Kryptologie

Eine Einführung

Reinhard Völler

30. Juni 2003

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Zitronensaft und kahle Köpfe

In den folgenden Kapiteln wollen wir uns mit der Frage beschäftigen, wie man vertrauliche Nachrichten übermitteln kann. Ebenso wichtig ist natürlich auch die Frage, wie man erreichen kann, daß eine Nachricht **unverändert** beim Empfänger ankommt.

Im Prinzip hat man zwei Möglichkeiten vorzugehen: man kann versuchen, die **Existenz** der Nachricht zu **verheimlichen**, oder man verhindert, dass ein Unbefugter in der Nachricht einen Sinn erkennen kann, dadurch, dass man die Nachricht **verschlüsselt**.

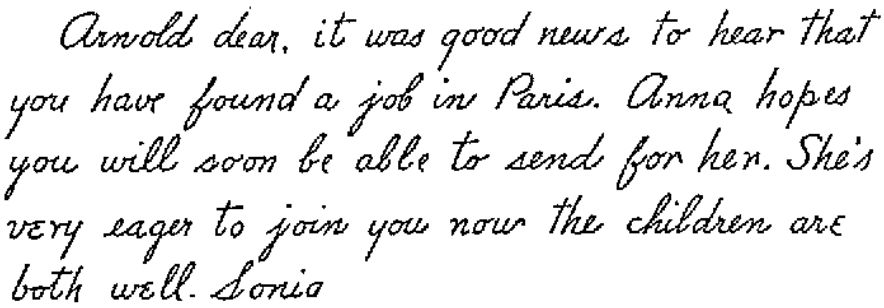
1.1.1 Kryptographie

- Der Begriff *cryptographia - secrecy in writing -* wurde 1661 erstmals von *John Williams* verwendet.
- Nachrichten werden **unlesbar** gemacht.
- Man spricht hier von **offenen Geheimschriften** .

Wir werden uns fast ausschließlich mit **kryptographischen** Verfahren beschäftigen.

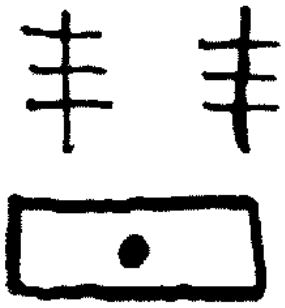
1.1.2 Steganographie

- Der Begriff wurde zuerst 1499 von *Trithemius* verwendet.
- Die **Existenz** einer Nachricht wird verborgen.
- Im Gegensatz zu den offenen Geheimschriften, werden die steganographischen Verfahren als **gedeckte Geheimschriften** bezeichnet.
- Mit **technischer Steganographie** sind Tricks wie die Verwendung von Geheimtinte, Zitronensaft, Microdots oder Schnelltelegraphie (spurts) gemeint. Der Grieche *Histiaeus* schor einem Sklaven den Kopf, schrieb ihm eine Nachricht auf die Kopfhaut und ließ ihn die Nachricht überbringen, nachdem die Haare wieder nachgewachsen waren. Wegen der geringen Übertragungsrate hat sich dieses Verfahren nicht durchsetzen können.



In der obigen Abbildung [1] ist ein geheimer Zahlencode versteckt: die einzelnen Zahlen werden durch die Anzahl der Buchstaben dargestellt, die einem Aufschwung vorangehen, also 3 3 5 1 5 1 4 1 2 3.

1.1.3 Maskierung



[1]

Bis in die heutige Zeit haben sich **Zinken** erhalten. Das sind Zeichen mit einer vordefinierten Bedeutung, die von mehr oder weniger vertrauenswürdigen Zeitgenossen z.B. an Haustüren angebracht wurden und vor unfreundlichen Hausbesitzern oder Polizisten warnen.

Auch durch Sprache kann man Nachrichten **maskieren** . Bekannt sind hier Spezialsprachen, die auch als **Jargon** bezeichnet werden. Eingeweihte wissen natürlich wovon die Rede ist:

- Koks, Schnee
- Kies, Kohle, Mäuse
- Loch, Bulle

1.1.4 Stichworte

Tag / Uhrzeit / Ort und Art der Unterkunft	Darstellung der Ereignisse (Dabei wichtig: Beurteilung der Lage (Feind- und eigene), Eingangs- und Abgangszeiten von Meldungen und Befehlen)
5.6.44	Am 1., 2. und 3.6.44 ist durch die Nast innerhalb der "Messages personelles" der französischen Sendungen des britischen Rundfunks folgende Meldung abgehört worden : "Les sanglots longs des violons de l'automne ". Nach vorhandenen Unterlagen soll dieser Spruch am 1. oder 15. eines Monats durchgegeben werden, nur die erste Hälfte eines ganzen Spruches darstellen und ankündigen, dass binnen 48 Stunden nach Durchgabe der zweiten Hälfte des Spruches, gerechnet von 00.00 Uhr des auf die Durchgabe folgenden Tages ab, die anglo-amerikanische Invasion beginnt.
21.15 Uhr	Zweite Hälfte des Spruches "Blessent mon coeur d'une langueur monotone" wird durch Nast abgehört.
21.20 Uhr	Spruch an Ic-AO durchgegeben. Danach mit Invasionsbeginn ab 6.6. 00.00 Uhr innerhalb 48 Stunden zu rechnen. Überprüfung der Meldung durch Rückfrage beim Militärbefehlshaber Belgien/Nordfrankreich in Brüssel (Major von Wangenheim).
22.00 Uhr	Meldung an O.B. und Chef des Generalstabes.
22.15 Uhr	Weitergabe gemäss Fernschreiben (Anlage 1) an Generalkommandos. Mündliche Weitergabe an 16. Flak-Division.

[10]

Am 1. Juni 1944 wurde in den 21-Uhr-Nachrichten der **BBC** die erste Hälfte der ersten Strophe des Gedichts *Chanson d'Automne* von *Verlaine* gesendet. Die zweite Hälfte folgte am 5. Juni 1944 und kündigte die Invasion der Alliierten an. Aus ungeklärten Gründen wurde die deutsche 7. Armee nicht alarmiert, obwohl der deutschen Seite die Bedeutung dieses **Stichworts** bekannt war.

1.1.5 Getarnte Geheimschriften

Anders als bei Stichworten enthalten **getarnte Geheimschriften** die zu übermittelnde Nachricht selbst, verdecken sie aber durch **Blender** oder **Füllzeichen** . Auch hierfür zwei Beispiele:

Worthie Sir John: — Hope, that is ye beste comfort of ye afflicted, cannot much, I fear me, help you now. That I would saye to you, is this only: if ever I may be able to requite that I do owe you, stand not upon asking me. 'Tis not much that I can do: but what I can do, bee ye verie sure I wille. I knowe that, if dethe comes, if ordinary men fear it, it frights not you, accounting it for a high honour, to have such a rewarde of your loyalty. Pray yet that you may be spared this soe bitter, cup. I fear not that you will grudge any sufferings; only if bie submission you can turn them away, 'tis the part of a wise man. Tell me, an if you can, to do for you anythinge that you wolde have done. The general goes back on Wednesday. Restinge your servant to command. — R. T. [1]

Was verbirgt sich hinter der folgenden Nachricht, die im 1. Weltkrieg abgefangen wurde?

PRESIDENTS EMBARGO RULING

SHOULD HAVE IMMEDIATE NOTICE. GRAVE

SITUATION AFFECTING INTERNATIONAL LAW.

STATEMENT FORESHADOWS RUIN OF MANY

NEUTRALS. YELLOW JOURNALS UNIFYING

NATIONAL EXCITEMENT IMMENSELY

1.2 Aufgaben und Anwendungsbereiche

Klassisches Ziel der Kryptographie ist die **Geheimhaltung** von Nachrichten. Kein Wunder, daß in der Vergangenheit die wichtigsten Kunden der Kryptographen Militärs und Regierungen waren.

In neuerer Zeit gewinnen mit dem Ausbau der Datennetze zunehmend Anwendungsbereiche in der Wirtschaft an Bedeutung. Hier geht es nicht nur um die **Vertraulichkeit** von Daten, sondern insbesondere um den Schutz vor **Verfälschung** bzw. die **Authentizität**.

Beispiele:

- Verschlüsselung von Telefongesprächen
- Pay-TV
- Vertraulichkeit von e-mail
- Authentisierung von e-mail
- Finanztransaktionen
- Kreditkartenkäufe online
- Identifizierung gegenüber Rechnersystemen

1.3 Terminologie

Klartext

Klartextalphabet : Zeichenvorrat P

Menge der Klartextworte: P^*

Geheimtext :

Geheimtextalphabet , Code : Zeichenvorrat C

Menge der Geheimtextworte: C^*

leeres Wort : ε

Menge aller n-Tupel aus V: V^n

Menge aller Tupel der Länge $L <= n$: $V^{(n)}$

P und C sind in der Regel endlich.

Chiffrierung : **injektive** Relation:

$R : P^* \longrightarrow C^*$

injektiv : $(x, z) \in R \wedge (y, z) \in R \Rightarrow x = y$

Rechtsfaser von x: $H_x = \{y \in C^* : xRy\}$

Ist die Relation R auch **rechtseindeutig**, so ist H_x höchstens einelementig.

Ist H_x nicht einelementig oder leer, so heißen die Elemente aus H_x **Homophone** für x.

Meistens gilt $H_\varepsilon = \{\varepsilon\}$.

Enthält H_ε auch von ε verschiedene Elemente, so nennt man diese **Blender** oder **Blindtexte** .

Zeichenvorräte

Mächtigkeit des Zeichenvorrats V: $\mid V \mid$

1565 **Porta** $Z_{20} = \{a, b, ..., i, l, ..., t, v, z\}$

1600 $Z_{24} = Z_{20} \cup \{k, w, x, y\}$

1900 $Z_{26} = Z_{24} \cup \{j, u\}$

Die Alphabete P und C können natürlich auch verschieden sein. Ein hübsches Beispiel findet man bei *E. A. Poe* in der Erzählung *Der Goldkäfer*:

53‡‡†305))6*;4826)4‡.)4‡);806*;48†8¶60
))85;1‡(;:‡*8†83(88)5*†;46(;88*96*?;8)*
‡(;485);5*†2:*‡(;4956*2(5*—4)8¶8*;40
9285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;
4)485†528806*81(‡9;48;(88;4(‡?34;48)4
‡;161;:188;‡?;

[1]

Chiffriergleichungen :

Verschlüsseln: $E(P) = C$

Entschlüsseln: $D(C) = P$

Es muss gelten: $D(E(P)) = P$

Verwendung eines Schlüssels k:

Ein **Schlüssel** dient zur Auswahl eines Chiffrierschrittes.

Symmetrisches Verfahren: $D_k(E_k(P)) = P$

Bei einem symmetrischen Verfahren wird zum Ver- und Entschlüsseln der gleiche Schlüssel verwendet.

Asymmetrisches Verfahren: $D_{k_2}(E_{k_1}(P)) = P$

Bei asymmetrischen Verfahren wird zum Entschlüsseln ein anderer Schlüssel als zum Verschlüsseln benutzt. Dies kann große Vorteile haben, da weniger Schlüssel ausgetauscht werden müssen.

1.4 Wie man die Post anderer Leute liest

Es werden nur selten **absolut sichere** Verfahren benötigt. In der Regel ist es ausreichend, dem unberufenen Entzifferer das Leben so lange zu erschweren, bis die verschlüsselte Nachricht wertlos geworden ist. Aber wir werden auch noch ein **beweisbar** absolut sicheres Verfahren kennenlernen.

In der Vergangenheit wurden Verfahren häufig voreilig als absolut sicher bezeichnet. Ein Problem der Kryptographen war dabei, daß die verwendeten Verfahren auch für wenig mathematisch geschulten Anwendern (Militärs und Politiker) handhabbar sein mussten.

Später werden wir das **VIGENÈRE** -Verfahren kennenlernen, das noch 1917 im *Scientific American* als absolut sicher bezeichnet wurde. Ich denke, dass es Ihnen gelingen wird, dieses Verfahren zu "knacken".

1.4.1 Ciphertext-Only Attack

In diesem Fall hat der Kryptoanalytiker einen oder mehrere Geheimtexte, aus denen er auf den verwendeten Algorithmus bzw. Klartext schließen soll.

$C_1 = E_k(P_1), ..., C_n = E_k(P_n)$

Gesucht: $P_1, P_2, ..., P_n$

Durch häufig wechselnde Schlüssel kann man dem Gegner das Leben heftig erschweren. In seinem Buch *La cryptographie militaire* formuliert der niederländische Philologe *Jean Guillaume Hubert Victor Francois Alexandre Auguste* **Kerkoffs von Nieuwenhof** ein wichtiges Prinzip, das als **Kerckhoffs Maxime** bekannt geworden ist:

> *Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Algorithmus abhängen. Die Sicherheit gründet sich nur auf die Geheimhaltung des Schlüssels.*

Ein Verfahren muss also auch dann noch sicher sein, wenn der Gegner den Algorithmus kennt oder eine Verschlüsselungsmaschine erbeutet hat. Im 2. Weltkrieg besaßen die Engländer zwar Exemplare der **ENIGMA**, mussten aber dennoch hart arbeiten, um die täglichen Funksprüche der deutschen U-Boote entschlüsseln zu können.

Heute werden viele Algorithmen öffentlich diskutiert, um Schwachstellen aufdecken zu können.

1.4.2 Known-Plaintext Attack

In diesem Fall sind ein oder mehrere
zusammengehörige Klartext-/Geheimtextpaare bekannt.

$(P_1, C_1 = E_k(P_1)), ..., (P_n, C_n = E_k(P_n))$

Gesucht ist E_k, um zukünftige Texte entziffern zu können.

1.4.3 Chosen-Plaintext Attack

Hier hat der Angreifer die Möglichkeit, Klartexte vorzugeben, um so zugehörige Geheimtext zu provozieren. Ein bekanntes Beispiel ist der Angriff britischer Flieger auf Seezeichen, der dann den deutschen verschlüsselten Spruch: "ERLOSCHEN IST LEUCHTTONNE" bewirkte.

$(P_1, C_1 = E_k(P_1)), ..., (P_n, C_n = E_k(P_n))$

Die P_i sind wählbar!

Gesucht ist E_k, um zukünftige Texte entziffern zu können.

1.4.4 Adaptive-Chosen-Plaintext Attack

Wie im letzten Abschnitt, nur dass der Angreifer die gewonnenen Informationen aus dem Schritt i verwendet, im die Klartexte $P_{i+1}, ...$ zu wählen.

Kapitel 2

Klassische Verfahren

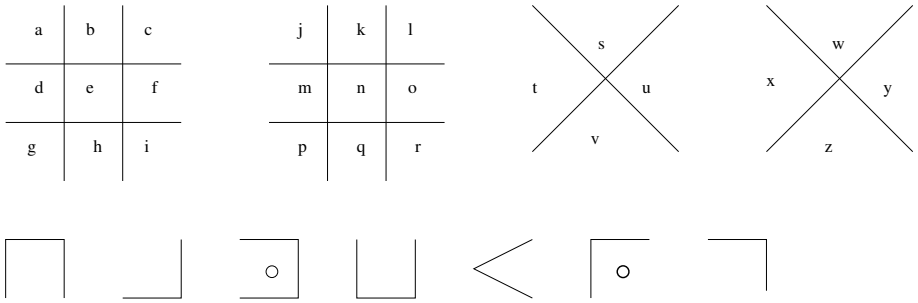
Wir werden uns jetzt einigen einfachen klassischen Verfahren zuwenden, bei denen jeweils ein Klartextzeichen durch ein Geheimtextzeichen ersetzt wird. Es wird sich zeigen, dass dieses Vorgehen, obwohl sehr viele verschiedene Verschlüsselungsmöglichkeiten existieren, einem ernsthaften Angriff nicht lange standhält.

2.1 Einfache Substitutionen

Betrachten wir zunächst einige **monopartite einfache Substitutionen**.

2.1.1 Freimaurerchiffre

Hier werden als Geheimtextzeichen Zeichen verwendet, die sich gut in Stein ritzen lassen. Man kann sie sehr einfach merken:



Bei den Buchstaben j..r bzw. w..z wird einfach zusätzlich ein Punkt gesetzt.

2.1.2 Umgeordnete Alphabete

In diesem Fall wird für Klar- und Geheimtext das gleiche Alphabet verwendet. Wir betrachten also eine Abbildung: $P \longrightarrow P$.

Üblicherweise verwendet man für die Zeichen des Klartextalphabets Klein-, für das Geheimtextalphabet Großbuchstaben.

```
abcdefghijklmnopqrstuvwxyz
CFILORUXADGJMPSVYBEHKNQTWZ
```

Verbreitet ist auch die **Zyklenschreibweise** :

(aci)(bfr)(dlj)(eos)(guk)(hxt)(m)(npv)(qyw)(z)

Bei **Porta** findet man die folgende **involutorische** Abbildung, die nur aus Zweierzyklen des damals gebräuchlichen Alphabets Z_{22} besteht:

(an)(bo)(cp)(dq)(er)(fs)(gt)(hv)(ix)(ly)(mz)

Von Porta stammt auch eine der ältesten "Chiffriermaschinen":

[10]

2.1.3 Caesar

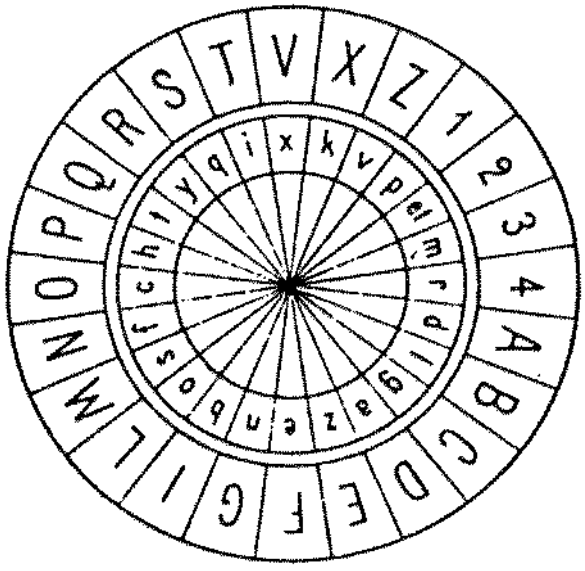
Sueton berichtet, dass *Julius Cäsar* für vertrauliche Mitteilungen eine Chiffre benutzt haben soll:

> *Exstant et epistolae ad Ciceronem, item ad familiares de rebus, in quibus, si qua occultis perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset; quae si qui investigare et persequi velit, quartam elementorum litteram, it est D pro A et perinde reliquas commutet.*

Caesar verschob also das Klartextalphabet um 3 Stellen nach links:

```
abcdefghijklmnopqrstuvwxyz
DEFGHIJKLMNOPQRSTUVWXYZABC
```

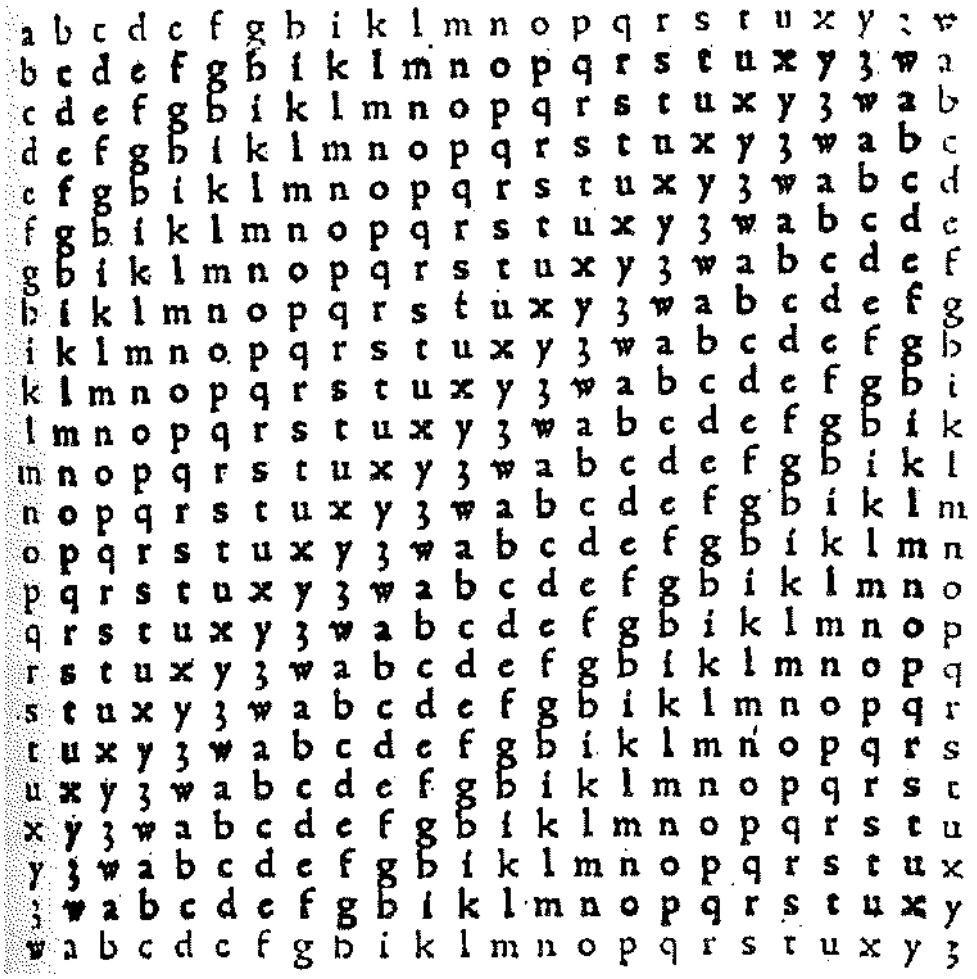
Verschiebechiffren für permutierte Alphabete wurden schon 1466 von *Alberti* mit **Drehscheiben** automatisiert:



[10]

2.1.4 Das Vigenère Tableau

Die **Verschiebechiffren** werden uns auch bei etwas komplizierteren Verfahren noch beschäftigen. Die folgende Tabelle von *Trithemius* von 1508 zeigt die 26 Möglichkeiten:



[1]

2.1.5 Additive und multiplikative Chiffren

Das Vigenère-Tableau aus dem letzten Abschnitt lässt sich leicht folgendermaßen erstellen:

1. Zuordnung Z: $a \rightarrow 1, b \rightarrow 2, ..., y \rightarrow 25, z \rightarrow 0$
2. Berechne $char((Z(c) + s)\%26)$

So erhält man 26 verschiedene Alphabete. Interessanter wird es, wenn man das Zahlenäquivalent eines Buchstabens nicht addiert sondern multipliziert:

1. Zuordnung Z: $a \rightarrow 1, b \rightarrow 2, ..., y \rightarrow 25, z \rightarrow 0$
2. Berechne $char((Z(c) * t)\%26)$

Man sieht, dass nicht für alle t ein brauchbares Geheimtextalphabet erzeugt wird:

```
 01: abcdefghijklmnopqrstuvwxyz
00: ZZZZZZZZZZZZZZZZZZZZZZZZZZ
02: BDFHJLNPRTVXZBDFHJLNPRTVXZ
03: CFILORUXADGJMPSVYBEHKNQTWZ
04: DHLPTXBFJNRVZDHLPTXBFJNRVZ
05: EJOTYDINSXCHMRWBGLQVAFKPUZ
13: MZMZMZMZMZMZMZMZMZMZMZMZMZ
15: ODSHWLAPETIXMBQFUJYNCRGVKZ
17: QHYPGXOFWNEVMDULCTKBSJARIZ
19: SLEXQJCVOHATMFYRKDWPIBUNGZ
21: UPKFAVQLGBWRMHCXSNIDYTOJEZ
23: WTQNKHEBYVSPMJGDAXUROLIFCZ
25: YXWVUTSRQPONMLKJIHGFEDCBAZ
```

Es zeigt sich, daß es nur für die Werte 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25 multiplikative Chiffren gibt. (Warum??)

Additive und multiplikative Chiffren kann man auch kombinieren.

Zunächst wird s addiert, dann mit t multipliziert. Die so entstandene Chiffre $[s, t]$ wird als **Tauschchiffre** bezeichnet.

2.1.6 Umordnung mit Schlüsselworten

Viele Umordnungen lassen sich durch die Verwendung eines Schlüsselwortes erzeugen. Hierzu werden aus dem Schlüsselwort die mehrfach auftretenden Zeichen entfernt, das Schlüsselwort unter das Klartexalphabet geschrieben und die im Schlüsselwort nicht vorkommenden Zeichen hinten angefügt.

Man muss das Schlüsselwort nicht am Anfang positionieren. Auch kann die Anordnung der restlichen Buchstaben verändert werden. Die folgenden Beispiele zeigen einige der möglichen Chiffren:

```
abcdefghijklmnopqrstuvwxyz
SECURITYABDFGHJKLMNOPQVWXZ
abcdefghijklmnopqrstuvwxyz
ZXWVQPONMLKJHGFDBASECURITY
abcdefghijklmnopqrstuvwxyz
ABDSECURITYFGHJKLMNOPQVWXZ
```

Ferner kann man die Sache verkomplizieren, in dem man die Alphabete zeilenweise in eine Matrix schreibt und spaltenweise ausliest:

```
SECURITY
ABDFGHJK
LMNOPQVW
 XZ
abcdefghijklmnopqrstuvwxyz
SALXEBMZCDNUFORGPIHQTJVYKWZ
```

Die Spalten kann man dann noch permutieren, z.B. nach der **Buchstabenordnung** des Schlüsselwortes. Die Buchstabenordnung erhält man durch alphabetische Sortierung der Buchstaben des Schlüsselwortes und anschließende Angabe der Spalten des ursprünglichen Wortes, in denen die sortierten Buchstaben stehen. Das hört sich komplizierter an, als es ist:

Schlüsselwort:SECURITY
sortiert gibt das: CEIRSTUY
S steht an Stelle 5, E an 2, C an 1 u.s.w
ergibt also: 52174368

```
12345678
SECURITY
ABDFGHJK
LMNOPQVW
 XZ
abcdefghijklmnopqrstuvwxyz
RGPEBMZSALXTJVUFOCDNIHQYKW
```

2.2 Bipartite einfache Substitutionen

Die Beispiele dieses Abschnitts sind dem Buch von *F.L.Bauer* [1] entnommen.

Bisher wurde immer ein Klartextzeichen auf ein Geheimtextzeichen abgebildet. Bei den **bipartiten einfachen Substitutionen** wird einem Klartextzeichen ein **Paar** von Geheimtextzeichen zugeordnet:

$$P^{(1)} \longrightarrow C^{(2)}$$

In dieser allgemeinen Form kann es auch vorkommen, dass einigen Klartextzeichen **mehr als ein** Codezeichenpaar zugeordnet wird. Auf diese Weise lassen sich die Häufigkeiten verschleiern.

Beispiel: $Z_{25} \longrightarrow Z_5 \times Z_5$

	1	2	3	4	5
1	a	b	c	d	e
2	f	g	h	i	k
3	l	m	n	o	p
4	q	r	s	t	u
5	v	w	x	y	z

Diese Chiffre fand besonders im Zarenreich Verwendung und wird auch von A. Solschenizyn im "Archipel GULAG" erwähnt.

Giovanni Battista Argenti benutzte bereits 1580 das folgende Schema:

	0	1	2	3	4	5	6	7	8	9
1	p	i	e	t	r	o	a	b	c	d
2	f	g	h	l	m	n	q	s	u	z

Diese Chiffre benutzt auch zum ersten Mal ein **Schlüsselwort** .

Mit einem ähnlichen Schema lassen sich leicht **Homophone** einführen:

			1	2	3	4	5	6	7	8	9
9	6	3	a	b	c	d	e	f	g	h	i
8	5	2	j	k	l	m	n	o	p	q	r
7	4	1	s	t	u	v	w	x	y	z	

Besser ist es, auf die Häufigkeitsverteilung zu achten. Für englische Texte kann man das folgende Schema verwenden:

						1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	e	a	t	o	n	i	r	s	h
2	3					b	c	d	f	g	j	k	l	m
1						p	q	u	v	w	x	y	z	

2.3 Spreizen des Alphabets

Bereits *Matteo Argenti* kam 1590 auf die Idee eine Mischung von Einzelbuchstaben und Buchstabenpaaren zu verwenden:

$$Z_{24}^{(1)} \longrightarrow Z_{10}^{(1)} \bigcup Z_{10}^{(2)}$$

a	b	c	d	e	f	g	h	i	l	m	n	o
1	86	02	20	62 82	22	06	30	3	24	26	84	9

p	q	r	s	t	v	z	et	con	non	che	∞
66	68	28	42	80 40	04	88	08	64	00	44	5

Wichtig ist hier, daß die **Fano-Bedingung** erfüllt ist: keine Chiffre darf Anfang einer anderen Chiffre sein.
Aus neuerer Zeit, nämlich vom Funker des Spions *Dr. Richard Sorge* stammt die folgende Chiffre:

	0	1	2	3	4	5	6	7	8	9
	s	i	o	e	r	a	t	n		
8	c	x	u	d	j	p	z	b	k	q
9	.	w	f	l	/	g	m	y	h	v

Diese Chiffre wurde folgendermaßen konstruiert:

1. Schlüsselwort: SUBWAY
2. *a sin to err* enthält die häufigsten 8 Buchstaben der englischen Sprache - und ist auch sonst ganz passend
3. Mit dem Schlüsselwort beginnend wird ein Rechteck mit den Buchstaben des Alphabets gefüllt.
4. Dann werden von links her spaltenweise die Buchstaben des Merksatzes mit den Buchstaben aus $\{0..7\}$ belegt, danach der Rest mit 80..99

s	u	b	w	a	y
0	82	87	91	5	97
c	d	e	f	g	h
80	83	3	92	95	98
i	j	k	l	m	n
1	84	88	93	96	7
o	p	q	r	t	v
2	85	89	4	6	99
x	z	.	/		
81	86	90	94		

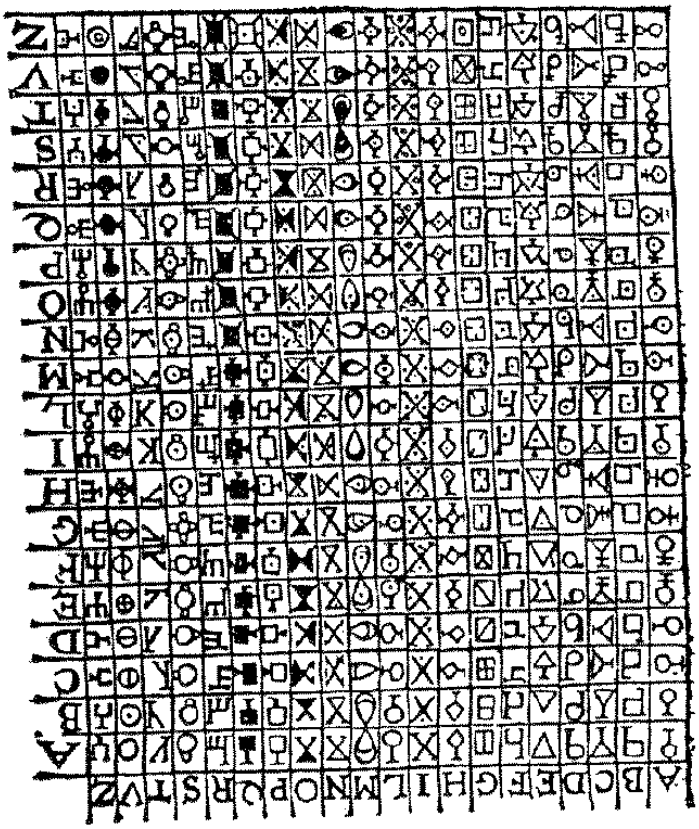
2.4 Polygraphische Substitution

Bei der **polygraphischen Substitution** sind alle Chiffrierschritte von der Form $P^{(n)} \longrightarrow C^{(m)}, m > 1$.

(Auch in diesem Abschnitt stammen die Beispiele aus dem Buch von *F.L.Bauer* [1]).

2.4.1 Bigramm-Substitutionen

Bei [10] findet man ein Bild der ältesten bekannten polygraphischen Chiffrierung: *Porta* 1563



[10]

Die folgende Substitution ist involutorisch, d.h $af \rightarrow EL, el \rightarrow AF$

	a	b	c	d	e	f	g	...
a	XZ	KJ	YJ	HP	PL	EL	VB	...
b	LP	QT	HE	RS	UR	CR	ZH	...
c	DX	MN	AO	NH	SF	GI	WL	...
d	KM	YZ	RY	FP	TR	CT	XE	...
e	QU	HP	QG	JQ	YQ	OB	SA	...
...								

Auch hier ist die Verwendung von Schlüsselwörtern möglich - dabei geht aber der involutorische Charakter verloren:

	a	m	e	r	i	c	b	...
e	XZ	KJ	YJ	HP	PL	EL	VB	...
q	LP	QT	HE	RS	UR	CR	ZH	...
u	DX	MN	AO	NH	SF	GI	WL	...
a	KM	YZ	RY	FP	TR	CT	XE	...
l	QU	HP	QG	JQ	YQ	OB	SA	...
...								

Wichtig ist es, die Häufigkeiten zu verstecken. Idealerweise kommt in jeder Zeile und Spalte jeder Buchstabe genau einmal als erster und einmal als zweiter vor.

2.4.2 Playfair

Die Playfair-Chiffre wurde 1854 von *Charles Wheatstone* erfunden und von seinem Freund *Lyon Playfair, Baron of St. Andrews* verbreitet.

Zunächst wird ein aus einem Schlüsselwort erzeugtes permutiertes Alphabet in eine 5×5-Matrix geschrieben:

S	E	C	U	R
I	T	Y	A	B
D	F	G	H	K
L	M	N	O	P
Q	V	W	X	Z

Diese Matrix wird als Torus betrachtet.

1. Steht das zu verschlüsselnde Buchstabenpaar in der gleichen Zeile, so wird als Chiffre jeweils der rechte Nachbar gewählt: $mp \rightarrow NL$
2. Steht das zu verschlüsselnde Buchstabenpaar in der gleichen Spalte, so wird als Chiffre jeweils der untere Nachbar gewählt: $gw \rightarrow NC$
3. Stehen die beiden Buchstaben in verschiedenen Zeilen/Spalten, so wählt man statt des ersten den Buchstaben in der gleichen Zeile aber in der Spalte des zweiten Buchstabens, und statt des zweiten den Buchstaben in der gleichen Zeile, aber in der Spalte des ersten: $TG \rightarrow YF$

2.5 Transposition

Transpositionsverfahren lassen die Buchstabenmenge des Klartextes unverändert; lediglich die Buchstabenpositionen werden vertauscht.

Eine nichtkryptographische Transposition stellen **Schüttelreime** dar [1]:

reine Sache - seine Rache
schwarzen Wein - Warzenschwein
dear old queen - queer old dean
wirken bald - Birkenwald

2.5.1 Würfel

Nach Wahl einer festen Zeilenlänge wird der Klartext zeilenweise in eine Matrix geschrieben und spaltenweise ausgelesen:

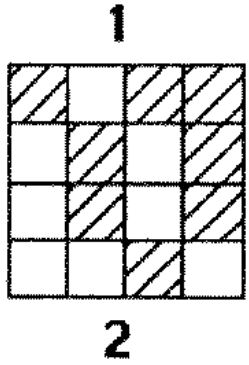
```
ESTRAF
SICHSO
DASSSI
ESICHT
RAFENX
ESDER SIASA TCSIF RHSCE ASSHN FOITX
```

Varianten sind **Diagonalwürfel** ,
Schlangenwürfel oder **Rösselsprungwürfel** .

1	14	9	20	3
24	19	2	15	10
1	8	25	4	21
18	23	6	11	16
7	12	17	22	5

Die Nachricht wird in der Reihenfolge der Zahlen auf den Schachfeldern in die Matrix geschrieben und dann z.B. zeilenweise wieder ausgelesen.

Zweizähliges Drehraster



o e u r
r b t r
o o m t
h a h s

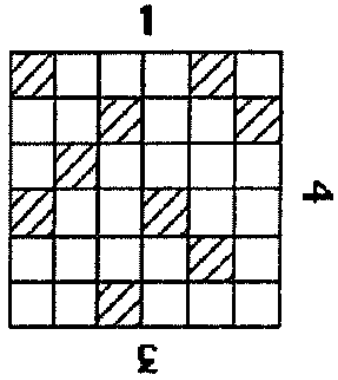
o u r b r o t h
e r t o m h a s

[1]

Bei diesem zweizähligen Drehraster werden die Buchstaben der Nachricht zunächst in die schraffierten Positionen geschrieben, also in die Felder 1, 3, 4, 6, 8, 10, 12, 15. Dann wird das Raster um 180 Grad gedreht, jetzt sind die Positionen 2, 5, 7, 9, 11, 13, 14 schraffiert, die dann mit den restlichen Buchstaben der Nachricht belegt werden.

Bei einem vierzähligen Raster erfolgt die Drehung in Schritten von 90 Grad.

Vierzähliges Drehraster



o p r u u t
i s r o l b
m r t e h g
o s a t o j
t t o t h h
h h e n j e

o u r b r o t h e
r t o m h a t h j
u s t g o t t h e
p i l e s j o h n

[1]

2.5.2 Spaltentranspositionen

Für Spaltentranspositionen wird ein Schlüssel verwendet, der angibt, in welcher Reihenfolge die Spalten einer Matrix ausgelesen werden, in die eine Nachricht zeilenweise geschrieben wurde.

Es traf sich so dass sie sich trafen

4	3	2	1	6	5
E	*S*	*T*	*R*	*A*	*F*
S	*I*	*C*	*H*	*S*	*O*
D	*A*	*S*	*S*	*S*	*I*
E	*S*	*I*	*C*	*H*	*T*
R	*A*	*F*	*E*	*N*	*X*

RHSCE TCSIF SIASA ESDER FOITX ASSHN

Bei der **Blocktransposition** werden die Spalten permutiert, aber dann zeilenweise ausgelesen:

4	3	2	1	6	5
E	*S*	*T*	*R*	*A*	*F*
S	*I*	*C*	*H*	*S*	*O*
D	*A*	*S*	*S*	*S*	*I*
E	*S*	*I*	*C*	*H*	*T*
R	*A*	*F*	*E*	*N*	*X*

$\longrightarrow$

1	2	3	4	5	6
R	*T*	*S*	*E*	*F*	*A*
H	*C*	*I*	*S*	*O*	*S*
S	*S*	*A*	*D*	*I*	*S*
C	*I*	*S*	*E*	*T*	*H*
E	*F*	*A*	*R*	*X*	*N*

RTSEF AHCIS OSSSA DISCI SETHE FARXN

Bei der **gemischten Zeilen-Spaltentransposition** wird die Nachricht in eine Matrix geschrieben, die **Zeilen** permutiert und die **Spalten** mit einer anderen Permutation wieder ausgelesen:

	4	3	2	1	6	5
3	*E*	*S*	*T*	*R*	*A*	*F*
2	*S*	*I*	*C*	*H*	*S*	*O*
5	*D*	*A*	*S*	*S*	*S*	*I*
1	*E*	*S*	*I*	*C*	*H*	*T*
4	*R*	*A*	*F*	*E*	*N*	*X*

$\longrightarrow$

	4	3	2	1	6	5
1	*E*	*S*	*I*	*C*	*H*	*T*
2	*S*	*I*	*C*	*H*	*S*	*O*
3	*E*	*S*	*T*	*R*	*A*	*F*
4	*R*	*A*	*F*	*E*	*N*	*X*
5	*D*	*A*	*S*	*S*	*S*	*I*

$\longrightarrow$

	1	2	3	4	5	6
1	*C*	*I*	*S*	*E*	*T*	*H*
2	*H*	*C*	*I*	*S*	*O*	*S*
3	*R*	*T*	*S*	*E*	*F*	*A*
4	*E*	*F*	*A*	*R*	*X*	*N*
5	*S*	*S*	*A*	*D*	*I*	*S*

CISET HHCIS OSRTS EFAEF ARXNS SADIS

2.6 Kryptoanalyse

In diesem Abschnitt befassen wir uns mit den Entschlüsseln von Geheimtexten. Zunächst einmal werden wir im Eilschritt ein Verfahren zur Analyse von monoalphabetischen Substitutionen kennenlernen.

In einfachen Fällen kann man auch mit der **Exhaustionsmethode** einfach alle Möglichkeiten ausprobieren. Das geht aber nur, wenn die Komplexität der Verfahrensklasse nicht zu groß ist.

Abschließend betrachten wir dann die **Häufigkeitsanalyse** etwas genauer.

2.6.1 Entziffern im Schnelldurchgang

Die folgende Tabelle gibt einen Anhalt, wieviele verschiedene Geheimtextalphabete, wie eben beschrieben, erzeugt werden können:

Anzahl	Formel	$N = 26$	$N = 10$
alle Permutationen	$N!$	$4.03 * 10^{26}$	3628800
einfache zykl. Perm.	$(N-1)!$	$1.55 * 10^{25}$	362880
involutorische Perm.	$\approx (N!)^{\frac{1}{2}}$	$7.91 * 10^{12}$	945
Perm. aus sinnvollen Schlüsseln		$10^4 ... 10^6$	

Wenn man weiß, dass ein Text mit einer der 26 **Verschiebechiffren** verschlüsselt wurde, kann man die Möglichkeiten einfach durchprobieren. In der Regel wird man aber **statistische Methoden** anwenden müssen. Hier kommt dem Kryptographen die Tatsache zu Hilfe, dass natürliche Sprache viele Eigenheiten und Regelmäßigkeiten enthält, die sich nur schwer verbergen lassen. Bei den bis jetzt betrachteten monoalphabetischen Chiffren ist das Vorgehen sogar relativ einfach.

Schritt 1: Aufstellen einer Häufigkeitstabelle

Buchstabe	**Häufigkeit in %**	**Buchstabe**	**Häufigkeit in %**
a	6.51	n	9.78
b	1.89	o	2.51
c	3.06	p	0.79
d	5.08	q	0.02
e	17.4	r	7.00
f	1.66	s	7.27
g	3.01	t	6.15
h	4.76	u	4.35
i	7.55	v	0.67
j	0.27	w	1.89
k	1.21	x	0.03
l	3.44	y	0.04
m	2.53	z	1.13

Häufigkeiten von Buchstabengruppen

Buchstabengruppe	**Häufigkeit in %**
e, n	27.18
i, s, r, a, t	34.48
d, h, u, l, c, g, m, o, b, w, f, k, z	36.52
p, v, j, y, x, q	1.82

Mit diesen Informationen lassen sich schon die Geheimtextbuchstaben für die häufigsten Buchstaben **e, n, i, s, r, a, t** bestimmen.

Schritt 2: Betrachtung der Bigramme

Paar	**Häufigkeit in %**	**Paar**	**Häufigkeit in %**
en	3.88	nd	1.99
er	3.75	ei	1.88
ch	2.75	ie	1.79
te	2.26	in	1.67
de	2.00	es	1.52

Die Buchstaben **e** und **n** konnten im letzten Schritt **direkt** bestimmt werden. Außerdem wissen wir, welche **Menge** von Geheimtextbuchstaben den Klartextbuchstaben {**i, s, r, a, t** } entsprechen.

Das Paar für **er** liefert uns das Geheimtextäquivalent zu **r**. Die Paare **ei** und **ie** kommen mit fast gleicher Häufigheit vor, so kann das **i** isoliert werden. **et** kommt mit weniger als 0.5% vor, **te** mit 2.26%; das liefert uns das **t**. Auch **ea** hat eine Häufigkeit von weniger als 0.5%. **c** und **h** kommen zwar häufig zusammen, aber nur ganz selten allein vor.

Auf diese Weise lassen sich die Buchstaben **e, n, i, s, r, a, t, h, c** identifizieren, die mehr als zwei Drittel eines deutschen Textes ausmachen. Setzt man diese Zeichen in den Geheimtext ein, lassen sich die verbleibenden Buchstaben relativ leicht erraten.

Wir werden in Kürze sehen, wie man die Häufigkeiten verschleiern kann - aber auch diese Komplikation kann überwunden werden!

2.6.2 Die Exhaustionsmethode

Diese Methode funktioniert nur, wenn es nicht allzu viele Möglichkeiten gibt: weiß man z. B., dass ein Text mit einer Verschiebechiffre verschlüsselt wurde, kann man einfach alle 26 Möglichkeiten ausprobieren. Eine **Transposition** der Breite 4 läßt auch nur $4 * 3 * 2 * 1$ Möglichkeiten zu, die man relativ leicht überblicken kann.

Wir betrachten hierzu ein Beispiel für eine Verschiebechiffre:

```
AFLZW TWYAF FAFYG VU
BGMAX UXZBG GBGZH WV
CHNBY VYACH HCHAI XW
DIOCZ WZBDI IDIBJ YX
EJPDA XACEJ JEJCK ZY
FKQEB YBDFK KFKDL AZ
GLRFC ZCEGL LGLEM BA
HMSGD ADFHM MHMFN CB
INTHE BEGIN NINGO DC <---
JOUIF CFHJO OJOHP ED
KPVJG DGIKP PKPIQ FE
```

2.6.3 Die Unizitätslänge

Beim Aufbau der verschiedenen möglichen Klartexte stellt man fest, dass die Entscheidung für einen bestimmten Klartext von einer gut abzugrenzenden Stelle an einfach zu fällen ist. Die Anzahl der Zeichen bis zu dieser Stelle ist die **empirische Unizitätslänge** U. Im Beispiel mit der Verschiebechiffre liegt dieser Wert ungefähr bei 4.

Bei allgemeinen monoalphabetischen Substitutionen ($Z = 26!$) ist $U \approx 25$. U hängt nur von der kombinatorischen Komplexität Z der Verfahrensklasse ab und ist proportional zu $ld\ Z$:

$$U \approx \frac{1}{3.5} ld\ Z$$

Beispiel:

```
ZNRVM NXCJ
AOSWN
BPTX
CQUY
DRV
ESWAR
```

2.6.4 Häufigkeit

Bestimmte Charakteristika natürlicher Sprachen kann man auch durch Verschlüsselung nicht verstecken. Es gilt der folgende Satz:

Bei Transpositionen bleiben die Häufigkeiten der Einzelzeichen innerhalb des Textes erhalten.

Mit diesem Satz kann man Chiffrierverfahren ausschließen: wenn die Buchstabenhäufigkeiten nicht denen einer bekannten Sprache entsprechen, so kann man mit einiger Sicherheit eine Transposition als Chiffrierverfahren nicht verwendet worden sein.

Anders gesagt: ist die Häufigkeitsverteilung die einer bekannten Sprache, so liegt **möglicherweise** eine Transposition vor. Es kann sich aber auch um eine Substitution handeln, die nur eine bestimmte Häufigkeit vorspiegelt.

Für alle monoalphabetischen Substitutionen gilt, dass die Partitionen der Einzelzeichen erhalten bleiben.

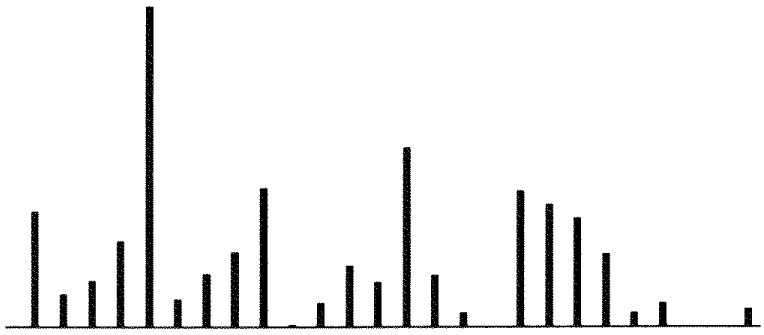
Die Verschlüsselung von FACHHOCHSCHULE wird also aus drei Zeichen für C, vier Zeichen für H, und je einem Zeichen für A, E, F, L, O und U bestehen.

```
FACHHOCHSCHULE
GBDIIPDITDIVMF -->

{I} = {H}, {D} = {C}, {B, F, G, M, P, V} = {A, E, F, L, O, U}
```

Bei der letzten Menge weiß man nur nicht, welches Geheimtextzeichen zu welchem Klartextzeichen gehört.

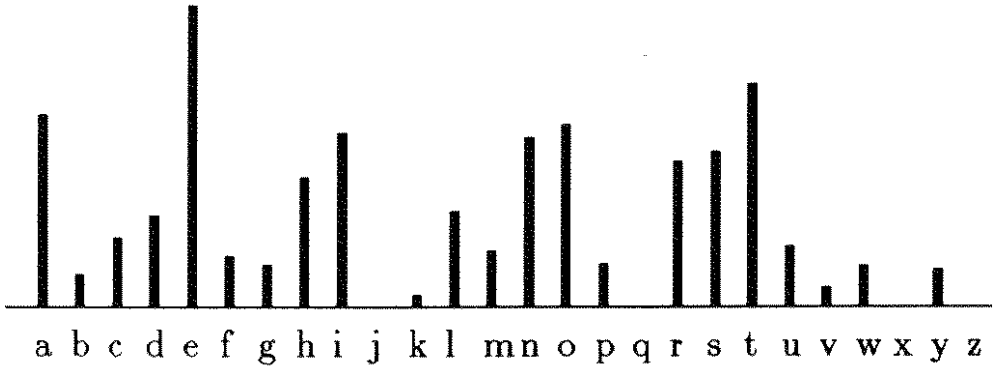
Das folgende Histogramm aus [1] zeigt die typische Häufigkeitsverteilung für einen deutschen Text:



[1]

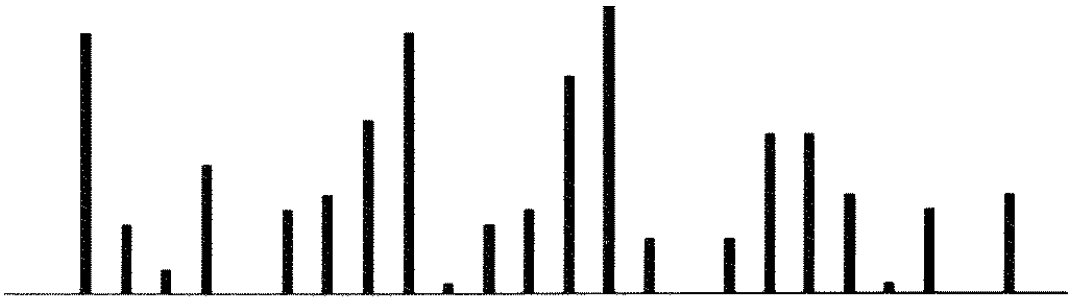
Man beachte die E-Spitze, und den N-Gipfel; weiterhin sind charakteristisch die FGHI-Flanke, die JK- und die OPQ-Senken, sowie der RSTU-Kamm.

Das gleiche Histogramm für typische englische Texte sieht anders aus:



[1]

Bei Cäsar-Additionen ist das Häufigkeitsgebirge lediglich verschoben. Das folgende Beispiel aus [1] zeigt das Histogramm für das Chiffrat eines bekannten Romans von H. Böll:



[1]

```
HVZDU  VFKRQ  GXQNH  ODOVL  FKLQE  RQQDQ
NDPLF  KCZDQ  JPLFK  PHLQH  DQNXQ  IWQLF
KWPLW  GHUDX  WRPDW  LNDEO  DXIHQ  CXODV
VHQGL  HVLFK  LQIXH  QIMDH  KULJH  PXQWH
UZHJV  VHLQK  HUDXV  JHELO  GHWKD  WEDKQ
VWHLJ  WUHSS  HUXQW  HUEDK  QVWHL  JWUHS
SHUDX  IUHLV  HWDVF  KHDEV  WHOOH  QIDKU
NDUWH  DXVGH  UPDQW  HOWDV  FKHQH  KPHQU
HLVHW  DVFKH  DXIQH  KPHQI  DKUND  UWHDE
JHEHQ  CXPCH  LWXQJ  VVWDQ  GDEHQ  GCHLW
XQJHQ  NDXIH  QQDFK  GUDXV  VHQJH  KHQXQ
GHLQW  DALKH  UDQZL  QNHQ
```

Entschlüsselt erhält man:

```
eswar schon dunke lalsi chinb onnan
kamic hzwan gmich meine ankun ftnic
htmit derau tomat ikabl aufen zulas
sendi esich infue nfjae hrige munte
rwegs seinh eraus gebil detha tbahn
steig trepp erunt erbah nstei gtrep
perau freis etasc heabs telle nfahr
karte ausde rmant eltas chene hmenr
eiset asche aufne hmenf ahrka rteab
geben zumze itung sstan daben dzeit
ungen kaufe nnach draus senge henun
deint axihe ranwi nken
```

Diese Methode kann aber auch in die Irre führen: das folgende Histogramm nicht typisch für eine natürliche Sprache:

[1]

Es gehört zu dem folgenden Geheimtext:

```
VQPOU  IKTCB  NHPKO  HUPTI  PXZPV  IPXBC
VODIP  GCSKH  IUZPV  OHGPM  LTEKE  GKOEB
DIBNQ  KPOBN  BOXKU  ICPZT  BOEHK  SMTPG
IKTPX  OBNBO  PGTPE  PNKOU  KOHBO  EIBQQ
ZKOEK  WKEVB  MKUZU  IBUZP  VUIKT  ESBXO
UPIKN  BTKTB  GMZUP  BTVHB  SCPXM
```

Der Grund, warum man getäuscht wird, liegt darin, dass im zugehörigen Klartext z. B. der Buchstabe *e* überhaupt nicht vorkommt. Es handelt sich um ein sogenanntes *Lipogramm* aus dem englischen Roman *Gadsby*, dass nur um ein Zeichen verschoben ist (i = j!):

```
upont hjsba mgojn gtosh owyou howab
uncho fbrjg htyou ngfol ksdjd fjnda
champ jonam anwjt hboys andgj rlsof
hjsow naman ofsod omjnt jngan dhapp
yjndj vjdua ljtyt hatyo uthjs drawn
tohjm asjsa flyto asuga rbowl
```

2.6.5 Häufigkeitsreihenfolge

In [1] findet man zahlreiche Häufigkeitsreihenfolgen für natürliche Sprachen. Einige Beispiele:

Deutsch
enrisdutaghlobmfzkcwvjpqxy (*Romanini 1840*)
enirsahtudlcgmwfbozkpjvqxy (*Kasiski 1863*)
enisratduhglcmwobfzkvpjqxy (*Jensen 1955*)

Englisch:

etaoinshrdlucmfwypvbgkqjxz (*Mergenthaler 1884*)
etaonirshdlucmpfywgbvjkqxz (*Kahn 1967*)
etaoinsrhldcumfpgwybvkxjqz (*Meyer , Matyas 1982*)

Französisch:

esriantouldmcpvfqgxjbhzykw (*Kerckhoffs 1883*)
easintrulodcpmvqfgbhjxyzkw (*de Viaris 1893*)
etainroshdlcfumgpwbyvkqxjz (*Eyraud 1953*)

2.6.6 Cliquenbildung

Bei zu kurzen Texten kann die Analyse der Häufigkeiten **allein** nicht angewendet werden. Man wird also nicht nur *eine Zahl* als Häufigkeit eines Zeichens angeben, sondern ein *Intervall*, in dem die Häufigkeit schwanken kann. Je größer die Textlänge desto geringer wird die Größe dieser Schwankung.

Eine weiteres Verfahren ist die Zusammenfassung von Buchstabengruppen nach ihrer Häufigkeit (*Cliquenbildung*). Für die deutsche Sprache findet man in [1]:

$\{e\}\{nirsatdhu\}\{lgocmbfwkz\}\{pvjyxq\}$

Das kann man bei längeren Texten unterteilen in

$\{e\}\{n\}\{irsat\}\{dhu\}\{lgocm\}\{bfwkz\}\{pv\}\{jyxq\}$

Für englische Texte erhält man:

$\{e\}\{t\}\{oani\}\{rsh\}\{dl\}\{ucwm\}\{fygpb\}$

Bei einer rechnergestützten Analyse sind natürlich gerade die kleinen Cliquen hilfreich, da hier nur wenige Kombinationen getestet werden müssen.

2.6.7 Häufigkeit von n-grammen

Bei längeren Texten kann man auch die Häufigkeitsverteilung von Bi- oder Trigrammen betrachten, die ebenfalls für jede Sprache charakteristisch ist. Für einfach monoalphabetische Substitutionen gilt:

Partitionen der n-gramme innerhalb eines Textes bleiben erhalten.

Die folgenden 18 häufigsten Bigramme im Deutschen, machen 92% aller Bigramme aus:
er en ch de ei nd
te in ie ge es ne
un st re he an be

Die häufigsten Bigramme im Englischen sind:

th he an in er re
on es ti at st en
or nd to nt ed is ar

Eine weitere Angriffsmöglichkeit ergibt sich, wenn die Wortzwischenräume nicht unterdrückt werden. Dann kann man die häufigsten Worte betrachten. Für die wichtigsten Sprachen sind das:

Deutsch:

die der und den am in zu ist dass es

Englisch:

the of and to a in that it is I

Französisch:

de il le et que je la ne on les

In [1] finden sich noch zahlreiche weitere Kennzahlen:

- Wortlängenhäufigkeiten
- Buchstabenhäufigkeit in Abhängigkeit von der Lage im Wort
- Vokalabstände
- Vokal- und Konsonantenwechsel

2.7 Polyalphabetische Chiffrierung

Bei der monoalphabetischen Chiffrierung wird für alle Zeichen des Klartextes nur ein Geheimtextalphabet verwendet, z.B. bei der Cäsar-Chiffrierung ein um drei Plätze verschobenes Alphabet.

Verwendet man nun mehr als einen Chiffrierschritt, ersetzt man z. B. alle Buchstaben auf gerader Position durch ihr Äquivalent aus dem um drei, alle Buchstaben an ungerader Position durch ihr Äquivalent aus dem um sieben Stellen verschobenen Alphabet, so spricht man von einer **polyalphabetischen Chiffrierung**.

Es geht nun darum, auf relativ einfache Weise möglichst viele verschiedene Chiffrierschritte zu erzeugen, d.h. möglichst viele verschiedene Alphabete festzulegen.

2.7.1 Potenzierung

Wir hatten bereits bei der Cäsar-Chiffrierung gesehen, wie man in der Zyklenschreibweise Permutationen des Alphabets kompakt beschreiben kann. Sei A das **Standardalphabet** mit 26 Zeichen. Dann beschreibt die folgende Permutation S das um ein Zeichen verschobene Alphabet:

$\rho = A \longleftrightarrow A : (abcdefghijklmnopqrstuvwxyz)$

ρ^2 ist dann $(acegikmoqsuwy)(bdfhjlnprtvxz)$

und für ρ^3 erhält man: $(adgjmpsvybehknqtwzcfilorux)$

Allgemein:

Sei S eine Permutation des Alphabets A, dann ist

$\{S^i : i \in \mathcal{N}\}$ die Menge der **potenzierten S-Alphabete**.

Hat man weitere Permutationen Q, P, Z von A, so kann man die folgenden Alphabete konstruieren:

$\{QS^iP : i \in \mathcal{N}\}$

$\{S^iP : i \in \mathcal{N}\}$

$\{QS^i : i \in \mathcal{N}\}$

$\{S^iZS^{-i} : i \in \mathcal{N}\}$

2.7.2 Verschobene und rotierte Alphabete

Mit ρ bezeichnen wir den **Zyklus des Standardalphabets**:

$$\rho = (abcdefghijklmnopqrstuvwxyz)$$

Dann ist die **Menge der verschobenen Standardalphabete**:

$\{\rho^i : i \in \mathcal{N}\} = \{\rho^i\rho : i \in \mathcal{N}\}$

Mit einer frei gewählten Permutation P erhält man dann:

$\{\rho^i P : i \in \mathcal{N}\}$, die Menge der **horizontal verschobenen P-Alphabete**,

```
  abcdefghijklmnopqrstuvwxyz
0 SECURITYABDFGHJKLMNOPQVWXZ
1 ECURITYABDFGHJKLMNOPQVWXZS
2 CURITYABDFGHJKLMNOPQVWXZSE
3 URITYABDFGHJKLMNOPQVWXZSEC
4 RITYABDFGHJKLMNOPQVWXZSECU
5 ITYABDFGHJKLMNOPQVWXZSECUR
6 TYABDFGHJKLMNOPQVWXZSECURI
7 YABDFGHJKLMNOPQVWXZSECURIT
8 ABDFGHJKLMNOPQVWXZSECURITY
9 BDFGHJKLMNOPQVWXZSECURITYA
...
```

$\{P\rho^i : i \in \mathcal{N}\}$, die Menge der **vertikal verschobenen P-Alphabete**,

```
  abcdefghijklmnopqrstuvwxyz
0 SECURITYABDFGHJKLMNOPQVWXZ
1 TFDVSJUZBCEGHIKLMNOPQRWXYA
2 UGEWTKVACDFHIJLMNOPQRSXYZB
3 VHFXULWBDEGIJKMNOPQRSTYZAC
4 WIGYVMXCEFHJKLNOPQRSTUZABD
5 XJHZWNYDFGIKLMOPQRSTUVABCE
6 YKIAXOZEGHJLMNPQRSTUVWBCDF
7 ZLJBYPAFHIKMNOQRSTUVWXCDEG
8 AMKCZQBGIJLNOPRSTUVWXYDEFH
9 BNLDARCHJKMOPQSTUVWXYZEFGI
...
```

$\{\rho^i P\rho^{-i} : i \in \mathcal{N}\}$, die Menge der **rotierten P-Alphabete**,

```
  abcdefghijklmnopqrstuvwxyz
0 SECURITYABDFGHJKLMNOPQVWXZsecuritya
1 DBTQHSXZACEFGIJKLMNOPUVWYRdbtqhsxz
2 ASPGRWYZBDEFHIJKLMNOTUVXQCaspgrwy
3 ROFQVXYACDEGHIJMLMNSTUWPBZrofqvx
4 NEPUWXZBCDFGHIJKLMRSTVOAYQnepuw
5 DOTVWYABCEFGHIJKLQRSUNZXPMdotv
6 NSUVXZABDEFGHIJKPQRTMYWOLCnsu
7 RTUWYZACDEFGHIJOPQSLXVNKBMrt
8 STVXYZBCDEFGHINOPRKWUMJALQs
9 SUWXYABCDEFGHMNOQJVTLIZKPR
...
```

Betrachten wir noch ein weiteres, kleines Beispiel:

Sei $P = \begin{pmatrix} a & b & c & d & e \\ D & E & A & B & C \end{pmatrix}$ und $\rho = (ace)(bd)$

Damit ergibt sich für

$$\rho P = \begin{pmatrix} a & b & c & d & e \\ C & D & E & B & A \end{pmatrix} \begin{pmatrix} a & b & c & d & e \\ D & E & A & B & C \end{pmatrix} = \begin{pmatrix} a & b & c & d & e \\ A & B & C & E & D \end{pmatrix}$$

$$P\rho = \begin{pmatrix} a & b & c & d & e \\ D & E & A & B & C \end{pmatrix} \begin{pmatrix} a & b & c & d & e \\ C & D & E & B & A \end{pmatrix} = \begin{pmatrix} a & b & c & d & e \\ B & A & C & D & E \end{pmatrix}$$

$$\rho P \rho^{-1} = \begin{pmatrix} a & b & c & d & e \\ A & B & C & E & D \end{pmatrix} \begin{pmatrix} a & b & c & d & e \\ E & D & A & B & C \end{pmatrix} = \begin{pmatrix} a & b & c & d & e \\ E & D & A & C & B \end{pmatrix}$$

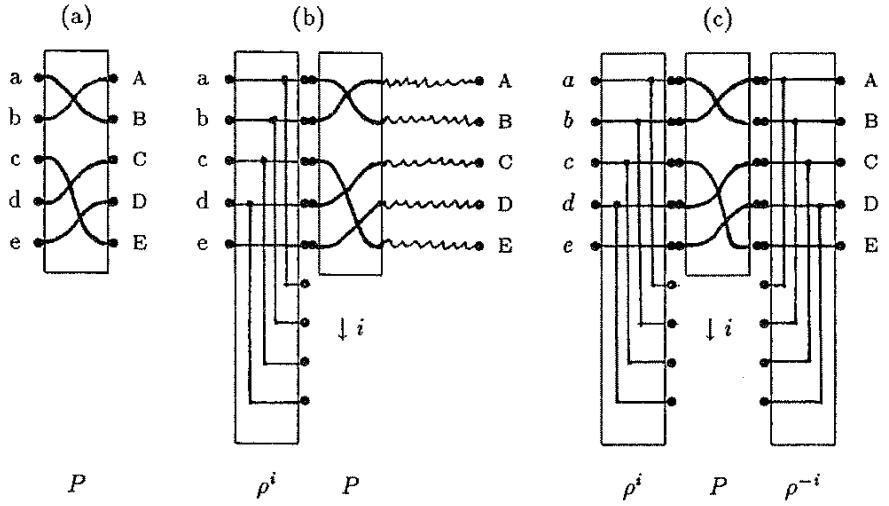
Übungsaufgabe:

Man berechne für die oben angegebenen P und ρ die Mengen der horizontal und vertikal verschobenen, sowie die rotierten Alphabete.

Die rotierten Alphabete haben ihren Namen wegen der Tatsache bekommen, dass sie sich mechanisch durch drehbare Scheiben realisieren lassen. Die Transformation $\rho^i P \rho^{-i}$ läßt sich dann durch drei Scheiben realisieren, bei denen die drehbare mittlere Scheibe die Permutation P realisiert. Die Permutation ρ^i wird durch eine vor diesem Rotor angebrachte Scheibe implementiert, ebenso wie ρ^{-i} durch eine zweite Scheibe hinter dem Rotor.

Die folgenden Skizzen aus [1] zeigen das Prinzip. Die **ENIGMA** verwendete drei bzw. vier drehbare Scheiben.

Feste Permutation, Verschiebung und Rotation



[1]

Input rechts, Output links!

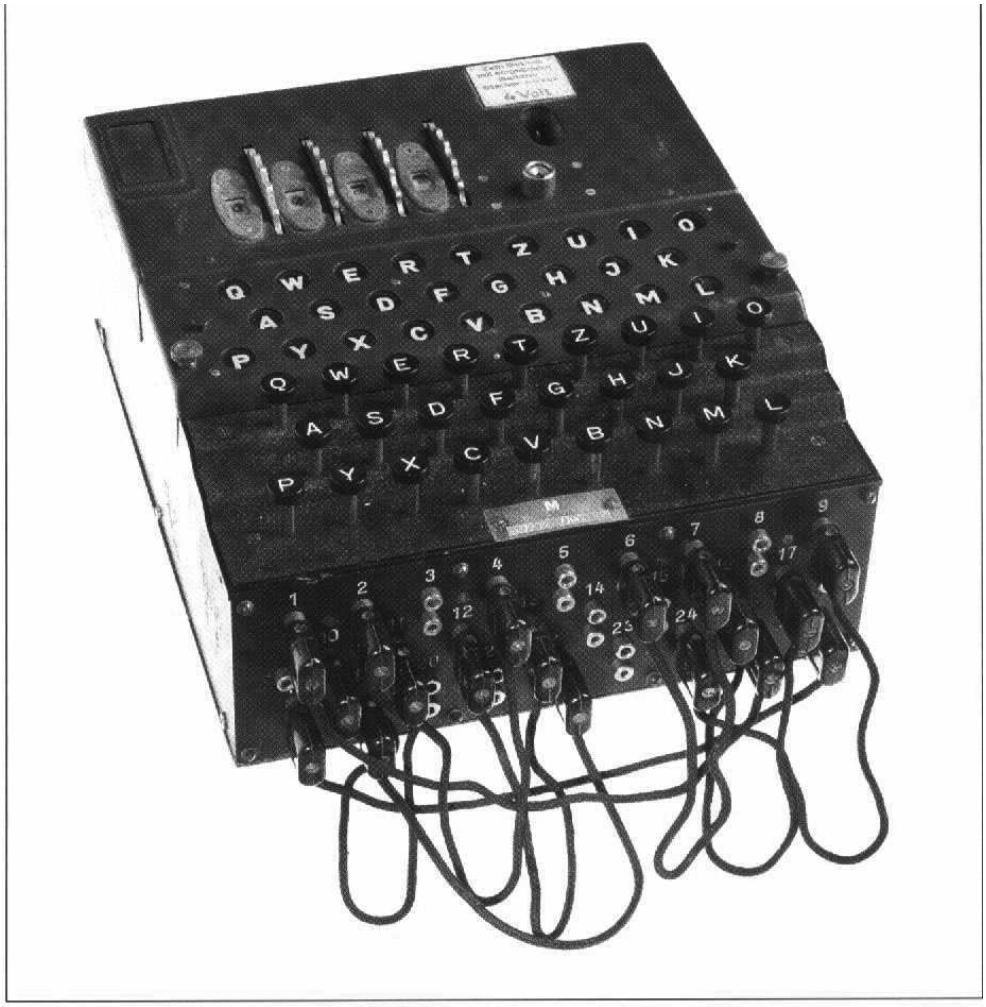
(a) implementiert hier die feste Permutation P.

(b) realisiert $P\rho^i$ durch verschiebbare elektrische Kontakte.

(c) verwendet einen Rotor zur Implementierung von $\rho^{-i} P \rho^i$

2.8 Die Wehrmachtsenigma

Die Beschreibung der Arbeitsweise der ENIGMA-Entschlüssler in Polen und in Bletchley Park folgt eng der Darstellung in [2]. Man kann hier sehr schön sehen, wie sich der (nicht ungefährliche) Einsatz von Geheimagenten und die Kreativität von Mathematikern ergänzen. Zur weiteren Lektüre seien [9] und [7] empfohlen!



[2]

2.8.1 Historie

- symmetrisches Verfahren
- polyalphabetisch, speziell: verschobene, rotierte Alphabete

1920 hergestellt durch die Chiffriermaschinen AG (Arthur Scherbius)

1930 erster Einsatz in der Wehrmacht

1934 Weiterentwicklung (5, 7, 8 Rotoren)

1936 monatl. Wechsel der Rotorreihenfolge und der Steckerverbindungen

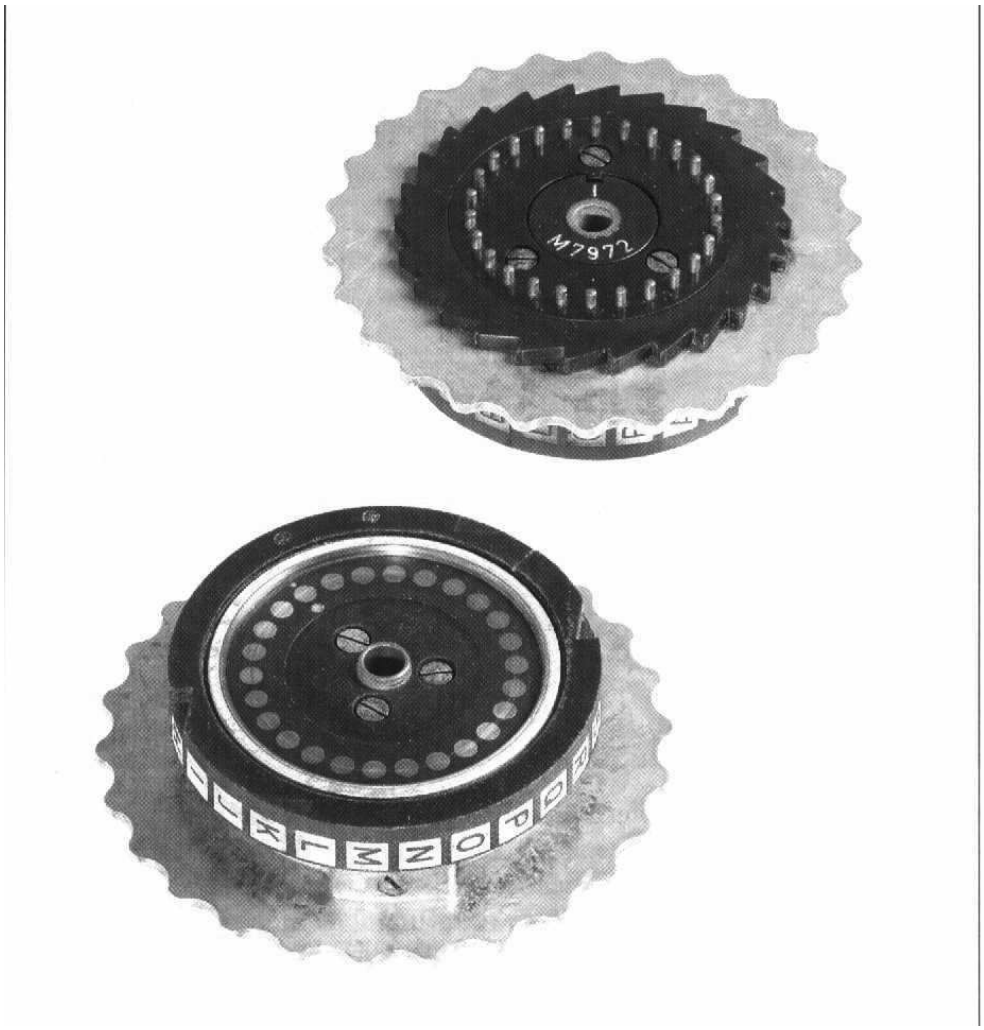
1939 tägl. Wechsel

1940 ENIGMA-Verkehr der Luftwaffe gebrochen

1941 neue Umkehrwalze

1942 Beherrschung durch die Engländer

2.8.2 Die Rotoren



[2]

Rotor No.	Wiring ABCDEFGHIJKLMNOPQRSTUVWXYZ	Notch	Window	Remarks
I	EKMFLGDQVZNTOWYHXUSPAIBRCJ	Y	Q	
II	AJDKSIRUXBLHWTMCQGZNPYFVOE	M	E	
III	BDFHJLCPRTXVZNYEIWGAKMUSQO	D	V	
IV	ESOVPZJAYQUIRHXLNFTGKDCMWB	R	J	
V	VZBRGITYUPSDNHLXAWMJQOFECK	H	Z	
VI	JPGVOUMFYQBENHZRDKASXLICTW	H,U	Z,M	
VII	NZJHGRCXMYSWBOUFAIVLPEKQDT	H,U	Z,M	
VIII	FKQHTLXOCBJSPDZRAMEWNIUYGV	H,U	Z,M	
Beta	LEYJVCNIXWPBQMDRTAKZGFUHOS			M-4 only
Gamma	FSOKANUERHMBTIYCWLQPZXVGJD			M-4 only

Reflector B (Thick, normal):
(AY),(BR),(CU),(DH),(EQ),(FS),(GL),(IP),(JX),(KN),(MO),(TZ),(VW)

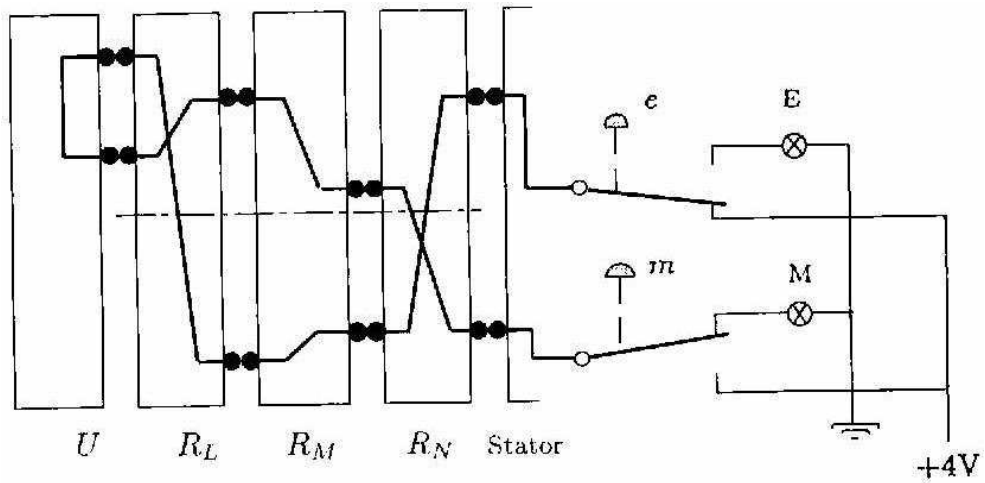
Reflector C (Thick, normal):
(AF),(BV),(CP),(DJ),(EI),(GO),(HY),(KR),(LZ),(MX),(NW),(QT),(SU)

Reflector B (Thin, M-4 only) :
(AE),(BN),(CK),(DQ),(FU),(GY),(HW),(IJ),(LO),(MP),(RX),(SZ),(TV)

Reflector C (Thin, M-4 only) :
(AR),(BD),(CO),(EJ),(FN),(GT),(HK),(IV),(LM),(PW),(QZ),(SX),(UY)

2.8.3 Stromlaufplan

Das folgende Diagramm zeigt den Stromverlauf bei der Eingabe des Buchstabens **e**.



[2]

2.8.4 Die ENIGMA-Gleichungen

Mit drei Rotoren R_L, R_M, R_N und einer echt involutorischen Substitution U entsteht die Familie $\{P_{i_1,i_2,i_3}\}$, mit

$P_{i_1,i_2,i_3} = S_{i_1,i_2,i_3} U S^{-1}_{i_1,i_2,i_3}$ und

$S_{i_1,i_2,i_3} = \rho^{-i_1} R_N \rho^{i_1-i_2} R_M \rho^{i_2-i_3} R_L \rho^{i_3}$

Alle Elemente dieser Familie sind echt involutorische Substitutionen.
1934 wurde zusätzlich ein Steckbrett eingeführt, das eine fest vorgeschaltete Substitution T erlaubte. damit ergibt sich die **ENIGMA-Gleichung**:

$c_i = p_i T S_i U S_i^{-1} T^{-1} = p_i T P_i T^{-1}$

für ein Klartextzeichen p_i in ein Geheimtextzeichen c_i.

Es gilt: $c_i T S_i = p_i T S_i U$

2.8.5 Komplexität und Anzahl der Schlüssel

Schlüssel:

- verwendete Rotoren und deren Reihenfolge
  ab 1938 3 aus 5 **IV II III**
- Ringstellung **A B C**
- Steckerverbindung **(UA) (PF) (RQ) ...**
- Grundstellung **X Y Z**

Komplexität:

- 3 Rotoren aus 5: $\binom{5}{3} = \frac{5!}{(5-3)!} = 60$
- Ringstellung bei drei ausgewählten Rotoren: $26^3 = 17576$
- Grundstellung bei drei ausgewählten Rotoren: $26^3 = 17576$
- Steckbrettbelegung: $\frac{n!}{b_1!*b_2!*...b_s!*1^{b_1}*2^{b_2}*...s^{b_s}}$
  b_s = Häufigkeit der Zyklen der Länge s

Beispiel:

Alphabet mit n = 26 Buchstaben:

10 Paare: $\frac{26!}{6!*10!*1^6*2^{10}} = 1.507382749 * 10^{14}$

13 Paare: $\frac{26!}{13!*2^{13}} = 7.905853581 * 10^{12}$

In der Praxis wurden 4 - 10 Paare benutzt.

Gesamtkomplexität bei 13 Paaren: $2.79392587 * 10^{24}$

2.8.6 Entzifferung der Spruchschlüssel

Bis 1938 wurde als Spruchschlüssel eine frei gewählte Dreiergruppe von Buchstaben verwendet. Diese Dreiergruppe wurde verdoppelt und mit dem **Tagesschlüssel** verschlüsselt dem Text vorangestellt.

```
SSS SSS ==> AUQ AMN
```

Die Tastatur der ENIGMA:

```
Q W E R T Z U I O
 A S D F G H J K
P Y X C V B N M L
```

Der Spion Hans Thilo Schmidt (Asche) hatte von 10/31 bis 10/32 gebrauchsanleitungen, Schlüsselanleitungen, Tagesschlüssel über die Franzosen an Polen geliefert. Der Pole **Marian Rejewski** knackte dann den Spruchschlüssel. Und das ging so:

Chiffrierte Spruchschlüssel

1.	AUQ	AMN	14.	IND	JHU	27.	PVJ	FEG	40.	SJM	SPO
2.	BNH	CHL	15.	JWF	MIC	28.	QGA	LYB	41.	SJM	SPO
3.	BCT	CGJ	16.	JWF	MIC	29.	QGA	LYB	42.	SJM	SPO
4.	CIK	BZT	17.	KHB	XJV	30.	RJL	WPX	43.	SUG	SMF
5.	DDB	VDV	18.	KHB	XJV	31.	RJL	WPX	44.	SUG	SMF
6.	EJP	IPS	19.	LDR	HDE	32.	RJL	WPX	45.	TMN	EBY
7.	FBR	KLE	20.	LDR	HDE	33.	RJL	WPX	46.	TMN	EBY
8.	GPB	ZSV	21.	MAW	UXP	34.	RFC	WQQ	47.	TAA	EXB
9.	HNO	THD	22.	MAW	UXP	35.	SYX	SCW	48.	USE	NWH
10.	HNO	THD	23.	NXD	QTU	36.	SYX	SCW	49.	VII	PZK
11.	HXV	TTI	24.	NXD	QTU	37.	SYX	SCW	50.	VII	PZK
12.	IKG	JKF	25.	NLU	QFZ	38.	SYX	SCW	51.	VQZ	PVR
13.	IKG	JKF	26.	OBU	DLZ	39.	SYX	SCW	52.	VQZ	PVR

53.	WTM	RAO	
54.	WTM	RAO	
55.	WTM	RAO	
56.	WKI	RKK	
57.	XRS	GNM	
58.	XRS	GNM	
59.	XOI	GUK	
60.	XYW	GCP	
61.	YPC	OSQ	
62.	YPC	OSQ	
63.	ZZY	YRA	
64.	ZEF	YOC	
65.	ZSJ	YWG	

[2]

$P_1, P_2, ..., P_6$ bezeichnen die Permutationen der ersten 6 Zeichen.

$aP_i = x \; aP_{i+3} = y \Rightarrow xP_i^{-1}P_{i+3} = y$

Alle P_i sind involutorisch : $xP_iP_{i+3} = y$

Sobald jedes Zeichen an der 1/2/3 Stelle einmal aufgetreten ist, kann man diese Produkte ausrechnen. Das ist nach etwa 50 - 100 Sprüchen der Fall.

Also erhält man z.B. aus

AUQ AMN $AP_1P_4 = A$, $SP_1P_4 = S$

Produkte der Permutationen

So erhält man nach und nach:
$P_1P_4 = $ (a)(s)(bc)(rw)(dvpfkxgzyo)(eijmunqlht)
$P_2P_5 = $ (axt)(blfqveoum)(cgy)(d)(hjpswizrn)(k)
$P_3P_6 = $ (abviktjgfcqny)(duzrehlxwpsmo)
Wichtig sind nun die Einerzyklen. Wenn $aP_1P_4 = a$ so muss es ein Zeichen x in P_1 und P_4 geben, so dass $aP_1 = x$ und $xP_4 = a$.
Es gilt nun, dass in diesen Produkten von Permutationen bei geradem Alphabetumfang die Zyklen in gleich langen Paaren auftreten müssen. Da die P_i alle involutorisch sind, bestehen sie nur aus Zweierzyklen. Wenn man nun die Zyklen der Produkte einander geeignet gegenüberstellt und einen dabei in umgekehrter Reihenfolge aufschreibt, kann man diese Zweierzyklen direkt ablesen. Dazu ein Beispiel:

```
P: (ab) (cd) (ef)
Q: (bc) (de) (fa)

PQ: (ace) (bfd)

a c e
|/|/|
d f b
f b d
b d f
```

Hat man also eine Entsprechung, kann man alle Paare zu diesem Zyklenpaar ablesen.

Berechnung P3 und P6

Nun braucht man eine Intuition. Im Beispiel haben wir gesehen, dass die Folge **SYX SCW** sehr häufig vorkommt. Wir raten nun, dass der Schlüssel aaa dazugehört. Dann erhält man:

$(as) \in P_1, P_4, (ay) \in P_2, (ac) \in P_5$ und $(aw) \in P_6$

```
--> (a b v i k t j g f c q n y)
<-- (x l h e r z u d o m s p w)
```

$P_3 = (ax)(bl)(vh)(ie)(kr)(tz)(ju)(gd)(fo)(cm)(qs)(np)(yw)$
$P_6 = (xb)(lv)(hi)(ek)(rt)(zj)(ug)(df)(oc)(mq)(sn)(py)(wa)$

Berechnung P2 und P5

P_3 enthält (qs). Daher hat der zu AUQ AMN gehörige Schlüssel die Gestalt $**s$; da (as) in P_1 enthalten ist, sogar $s*s$. Nun raten wir, dass der Schlüssel sss ist, und damit muss in P_2 neben (ay) auch (su) enthalten sein.

```
--> (a x t) (b l f q v e o u m) (d)
<-- (y g c) (j h n r z i w s p) (k)
```

$P_2 = (ay)(xg)(tc)(bj)(ln)(fh)(qr)(vz)(ei)(ow)(us)(mp)(dk)$
$P_5 = (yx)(gt)(ca)(jl)(nf)(hq)(rv)(ze)(io)(wu)(sm)(pb)(kd)$

Berechnung P1 und P4

zu RJL WPX muss ein Schlüssel der Form $*bb$ gehören. Paart man in P_1 r mit b, so ergibt sich der wahrscheinlichere Schlüssel bbb

```
--> (a) (b c) (d v p f k x g z y o)
<-- (s) (r w) (i e t h l q n u m j)
```

$P_1 = (as)(br)(cw)(di)(ve)(pt)(fh)(kl)(xq)(gn)(zu)(ym)(oj)$
$P_4 = (sa)(rc)(wb)(iv)(ep)(tf)(hk)(lx)(qg)(nz)(uy)(mo)(jd)$

P1 P2 P3 geordnet

$P_1 = (as)(br)(cw)(di)(ev)(fh)(gn)(jo)(kl)(my)(pt)(qx)(uz)$
$P_2 = (ay)(bj)(ct)(dk)(ei)(fh)(gx)(ln)(mp)(ow)(qr)(su)(vz)$
$P_3 = (ax)(bl)(cm)(dg)(ei)(fo)(hv)(ju)(kr)(np)(qs)(tz)(wy)$

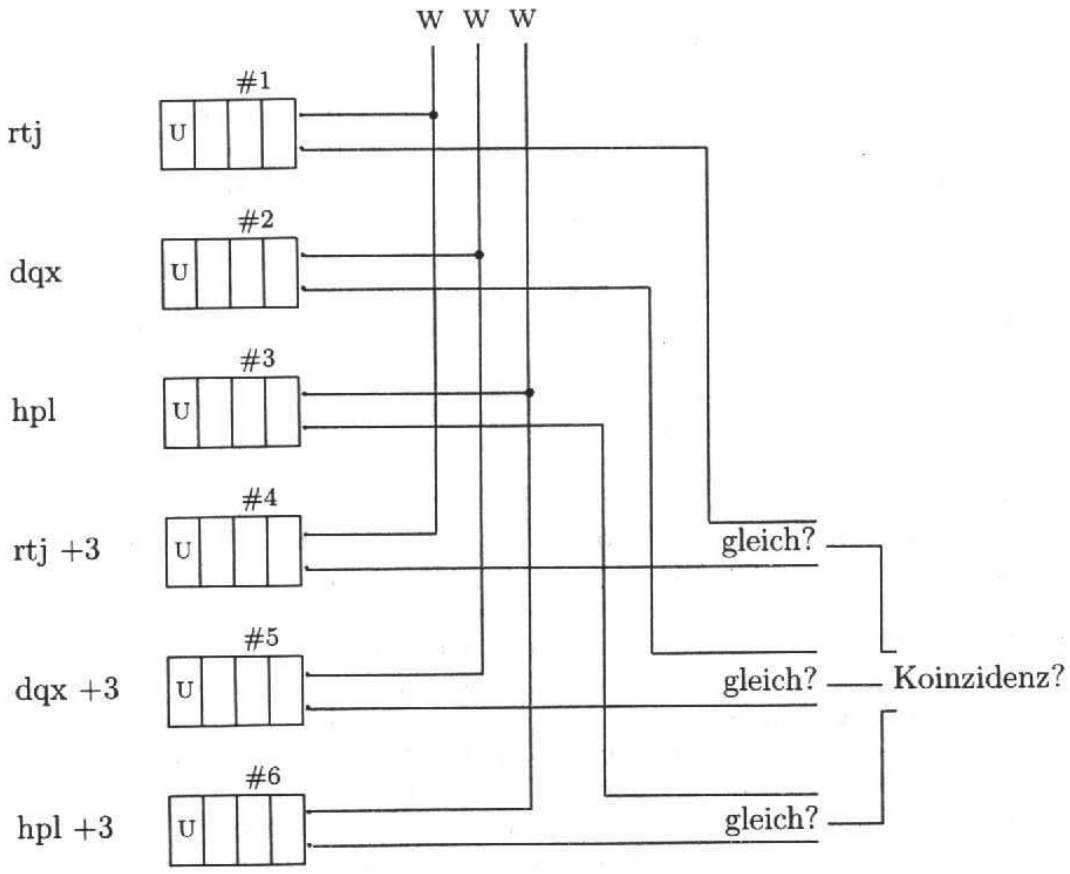
Die entzifferten Schlüssel

AUQ AMN : sss	IKG JKF : ddd	QGA LYB : xxx	VQZ PVR : ert
BNH CHL : rfv	IND JHU : dfg	RJL WPX : bbb	WTM RAO : ccc
BCT CGJ : rtz	JWF MIC : ooo	RFC WQQ : bnm	WKI RKK : cde
CIK BZT : wer	KHB XJV : lll	SYX SCW : aaa	XRS GNM : qqq
DDB VDV : ikl	LDR HDE : kkk	SJM SPO : abc	XOI GUK : qwe
EJP IPS : vbn	MAW UXP : yyy	SUG SMF : asd	XYW GCP : qay
FBR KLE : hjk	NXD QTU : ggg	TMN EBY : ppp	YPC OSQ : mmm
GPB ZSV : nml	NLU QFZ : ghj	TAA EXB : pyx	ZZY YRA : uvw
HNO THD : fff	OBU DLZ : jjj	USE NWH : zui	ZEF YOC : uio
HXV TTI : fgh	PVJ FEG : tzu	VII PZK : eee	ZSJ YWG : uuu

[2]

2.8.7 Die Bomben

Später benutzte man nicht mehr die selbe Grundstellung für den ganzen Tag, sondern die Funker stellten für jeden Spruch eine neue Grundstellung **unverschlüsselt** dem Spruch voran, danach kam wieder der mit dieser individuellen Grundstellung verschlüsselte verdoppelte Spruchschlüssel.
Da die Rotorenlage und die interne Ringstellung eines jeden Rotors unbekannt war, hatte der Gegner immer noch einen Suchraum von $\binom{5}{3} * 26^3 = 1.054.560$ Elementen zu durchforschen.
Rejewski ließ nun eine Maschine bauen, die die Ringstellung durch systematisches Probieren ermittelte.



[2]

Dazu benötigte er drei Funksprüche, in denen jeweils an den Positionen 1/4, 2/5 und 3/6 die Zeichen übereinstimmten:

```
rtj | WAH WIK
dqx | DWJ MWR
hpl | RAW KTW
```

Man startete nun die Maschine mit den drei Voreinstellungen `rtj, dqx, hpl`, wiederholte ständig die Eingabe des Testbuchstabens `W` und wartete, bis sich in jedem der drei Paare jeweils ein gleicher Buchstabe einstellte, also das Muster

```
W - - W - -
- W - - W -
- - W - - W
```

auftrat.
Das Verfahren funktionierte nur, wenn `W` nicht "gesteckert" war. Nach Ermittlung der Ringstellung konnte man dann auch die Belegung des Steckbrettes knacken.
Dieses Verfahren wurde später durch Alan Turing verbessert.

2.9 Die Vigenère-Chiffre

Die Vigenère-Verschlüsselung beruht auf einer Idee des französischen Diplomaten *Vigenère* aus dem Jahr 1586. Sie verwendet die 26 verschobenen Standardalphabete im Wechsel. Es handelt sich also um ein **polyalphabetisches Verfahren**.

Man benötigt ein Schlüsselwort, das wiederholt über den Klartext geschrieben wird, und die *tabula recta* des *Trithemius* , die wir ja bereits kennengelernt haben.

Zur Erinnerung:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
BCDEFGHIJKLMNOPQRSTUVWXYZA
CDEFGHIJKLMNOPQRSTUVWXYZAB
DEFGHIJKLMNOPQRSTUVWXYZABC
EFGHIJKLMNOPQRSTUVWXYZABCD
FGHIJKLMNOPQRSTUVWXYZABCDE
GHIJKLMNOPQRSTUVWXYZABCDEF
HIJKLMNOPQRSTUVWXYZABCDEFG
IJKLMNOPQRSTUVWXYZABCDEFGH
JKLMNOPQRSTUVWXYZABCDEFGHI
KLMNOPQRSTUVWXYZABCDEFGHIJ
LMNOPQRSTUVWXYZABCDEFGHIJK
MNOPQRSTUVWXYZABCDEFGHIJKL
NOPQRSTUVWXYZABCDEFGHIJKLM
OPQRSTUVWXYZABCDEFGHIJKLMN
PQRSTUVWXYZABCDEFGHIJKLMNO
QRSTUVWXYZABCDEFGHIJKLMNOP
RSTUVWXYZABCDEFGHIJKLMNOPQ
STUVWXYZABCDEFGHIJKLMNOPQR
TUVWXYZABCDEFGHIJKLMNOPQRS
UVWXYZABCDEFGHIJKLMNOPQRST
VWXYZABCDEFGHIJKLMNOPQRSTU
WXYZABCDEFGHIJKLMNOPQRSTUV
XYZABCDEFGHIJKLMNOPQRSTUVW
YZABCDEFGHIJKLMNOPQRSTUVWX
ZABCDEFGHIJKLMNOPQRSTUVWXY
```

Der über einem Klartextbuchstaben stehende Schlüsselwortbuchstabe selektiert nun das zu verwendende Alphabet:

```
SECURITYSECURITYSECURITYSECURITY
DieserKlartextwirdverschluesselt

vmgmvzdjsvvyobpgjhxyiavfdygmjmer
```

Dieser Vorgang lässt sich leicht programmieren. In **C** sieht das etwa so aus:

```
void Encode(char bufferin[], char bufferout[],
            const int wasTun, char key[], int &keypos)
{
 int i, o;
 int tmp;

//...

   if (wasTun == vigenere)
    tmp = (bufferin[i] + key[keypos] - 2 * 'a') % ALPHA;
   else
    tmp = (bufferin[i] - key[keypos]) % ALPHA;
   keypos = (keypos + 1) % keylength;
   while (tmp < 0) tmp += ALPHA;
   bufferout[o++] = char(tmp + 'a');

//...
};
```

2.10 Der Kasiski-Test

Eine Vigenère-Verschlüsselung verdeckt zunächst einmal die charakteristischen Buchstabenhäufigkeiten, die bei der Analyse monoalphabetischer Substitutionen so hilfreich waren. Aber ganz so einfach lassen sich die Eigenheiten einer natürlichen Sprache nicht ausrotten.

Die Idee des preußischen Infanteriemajors *Friedrich Wilhelm Kasiski* war nun, zunächst einmal die Länge des Schlüsselworts zu bestimmen. Kasiski machte folgende Beobachtung:

Beträgt im Klartext der Abstand zweier gleicher Zeichenfolgen ein ganzzahliges Vielfaches der Schlüsselwortlänge, so werden beide Zeichenfolgen mit den gleichen Schlüsselwortbuchstaben verschlüsselt und so in den gleichen Geheimtext übersetzt.

```
SECURITYSECURITYSECURITYSECURITY

.Spion......Spion........Spion..

pWRCFVqvpbzrJXBMFbzrofqvpWRCFVqv
```

Man kann auch umgekehrt argumentieren: findet man in einem Geheimtext zwei gleiche Zeichenfolgen, so **vermutet** man, dass ihr Abstand ein Vielfaches der Schlüsselwortlänge beträgt.

Betrachten wir den folgenden Geheimtext:

```
vmgmv zmcpx ycils sexgm kdxpo ipxvb
mcpxg qzmxp zinzv vblvm gmvul azagl
vvzck gjuvn mxmpg mvvnl vxgrk mqxmz
gljbx fwr

VMGMv zMCPX ycils sexGM kdxpo ipxvb
MCPXg qzmxp zinzv vblVM GMvul azagl
vvzck gjuvn mXMpg mvvnl vxgrk mqXMz
gljbx fwr

SECURITY SECURITY SECURITY
DIESerTE XTwirdzu mtEStver
wendetTE XTewieer helfenin
DIESemsc hwerenge schaeftZ
Ulesenun dtextexZ Uversteh
en
```

Abstände:

Text	Abstand	Zerlegung
VMGM	48	$2 * 2 * 2 * 2 * 3$
MCPX	24	$2 * 2 * 2 * 3$
XM	16	$2 * 2 * 2 * 2$
GM	32	$2 * 2 * 2 * 2 * 2$
mx	34	$2 * 17$

Es kann natürlich auch vorkommen, dass Geheimtexte übereinstimmen, ohne zu den gleichen Klartexten zu gehören. Dieser Ausreißer muss man erkennen. Im Beispiel ist das die Zeichenfolge `mx`.

Die Länge des Schlüsselwortes ist nun bis auf Vielfache bekannt. Einen Anhalt für die **Größenordnung** liefert der nun folgende **Friedman-Test**. Beide Verfahren zusammen sollten dann ausreichen, die Länge des Schlüsselwortes festzulegen.

2.11 Der Friedmann-Test

William Friedman entwickelte 1925 diesen Test, der auch unter dem Namen **Kappa-Test** bekannt ist.

Zunächst einmal geht es um die Frage: *mit welcher Wahrscheinlichkeit sind zwei zufällig aus einem Klartext gegriffene Buchstaben gleich?*

Sei F eine Buchstabenfolge der Länge n, n_1 die Anzahl der as, n_2 die Anzahl der bs, ..., n_{26} die Anzahl der zs. Dann bestimmt sich die Zahl der möglichen Paare aus gleichen Buchstaben so:

1. Möglichkeiten für die Wahl des ersten a: n_1
2. Möglichkeiten für die Wahl des zweiten a: $n_1 - 1$
3. Die Reihenfolge ist egal, also ergibt sich: $n_1(n_1 - 1)/2$

4. Gesamtzahl der Paare aus gleichen Buchstaben:

$$n_1(n_1-1)/2 + n_2(n_2-1)/2 + n_3(n_3-1)/2 + ... + n_{26}(n_{26}-1)/2 =$$

$$\sum_{i=1}^{26} \frac{n_i(n_i-1)}{2}$$

5. Gesamtzahl **aller** Paare: $n(n-1)/2$

6. Damit erhält man für den **Friedmanschen Koinzidenzindex** I:

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

Für Texte einer natürlichen Sprache sind die Häufigkeiten des Auftretens einzelner Buchstaben recht genau bekannt. So können wir den Koinzidenzindex I auch anders berechnen:

1. Sei p_1 die Wahrscheinlichkeit des Buchstabens a.
2. Dann ist die Wahrscheinlichkeit des Paares aa ${p_1}^2$.
3. Damit ergibt sich die Wahrscheinlichkeit eines Paares aus **beliebigen** Buchstaben als
$$\sum_{i=1}^{26} {p_i}^2$$
4. Für deutsche Texte erhält man:
$$\sum_{i=1}^{26} {p_i}^2 = 0.0762$$
5. Für ein Gemisch aus zufälligen Buchstaben ergibt sich:
$$p_i = 1/26 \;\; \forall i, \sum_{i=1}^{26} {p_i}^2 = 0.0385$$

Die Summe der Quadrate der Wahrscheinlichkeiten ist also gerade der Koinzidenzindex I.
I wird größer, wenn der Text **unregelmäßiger** wird, die Buchstabenverteilung also der einer natürlichen Sprache entspricht.

I wird kleiner, wenn die Buchstaben **gleichmäßiger** verteilt sind.

Jetzt ist die folgende Überlegung wichtig: bei einer **monoalphabetischen** Chiffrierung, die ja nur eine Permutation der Buchstaben ist, werden die Häufigkeiten zusammen mit den Buchstaben permutiert. Wird z.B. e durch r ersetzt, so hat im Geheimtext das r die Häufigkeit 0.17, die in der deutschen Sprache zu e gehört.

Also verändert sich I bei einer monoalphabetischen Chiffrierung nicht, wohl aber bei einer **polyalphabetischen**, da hier ja die Häufigkeiten nivelliert werden. In diesem Fall wird I kleiner.

Somit haben wir jetzt einen Test, der es erlaubt, zu entscheiden, ob ein Text aus einer mono- oder einer polyalphabetischen Chiffrierung stammt: wir berechnen dazu den Koinzidenzindex; liegt dieser in der Gegend um 0.0762, so haben wir es mit möglicherweise mit einer monoalphabetischen Chiffrierung zu tun.

Wie berechnet man mit diesen Informationen nun die Schlüsselwortlänge l eines Vigenère-verschlüsselten Textes?

Da eine polyalphabetische Chiffre verwendet wurde, ist $I < 0.0762$. Sei l die Schlüsselwortlänge. Wir schreiben den Geheimtext in l Spalten:

S_1	S_2	S_3	S_4	S_5	S_6	...	S_l
$l+1$	$l+2$	$l+3$	$l+4$	$l+5$	$l+6$	...	$2l$
$2l+1$	$2l+2$	$2l+3$	$2l+4$	$2l+5$	$2l+6$	...	$3l$
$3l+1$	$3l+2$	$3l+3$	$3l+4$	$3l+5$	$3l+6$	...	$4l$
$4l+1$	$4l+2$	$4l+3$	$4l+4$	$4l+5$	$4l+6$	...	$5l$
$5l+1$	$5l+2$	$5l+3$	$5l+4$	$5l+5$	$5l+6$	...	$6l$
...							

Jede **Spalte** ist aus einer **monoalphabetischen Chiffre** entstanden. Daher ist hier die Wahrscheinlichkeit für ein Paar aus gleichen Buchstaben $= 0.0762$

Die Wahrscheinlichkeit für ein paar gleicher Buchstaben in **verschiedenen Spalten** ist etwa 0.0385.

Wir zählen nun die Anzahl der Buchstabenpaare aus gleichen und verschiedenen Spalten:

1. Die Textlänge ist n, also haben wir für den ersten Buchstaben n Möglichkeiten.
2. Nach der Wahl des ersten Buchstaben liegt die Spalte fest. In jeder Spalte stehen n/l Buchstaben, also haben wir für den zweiten Buchstaben $n/l - 1$ Möglichkeiten. Die Reihenfolge ist egal, also erhalten wir für die Anzahl der Paare, die sich in der gleichen Spalte befinden
$$n(n/l - 1)/2 = \frac{n(n-l)}{2l}$$
3. Es gibt $n - n/l$ Buchstaben **außerhalb** einer bestimmten Spalte, die Anzahl möglicher Paare ist damit
$$n(n - n/l)/2 = \frac{n^2(l-1)}{2l}$$
4. Und damit erhalten wir als erwartete Anzahl A von Paaren aus gleichen Buchstaben:
$$A = \frac{n(n-l)}{2l}0.0762 + \frac{n^2(l-1)}{2l}0.0385$$

5. Die **Wahrscheinlichkeit für ein Paar aus gleichen Buchstaben** ist damit

   Anzahl der günstigen Möglichkeiten / Anzahl aller Möglichkeiten =

   $$\frac{A}{\frac{n(n-1)}{2}} =$$
   $$\frac{n-l}{l(n-1)}0.0762 + \frac{n(l-1)}{l(n-1)}0.0385 =$$
   $$\frac{1}{l(n-1)}(0.0377n + l(0.0385n - 0.0762))$$

   Der Koinzidenzindex I ist eine Annäherung an diese Zahl, also

   $$I \approx \frac{0.0377n}{l(n-1)} + \frac{0.0385n - 0.0762}{n-1}$$

   Auflösen nach l:

   $$l \approx \frac{0.0377n}{(n-1)I - 0.0385n + 0.0762}$$

   Für einen gegebenen Text kann man n und die n_i bestimmen und I berechnen; daraus erhält man dann mit der letzten Formel die ungefähre Schlüsselwortlänge l.

6. Nun ordnet man den Text in l Spalten an. Jede Spalte ist aus einer Verschiebechiffre entstanden. Hier muss man nur noch das Äquivalent eines Buchstabens, z.B. e bestimmen und schon kann man die Post fremder Leute lesen.

Kapitel 3

Sicherheit von Chiffriersystemen

In diesem Kapitel werden wir uns mit der Sicherheit von Chiffriersystemen beschäftigen. Häufig ist eine *perfekte* Sicherheit gar nicht gefordert. Es genügt, wenn der unbefugte Mitleser erst dann die Nachricht entziffern kann, wenn die Information wertlos geworden ist.

Wenn ein Mitbewerber das vertrauliche Kaufangebot seines Konkurrenten erst nach Abschluss des Kaufvertrages erfährt, kann er sein Angebot nicht mehr anpassen. Und die Meldung einer bevorstehenden Invasion, die drei Jahre nach Ende der Friedensverhandlungen entschlüsselt wird, ist auch nicht mehr besonders aufregend.

Wir werden aber dennoch sehen, dass es das **perfekte Verfahren** gibt. Wir beginnen mit der Klärung einiger wichtiger Begriffe

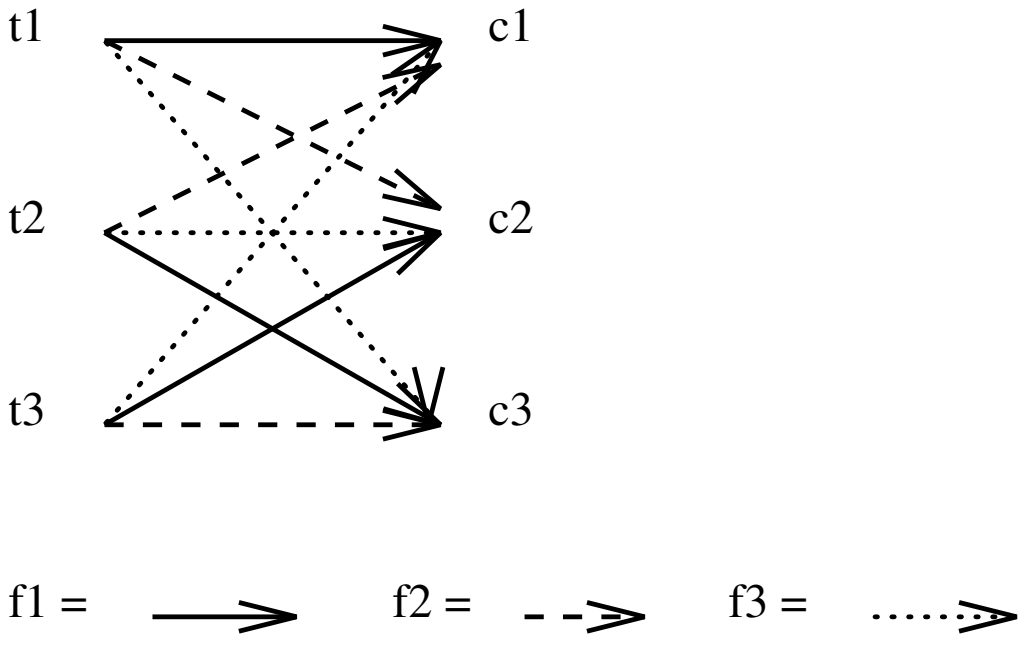
3.1 Chiffriersysteme

Definition:

Ein **Chiffriersystem** S besteht aus einer Menge K von **Klartexten**, einer Menge C von **Geheimtexten**und einer Menge T von **Transformationen**, die die Klartexte mit Hilfe eines Schlüssels k in die Geheimtexte überführen:

$$S = (K, C, T)$$

Das folgende Beispiel zeigt ein Chiffriersystem, das mit drei Transformationen drei Klartexte in drei Geheimtexte abbildet.



Damit der Empfänger den Geheimtext c_i entschlüsseln kann, muss für jede Transformation f_k die **inverse Transformation** ${f_k}^{-1}$ existieren:

$$ {f_k}^{-1}(f_k(t_i)) = t_i $$

Es kann durchaus sein, dass zwei verschiedene Transformationen verschiedene Klartexte auf den gleichen Geheimtext abbilden.

Die Anzahl der Geheimtexte muss aber mindestens so groß sein, wie die Anzahl der Klartexte, da bei keinem Geheimtext zwei Pfeile mit gleichem Namen ankommen dürfen: $|K| <= |C|$

3.2 Perfekte Sicherheit

Perfekte Sicherheit soll bedeuten, dass ein Angreifer keine Chance hat, seine Kenntnisse über das Chiffriersystem zu vergrößern, auch wenn er alle Rechner dieser Welt benutzt.

Beispiel:

Wir betrachten als Klartexte die Buchstaben, die in Goethes *Faust* vorkommen. Diese Texte haben also alle die Länge 1. Die Wahrscheinlichkeit $p(e)$, dass eine bestimmte Nachricht gerade der Buchstabe e ist, beträgt dann etwa 0.174. Der Wert $p(x)$ heißt auch *a priori-Wahrscheinlichkeit* von x.

Betrachten wir einen Geheimtext y: unter der *a posteriori-Wahrscheinlichkeit* $p_y(x)$ verstehen wir die Wahrscheinlichkeit, dass der Geheimtext y zum Klartext x gehört.

Beispiel:

Betrachten wir wieder die Buchstaben aus *Faust*: Wenn wir alle diese Buchstaben/Klartexte mit allen möglichen Verschiebechiffren auf Geheimtexte abbilden, so gehen von jedem Klartext 26 Pfeile zu den entsprechenden Geheimtexten aus.

Betrachten wir nun einen beliebigen Geheimtextbuchstaben y. Etwa 17.4% der Pfeile, die bei ihm ankommen, gehen von Klartexten aus, die aus dem Buchstaben e bestehen. Also ist $p_y(e) = p(e)$. Dies gilt für **alle** Geheimtexte und für **alle** Klartexte.

Ein Angreifer kann also soviele Texte analysieren, wie er will: er kann sein Wissen über das System nicht vergrößern. Ein solches Chiffriersystem bietet **perfekte Sicherheit**.

Verschiebechiffren bieten also perfekte Sicherheit, wenn sie auf einzelnen Buchstaben arbeiten!

Anders sieht es mit Systemen aus, bei denen $p(e) <> p_y(e)$ gilt.

Wir betrachten hierzu als Klartexte die einzelnen Seiten des Romans *Die Firma*. Diese Texte bilden wir mit Verschiebechiffren auf Geheimtexte ab. Wir haben also soviele Klartexte wie *Die Firma* Seiten hat, und 26 mal soviele Geheimtexte. Für jeden Klartext t ist $p(t) = 1/(Seitenzahl)$.

Für einen Geheimtext y und einen Klartext s kann man $p_y(s)$ berechnen: wenn die Buchstabenverteilung in y der verschobenen Verteilung der Buchstaben in s entspricht, ist $p_y(s) = 1$ sonst ist $p_y(s) = 0$.

Gilt $p_y(s) > p(s)$ bedeutet dies, dass y mit großer Wahrscheinlichkeit zu s gehört; ist $p_y(s) < p(s)$ so weiß man, dass y mit hoher Wahrscheinlichkeit nicht von s stammt. In beiden Fällen hat man etwas gelernt.

3.3 Kriterien für perfekte Sicherheit

Kriterium 1:

In einem perfekten Chiffriersystem S kann jeder Klartext mit einem geeigneten Schlüssel in jeden beliebigen Geheimtext abgebildet werden.

Von jedem Klartext führt also (mindestens) ein Pfeil zu jedem Geheimtext. Da S perfekt ist, gilt $p(x) = p_y(x)$. Da $p(x) > 0$ folgt daraus, dass $p_y(x) > 0$. Also gibt es einen Schlüssel, der x in y überführt. (Sonst wäre $p_y(x) = 0$).

Kriterium 2:

Wenn $S = (K, C, T)$ perfekt ist, gilt $|T| >= |C| >= |K|$

Wir hatten bereits gesehen, dass in jedem System mindestens soviele Geheimtexte wie Klartexte vorhanden sein müssen. Für jeden Geheimtext braucht man aber auch mindestens eine Transformation.

Kriterium 3:

$S = (K, C, T), |K| = |C| = |T|$; jeder Schlüssel komme mit der gleichen Wahrscheinlichkeit vor. Für jeden Klartext x und jeden Geheimtext y existiert genau eine Transformation t, die x in y abbildet. Dann ist S perfekt.

3.4 One-time Pads

Wir stellen nun ein perfektes System vor. Die Klartexte seien Buchstabenfolgen der Länge n; $|K| = 26^n$. Als Schlüssel wählen wir ebenfalls alle 26^n Folgen von Buchstaben der Länge n. Jede Folge wird mit der gleichen Wahrscheinlichkeit gewählt. Als Verschlüsselungsalgorithmus verwenden wir die Vigenère-Chiffre. Mit dem 3. Kriterium sieht man, dass dieses System perfekt ist.

Dieses System wurde 1917 von *Gilbert S. Vernam* erfunden und **one-time pad** genannt, weil man die Schlüssel auf den Blättern eines Papierblocks notierte, die nach einmaligem Gebrauch abgerissen wurden.

Im zweiten Weltkrieg wurden die Ergebnisse der Entschlüsselungstruppe in *Bletchley Park* an *Churchill* mit Hilfe eines one-time pad übermittelt. Der Nachteil dieser Methode besteht natürlich in der Generierung und Übermittlung der Schlüssel.

Die moderne Variante arbeitet auf Bits. Die einzelnen Bits der Nachricht werden mit den Bits des Schlüssels mit der **XOR**-Operation verknüpft:

```
Nachricht:  10011 01110 01110
Schluessel: 10101 11010 10011

Code:       00110 10100 11101
```

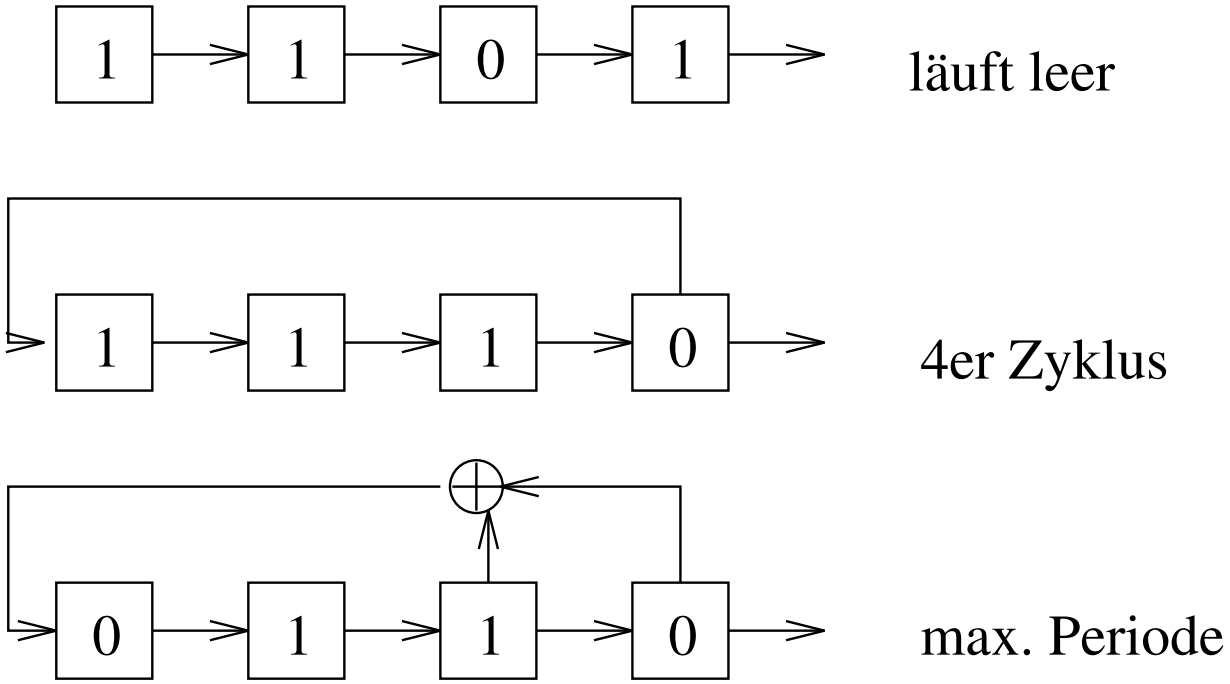
Im nächsten Abschnitt sehen wir, wie man das Problem der Schlüsselerzeugung und -verteilung angehen kann.

3.5 Schieberegister

Um das Problem des Austauschs langer, zufälliger Bitfolgen zu umgehen, kann man Folgen von **Pseudozufallszahlen** verwenden, die mit einem **Zufallszahlengenerator** erzeugt werden. Dann muss nur noch die Initialisierungsfolge für den Generator übertragen werden.

Ein solches System bietet dann zwar keine perfekte Sicherheit mehr, kann aber als Teil eines kryptographischen Verfahrens immer noch sinnvoll sein.

In der Praxis lassen sich Pseudozufallszahlen durch **Schieberegister** erzeugen, die mit einem Anfangswert versehen und geeignet rückgekoppelt werden. Die Rückkopplung ist nötig, weil sonst ein Schieberegister der Länge n nach n Takten leergelaufen wäre.



Das letzte Schieberegister nimmt nacheinander die folgenden Zustände an:

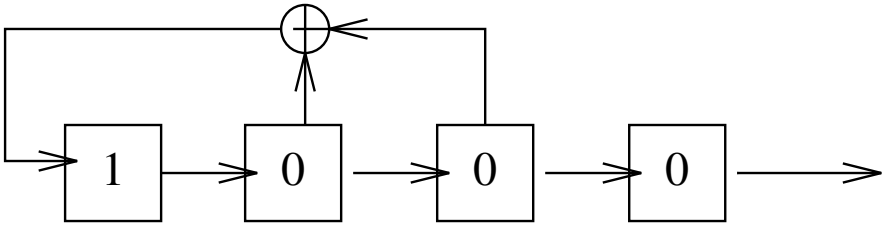
0	1	1	**0**
1	0	1	**1**
0	1	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**
1	1	1	**1**
0	1	1	**1**
0	0	1	**1**
0	0	0	**1**
1	0	0	**0**
0	1	0	**0**
0	0	1	**0**
1	0	0	**1**
1	1	0	**0**
0	1	1	**0**

Die **Periode** dieses Schieberegisters ist **maximal**. Ein Schieberegister der Länge n kann maximal 2^n verschiedene Zustände annehmen. Da der Nullzustand nicht wieder verlassen werden kann, sind für uns nur Schieberegister interessant, die diesen Zustand nicht annehmen; die maximale Periode beträgt also $2^n - 1$.

3.6 Kryptoanalyse für lineare Schieberegister

Das Problem bei der Anwendung linearer Schieberegister liegt darin, dass bei einer Klartext/Geheimtextkompromittierung eine relativ kurze Folge von bits genügt, um Initialisierung und Aufbau des Schieberegisters berechnen zu können.

Betrachten wir dazu das folgende lineare Schieberegister:

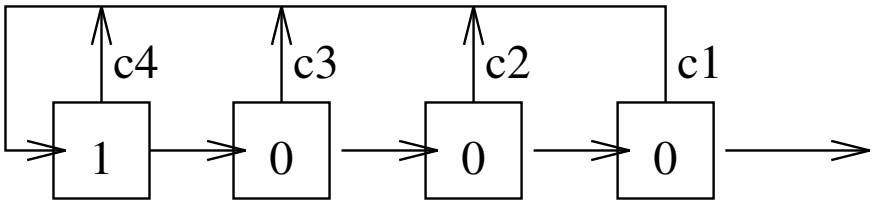


1	0	0	**0**
0	1	0	**0**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**
0	1	1	**1**
0	0	1	**1**
1	0	0	**1**
0	1	0	**0**

Angenommen dem Angreifer sind acht zusammengehörige Klartext/Geheimtextbits bekannt; dann kann er durch die **XOR**-Operation von Klartext und Geheimtext den Schlüssel errechnen:

Klartext:	0	0	1	0	0	0	1	1
Geheimtext:	1	1	0	0	1	0	1	1
Schlüssel:	1	1	1	0	1	0	0	0

Wenn der Angreifer nun vermutet, dass ein vierstelliges Schieberegister verwendet wurde, kann er folgendermaßen vorgehen:
Er nimmt zunächst einmal an, dass **alle** bits rückgekoppelt wurden. Jeder Rückkopplung i wird ein Faktor c_i zugeordnet, der den Wert 1 hat, wenn die Rückkopplung existiert, 0 sonst:



Nun takten wir:

1	0	0	0	
$c_4 \cdot 1$	1	0	0	
$c_3 \cdot 1 + c_4 \cdot c_4 \cdot 1 = c_3 + c_4$	$c_4 \cdot 1 = c_4$	1	0	
$c_4 \cdot (c_3 + c_4) + c_3 \cdot c_4 + c_2 \cdot 1 = c_4 + c_2$	$c_3 + c_4$	c_4	1	
$c_1 \cdot 1 + c_2 \cdot c_4 + c_3 \cdot (c_3 + c_4) + c_4 \cdot (c_4 + c_2)$	$c_4 + c_2$	$c_3 + c_4$	c_4	$=$
$c_1 + c_3 + c_4 + c_3 \cdot c_4$	$c_4 + c_2$	$c_3 + c_4$	c_4	

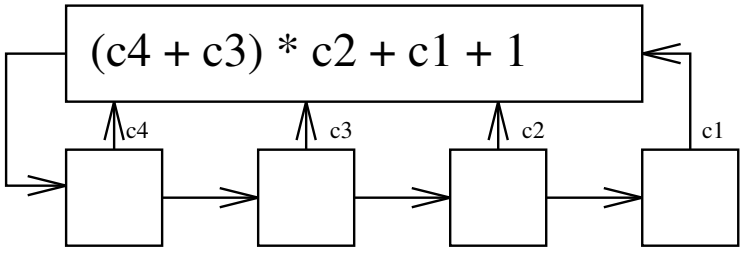
Die nun folgenden bits sind aber bekannt:

$c_4 = 0$
$c_3 + c_4 = 1$
$c_2 + c_4 = 1$
$c_1 + c_3 + c_4 + c_3 \cdot c_4 = 1$

also:
$c_4 = 0\ c_3 = 1\ c_2 = 1\ c_1 = 0$

Man kann beweisen, dass man ein lineares Schieberegister mit der maximalen Periode $2^n - 1$ "knacken" kann, wenn $2n$ aufeinanderfolgende, zusammengehörige Klartext/Geheimtextbits bekannt sind. Eine Folge der Länge $1048575 = 2^{20} - 1$ kann man also bereits aus $2 * 20 = 40$ bits rekonstruieren.

Man kann diese Probleme umgehen, wenn man die Rückkopplung nicht linear macht, d.h. auch u.a. Multiplikationen verwendet:



Insbesondere geht hier der Nullzustand nicht in sich selbst über.

Kapitel 4

Der DES-Algorithmus

Die Beschreibung der Arbeitsweise des DES-Algorithmus folgt der Darstellung in [13].

4.1 Feistel-Netzwerke

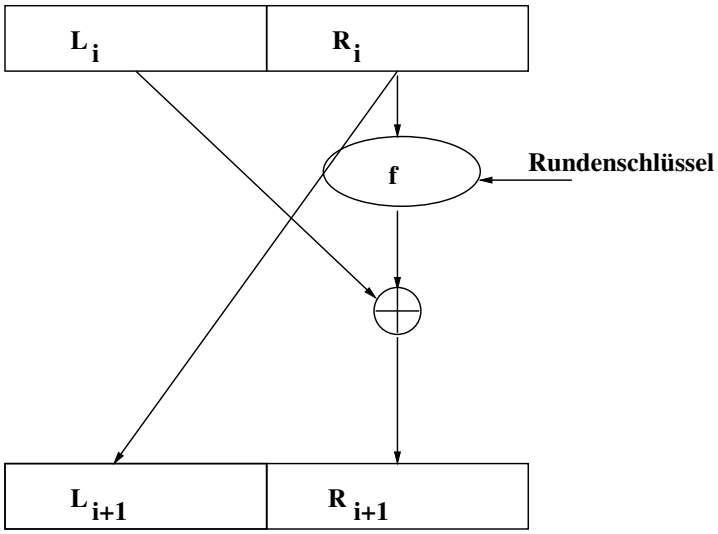
Feistel-Netzwerke wurden in den Siebzigerjahren von Horst Feistel, einem Mitarbeiter der IBM, veröffentlicht.

Ein Klartextblock wird zunächst in zwei Hälften geteilt, die in der Runde i mit L_i und R_i bezeichnet werden.

Ferner existiert ein geheimer Schlüssel S und eine Funktion $f_{S,i}$, die auf R_i angewendet wird.

Verschlüsselt wird durch die Verknüpfung mit XOR von L_i und $f_{S,i}(R_i)$ und Vertauschen der beiden Hälften:
$L_{i+1} = R_i$

$R_{i+1} = L_i$ XOR $f_{S,i}(R_i)$



4.1.1 Dechiffrieren in Feistel-Netzwerken

Normalerweise würden wir fordern, dass $f_{S,i}$ bei bekanntem Schlüssel umkehrbar ist. Bei einem Feistel-Netzwerk fordern wir nur, dass $f_{S,i}$ nur mit Hilfe des Schlüssels berechnet werden kann. $f_{S,i}$ kann damit beliebig kompliziert werden, wir können dennoch dechiffrieren, denn

x XOR $x = 0$, daher

$f_{S,i}(R_i)$ XOR $f_{S,i}(R_i) = 0$ also

$L_i = L_i$ XOR $f_{S,i}(R_i)$ XOR $f_{S,i}(R_i) = R_{i+1}$ XOR $f_{S,i}(R_i)$

Damit können wir dechiffrieren: L_n, R_n seien gegeben, dann gilt:

$R_{n-1} = L_n$

$L_{n-1} = R_n$ XOR $f_{S,n-1}(R_{n-1})$

$R_0 = L_1$

$L_0 = R_1$ XOR $f_{S,0}(R_0)$

4.2 Historie

1973 Das **National Bureau of Standards** (NBS) im US-Handelsministerium schreibt einen Wettbewerb für einen neuen Verschlüsselungsstandard aus.

1977 wird der **Data Encryption Standard** von **IBM** vorgestellt. Die **NSA** hatte bei der Evaluierung mitgewirkt.

Die **Schlüssellänge** beträgt 56 bit.

Hinterfragt wird die Struktur der Substitutionstabellen (S-Boxes).

Verdacht einer durch die NSA eingebauten Hintertür.

1991 **Shamir** entwickelt eine Methode, die für die Analyse von DES "nur" noch 2^{47} Vergleiche benötigt, 2^9 mal weniger als vorher.

Es stellt sich heraus, daß dieses Verfahren der **differentiellen Cryptoanalyse** bereits 1973 bekannt war und **DES** besonders dagegen gestärkt wurde.

4.3 Die Ausschreibungsbedingungen

- hohes Sicherheitsniveau
- vollständig spezifiziert und einfach zu verstehen
- alle Sicherheit liegt im Schlüssel, der Algorithmus ist offen
- verfügbar für alle Anwender
- adaptierbar für unterschiedliche Anwendungen
- ökonomisch in Hardware implementierbar
- effizient im Gebrauch
- validierbar
- exportierbar

4.4 Der Algorithmus im Überblick (1)

DES ist eine Blockchiffre, 64 bit Klartext werden in 64 bit Geheimtext überführt.

Das Verfahren ist **symmetrisch**, Ver- und Entschlüsseln geschieht mit dem gleichen Algorithmus.

Die **Schlüssellänge** beträgt 56 bits (plus 8 Paritätsbits).

Abhängig vom Schlüssel werden in jeder "Runde" **Substitutionen und Permutationen** durchgeführt.

DES besteht aus **16 Runden**.

Es werden nur einfache **logische** und **arithmetische** Operationen verwendet, die schon 1974 leicht in Hardware zu implementieren waren.

4.5 Der Algorithmus im Überblick (2)

64 bit Klartext werden **permutiert**.

Aufteilung in 32 bit linke/rechte Hälfte.

Kombinieren von Daten und Schlüssel in 16 Runden.

Dann **Zusammensetzen** der beiden Hälften und abschließende Permutation.

Im Detail:
In jeder Runde **shift der Schlüsselbits und Auswahl** von 48 aus 56 bits.

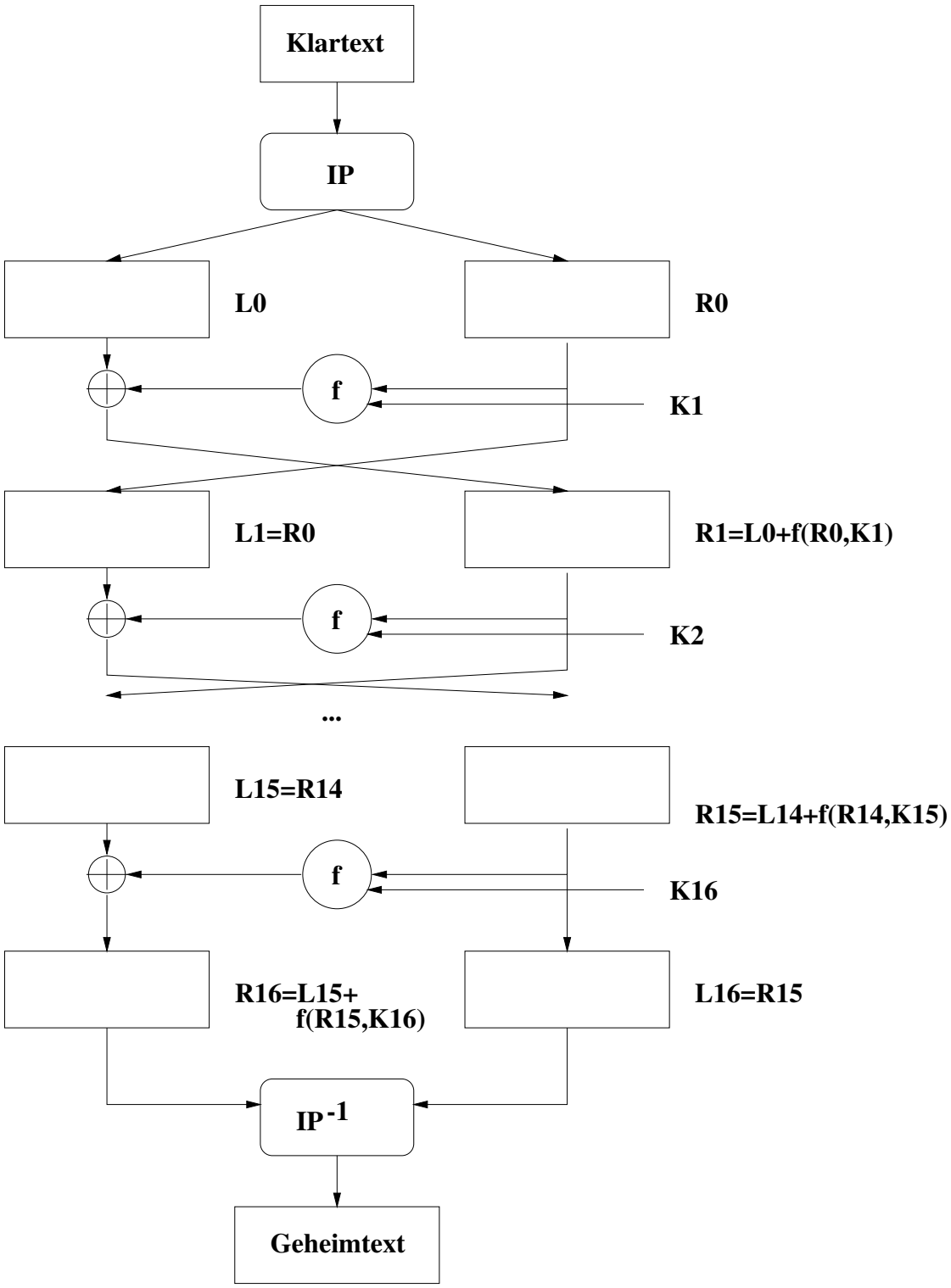
Expansion der rechten 32 bit zu 48 bit, **XOR** mit den permutierten Schlüsselbits.

Dann **Substitution** und **Permutation**.

Verknüpfen des Outputs und der linken Hälfte mit **XOR**.

Resultat wird die neue linke, die alte linke die neue rechte Hälfte.

4.6 Der Algorithmus im Überblick (3)



4.7 Die Anfangspermutation

```
/* Sofware DES functions
 * written 12 Dec 1986 by Phil Karn, KA9Q;
 * large sections adapted from
 * the 1977 public-domain program by Jim Gillogly
 */

/* Tables defined in the
   Data Encryption Standard documents */

/* initial permutation IP */
static char 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
};
```

Es wird hier bit 1 zu bit 58, bit 2 zu bit 50 etc..

4.8 Die Abschlusspermutation

Die Abschlusspermutation ist die Inverse zur Anfangspermutation:

```
/* final permutation IP^-1 */
static char fp[] = {
    40,  8, 48, 16, 56, 24, 64, 32,
    39,  7, 47, 15, 55, 23, 63, 31,
    38,  6, 46, 14, 54, 22, 62, 30,
    37,  5, 45, 13, 53, 21, 61, 29,
    36,  4, 44, 12, 52, 20, 60, 28,
    35,  3, 43, 11, 51, 19, 59, 27,
    34,  2, 42, 10, 50, 18, 58, 26,
    33,  1, 41,  9, 49, 17, 57, 25
};
```

Die Eingabe ist $R_{16}L_{16}$, die Hälften werden nach der letzten Runde also nicht vertauscht. So kann der gleiche Algorithmus zum Ver- und Entschlüsseln verwendet werden.

Bemerkung:
Durch die Anfangs- und Abschlußpermutation wird die Sicherheit von DES nicht beeinflusst.

Daher lassen manche Softwareimplementationen diese beiden Schritte auch weg, da sie in Software mühsam zu implementieren sind. In Hardware ist die Implementation kein Problem.

4.9 Die Schlüsseltransformation (1)

Jedes 8. bit des 64-bit Schlüssels wird ignoriert und kann daher zur Paritätskontrolle verwendet werden.

Nach der Extraktion des 56-bit Schlüssels wird in jeder Runde ein neuer Schlüssel K_i generiert:

Die Anfangspermutation - extraktion beschreibt die folgende Tabelle:

```
/* permuted choice table (key) */
static char pc1[] = {
    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};
```

4.10 Die Schlüsseltransformation (2)

Dann teilt man den Schlüssel in zwei 28-bit Hälften, die abhängig von der aktuellen Runde 1 oder 2 bits nach links geshiftet werden.

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

Eine Kompressionspermutation wählt dann 48 der 56 bits aus:

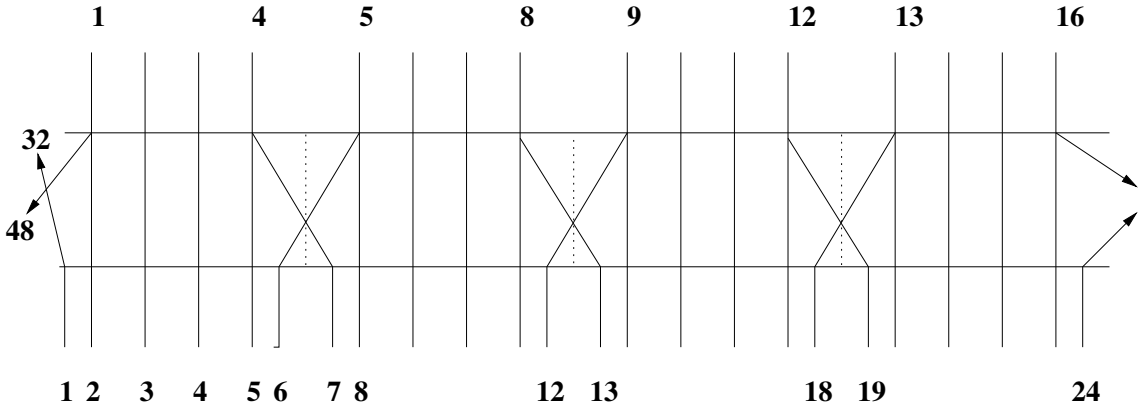
```
/* permuted choice key (table) */
static char pc2[] = {
    14, 17, 11, 24,  1,  5,
     3, 28, 15,  6, 21, 10,
    23, 19, 12,  4, 26,  8,
    16,  7, 27, 20, 13,  2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32};
```

4.11 Die Expansionspermutation

Diese Operation expandiert die rechte Seite R_i von 32 auf 48 bits. Danach haben Daten und Schlüssel wieder die gleiche Länge.

Hierdurch wird auch erreicht, dass möglichst schnell jedes bit der Ausgabe von jedem bit der Eingabe und des Schlüssels abhängt:

```
static char ei[] = {
    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 };
```



[18]

4.12 Die S-Boxen (1)

Nachdem der komprimierte Schlüssel mit dem expandierten Block durch **XOR** verknüpft wurde, schließt sich eine Substitutionsoperation an, die durch 8 S-Boxen definiert wird.

Die 48 bits werden in 8 6-bit-Blöcke zerlegt, die jeweils von einer S-Box bearbeitet werden. Jede S-Box besteht aus 4 Zeilen und 16 Spalten, jeder Eintrag ist eine 4-bit Zahl.

Seien $b_1, ..., b_6$ die bits eines Blocks. Dann werden b_1 und b_6 kombiniert, um eine Zahl zwischen 0 und 3 zu bilden, die eine Zeile in der S-Box selektiert. Die Spalte wird durch $b_2b_3b_4b_5$ angewählt.

Beispiel:

Angenommen die bits 31 bis 36 der Eingabe, die den Input für die 6. S-Box bilden, sind 110010.

$b_1b_6 = 10, b_2...b_5 = 1001$

dann werden Zeile 2 und Spalte 9 selektiert (Zählung ab 0).
Dieser Eintrag ist 0, also wird 110010 durch 0000 ersetzt.

4.13 Die S-Boxen (2)

```
static char si[8][64] = {
    /* S1 */
    14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7,
     0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
     4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
    15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13,
    /* S2 */
    15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10,
     3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5,
     0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15,
    13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9,
    /* S3 */
    10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8,
    13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
    13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
     1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12,
    /* S4 */
     7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15,
    13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
    10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
     3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14,

    /* S5 */
     2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9,
    14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
     4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
    11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3,
    /* S6 */
    12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11,
    10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
     9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
     4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13,
    /* S7 */
     4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1,
    13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
     1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
     6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12,
    /* S8 */
    13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7,
     1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
     7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
     2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11};
```

4.14 Die P-Box

Die entstandenen 8 4-bit Blöcke werden zu einem 32-bit-Block kombiniert. Anschließend wird das Ergebnis mit der linken Hälfte der 64 bit mit **XOR** verknüpft, rechte und linke Hälfte vertauscht und die nächste Runde beginnt.

```
static char p32i[] = {
    16,  7, 20, 21,
    29, 12, 28, 17,
     1, 15, 23, 26,
     5, 18, 31, 10,
     2,  8, 24, 14,
    32, 27,  3,  9,
    19, 13, 30,  6,
    22, 11,  4, 25};
```

4.15 Entschlüsseln von DES

Zur Entschlüsselung werden die 16 Rundenschlüssel in umgekehrter Reihenfolge verwendet.

Der Algorithmus zur Schlüsselerzeugung ist ebenfalls zirkular, d.h. es wird rechts geshiftet und die Tabelle mit der Anzahl der Shifts von rechts nach links gelesen.

4.16 DES-Anwendungsmodi (1)

ECB Electronic Codebook Mode

64-bit-Blöcke werden unabhängig voneinander chiffriert.

Vorteil:

Ver- und Entschlüsseln kann im wahlfreien Zugriff erfolgen. Das ist z.B. bei Datenbanken wichtig.

Nachteil:

In verschiedenen Klartexten werden gleiche Blöcke gleich verschlüsselt. Dies erleichtert eine Klartext/Geheimtext-Kompromittierung. Insbesondere bei email treten gleiche Textblöcke am Anfang auf. Wenn die Schlüssel nicht sehr häufig gewechselt werden, gibt es die folgende Angriffsmöglichkeit:

Beispiel: Geldtransfer
Der Angreifer hört den Verkehr zwischen Bank 1 und Bank 2 ab und transferiert mehrfach 100 Euro von Bank 1 nach Bank 2. Wenn er die richtige Message identifiziert hat, speist er sie mehrfach ein und wird reich!

Diesen Betrug kann man durch einen Zeitstempel verhindern, aber:

wenn der Angreifer weiß, in welchen Blöcken Name und Konto stehen, kann er diese Blöcke austauschen und andere Transfers auf sein Konto umleiten.

Diese Technik wird als **block replay** bezeichnet.

4.17 DES-Anwendungsmodi (2)

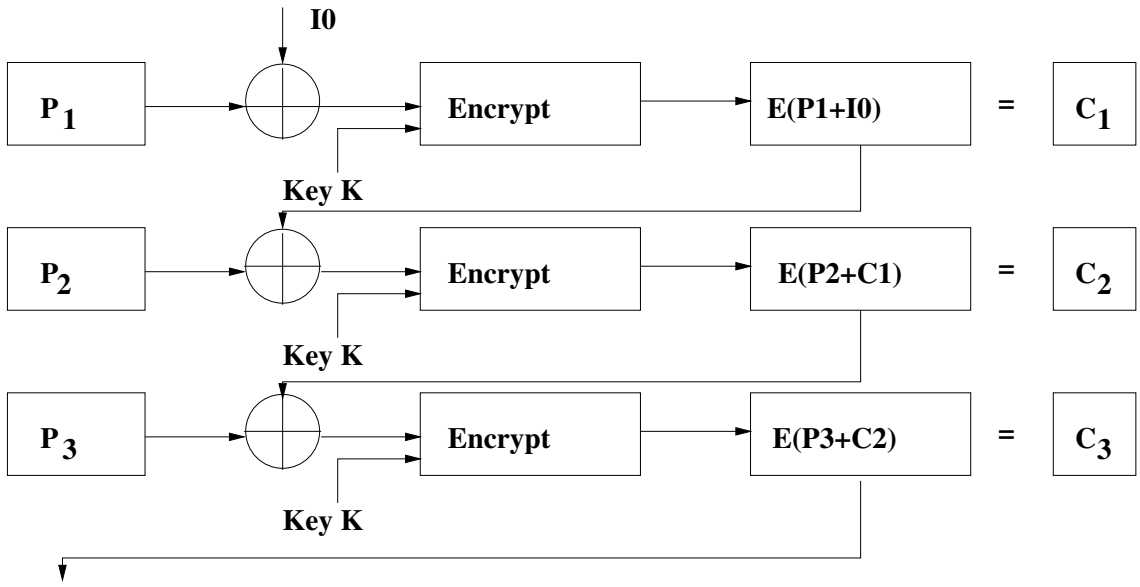
CBC Cipher Block Chaining Mode

Hier werden die Ergebnisse der Verschlüsselung vorangegangener Blöcke für die Verschlüsselung des aktuellen Blocks benutzt. Damit hängt die Verschlüsselung eines Blocks von **allen** vorangegangenen Blöcken ab.

$C_i = E_k(P_i \ \mathbf{XOR} \ C_{i-1})$

$P_i = C_{i-1} \ \mathbf{XOR} \ D_k(C_i)$

Der erste Block wird mit einem beliebigen Initialisierungsvektor verknüpft.



4.18 Sicherheit von DES

Befürchtung einer durch die **NSA** eingebauten Hintertür

Untersuchung durch ein **Senatskomitee 1978** erbringt keinen Hinweis

Tuchman: "NSA did not dictate a single wire!"

Einzelheiten der Untersuchung sind immer noch geheim

4.18.1 Schwache Schlüssel

Bei diesen Schlüsseln ändern sich die Rundenschlüssel nicht:

Schlüsselwert	aktueller Schlüssel
0101 0101 0101 0101	0000000 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF
1F1F 1F1F 1F1F 1F1F	0000000 FFFFFFF
E0E0 E0E0 E0E0 E0E0	FFFFFFF 0000000

4.18.2 Semi-schwache Schlüssel und möglicherweise schwache Schlüssel

Die folgenden Schlüssel erzeugen nur zwei verschiedene Rundenschlüssel:

```
01FE 01FE 01FE 01FE  FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1  E01F E01F F10E F10E
01E0 01E0 01F1 01F1  E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE  FE1F FE1F FE0E FE0E
011F 011F 010E 010E  1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE  FEE0 FEE0 FEF1 FEF1
```

Ferner gibt es noch 48 “possibly weak keys”, die nur 4 verschiedene Rundenschlüssel generieren.
Insgesamt sind aber nur 64 von 72,057,594,037,927,936 Schlüsseln schwach.

Schneier: “Wenn Sie echt paranoid sind, kann man diese bei der Schlüsselgenerierung ausschließen.”

4.18.3 Ist DES eine Gruppe?

Man hat lange Zeit gerätselt, ob es sinnvoll ist, eine Nachricht mit DES mehrfach zu verschlüsseln:

$E_{K_2}(E_{K_1}(P))$

Wenn es **keinen** Schlüssel K_3 gibt, mit

$E_{K_3}(P) = E_{K_2}(E_{K_1}(P))$,

bringt dies mehr Sicherheit.

Erst 1992 wurde bewiesen, das DES **keine Gruppe** ist.

4.18.4 Schlüssellänge

1979 behaupteten Diffie und Hellman, dass ein Spezialcomputer für 20 Millionen US-Dollar DES in einem Tag brechen könnte.

1981 setzten sie ihre Schätzung auf 2 Tage und 50 Millionen US-Dollar herauf und prophezeiten, dass DES 1990 völlig unsicher sein würde.

Hellman bezweifelt, dass heute private Firmen für weniger als 10 Millionen Dollar eine solche Maschine bauen könnten. Zudem besteht dann immer noch die Notwendigkeit, in die Kommunikatiosnnetze einzudringen.

Regierungsinstitutionen wie der NSA oder dem französischen Geheimdienst DGSE ist dies aber durchaus zuzutrauen.

1980 lieferte **Robotron** mehrere 100.000 DES-Chips in die Sowjetunion. Entweder verschlüsseln die mit **DES** - dann hat die **NSA** bestimmt eine Entschlüsselmaschine - oder die Russen haben eine solche gebaut ...

4.18.5 Anzahl der Runden

1982 wurde **DES** mit 3 und 4 Runden gebrochen.

Biham und Shamir zeigten, dass **DES** mit weniger als 16 Runden durch einen Known-Plaintext-Angriff leichter als durch brute-force zu brechen ist.

4.18.6 Die S-Boxen

Es gibt kein System linearer Gleichungen, um die Beziehungen zwischen den 6 Input- und den 4 Output-bits zu beschreiben.

Die Änderung eines Input-bits bewirkt die Änderung von mindestens zwei Output-bits.

4.18.7 Zum Design des Algorithmus

In [18] finden sich die folgenden Erklärungen für die Komplexität des DES-Algorithms:

- Die Erweiterungspermutation und die P-Box sorgen für den Lawineneffekt.
- Die P-Boxen bewirken, dass ein Klartextbit bei jeder Runde möglichst eine andere S-Box durchläuft.
- Die S-Boxen bewirken Nichtlinearität und Immunität gegen differentielle Kryptoanalyse.
- Die Erzeugung der Rundenschlüssel bewirkt, dass jede Änderung eines Schlüsselbits möglichst schnell alle Geheimtextbits beeinflußt.
- Nur die Anfangs- und Endpermutation sind kryptologisch bedeutungslos.

4.18.8 Brute-Force-Angriff

Man probiert einfach 2^{56} mögliche Schlüssel durch, das sind allerdings 72 Billiarden! ;-)

Bei der **RSA-Challenge** wurde dieser Angriff auf 50.000 Prozessoren im Internet verteilt. 22.000 Anwender machten mit und hatten im Februar 1998 nach 39 Tagen den Code geknackt.

In [18] wird vorgeschlagen, einen wahrscheinlichen Klartextblock von 8 byte Länge mit allen möglichen Schlüsseln zu verschlüsseln und die resultierenden Geheimtextblöcke auf 850 Millionen CDs zu speichern.

Der folgende Abschnitt zeigt, dass man auch mit weniger Platz auskommen kann.

4.18.9 Time-Memory Tradeoff 1

Dieses Verfahren wurde von Hellmann bereits 1980 beschrieben. Wir folgen der Darstellung in [18]. Hier werden nicht alle möglichen Geheimtexte abgespeichert, sondern nur ein kleiner Teil, während der Rest während der Analyse berechnet wird.

Vorausgesetzt wird ein zusammengehöriges Paar aus Klartext P und Geheimtext C. Außerdem brauchen wir eine Funktion R, die 64-Bit-Blöcke auf 56-Bit-Blöcke verkürzt, z.B. durch Abschneiden des obersten Bytes.

Dann betrachten wir die Funktion
$f(S) = R(E_S(P))$
$E_S(P)$ ist dabei die Verschlüsselungsfunktion. Wir suchen alle Schlüssel S, für die gilt:
$f(S) = R(C)$
S kann der gesuchte Schlüssel sein, muss es aber nicht, da 8 Bits noch nicht getestet wurden.

Man wählt nun zufällig Schlüssel S_1, S_m und berechnet folgende Tabelle:

$S_1 f(S_1) f^2(S_1) ... f^t(S_1)$

...

$S_n f(S_n) f^2(S_n) ... f^t(S_n)$

Es werden nur die erste und letzte Spalte gespeichert.

4.18.10 Time-Memory Tradeoff 2

In der letzten Spalte sucht man nun $R(C)$. Falls gefunden, berechnen wir $f^{t-1}(S_k)$

Finden wir den Eintrag nicht, suchen wir $f(R(C))$, falls gefunden, berechnen wir $f^{t-2}(S_k)$, usw..

Zusammengefasst:

- Tabelle berechnen, 1. und letzte Spalte speichern.
- R(C) in der letzten Spalte suchen.
- Wenn nicht gefunden f(R(C)) suchen, usw..
- Wenn gefunden, Tabellenelement in gleicher Zeile, aber eine Spalte vorher berechnen.

Dieses Verfahren führt mit hoher Wahrscheinlichkeit zum Ziel.

Hellmann schlug vor, eine Million Tabellen mit 100.000 Zeilen und 1 Million Spalten von 10.000 DES Chips berechnen zu lassen. Als Hauptspeicherbedarf gab er 125 MB, Massenspeicherbedarf 1 TByte an.

1993 schaffte der DES-Chip VM007 25 Millionen Verschlüsselungen pro Sekunde, 100.000 dieser Chips wären in einem halben Tag fertig.

4.18.11 Differenzielle Kryptoanalyse

Paare von Klartext/Klartext bzw. Geheimtext/Geheimtext, die bestimmte Differenzen aufweisen.

Bestimmte Differenzen im Klartext haben eine hohe Wahrscheinlichkeit in den zugehörigen Geheimtexten wiederaufzutauchen. Wenn der Unterschied z.B. `0080 8200 6000 0000` beträgt, so gibt es eine 5% Wahrscheinlichkeit, dass die resultierenden Geheimtexte die gleiche Differenz aufweisen.

Auf diese Weise werden mögliche Schlüssel mit Wahrscheinlichkeiten versehen und der wahrscheinlichste ausgewählt.

Der Angriff ist allerdings eher theoretischer Natur. Um einen Schlüssel zu finden müßte man **chosen plaintext** mit 1.5Mbits/sec für drei Jahre verschlüsseln.
Coppersmith von **IBM** erklärte, dass dieser Angriff bereits 1974 bekannt war. Zu einer Frage von **Shamir**, ob inzwischen bessere Angriffe bekannt seien, gab er keinen Kommentar.

4.19 Nachfolger von DES: IDEA

- 1992 International Data Encryption Algorithm
- Schneier: "best and most secure block algorithm available to the public at this time"
- Blockchiffre
- Schlüssellänge 128 bits
- Klartextblöcke von 64 bit
- Idee: Mischung von Operationen verschiedener algebraischer Gruppen:
  - XOR
  - Addition mod 2^{16}
  - Multiplikation mod $2^{16} + 1$
- alle Operationen auf 16 bit Teilblöcken

4.19.1 Schlüssel

Es werden insgesamt 52 Schlüssel benötigt.

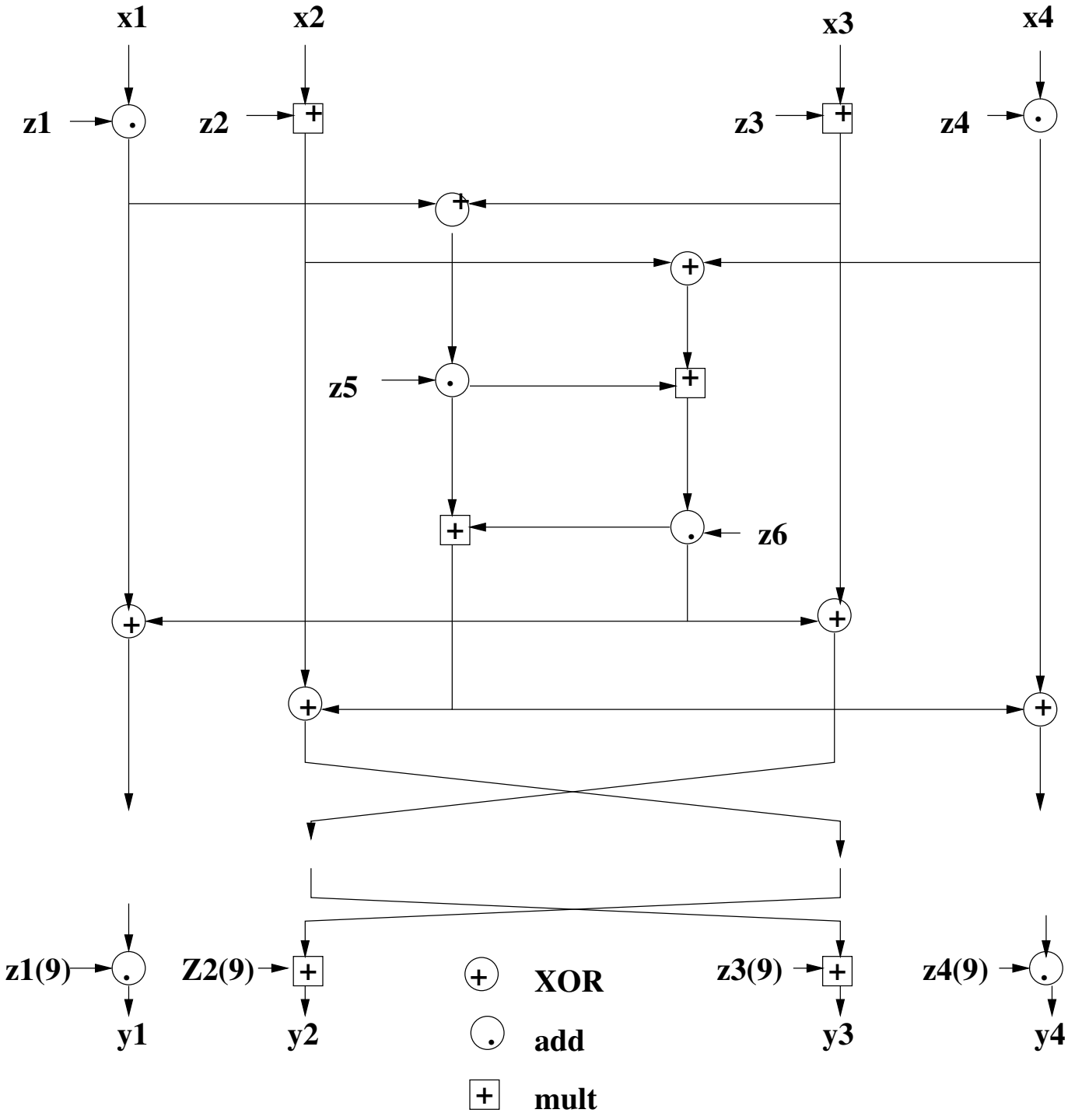
Zunächst wird der 128-bit Schlüssel in 8 16-bit Teilschlüssel zerlegt.

Davon werden 6 für die erste, 2 für die 2. Runde benötigt.

Dann folgt ein 25 bit Linksshift und wieder Aufteilung in 8 Teilschlüssel: 4 für die 2., 4 für die 3. Runde.

Insgesamt werden 8 mal 6 Schlüssel und 4 Schlüssel für die Ausgabetransformation benötigt.

Zum Entschlüsseln sind die additiven bzw. multiplikativen Inversen der Teilschlüssel zu berechnen.


x1
x2
x3
x4
z1
z2
z3
z4
z5
z6
z1(9)
Z2(9)
z3(9)
z4(9)
y1
y2
y3
y4
XOR
add
mult

Runde	Verschlüsselung	Entschlüsselung
1:	$Z_1^{(1)} Z_2^{(1)} Z_3^{(1)} Z_4^{(1)} Z_5^{(1)} Z_6^{(1)}$	$Z_1^{(9)^{-1}} - Z_2^{(9)} - Z_3^{(9)} Z_4^{(9)^{-1}} Z_5^{(8)^{-1}} Z_6^{(8)^{-1}}$
2:	$Z_1^{(2)} Z_2^{(2)} Z_3^{(2)} Z_4^{(2)} Z_5^{(2)} Z_6^{(2)}$	$Z_1^{(8)^{-1}} - Z_2^{(8)} - Z_3^{(8)} Z_4^{(8)^{-1}} Z_5^{(7)^{-1}} Z_6^{(7)^{-1}}$
3:	$Z_1^{(3)} Z_2^{(3)} Z_3^{(3)} Z_4^{(3)} Z_5^{(3)} Z_6^{(3)}$	$Z_1^{(7)^{-1}} - Z_2^{(7)} - Z_3^{(7)} Z_4^{(7)^{-1}} Z_5^{(6)^{-1}} Z_6^{(6)^{-1}}$
4:	$Z_1^{(4)} Z_2^{(4)} Z_3^{(4)} Z_4^{(4)} Z_5^{(4)} Z_6^{(4)}$	$Z_1^{(6)^{-1}} - Z_2^{(6)} - Z_3^{(6)} Z_4^{(6)^{-1}} Z_5^{(5)^{-1}} Z_6^{(5)^{-1}}$
5:	$Z_1^{(5)} Z_2^{(5)} Z_3^{(5)} Z_4^{(5)} Z_5^{(5)} Z_6^{(5)}$	$Z_1^{(5)^{-1}} - Z_2^{(5)} - Z_3^{(5)} Z_4^{(5)^{-1}} Z_5^{(4)^{-1}} Z_6^{(4)^{-1}}$
6:	$Z_1^{(6)} Z_2^{(6)} Z_3^{(6)} Z_4^{(6)} Z_5^{(6)} Z_6^{(6)}$	$Z_1^{(4)^{-1}} - Z_2^{(4)} - Z_3^{(4)} Z_4^{(4)^{-1}} Z_5^{(3)^{-1}} Z_6^{(3)^{-1}}$
7:	$Z_1^{(7)} Z_2^{(7)} Z_3^{(7)} Z_4^{(7)} Z_5^{(7)} Z_6^{(7)}$	$Z_1^{(3)^{-1}} - Z_2^{(3)} - Z_3^{(3)} Z_4^{(3)^{-1}} Z_5^{(2)^{-1}} Z_6^{(2)^{-1}}$
8:	$Z_1^{(8)} Z_2^{(8)} Z_3^{(8)} Z_4^{(8)} Z_5^{(8)} Z_6^{(8)}$	$Z_1^{(2)^{-1}} - Z_2^{(2)} - Z_3^{(2)} Z_4^{(2)^{-1}} Z_5^{(1)^{-1}} Z_6^{(1)^{-1}}$
outtransf:	$Z_1^{(9)} Z_2^{(9)} Z_3^{(9)} Z_4^{(9)}$	$Z_1^{(1)^{-1}} - Z_2^{(1)} - Z_3^{(1)} Z_4^{(1)^{-1}}$

4.20 Analyse von IDEA

brute force benötigt 2^{128} Versuche.

10^9 Chips, die 10^9 Schlüssel/sec testen brauchen immer noch 10^{13} Jahre, länger als das Alter des Universums.

10^{24} Chips finden den Schlüssel in 1 Tag - aber es gibt im Universum nicht genügend Materie.

Zudem benötigt die Berechnung der Inversen zu viel Zeit.

Verbesserung der Sicherheit durch **Triple-IDEA**:

$C = E_{K_1}(D_{K_2}(E_{K_1}(P)))$

$P = D_{K_1}(E_{K_2}(D_{K_1}(C)))$

Die Theorie hinter dem Algorithmus beruht darauf, dass $2^{16}+1$ eine Primzahl ist. $2^{32}+1$ ist nicht prim, daher hätte der Algorithmus bei einer Blockgröße von 32 völlig andere Eigenschaften.

Schneier: *IDEA looks very secure ... There are several military and academic groups currently cryptanalyzing IDEA. None of them has succeeded yet, but one might someday.*

Kapitel 5

Der RSA-Algorithmus

Alle bisher verwendeten Verschlüsselungsverfahren setzen voraus, dass die beteiligten Parteien sich auf ein **gemeinsamen** Schlüssel einigen, der dann sowohl zum Chiffrieren als auch zum Dechiffrieren verwendet wird. Diese Verfahren werden auch als **symmetrische Verfahren** bzw. **private-key** Verfahren bezeichnet.

Das Problem liegt hier darin, dass dieser geheime Schlüssel ja auch irgendwie übermittelt werden muss. Zudem wird auch noch eine **große Anzahl** Schlüssel benötigt: wenn n Personen miteinander paarweise kommunizieren wollen, müssen $n \cdot (n-1)/2$ Schlüssel verteilt werden.

1977 wurde nun von *Rivest , Shamir* und *Adleman* ein verblüffend einfaches Verfahren vorgestellt, dass zwei verschiedene Schlüssel verwendet: einen **öffentlichen** Schlüssel (*public key*) zum Verschlüsseln und einen **privaten** Schlüssel (*private key*) zum Entschlüsseln.

Der *public key* kann in einer Art Telefonbuch veröffentlich werden, der *private key* ist nur dem Empfänger bekannt. Nur der kann eine mit seinem *public key* verschlüsselte Nachricht entziffern.

Der RSA-Algorithmus beruht auf der Tatsache, dass es Funktionen gibt, die sich zwar sehr einfach berechnen lassen, deren *Umkehrfunktion* jedoch mit endlichem Rechenaufwand nicht zu berechnen ist. Diese Funktionen werden auch als *Einwegfunktionen (one-way functions)* bezeichnet.

In den folgenden Abschnitten werden wir die Grundlagen für das Verstehen des RSA-Algorithmus legen.

5.1 Die Eulersche Phi-Funktion

Definition:

$$\phi(n) = |\{t : t \in \mathcal{N} \wedge t < n \wedge ggt(t,n) = 1\}| = n * \prod_{p|n} (1 - 1/p)$$

$\phi(n)$ bezeichnet die Anzahl der zu n teilerfremden positiven ganzen Zahlen, die kleiner als n sind. Im zweiten Ausdruck nimmt p die Werte aller Teiler von n an.

Beispiel:

$\phi(3) = |\{1,2\}| = 2 \quad \phi(12) = |\{1,5,7,11\}| = 4$

Satz:
p Primzahl $\Rightarrow \phi(p) = p - 1$

Satz:
p, q Primzahlen; dann gilt: $\phi(pq) = (p-1)(q-1)$

Beweis:
$|\{t : t \in \mathcal{N} \wedge t < pq\}| = pq - 1$
Die $(q-1)$ Vielfachen von p, also p, $2p$, .. $(q-1)p$
und die $(p-1)$ Vielfachen von q, also q, $2q$, ... $(p-1)q$
sind **nicht** teilerfremd zu pq.
Damit erhält man:
$\phi(pq) = pq - 1 - (q-1) - (p-1) = pq - q - p + 1 = (p-1)(q-1)$

5.2 Der Satz von Euler

Die folgenden Resultate haben wir schon im Abschnitt über die mathematischen Grundlagen kennen gelernt.

Satz:
$m, n \in \mathcal{N},\ ggt(m,n) = 1 \ \Rightarrow\ m^{\phi(n)}$ **mod** $n = 1$

Spezialfall (Der kleine Satz von Fermat):
$m \in \mathcal{N},\ p, q \in \mathcal{N},\ p,\ q$ Primzahlen,
$p \neq q,\ ggt(m,p) = ggt(m,q) = 1 \Rightarrow$
$m^{(p-1)(q-1)}$ **mod** $pq = 1$

Mit diesem Satz kann man leicht nachweisen,
dass $7^{\phi(4)}$ **mod** $4 = 7^2$ **mod** $4 = 1$ gilt.
Das ist noch nicht so furchbar aufregend.
Aber man kann auch leicht die folgende Behauptung beweisen:

47^{1932} **mod** $2021 = 1$ Warum? Ganz einfach:

47^{1932} **mod** $2021 = 47^{42*46}$ **mod** $(43 * 47) = 47^{\phi(43*47)}$ **mod** $(43 * 47) = 1$

Dieser Spezialfall wird noch sehr wichtig werden!

5.3 Der größte gemeinsame Teiler (ggt)

Der *ggt* ist wichtig für die Berechnung der RSA-Schlüssel; daher folgt noch ein wenig Zahlentheorie.

Wir betrachten zwei natürliche Zahlen a, b o.B.d.A $a > b$. Ein positive Zahl d, die sowohl a als auch b teilt ist ein **gemeinsamer Teiler** von a und b. Es kann natürlich mehrere solcher gemeinsamen Teiler geben. Den größten dieser gemeinsamen Teiler bezeichnen wir als $ggt(a, b)$.

Man überlegt sich leicht, dass ein gemeinsamer Teiler von a und b auch jede Linearkombination von a und b teilt:

$d|a \ \wedge \ d|b \ \Rightarrow \ d|(a * x + b * y), \ x, y \in \mathcal{Z}.$

Außerdem gilt: $ggt(a, b) \ = \ d, \ d'|a \ \wedge \ d'|b \ \Rightarrow \ d'|d,$
d.h jeder gemeinsame Teiler von a und b teilt $ggt(a, b)$.

Wichtig ist im Folgenden der

Satz:

$a, b \in \mathcal{N} \Rightarrow ggt(a, b)$ ist das minimale positive Element aus
$\{a * x + b * y | x, y \in \mathcal{Z}\}$

Satz:

Es gilt: $a = \lfloor a/n \rfloor * n + a \textbf{ mod } n$

Sei $s = a * x + b * y$ minimal, $q = \lfloor a/s \rfloor$

Wir zeigen $s|a \wedge s|b$:

$a \textbf{ mod } s = a - q * s = a - q * (a * x + b * y) = a * (1 - q * x) + b * (-q * y)$

Also ist $a \textbf{ mod } s$ auch eine Linearkombination aus a und b. $a \textbf{ mod } s < s$, s war minimal, also gilt $a \textbf{ mod } s = 0 \Rightarrow s|a$. Analog zeigt man $s|b$.

Dann gilt aber: $ggt(a, b) >= s$.

Wir wissen aber
$ggt(a, b)|(a * x + b * y) \Rightarrow ggt(a, b) <= s \Rightarrow ggt(a, b) = s$

q.e.d.

Satz:
$ggt(a, b) = 1 \Rightarrow \exists x, y \in \mathcal{Z} : 1 = a * x + b * y$

Für die effektive Berechnung des ggt ist die folgende Tatsache wichtig:

Satz:
$ggt(a, b) = ggt(b, a \textbf{ mod } b)$

Beweis:
$a, b \in \mathcal{N}$, $d = ggt(a, b), a = q * b + a \textbf{ mod } b, q = \lfloor a/b \rfloor$

$d|a \wedge d|b \Rightarrow d|a - q * b = a \textbf{ mod } b$

$d|b \wedge d|a \textbf{ mod } b \Rightarrow d|ggt(b, a \textbf{ mod } b)$

Sei $d = ggt(b, a \textbf{ mod } b); a = q * b + a \textbf{ mod } b \Rightarrow$

a ist Linearkombination aus b und $a \textbf{ mod } b \Rightarrow d|a$. Aber $d|b \Rightarrow d|ggt(a, b)$

q.e.d.

5.4 Der Euklidische Algorithmus - rekursive Version

Die vorangegangenen Überlegungen machen die Formulierung eines Algorithmus zur Berechnung des $ggt(a, b)$ trivial. Der **Euklidische Algorithmus** erlaubt die einfache Berechnung des größten gemeinsamen Teilers (ggt) zweier Zahlen. Bei kleinen Zahlen könnte man auch die gemeinsamen Primfaktoren bestimmen. Das ist bei großen Zahlen aber nicht mehr machbar.

```
static long euclid(long a, long b){
  return (b == 0) ? a : euclid(b, a % b);}
```

Wir werden für das **RSA-Verfahren** auch die Berechnung des $ggt(a, b)$ als Linearkombination aus a und b benötigen. Auch diese Funktion wird einfach:

```
static void extEuclid(int a, int b, LinComb o) {
    LinComb o1;
    if (b == 0) {o.setD(a); o.setX(1); o.setY(0);}
    else {o1 = new LinComb();
          extEuclid(b, a % b, o1);
          o.setD(o1.getD()); o.setX(o1.getY());
          o.setY(o1.getX() - (a/b) * o1.getY());}}
```

Die Korrektheit von `extEuclid` sieht man leicht ein:

Für $b == 0$ ist die Sache klar;
$b! = 0 \Rightarrow d1 = ggt(b, a \textbf{ mod } b) = x1 * b + y1 * (a \textbf{ mod } b)$

$$d = d1 = b * x1 + (a \ \mathbf{mod}\ b) * y1 = x1 * b + (a - \lfloor a/b \rfloor * b) * y1$$
$$= a * y1 + b * (x1 - \lfloor a/b \rfloor * y1)$$

5.5 Der Euklidische Algorithmus - iterative Version

Dieser Abschnitt zeigt noch einmal eine **iterative** Formulierung des Euklidischen Algorithmus, die im Vergleich zur **rekursiven** Definition wesentlich uneleganter ist:

Beispiel:

$a = 38,\ b = 8$
$38 = 4 * 8 + 6$
$8 = 1 * 6 + 2$
$2 = 1 * 2 + 0$

Offenbar ist $ggt(38, 8) = 2$. Berechnet haben wir $ggt(a, b)$ so:

1. Dividiere die größere Zahl durch die kleinere.
2. Setze die kleinere Zahl an die Stelle der größeren, den Rest an die Stelle der kleineren Zahl.
3. Wiederholen, bis der Rest Null wird.

Allgemein heißt das:

$x_0 = a$
$x_1 = b$
$x_0 = q_1 * x_1 + x_2 \quad 0 < x_2 < x_1$
$x_1 = q_2 * x_2 + x_3 \quad 0 < x_3 < x_2$
$x_2 = q_3 * x_3 + x_4 \quad 0 < x_4 < x_3$
...
$x_{i-1} = q_i * x_i + x_{i+1} \quad 0 < x_{i+1} < x_2$
...
$x_{n-2} = q_{n-1} * x_{n-1} + x_n$
$x_{n-1} = q_n * x_n$

1. x_n teilt a, b denn:

x_n teilt x_{n-1} also auch $q_{n-1} * x_{n-1} + x_n = x_{n-2}$ ...
also auch x_0, x_1

2. x_n ist auch der **größte** gemeinsame Teiler:

Angenommen y teilt x_0, x_1. Dann teilt y auch $x_2 = x_0 - q_1 * x_1$
und daher auch $x_3 = x_1 - q_2 * x_2$.
Das kann man fortsetzen, bis man bei x_n angekommen ist.

Der Algorithmus sieht im Prinzip so aus:

```
int a, b, r;

do
{
 read(a); read(b);
}while ((a <= 0) || (b <= 0)); // a, b > 0!!

while (b != 0)
{
 r = a % b;
 a = b;
 b = r;
}

return b;
```

5.6 Modulare Inverse

Den folgenden Satz haben wir bereits kennengelernt. Hier noch einmal eine Formulierung, die auf der iterativen Version des Euklidischen Algorithmus aufbaut.

Satz: (Vielfachsummendarstellung des ggt):

$a, b \in \mathcal{N},\ d = ggt(a,b) \ \Rightarrow\ \exists x, y \in \mathcal{Z} :\ d = x * a + y * b$

Beweis:
$d = x_n = x_{n-2} - q_{n-1} * x_{n-1} = x_{n-2} + (-q_{n-1}) * x_{n-1} =$

$x_{n-2} + (-q_{n-1}) * [x_{n-3} - q_{n-2} * x_{n-2}] =$

$(-q_{n-1}) * x_{n-3} + [1 + q_{n-1} * q_{n-2}] * x_{n-2} =$

$[...] * x_{n-4} + [...] * x_{n-3} =$

...

$x * x_0 + y * x_1 = x * a + y * b$

Satz: (Modulare Inverse)

$a, b \in \mathcal{N},\ ggt(a,b) = 1 \Rightarrow \exists c \in \mathcal{Z} : (b * c)\ \mathbf{mod}\ a = 1$

Beweis:

$d = ggt(a,b) = 1\ \Rightarrow\ \exists x, y \in \mathcal{Z} : 1 = x * a + b * y$

$y * b = 1 - x * a$

$(y * b)\ \mathbf{mod}\ a = (1 - x * a)\ \mathbf{mod}\ a$

a teilt $x * a \Rightarrow (y * b)\ \mathbf{mod}\ a = 1 \Rightarrow$ Beh. mit $c = y$

Wir halten fest:

1. Der Euklidische Algorithmus kann sehr einfach und effizient implementiert werden.
2. Es ist leicht, herauszufinden, ob zwei ganze Zahlen teilerfremd sind.
3. Für zwei teilerfremde Zahlen a und b kann man leicht eine Zahl c finden, so dass $(b * c)\ \mathbf{mod}\ a = 1$

5.7 Schlüsselgenerierung beim RSA-Verfahren

1. Man wähle zwei **große** Primzahlen p, q und bilde ihr Produkt $n = p*q$. n wird auch als *Modul* bezeichnet.
2. Man berechne $\phi(n) = (p-1) * (q-1)$
3. Man wähle Zahlen $e, d \; in \mathcal{Z}$ mit $(e * d)$ **mod** $\phi(n) = 1$.
4. (e, n) bildet den **öffentlichen Schlüssel**,
   (d, n) den **privaten Schlüssel**.
5. $p, q, \phi(n)$ kann man vernichten.

Für e kann man jede Primzahl wählen, die größer als $\phi(n)$ ist. Ein (kleines) Problem entsteht bei der Wahl der Primzahlen p und q. Es gibt zwar unendlich viele Primzahlen, aber es ist sehr schwer, von einer Zahl mit z.B. 100 Ziffern festzustellen, ob sie eine Primzahl ist. Die Lösung ist hier, sogenannte *Pseudoprimzahlen* zu verwenden. Das sind Zahlen sich wie "echte" Primzahlen verhalten, alle leicht nachprüfbaren Primzahltests erfüllen und von denen man mit beliebiger Wahrscheinlichkeit ausschließen kann, dass sie keine Primzahlen sind.

5.8 Anwendung des RSA-Algorithmus

Die zu verschlüsselnde Nachricht sei eine Folge von Zahlen: $0 <= m < n$.
Der **Verschlüsselungsalgorithmus** ist dann:

$c = m^e \textbf{ mod } n$

Der **Entschlüsselungsalgorithmus:** $m' = c^d \textbf{ mod } n$

Satz: $m = m'$

Beweis:
Wir beweisen diese Behauptung in mehreren Schritten.

Zunächst einmal gilt:

$\forall a, b, c \in \mathcal{N} : (a * b) \textbf{ mod } c = ((a \textbf{ mod } c) * (b \textbf{ mod } c)) \textbf{ mod } c$

Also: $m' = c^d \textbf{ mod } n = (m^e \textbf{ mod } n)^d \textbf{ mod } n = m^{e*d} \textbf{ mod } n$

Wir müssen also zeigen: $\forall m : \ m^{e*d} \textbf{ mod } n = m$

Um diese Behauptung zu beweisen zeigen wir,
dass gilt: $(m^{e*d} - m) \textbf{ mod } n = 0$

$n = p * q$, wir werden beweisen:
$(m^{(e*d)} - m) \textbf{ mod } p = 0$ und $(m^{(e*d)} - m) \textbf{ mod } q = 0$

Es gilt $e * d \textbf{ mod } \phi(n) = 1$. Daraus folgt:

$\exists k \in \mathcal{Z} : e * d = k * \phi(n) + 1 = 1 + k * (p-1)(q-1)$

Angenommen m, p sind **nicht** teilerfremd. Da p Primzahl ist, gilt dann:

$m \textbf{ mod } p = 0$ und $m^{e*d} \textbf{ mod } p = 0$ also die Behauptung.

Sind m, p teilerfremd, gilt mit dem Satz von Euler $m^{\phi(p)} \textbf{ mod } p = 1$

$m^{e*d} \textbf{ mod } p = m^{1+k*\phi(n)} \textbf{ mod } p = m^{1+k*(p-1)*(q-1)} \textbf{ mod } p =$

$(m * m^{k*(p-1)*(q-1)}) \textbf{ mod } p =$

$(m * m^{k*\phi(p)*(q-1)}) \textbf{ mod } p = (m * m^{\phi(p)*k*(q-1)}) \textbf{ mod } p =$

$(m * [m^{\phi(p)} \textbf{ mod } p]^{k*(q-1)}) \textbf{ mod } p =$

$(m * [1]^{k*(q-1}) \textbf{ mod } p = m \textbf{ mod } p$

Für q gilt die analoge Rechnung: $m^{e*d} \textbf{ mod } q = m \textbf{ mod } q$

Also gilt:$(m^{e*d} - m) \textbf{ mod } q = 0$ und $(m^{e*d} - m) \textbf{ mod } p = 0$

und damit: $(m^{e*d} - m) \textbf{ mod } p * q = 0$

q.e.d

5.9 Beispiele für kleine Primzahlen

Die folgende Tabelle gibt konkrete Zahlenbeispiele. Man beachte, dass die Primzahlen p und q für realistische Anwendungen zu klein und damit ungeeignet sind!

p	13	103	83
q	19	127	89
n	247	13081	7387
$\phi(n)$	216	12852	7216
e	307	14983	7219
d	19	1363	4811
m_1	$1 \rightarrow 1$	$1 \rightarrow 1$	$2 \rightarrow 8$
m_2	$2 \rightarrow 154$	$2 \rightarrow 11184$	$200 \rightarrow 7266$
m_3	$5 \rightarrow 138$	$200 \rightarrow 9956$	$400 \rightarrow 6419$
m_4		$13079 \rightarrow 1897$	$255 \rightarrow 4987$

5.10 Modulare Exponentiation

Beim RSA-Verfahren benötigt man einen effizienten Algorithmus, um $c = m^e$ **mod** n und $m = c^d$ **mod** n berechnen zu können.

Das Verfahren des **wiederholten Quadrierens** benutzt die Binärdarstellung des Exponenten für die Berechnung:

```
  static int modExp(int a, int b, int n)
//   compute a^b mod n
  {int exp = 0x80;          // bitmask
   int e = 0;               // invariant r = a^e mod n
   int r = 1;
   int sz = 4;
   for (int i = 1; i < sz; i++)
    exp = exp << 8;         // for 4 byte integers:
                            // exp = 0x80.00.00.00
   while (exp != 0)         // beginning with the m.s.b.
   {e *= 2;
    r = (r * r) % n;
    if ((b & exp) != 0)     // if m.s.b == 1
    {e++;                   // increment exponent
     r = (r * a) % n;}
    exp = exp >>> 1;}       // next bit
   return r;}
```

In jedem Durchlauf wird e verdoppelt und - falls das aktuelle bit des Exponenten = 1 ist - um 1 erhoeht. Wenn $(b_k, b_{k-1}, ..., b_i)$ die höchsten bits des Exponenten sind, ist nach dem Verarbeiten von bit b_i $r = a^{b_k b_{k-1} \cdots b_i}$.

5.11 Verteilung der Primzahlen

Da für jedes RSA-Schlüsselpaar zwei große Primzahlen benötigt werden, stellt sich die Frage, ob nicht die Gefahr besteht, dass irgendwann einmal alle Primzahlen verbraucht sind.

Glücklicherweise kann man hier aber ganz beruhigt sein: wenn man sich auf Zahlen mit der Maximallänge von 512 bits beschränkt, so weiß man, dass es allein in diesem Bereich mehr als 10^{150} Primzahlen gibt. dass dies eine ganze Menge ist, kann man sich daran veranschaulichen, dass die Physiker davon ausgehen, dass das Universum aus etwa 10^{84} Atomen besteht. Hätte jedes Atom seit dem Urknall vor 10^{10} Jahren pro Mikrosekunde eine Milliarde Primzahlen verbraucht (wozu auch immer), so hätte man erst 10^{115} Primzahlen verbraucht: $10^{84} * 10^{10} * 3.65 * 10^{2} * 2.4 * 10^{1} * 3.6 * 10^{3} * 10^{6} * 10^{9} \approx 10^{115}$

Große Zahlen sind ja in der Kryptographie von beonderer Bedeutung, daher hier noch einmal einige Größenvergleiche, die dem Buch von Schneier [13] entnommen sind. Eine gute Referenz für die folgenden Abschnitte ist auch [4].

Wahrscheinlichkeit durch einen Autounfall zu sterben	1:75
Wahrscheinlichkeit zu ertrinken	1:59.000
Alter der Erde	10^9 Jahre
Alter des Universums	10^{10} Jahre
Lebensdauer des Universums (geschlossenes Universum)	10^{11} Jahre
Anzahl der Atome in der Sonne	10^{57}
Volumen des Universums	$10^{84} cm^3$

5.12 Der Primzahlsatz

Die Zerlegung großer Zahlen in Primfaktoren ist ein schwieriges Problem. Im Prinzip ist es genauso schwierig, von einer großen Zahl festzustellen, ob sie eine Primzahl ist oder nicht. Das sukzessive Ausprobieren bei einer Zahl der Größenordnung 10^{150} ist schlichtweg unmöglich.

Man kann sich aber damit zufrieden geben, dass man von einer Zahl mit beliebig **großer Wahrscheinlichkeit** sagen kann, dass sie eine Primzahl ist.

Definition:

Die **Primzahlverteilungsfunktion** $\pi(n)$ bezeichne die Anzahl der Primzahlen, die kleiner oder gleich n sind.

Beispiel:

$\pi(12) = 5$, denn 2, 3, 5, 7, 11, sind alle Primzahlen <= 12.

Primzahlsatz :

$$\lim_{n\to\infty} \frac{\pi(n)}{n/ln(n)} = 1$$

Selbst für kleine n erhalten wir so brauchbare Abschätzungen: für $n = 10^9$ ist $\pi(n) = 50.847.478$ und $n/ln(n) = 48.254.942$. Dies ist eine Abweichung von weniger als 6%.

Zu einem festen n gibt es also ungefähr $n/ln(n)$ Primzahlen, die kleiner als n sind. Daher ist die Wahrscheinlichkeit, dass eine zufällig gezogene Zahl, die kleiner als n ist, eine Primzahl ist, $(n/ln(n))/n = 1/ln(n)$.

Um also eine 100-stellige Primzahl zu finden, muss man ungefähr $ln(10^{100}) \approx 230$ 100-stellige Zahlen testen. Da man gerade Zahlen gleich vergessen kann, reduziert sich dieser Aufwand auf die Hälfte.

5.13 Ein Primzahltest

Für die folgenden Überlegungen stellen wir eine einfache Ergebnisse der Zahlentheorie zusammen.

Definition:
$Z_n = \{x : 0 <= x <= n-1\}$,
$+_n$ bezeichne die **Addition mod** n.

Satz:$(Z_n, +_n)$ ist eine abelsche Gruppe.

Assoziativität und Kommutativität folgen aus der Definition von $+_n$. Die 0 ist das Einselement, das Inverse zu i ist $n - i$.

Beispiel:

$+_5$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Definition:

$Z_n^* = \{a : \ a \ \in \ \mathcal{N} \ \wedge \ a < n \ \wedge \ ggt(a,n) = 1\}$
$*_n$ bezeichne die **Multiplikation mod** n.

Satz:$(Z_n^*, *_n)$ ist eine abelsche Gruppe.

Die Assoziativität und die Kommutativität folgen aus der Definition von $*_n$. Die 1 ist das Einselement, das Inverse zu a erhält man, wenn man $ggt(a,n) = 1 = a * x + n * y$ betrachtet. Dann gilt ax **mod** $n = 1$. Somit ist x das Inverse zu a.

Beispiel:

$*_5$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Satz:

Sei $Z_n^+ = Z_n \setminus \{0\}$. Ist n eine Primzahl, so gilt $Z_n^+ = Z_n^*$.

Der **Satz von Fermat** impliziert, dass, wenn n eine Primzahl ist, a^{n-1} **mod** $n = 1$ $\forall a \in Z_n^+$ gilt. Daher darf man schließen: wenn es ein $a \in Z_n^+$ gibt, so dass a^{n-1} **mod** $n \neq 1$, dann ist n **sicher keine Primzahl**.

Die Umkehrung dieses Satzes gilt aber nur *beinahe*. Wir testen nun, ob für n die Beziehung 2^{n-1} **mod** $n = 1$ gilt. Ist dies **nicht** der Fall, so ist n sicher keine Primzahl. Ist die Bedingung erfüllt, vermuten wir, dass n eine Primzahl ist.

Wir machen nur dann einen Fehler, wenn es sich bei n um eine sogenannte *Basis-2* Pseudoprimzahl handelt. Das ist aber nur sehr selten der Fall. Unter 10.000 gibt es nur 22 Zahlen, die Basis-2 Pseudoprimzahlen sind. Die ersten vier lauten: 341, 561, 645, 1105.

Man kann zeigen, dass eine 50-stellige Zahl, die auf diese Weise als Primzahl gewählt wird, nur mit einer Chance von $1{:}10^6$ eine Basis-2 Pseudoprimzahl ist. Bei einer 100-stelligen Zahl ist die Chance kleiner als $1{:}10^{13}$.

Das folgende Codestück zeigt den Primzahltest:

```
n = myRandom(MAXPRIM); // get number < MAXPRIM
r = modExp(2, n-1, n);
if (r == 1) System.out.println("Probably Prime");
else System.out.println("Composite");
```

5.14 Der Miller-Rabin-Test

Leider kann man den Fehler nicht dadurch ausschließen, dass man einfach ein anderes $a \neq 2$ wählt. Es gibt nämlich die sogenannten *Carmichael-Zahlen* , die die Beziehung a^{n-1} **mod** $n = 1 \forall a \ \in Z_n^*$ erfüllen. Die sind aber erfreulicherweise noch seltener als die Basis-2 Pseudoprimzahlen. Es gibt nur 255 Carmichael-Zahlen, die kleiner als 10^9 sind. Die ersten drei lauten: 561, 1105, 1729.

Wir werden nun, den Test so verbessern, dass wir nicht mehr auf die Carmichael-Zahlen hereinfallen.
Im Gegensatz zum vorangegangenen Verfahren wird nicht ein Wert für a fest gewählt, sondern wir probieren mehrere, zufällige Werte für a.

Zusätzlich wird die folgende Tatsache ausgenutzt:

Wenn p eine ungerade Primzahl ist, so hat die Gleichung x^2 **mod** $p = 1$ *nur die Lösungen 1 und -1.*

Das bedeutet aber auch: wenn es eine nichttriviale Quadratwurzel von 1 **mod** n gibt, so ist n zusammengesetzt.

Beispiel: $(3 * 3)$ **mod** $4 = 1, \ \ (6 * 6)$ **mod** $35 = 1$

Der folgende Test ist eine Erweiterung des Algorithmus zur modularen Exponentiation. Er registriert in jedem Schritt das Auftreten nichttrivialer Quadradwurzeln **mod** n von 1 und erkennt so zusammengesetzte Zahlen.

```
static int witness(int a, int n){
        int exp = 0x80;
        int sz  = 4;
        int x;
        int r = 1;

         for (int i = 1; i < sz; i++) // construct bitmask
              exp = exp << 8;

         while (exp != 0)
         {x = r;
          r = (r * r) % n;
          if((r == 1) &&
             (x != 1) &&
             (x != n-1))  // found nontrivial root
          {System.out.println("Composite! Non trivial root!");
           return 1;}
          if (((n-1) & exp) != 0)
          {r = (r * a) % n;}
          exp = exp >>> 1;
         }
         if (r != 1) //a^(n-1) mod n != 1 // ==> composite
         {System.out.println("Composite! a^(n-1) mod n != 1");
          return 1;}
         return 0;}
```

```
System.out.println("Miller-Rabin");
n = read("n = ");
a = read("a = ");
r = witness(a, n);
if (r == 0)
System.out.println("Probably Prime");
else System.out.println("Composite");
```

Die Fehlerrate hängt nur von der Anzahl der Prüfungen ab. Es gilt der folgende

Satz:
Wenn n eine ungerade zusammengesetzte Zahl ist, so gibt es mindestens $(n-1)/2$ Zahlen, bei denen die Funktion `Witness` den Wert 1 liefert.

Damit ergibt sich:

Für $n \in \mathcal{N}$, n ungerade, $s \in \mathcal{N}$, ist die Wahrscheinlichkeit, dass `MillerRabin(n, s)` sich irrt, höchstens 2^{-s}.

Ein Wert von $s = 50$ ist damit in jedem Fall groß genug. Wenn n zufällig gewählt wird, kann man beweisen, dass $s = 3$ ausreichend ist.

Der Miller-Rabin-Test ist der schnellste bekannte Primzahltest!

5.15 Das Rho-Verfahren von Pollard

Zum Abschluss dieses Kapitels soll kurz ein Verfahren vorgestellt werden, das die Faktorisierung großer Zahlen erlaubt. Wenn man für eine Zahl n alle Zahlen $z <= \sqrt{n}$ ausprobiert, wird man sicher einen Faktor finden, falls n keine Primzahl ist.

Das folgende Verfahren kann mit dem gleichen Aufwand Zahlen bis zur Größe n^2 faktorisieren. Der Haken ist nur, dass es möglicherweise manchmal nicht terminiert und keine Lösung abliefert. Wenn es aber einen Faktor berechnet, so ist dieser auch richtig.

Die Fälle wo bei einer Zahl, die keine Primzahl ist, kein Faktor gefunden wird, sind aber sehr selten.

```
static void pollardRho(long n)
{int c = 0, i = 1, k = 2;
 long x = Math.round(Math.random() * (n-1)),
      y = x,
         d;
 int b;
 while (i <= 2 * n)
 {i++;
  if ((i % 1000) == 0) System.out.println('*');
  x = (x * x - 1) % n;
  d = euclid(Math.abs(y-x), n);
  if ((d != 1) && (d != n))
  {System.out.println(
    "\n" + "A factor after " + i +
    " iterations is: " + d + "\n" + "n / d = " + n/d);
    c = read("to continue enter 1);");
    if (c == 1) {n = n/d;
     System.out.println(n*d);
     i = 1; k = 2;
     x = Math.round(Math.random() * (n-1));
     y = x;}
    else break;}
  if (i == k)
  {y = x;
   k *= 2;}}
 if (c!=0) System.out.println("\nAfter " + i
            + " iterations no factor found!");
}
```

Wie funktioniert nun dieser seltsame Algorithmus? Auf einen exakten Beweis werden wir hier verzichten. Die Funktionsweise soll nur plausibel gemacht werden.

Zunächst einmal wird x mit einem Zufallswert aus Z_n initialisiert. Die Zuweisung $x = (x^2 - 1)$ **mod** n erzeugt eine (fast) zufällige Folge von x-Werten, von denen wir aber immer nur den letzten Wert benötigen.

In der Variablen y merken wir uns nacheinander den 1., 2., 4. 8. 16. x-Wert. k gibt jeweils den nächsten in y zu speichernden Wert an.

Wenn d ein nichttrivialer Teiler von n ist, wird d ausgegeben.

Da Z_n endlich ist, wird irgendwann einmal ein x zum zweiten Mal berechnet. Von diesem Zeitpunkt an wiederholt sich die ganze Folge. Wegen des *Geburtstagsparadoxons* ist dies nach etwa höchstens $\sqrt{n}$ Schritten der Fall.

Sei p nun ein nichttrivialer Faktor von n, so dass $ggt(n, n/p) = 1$. Dann induziert die Folge (x_i) ein Folge (x'_i) mit $x'_i = x_i$ **mod** p. Mit dem gleichen Schluss wie eben, wiederholt sich diese Folge nach $\sqrt{p}$ Schritten. Das können nun aber wesentlich weniger Schritte als bei der Folge (x_i) sein.

Seien nun t, u die kleinsten Werte, für die gilt: $x'_{t+i} = x'_{t+u+i}$, $\forall i >= 0$. Wenn $x'_{t+i} = x'_{t+u+i}$ dann teilt p $(x_{t+u+i} - x_{t+i})$.

Wenn k groß genug geworden ist, werden wir einmal den ganzen Zyklus durchlaufen und mit dem in y gespeicherten Wert den ggt berechnen.

Also ist $ggt((x_{t+u+i} - x_{t+i}), n) > 1$ und wir haben einen Teiler von n gefunden.

Pollards Rho Heuristik

5.16 Digitale Signaturen mit RSA

Beim Austausch elektronischer Nachrichten ist die Unterschrift fast noch wichtiger als im "richtigen" Leben, da man oft nicht sicher weiß, wem man wirklich gegenüber sitzt. Man muss sichergehen können, dass eine Nachricht wirklich von der Person abgeschickt wurde, die im Absender steht und dass diese Nachricht auch nicht von Unbefugten verändert wurde. Eine gültige Unterschrift unter einem Brief oder Vertrag kann nur vom Absender erzeugt, aber von jedem Empfänger überprüft werden. Wir werden sehen, dass eine **digitale Unterschrift** wesentlich fälschungssicherer ist als eine gewöhnliche Unterschrift mit Tinte. Mit dem RSA-Verfahren kann man sehr einfach elektronische Signaturen produzieren.

Normalerweise verwendet der Absender einer Nachricht den **öffentlichen Schlüssel** des Empfängers, um eine Nachricht zu verschlüsseln. Soll die Nachricht nur signiert werden, so wendet der Absender **seinen privaten Schlüssel** auf die Nachricht an - $c = m^{d_a}$ **mod** n - und sendet das Paar (m, c) an den Empfänger.

Der Empfänger benutzt nun den **öffentlichen Schlüssel** e_a des Absenders - $m' = c^{e_a}$ **mod** n - und prüft $m = m'$. Wir haben gesehen, dass die Exponentiation kommutativ ist, daher ist klar, dass dieses Verfahren funktioniert. Wurde m' nachträglich verändert, stellt man dieses sofort fest. Außerdem ist man sicher, dass die Nachricht vom Absender stammt, da ja nur er den privaten Schlüssel besitzt (hoffentlich).

Die Übermittlung der Nachricht m kann man sich bei natürlichsprachlichen Nachrichten sparen, wenn man davon ausgeht, dass ein sinnvoller Text ein ausreichendes Indiz für die unveränderte Übermittlung von m ist.

5.17 Diffie-Hellman -Schlüsselvereinbarung

Asymmetrische Verfahren sind etwa 1000 mal langsamer als symmetrische. Daher verwendet man in der Praxis häufig eine Kombination aus beiden Verfahren: die eigentliche Nachricht wird mit einem effizienten symmetrischen Verfahren verschlüsselt und der zugehöriger Schlüssel mit einem asymmetrischen. Da der Schlüssel relativ kurz im Vergleich zur Nachricht ist, hat man mit der Effizienz des asymmetrischen Verfahrens keine Probleme. **PGP** benutzt bekanntlich dieses Vorgehen.

Allerdings hat dieser Ansatz einen kleinen Schönheitsfehler: der Schlüssel für das symmetrische Verfahren wird nur vom Absender ausgewählt. Besser wäre es, wenn beide Partner einen Schlüssel **vereinbaren** könnten. 1978 haben **Diffie** und **Hellman** in ihrem berühmten Aufsatz *New Directions in Cryptography* dieses Problem gelöst.

Die grundlegende Idee kann man sich so veranschaulichen: beide Partner erhalten je einen identischen Koffer, der mit zwei Schlössern versehen ist, von denen jeder nur ein Schloss öffnen kann. Beide entfernen nun das Schloss, das sie öffnen können, tauschen die Koffer und haben dann jeder einen Koffer, den sie für sich aufschließen können: Nachricht ausgetauscht, alle glücklich.

Mathematisch sieht das Verfahren so aus:

Beide Teilnehmer *Alice* und *Bob* einigen sich auf eine **Primzahl** p und eine **natürliche Zahl** $1 < s < p$. Diese Zahlen dürfen öffentlich sein! p sollte um die 200 Dezimalstellen haben.

Dann wählen beide, jeder für sich,
je eine **geheime** Zahl $a < p - 1$ bzw. $b < p - 1$.

Alice berechnet $\alpha = s^a \ \mathbf{mod}\ p$.

Bob berechnet $\beta = s^b \ \mathbf{mod}\ p$.

α und β werden nun ausgetauscht.

Jetzt berechnet *Alice* $k = \beta^a \ \mathbf{mod}\ p$,

Bob $k' = \alpha^b \ \mathbf{mod}\ p$.

Nun gilt aber:

$k = \beta^a \textbf{ mod } p = (s^b \textbf{ mod } p)^a \textbf{ mod } p = s^{b*a} \textbf{ mod } p =$

$s^{a*b} \textbf{ mod } p = (s^a)^b \textbf{ mod } p = (s^a \textbf{ mod } p)^b \textbf{ mod } p = \alpha^b \textbf{ mod } p = k'$

Beide haben so den **gleichen Wert** erhalten, der nun in einem symmetrischen Verfahren verwendet werden kann.

Das Verfahren ist deshalb sicher, weil es extrem schwer ist,
von $\alpha = s^a \textbf{ mod } p$ auf a zu schließen, also den **diskreten Logarithmus** von α zu berechnen.

Die Komplexität dieses Problems ist vergleichbar mit der des Faktorisierens.

Beispiel: $p = 11, s = 4, 4^a = 3 \textbf{ mod } 11$, bestimme a.

Kapitel 6

Vermischtes

Dieses letzte Kapitel enthält noch einige interessante *bits and pieces*, die vielleicht in einer späteren Veranstaltung noch ausführlicher behandelt werden.

6.1 Das ElGamal -Schema

Mit diskreten Logarithmen lassen sich nicht nur Schlüssel austauschen, man kann sie auch als Grundlage für asymmetrische Verfahren verwenden. Ein solches Verfahren wurde von *Taher ElGamal* 1985 vorgestellt.

6.1.1 Verschlüsselung mit ElGamal

Die Teilnehmer *Alice* und *Bob* einigen sich wieder auf eine Primzahl p und eine natürliche Zahl $s < p$.

Als private Schlüssel wählen sie Zahlen $1 < a, b < p$.

Als **öffentliche Schlüssel** dienen $\alpha = s^a \textbf{ mod } p$ und $\beta = s^b \textbf{ mod } p$.

Zum Verschlüsseln wählt *Alice* eine Zahl a_1 mit $ggt(a_1, p-1) = 1$ und berechnet daraus mit *Bobs* öffentlichen Schlüssel β $k = \beta^{a_1} \textbf{ mod } p$. Mit diesem Schlüssel wird die Nachricht m mit dem Verfahren f verschlüsselt: $c = f_k(m)$. Dann sendet sie c und $s^{a_1} \textbf{ mod } p$ an *Bob*.

Bob berechnet mit seinem privaten Schlüssel b

$k = (s^{a_1})^b \textbf{ mod } p = (s^{a_1} \textbf{ mod } p)^b \textbf{ mod } p = s^{b*a_1} \textbf{ mod } p =$

$= (s^b \textbf{ mod } p)^{a_1} \textbf{ mod } p = \beta^{a_1} \textbf{ mod } p$

und entschlüsselt damit die Nachricht von *Alice*.

6.1.2 Digitale Signaturen mit ElGamal

Man kann das Verfahren von *ElGamal* auch verwenden, um eine Unterschrift für eine Nachricht zu erzeugen:

Sei p eine Primzahl und $g < p \in \mathcal{N}$ wie gehabt.

Der Teilnehmer T besitze den geheimen Schlüssel $t < p$ und den öffentlichen Schlüssel $\tau = g^t \ \mathbf{mod}\ p$.

Er wählt $r \in \mathcal{N}, ggt(r, p-1) = 1$

und berechnet $k = g^r \ \mathbf{mod}\ p$.

T berechnet die **modulare Inverse** zu r:

$r^{-1} * r \ \mathbf{mod}\ (p-1) = 1$.

Das geschieht mit dem **erweiterten Euklidischen Algorithmus** .

Dann setzt T $s = ((m - t * k) * r^{-1}) \ \mathbf{mod}\ (p-1)$.

Damit gilt: $(t * k + r * s) \ \mathbf{mod}\ (p-1) = m$.

Das Paar (k, s) ist dann die Signatur.

Bei einer unveränderten Nachricht m muss dann gelten:

$g^m \textbf{ mod } p = \tau^k * k^s \textbf{ mod } p$

Denn: $t * k + r * s = x * (p-1) + m$

$(\tau^k * k^s) \textbf{ mod } p = ((g^t \textbf{ mod } p)^k * (g^r \textbf{ mod } p)^s) \textbf{ mod } p =$

$g^{(t*k+r*s)} \textbf{ mod } p = g^{(x*(p-1)+m)} \textbf{ mod } p =$

$(g^m * (g^x \textbf{ mod } p)^{(p-1)}) \textbf{ mod } p = (g^m * 1) \textbf{ mod } p = g^m \textbf{ mod } p$

Denn $(g^x \textbf{ mod } p) < p$. Daher gilt mit dem Satz von *Fermat*

$(g^x \textbf{ mod } p)^{(p-1)} \textbf{ mod } p = 1$

6.2 Shamirs No-Key Algorithmus

Shamirs No-Key Algorithmus, der auch als **Massey-Omura-Schema** bekannt ist, erlaubt den Austausch verschlüsselter Nachrichten, **OHNE** dass sich die Kommunikationspartner auf einen Schlüssel einigen müssen.

Das hört sich verblüffend an, ist aber auch wieder "ganz einfach" - man muss nur drauf kommen.

Alice und *Bob* einigen sich auf eine große Primzahl p, die öffentlich sein darf.

Dann wählen sie jeder für sich zwei natürliche Zahlen e_a, d_a bzw. e_b, d_b für die gilt $(e_a * d_a)$ **mod** $(p-1) = 1$ und $(e_b * d_b)$ **mod** $(p-1) = 1$. Diese Zahlen behält jeder für sich!

Jetzt erinnern wir uns, dass nach dem *Satz von Euler* gilt:

$$\forall m \in \mathcal{N} : m < p \Rightarrow m^{e_a * d_a} \textbf{ mod } p = m$$

Dies sieht man leicht ein, denn:

$$e_a * d_a = x * (p-1) + 1$$

$$m^{e_a * d_a} \textbf{ mod } p = m^{x*(p-1)+1} \textbf{ mod } p = (m * m^{(p-1)*x}) \textbf{ mod } p$$

$$(m * (m^{(p-1)})^x) \textbf{ mod } p = (m * 1) \textbf{ mod } p$$

Wenn *Alice* *Bob* nun eine Nachricht schicken will, packt sie diese in einen Koffer und verschließt ihn mit einem Vorhängeschloss, das nur sie öffnen kann. Wenn *Bob* den Koffer erhält, befestigt er ein zweites Schloss, das nur er öffnen kann und schickt den Koffer an *Alice*. Die entfernt ihr Schloss und schickt den Koffer zurück an *Bob*. Dieser entfernt sein Schloss und kann die Nachricht entnehmen.

Elektronisch geht das so:

Alice verschlüsselt m und sendet $m' = m^{e_a}$ **mod** p an *Bob*.

Bob sendet $m'' = m'^{e_b}$ **mod** p an *Alice*.

Alice sendet m''^{d_a} **mod** $p = m^{e_a * e_b * d_a}$ **mod** $p = m^{e_b}$ **mod** p an *Bob*.

Der potenziert noch einmal mit d_b und kann die Nachricht lesen.

6.3 Zero-Knowledge Verfahren

Üblicherweise weist man sich gegenüber einem Computer durch die Eingabe eines **Passworts** aus. Dieses Vorgehen ist nicht ohne Probleme, da Passwortdateien ausgespäht oder Passwörter geraten werden können.

Ein Passwort kann man als **Geheimnis** betrachten, dessen Besitz zur **Authentisierung** gegenüber dem Rechner dient. Normalerweise muss man sein Geheimnis dem Rechner verraten.

Besser wäre ein Verfahren, das es erlaubt, den Rechner davon zu überzeugen im Besitz eines Geheimnisses zu sein, **ohne** dieses preiszugeben. Dass dies möglich ist, zeigt der nächste Abschnitt.

6.3.1 Interaktive Beweise

Um 1500 hatte *Tartaglia* eine Formel zum Lösen von Gleichungssystemen dritten Grades entwickelt:

$$x^3 + ax^2 + bx + c = 0$$

kann man durch **Substitution** auf die Form

$$x^3 + px + q = 0$$

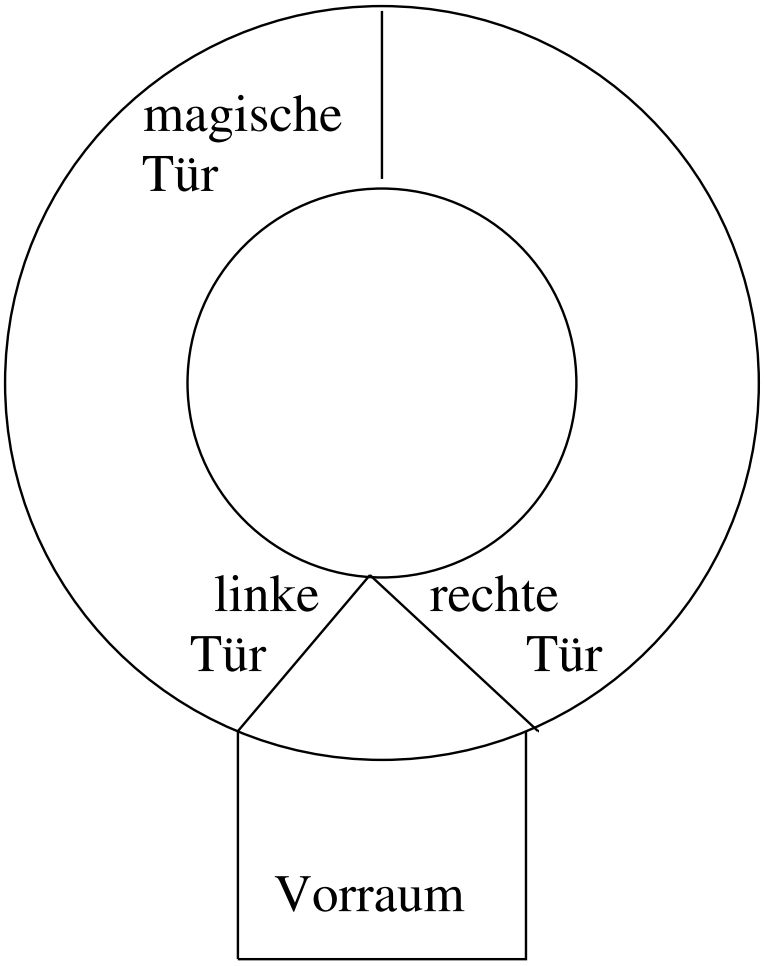
bringen. *Tartaglia* fand hierfür eine Lösung in der Form:

$$x = \sqrt[3]{\sqrt{(p/3)^3 + (q/2)^2} + q/2} + \sqrt[3]{\sqrt{(p/3)^3 + (q/2)^2} - q/2}$$

In einem Wettstreit mit dem italienischen Rechenmeister *Fior* legte *Fior* *Tartaglia* 30 Aufgaben vor, die dieser löste, ohne zu verraten wie. *Fior* war dann von *Tartaglias* Behauptung überzeugt, da er die Lösungen ja leicht nachrechnen konnte.

6.3.2 Die magische Tür

Im folgenden Beispiel kann *Alice Bob* überzeugen, ein Geheimnis zu kennen, ohne dieses zu verraten. Das Bild zeigt den Grundriss eines Gebäudes, das durch einen Vorraum betreten wird. Das Geheimnis ist die Zahlenkombination, die die magische Tür öffnet. Nur dann kann man durch die linke Tür in das Innere gehen und durch die rechte wieder herauskommen.



Alice behauptet, die Kombination zu kennen. Um *Bob* zu überzeugen betritt sie den Vorraum, schließt die Eingangstür und wählt die linke oder die rechte Tür. Wenn sie dahinter verschwunden ist, betritt *Bob* den Vorraum und wählt die Tür aus der *Alice* wieder auftauchen soll.

Kennt *Alice* das Geheimnis, so kann sie **immer** aus der richtigen Tür treten. Kennt sie es nicht, hat sie nur eine 50%-Chance, aus der richtigen Tür zu kommen.

Wird dieses Spiel lange genug gespielt, ist *Bob* mit beliebiger Sicherheit überzeugt, dass *Alice* die Kombination kennt.

6.3.3 Das Fiat-Shamir-Protokoll

In der Praxis ist das **Fiat-Shamir-Protokoll** wohl das bekannteste Verfahren. Es beruht auf der praktischen Unmöglichkeit in $\mathcal{Z}_n^*$ für sehr große n Quadratwurzeln zu berechen. Unter dem Namen *Videocrypt* wird es in **Pay-TV-Systemen** eingesetzt.

Der Algorithmus besteht aus der **Schlüsselerzeugungsphase** und der **Anwendungsphase**.

In der **Schlüsselerzeugungsphase** erzeugt *Alice* zunächst zwei große Primzahlen p, q und bildet $n = pq$. n ist öffentlich, p, q sind geheimzuhalten. Nun wählt *Alice* ihr **Geheimnis**, eine Zahl $s \in \mathcal{N}, 1 < s < n$ und berechnet $v = s^2 \textbf{ mod } n$. s muss geheim bleiben, v ist öffentlich und dient zum **V**erifizieren.

In der **Anwendungsphase** wählt *Alice* ein $r \in \mathcal{Z}_n^*$ und sendet $x = r^2 \textbf{ mod } n$ an *Bob*.

Dieser wählt ein zufälliges Bit b und sendet es an *Alice*.

Falls $b = 0$ sendet *Alice* $y = r$ an *Bob*.

Falls $b = 1$ sendet sie $y = (rs) \textbf{ mod } n$ an *Bob*.

Bob verifiziert, ob für $b = 0$ $y^2 \textbf{ mod } n = x$ ist,

für $b = 1$ prüft er $y^2 \textbf{ mod } n = xv \textbf{ mod } n$

Wenn *Alice* wirklich den Wert s kennt, kann sie *Bob*s Fragen immer richtig beantworten, und es gilt:

$$y^2 \textbf{ mod } n = (rs^b)^2 \textbf{ mod } n = (r^2 s^{2b}) \textbf{ mod } n = r^2 v^b \textbf{ mod } n = xv^b \textbf{ mod } n$$

Falls sich eine dritte Person als *Alice* ausgibt, die das Geheimnis, also eine Quadratwurzel von v, nicht kennt, kann diese immer höchstens **eine** der Fragen beantworten.

Könnte sie nämlich **immer** richtig mit $y_0^2 = x$ bzw. $y_1^2 = xv$ antworten, **ohne** das Geheimnis zu kennen, hätte sie auch eine Wurzel von v, da $(y_1/y_0)^2 = (xv)/x$. Und diese Wurzel kennt sie ja nach Voraussetzung nicht.

Sie kann aber auch immer **mindestens** mit der Wahrscheinlichkeit 1/2 erfolgreich betrügen:

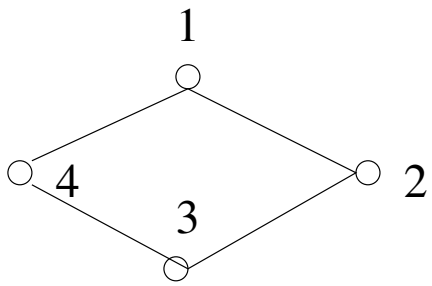
Wenn sie vermutet, dass *Bob* $b = 0$ wählt, so setzt sie $x = r^2$ und $y = r$, vermutet sie, dass als nächstes $b = 1$ kommt, so wählt sie $x = r^2 v^{-1}$ **mod** n und $y = r$ und damit ergibt sich

$$y^2 \textbf{ mod } n = r^2 \textbf{ mod } n = r^2 v^{-1} v \textbf{ mod } n = xv \textbf{ mod } n$$

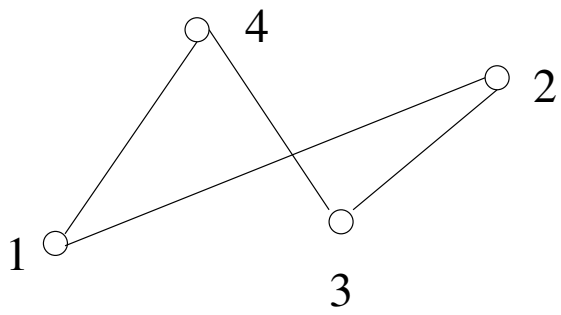
Also beträgt die Wahrscheinlichkeit **genau** 50%. Spielt *Bob* dieses Spiel mit genügend vielen Bits, so kann er sich mit beliebiger Sicherheit von der Identität von *Alice* überzeugen.

6.3.4 Isomorphie von Graphen

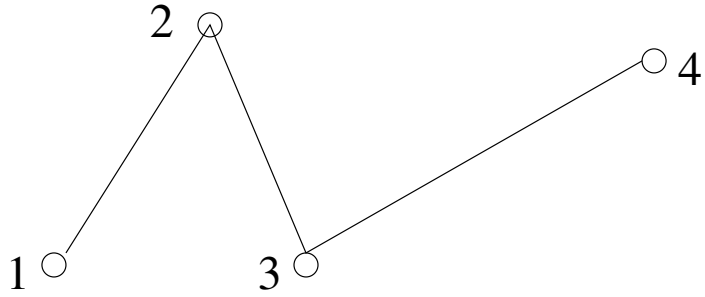
Ein weiteres Zero-Knowledge-Protokoll basiert auf der Tatsache, dass es praktisch unmöglich ist, einen Isomorphismus zwischen zwei **isomorphen** Graphen durch reines Probieren zu finden, wenn diese Graphen genügend groß sind. Bei einem Graphen mit 12 Knoten muss man schon $12! \approx 479001600$ Permutationen ausprobieren.



G1



G2



G3

Die Graphen G1 und G2 sind offenbar isomorph, G1 und G3 aber nicht.

Zur Durchführung des Protokolls wählt *Alice* nun zwei große, isomorphe Graphen G_0 und G_1. Der Isomorphismus σ ist ihr Geheimnis, die beiden Graphen sind öffentlich bekannt.

Alice sendet nun einen Graphen H an *Bob*, der zu G_0 oder G_1 isomorph ist: $H = \tau(G_0)$ oder $H = \tau(G_1)$.

Bob fordert *Alice* nun auf, einen Isomorphismus c_0 zwischen H und G_0 oder einen Isomorphismus c_1 zwischen H und G_1 anzugeben.

Wenn zum Beispiel $H = \tau(G_0)$ gilt, sendet *Alice* die Permutation τ an *Bob*, wenn er c_0 gewählt hat. Gilt $H = \tau(G_1)$ sendet *Alice* $\sigma^{-1}\tau$ an *Bob*.

Wenn *Alice* und *Bob* dieses Spiel genügend oft wiederholen, kann sich *Bob* mit beliebiger Sicherheit von *Alices* Identität überzeugen.

6.4 Multi Party Communication

Zum guten Schluss folgen noch einige Protokolle, die eine korrekte Kommunikation zwischen zwei oder mehr Personen erlauben.

6.4.1 Secret Sharing Schemes

Häufig liegt die Situation vor, dass eine bestimmte Aktivität nur von mehreren Personen **zusammen** ausgeführt werden darf:

- Verträge benötigen zwei Unterschriften
- nur zwei Abteilungsleiter dürfen einen Tresor öffnen
- Atomraketen dürfen nur von zwei Offizieren zusammen zum Abschuß freigegeben werden (man sollte eher die Offiziere, ... - na egal)

Threshold-Verfahren verteilen ein Geheimnis S auf n **Teilgeheimnisse** $S_1, S_2, ..., S_n$.

$S = (0, s)$ sei zum Beispiel ein Punkt auf der y-Achse eines zweidimensionalen Koordinatensystems.

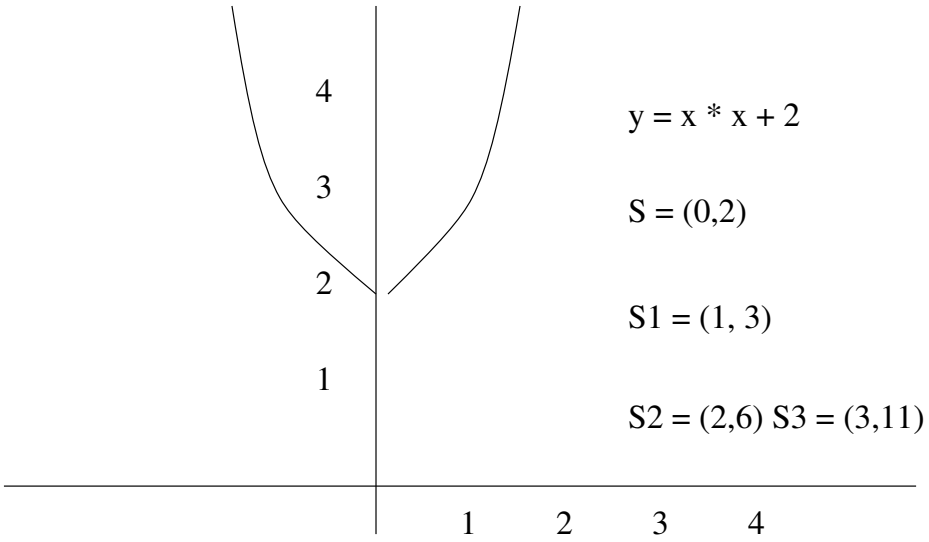
Man wähle nun zufällig zwei Zahlen a und b und betrachte die Funktion

$$f(x) = ax^2 + bx + s$$

Dann kann man S aus drei Punkten S_1, S_2, S_3 berechnen - mit weniger Punkten geht es aber nicht. Wir verwenden hierzu die **Lagrange-Interpolation** , die es erlaubt, aus t Punkten ein Polynom vom Grad $t-1$ zu bestimmen:

$$f(x) = \sum_{i=1}^{t} \frac{(x-a_1)...(x-a_{i-1})(x-a_{i+1})...(x-a_t)}{(a_i-a_1)...(a_i-a_{i-1})(a_i-a_{i+1})...(a_i-a_t)} b_i$$

Beispiel:



$$\frac{(x-2)(x-3)}{(1-2)(1-3)}3 + \frac{(x-1)(x-3)}{(2-1)(2-3)}6 + \frac{(x-1)(x-2)}{(3-1)(3-2)}11 =$$

$$\frac{x^2-5x+6}{2}3 + \frac{x^2-4x+3}{-1}6 + \frac{x^2-3x+2}{2}11 =$$

$$\frac{3x^2 -^5 x + 11x^2 - 33x + 22}{2} + \frac{-12x^2+48x-36}{2} = \frac{2x^2+4}{2} = x^2+2$$

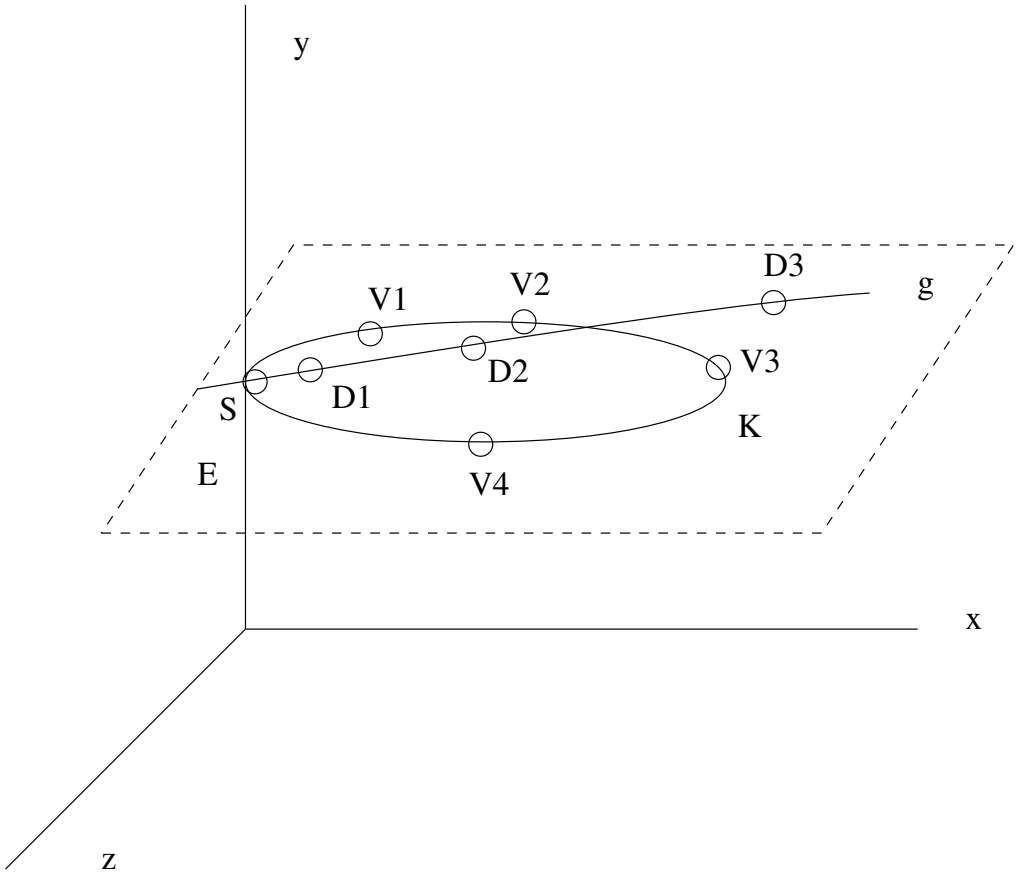
6.4.2 2 Direktoren und 3 Vizes

Nehmen wir einmal einmal an, ein Tresor darf nur von

- 2 Direktoren
- 3 Vizedirektoren oder
- 1 Direktor und zwei Vizedirektoren

geöffnet werden.

Als Geheimnis wählen wir einen Punkt S auf der z-Achse eines dreidimensionalen Raumes. Wir legen zufällig eine Ebene E durch S, die die z-Achse nicht enthält. Ferner legen wir durch S eine Gerade g und einen Kreis K.



Jeder Direktor kennt einen der Punkte D_i, jeder Vizedirektor einen der Punkte V_j.

Ein Direktor allein, kann den Punkt S nicht bestimmen, bei zwei Direktoren ist aber g leicht auszurechnen.

Zwei Vizedirektoren können aus Ihren beiden Punkten die Orientierung der Ebene nicht berechnen.

Findet sich aber noch ein Direktor oder ein Vize, so kann S bestimmt werden.

Es ist noch zu beachten, dass jede Gerade V_iV_j nicht durch einen der Punkte D_k gehen darf.

6.4.3 Durchschnittsgehalt

In manchen Firmen ist die Höhe der Gehälter der Mitarbeiter ein gut gehütetes Geheimnis. Wenn nun *Alice*, *Bob* und *Carol* wissen wollen, wie hoch ihr **Durchschnittsgehalt** ist, ohne dabei die exakte Gehaltshöhe zu verraten, so können sie dies folgendermaßen erreichen:

Alice wählt eine Zufallszahl und addiert diese zu ihrem Gehalt. Die Summe gibt sie flüsternd an *Bob* weiter, der sein Gehalt addiert und den neuen Wert vertraulich *Carol* mitteilt. Diese addiert ihr Gehalt und sagt die Endsumme *Alice*.

Alice subtrahiert die Zufallszahl und gibt dann den Durchschnittswert bekannt.

Da hier **vier** Unbekannte im Spiel sind, aber jeder Teilnehmer nur **drei** von ihnen kennt, kann niemand das Gehalt der anderen Kollegen ausrechnen.

Dieses einfache Protokoll hat aber den Nachteil, dass sich zwei Teilnehmer verbünden können. Außerdem funktioniert es nur, wenn jeder Teilnehmer die Wahrheit sagt.

6.4.4 Wer verdient mehr?

Wenn *Alice* und *Bob* nicht nur ihr Durchschnittsgehalt wissen wollen, sondern auch, wer von ihnen denn nun besser bezahlt wird, dann können sie das folgende Protokoll verwenden, das auch noch die Vertraulichkeit der Gehaltshöhe sichert:

Die beiden einigen sich, dass die maximale Gehaltshöhe DM 10.000 beträgt, und dass sie auf DM 100,00 genau rechnen wollen.

Mit diesen Einschränkungen können sie ihr Gehalt als ganze Zahlen a, b $0 <= a, b <= 100$ darstellen.

Alice wählt eine Zufallszahl x und berechnet mit *Bobs* öffentlichem Schlüssel $c = E_B(x)$. Dann sendet sie an ihn $d = c - a$.

Bob führt nun für alle Zahlen $0 <= i <= 100$ die Entschlüsselungsoperation $y_i = D_B(d + i)$ aus. Unter diesen 101 Werten findet sich auch der Wert x. *Bob* weiß aber nicht, welches y_i diesen Wert repräsentiert.

Nun verwendet *Bob* eine Einwegfunktion f und berechnet $z_i = f(y_i)$. Wenn sich zwei benachbarte Werte z_j, z_{j+1} nur um 1 unterscheiden, muss *Alice* ein neues x suchen.

Dann sendet *Bob* die Folge

$$z_0, z_1, ...z_b, z_{b+1} + 1, z_{b+2} + 1, ..., z_{100} + 1$$

in beliebiger Reihenfolge an *Alice*.

Alice berechnet $f(x)$ und $f(x)+1$. Findet sie in der Folge den Wert $f(x)$, so verdient *Bob* mehr, findet sie $f(x) + 1$, dann liegt sie vorn.

Würde *Alice* die y_i kennen, dann wüsste sie, ab wann $y_i + 1$ auftritt, und könnte so *Bobs* Gehalt bestimmen.

6.4.5 Skat am Telefon

Die Herren Rivest , Shamir und Adleman , die uns schon mehrfach begegnet sind, haben ein Protokoll erfunden, das es ermöglicht, am Telefon Skat zu spielen. Es verwendet Shamirs *No-Key-Algorithmus* . Wir betrachten eine Variante, mit der man **Skat** am Telefon spielen kann.

Das Problem ist hier, die Karten korrekt zu mischen und unter die Spieler zu verteilen.

Die drei Spieler A, B, C einigen sich auf eine große Primzahl p und berechnen

$$aa' = bb' = cc' = 1 \textbf{ mod } (p-1)$$

Die n Karten werden als Paare (i, x_i) dargestellt, wobei i die Position im Kartenstapel und x_i den Wert darstellt.

Unser Skatblatt hat dann die Gestalt: $\{(1, x_1), ..., (32, x_{32})\}$.

A wendet nun eine **Permutation** α an und berechnet

$$\{(\alpha(1), x_1^a \textbf{ mod } p), ..., (\alpha(32), x_{32}^a \textbf{ mod } p)\}$$

Wir sortieren die Karten nach der ersten Komponente:

$$\{(1, x_k^a \textbf{ mod } p), ...\}, k = \alpha^{-1}(1)$$

B berechnet nun

$$\{(\beta(\alpha(1)), (x_1^a)^b \textbf{ mod } p), ..., (\beta(\alpha(32)), (x_{32}^a)^b \textbf{ mod } p)\}$$

und C

$$\{\gamma((\beta(\alpha(1))), ((x_1^a)^b)^c \textbf{ mod } p), ..., (\gamma(\beta(\alpha(32))), ((x_{32}^a)^b)^c \textbf{ mod } p)\}$$

Damit haben wir das Mischen erledigt, jetzt werden die Karten verteilt: dazu nimmt A die oberste Karte $(1, x_i^{abc}) \textbf{ mod } p)$ mit $\gamma(\beta(\alpha(i)) = 1$ und berechnet

$$(x_i^{abc})^{a'} = x_i^{bc} \textbf{ mod } p$$

.
Diesen Wert gibt er an B, der seinen Schlüssel entfernt:

$$(x_i^{bc})^{b'} = x_i^c \textbf{ mod } p$$

und den Wert an C weitergibt. C entfernt ebenfalls seinen Schlüssel.

Die Karten an A und B werden analog verteilt. Nach dem Spiel werden die Werte a, b, c, a', b', c' bekanntgegeben, damit man überprüfen kann, dass nicht gemogelt wurde.

Kapitel 7

Mathematische Grundlagen

In diesem Abschnitt versammeln wir einige wichtige mathematische Grundbegriffe.

7.1 Entropie

Die Ausführungen über die Entropie sind der Hilfe von CrypTool entnommen.
Die Entropie eines Dokuments ist eine Kennzahl für dessen Informationsgehalt. Die Entropie wird in (Bit pro Zeichen) bit/char gemessen.

7.1.1 Informationsgehalt einer Quelle

Die Zeichen in einer Datei können als Nachrichtenquelle im informationstheoretischen Sinne angesehen werden. Für die Berechnung des Informationsgehaltes betrachtet man die Wahrscheinlichkeitsverteilung dieser Quelle. Dabei geht man davon aus, dass die einzelnen Nachrichten (Zeichen des Dokuments / der Datei) stochastisch unabhängig voneinander sind und von der Quelle mit konstanter Wahrscheinlichkeit ausgestrahlt werden.

7.1.2 Informationsgehalt einer Nachricht $M[i]$

Der Informationsgehalt einer Nachricht $M[i]$ ist definiert durch:

$$\boxed{\text{Informationsgehalt}(M[i]) := log(1/p[i]) = -log(p[i])}$$

.
Dabei ist $p[i]$ die Wahrscheinlichkeit, mit der die Nachricht $M[i]$ von der Nachrichtenquelle ausgestrahlt wird. Mit log ist (wie auch im folgenden) der Logarithmus zur Basis 2 gemeint.
Der Informationsgehalt hängt damit ausschließlich von der Wahrscheinlichkeitsverteilung ab, mit der die Quelle die Nachrichten erzeugt.
Der semantische Inhalt der Nachricht geht nicht in die Berechnung ein.

Da der Informationsgehalt einer seltenen Nachricht höher als der einer häufigen Nachricht ist, wird in der Definition der Kehrwert der Wahrscheinlichkeit verwendet.
Ferner ist der Informationsgehalt zweier unabhängig voneinander ausgewählter Nachrichten gleich der Summe der Informationsgehalte der einzelnen Nachrichten.

7.1.3 Berechnung der Entropie

Mit Hilfe des Informationsgehaltes der einzelnen Nachrichten kann nun die mittlere Information berechnet werden, die eine Quelle mit einer gegebenen Verteilung liefert. Für die Durchschnittsbildung werden die einzelnen Nachrichten mit der Wahrscheinlichkeit ihres Auftretens gewichtet.

$$\boxed{\text{Entropie}(p[1], p[2], ..., p[r]) := -\sum_{i=1}^{r} p[i] * log(p[i])}$$

Die Entropie einer Quelle bezeichnet somit die sie charakterisierende Verteilung. Sie misst die Information, die man durch Beobachten der Quelle im Mittel gewinnen kann, oder umgekehrt die Unbestimmtheit, die über die erzeugten Nachrichten herrscht, wenn man die Quelle nicht beobachten kann.

7.1.4 Anschauliche Beschreibung der Entropie

Die Entropie gibt die Unsicherheit als Anzahl der notwendigen Ja/Nein-Fragen zur Klärung einer Nachricht oder eines Zeichens an. Hat ein Zeichen eine sehr hohe Auftrittswahrscheinlichkeit, so hat es einen geringen Informationsgehalt. Dies entspricht etwa einem Gesprächspartner, der re-

gelmäßig mit "ja" antwortet. Diese Antwort lässt auch keine Rückschlüsse auf Verständnis oder Aufmerksamkeit zu. Antworten, die sehr selten auftreten, haben einen hohen Informationsgehalt.

7.1.5 Extremwerte der Entropie

Für Dokumente, die ausschließlich Großbuchstaben enthalten, ist die Entropie mindestens 0 bit/char (bei einem Dokument, das nur aus einem Zeichen besteht) und höchstens log(26) bit/char = 4,700440 bit/char (bei einem Dokument, in dem alle 26 Zeichen gleich oft vorkommen).
Für Dokumente, die jedes Zeichen des Zeichensatzes (0 bis 255) enthalten können, ist die Entropie mindestens 0 bit/char (bei einem Dokument, das nur aus einem Zeichen besteht) und höchstens log(256) bit/char = 8 bit/char (bei einem Dokument, in dem alle 256 Zeichen gleich oft vorkommen).

7.2 Mengen und Logik

In diesem Abschnitt wollen wir kurz einige Begriffe zusammenstellen, die wir für die folgenden Überlegungen benötigen.
Wir folgen hier weitgehend der Darstellung in [16] und [17].

7.2.1 Mengen

Aufzählung:

$$M = \{a_1, a_2, a_3, ..., a_n\}$$
$$M = \{a_1, a_2, a_3, ...\}$$

Beschreibung:

$$M = \{x \mid p(x)\}$$
$$M = \{x \mid 2 \leq x \leq 200 \text{ und ist Primzahl}\}$$

Elemente einer Menge:

$$a \in M, a \notin M$$

Kardinalität von M:

Anzahl der Elemente in M: $|M|, |\mathcal{N}_0 = \infty|$

Russelsche Antinomie:

$M = \{A \mid A \notin A\}$
oder die Geschichte von der Schlange, die alle Schlangen in den Schwanz beißt, die sich nicht selbst in den Schwanz beißen.

7.2.2 Aussagenlogik

Operatorsymbole:

$$O = \{\underline{0}, \underline{1}, \neg, \wedge, \vee, (,)\}$$

Variablen:

$$V = \{x_1, x_2, x_3, ...\}$$

Formeln $\mathcal{A}$:

- $\underline{0}, \underline{1} \in \mathcal{A}$
- alle Variablen sind Formeln: $x_i \in V$, dann auch $x_i \in \mathcal{A}$
- $\alpha, \beta \in \mathcal{A}$ dann gilt auch $(\alpha \wedge \beta), (\alpha \vee \beta), \neg\alpha \in \mathcal{A}$

Semantik von Formeln:

0 steht für **falsch**, 1 steht für **wahr**
$B = \{0, 1\}, 0 < 1, 1 - 1 = 0, 1 - 0 = 1$

Interpretation I:

$I(\underline{0}) = 0, I(\underline{1}) = 1$

Sei ν eine Formel, V_ν die Menge aller Variablen in ν.

$I_\nu : V_\nu \longrightarrow B$ eine **Belegung**,
die jeder Variablen $x \in V_\nu$ einen Wert aus B zuweist: $I_\nu(x) \in B$

Wahrheitswert einer Formel ν

$I^*(\underline{0}) = I(\underline{0}), I^*(\underline{1}) = I(\underline{1})$

$I^*(\nu) = I_\nu(x)$, falls $\nu = x \in V_\nu$

Seien nun α, β Formeln

$I^*(\neg\alpha) = 1 - I^*(\alpha)$

$I^*(\alpha \wedge \beta) = min(I^*(\alpha), I^*(\beta))$

$I^*(\alpha \vee \beta) = max(I^*(\alpha), I^*(\beta))$

Beispiel:

$\nu = (((p \vee (q \wedge r)) \wedge \neg(q \vee \neg r)) \vee \underline{0})$

$V_\nu = \{p, q, r\}, I_\nu(p) = 1, I_\nu(q) = 0, I_\nu(r) = 1$

Weitere Operationen

Subjunktion:

$\alpha \rightarrow \beta : I^*(\alpha \rightarrow \beta) = I^*(\neg\alpha \vee \beta)$

Bijunktion:

$\alpha \leftrightarrow \beta : I^*(\alpha \leftrightarrow \beta) = I^*((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$

Exklusiv Oder:

$\alpha \oplus \beta : I^*(\alpha \oplus \beta) = I^*((\alpha \wedge \neg\beta) \vee (\neg\alpha \wedge \beta))$

Allquantor "für alle": $\forall$

$\forall x : p(x)$

Existenzquantor "es existiert": $\exists$

$\exists x : p(x)$

Definition: Eine Formel α ist **erfüllbar**, wenn es eine Belegung I_α gibt, mit $I^*(\alpha) = 1$.

Eine Formel α ist eine **Tautologie**, wenn für **jede** Belegung I_α gilt: $I^*(\alpha) = 1$.

Eine Formel α ist eine **Kontradiktion**, wenn für **jede** Belegung I_α gilt: $I^*(\alpha) = 0$.

Gilt für Formeln $\alpha_1, \alpha_2, ..., \alpha_n, \beta$,
dass die Subjunktion $((\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n) \rightarrow \beta)$
eine Tautologie ist, dann schreiben wir
$((\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n) \Rightarrow \beta)$
und bezeichnen die Subjunktion als **Implikation**

Abschwächung der Nachbedingung: $\alpha \Rightarrow (\alpha \vee \beta)$

Verschärfung der Vorbedingung: $(\alpha \wedge \beta) \Rightarrow \alpha$

Kettenschluss: $((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \gamma)) \Rightarrow (\alpha \rightarrow \gamma)$

7.3 Beweismethoden

7.3.1 Direkter Beweis

Prinzip:

$\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 ... \Rightarrow \gamma_n \Rightarrow \beta$

Beispiel:

Ist eine natürliche Zahl durch 2 und 3 teilbar, so ist sie auch durch 6 teilbar!

$$\frac{x}{2} \in \mathcal{N}_0 \wedge \frac{x}{3} \in \mathcal{N}_0 \Rightarrow \frac{x}{6} \in \mathcal{N}_0$$

$$\frac{x}{2} \in \mathcal{N}_0 \wedge \frac{x}{3} \in \mathcal{N}_0 \Rightarrow (\exists y \in \mathcal{N}_0 : (x = 2y) \wedge \exists z \in \mathcal{N}_0 : (x = 3z))$$

$$\Rightarrow$$

$$\exists y, z \in \mathcal{N}_0 : (2y = 3z)$$

$$\Rightarrow$$

$$\exists k \in \mathcal{N}_0 : (z = 2k)$$

$$\Rightarrow$$

$$\exists k, z \in \mathcal{N}_0 : ((x = 3z) \wedge (z = 2k))$$

$$\Rightarrow$$

$$\exists k \in \mathcal{N}_0 : (x = 3 * 2k)$$

$$\Rightarrow$$

$$\frac{x}{6} \in \mathcal{N}_0$$

7.3.2 Indirekter Beweis

Prinzip:
$(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\beta \Rightarrow \neg\alpha)$

Beispiel:
Wenn die letzten beiden Ziffern einer Zahl durch 4 teilbar sind, so ist die ganze Zahl durch 4 teilbar.

$$\alpha = (x \in \mathcal{N}_0^{99}) \wedge (\frac{x}{4} \in \mathcal{N}_0) \wedge (y \in \mathcal{N}_0)$$

$$\beta = (x \in \mathcal{N}_0^{99}) \wedge (y \in \mathcal{N}_0) \wedge (\frac{100y + x}{4} \in \mathcal{N}_0)$$

$$\neg\beta \Rightarrow \neg\alpha$$

$$\neg\beta = (x \notin \mathcal{N}_0^{99}) \vee (y \notin \mathcal{N}_0) \vee (\frac{100y + x}{4} \notin \mathcal{N}_0)$$

$$\neg\alpha = (x \notin \mathcal{N}_0^{99}) \vee (\frac{x}{4} \notin \mathcal{N}_0) \vee (y \notin \mathcal{N}_0)$$

bleibt zu zeigen

$$(\frac{100y + x}{4} \notin \mathcal{N}_0) \Rightarrow (\frac{x}{4} \notin \mathcal{N}_0)$$

das geht direkt

$$(\frac{100y + x}{4} = 25y + \frac{x}{4} \notin \mathcal{N}_0) \Rightarrow \frac{x}{4} \notin \mathcal{N}_0$$

da ja $25y$ immer aus $\mathcal{N}_0$

7.3.3 Widerspruchsbeweis

Prinzip:

$$(\alpha \Rightarrow \beta) \Leftrightarrow ((\alpha \wedge \neg\beta) \Rightarrow \neg\alpha)$$

Beispiel:

Die Quadratwurzel aus 2 ist irrational.

$$\alpha = \exists\, p, q \in \mathcal{N}_0 : (ggt(p,q) = 1)$$

$$\beta = \sqrt{2} \neq \frac{p}{q}$$

$$\alpha \wedge \neg\beta = \exists\, p, q \in \mathcal{N}_0 : (ggt(p,q) = 1 \wedge \sqrt{2} = \frac{p}{q})$$

$$\sqrt{2} = \frac{p}{q} \Rightarrow 2 = \frac{p^2}{q^2} \Rightarrow 2q^2 = p^2 \Rightarrow \exists k \in \mathcal{N}_0 : (p = 2k)$$

$$\Rightarrow p^2 = 4k^2 \Rightarrow 2q^2 = 4k^2 \Rightarrow q^2 = 2k^2 \Rightarrow$$

2 teilt p und 2 teilt q. Widerspruch zu $\alpha : ggt(p,q) = 1$

7.3.4 Zirkelschluss

Prinzip:

$\alpha_1 \Leftrightarrow \alpha_2 \Leftrightarrow ... \Leftrightarrow \alpha_n$

$\Longleftrightarrow$

$\alpha_1 \Rightarrow \alpha_2 \Rightarrow ... \Rightarrow \alpha_n \Rightarrow \alpha_1$

7.3.5 Beweis durch vollständige Induktion

Prinzip:
Sei A eine Aussage, die von einer natürlichen Zahl n abhängt: $A(n)$.
Wenn wir wissen, dass die **Induktionsvoraussetzung:** $A(1)$ gilt
und **(Induktionsschritt)** wenn man für ein beliebiges $n \in \mathcal{N}_0$ zeigen kann, dass $A(n) \Rightarrow A(N+1)$,
dann gilt $A(n)$ für alle $n \geq 1$.
Man kann auch bei $n = 0$ oder einem $n > 0$ anfangen!
Beispiel:
$A(n) : \forall\ n \geq 1 : (\sum_{i=0}^{n} i = \frac{n(n+1)}{2})$

$$A(1) : 0 + 1 = \frac{1(1+1)}{2}$$

$$A(n+1) = \sum_{i=0}^{n+1} i = \sum_{i=0}^{n} i + n + 1 = \frac{n(n+1)}{2} + n + 1 = \frac{n(n+1)}{2} + \frac{2(n+1)}{2} =$$

$$\frac{n^2 + n + 2n + 2}{2} = \frac{(n+2)(n+1)}{2}$$

Weitere Beispiele:

1. $\sum_{i=1}^{n}(2i-1) = n^2$

2. $\sum_{i=1}^{n} i^3 = (\sum_{i=1}^{n} i)^2$

3. $\sum_{k=0}^{n}(aq^k) = a\frac{1-q^{n+1}}{1-q}$

4. Bernoulli-Ungleichung: $(1+x)^n \geq 1+nx$

5. $\forall\ n \geq 4 \in \mathcal{N}_0 : (n! \geq 2^n)$

6. $\sum_{i=0}^{n} 2^i = 2^i - 1$

7. **Fibonacci-Zahlen:** $f_1 = f_2 = 1, f_{n+2} = f_{n+1} + f_n$

   $\forall\ n \geq 1 : (f_n = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}})$

   Beachte: $f_{n+1} = f_n + f_{n-1}$ und $\frac{1+\sqrt{5}}{2}+1 = (\frac{1+\sqrt{5}}{2})^2, \frac{1-\sqrt{5}}{2}+1 = (\frac{1-\sqrt{5}}{2})^2$
   Außerdem gilt: $f_{n+1} * f_{n-1} - f_n^2 = (-1)^n$

   Betrachte z.B. $f_7 * f_5 - f_6^2$

7.4 Algebraische Strukturen

Für sichere (was immer das exakt sein mag) kryptographische Algorithmen benötigen wir Strukturen, die es erlauben, Operationen zu definieren, die einfach zu berechnen sind, für die es aber sehr schwierig ist, die inverse Operation zu berechnen.
Wir folgen hier zum großen Teil der Darstellung in [17].

7.4.1 Mengen, Operatoren, Erzeugendensysteme

Definition:
Eine **algebraische Struktur** $A = (M, OP)$ besteht aus einer Menge M, der **Trägermenge**, sowie einer endlichen Menge von **Operationen** $OP = \{op_1, ..., op_n\}$

$$op_i : M^{k_i} \longrightarrow M, k_i \geq 0, 1 \leq i \leq n$$

Die op_i sind total definiert: $D(op_i) = M^{k_i}$.
In der Regel sind die Operationen unär oder binär.

Beispiel:

$$(\mathcal{N}_0, \{+, *\})$$
$$(\mathcal{Z}, \{+, -, *\})$$
$$(\mathcal{Q}, \{+, -, *, /\})$$
$$(\{0, 1\}, \{\vee, \wedge, \neg\})$$
$$(\{1, 3, 4, 12, 16\}, \{ggt\})$$

Eigenschaften von Operatoren

Kommutativität:

$(A, *) : \forall\ a, b \in A : (a * b = b * a)$
dann heißt A **kommutativ**.

Assoziativität:

$(A, *) : \forall\ a, b, c \in A : (a * (b * c) = (a * b) * c)$
dann heißt A **assoziativ**.

Neutrales Element:

$(A, *) : \exists\ e \in A : (\forall\ a \in A : (a * e = e * a = a))$
dann ist e das **neutrale Element** oder **Einselement** von A.
Das Einselement ist eindeutig bestimmt!

Inverses Element:

$(A, *) : \forall\ a \in A : (\exists\ b \in A : (a * b = b * a = e))$
dann heißt b das Inverse zu a.
Es gilt $(a^{-1})^{-1} = a$!

Beispiel:

Operation	Menge	Neutrales Element
$+$	$\mathcal{N}, \mathcal{Z}, \mathcal{Q}, \mathcal{R}$	0
$*$	$\mathcal{N}, \mathcal{Z}, \mathcal{Q}, \mathcal{R}$	1
$\bigcap$	$\mathcal{P}(M)$	M
$\bigcup$	$\mathcal{P}(M)$	$\emptyset$
$\cdot$	Σ^*	ε
$\circ$	$f : M \longrightarrow M$, f bij.	id_M

Operation	Menge	Inverses
$+$	$\mathcal{N}, \mathcal{Z}, \mathcal{Q}, \mathcal{R}$	$-a$
$*$	$\mathcal{N}, \mathcal{Z}, \mathcal{Q}, \mathcal{R}$	$a^{-1} = 1/a$
$\bigcap$	$\mathcal{P}(M)$	—
$\bigcup$	$\mathcal{P}(M)$	—
$\cdot$	Σ^*	—
$\circ$	$f : M \longrightarrow M$, f bij.	f^{-1}

Erzeugendensystem

Definition:

$E \subseteq M$ heißt **Erzeugendensystem** oder **Basis** von M, wenn es für jedes $a \in M$ $b_1, ..., b_k \in E$ gibt, mit $a = b_1 * ... * b_k$

Beispiel:

Für $(\mathcal{N}_2, *)$ ist die Menge der Primzahlen ein Erzeugendensystem.
Für $(\mathcal{N}_1, +)$ ist die$\{1\}$ ein Erzeugendensystem.

Betrachte $(\{u, v, w, x, y\}, *)$

$*$	u	v	w	x	y
u	u	v	w	x	w
v	v	w	v	w	v
w	w	v	w	v	w
x	x	w	v	u	w
y	w	v	w	v	y

$*$ ist kommutativ und assoziativ. Es gibt kein neutrales Element.
$\{x, y\}$ ist ein Erzeugendensystem.
$x^3 = x$: Erzeugendensystem mit Identitäten.

Ein Erzeugendensystem ohne Identitäten heißt **freies Erzeugendensystem**.

Beispiel:

Primzahlen in $\mathcal{N}_1$

7.4.2 Halbgruppen

Definition:
Eine algebraische Struktur $H = (M, *)$ heißt **Halbgruppe**
falls $*$ **assoziativ** auf M ist.

Definition:
Eine Halbgruppe mit neutralem Element heißt **Monoid**.

Definition:
Eine Halbgruppe $H = (M, *)$ mit kommutativer Operation $*$
heißt **abelsch**.

Definition:
Eine Halbgruppe mit einem einelementigen Erzeugendensystem heißt
zyklisch.

Beispiel:
$(\mathcal{N}, +)$

Sei $H = (M, *)$ Halbgruppe; dann ist für jedes $a \in M$
$< a >= (\{a^k | k \in \mathcal{N}\}, *)$ eine Halbgruppe mit $a^m * a^n = a^{m+n}$.

Definition:
$H = (M, *)$, dann ist für $a \in M < a >$
die von a erzeugte zyklische Halbgruppe.

Definition:

Sei $H = (M, *)$ eine Halbgruppe, $U \subseteq M$, U abgeschlossen bzgl. $*$, d.h. $\forall\ a, b \in U : (a * b \in U)$. Dann heißt $(U, *)$ **Unterhalbgruppe** von H.

Beispiel:

$(\mathcal{N}_0, +) :< 2 >$ ist Unterhalbgruppe.

$(\{2^i * 3^j \mid i, j \geq 0\}, *)$ ist Unterhalbgruppe von $\mathcal{N}_0$.

7.4.3 Gruppen

Definition:
Sei $G = (M, *)$ ein Monoid, e Einselement in G. Falls für jedes $a \in G$ ein Inverses $b \in G$ existiert, mit $a * b = b * a = e$, so heißt G **Gruppe**.
Ist G kommutativ, so heißt G auch **abelsch**.
Ist G endlich, so heißt $|M|$ die **Ordnung** von G.

Beispiel:
$(\mathcal{Z}, +)$ ist abelsch.

$(\{f \mid f : M \longrightarrow M\}, \circ)$ f bijektiv, ist Gruppe, aber im Allgemeinen nicht abelsch.

Satz:

Für eine Gruppe $G = (M, *)$ gelten folgende Eigenschaften:

1. G besitzt genau ein neutrales Element.
2. Zu jedem Element von G gibt es genau ein Inverses.
3. Das neutrale Element ist zu sich selbst invers.
4. $\forall\ a \in G : ((a^{-1})^{-1} = a)$
5. $(\forall\ a, b, c \in G : (a * b = a * c) \Longrightarrow b = c)$ und
   $(\forall\ a, b, c \in G : ((b * a = c * a) \Longrightarrow (b = c)))$ **Kürzungsregel**
6. $\forall\ a, b \in G : ((a * b)^{-1} = b^{-1} * a^{-1})$
7. $a, b, c, d \in G$ bekannt, dann sind $a * x = b$ und $y * c = d$ eindeutig lösbar.

In Halbgruppen gilt die Kürzungsregel normalerweise nicht:

$(P(\{1,2,3,4\}, \bigcup) : \{2,3\} \bigcup \{1\} = \{2,3\} \bigcup \{1,2\}$ aber $\{1\} \neq \{1,2\})$

Beweisskizzen:

1. gilt allgemein für algebraische Strukturen
2. $a^{-1} = a^{-1} * e = a^{-1} * (a * a'^{-1}) = (a^{-1} * a) * a'^{-1} = e * a'^{-1} = a'^{-1}$
3. $e * e = e * e^{-1}$
4. $e = a^{-1} * (a^{-1})^{-1} \Longrightarrow a * e = ... = (a^{-1})^{-1}$
5. Multipliziere von links mit a^{-1}
6. $(a * b) * (a * b)^{-1} = e$ und $(a * b) * (b^{-1} * a^{-1}) = e$
7. Linksmultiplikation mit a^{-1}

Gruppenisomorphismen

Definition:

Seien $G_1 = (M_1, *_1)$ und $G_2 = (M_2, *_2)$ Gruppen. G_1 und G_2 heißen **isomorph**, wenn es eine bijektive Abbildung $\phi : M_1 \longrightarrow M_2$ gibt, mit $\forall\, a, b \in G_1 : (\phi(a *_1 b) = \phi(a) *_2 \phi(b))$.
ϕ heißt **Isomorphismus** zwischen G_1 und G_2.

Beispiel:

$\mathcal{Z}_4$

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$\mathcal{Z}_5$

$\otimes$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

$\mathcal{K}_4 = (\{0, 1, a, b\}, \diamond)$

$\diamond$	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

$\mathcal{K}_4$ ist abelsch, jedes Element zu sich selbst invers.

$\mathcal{K}_4$ ist die **Kleinsche Vierergruppe**

Gesucht Isomorphismen zwischen den Gruppen.

$\phi_1 : \mathcal{Z}_4 \longrightarrow \mathcal{Z}_5$

$\phi_1(0) = 1, \phi_1(1) = 4, \phi_1(2) = 3, \phi_1(3) = 2$

$\phi_1(3 \oplus 3) \neq \phi_1(3) \oplus \phi_1(3)$ aber

$\phi_2(0) = 1, \phi_2(1) = 2, \phi_2(2) = 4, \phi_2(3) = 3$ tut's.

Es gibt keinen Isomorphismus zwischen $\mathcal{K}_4$ und $\mathcal{Z}_4$

Satz:
Sei $G = (M, *)$ Gruppe, $|M| = 4$, dann gilt $G \simeq \mathcal{K}_4$ oder $G \simeq \mathcal{Z}_4$

Zyklische Gruppen

$G = (M, *)$ Gruppe mit Einselement e.

Potenzschreibweise: für jedes $a \in M$

$$a^0 = e$$

$$a^{n+1} = a^n * a, n \geq 0$$

$$a^{-(n+1)} = a^{-n} * a^{-1}, n \geq 0$$

Dann gilt

$$(a^r)^s = a^{r*s}, r, s \in \mathcal{Z}$$

$$a^r * a^s = a^{r+s}, r, s \in \mathcal{Z}$$

$$(a^n)^{-1} = (a^{-1})^n = a^{-n}$$

Definition:
Eine Gruppe $G = (M, *)$ heißt **zyklisch**, falls es ein $a \in M$ gibt, so dass für alle $x \in M$ ein $k \in \mathcal{Z}$ existiert mit $x = a^k$.
Man schreibt $G = (< a, * >)$ und
a heißt **erzeugendes Element von G**.

Folgerung:

Zyklische Gruppen sind abelsch.

Folgerung:

Sei $G = (< a, * >)$. Gibt es ein $k \in \mathcal{N}_0$ mit $a^k = e$ und ist k die kleinste Zahl mit dieser Eigenschaft, so gilt

$$\forall\, i, j \in \mathcal{N}_0 : (0 \leq i, j \leq k-1 \Rightarrow (a^i \neq a^j))$$

und

$$M = \{e, a^1, ..., a^{k-1}\}$$

.

k ist dann die Ordnung von G.

Definition:

Sei $G = (M, *)$ Gruppe, $U \subseteq M$ und für alle $a, b \in U$ gilt $a * b \in U$ und für alle $a \in U$ gilt $a^{-1} \in U$. Dann heißt U **Untergruppe** von G.

Permutationsgruppen

Es gibt $n!$ Permutationen einer Menge mit n Elementen.

	1	2	3
f_0	1	2	3
f_1	2	3	1
f_2	3	1	2
f_3	1	3	2
f_4	3	2	1
f_5	2	1	3

$\circ$	f_0	f_1	f_2	f_3	f_4	f_5
f_0	f_0	f_1	f_2	f_3	f_4	f_5
f_1	f_1	f_2	f_0	f_4	f_5	f_3
f_2	f_2	f_0	f_1	f_5	f_3	f_4
f_3	f_3	f_5	f_4	f_0	f_2	f_1
f_4	f_4	f_3	f_5	f_1	f_0	f_2
f_5	f_5	f_4	f_3	f_2	f_1	f_0

Satz:
Die algebraische Struktur

$$S_n = (\{f : \mathcal{N}_1^n \longrightarrow \mathcal{N}_1^n | f \text{ bijektiv}\}, \circ) = (\{f_0, ..., f_{n!-1}\}, \circ)$$

wobei die Funktionen $f_i, 0 \leq i \leq n! - 1$ die Permutationen auf $\mathcal{N}_1^n$ sind, ist eine Gruppe der Ordnung $n!$ mit Einselement f_0.

Definition:
S_n heißt **vollständige symmetrische Gruppe**.

Definition:
Jede Untergruppe einer vollständigen symmetrischen Gruppe heißt **Permutationsgruppe**.

Der Satz von Cayley

Satz:

Jede endliche Gruppe ist isomorph zu einer Permutationsgruppe.

Beweis:

Sei $G = (M, *)$ endliche Gruppe der Ordnung n mit Einselement e.
Definiere

$$\forall\, a \in M : (f_a : M \longrightarrow M, f_a(x) = a * x)$$

Dann ist f_a eine Permutation auf M, denn $x \neq y \Rightarrow f_a(x) \neq f_a(y)$
weil $f_a(x) = a * x = a * y = f_a(y) \Rightarrow x = y$
wegen der Kürzungsregel.

$F^M = (\{f_a \mid a \in M\}, \circ)$ ist eine Gruppe, denn

1. $\circ$ ist assoziativ
2. f_e ist Einselement
3. $f_{a^{-1}}$ ist Inverses zu f_a

Definiere $\Phi : M \longrightarrow F^M : \Phi(a) = f_a$, dann ist Φ ein Isomorphismus.

1. Φ ist total und surjektiv.
2. Φ ist injektiv, (Kürzungsregel!).
3. Φ ist Homomorphismus, d.h.
   $\Phi(a*b)(x) = f_{a*b}(x) = (a*b)*x = a*(b*x) = f_a(b*x) = f_a(f_b(x))$
   $= f_a(\Phi(b)(x)) = \Phi(a)(\Phi(b)(x)) = \Phi(a) \circ \Phi(b)(x)$

Der Satz von Lagrange

Definition:

Sei $G = (M, *)$ eine Gruppe, $G_U = (U, *)$ eine Untergruppe von G. Für jedes $a \in M$ heißt $a * U = \{a * x | x \in U\}$ **Linksnebenklasse** von U und $U * a = \{x * a | x \in U\}$ **Rechtsnebenklasse** von U.

Beispiel:

$(4 * \mathcal{Z}, +)$ ist eine Untergruppe von $(\mathcal{Z}, +)$.

$4 * \mathcal{Z} + 1$
$4 * \mathcal{Z} + 2$
$4 * \mathcal{Z} + 3$

sind drei Linksnebenklassen.

$U = (\{-1, 1\}, *)$ ist Untergruppe von $(\mathcal{Q} \setminus \{0\}, *)$. Es gibt unendlich viele Nebenklassen $x * U = (\{-x, x\}, *), x > 0$.

Betrachte $(\{f_0, f_1, f_2\}, \circ)$ Untergruppe von S_3,
dann gilt für alle $f \in \{f_0, f_1, f_2\}$ $f_0 \circ f \in \{f_0, f_1, f_2\}$, da $\{f_0, f_1, f_2\}$ Gruppe und alle Nebenklassen mit Elementen nicht aus $\{f_0, f_1, f_2\}$ sind identisch und haben die gleiche Anzahl von Elementen wie $\{f_0, f_1, f_2\}$. Die Nebenklassen sind $\{f_3, f_5, f_4\}$.

Satz:
Sei $G = (M, *)$ eine endliche Gruppe und $G_U = (U, *)$ eine Untergruppe von G. Die Relation $\sim \subseteq M \times M$ sei definiert als: $a \sim b \Leftrightarrow b * a^{-1} \in U$.
Dann gilt:
a) $\sim$ ist eine Äquivalenzrelation
b) Für jedes $a \in M$ ist die Rechtsnebenklasse $U*a = [a]_\sim$ die Äquivalenzklasse, die a enthält.
c) $|U| \mid |M|$, die Ordnung von U teilt die Ordnung von M.
Beweis:
a) U Untergruppe, $a * a^{-1} \in U$ also ist $\sim$ **reflexiv**.

$a \sim b$: also $b * a^{-1} \in U$, also $(b * a^{-1})^{-1} = a * b^{-1} \in U$
also ist $b \sim a \sim$ **symmetrisch**.
$a \sim b, b \sim c \Rightarrow b * a^{-1}, c * b^{-1} \in U \Rightarrow$
$c * b^{-1} * b * a^{-1} = c * a^{-1} \in U \Rightarrow a \sim c$
$\sim$ ist **transitiv**.

b) z.z. $b \in U * a \Leftrightarrow a \sim b$
$b \in U * a \Rightarrow \exists x \in U : b = x * a \Rightarrow b * a^{-1} = x \in U \Rightarrow a \sim b$
$a \sim b \Rightarrow b * a^{-1} \in U \Rightarrow b * a^{-1} * a \in U * a \Rightarrow b \in U * a \Rightarrow U * a = [a]_\sim$

c) Bijektion zwischen $U * a$ und U: $f_a : U \to U * a$ mit $f_a(x) = x * a$
f_a definiert für alle x also **total**.
f_a ist **surjektiv**, denn für jedes $y \in U*a$ existiert ein $x \in U$ mit $y = x*a$.
$f_a(x_1) = f_a(x_2) \Rightarrow x_1 * a = x_2 * a \Rightarrow x_1 = x_2$
f_a ist **injektiv**.

Also haben alle Äquivalenzklassen von $\sim$ die gleiche Anzahl von Elementen, nämlich $|U|$.

Die Äquivalenzklassen bilden eine Partition von M: wenn es r Klassen gibt, so hat M $r * |U|$ Elemente.

Folgerung:
Sei $G = (M, *)$ eine endliche Gruppe und $G_U = (U, *)$ eine Untergruppe von G, $a, b \in M$:

a) Dann ist entweder $U * a = U * b$ oder $U * a \bigcap U * b = \emptyset$.
Die Nebenklassen einer Gruppe bilden also eine Äquivalenzrelation.

b) Alle Nebenklassen haben die gleiche Anzahl von Elementen.

Satz:

Sei $G = (M, *)$ eine endliche Gruppe der Ordnung p, p Primzahl. Dann besitzt G außer den trivialen Untergruppen keine weiteren Untergruppen.

Folgerung:

Sei $G = (M, *)$ eine endliche Gruppe der Ordnung p, p Primzahl.
Dann ist G zyklisch und damit auch kommutativ.

Beispiel:

Betrachte $\mathcal{Z}_{13} = (Z_{13} - \{0\}, \otimes)$
2 hat die Ordnung 12, $\mathcal{Z}_{13} = \{2^k | 0 \leq k < 12\}$

Definition:
Sei $G = (M, *)$ eine Gruppe mit Einselement e und $a \in M$. Die kleinste Zahl $k \in \mathcal{N}$ mit $a^k = e$ heißt Ordnung von $a : k = ord_G(a)$.

Folgerung:
Das Einselement hat immer die Ordnung 1.

Satz:
Sei $G = (M, *)$ eine endliche Gruppe. Dann besitzt jedes Element von G endliche Ordnung.

Beweis:
Sei $G = (M, *)$ Gruppe mit endlicher Ordnung n, $x \in M$.
Betrachte $x^0, x^1, x^2, ..., x^n$.
Dann gibt es $i, j : 0 \leq i, j \leq n$ und $x^i = x^j$
oBdA $j > i$: setze $k = j - i \Rightarrow 0 < k < j \leq n$
$x^k = x^{j-i} = x^j * (x^i)^{-1} = x^j * (x^j)^{-1} = e$

Folgerung:
Sei $G = (M, *)$ eine endliche Gruppe der Ordnung n. Dann besitzt jedes Element $a \in M$ eine endliche Ordnung $1 \leq ord_G(a) \leq n - 1$.

Der kleine Satz von Fermat

Den folgenden Satz benötigen wir gleich für unseren Beweis:

Satz:
Sei $G = (M, *)$ eine Gruppe mit Einselement e und $U \subseteq M$. Dann gilt $G_U = (U, +)$ ist eine Untergruppe von G genau dann, wenn $\forall a, b \in U$: $a^{-1} * b \in U$.

Beweis:
$\Rightarrow$:
gilt wegen Definition der Untergruppe
$\Leftarrow$:
$\forall a, b \in U : \ a^{-1} * b \in U$
$\Rightarrow$ mit $b = a, \ a^{-1} * a = e \in U$, also enthält U das Einselement

$a \in U, e \in U \Rightarrow$ mit $b = e$: $a^{-1} * e = a^{-1} \in U$

Mit $x, y \in U$ ist also auch $x^{-1}, y^{-1} \in U$
dann ist mit $a = x^{-1}, b = y$ auch $a^{-1} * b = (x^{-1})^{-1} * y = x * y \in U$

Satz:

Sei $G = (M, *)$ eine Gruppe, $a \in M,\ k = ord_G(a)$.
Dann ist $< a >= (a, a^2, ..., a^k, *)$ eine Untergruppe der Ordnung k.

Beweis:

Wir zeigen $x, y \in < a > \Rightarrow x^{-1} * y \in < a >$:

$x = a^i,\ y = a^j,\ 1 \leq i, j \leq k$

$x^{-1} * y = (a^i)^{-1} * a^j = a^{j-i}$

$j > i \Rightarrow 1 \leq j - i \leq k \Rightarrow a^{j-i} \in < a >$

$j \leq i$: $1 \leq i \leq k \Rightarrow i \leq k + j \leq k + i \Rightarrow 0 \leq k + j - i \leq k$

$\Rightarrow a^{j-i} = e * a^{j-i} = a^k * a^{j-i} = a^{k+j-i} \in < a >$

Definition:

Sei $G = M, *)$ eine Gruppe, $a \in M$ habe die Ordnung k. Dann heißt $< a >= (\{a, a^2, ..., a^k\}, *)$ die von a erzeugte zyklische Untergruppe.

Beispiel:

Betrachte $\mathcal{Z}_{13}$:
2 hat die Ordnung 12; $< 2 >= (\{2^k | 0 \leq k < 12\}, \otimes) = \mathcal{Z}_{13}$

4 hat die Ordnung 6; $< 4 >= (\{4^k | 0 \leq k < 6\}, \otimes)$ ist Untergruppe der Ordnung 6.

Satz:
Sei $G = (M, *)$ eine endliche Gruppe der Ordnung n mit Einselement e. Dann gilt für jedes $a \in M$ $a^n = e$.

Beweis:
Sei k die Ordnung von a und damit von $< a >$. Dann gilt nach Lagrange $| < a > | \; | \; n$, also n = k * r:

$a^n = a^{k*r} = (a^k)^r = e^r = e$

Folgerung:
Sei $G = (M, *)$ eine endliche Gruppe der Ordnung n, $a \in M$, dann gilt $ord_G(a)|n$.

Satz:
((Kleiner Satz von Fermat))
Sei p eine Primzahl. Für alle Elemente x von $\mathcal{Z}_p = (Z_p \setminus \{0\}, \otimes)$ gilt $x^{p-1} = 1$.

Folgerung:
a) p Primzahl, $x \in \mathcal{Z},\ ggt(x, p) = 1$ dann ist $x^p = x \; mod \; p$

b) $n \in \mathcal{N}_2$. Dann gilt $\exists x \in \mathcal{N} : x^n \neq x \; mod \; n \Rightarrow x$ ist nicht prim.

Index

Literaturverzeichnis

[1] Friedrich L. Bauer. *Kryptologie*. Springer-Verlag, Berlin Heidelberg New York, 1993.

[2] Friedrich L. Bauer. *Entzifferte Geheimnisse*. Springer-Verlag, Berlin Heidelberg New York, 1995.

[3] Albrecht Beutelspacher. *Kryptologie*. Vieweg, Braunschweig, 1994.

[4] Thomas H. Cormen. *Introduction to Algorithms*. McGraw-Hill MIT Press, New York, 1990.

[5] F.H.Hinsley and Alan Stripp. *Codebreakers*. Oxford University Press, Oxford, GB, 1993.

[6] Simon Garfinkel. *PGP Pretty Good Privacy*. O'Reilley & Associates, Inc., Sebastopol, CA, U.S.A, 1995.

[7] Robert Harris. *Enigma*. Heyne, München, 1995.

[8] Gilbert Held. *Top Secret Data Encryption Techniques*. SAMS Publishing, Carmel, Indiana, U.S.A., 1993.

[9] Andrew Hodges. *Alan Turing - the enigma*. Vintage, London, 1992.

[10] David Kahn. *The Codebreakers*. Scribner, New York, 1996.

[11] Rudolf Kippenhahn. *Verschlüsselte Botschaften*. Rowohlt, Reinbek, 1997.

[12] Wladyslaw Kozaczuk. *Geheimoperation Wicher*. Bernard & Graefe, Koblenz, 1989.

[13] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., New York, 1994.

[14] Simon Singh. *The Codebook*. Doubleday, New York, 1999.

[15] Albrecht Beutelspacher und Jörg Schwenk und Klaus-Dieter Wolfenstetter. *Moderne Verfahren der Kryptographie*. Vieweg, Braunschweig, 1995.

[16] Albrecht Beutelspacher und Marc-Alexander Zschiegner. *Diskrete Mathematik für Einsteiger*. Vieweg, Braunschweig, 2002.

[17] Kurt-Ulrich Witt. *Algebraische Grundlagen der Informatik*. Vieweg, Braunschweig, 2001.

[18] Reinhard Wobst. *Abenteuer Kryptologie*. Addison-Wesley, Bonn, 1998.