lage sein, an ihn gerichtete Nachrichten zu lesen. Darum benutzt man einen Trick: Der Empfänger besitzt in der Public Key Kryptographie eine geheime Zusatzinformation, eine Trapdoor-Information, mit deren Hilfe er entschlüsseln kann, nämlich seinen privaten Schlüssel. Einwegfunktionen, die mit Hilfe einer Zusatzinformation leicht invertierbar sind, heißen **Trapdoor-Einwegfunktionen**.

Ob es überhaupt Einwegfunktionen gibt, ist unbekannt. Man kann sich als Kryptograph nur damit trösten, dass die in Public Key Kryptographie-Verfahren verwendeten zahlentheoretischen Probleme schon sehr lange als Einwegfunktionen gelten.

Was müsste man tun, um die Sicherheit eines Public Key Verfahrens streng mathematisch zu beweisen? Sicher müsste man zeigen, dass es nicht möglich ist, einen geheimen Schlüssel aus einem öffentlichen Schlüssel zu bestimmen. Aber das genügt nicht. Man müsste auch zeigen, dass ein Angreifer aus beobachteten Geheimtexten keine relevanten Informationen über die verschlüsselten Klartexte

erhalten kann; jedenfalls nicht in annehmbarer Zeit. Wir beschränken uns darauf zu zeigen, wie schwierig es ist, geheime Schlüssel aus öffentlichen Schlüsseln zu ermitteln und bekannte Angriffe zu erläutern.

5.1 Der man-in-the-middle-Angriff

In der kryptologischen Literatur heißen die "guten" Kommunikationspartner Alice und Bob. Alice sendet verschlüsselte Nachrichten an Bob. Der "böse Angreifer" heißt Mallory.

Entscheidend für die Verwendung von Public-Key-Verfahren ist es, dass niemand einen öffentlichen Schlüssel unter dem Namen eines anderen Teilnehmers veröffentlichen kann. Denkbar ist der man-in-the-middle-Angriff. Alice sendet Bob eine E-Mail: "Bitte maile mir Deinen öffentlichen Schlüssel". Bob erhält die Mail und sendet Alice den Schlüssel. Alice erhält ihn, sendet die chiffrierte Nachricht, und Bob kann sie auch lesen. In Wirklichkeit hat aber Mallory, der Angreifer, alles mitgelesen. Wie hat er das gemacht?

Zunächst las Mallory die Aufforderung an Bob mit, den öffentlichen Schlüssel zu schicken. Dann fing Mallory Bobs Antwort ab, also Bobs öffentlichen Schlüssel. Er schickte aber an dessen Stelle einen eigenen öffentlichen Schlüssel an Alice. Nun konnte er jede Mail von Alice an Bob dechiffrieren, lesen, dann die Nachricht mit Bobs öffentlichem Schlüssel neu chiffrieren und an Bob schicken.

Beim man-in-the-middle-Angriff sitzt Mallory in der Mitte der Leitung und gibt jeder Seite vor, der jeweils andere zu sein.

Es gibt mehrere Möglichkeiten, diesen Angriff zu verhindern. 1984 wurde von Rivest und Shamir das **Interlock-Protokoll** vorgeschlagen. Dabei müssen sich Alice und Bob gegenseitig Nachrichten senden, anhand derer jeder von ihnen erkennen kann, dass eine Nachricht eindeutig vom anderen stammt. Der Ablauf des Protokolls ist so:

1. Alice und Bob senden sich ihre öffentlichen Schlüssel zu. Mallory könnte wie bisher eigene Schlüssel dafür einsetzen.
2. Alice chiffriert eine individuelle, aber nicht zu vertrauliche Nachricht mit dem erhaltenen öffentlichen Schlüssel (der von Bob oder Mallory stammt, je nachdem). Von dieser Nachricht schickt sie aber nur einen nicht dechiffrierbaren Teil an Bob, z. B. nur die Bits an ungeradzahlig Positionen.
3. Bob verfährt nach Erhalt der ersten Rate ebenso: Er sendet einen nicht dechiffrierbaren Teil einer chiffrierten Nachricht an Alice.
4. Nun sendet Alice den zweiten Teil ihrer Nachricht.

5. Bob setzt beide Teile von Alice Nachricht zusammen und dechiffriert sie mit seinem privaten Schlüssel.
6. Ist alles korrekt, sendet auch Bob den zweiten Teil an Alice. Diese setzt ebenfalls beide Nachrichten zusammen und dechiffriert sie.

Der Trick, dass sich beide Seiten abwechselnd nicht dechiffrierbare Nachrichten senden und dann abwechselnd diese Teile ergänzen, macht das Protokoll sicher.

5.2 Erzeugung großer Primzahlen

Für Public-Key-Kryptosysteme braucht man häufig zufällige große Primzahlen. Dazu erzeugt man natürliche Zahlen der richtigen Größe und prüft, ob sie Primzahlen sind. Eine einfache Methode, Probedivision, beruht auf folgendem Satz:

Satz 5.2 *Wenn n eine zusammengesetzte natürliche Zahl ist, dann hat n einen Primteiler p , der nicht größer ist als $\sqrt{n}$.*

Um festzustellen, ob n eine ungerade Primzahl ist, braucht man also nur für alle Primzahlen p , die nicht größer als $\sqrt{n}$ sind, testen, ob sie n teilen.

Den einfachsten Primzahl-Test, der nicht auf Probedivision beruht, gewinnt man aus dem folgenden Satz, der eine unmittelbare Konsequenz des kleinen Satzes von Fermat (Satz 5.5) ist:

Satz 5.3 *Ist $a^{n-1} \not\equiv 1 \pmod{n}$ für eine Zahl $a \in \mathbb{Z}$ mit $\text{ggT}(a, n) = 1$, so ist n nicht prim.*

Der Fermat-Test ist sehr schnell, aber nicht sehr zuverlässig, weil ihm zu viele zusammengesetzte Zahlen durch die Maschen schlüpfen, und zwar auch dann noch, wenn man ihn zu verschiedenen Basen a wiederholt. Ein wesentlich besserer Test, der kaum mehr Aufwand erfordert, ist der **Rabin-Miller-Test**. Man führt ihn für die Zahl n folgendermaßen durch:

1. Man zerlegt $n - 1 = q \cdot 2^u$, wobei q ungerade ist.
2. Man bestimmt für ein $a \in \mathbb{Z}$ mit $\text{ggT}(a, n) = 1$, die Potenzen $a^q, (a^q)^2, \dots, (a^q)^{2^u}$, wobei man aber stoppt, wenn $(a^q)^{2^\nu} \equiv 1 \pmod{n}$ ist.
3. n ist keine Primzahl, wenn einer der folgenden Fälle eintritt:

$$(1) \quad a^{n-1} \not\equiv 1 \pmod{n}, \quad (2) \quad \nu \geq 1, (a^q)^{2^{\nu-1}} \not\equiv -1 \pmod{n}.$$

Im Fall (1) besteht n ja nicht einmal den Fermat-Test zur Basis a . Im Fall (2) gilt für $x = (a^q)^{2^{\nu-1}}$, dass $x^2 \equiv 1 \pmod{n}$, aber $x \not\equiv \pm 1 \pmod{n}$ ist. Dies ist aber

unmöglich, wenn n eine Primzahl ist:

$$x^2 - 1 = (x + 1)(x - 1) = 0 \pmod n$$

impliziert, dass $x + 1$ oder $x - 1$ von n geteilt wird. Es gilt der folgende Satz:

Satz 5.4 *Ist n keine Primzahl, so beträgt die Anzahl der Restklassen a , zu denen n den Rabin-Miller-Test übersteht, höchstens $\varphi(n)/4$.*

Ein Beweis dieses Satzes findet sich in [Fo]. Nach Satz 5.4 ist die Wahrscheinlichkeit dafür, dass n den Test besteht und zusammengesetzt ist, kleiner als $1/4$. Wiederholt man den Miller-Rabin-Test t -mal und ist n zusammengesetzt, so ist die Wahrscheinlichkeit, dass n den Test besteht, kleiner als $(1/4)^t$. Für $t = 10$ ist dies $1/2^{20} \approx 1/10^6$. Das ist sehr unwahrscheinlich.

5.3 Das RSA-Verfahren

5.3.1 Schlüsselerzeugung, Verschlüsselung und Entschlüsselung

Der RSA-Algorithmus wurde 1978 von Rivest, Shamir und Adleman erfunden, als sie zu zeigen versuchten, daß Public-Key Kryptographie unmöglich sei. Um den Algorithmus zu erklären, zitieren wir einen wichtigen Satz der Zahlentheorie, den kleinen Satz von Fermat:

Satz 5.5 *Seien $n \in \mathbb{N}$, $n \geq 2$ und $a \in \mathbb{N}$ mit $\text{ggT}(a, n) = 1$. Dann gilt*

$$a^{\varphi(n)} = 1 \pmod n.$$

Dabei bezeichnet $\varphi : \mathbb{N} \rightarrow \mathbb{N}$, $n \mapsto |\{a < n \mid \text{ggT}(a, n) = 1\}|$ die Eulersche φ -Funktion.

Ein Beweis dieses Satzes findet sich beispielsweise in [Bu]

Beispiel 5.6 Wir wollen für zwei ungerade Primzahlen p und q den Wert $\varphi(pq)$ der φ -Funktion bestimmen. Es gibt pq Zahlen $1, 2, \dots, pq$. Von diesen sind nicht teilerfremd zu pq : p Zahlen, die q teilt sowie q Zahlen, die p teilt. Dabei wurde pq doppelt gezählt. Die Zahlen, die weder durch p noch durch q teilbar sind, sind teilerfremd zu pq , da p und q Primzahlen sind. Es ergibt sich

$$\varphi(pq) = pq - (p + q - 1) = (p - 1)(q - 1).$$

Im folgenden wollen wir die Grundidee des RSA-Verfahrens herleiten und begründen, dass es sich um ein kryptographisches Verfahren handelt, d. h. dass verschlüsselte Texte korrekt entschlüsselt werden.

Wir wählen zwei verschiedene ungerade Primzahlen p und q und setzen $n := pq$. Dann gilt gemäß Satz 5.5 für alle $m \in \mathbb{N}$ mit $ggT(m, n) = 1$

$$m^{\varphi(n)} = m^{(p-1)(q-1)} = 1 \pmod{n},$$

also auch für $k \geq 1$

$$m^{k(p-1)(q-1)} = 1^k = 1 \pmod{n}.$$

Es folgt

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

Dies führt zu folgender Idee: Wir suchen ganze Zahlen $d > 1$ und $e > 1$, die

$$de = k(p-1)(q-1) + 1$$

erfüllen. Es soll also gelten

$$de = 1 \pmod{\varphi(n)}.$$

Genau für die $e \leq \varphi(n)$ mit $ggT(e, \varphi(n)) = 1$ existiert ein d , so dass obige Bedingungen erfüllt sind. d kann beispielsweise mit dem erweiterten euklidischen Algorithmus bestimmt werden.

Für e und d gilt somit

$$m^{ed} = m \pmod{n}. \quad (1)$$

Veröffentlichen wir n und e , so kann jeder

$$c := m^e \pmod{n}$$

berechnen. Wir können, da wir d kennen, m aus c zurückgewinnen, indem wir

$$m = c^d \pmod{n}$$

berechnen. Dieses Verfahren beschreibt die Grundidee des RSA Public-Key-Verfahrens. Das Paar (n, e) ist der öffentliche Schlüssel. Jeder kann mit dem öffentlichen Schlüssel einen Klartext m verschlüsseln. Nach heutigem Kenntnisstand ist die Verschlüsselungsfunktion, die Potenzfunktion, eine Einwegfunktion. Mit dem privaten Schlüssel d können Nachrichten korrekt entschlüsselt werden. d muss geheim gehalten werden, es ist die Trapdoor-Information.

Die bisherigen Überlegungen gingen vom kleinen Satz von Fermat aus, bei dem vorausgesetzt wurde, dass m und n teilerfremd sind. Was ist mit $m < n$ und $ggT(m, n) > 1$? Wir überlegen uns, dass Gleichung 1 auch für diese m gilt. Zur Erinnerung: $n = pq$. Für jedes $t < p$ gilt nach Satz 5.5

$$t^{p-1} = 1 \pmod{p}.$$

Außerdem ist

$$q^{p-1} = 1 \pmod{p},$$

also gilt auch für $k \geq 0$

$$t^{k(p-1)(q-1)} = 1 \pmod{p} \text{ und } q^{k(p-1)(q-1)} = 1 \pmod{p}.$$

Wir multiplizieren beide Kongruenzen miteinander und anschließend das Ergebnis mit t :

$$t(tq)^{k(p-1)(q-1)} = t \pmod{p}$$

Bei einer Kongruenz darf man beide Seiten *und* den Modul mit einer festen Zahl multiplizieren. Multiplikation mit q liefert

$$(tq)^{k(p-1)(q-1)+1} = tq \pmod{pq}.$$

Gleichung 1 gilt also auch für $m = tq$, d. h. für die Vielfachen von q . Analog zeigt man, daß sie für die Vielfachen von p gilt. Damit gilt sie für alle m mit $m < n$.

Zusammengefasst ergibt sich folgendes Verfahren:

1. Erzeuge zwei verschiedene Primzahlen p und q .
2. Bilde $n := pq$ und $\varphi(n) = (p-1)(q-1)$
3. Wähle einen öffentlichen Verschlüsselungsexponenten $e > 1$ mit $\text{ggT}(e, \varphi(n)) = 1$. Den öffentlichen Schlüssel bildet das Paar (n, e) .
4. Die Verschlüsselung eines Klartexts $m < n$ erfolgt gemäß $c = m^e \pmod{n}$.
5. Berechne d mit $de = 1 \pmod{\varphi(n)}$. d ist der private Schlüssel.
6. Die Entschlüsselung erfolgt gemäß $m = c^d \pmod{n}$.

5.3.2 RSA Verschlüsselung als Blockchiffre

"Klartexte" und "Geheimtexte" beim RSA-Verfahren mit Modul n sind die Elemente von $\mathbb{Z}_n$. Klartext, der in anderer Form gegeben ist, etwa als Zeichen des ASCII-Codes, muss also zunächst geeignet mit einem Element von $\mathbb{Z}_n$ identifiziert werden.

Das verwendete Alphabet $\mathcal{A}$ enthalte N Buchstaben, die mit den Zahlen $0, 1, \dots, N-1$ identifiziert werden. Wir setzen

$$k := \lfloor \log_N n \rfloor.$$

k Buchstaben $m_1 \dots m_k$ des Alphabets werden zusammengefasst als Zahl m zur Basis N gemäß

$$m = \sum_{i=1}^k m_i N^{k-i}.$$

Es gilt

$$0 \leq m \leq (N-1) \sum_{i=1}^k N^{k-i} = N^k - 1 < n.$$

Der Block m wird verschlüsselt, indem $c = m^e \bmod n$ bestimmt wird. Die Zahl c kann dann wieder zur Basis N geschrieben werden:

$$c = \sum_{i=0}^k c_i N^{k-i}, \quad c_i \in \mathcal{A}, \quad 0 \leq i \leq k.$$

Der Geheimtextblock ist dann $c = c_0 c_1 \dots c_k$. In der beschriebenen Weise bildet RSA Blöcke der Länge k injektiv auf Blöcke der Länge $k+1$ ab.

5.3.3 Angriffe auf RSA

Die in diesem Abschnitt beschriebenen Risiken betreffen jeweils nur Spezialfälle des RSA-Algorithmus. Im Allgemeinen gilt RSA als ungebrochen.

Gleicher Modul für mehrere Benutzer: Man könnte sich überlegen, die Schlüsselerzeugung einer Zentralstelle zu überlassen. Diese erzeugt zu einem festen Modul n Paare e und d von Ver- und Entschlüsselungsexponenten und übergibt sie dem jeweiligen Nutzer. Dass das keine gute Idee ist, sieht man so: Bob kennt den Verschlüsselungsexponenten e_A von Alice sowie seine eigenen Exponenten e_B und d_B . Er berechnet

$$h = \frac{e_B d_B - 1}{ggT(e_A, e_B d_B - 1)}.$$

Die Zahl h ist teilerfremd zu e_A ; also kann man mit Hilfe des erweiterten euklidischen Algorithmus die Vielfachsummandarstellung

$$c \cdot h + d \cdot e_A = 1$$

berechnen. Da h ein Vielfaches von $\varphi(n)$ ist, gilt $ch \bmod \varphi(n) = 0$, also hat Bob eine Zahl d gefunden, die $d \cdot e_A \bmod \varphi(n) = 1$ erfüllt. Er kann durch Potenzieren mit d alle Nachrichten, die für Alice bestimmt sind, entschlüsseln.

Übrigens kann man, falls man n sowie ein Zahlenpaar e, d mit $ed = 1 \bmod \varphi(n)$ kennt, auch die Primfaktoren p und q von n effizient berechnen. Ein Beweis dieser Aussage findet sich in [Bu].

Gleiche Primzahlen in den Moduln verschiedener Benutzer: Falls jemand feststellt, dass sein Modul nicht teilerfremd zu einem fremden Modul ist, so kennt er einen Faktor des fremden Moduls und kann ihn problemlos faktorisieren. Wenn eines Tages Millionen öffentlicher Schlüssel publiziert sind, dann gibt es Millionen von Paaren - da könnte doch wenigstens ein Paar einen gemeinsamen Teiler haben?

Theoretisch ja, wahrscheinlich ist es aber nicht. Wir bezeichnen mit $\pi(x)$ die Anzahl der Primzahlen, die nicht größer als eine reelle Zahl x sind. Dann gilt für $x \geq 17$

$$\pi(x) > x / \ln(x).$$

Da man außerdem zeigen kann, dass $\pi(x) < 1.25506(x / \ln(x))$ ist, erhalten wir, dass es mehr als $3,75 \cdot 10^{98}$ 100-stellige Primzahlen gibt. Das genügt.

Viel wahrscheinlicher ist aber, dass mehrere Anwender durch schlechte Wahl von Zufallszahlen auf die gleiche Primzahl treffen. Dieses Risiko muss durch sorgfältige Implementierung der Zufallszahlenerzeugung ausgeschlossen werden.

Verwendung kleiner Exponenten: Um diese Angriffsart zu beschreiben, benötigen wir einen wichtigen Satz der Zahlentheorie, den chinesischen Restsatz:

Satz 5.7 *Seien p und q zwei teilerfremde natürliche Zahlen, und sei $1 = sp + tq$ die zugehörige Vielfachsummendarstellung. Dann hat das Gleichungssystem*

$$\begin{aligned} x &= a \pmod{p} \\ x &= b \pmod{q} \end{aligned}$$

die Lösung

$$x = b \cdot sp + a \cdot tq.$$

Diese Lösung ist eindeutig modulo $n = pq$.

Ein kleiner Wert des Verschlüsselungsexponenten e spart Rechenzeit bei der Verschlüsselung. Deshalb könnte man für verschiedene Moduln $e = 3$ wählen. Wenn jedoch in einem solchen System drei Teilnehmer die gleiche Nachricht erhalten sollen, so können wir aus dem Kryptogramm leicht den Klartext berechnen: Seien

$$\begin{aligned} y_1 &= m^3 \pmod{n_1}, \\ y_2 &= m^3 \pmod{n_2}, \\ y_3 &= m^3 \pmod{n_3} \end{aligned}$$

die Geheimtexte. Dann können wir mit Hilfe von Satz 5.7 die eindeutige Lösung

$$y = m^3 \pmod{n_1 n_2 n_3}$$

berechnen. Die Nachricht m muss kleiner sein als jeder der drei Moduln n_1, n_2, n_3 , also ist m^3 auch kleiner als $n_1 n_2 n_3$. Dies hat aber zur Folge, dass

$$y = m^3 \pmod{n_1 n_2 n_3} = m^3$$

ist, d. h. man kann m ganz einfach berechnen, indem man in $\mathbb{Z}$ die dritte Wurzel aus y zieht.

Multiplikativität: Für den öffentlichen Schlüssel (n, e) eines RSA-Systems gilt

$$m_1^e m_2^e \pmod{n} = (m_1 m_2)^e \pmod{n}.$$

Man kann daher die Nachricht $m_1 m_2$ verschlüsseln, ohne m_1 und m_2 zu kennen. Normalerweise erkennt man diesen Angriff daran, dass die Nachrichten m_1 und m_2 einen Sinn ergeben, die Nachricht $m_1 m_2$ aber nicht. Das liegt daran, dass durch die Multiplikation die Redundanz der Nachrichten, z. B. die Redundanz der natürlichen Sprache, zerstört wird.

Für die Sicherheit des RSA-Verfahrens ist die Größe von n entscheidend. Computer werden immer schneller und damit können Faktorisierungsalgorithmen immer schneller ausgeführt werden. Aber es ist auch nicht bekannt, ob das Faktorisierungsproblem wirklich so schwierig ist. Sollte einmal ein effizienter Faktorisierungsalgorithmus gefunden werden, so ist RSA gebrochen.

5.4 Diffie-Hellman-Schlüsselaustausch

Neben dem Faktorisierungsproblem gibt es ein weiteres zahlentheoretisches Problem, das nach heutigem Kenntnisstand eine Einwegfunktion liefert, nämlich das Problem des diskreten Logarithmus.

Wir betrachten das Problem des diskreten Logarithmus in den Gruppen $\mathbb{Z}_p^*$ für eine Primzahl p . Es ist bekannt, dass für jede Primzahl p die Gruppe $\mathbb{Z}_p^*$ eine **zyklische Gruppe** ist, d. h. es gibt mindestens ein $g \in \mathbb{Z}_p^*$ mit

$$\mathbb{Z}_p^* = \{g^n \mid n = 1, \dots, p-1\}.$$

Jedes $g \in \mathbb{Z}_p^*$ heißt ein **erzeugendes Element**. Sei g ein erzeugendes Element von $\mathbb{Z}_p^*$. Dann ist die diskrete Exponentialfunktion

$$\exp_g : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*, \quad x \mapsto g^x$$

bijektiv. Ihre Umkehrfunktion wird als **diskreter Logarithmus** bezeichnet. Der diskrete Logarithmus von y zur Basis g ist die kleinste nicht negative ganze Zahl x , für die

$$y = g^x$$

gilt.

Es gibt einige wichtige Public-Key-Verfahren, die auf dem Problem des diskreten Logarithmus basieren. Wir beschreiben als Beispiel den Diffie-Hellman-Schlüsselaustausch. Dieses wohl erste veröffentlichte Public-Key-Verfahren überhaupt funktioniert so:

1. Alice wählt eine Primzahl p , ein erzeugendes Element g von $\mathbb{Z}_p^*$ und eine Zahl $a \in \mathbb{Z}_{p-1}$. Sie bestimmt $A = g^a \bmod p$.
2. Um einen Schlüssel mit Bob zu vereinbaren, sendet sie ihm die Daten (p, g, A) .
3. Bob wählt eine Zahl $b \in \mathbb{Z}_{p-1}$, bestimmt $B = g^b$ und sendet B an Alice zurück.
4. Der vereinbarte Schlüssel ist

$$K = B^a = A^b.$$

Dabei können p und g vorab zum öffentlich zugänglichen Schlüssel von Alice gemacht werden. Dann muss nur A an Bob übermittelt werden. Die Zahl a muss Alice natürlich geheimhalten - sie öffnet die Falltür für ihre Bestimmung von K . Gleiches gilt für b auf Bobs Seite.

Die Sicherheit des Verfahrens beruht darauf, dass sich der diskrete Logarithmus von A bezüglich g nur schwer bestimmen lässt. Es wäre allerdings auch möglich, dass ein Weg gefunden wird, g^{ab} aus g^a und g^b zu bestimmen, ohne a oder b zu berechnen. Die Äquivalenz dieses "Diffie-Hellman-Problems" zu dem der Bestimmung des diskreten Logarithmus ist nicht bekannt.

6 Faktorisierung

6.1 Probedivision

Gemäß Satz 5.1 wissen wir, daß jede natürliche Zahl n in Primfaktoren zerfällt. Wie aber finden wir sie? Um die kleinen Primfaktoren von n zu finden, berechnet man die Liste aller Primzahlen unter einer festen Schranke B . Dafür kann man das **Sieb des Erathostenes** verwenden. Dann bestimmt man für jede Primzahl p in dieser Liste den maximalen Exponenten $e(p)$, für den p die Zahl n teilt. Eine typische Schranke ist $B = 10^6$.

6.2 Das quadratische Sieb

Das quadratische Sieb ist einer der effizientesten Faktorisierungsalgorithmen. Die folgende Beschreibung des quadratischen Siebs ist [Bu] entnommen.

Es soll die zusammengesetzte ganze Zahl n faktorisiert werden. Im folgenden wird beschrieben, wie man einen echten Teiler von n findet, d. h. einen Teiler, der nicht gleich n ist und nicht gleich eins. Im quadratischen Sieb werden ganze Zahlen x und y bestimmt, für die

$$x^2 = y^2 \pmod{n}$$

und

$$x \neq \pm y \pmod{n}$$

ist. Dann ist nämlich n ein Teiler von $x^2 - y^2 = (x - y)(x + y)$, aber weder von $x - y$ noch von $x + y$. Also ist $d = \text{ggT}(x - y, n)$ ein echter Teiler von n und das Berechnungsziel ist erreicht.

Beispiel 6.1 Sei $n = 7429$, $x = 227$, $y = 210$. Dann ist $x^2 - y^2 = n$, $x - y = 17$, $x + y = 437$. Daher ist $\text{ggT}(x - y, n) = 17$. Das ist ein echter Teiler von n .

Wie bestimmt man x und y im Quadratischen Sieb? Sei

$$m = \lfloor \sqrt{n} \rfloor$$

und

$$f(X) = (X + m)^2 - n.$$

Wir erläutern das Verfahren an einem Beispiel:

Beispiel 6.2 Wie in Beispiel 6.1 sei $n = 7429$. Dann ist $m = 86$ und $f(X) = (X + 86)^2 - 7429$. Es gilt

$$\begin{aligned} f(-3) &= 83^2 - 7429 = -540 = -1 \cdot 2^2 \cdot 3^3 \cdot 5 \\ f(1) &= 87^2 - 7429 = 140 = 2^2 \cdot 5 \cdot 7 \\ f(2) &= 88^2 - 7429 = 315 = 3^2 \cdot 5 \cdot 7. \end{aligned}$$

Hieraus folgt

$$\begin{aligned} 83^2 &= -1 \cdot 2^2 \cdot 3^2 \cdot 5 \pmod{7429} \\ 87^2 &= 2^2 \cdot 5 \cdot 7 \pmod{7429} \\ 88^2 &= 3^2 \cdot 5 \cdot 7 \pmod{7429} \end{aligned}$$

Multipliziert man die letzten beiden Kongruenzen, so erhält man

$$(87 \cdot 88)^2 = (2 \cdot 3 \cdot 5 \cdot 7)^2 \pmod{n}.$$

Man kann also

$$x = 87 \cdot 88 \pmod{n} = 227, \quad y = 2 \cdot 3 \cdot 5 \cdot 7 \pmod{n} = 210$$

setzen. Das sind die Werte für x und y aus Beispiel 6.1.

In Beispiel 6.2 werden Zahlen s ausgegeben, für die $f(s)$ nur kleine Primfaktoren hat. Es wird ausgenutzt, daß

$$(s + m)^2 = f(s) \pmod{n}$$

ist, und es werden Kongruenzen dieser Form ausgewählt, deren Produkt auf der linken und rechten Seite ein Quadrat ergibt. Auf der linken Seite einer solchen Kongruenz steht ohnehin ein Quadrat. Auf der rechten Seite ist die Primfaktorzerlegung bekannt. Man erhält also ein Quadrat, wenn die Exponenten aller Primfaktoren und der Exponent von -1 gerade sind.

Wir erklären als nächstes, wie geeignete Kongruenzen ausgewählt werden. In Beispiel 6.2 kann man direkt sehen, welche Kongruenzen multipliziert werden müssen, damit das Produkt der rechten Seiten der Kongruenzen ein Quadrat ergibt. Bei großen Zahlen n muss man die geeigneten Kongruenzen aus mehr als 100 000 möglichen Kongruenzen auswählen. Dann wendet man lineare Algebra an. Dies wird im nächsten Beispiel illustriert.

Beispiel 6.3 Wir zeigen, wie die Auswahl der geeigneten Kongruenzen in Beispiel 6.2 durch Lösung eines linearen Gleichungssystems erfolgt. Drei Kongruenzen stehen zur Wahl. Daraus sollen die Kongruenzen so ausgewählt werden, daß das Produkt der rechten Seiten ein Quadrat ergibt. Man sucht also Zahlen $\lambda_i \in \{0, 1\}$, $1 \leq i \leq 3$, für die

$$(-1 \cdot 2^2 \cdot 3^3 \cdot 5)^{\lambda_1} \cdot (2^2 \cdot 5 \cdot 7)^{\lambda_2} \cdot (3^2 \cdot 5 \cdot 7)^{\lambda_3} =$$

$$(-1)^{\lambda_1} \cdot 2^{2\lambda_1+2\lambda_2} \cdot 3^{3\lambda_1+2\lambda_3} \cdot 5^{\lambda_1+\lambda_2+\lambda_3} \cdot 7^{\lambda_2+\lambda_3}$$

ein Quadrat ist. Diese Zahl ist genau dann ein Quadrat, wenn der Exponent von -1 und die Exponenten aller Primzahlen gerade sind. Man erhält also das folgende Kongruenzensystem:

$$\begin{aligned}\lambda_1 &= 0 \pmod{2} \\ 2\lambda_1 + 2\lambda_2 &= 0 \pmod{2} \\ 3\lambda_1 + 2\lambda_3 &= 0 \pmod{2} \\ \lambda_1 + \lambda_2 + \lambda_3 &= 0 \pmod{2} \\ \lambda_2 + \lambda_3 &= 0 \pmod{2}.\end{aligned}$$

Die Koeffizienten der λ_i kann man modulo 2 reduzieren. Man erhält dann das vereinfachte System

$$\begin{aligned}\lambda_1 &= 0 \pmod{2} \\ \lambda_1 + \lambda_2 + \lambda_3 &= 0 \pmod{2} \\ \lambda_2 + \lambda_3 &= 0 \pmod{2}.\end{aligned}$$

Daraus ergibt sich die Lösung

$$\lambda_1 = 0, \quad \lambda_2 = \lambda_3 = 1.$$

Wir skizzieren kurz, wie das quadratische Sieb geeignete Kongruenzen im allgemeinen findet. Man wählt eine natürliche Zahl B . Gesucht werden kleine ganze Zahlen s , für die $f(s)$ nur Primfaktoren in der **Faktorbasis**

$$F(B) = \{p \in \mathbb{Z} \mid p \text{ Primzahl}, p \leq B\} \cup \{-1\}$$

hat. Solche Werte $f(s)$ heißen B -glatt. Hat man so viele Zahlen gefunden, wie die Faktorbasis Elemente hat, so stellt man das entsprechende lineare Kongruenzensystem auf und löst es. Da das Kongruenzensystem tatsächlich ein lineares Gleichungssystem über dem Körper $\mathbb{Z}_2$ ist, kann man zu seiner Lösung den Gauß-Algorithmus verwenden.

Es bleibt noch zu klären, wie die Zahlen s gefunden werden, für die $f(s)$ B -glatt ist. Man könnte für $s = 0, \pm 1, \pm 2, \pm 3 \dots$ den Wert $f(s)$ berechnen und dann durch Probedivision ausprobieren, ob $f(s)$ B -glatt ist. Das aber sehr aufwendig. Um herauszufinden, daß $f(s)$ nicht B -glatt ist, muss man nämlich durch alle Primzahlen $p \leq B$ dividieren. Die Faktorbasen sind sehr groß und daher kann die Probedivision sehr lange dauern. Schneller geht ein Siebverfahren.

Wir beschreiben eine vereinfachte Form des Siebverfahrens. Man fixiert ein Siebintervall

$$S = \{-C, -C + 1, \dots, 0, 1, \dots, C\}.$$

Gesucht werden alle $s \in S$ für die $f(s)$ B -glatt ist. Man berechnet zuerst alle Werte $f(s)$, $s \in S$. Für jede Primzahl p der Faktorbasis dividiert man alle Werte

$f(s)$ durch die höchstmögliche p -Potenz. Die B -glatten Werte $f(s)$ sind genau diejenigen, bei denen eine 1 oder -1 übrigbleibt.

Um herauszufinden, welche Werte $f(s) = (s + m)^2 - n$ durch eine Primzahl p der Faktorbasis teilbar sind, bestimmt man zuerst die Zahlen $s \in \{0, 1, \dots, p-1\}$ für die $f(s)$ durch p teilbar ist. Da das Polynom $f(X)$ höchstens zwei Nullstellen modulo p hat, sind das entweder zwei, eine oder keine Zahl s . Für kleine Primzahlen kann man die Nullstellen durch ausprobieren finden. Ist p groß, muss man andere Methoden anwenden. Geht man von diesen Nullstellen in Schritten der Länge p nach rechts und links durch das Siebintervall, so findet man alle s -Werte, für die $f(s)$ durch p teilbar ist. Diesen Vorgang nennt man **Sieb** mit p . Man dividiert dabei nur die teilbaren Werte $f(s)$. Es gibt keine erfolglosen Probedivisionen.

Beispiel 6.4 Wie in den Beispielen 6.1 und 6.2 seien $n = 7429$, $m = 86$ und $f(X) = (X + 86)^2 - 7429$. Als Faktorbasis wähle die Menge $\{2, 3, 5, 7\} \cup \{-1\}$ und als Siebintervall die Menge $\{-3, -2, \dots, 3\}$. Das Sieb sieht folgendermaßen aus:

s	-3	-2	-1	0	1	2	3
$(s + m)^2 - n$	-540	-373	-204	-33	140	315	492
Sieb mit 2	-135		-51		35		123
Sieb mit 3	-5		-17	-11		35	41
Sieb mit 5	-1				7	7	
Sieb mit 7					1	1	

Das Sieb kann noch sehr viel effizienter gestaltet werden. Das wird hier aber nicht weiter beschrieben.

7 Diskrete Logarithmen

Einige Public-Key-Kryptosysteme basieren auf der Schwierigkeit, diskrete Logarithmen in gewissen endlichen Gruppen zu berechnen, etwa der Diffie-Hellman-Schlüsselaustausch aus Abschnitt 5.4. Wir stellen in den folgenden Abschnitten zwei Verfahren zur Berechnung diskreter Logarithmen in $\mathbb{Z}_p^*$ vor. Dies sind zwar nicht die besten derzeit bekannten Verfahren, aber es ist auch kein Algorithmus bekannt, der im Allgemeinen diskrete Logarithmen effizient berechnet.

7.1 Enumeration

Die einfachste Methode, den diskreten Logarithmus x aus $y = g^x$ zu berechnen besteht darin, für $x = 0, 1, 2, \dots$ zu prüfen, ob diese Gleichung erfüllt ist. Sobald dies der Fall ist, ist der diskrete Logarithmus gefunden. Für kryptographische Verfahren ist diese Methode allerdings nicht anwendbar, weil sie viel zu langsam ist.

7.2 Der Babystep-Giantstep-Algorithmus

Wir stellen nun eine schneller Methode zur Berechnung diskreter Logarithmen in $\mathbb{Z}_p^*$ vor. Dazu setzen wir

$$m := \lfloor \sqrt{p} \rfloor + 1$$

und

$$x = qm + r \quad \text{mit } 0 \leq r \leq m.$$

Dabei ist r der Rest und q der Quotient der Division von x durch m . Der Babystep-Giantstep-Algorithmus berechnet q und r nach folgendem Prinzip: Es ist

$$g^{qm+r} = g^x = y$$

und damit

$$(g^m)^q = yg^{-r}.$$

Die Menge der **Babysteps** wird berechnet als

$$B = \{(yg^{-r}, r) : 0 \leq r < m\}.$$

Findet man dabei ein Paar $(1, r)$, so ist $x = r$ und das Problem ist gelöst. Andernfalls berechnet man die **Giantsteps** folgendermaßen: Man berechnet

$$d = g^m$$

und prüft, ob für $q = 1, 2, 3, \dots$ das Gruppenelement d^q als erste Komponente eines Elements von B vorkommt, ob also ein Paar (d^q, r) zu B gehört. Dann gilt nämlich

$$yg^{-r} = d^q = g^{qm}$$

und der diskrete Logarithmus

$$x = qm + r$$

ist gefunden.

Um den Aufwand dieses Algorithmus abzuschätzen, kann man folgenden Satz zeigen:

Satz 7.1 *Der Babystep-Giantstep-Algorithmus benötigt $O(\sqrt{p})$ Multiplikationen und Vergleiche in $\mathbb{Z}_p^*$. Er muss $O(\sqrt{p})$ viele Elemente von $\mathbb{Z}_p^*$ speichern.*

Für sehr große Primzahlen (etwa $p > 2^{160}$) ist der Algorithmus in der Praxis nicht mehr einsetzbar. Es gibt allerdings auch noch bessere Algorithmen zum Auffinden diskreter Logarithmen.

8 Komplexitätstheorie

Die Komplexitätstheorie beschäftigt sich mit der Frage, wie aufwendig es ist, die Lösung eines Problems zu finden. Selbstverständlich wächst der Aufwand, eine Lösung eines Problems zu finden, wenn das Problem selbst "wächst", d. h. wenn seine Eingangsparameter wachsen. Wie stark aber wächst dieser Aufwand?

In der Komplexitätstheorie interessiert man sich nur für die bestmöglichen Algorithmen zur Lösung eines Problems. Dabei ist folgendes zu beachten: Man betrachtet den "worst case", d. h. das Problem, für das der Algorithmus am längsten braucht.

Die wichtigste Unterscheidung in der Komplexitätstheorie ist die in Probleme mit **polynomialer** und Probleme mit **nichtpolynomialer** Komplexität. Im ersten Fall wächst der Aufwand zur Lösung eines Problems polynomial, d. h. es gibt ein Polynom f , so dass die Komplexität höchstens $f(n)$ ist, wenn n die Länge der Eingangsparameter ist. Im zweiten Fall wächst die Komplexität schneller als polynomial, etwa exponentiell. Wir verdeutlichen diesen Unterschied in folgender Tabelle aus [Be]:

Eingabelänge/ Zeitkomplexität (μs)	20	30	40	50	60
x	$2 \cdot 10^{-5}$ Sek	$3 \cdot 10^{-5}$ Sek	$4 \cdot 10^{-5}$ Sek	$5 \cdot 10^{-5}$ Sek	$6 \cdot 10^{-5}$ Sek
x^2	$4 \cdot 10^{-4}$ Sek	$9 \cdot 10^{-4}$ Sek	$1,6 \cdot 10^{-3}$ Sek	$2,5 \cdot 10^{-3}$ Sek	$3,6 \cdot 10^{-3}$ Sek
x^5	3,2 Sek	24,3 Sek	1,7 Min	5,2 Min	13,0 Min
2^x	1 Sek	17,9 Min	12,7 Tage	37,7 Jahre	366 Jahrhunderte
3^x	5,2 Sek	6,5 Jahre	3855 Jahrhunderte	$2 \cdot 10^8$ Jahrhunderte	$1,3 \cdot 10^{13}$ Jahrhunderte

Man sieht an dieser Tabelle, dass sich exponentielle Zeitkomplexität schon für kleine Eingabewerte sehr negativ auf die Rechenzeit auswirkt. Auch schnellere Computer schaffen hier kaum Abhilfe. Die Verdoppelung der Rechengeschwindigkeit von Computern wirkt sich auf die Lösbarkeit exponentieller Probleme fast nicht aus. Die Tabelle verdeutlicht dies: Kann man mit den alten Computern in 12,7 Tagen ein Problem der Länge 40 lösen, so sind es bei den neuen Computern gerade einmal die Probleme der Länge 41.

Man fasst Probleme ähnlicher Komplexität zu Klassen zusammen. Die beiden wichtigsten Klassen sind die Klassen **P** und **NP**.

In die Klasse **P** fallen alle Probleme, die mit polynomialem Zeitaufwand lösbar sind. Dazu gehören die Addition, die Multiplikation und das Potenzieren natürlicher Zahlen.

In die Klasse **NP** fallen alle Probleme, bei denen eine gegebene Lösung mit poly-

nomialem Zeitaufwand verifiziert werden kann. Dass es viel schwieriger sein kann, die Lösung eines Problems zu finden, als eine gegebene Lösung zu überprüfen, wissen wir schon. Beispiele für derartige Probleme sind die Einwegfunktionen der Public-Key-Kryptosysteme:

- Das Faktorisieren großer Zahlen ist schwierig, die Überprüfung einer solchen Faktorisierung dagegen einfach: Man muss die gegebenen Faktoren nur miteinander multiplizieren und das Produkt mit der gegebenen Zahl vergleichen.
- Die diskrete Exponentialfunktion ist einfach, während das Problem des diskreten Logarithmus sehr schwierig ist.

Die Klasse $\mathbf{P}$ ist in der Klasse $\mathbf{NP}$ enthalten. Ein berühmtes offenes Problem ist die Frage, ob $\mathbf{P} = \mathbf{NP}$ gilt oder nicht.

Die Klasse $\mathbf{NP}$ enthält sogenannte "vollständige" Probleme. Dies sind Probleme, für die folgendes gilt: Wenn es einen effizienten (polynomialen) Algorithmus für ein $\mathbf{NP}$ -vollständiges Problem gibt, dann existieren für alle Probleme in $\mathbf{NP}$ effiziente Algorithmen. Falls also ein $\mathbf{NP}$ -vollständiges Problem in $\mathbf{P}$ läge, so würde $\mathbf{P} = \mathbf{NP}$ gelten. In diesem Sinne sind die $\mathbf{NP}$ -vollständigen Probleme die schwierigsten Probleme in $\mathbf{NP}$.

In der Kryptographie werden Verfahren entwickelt, bei denen die "guten" Teilnehmer Probleme aus $\mathbf{P}$ lösen müssen, während ein Angreifer Probleme aus $\mathbf{NP}$ zu lösen hat.

Literatur

- [Ad] D. Adams, *Per Anhalter durch die Galaxis*, Heyne, 2000.
- [Bi] E. Biham, A. Shamir, *Differential Cryptanalysis of DES-like Cryptosystems*
Journal of Cryptology, Vol. 4 No.1, 1991
- [Bu] J. Buchmann, *Einführung in die Kryptographie*, Springer, 2001.
- [Be] A. Beutelspacher, *Moderne Verfahren der Kryptographie*, Vieweg, 2001.
- [Cr] <http://www.cryptool.de/>. Unter dieser Internet-Adresse kann man das
Cryptool beziehen, in dem viele klassische und moderne Verfahren imple-
mentiert sind.
- [Fo] O. Forster, *Algorithmische Zahlentheorie*, Vieweg, 1996.