Wirtschaftsinformatik

Informatik Grundlagen

Grundlagen der Codierung

Information und Kommunikation

„Kommunikation ist der Austausch von Informationen“. Dies setzt Verschlüsselung der **Information** voraus. (z.B. gesprochene/geschriebene Sprache, Bilder, etc.)
Verschlüsselung geschieht bei uns im geschriebenen Wort durch 26 Buchstaben
10 Ziffern
mind. 10 Sonderzeichen

Was aber sind „Informationen“ oder was ist eine „Information“? Dazu später genaureres.

In der Informatik bedient man sich anderer **Codes**. Man benutzt ein sog. **Binär-Alphabet**.
Die Binärität dieses Alphabet entsteht durch die Verwendung von nur zwei Zeichen, also z.B. {0,1},{.,-}
Der dabei kleineste, nicht mehr teilbarer Träger von Information, bzw. Daten, bezeichnet man als „**Zeichen**“
Die Übermittlung von Zeichen baut auf der Übertragung von **Signalen** auf. Dabei unterscheidet man

- analoge Signale (z.B. Schallschwingungen oder Spannungsschwankungen)
- diskrete (digitale) Signale

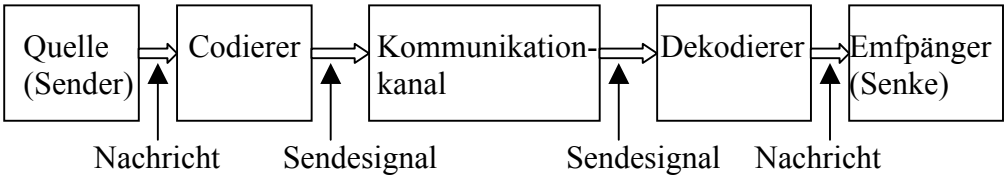
diskret meint dabei, daß die digitalen Signale nur eine endliche Anzahl Zustände annehmen.

Die Menge aller unterschiedlichen Zeichen eines Codes nennt man „**Alphabet**“.
Beispiele für Alphabete sind :

{A,B,C,...,X,Y,Z}
{α,β,γ,...,ψ,ω}

Kommunikation funktioniert grundsätzlich nach folgendem Schema:

Der Kommunikationsprozess



In diesem Fluß der Information können jedoch jederzeit Sörungen auftreten. Es ist daher wichtig, diese auch erkennen zu können.

Informationstheorie nach Shannon

Ein paar Beispiele zu der Codierung:

Beispiel 1) Ein Alphabet besteht aus 8 Zeichen : {A,B,C,D,E,F,G,H}
Soll nun Information aus diesem Alphabet codiert werden, so muß sich der Codierer einer Übersetzungsforschrift bedienen, also z.B. einer Tabelle :

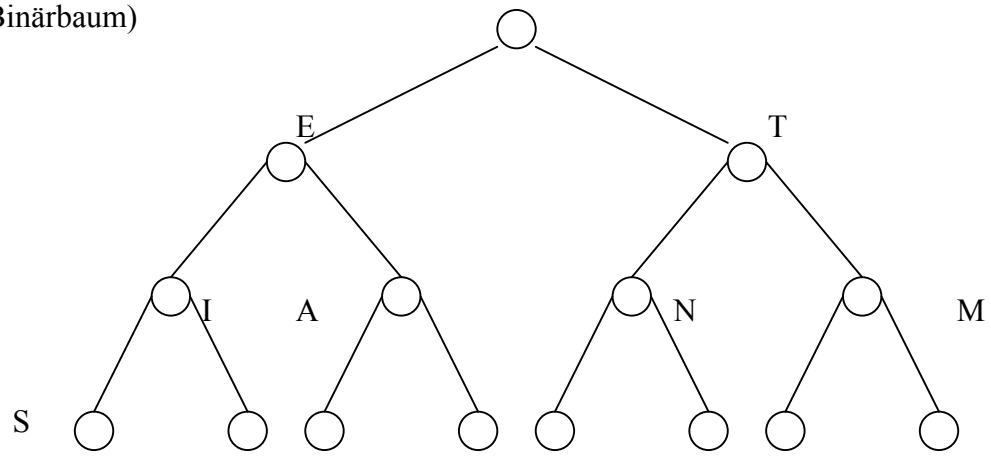
Alph1	Alph2
A	000
B	001
C	010
D	011
E	100
F	101
G	110
H	111

Die Codierung der Nachricht ist notwendig, da zum Senden möglicherweise nur eine Taschenlampe vorhanden ist, mit der man mit Licht an/aus das Aph2 senden kann, aber nun mal nicht den Buchstaben „A“.
Wenn dann der Empfänger wieder die gleiche Tabelle verwendet, kann er die Nachricht verstehen.

Der Code: Eine Abbildungsvorschrift (Zuordnung) einer Menge (Alph1) auf eine andere (Alph2). Diese Vorschrift muß umkehrbar sein, also Ein-Eindeutig.

Beispiel 2) Der Morsecode. Der Morsecode besteht aus drei Zeichen: . – Pause

Der Codebaum. Linke Äste sind Punkte, rechte Äste sind Striche.
(Der Binärbaum)



Also SOS ist ... --- ...
EIS ist . .. ...
Ohne Pause als zusätzliches Zeichen wäre keine Codierung eindeutig möglich, da das Wort EIS nur aus Punkten besteht.

Ist das Zielalphabet ein Binäres Alphabet (nur zwei Zeichen), so nennt man es „**Binärcode**“, das einzelne Zeichen nennt man „**BIT**“ (Binary Digit).
Dabei verwendet man folgende Stückelung, die auf der Basis 2^{10}.

1024 BIT	=	1 Kilobit
1024 KBit	=	1 Megabit
1024 MBit	=	1 Gigabit
1024 GBit	=	1 Terabit

Eine andere Art der Gruppierung der Bits ist die Anordnung zu jeweils 8 Bit gleich einem **Byte**.

8 Bit = 1 Byte => 256 Zeichen (2^8) können codiert werden.

Entsprechend der Bitstaffelung können auf Bytes in 1024er-Gruppen zusammengefasst werden.

Der Informationsgehalt einer Nachricht.

Informationstheorie

C.E. Shannon (1948) versuchte, ein Maß für den Informationsgehalt einer Nachricht zu finden, d.h. die Frage zu beantworten, wieviel Information enthällt eine (diskrete) Nachricht, die von einem Sender zu einem Empfänger übermittelt wird. (**Nachricht** = Zeichenfolge, in der Zeichen mit bestimmten Wahrscheinlichkeiten auftreten)

Beispieltext : OTTO HAT EIN BOOT (Insgesamt 14 Zeichen)

O	T	H	A	E	I	N	B	
4	4	1	1	1	1	1	1	absolute Häufigkeit
$\frac{4}{14}$	$\frac{4}{14}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{14}$	$\frac{1}{14}$	relative Häufigkeit

Der Informationsgehalt *h* soll folgende Eigenschaften haben :

1.) *h* ist von der Wahrscheinlichkeit abhängig, mit der Zeichen auftreten (nicht von der Art der kodierung) Häufig gesendete Zeichen sollen einen niedrigen Informationsgehalt haben, selten gesendete Zeichen einen entsprechend höheren.
2.) Besteht eine Nachricht aus mehreren, von einander unabhängigen Zeichen, so soll deren Informationsgehalt gleich der Summe der Informationsgehalte einzelner Zeichen sein.

Abhängigkeit von Zeichen

In natürlichen Sprachen sind die Zeichen des Alphabetes von einander abhängig, da man einzelne Buchstaben bis ganze Worte weglassen könnte, ohne dem Informationsgehalt einer Nachricht zu schaden.

Zu 1. Sei *p* die Auftrittswahrscheinlichkeit eines Zeichens, dann ist

$$f\left(\tfrac{1}{p}\right)$$

eine streng monotonwachsende Funktion, welche den Informationsgehalt *h* beschreiben soll.

Zu 2. Bei Unabhängigkeit der Zeichen => Das Produkt der beiden Einzelwahrscheinlichkeiten ergibt die Wahrscheinlichkeit, daß zwei Zeichen kombiniert auftreten. Daher muß für die Funktion, die den Informationsgehalt beschreibt, gelten :

$$f(x) + f(y) = f(x \cdot y)$$

Das ist bei Logarithmusfunkionen der Fall. Denn es gilt :

$$\log_{10}(10 \cdot 10) = \log_{10}(10) + \log_{10}(10) = 2$$

Beide Forderungen (1 und 2) werden erfüllt durch die Logarithmusfunktion log zu einer beliebigen Basis. Als Basis in der Informatik wählt man nun die 2, den dualen Logarithmus, bzw. den logarithmus dualis (ld).

Informationsgehalt, mittlere Wortlänge, Redundanz

Def.: Der Informationsgehalt für ein einzelnes Zeichen ist

$$h = \mathrm{ld}\left(\frac{1}{p}\right) = \mathrm{ld}\,1 - \mathrm{ld}\;p = 0 - \mathrm{ld}\;p = -\mathrm{ld}\;p$$

Da für p nur Werte im Bereich $0<p\leq 1$ interessierren, ist $-\mathrm{ld}\;p$ immer positiv. (s. Kopie)

Exkurs: *Exponentialfunktionen und Logarithmusfunktionen*

Die Funktion $f(x) = a^x$; $a>0$, $a\neq 1$ heißt Exponentialfunktion zur Basis a.

z.B. $a=10 \quad \Rightarrow f(x) = 10^x \qquad \left(f(x) = 10^{-3} = \frac{1}{10^3}\right)$

x	-3	-2	-1	0	1	2	3
10^x	$\frac{1}{1000}$	$\frac{1}{100}$	$\frac{1}{10}$	1	10	100	1000

oder z.B. $a=2 \quad \Rightarrow f(x) = 2^x$

x	-2	-1	0	1	2	3	4
2^x	$\frac{1}{4}$	$\frac{1}{2}$	1	2	4	8	16

Die Umkehrfunktion der Exponentialfunktion zur Basis a heißt Logarithmus zur Basis a

Für eine Zahl $b>0$ ist $\log_a b$ diejenige Zahl reelle Zahl, für die gilt $a^x = b$

z.B. $\log_{10} 10 = 1$; da $10^1 = 10$

$\log_{10} 1000 = 3$; da $10^3 = 1000$

$\log_2 4 = 2$; da $2^2 = 4$

$\log_2 8 = 3$; da $2^3 = 8$

$\log_3 \frac{1}{9} = -2$; da $3^{-2} = \frac{1}{3^2} = \frac{1}{9}$

$\log_4 1 = 0$; da $4^0 = 1$

$\log_2 1024 = 10$; da $2^{10} = 1024$

Bezeichnungen:

$$\log_{10} x = \log x = \lg x$$
$$\log_e x = \ln x$$
$$\log_2 x = \mathrm{ld}\, x$$

Regeln:

$$\log_a (x \cdot y) = \log_a x + \log_a y$$
$$\log_a \left(\frac{x}{y}\right) = \log_a x - \log_a y$$
$$\log_a x^y = y \cdot \log_a x$$
$\log_a a = 1$; da $a^1 = a$

$\log_a 1 = 0$; da $a^0 = 1$

$$\log_a \left(\frac{1}{y}\right) = \log_a 1 - \log_a y = -\log_a y$$

Umrechnungsregel:

$$\log_a b = \frac{\log_c b}{\log_c a} = \frac{\ln b}{\ln a} = \frac{\lg b}{\lg a}$$

und für die Informatik ganz wichtig:

$$\mathrm{ld}\, b = \frac{\ln b}{\ln 2} = \frac{\ln b}{0.69314718}$$

Beispiele für die Nachrichtenqualitäten:

1. Konsequentes senden des gleichen Zeichens:
   $\Rightarrow p = 1$, $h = -\mathrm{ld}\ p = -\mathrm{ld}\, 1 = 0$

2. Zeichen, das nie gesendet wird:

$\Rightarrow p = 0 \quad , h = -\text{ld } 0 = \text{nicht definiert.}$

3. Senden von zwei Zeichen mit gleicher wahrscheinlichkeit des Auftretens:

$$\Rightarrow p = \frac{1}{2} \quad , h = -\text{ld}\,\frac{1}{2} = -(\text{ld } 1 - \text{ld } 2) = -(0-1) = 1$$

4. Senden von *n* Zeichen in **Gleichverteilung**:

$$\Rightarrow p = \frac{1}{n} \quad , h = -\text{ld}\,\frac{1}{n} = \text{ld } n$$

Falls $n = 2^k$

$$\Rightarrow p = \frac{1}{2^k} \quad , h = -\text{ld}\,\frac{1}{2^k} = -(\text{ld } 1 - \text{ld } 2^k) = \text{ld } 2^k = k$$

Beispiel : ASCII-Code besteht aus $256 = 2^8$ Zeichen. Für $n = 2^8$ gilt daher: $h=8$

1. Beispiel:

„HALLO“ 5 Bytes=40 Bit werden normalerweise zum Speichern der Nachricht verwendet.

$$p := \frac{1}{5}; \frac{1}{5}; \frac{2}{5}; \frac{1}{5}$$

Vollständiger Informationsgehalt der Nachricht =

$$\sum_i h_i = \sum_i -\text{ld } p_i = 2.32 + 2.32 + 1.32 + 2.32 \approx 8.3 \text{ bit}$$

8.3 bit würden genügen, um die Nachricht zu speichern

Def.: Sei h_i der Informationsgehalt des i-ten Zeichens und p_i die Wahrscheinlichkeit des Auftretens dieses Zeichens, dann ist der mittlere Informationsgehalt (***Entropie***) dieses Alphabetes folgendermaßen definiert :

$$H := \sum_i p_i h_i = \sum_i p_i \,\text{ld}\left(\frac{1}{p_i}\right) = \underbrace{-\sum_i p_i \,\text{ld}\, p_i}_{\text{Erwartungswert}}$$

Die Einheit des mittleren Informationsgehaltes H ist **bit**.

1. Beispiel fortgesetzt:

Der mittlere Informationsgehalt der Nachricht wäre:

$$H = -\left(\frac{1}{5}\cdot(-2.32) + \frac{1}{5}\cdot(-2.32) + \frac{2}{5}\cdot(-1.32) + \frac{1}{5}\cdot(-2.32)\right) = 1.92 \text{ bit}$$

2. Beispiel:

3 Zeichen : x ,y ,z

$$p_x = \frac{1}{2} \qquad p_y = p_z = \frac{1}{4}$$

$$h_x = -\text{ld}\frac{1}{2} = 1$$

$$h_y = h_z = -\text{ld}\frac{1}{4} = 2$$

$$H = \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{4}\cdot 2 = 1\frac{1}{2} \text{ bit}$$

3. Beispiel:

Der Text „VOLLTOLL“

	V	O	L	T
p_i	$\frac{1}{8}$	$\frac{2}{8}$	$\frac{4}{8}$	$\frac{1}{8}$
h_i	3	2	1	3
$p_i h_i$	$\frac{3}{8}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{8}$

$$\Rightarrow H = \frac{3}{8} + \frac{1}{2} + \frac{1}{2} + \frac{3}{8} = 1\frac{3}{4} \text{ bit}$$

4. Beispiel:

Das Nachrichtenalphabet sei das gewöhnliche Alphabet inclusive Sonder- und Leerzeichen. Die Wahrscheinlichkeiten einzelner Buchstaben werden durch umfangreiche Zählungen an Texten ermittelt. (→ Folie/Kopie)
Ergebnis der Studie aller Wahrscheinlichkeiten ist eine mittlere Wahrscheinlichkeit von

$$H = 4.11 \text{ bit}$$

Die mittlere Wortlänge:

5. Beispiel:

Die Zeichen mit gleichen Wahrscheinlichkeiten aus Beispiel 2:

3 Zeichen : x, y, z

$$p_x = \frac{1}{2} \qquad p_y = p_z = \frac{1}{4}$$

	Codiert wie folgt:	Wortlänge:
x	1	$1=l_x$
y	01	$2=l_y$
z	00	$3=l_z$

z.B. folgender Code: 01 1 1 00 01 1 steht für yxxzyx. Der Code ist so gewählt, daß auch unterschiedliche Code-Längen noch immer zum richtigen Ergebnis führen.

Def.:

Die mittlere Wortlänge L ist definiert durch

$$L := \sum_i p_i l_i$$

dabei ist l_i die Länge des i-ten Zeichens. Die Einheit ist wieder das bit.

5b. Beispiel aus Beispiel 2:

mittlere Wortlänge ist:

$$L = \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{4}\cdot 2$$
$$= \frac{1}{2} + \frac{1}{2} + \frac{1}{2}$$
$$= 1\frac{1}{2}\,\text{bit}$$

Der Code ist offensichtlich sehr optimal gewählt, da der mittlere Abstand gleich der mittleren Wahrscheinlichkeit ist. Er enthällt somit keine Redundanzen. So kommen wir auch auf die Definition einer Redundanz:

Def.:

Die ***Redundanz*** *R* eines Codes ist definiert als Differenz zwischen dem mittleren Abstand und der mittleren Länge eines Codes:

$$R = L - H$$

Eineit ist auch hier wieder das bit.

Übungsaufgabe 1:

Wie hoch ist der Informationsgehalt eines einzelnen Zeichens bei der Verwendung einer k-Stelligen Dezimalzahl.
D.h. Code besteht aus 10 Zeichen (Dezimalzahlen) mit Gleichverteilung.

$n = 10^k$ k-Stelliger Code auf der Basis von Zehn verschiedenen Zeichen.

$\Rightarrow p = \frac{1}{n} = \frac{1}{10^k}$ (Wahrscheinlichkeit des Auftretens einer Kombination des k-Stelligen Codes)

$$\Rightarrow h = -\text{ld}\frac{1}{10^k} = -\left(\text{ld}1 - \text{ld}10^k\right) = \text{ld}10^k = k\cdot\text{ld}10$$

AUCH:

$$p = \frac{1}{10} \Rightarrow h = -\text{ld}\frac{1}{10} = \text{ld}10$$

p ist nun die Wahrscheinlichkeit für das Auftreten eines Zeichens. Dieses passiert aber *k*-mal. Daher : $k\cdot h = k\cdot\text{ld}10 \approx k\cdot 3{,}32$

Übungsaufgabe 2:

4 Zeichen A,B,C,D

$$p_a = \frac{1}{2}; p_b = \frac{1}{4}; p_c = p_d = \frac{1}{8}$$

Jeweiliger Informationsgehalt:

$$h_a = -\text{ld}\frac{1}{2} = 1 \quad ; h_b = -\text{ld}\frac{1}{4} = 2 \quad ; h_c = h_d = -\text{ld}\frac{1}{8} = 3$$

Übungsaufgabe 3:

Zeichen A,B,C,D

$$p_a = \frac{1}{2}; p_b = \frac{1}{4}; p_c = p_d = \frac{1}{8}$$

$$H = \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{8}\cdot 3 + \frac{1}{8}\cdot 3 = 1\frac{3}{4}$$

Übungsaufgabe 4:

gegeben sind 3 Zeichen: x,y,z

mit p_x=0.7 p_y=0.2 p_z=0.1

Codiert wurde mit :

x	1	l_x=1
y	01	l_y=2
z	00	l_z=2

	p_i	h_i=-ld p_i	$p_i h_i$	l_i	$p_i l_i$
x	0.7	0.514573172	0.360201221	1	0.7
y	0.2	2.321928095	0.464385619	2	0.4
z	0.1	3.321928095	0.332192809	2	0.2

Datenverdichtung (Huffman-Code, Arithmetische Codierung)

Ziel: Konstruktion von möglichst redundanzarmen Code durch Optimierung der Bit-Längen des Codes.

A: Der Huffman-Code:

Verfahren:

1. Es wird schrittweise ein Codebaum aufgebaut.
2. Die zwei Zeichen mit der geringsten Auftrittswahrscheinlichkeit werden zusammengefasst und dann als einzelnes Zeichen behandelt, dessen Wahrscheinlichkeit gleich der Summe der Einzelwahrscheinlichkeit der beiden Zeichen ist.
3. Das Verfahren wird so oft wiederholt, bis alle Zeichen zu einem einzigen zusammengefasst wurden, das die Wahrscheinlichkeit 1 hat.

Es läßt sich zeigen, daß dieses Verfahren für die Codierung einzelner Zeichen von keinem Verfahren im Hinblick auf die Redundanz übertroffen wird.

Beispiel:

5 Zeichen {a,b,c,d,e} [Das Zielalphabet ist binär]
mit den Wahrscheinlichkeiten:

	p
a	0.40
b	0.20
c	0.18
d	0.11
e	0.11

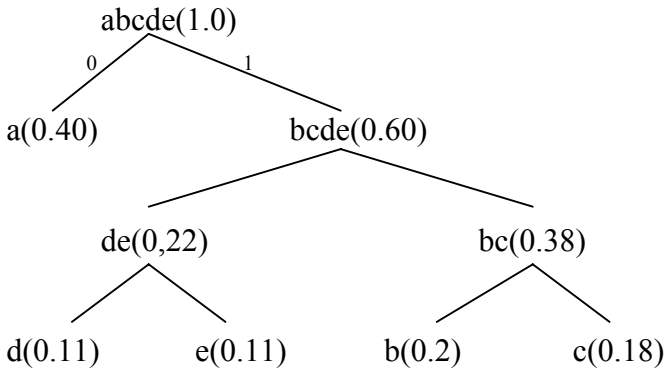
	p
a	0.40
de	0.22
b	0.20
c	0.18

	p
a	0.40
bc	0.38
de	0.22

	p
bcde	0.60
a	0.40

	p
abcde	1.0

=> Der Codebaum:



	p	Code	*l*	*p*·*l*	-ld *p*	*p*
a	0.40	0	1	0.40	1.322	0.52
b	0.20	110	3	0.60	2.322	0.46
c	0.18	111	3	0.54	2.474	0.44
d	0.11	100	3	0.33	3.184	0.34
e	0.11	101	3	0.33	3.184	0.34
				L=2.20		*H*=2.14

Die Redundanz dieses Codes ist somit $R = L - H = 2.20 - 2.14 = 0.06$

Bemerkungen

1. Die Zeichen mit der größten Wahrscheinlichkeit bekommen den kürzesten Code.
2. Es wird vorausgesetzt, daß die Wahrscheinlichkeiten der Zeichen von der Übertragung feststehen. Ansonsten kann man wärend der Übertragung die Wahrscheinlichkeiten ermitteln und dann gemäß dieser Wahrscheinlichkeiten wärend der Übertragung den Code ändern (-> adaptiver Huffman-Code). Dafür eignet sich aber der arithmetische Code besser.
3. Anwendung auf Bilder, Audio und Texte.

B: Der arithmetische Code:

Der arithmetische Code ist besser geeignet für adaptive Änderungen. Beim Huffman-Code muß der gesamte Codebaum neu generiert werden, wenn sich die einzelnen Wahrscheinlichkeiten ändern.

Die Idee:

Eine Nachricht wird durch ein Teilintervall von [0;1[dargestellt. Je länger die Nachricht, je kürzer wird das Teilintervall. Jedes zu übertragende Zeichen verkleinert dieses Intervall entsprechend seiner Auftrittswahrscheinlichkeit. Dabei verkürzen häufig auftretende Zeichen das Intervall weniger.

Beispiel:

Das Alphabet: {a,e,i,o,u,!}

	p	Intervall
a	0,2	[0 ; 0,2[
e	0.3	[0,2 ; 0,5[
i	0.1	[0.5 ; 0.6[
o	0.2	[0.6 ; 0.8[
u	0.1	[0.8 ; 0.9[
!	0.1	[0.9 ; 1.0[

Die Nachricht sei : „eaii!“

1. Schritt: Zu beginn ist das Intervall [0 ; 1[
2. Schritt: Erstes Zeichen ist ein e: => Verkleinert das Intervall auf [0,2 ; 0,5[
3. Schritt: Zweites Zeichen ist ein a: =>Verkleinert das jetzige Intervall auf [0,2 ; 0,26[
4. Schritt: Drittes Zeichen ist ein i: => Intervall nun [0,23 ; 0,236[
5. Schritt: Viertes Zeichen ist ein i: => Intervall nun [0,233 ; 0,2336]
6. Schritt: Fünftes Zeichen ist ein ! => Intervall nun [0,23354 ; 0,2336[

Also insgesamt:

	[0 ; 1[
e	[0,2 ; 0,5[
a	[0,2 ; 0,26[
i	[0,23 ; 0,236[
i	[0,233 ; 0,2336[
!	[0,23354 ; 0,2336[

Dabei findet folgende Formel Anwendung, um die neuen Intervalle zu berechnen:

O_{alt}	: alte Obergrenze
U_{alt}	: alte Untergrenze
l_{alt}	: Oalt – Ualt (Länge des Intervalles)
$[U_x,O_x[$	: Das den Zeichen x zugeordnete Intervall

neue Untergrenze:

$$U_{neu} = U_{alt} + l_{alt} \cdot U_x$$

neue Obergrenze:

$$O_{neu} = U_{alt} + l_{alt} \cdot O_x$$

Beispiel zur Anwendung der Formel:

Alte Intervall: [0,23 ; 0,236[
nächstes Zeichen „i“:

$l_{alt} = 0{,}006$
$U_x = 0{,}5$ => $0{,}23 + (0{,}006 * 0{,}5) = 0{,}233 = U_{neu}$
$O_x = 0{,}6$ => $0{,}23 + (0{,}006 * 0{,}6) = 0{,}2336 = O_{neu}$

Altes Intervall: [0,233 ; 0,2336[
nächstes Zeichen „!“:

$l_{alt} = 0{,}0006$
$U_x = 0{,}9$ => $0{,}233 + (0{,}0006 * 0{,}9) = 0{,}23354 = U_{neu}$
$O_x = 1{,}0$ => $0{,}233 + (0{,}0006 * 1{,}0) = 0{,}2336 = O_{neu}$

Was macht nun der Emfänger mit einer solchen Nachricht? Dekodieren natürlich!

$$I := [0{,}23354 ; 0{,}2336[$$

Das Intervall liegt mitten in dem Bereich, der in der Urskala für das „e“ vergeben war, da dort ja schließlich beim Codieren auch alles begann. Damit weiß der Empfänger aber schon, daß es mit dem „e“ losgeht.
Nun der folgende Buchstabe: Da ja das Intervall von „e“ weiter „geschrumpft“ wurde, muß nun das einzige gültige Intervall auf Position innerhalb von „e“ das Intervall *I* mit überdecken. Dies trifft nur für das „a“ zu. Die anderen Buchstaben funtkionieren dann analog.

Bemerkungen

1. Es reicht, irgendeine Zahl aus *I* zu übermitteln. Dann muß man allerdings eine Endekennung mitschicken, damit keine Mehrdeutigkeiten entstehen.
2. In der Praxis wird das Intervall selbstverständlich binär übermittelt.
3. Die Anwendung kann man auf der Folie ersehen.

Übung:

a [0 ; 0,2[
aa [0 ; 0,04[
aaa [0 ; 0,008[

Die Rekonstruktion fürt zu : a, aa, aaa, aaaa, ...
Es ist also erforderlich ein Endekennzeichen zu senden also => z.b. aaa!

Beispiel 2:

e [0,2 ; 0,5[
i [0,35 ; 0,38[
a [0,35 ; 0,356[
u [0,3548 ; 03554[

Ein paar Worte zu den Bildern auf der Kopie:

Ein Bild besteht aus 256 x 256 Bildpunkten (Pixel, Picture Element)
Je Pixel werden 8 Bit Farbinformationen verarbeitet, also 256 Graustufen
=> jedes Bild enthällt 256*256*8 Bit = 65.536 Byte Daten.

C: Datenkomprimierung:

Ziel: Platzeinsparung und sparen von Übertragungszeit
Die meisten Codes haben einen hohen Grad an Redundanz. Schaltet man diese Redundanz aus, kann man platzsparender Codieren. (->Huffman Code). Es gibt aber noch andere Techniken:

- Textdateien durch Huffman-Code (20%-50%)
- Bilder als Rasterdateien (Pixel) -> homogene Flächen mit gleicher Farbe bieten eine Mustererkennung oder Lauflängencodierung an.
- Dateien für die digitale Darstellung von akustischen Signalen -> umfangreiche Wiederholungen und akustische Muster.

Methoden:

- Lauflängencodierung
- Codieren mit variabler Länge (Huffman)

Die Lauflängencodierung:

Der einfachste Typ von Redundanz in einer Datei sind lange Folgen sich wiederholender Zeichen (Läufe, Runs).

Beispiel:

AAAABBBAABBBBBCCCCCCCCDABCBA

Kompakter:

Erst wird der Wiederholungsfaktor genant, dann das Zeichen. Wiederholt sich das Zeichen nicht, gibt es keine Zahl:

4A3B2A5B8CDABCBA

Das verlangt, daß in dem Text keine Zahlen vorkommen.

Verfeinerung für binäre Daten:

Bei binären Daten wechseln sich Läufe zwischen 0 und 1 ab. => Das Speichern der Zeichen selbst ist nicht mehr nötig. Es reicht, die Faktoren zu speichern, da jeder wechsel bei den Faktoren auch einen Wechsel der Zeichen meint.

(->Folie) Die Rastardarstellung des Buchstaben ‚q' zeigt, daß die nebenstehende Tabelle ausreicht, um den Buchstaben zu rekonstruieren.

Wählt man nun eine 6-Bit Darstellung der Zahlen($2^6 = 64$ verschiedene Zahlen) , wird das Bitmuster mit $63 \cdot 6 = 378$ Bits statt vorher mit 969 Bits gespeichert.

Zurück zu dem Beispiel mit den Buchstaben:

Die ursprüngliche Datei bestand nur aus Buchstaben, wogegen die komprimierte Datei aus Zahlen und Buchstaben besteht. Das bedeutet eine Vergrößerung des Alphabets, was nicht günstig ist. Daher sollte man das Alphabet der Urdatei verwenden, um die Wiederholungen zu zählen, da dann weniger Bits benötigt werden, um das Alphabet zu codieren.
Das Zeil ist also ein möglichst kleines Alphabet:

DACBBAEC...

Wir haben nun jede Zahl durch einen Buchstaben ersetzt, wobei seine Position in dem Alphabet repräsentativ für die Wertigkeit des Faktors stehen soll. Nachteilig ist nun, daß es nicht mehr möglich ist, zwischen Buchstabe und Faktor zu unterscheiden. Daher muß man eine Kennung hinzufügen, die dann angibt, ob es ein Faktor ist, oder ein Code. Als Escape-Zeichen (=Kennung) verwenden wir mal das ‚Q', da wir behaupten, es komme in unserem Alphabet nicht vor:

QDAQCBQBAQECQHC...

Da aber eine Codierung BBB -> QCB nichts bringt und eine Codierung von AA -> QBA sogar eine Verschlechterung darstellt, sieht unsere komprimierte Datei nun so aus:

=> QDABBBAAQECQHC...

Das Verfahren der Lauflängencodierung lohnt sich aber nur auf binärer Ebene.

Fehlererkennede und –korrigierende Codes

Ab jetzt verwenden wir nur noch Codes ohne variable Wortlänge.
Bei einer Übertragung treten immer wieder Fehler auf. Z.B. das Knacken in der Telefonleitung.

Beispiel:

Modem mit 9600 $\frac{\text{bit}}{\text{sec}}$ (langsam, aber immerhin!)

Knacken hat eine Dauer von $\frac{1}{100}\text{sec.} \Rightarrow 100$ Bit werden gestört.

Dabei gibt es :
- Einzeln auftretende Fehler und
- Fehlerbündel

Gesucht ist nun ein Maß für die Störsicherheit eines Codes.

A) Der Hammingabstand (Hammingdistanz) zur Fehlererkennung

Def.: Der Hammingabstand zweier Codewörter ist die Anzahl der Stellen, in welchen sich diese zwei Codewörter unterscheiden.

Beispiel: Alphabet mit Codierung : {a, e, i, o}

1.			2.		
	a	00		a	000
	e	01		e	011
	i	10		i	101
	o	11		o	110

Der Hammingabstand:

a -> e:	1	2
a -> o:	2	2

Tabelle für die Paarweisen Hammingabstände:

1.

	a	e	i	o
a	-	1	1	2
e	1	-	2	1
i	1	2	-	1
o	2	1	1	-

H.abstand: 1

2.

	a	e	i	o
a	-	2	2	2
e		-	2	2
i			-	2
o				-

H.abstand: 2

An der Achsendiagonalen ist das Ergebnis gespiegelt, was bei näherer Betrachtung auch zu erwatrten war.

Vorzug von Code 2 gegenüber Code 2:
Es müssen bereits zwei Bit umkippen, damit ein neuer Code entsteht. Kippt nur ein Bit, ist das als Fehler zu erkennen, da es das Codewort im Alphabet gar nicht gibt.

Nachteilig allerdings
ist, das Code 2 längere Codeworte benötigt.

Daraus ergibt sich das Problem des **Kompromiss**es zwischen **Datensicherheit** und **Redundanz**.

Def.: Der Hammingabstand eines Codes (Alphabet) ist der kleinste Hammingabstand zwischen Wörtern dieses Codes.

B) Fehlererkennende Codes:

Bei einem Code mit Hammingabstand $d > k$ können Störungen bis zu k Bits pro Codewort erkannt werden.

Ziel: Vergrößerung des Hammingabstandes.

1. Möglichkeit: Sende jedes Codewort zweimal. Der neuer Code hat den Hammingabstand von 2*d*, doch ist der Code auch plötzlich doppelt so lang!

2. Möglichkeit: Hänge jedem Wort ein „**Paritätsbit**“ an.

$$\text{Parität} \begin{cases} 1, \text{ falls Anzahl einsen gerade} \\ 0, \text{ falls Anzahl einsen ungerade} \end{cases}$$

Beispiel:

a	00	->	00<u>1</u>
e	01	->	01<u>1</u>
i	10	->	10<u>1</u>
o	11	->	11<u>0</u>

Bei einem Code mit $d = 1$ erhöht man diesen auf $d = 2$ durch Erhöhung der Wortlänge um eins.

Einschub

direkte Computerverbindung:

Geschwindigkeiten : $10^7 - 10^8 \frac{\text{bit}}{\text{sec}} \approx 1 - 10 \frac{\text{MByte}}{\text{sec}}$

Fehlerquote (niedrig): $1:10^{12}$ bis $1:10^{13}$

Also ein Fehler auf 10^{13} gesendeten Bits

Telefon:

Geschwindigkeiten : ca. $10^4 \frac{\text{bit}}{\text{sec}}$

Fehlerquote (hoch): $1:10^5$

Vergleichende Größenordnung sind 8 Zehnerpotenzen.

3. Möglichkeit: Polynomcodes oder CRC-Codes (Cyclic redundancy Codes) (Sehr wichtig für die Praxis!)

Exkurs Polynome:

Polynome sind Funktionen der Form:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

n bezeichnet man als den Grad des Polynoms.

z.B. $p(x) = 1 + x^2 \Rightarrow n = 2;\ a_0 = 1;\ a_1 = 0;\ a_2 = 1$

$q(x) = -3 + x^4 + 7x^7$ der Grad ist $n = 7$

Addition und Subtraktion von Polynomen:

$$p(x) + q(x) = -2 + x^2 + x^4 + 7x^7$$

Polynomdivision:

$$\left(x^3+2x^2+4x+5\right):\left(x+1\right)=x^2+x+3+\frac{2}{\left(x+1\right)}$$

$$\begin{array}{r} -\left(x^3+\ x^2\right) \\ \hline x^2+4x \\ -\left(x^2+\ x\right) \\ \hline 3x+5 \\ -\left(3x+3\right) \\ \hline 2 \end{array}$$

Bitfolgen werden als Polynome aufgefasst, mit Koeffizienten aus $\{0, 1\}$.

Polynom:

$$p(x)=a_n x^n+a_{n-1}x^{n-1}+\ldots+a_2x^2+a_1x+a_0$$
$$n=\text{Grad}\left(p(x)\right)\quad a_i\in R$$
z.B. $p(x)=3x^4+2x^2-7$

Bei Bitfolgen:

z.B. $\left(0,1,1,0\right)\rightarrow 0\cdot x^3+1\cdot x^2+1\cdot x^1+0\cdot x^0=x^2+x$

allgm.: $\left(a_n, a_{n-1}, \ldots., a_2, a_1, a_0\right)\qquad a_i\in\{0,1\}$

$\rightarrow a_n x^n+a_{n-1}x^{n-1}+\ldots+a_2x^2+a_1x+a_0$

Die Berechnungen werden mit MODULO 2 ausgeführt, da dann
1 + 1 = 0; 1 = -1; usw.

z.B. $\left(x^2+x\right)+\left(x^3+x^2\right)$
$=x^3+2x^2+x=$ (da mod 2) x^3+x

Polynomcodemethode:

Sender und Empfänger legen ein erzeugendes Generatorpolynom

$$G(x) \text{ mit } \text{Grad}\left(G(x)\right)=r$$

fest.
Mit diesem Generatorpolynom wird eine Prüfsumme berechnet, die mir übertragen wird. Dann könnte der Empfänger seine eigene Prüfsumme berechnen und damit durch Vergleich der Prüfsumme eventuelle Fehler feststellen.
Idee dabei: Das übertragene Wort (Nachricht+Prüfsumme) soll durch $G(x)$ teilbar sein. Ist das Ergebnis beim Empfänger nicht durch $G(x)$ teilbar, dann liegt ein Fehler vor.

Sei **$G(x)$** das Generatorpolynom und
$M(x)$ das dem ursprünglichen Codewort entsprechende Polynom, genannt „Messagepolynom“, mit den Graden
Grad($G(x)$) = r und
Grad($M(x)$) = m.

Der Algorithmus:

1.) Bilde $x^r \cdot M(x)$, das heißt, man hängt r Nullen an des Codewort an. => Codewort hat $m + r$ Bits.
2.) Schreibe $x^r \cdot M(x)$ in der Form $x^r \cdot M(x) = \left(Q(x) \cdot G(x)\right) + T(x)$ (immer in Modulo 2), mit geeigneten Polynomen $Q(x)$ und $T(x)$ mit dem Grad($T(x)$) < r. (Euklidischer Algorithmus)
3.) Übertrage das Codewort, das dem Polynom $\left(x^r M(x)\right) - T(x)$ entspricht.

Ein Beispiel:

$G(x) = x^3 + x^2 + 1 \qquad r = \text{Grad}\left(G(x)\right) = 3$

zu Übertragen sei : $\{0,1,1,0\} \leftrightarrow M(x) = x^2 + x$

zu 1) $x^r M(x) \Rightarrow x^3 \cdot \left(x^2 + x\right) = x^5 + x^4 \leftrightarrow 0110|000$

zu 2) Dividiere obiges durch $G(x)$ und ermittle den Rest:

$$\left(x^5 + x^4\right) \div \left(x^3 + x^2 + 1\right) = x^2$$
$$\underline{-\left(x^5 - x^4 + x^2\right)}$$
$$-x^2 = T(x)$$

Ergebnis : $Q(x) = x^2$

$T(x) = x^2$

$$\underbrace{x^5 + x^4}_{x^r M(x)} = \underbrace{x^2}_{Q(x)} \cdot \underbrace{(x^3 + x^2 + 1)}_{G(x)} + \underbrace{x^2}_{T(x)}$$

zu 3) $x^r M(x) - T(x) = x^5 + x^4 - x^2 = x^5 + x^4 + x^2 = \{0110|100\}$

(wegen Mod 2!)

Es können alle Fehler erkannt werden, deren zugehöriges Polynom nicht durch $G(x)$ teilbar ist. Angenommen wir hätten eine Störung $S(x)$ und beim Empfänger kommt statt $x^r M(x) - T(x) \rightarrow x^r M(x) - T(x) + S(x)$ an.
Wenn der Empfänger druch $G(x)$ dividiert, erhällt er den Rest:

$$\left(x^r M(x) - T(x) + S(x)\right) \div G(x) = \begin{cases} 0 \Rightarrow S(x) \text{ durch } G(x) \text{ teilbar} \Rightarrow \text{kein Fehler} \\ 1 \Rightarrow S(x) \text{ durch } G(x) \text{ nicht teilbar} \Rightarrow \text{Fehler} \end{cases}$$

$$\left(x^r M(x) - T(x) + S(x)\right) \div G(x) = 0$$

Z.B. Das Wort 0110100 wurde gestört und der Empfänger erhällt:

$1110100 \leftrightarrow x^6 + x^5 + x^4 + x^2$

Division durch $G(x)$:

$$\left(x^6 + x^5 + x^4 + x^2\right) \div \left(x^3 + x^2 + 1\right) = x^3 + x + \frac{x^2 + x}{x^3 + x^2 + 1}$$

Da der Rest nicht Null ist => es ist ein Fehler aufgetreten.

Codes, die in der Praxis verwendet werden:

Folgende standarisierte Generatorpolynome gibt es unter anderen:

CRC-12= $x^{12} + x^{11} + x^3 + x^2 + x + 1$ für Bitfolgen der Länge 6
CRC-16= $x^{16} + x^{15} + x^2 + 1$ für Bitfolgen der Länge 8
CRC-CCITT= $x^{16} + x^{12} + x^5 + 1$ für Bitfolgen der Länge 8

Die letzten beiden werden bei vielen Modems benutzt, aber dort meistens zusammen mit einer LZW-Komprimierung.

Welche Fehler sind durch die CRC-Methode zu erkennen:

1. Es können alle Einzelbitfehler erkannt werden, falls $G(x)$ zwei oder mehr Summanden enthällt. Dann kann $S(x)$ nicht mehr ohne Rest durch $G(x)$ geteilt werden. $\left[i\text{tes Bit fehlerhaft} : S(x) = x^i\right]$
2. Ein CRC-Code mit r Prüfbits (= dem Grad r) erkennt alle Fehlerbündel der Länge $\leq r$
3. Bei einem Fehlerbündel der Länge $r + 1$ ist die Wahrscheinlichkeit, nicht erkannt zu werden $\approx \frac{1}{2^{r-1}}$

   (CRC-16: Fehlerbündellänge ist $17 \Rightarrow w = \frac{1}{2^{15}} \approx 0{,}00003 = 0{,}003\%$)
4. Bei Fehlerbündel der Länge $> r + 1$ ist die Wahrscheinlichkeit, nicht erkannt zu werden $= \frac{1}{2r}$ (CRC-16: $w = \frac{1}{2 \cdot 16} = \frac{1}{32} \approx 0{,}03 = 3\%$)

CRC-16 und CRC-CCITT entdecken:

- alle Einzel- und Doppelfehler
- alle Fehler mit ungerader Bitzahl
- alle Fehler ≤ 16 Bits
- 99,997% aller 17-Bit-Fehlerbündel
- 99,998% aller 18-Bit-Fehlerbündel

Die Realisierung der Methode ist meist mit Hardware verbunden
(→Schieberegisterschaltung)
(Nachzulesen bei Tanenbaum, Computernetzwerke)

Übung:

11010110 ist mit dem CRC-16 $= \left(x^{16} + x^{15} + x^2 + 1\right)$ zu übertragen. Was wird als Prüfsumme angehängt?

1) $x^7 + x^6 + x^4 + x^2 + x$ ist das Polynom zur Bitfolge
2) $x^{23} + x^{22} + x^{20} + x^{18} + x^{17} = 11010110\underbrace{0.....0}_{16\ \text{Stellen}}$ ist das erweiterte Polynom
3) Polynomdivision (kann jeder selber machen!!!)

$$\left(x^{23} + x^{22} + x^{20} + x^{18} + x^{17}\right) \div \left(x^{16} + x^{15} + x^2 + 1\right) = x^7 + x^4 + x^3 + x + 1$$
$$\text{Rest} = x^{15} + x^9 + x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$$

Daraus folgt, daß folgende Bitkonstellation gesendet werden muß:

11010110|1000001011110111

C) Fehlererkennende Codes am Beispiel des Hammingcodes

Anwendung bei Simplexkanälen, also einer Kommunikation in nur eine Richtung, wobei daher keine Möglichkeit besteht, den Sender zu einer erneuten Sendung aufzufordern.
Daher sollen zusätzlich zur Fehlererkennung auch noch die Fehler korrigiert werden können. Zur Realisierung ist der Hammingabstand ein nützliches Instrument.

Für die Korrektur von *k*-Fehlern (*k* gestörten Bits) ist ein Hammingabstand von mindestens $2k + 1$ nötig. Bei *k* Störungen liegt dann das ursprüngliche Codewort immer noch dichter an dem falschen Wort, als irgend ein anderes Codewort aus dem Alphabet.

Beispiel:

```
100000      Abst. = 5 = 2k + 1 => 2 Bitfehler werden erkannt und
011011      sind korrigierbar.
```

Es sollen *m*-Bits ursprüngliche Informationen übertragen werden und *r* Prüfbits angehängt werden. (d.h. Gesamtzahl der Bits : $m + r$ Bit).
Aus der Ungleichung $m + r + 1 \le 2^r$ erhällt man eine untere Schranke für die Anzahl *r* der Prüfbits, wenn ein einzelner Fehler korrigiert werden soll.
Konkret :

$m = 1$ => $r = 2$
$m = 2,3,4$ => $r = 3$
$m = 5,..,11$ => $r = 4$
$m = 12,..,26$ => $r = 5$

Beweis:

1) Anzahl verschlüsselter Codewörter der Länge $m = 2^m$
2) Jedes Codewort der Länge $m + r$ hat $m + r = n$ „Nachbarn“ mit dem Hammingabstand 1
   => Anzahl Bitmuster, die von einem gegebenen Bitmuster einen Abstand ≤ 1 haben sind $n + 1$
3) Anzahl der verschiedenen Bitmuster der Länge n : 2^n

aus 1.2.3 folgt:

$$2^m \cdot (n+1) \leq 2^n$$
$$\Leftrightarrow 2^m \cdot (m+r+1) \leq 2^{m+r}$$
$$\Leftrightarrow (m+r+1) \leq 2^r$$

Für den Hammingabstand wird diese untere Schranke angenommen. Die Bits der Codewörter werden nummeriert. Die Bits mit den Nummern $2^r (= 1,2,4,8,16,..)$ sind die Kontrollbits und die übrigen sind die Datenbits.

P1	P2	D1	P3	D2	D3	D4	.....	.....	.....	.....	.....
Nr. 1	2	3	4	5	6	7	..				

In der Dualdarstellung haben die Nummern der Prüfbits eine 1 an erster Stelle und ansonsten nur noch Nullen.
Jedes Prüfbit wir mit der Parität der zugehörigen Datenbits gefüllt.

$$P_1 \leftrightarrow D_1, D_2, D_4$$
$$P_2 \leftrightarrow D_1, D_3, D_4$$
$$P_3 \leftrightarrow D_2, D_3, D_4$$

Parität bedeutet: Anzahl einsen gerade, dann ist Parität null, ansonsten eins.

Funtkioniert das nun auch?

Ein Datenbit habe die Nr. *k*. Schreibe *k* als Summe von Zweierpotenzen,
z.B. $k = 7 = 4+2+1$, dann gehört dieses Datenbit zu allen Prüfbits, deren Nummer hier auftaucht, also P_3, P_2, P_1.
D_4 gehört zu ebenfalls zu P_3, P_2, P_1, D_1 hat aber die Nummer 3 = 2+1, also P_2, P_1.

Beispiel:

0110 soll übertragen werden:

		0		1	1	0
Nr. 1	2	3	4	5	6	7

1	1		0			
Nr. 1	2	3	4	5	6	7

=> 1100110 als zu sendendes Bitmuster

Was macht nun der Empfänger:

- prüft, ob Daten- und Prüfbits zusammenpassen
- falls nicht (z.B. *k*-tes Prüfbit passt nicht), so addiert er diese Zahl *k* zu einem **Korrekturfaktor**

Am Ende enthällt der Korrekturfaktor etweder eine Null (kein Fehler), oder die Nummer des gestörten Datenbits, welches nur noch invertiert werden muß.

Beispiel:

Statt 1100110 (s.o.) wird 1110110 empfangen.

Rekonstruktion des Codewortes ergibt:
$P_1 = 1$: Parität (D_1, D_2, D_4) = Parität (1, 1, 0) = 0
=> Korrekturfaktor = 1 (+1)
$P_2 = 1$: Parität (D_1, D_3, D_4) = Parität (1, 1, 0) = 0
=> Korrekturfaktor = 3 (+2)
$P_3 = 1$: Parität (D_2, D_3, D_4) = Parität (1, 1, 0) = 0
=> Korrekturfaktor = 3 (+0)

Also muß offensichtlich die Stelle 3 gestört sein. Eine Invertierung ergibt den tatsächlich richtigen Code.

Hamming-Code: Die untere Schranke wird angenommen:

$$m + r + 1 = 2^r$$

$$m = 4 \quad r = 3 \Rightarrow 4 + 3 + 1 = 2^4 \Rightarrow 8 = 8$$

Der Hamming-Code kann leider nur einfache Bitfehler korrigieren. Jedoch mit einem Trick kann man auch die häufigeren Fehlerbündel eliminieren. Dazu ordnet man die Zu sendenden Daten in einer Matrix an und versendet nicht die Zeilen einzelnen, sondern fast die Spalten zusammen und sendet diese. Dadurch ist ein Fehlerbündel bei dieser Sendung zwar immer noch fatal, aber nach der Rückanordnung in einer Matrix, kann man dann die Zeilen wieder interpretieren, wo ja das Fehlerbündel jeweils nur ein Bit gestört hat, was erkenn- und korrigierbar ist.

D) Numerische und Alphanumerische Codes

A) Numerische Codes (Darstellung von Zahlen)

Zählcode (Wahlzeichen bei der Telephonie)

1:	1000000000
2:	1100000000
3:	1110000000
:	
0:	1111111111

1 aus 10 Code (Wahlzeichen bei der Telephonie)

1:	1000000000
2:	0100000000
3:	0010000000
:	
0:	0000000001

Tetradische Codes (4 Stellig)

- **BCD** (Binary Coded Dezimals, gepackte Zahlendarstellung)

0	0000	Tetraden
1	0001	
2	0010	
:		
9	1001	
10	1010	Pseudotetraden
11	1011	
:		
15	1111	

Beispiel:

75 = 0111|0101

Jedes Bit hat eine Wertigkeit, die mit der Zweierpotenz des Binärsystems korrespondiert, also 1, 2, 4, 8.

Der **Aikencode** verwendet nicht die binäre Darstellung, sondern siedelt Pseudotetraden zwischen der Zahl 4 und 5 an. Dadurch ergibt sich eine Bitwertigkeit von 1, 2, 4, 2, so daß dieser Code auch als 1-2-4-2 Code bezeichnet wird.
Der Aikencode ist zusätzlich ein sogenannter *komplementärer Code*, d.h. mit einer Tetrade $t_1t_2t_3t_4$ ist das Einerkomplement $\bar{t}_1\bar{t}_2\bar{t}_3\bar{t}_4$ wieder eine Tetrade des Codes.

z.B. 7 = 1101 1-Komplement: 0010 = 2

Dieser Umstand soll günstig für die technische Realisierung sein.

Der **Excess-3-Code** ist ebenfalls ein *Komplementätcode*

z.B. 2 = 0101 1-Komplement: 1010 = 7

B) Alphanumerische Codes

BCD-Code:
7 Stellig (7-Bit) von IBM, Vercodung von Ziffern und *Groß*buchstaben.

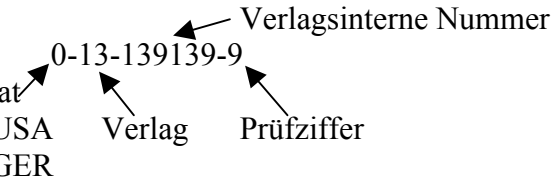
EBCDI-Code: (Extended Binacy Coded Dezimal Interchange Code)
- enthällt zusätzlich zum BCD-Code noch *Klein*buchstaben
- findet sich vornehmlich auf Großrechenanlagen der IBM (AS-400)

ASCII-Code: (American Standard Code for Information Interchange)
- sehr weit verbreitet (PC's , Unix)
- ursprünglich 7-Stellig
- diese 7 Stellen sind normiert von der *ISO* (International Standarisation Organisation) genormt.
- falls 8-Bit genutzt werden, werden dort Sonder- und Graphikzeichen abgelegt.

ISO-10.646-Standard (1993)
- 16-Bit Code
- enthällt als Teilmenge den ASCII-Code
- weitere Sonderzeichen (Japanisch, Kyrillisch,...)
- 65.536 verschiedene Zeichen möglich

Der ISBN-Code: (Internationale Standard Buch Nummer)
- 10 Stellig

Beispiel:



Die Zehn Stellen seien $Z_{10}Z_9Z_8...Z_1$
Eine ISBN ist zulässig, wenn die gewichtete Quersumme

$$1 \cdot Z_1 + 2 \cdot Z_2 + 3 \cdot Z_3 + ... + 10 \cdot Z_{10}$$

durch 11 Teilbar ist. Die Prüfziffer ist Z_1

Zulässigkeitsprüfung:

$$0 \cdot 10 + 1 \cdot 9 + 3 \cdot 8 + ... + 9 \cdot 2 + 9 \cdot 1 = 143$$

Ermittlung der Prüfziffer Z_1:

$$0 \cdot 10 + 1 \cdot 9 + 3 \cdot 8 + ... + 9 \cdot 2 = 134$$

$$134 \div 11 = 12 \text{ Rest } 2$$

$$\Rightarrow Z_1 = 11 - 2 = 9$$

Die 10 wird als X geschrieben.

Da 11 eine Primzahl ist, ist sie bestens hierfür geeignet, da sie gut mathematische Eigenschaften besitzt.

Kryptographie

Einleitung, Überblick, Terminologie

Ziele der Kryptographie

1. **Geheimhaltung** : Vertauliche Kommunikation
   Lösungen: - organisatorisch (Boten, Einstufung als Verschlußsache)
   - physikalisch (Tresor, Versiegeln, Geheimtinte)
   - kryptographische Maßnahmen (Verschlüsselung von Nachrichten, Empfänger braucht einen Schlüssel zum entschlüsseln)
   symetrische Verfahren:
   Sender und Empfänger einer Nachricht benutzen den gleichen Schlüssel.
   asymetrische Verfahren:
   Sender und Empfänger benutzen unterschiedliche Schlüssel (einen zum Versenden und einen zum Entschlüsseln)
   public Key secret Key
   -> daher auch Public-Key-Verfahren

2. **Autentifikation** (Autentikation)
   Wie kann ich mich einem anderen gegenüber zweifelsfrei ausweisen? (tägliches Leben: Stimme, Aussehen, Personalausweis, Handschrift)
   Rechner:
   Paßwort, Karte+PIN, in der Kryptographie durch geheimes Wissen
   -> *Teilnehmeridentifikation*
   Wie kann ich sicher sein, daß die Nachricht auch wirklich von dem angegebenen Sender stammt?
   -> *Nachrichtenidentifikation*
   Echtheitsmerkmale von Dokumenten
   (Silberstreifen, Wasserzeichen, eingeschweißte Dokumente)

3. **Anonymität**
   Kann man auch bei elektronischer Kommunikation seine Privatsphäre schützen? Nicht nur der Inhalt, sondern auch der Sender soll geheim bleiben.
   Gewünscht bei z.B.:
   - Bargeldzahlungen
   - Karitative Organisationen (Anonyme Alkoholiker...)
   - Chiffreanzeigen

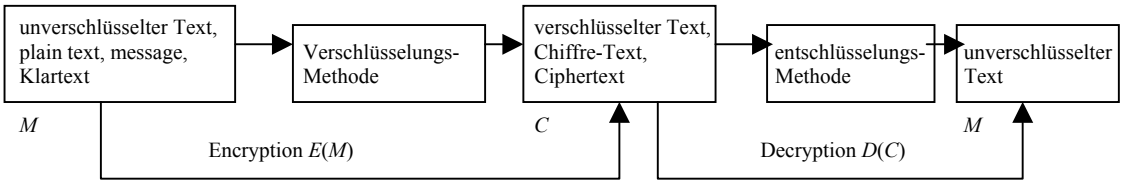
   Bei elektronischer Kommunikation ist das alles Problematisch (z.B. bei Elektronic Cash)

4. Protokolle
   Kommunikation erfordert Regeln, deren Gesamtheit man *Protokoll* nennt.
   Notwendig bei z.B. - Geldautomat.
   - elektronisches Geld
   - elektronische Verträge
   - E-Mail
   - Mobilfunk

Anwendungsgebiete der Kryptographie

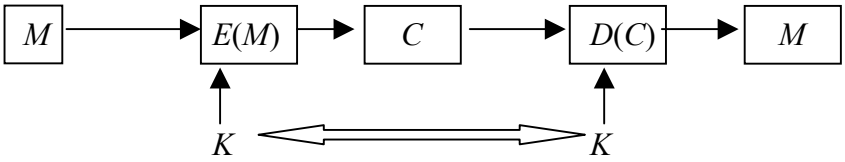
- Banken
- Datenschutz
- Militär, Geheimdienst
- Wirtschaft

Schema der Verschlüsselung:



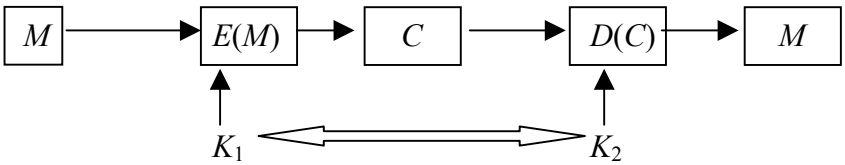
E ist eine Verschlüsselungsfunktion, die *M* in *C* umwandelt. ($E(M) = C$)
C wird dann von der Entschlüsselungsfunktion *D* wieder in *M* umgewandelt ($D(C) = M$)
Der ursprüngliche Klartext soll wieder hergestellt werden, d.h. $D(E(M)) = M$
Das ent- und verschlüsseln wird mit einem Schlüssel durchgeführt (=Key *K*)

a) *gleicher Schlüssel, symetrischer Fall*



$$E_K(M) = C\ ;\ D_K(C) = M \quad => D_K(E_K(M)) = M$$

b) *unterschiedliche Schlüssel, asymetrischer Fall*



$$E_{K1}(M) = C\ ;\ D_{K2}(C) = M \quad => D_{K2}(E_{K1}(M)) = M$$

c) Kryptosysteme bestehen aus einem Algorithmus einschließlich aller möglichen Klartexte, Chiffretext und Schlüssel.

zu a) Der Chiffreschlüssel läßt sich aus dem Dechiffrierschlüssel berechnen. Meist sind sie identisch. Sender und Empfänger müssen einen Schlüssel vereinbaren. Die Sicherheit in der Methode liegt im Schlüssel

zu b) Public-Key-Algorithmen
Der Dechiffrierschlüssel kann nicht aus dem Ciffrierschlüssel berechnet werden => Chiffrierschlüssel kann veröffentlicht werden.

Kryptoanalyse

Wiederherstellung der Klartexte ohne Kenntnis des Schlüssels (Sogn. Angriffe, Attacks)

Vier Arten werden unterschieden:

1. **Ciphertext-Only Angriff**
Der Angreifer kennt nur einige chiffriere Texte. Er möchte daraus die Klartexte oder den Schlüssel berechnen.
gegeben: $C_1 = E_k(P_1),...,C_n = E_k(P_n)$
gesucht: entweder $P_1...P_n$ oder E_k bzw. K

2. **Known-Plaintext-Angriff**
Bekannt sind einige Chiffretexte mit zugehörigem Klartext. Ziel ist entweder das Finden des Schlüssels oder aber das entwickeln eines Verfahrens, mit dem weitere Texte entschlüsselt werden können.
gegeben: $P_1, C_1 = E_k(P_1);...;P_n, C_n = E_k(P_n)$
gesucht: entweder K oder ein E_{neu}, um P_{n+1} aus $C_{n+1} = E_{neu}(P_{n+1})$ abzuleiten.

3. **Chosen-Plaintext-Angriff**
Ähnlich dem Verfahren in 2, jdeoch ist P_n ausgewählt, also z.B. eine Nachricht aus bestimmten Folgen von Buchstaben, damit das Decodieren etwas leicheter fällt.
gegeben: wie in 2., aber $P_1...P_n$ können selbst gewählt werden.

4. **Adaptive-Chosen-Plaintext-Angriff**
Der Angreifer wählt wie in 3 seine Texte aus, jedoch werden immer wieder durch die Art des Textes Erkenntnisse aus vorangegangenen Tests mit abgeprüft oder benutzt.

Klassische Methoden der Kryptographie

1. Ersetzungsmethoden
2. Verschiebungsmehtoden

1. Ersetzungsmethoden

a) Verschiebung des Alphabetes um *k* Buchstaben (geht auf Cäsar zurück)
Beispiel:

```
    abcdefg...
abcdefghij...
```

k ist in diesem Beispiel 3. Dann wird „informatik“ zu „lqiruptwln“
Der Schlüssel ist das *k*. Die maximale Anzahl verschiedener Schlüssel ist 26.

b) monoalphabetische Ersetzung = jeder Buchstabe wird auf einen anderen abgebildet.
Beispiel:

```
abcdefg...
xdeaikj...
```

Der Schlüssel ist in diesem Fall das gesamte Alphabet. Die maximale Anzahl verschiedener Schlüssel ist dann $26! \approx 4 \cdot 10^{26}$

Beide Methoden sind nicht sehr sicher, aufgrund der unterschiedlichen Auftrittswahrscheinlichkeiten der einzelnen Buchstaben in natürlichen Sprachen.

c) polialphabetische Ersetzung (Vegemère-Methode)

```
Zeile a)    abcdefg...  ┐
Zeile b)    bcdefgh...  │
Zeile c)    cdefghi...  } 26 Zeilen
:           :           │
Zeile z)    zabcdef...  ┘
              26 Spalten
```

Der Schlüssel besteht aus einem Wort: z.B. „CODE“

```
CODECODECODECODECODE
HEUTESCHEINTDIESONNE
```

Das ‚C‘ von Code gibt an, daß in der C-Spalte gesucht werden soll. Dazu das ‚H‘, das es mit ‚F‘ zu kodieren ist, da das der korrespondierende Eintrag in der A-Spalte ist.

Die Folge ist, daß gleiche Buchstaben immer wieder unterschiedlich kodiert werden und daher häufiges Auftreten von Kodewörtern keine Aussagekraft mehr hat.
Als Schlüssel dient das Wort.

Um diesen Kode zu knacken, muß man eine Wortgröße *x* annehmen und dann den codierten Text nach *x* Buchstaben jeweils umbrechen. Die erste Spalte ergäbe dann alle Codeworte, die mit der (z.B.) C-Spalte codiert wurden. Damit ergibt sich wieder eine monoalphabetische Codierung, welche sich lösen läßt.

2. Verschiebungsmethoden

Ersetzungsmethoden: vertauschen Buchstaben, aber erhalten die Reihenfolge
Verschiebungsmethoden: verändern die Reihenfolge aber das Alphabet bleibt unverändert.

Spaltenweise Verschiebung.

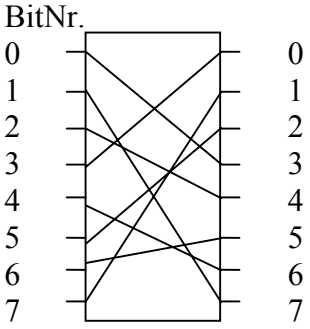
- Wähle als Schlüssel ein Wort, z.B. CODE, in dem kein Buchstabe doppelt vorkommt.
- Der Text wird Spaltenweise unter dem Schlüsselwort angeordnet
- Die Spalten werden aufsteigend gemäß dem Alphabet nummeriert
- Der Text wird nun Spaltenweise gelesen, wobei mit der niedrigsten Nummer begonnen wird.

```
C    O    D    E
1    4    2    3
H    E    U    T
E    S    C    H    => HEEDO UCNEN THTSE ESIIN
E    I    N    T
D    I    E    S
O    N    N    E
```

Beispiel einer modernen Methode:

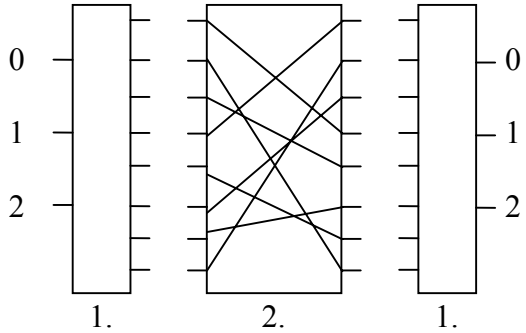
A. Data encryption Standard (DES), auf Hardwarerealisierung in sog. S-Boxen und P-Boxen. => *sehr schnell*

Prinzip der P-Box: *(P=Permutation, Realisierung einer Verschiebungsmehtode auf BIT-Ebene)*



Beispiel der Permutation: 0123 -> 3740
Durch interne Verdrahtung kann jede Permutation durchgeführt werden.

Prinzip der S-Box: *(S=Substitution, realisiert Ersetzen)*
Die S-Box besteht aus 3 Bestandteilen, wobei der mittlere eine P-Box ist.



Abhängig von den drei Eingängen an der S-Box wird eine der 8 Ausgangsleitungen aktiviert.

Eingänge 012	Ausgänge
000	0
001	1
010	2
011	3
:	:
111	7

Die Arbeit des 3. Teils ist genau invers zu der des 1. Teils.
Beispiel:
000 am Eingang angelegt-> Nr.0 aktiviert -> von 0 auf 3 in der P-Box -> 011 Ausgang

In der Realität findet man eine Zusammenschaltung von 19 Stufen (19mal erst eine P-Box mit nachgeschalteter S-Box) mit 64 Eingangsbits.

B. Der DES-Algorithmus

Der zu verschlüsselnde Text wird in 64 Bit-Blöcke zerteilt. Diese 64 Bit werden in 19 Stufen vertauscht. Die Stufen werden mit einem 56 BIT langen Schlüssel parametrisiert. Der selbe Schlüssel wird zum Ver- und Entschlüsseln verwendet (Symetrischer Algorithmus). DES ist im wesentlichen eine monoalphabetische Ersetzung, wenn auch etwas komplizierter. Jeder 64-Bit-Block wird auf einen anderen 64-Bit-Block abgebildet => 64! Möglichkeiten = ca.10^{89}.
Ein Rechner, der pro Sekunde 1 Mio Möglichkeiten durchprobieren könnte, würde immer noch $4 \cdot 10^{75}$ Jahre benötigen. (Das alter des Universums, wenn es geschlossen ist, schätzt man auf 10^{11} Jahre). Eine Schlüssellänge von 56 BIT reduziert die Zahl der Möglichkeiten stark, daher der Vorschlag von IBM auf 128 BIT zu wechseln (wollte die USA aber nicht).

Public Key Cryptosystem

Sei *P* eine Nachricht, *E* (=encrypted) der Verschlüsselungsalgorithmus und *D* (=decrypt) der Entschlüsselungsalgorithmus.

1976 von Diffie und Hellman entwickelt:
E und *D* so wählen, daß es nicht möglich ist, *D* zu ermitteln, selbst wenn *E* bekannt ist, d.h.:

1. $D(E(P)) = P$
2. Es ist sehr schwehr (praktisch unmöglich), *D* aus *E* zu ermitteln
3. Auch wenn beliebig viele Paare unverschlüsselt/verschlüsselter Nachrichten vorliegen, können *D* und *E* nicht ermittelt werden.

1978 RSA-Methode
(Realisierung des oben postulierten Algorithmus)
von Rivist, Shamier, Aidleman entwickelt
Prinzip (ohne Beweis)

1.) Suche zwei Primzahlen *p*, *q* mit der Eigenschaft, *p* und *q* jeweils $>10^{100}$
2.) Sei $n := p \cdot q;\ \ z := (p-1)(q-1)$
3.) Suche eine Zahl *d*, die relativ Prim zu *z* ist: d.h. *d* und *z* haben keinen echten gemeinsamen Teile, z.B. 4, 21
4.) Suche eine Zahl *e* mit $(e \cdot d) \bmod z \equiv 1$

Verschlüsselungsverfahren:

- Stelle jedes Zeichen nummerisch dar
- Teile den Text in Bitblöcke der Länge *k*, wobei $2^k < n$
  (sonst ist die Entschlüsselung nicht mehr möglich)
- Sei *p* der Wert des Bitblock, denn ist $c = p^e \bmod n$
  -> verschlüsselte Nachricht

Entschlüsselungsverfahren

- Berechne $p = c^d \bmod n$
- Man kann zeigen, daß beide Berechnungen zueinander invers sind.

Für die *Verschlüsselung* benötigen wir (e, n) e = *öffentlicher* Schlüssel.
Für die *Entschlüsselung* benötigen wir (d, n) d = *geheimer* Schlüssel.
d kann praktisch nicht aus e berechnet werden.

Bewertung:

- Es ist relativ leicht, große Primzahlen zu finden
- Es ist dagegen sehr schwehr, große Zahlen in ihre Primfaktoren zu zerlegen. (Schätzungsweise 4 Milliarden Jahre für eine 200stellige Zahl)

Beispiel:

Wähle $p = 3$ und $q = 11$ (wir bleiben mal klein!)
$n = p \cdot q = 33$
$z = (p-1)(q-1) = 20$
Wähle $d = 7$ (3,11,19... hätten es auch werden können)

Daraus ermitteln wir e : $(7 \cdot e) \bmod 20 = 1 \Rightarrow e = 3$
Verschlüsselung jeweils mit $C = P^3 \bmod 33$
Entschlüsselung jeweilt mit $p = C^7 \bmod 33$

Text	Numerisch	P^3	$C = P^3 \bmod 33$	C^7	$C^7 \bmod 33$
I	09	729	3	2.187	09
D	04	64	31	27.512.614.111	04
E	05	125	26	8.031.810.176	05
F	06	216	18	612.220.032	06
I	09	729	3	2.187	09
X	24	13.824	30	21.870.000.000	24

P muß (in diesem Fall) < 33 sein, daher kann auch nur immer ein Zeichen verschlüsselt werden. (=>monoalphabetische Esetzung)
Wären p und $q \approx 10^{100} \Rightarrow n \approx 10^{200}$, jeder der Blöcke könnte bis zu 664 Bit lang sein, da $2^{664} \approx 10^{100} \triangleq 83$ Zeichen

Es gibt viele Hardwareimplementierungen (RDA-Chips), jedoch immer noch 1000mal langsamer, als DES.
Geschwindigkeit von Softwareimplementierungen bei gängigen Blocklängen (=512, 768, 1024 BIT) liegen im bereich von ca. 0,03 – 0,9 Sek. pro Block.

Bei aber z.B. 0,9 Sek. pro 1024-BIT-Block => ca. 10 Sek. für 1 KByte => 50 KByte (2-3 Seiten Word-Text) = 500 Sek. = ca. 10 min.

Die Perfekte Verschlüsselungsmethode

One-Time-Pads (erfunden 1917)

Ein One-Time-Pad ist eine lange Folge von zufällig gewählten Buchstaben (→Schlüssel).

Der Sender:

Der Sender chiffriert mit jedem Buchstaben auf dem Pad genau ein Klartextzeichen. Dabei funktioniert die Verschlüsselung als Addition der beiden Zeichen und anschließendem Modulo 26.

Der Empfänger:

Der Empfänger besitzt das identische One-Time-Pad und dechiffriert die Nachricht Zeichenweise durch Subtraktion der beiden Zeichen. Negative Ergebnisse müssen von 26 abgezogen werden, damit der richtige Buchstabe herauskommt.

Beispiel:

Nachricht: `ONETIMEPAD`
Schlüssel: `TBFRGFARFM`
Chiffriert: `IPKLPSFHGQ`

Chiffrierung:

O + T => 15 + 20 = 35 => 35 mod 26 = 9 = I
N + B => P
E + F => 5 + 6 = 11 => 11 mod 26 = 11 = K

Dechiffrierung:

I – T = 9 – 20 = -11 => -11 + 26 = 15 = O
K – F = 5 = E

Dieser Code garantiert absolute Sicherheit der verschlüsselten Nachricht, da ein bestimmter Chiffriertext mit der selben Wahrscheinlichkeit zu jedem möglichen (auch Sinn ergebenden) Klartext der gleichen Länge gehören kann.

Beispiel dazu:

Schlüssel: `POYYAEZX??`
→ `SALMONEGGS`
oder auch: `BXFGBMTMXM`
→ `GREENFLUID`

Probleme:

- „Zufällige“ Generierung der Schlüsselbuchstaben, die meistens gar nicht so zufällig ist. (Auf Computern werden Zufallszahlen durch einen Algorithmus erzeugt und sind damit unter bestimmten Umständen sogar vorhersehbar.)
- jeder Schlüssel darf nur einmal verwendet werden.

- Die Länger einer Nachricht entspricht der Länge des Pads => die Verschlüsselung eignet sich nur für kleinere Nachrichten.

  Beispiel:

  Ein Übertragungskanal soll 1.544 MBits/Sec (ca. 200 KByte/Sec) übertragen können.
  Eine CD mit 650 MByte enthällt den Schlüssel => bei dieser Konstellation kann ungefähr eine Stunde lang verschlüsselt übertragen werden.

Heutige Verwendung findet der Code bei Kommunikationskanälen mit niedriger Bandbreite und höchsten Sicherheitsansprüchen (rotes Telefon des US-Präsidenten).

Heutzutage wird der One-Time-Pad-Code nur noch auf Bitebene verwendet. Die Nachricht wird digital als eine Folge von Bits interpretiert.

Beispiel:
101010 => mit Schlüssel 11001011 => 01100001 => mit Schlüssel => 101010

Protokolle

s. Kopien!

Autentikationsverfahren

Nachweis der Identfikation einer Person

A) Passwortverfahren (Festcodeverfahren)
(z.B. Anmeldung an einem Netzwerk)

Die anmeldende Person beweist, daß sie ein Geheimnis kennt. Das einfachste Verfahren ist dabei die Nennung eines Passwort.

Als Beispiel aus dem Alltag:
Grete und Hans telefonieren mit einander. Grete meldet sich mit „Hier ist Grete“ und verwendet dann den nur ihnen bekannten Code „Dein Schnuckiputzi“. Damit weiß Hans, daß es wirklich Grete sein muß, da nur sie diesen Kosenamen kennt.

Zur Identifikation einer Person eignen sich also Geheimnisse *individueller Natur*, z.B. Paßwörter, PINs, o.ä.

In Computernetzwerken sind alle Geheimnisse einer zentralen Stelle bekannt.
Die Autentikation läuft dabei folgendermaßen ab:
Der Benutzer schickt seine Identität (Name + UserID) und sein Geheimnis (p') an den zentralen Rechner.
Die Zentrale überprüft nun, ob Geheimnis (p') und Identität zusammenpassen, indem es das übermittelte Geheimnis (p') und ein vorher gespeichertes Geheimnis (p) vergleicht.

Schwächen:

(1) Zentrale Speicherung der Paßwörter
Lösung: Paßwörter (p) werden mit einer Einwegfunktion f verschlüsselt und $f(p)$ wird gespeichert. Die Zentrale vergleicht nun $f(p)$ und $f(p')$.

(2) offene Übetragung des Paßwortes
einmal abgehört kann es immer wieder verwendet werden.
Lösung: Man sollte Paßwörter so häufig wie möglich wechseln.
Beispiel: Homebanking : 1. PIN (fest) 2. TAN (nur ein mal gültig)

Einschub:

Was ist eine Einwegfunktion:

Eine Einwegfunktion ist eine einfach auszuführende, aber schwer (praktisch unmöglich) zu invertiernde Funktion.

$f : x \to y$ wobei $f(x)$ für jedes $x \in X$ leicht zu berechnen ist.

aber es ist für alle $y \in Y$ extrem schwer, ein $x \in X$ zu finden, mit $f(x) = y$

Beispiele:

(1) Telefonbuch: $f : Name \to Tel.-Nr.$

(2) Teller fallen lassen: $f : Teller \to Scherben$

(3) $f(x) = x^2$ ist einfach, aber $f(x) = \sqrt{x}$ ist kaum noch im Kopf zu rechnen.

(4) Public Key Cryptosystem (RSA)
Berechnung des Chiffretextes ist einfach, jedoch die Rekonstuktion des Plaintextes ohne den richtigen Schlüssel ist extrem schwierig.
(Daher auch Einwegfunktion mit Hintertür, dem Schlüssel)

(5) Die PIN ist der Funktionswert einer Einwegfunktion

(6) Der diskrete Logarithmus:
geg.: ist eine Primzahl p, $p, g \in N; g \le p-1$
Die diskrete Exponentialfunktion de zur Basis g:

$$\mathrm{de}_g(k) = g^k \bmod p$$

Die Umkehrfunktion ist der diskrete Logarithmus dl_g

$$\mathrm{dl}_g\left(g^k\right) = k \bmod p$$

Problem des diskreten Logarithmus:
geg.: p, g, y
ges.: k, so daß $y = g^k \bmod p$
z.B.:

$$y = 7,\ p = 17,\ g = 3$$

ges. k, so daß $7 = 3^k \bmod 17$

$k = 11$ ist die Lösung

oder:

$$4^k \bmod 11 = 3 \Rightarrow k = 5$$

Nachteilig ist, daß man alle Zahlen $0 \le k \le p-1$ ausprobieren muß

Beispiel:

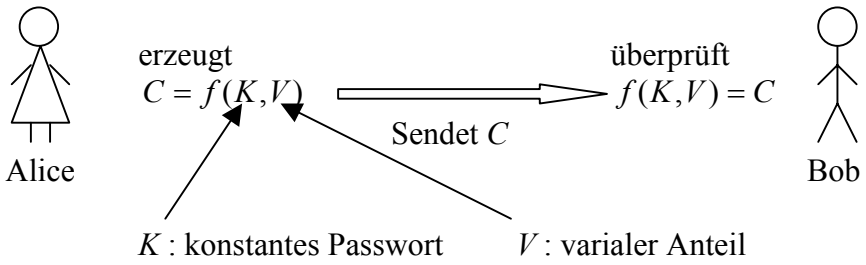
Die auftretenden Zahlen haben eine Länge von ca. 1.000 BIT ($\hat{=} 10^{301}$ Dezimalstellen)

Die Berechnung von g^k, dem diskreten Exponenten, erfodert ca. 2.000 Multiplikationen, d.h. der diskrete Exponent ist relativ schnell berechnet. (bei heutigen Maschinen ca. 1 Sek.)

Die Berechnung des diskreten Logarithmus benötigt dagegen ca. $2^{500} \hat{=} 10^{150}$ Operationen. Ein Rechner, der 10^6 Operationen pro Sek. schafft, benötigt etwa $\frac{10^{150}}{10^6} = 10^{144}$ Sek. für die Berechnung ($\cong 3 \cdot 10^{136}$ Jahre, zum Vergleich: die Lebensdauer des Universums beträgt ca. 10^{11} Jahre.)

B) Wechselcodeverfahren

Für jede Autentikation wird aus einem konstantem Geheimnis und einem variablen Wert ein Autentikationscode berechnet. Dabei bedient man sich für *f* einer Einwegfunktion.



Es gibt nun mehrere Möglichkeiten, diese Technik zu verwenden:

Beispiel 1:

Alice und Bob besitzen einen gemeinsamen Schlüssel *K* und einen gemeinsamen Anfangswert $Z\ (\in N)$ für *V*.

Alice bildet nun $C_i = f(K, Z+i)$ und sendet C_i. Bob prüft dann $f(K, Z+i) = C_i$

Da jedoch *K* und *Z* ausgetauscht werden müssen, ist hierin eine mögliche Schwachstelle zu sehen.

Beispiel 2:

Bob braucht das Geheimnis von Alice nicht zu kennen. Benötigt wird eine Einwegfunktion f, die ruhig öffentlich bekannt sein kann. Es wird festgelegt, wie oft das Verfahren benutzt wird. (z.B. 1.000)

Alice berechnet nun ausgehend von einem Startwert Z_0 :

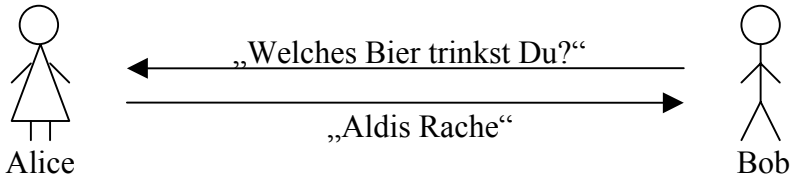
$Z_{i+1} = f(Z_i)\ ;\ \ i = 0,1,2,...,n$ also $Z_1 = f(Z_0),...,Z_{1000} = f(Z_{999})$

und legt sich mit den Werten eine Tabelle an. Dann übermittelt sie Z_{1000} an Bob.

Bei der ersten Autentikation sendet Alice dann Z_{999} an Bob. Dieser überprüft, ob $f(Z_{999}) = Z_{1000}$, da er ja das schließlich alles kennt. Das nächste mal kommt bei Bob Z_{998} an und er testet nun $f(Z_{998}) = Z_{999}$. Damit kann Bob immer die korrekte Autentikation testen.
Ein Angreifer müßte nun immer im Besitz des nächsten *Z* sein, wobei sich mehrere Probleme ergeben. Da er nicht weiß, wieviele *Z* es gibt, kann er Alice's Tabelle nicht erzeugen und da er zudem auch schwer den Index kennen kann, kommt er auch hier nicht zum Zuge, da er garantiert das falsche *Z* senden würde.
Dabei hilft es ihm auch nicht, wenn er f oder Z_{1000} kennen würde.

Beispiel 3: Challenge and Response-Verfahren:

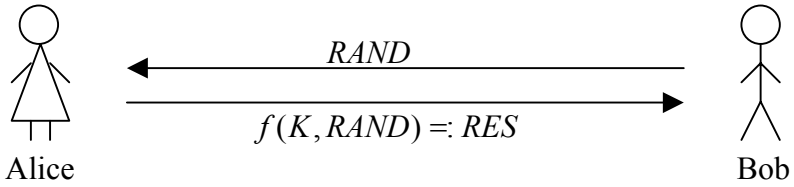
Das Challenge and Response-Verfahren kommt ohne dem Vorproduzieren von Autentikationswerten aus. Vielmehr lebt es von einer „frischen" Produktion solcher Werte. Dabei wird dem Autentikations-Kandidaten eine unvorhersehbare Frage gestellt:



Technische Realisierung:
Bob und Alice kennen eine Einwegfunktion f, die von einem Schlüssel *K* abhängt. Bei einem Autentikationsversuch gibt nun Bob an Alice einen Zufallswert *RAND*, welcher die „Challenge" darstellt, also die unerwartete Frage.

Alice berechnet nun $f(K, RAND) =: RES$ („Response")

und Bob prüft $RES = f(K, RAND)$



Dieses Verfahren sichert die Leitungsübermittlung der Autentikation, da durch den Zufallswert immer ein anderer Code übermittelt wird. Somit kann das Passwort nicht von der Leitung abgehört werden, sondern muß mit „über-die-Schulter-schaun"" erhascht werden.