Daten

Seminararbeit aus dem Proseminar
"Grundlagen wissenschaftlichen Arbeitens"
im SS 2003

Andreas Obermair
Karin Frauwallner

14. April 2003

Kurzfassung

Aufgabe dieser Seminararbeit war es, einen Überblick über den Umgang mit Daten in der Informatik zu geben.

Im vorliegenden Dokument wird dieses Thema in die folgenden Teile gegliedert:

1. *Zeichen- und Zahlendarstellung*
2. *Informations- und Codierungstheorie*
3. *Kryptologie und Datensicherheit*

Die Codierung von Daten ist die Grundvoraussetzung, um Information speichern, verarbeiten oder übermitteln zu können. Man kann hier zwischen Codierung und Verschlüsselung unterscheiden.

Während durch die Codierung der Inhalt einer Nachricht in einer speicherbaren, verarbeitbaren oder übermittelbaren Form dargestellt wird, ist es die Aufgabe der Verschlüsselung, diesen Inhalt für Unauthorisierte unverständlich zu machen.

Im ersten Teil dieser Arbeit wird also die Codierung von Zeichen und Zahlen durch Bitmuster behandelt.

Im zweiten Teil geht es dann um die Codierungstechniken zur Datenübertragung, wobei einerseits auf die codierungstheoretischen Grundlagen und andererseits auf die verschiedenen Codierungsverfahren eingegangen wird.

Teil drei befaßt sich schließlich mit Techniken zur Codierung, die den wichtigen Aspekt der Datensicherung zum Ziel haben.

Nach einer kurzen Einleitung in die jeweiligen Gebiete werden sowohl die theoretische Basis als auch verschiedene Anwendungsbereiche detailliert behandelt.

Inhaltsverzeichnis

1 Zeichen- und Zahlendarstellungen

1.1 Zeichendarstellung

In den Anfängen der elektronischen Datenverarbeitung war man aufgrund der geringen und verhältnismäßig teuren Speichermöglichkeiten vorerst auf die Zifferncodierung beschränkt.

Später wurde der verwendbare Zeichensatz um die 26 Großbuchstaben des lateinischen Alphabets sowie um einige Sonder- und Satzzeichen erweitert (*6-Bit-Code*). Die weitere Entwicklung, der *7-Bit-Code*, ermöglichte zusätzlich die Codierung von Kleinbuchstaben und einigen wenigen sprachenabhängigen Schriftzeichen (in unterschiedlichen, länderspezifischen Versionen).

Erst durch den *8-Bit-Code* konnte eine Vielzahl nationaler bzw. sprachenabhängiger Zeichen in weltweit einheitlicher Form eingebunden werden.

Mit dem **Unicode** (*16-Bit-Code*) und dem **UCS** (*32-Bit-Code*) erschloss sich schließlich die Möglichkeit zur Codierung aller wesentlichen (Unicode) oder sogar aller (UCS) weltweit vorkommenden Schriftzeichen.

Bei der Beschreibung von Codes unterscheidet man zwischen den *Zeichensätzen*, bestehend aus Schrift- und Steuerzeichen, und ihrer *Codierung*:

- *Schriftzeichensatz*: Menge aller Schriftzeichen, die codiert werden (Ziffern, Ziffern + Buchstaben des lat. Alphabets + Satzzeichen, usw.)

  Sein Umfang hängt vor allem vom verfügbaren Speicherplatz für die codierte Darstellung ab, weiters auch von den (länderspezifischen) Anforderungen. In Europa und Amerika ist ein 8-Bit-Code (191 Schriftzeichen) ausreichend, während z.B. in China über 10.000 Zeichen und ein demzufolge längerer Code benötigt werden.

- *Steuerzeichensatz*: Alle für die Anwendung (Eingabe, Ausgabe, Übermittlung) nötigen Funktionen

- *Codierung*: Abbildung der Schrift- und Steuerzeichen auf Codewörter (Folgen der Binärzeichen 0 und 1)

1.2 Zifferncodes

Für Zifferncodes besteht der Schriftzeichensatz aus den *zehn Ziffern 0 bis 9*. Um die Zahlen bis dezimal 9 zu codieren, sind 4 Binärstellen erforderlich. Das Kriterium "Gewichtung" bezieht sich im folgenden auf die Existenz eines Stellengewichtes dieser Binärstellen.

Bei den nun kurz beschriebenen handelt es sich um historische Codes:

- *Dualcode*: dient zur Darstellung der Dualzahlen.

  Jede Stelle $n \in \{0, ..., m-1\}$ (m =Anzahl aller Stellen) ist mit 2^n gewichtet.

- *BCDIC (binary coded decimal interchange code)*: wie Dualcode, allerdings fällt die Darstellung der Ziffer 0 aus der Gewichtung heraus.

  BCDIC wurde früher für Magnetbänder verwendet.

- *Excess-3-Code* oder Stibitz-Code: Der ungewichtete Code entsteht durch die Addition von dezimal 3 bzw. binär 0011 zum BCDIC-Code.

  Er ist *selbstkomplementierend*, was leicht ersichtlich wird, wenn man den Code für eine Ziffer mit dem Code für 9 minus dieser Ziffer vergleicht.

- *Aiken-Code*: Hier hat jede der Binärstellen eine vorgegebene Gewichtung, die höchstwertige Stelle hat jedoch den Stellenwert 2 statt 8.

  Um Eindeutigkeit zu gewährleisten, ist der Wert der höchstwertigen Stelle immer 0 für die Zahlen von 0-4 und 1 für 5-9, was den Vorteil hat, dass der Code *selbstkomplementierend* ist.

- *Gray-Code*: Zur erleichterten Fehlererkennung wird in einer Folge numerischer Werte immer nur ein Bit verändert (*stetiger Code*). Die einzelnen Stellen sind nicht gewichtet.

- *2-aus-5-Codes*: Er benutzt fünf Ziffern für die Darstellung der Ziffern 0-9. Jeweils zwei Bits sind 1, drei Bits sind 0. Die Darstellung der Ziffer 0 fällt aus der Gewichtung heraus.

- *Biquinär-Code*: Jeweils ein Bit des binären (Wertigkeit $5, 0$) und des quinären (Wertigkeit $4, 3, 2, 1, 0$) Teils enthält eine 1.

1.3 Codierung alphanumerischer Daten

1.3.1 6-Bit-Codes

Mit einem 6-Bit-Code sind $2^6 = 64$ Zeichen darstellbar.

Beispiel. BCDIC (*binary coded decimal interchange code*) von IBM.
Es gibt 16 Plätze für *Steuerzeichen*. Der *Schriftzeichensatz* enthält die 26 Großbuchstaben des lat. Alphabets, die Ziffern 0 bis 9 und zwölf Satz- und Sonderzeichen. Heute werden 6-Bit-Codes allerdings nicht mehr verwendet.

1.3.2 7-Bit-Codes

Mit einem 7-Bit-Code sind $2^7 = 128$ Zeichen darstellbar.

Beispiel. 7-Bit-Code nach ISO/IEC 646.
Es gibt 32 Plätze für *Steuerzeichen*. Dieser Code wurde hauptsächlich für Datenkommunikation, nicht -verarbeitung entwickelt. Der *Schriftzeichensatz* besteht aus 84 fix belegten und 12 länderspezifisch normierte Plätze.
In der *Deutschen Referenzversion (DRV)* wurden diese zwölf Plätze mit dem Paragraphzeichen, dem Gravis, den Groß- und Kleinbuchstaben der Umlaute und mit dem Buchstaben ß belegt.
Die *Internationale Referenzversion (IRV)* ist mit dem bekannteren US-amerikanischen *ASCII (American Standard Code for Information Interchange)* sowie dem *Internationalen Alphabet Nr.5 (IA5')* identisch.

Die 7-Bit-Codes selbst verlieren schon an Bedeutung, jedoch ist der Zeichensatz des IRV Bestandteil bzw. Grundlage des 8-Bit-Codes nach ISO/IEC und in weiterer Folge auch von Unicode und UCS.

Steuerzeichen. Sie stehen in einem von den Schriftzeichen abgesetzten Teil der Codetabellen. 7-Bit-Codes enthalten 32, 8-Bit-Codes 64 Steuerzeichen. Man kann hier verschiedene Anwendungsgruppen unterscheiden[1]:

- *Übertragungs-Steuerzeichen*: steuern den Ablauf der Datenübertragung

[1] Bezeichnungen vom Beispiel ASCII übernommen

- *Format-Steuerzeichen*: steuern Anordnung der Daten
- *Steuerzeichen zur Code-Erweiterung*
- *Geräte-Steuerzeichen*: zum Ein- und Ausschalten von Geräten
- *Informationstrennzeichen*: für logische Gliederung der Daten
- *sonstige*

Da im Laufe der Zeit die Steuerzeichen für die Gerätesteuerung nicht mehr ausreichend waren, legte man in einer ISO/IEC-Norm *Steuerfunktionen* fest, die aus einem einleitenden Steuerzeichen, einer oder mehrerer Bitkombinationen tatsächlichem Inhalt und einer abschließenden Bitkombination bestehen. Die letzte Version dieser Norm enthielt 164 Steuerfunktionen.

1.3.3 8-Bit-Codes

Mit einem 8-Bit-Code sind $2^8 = 256$ Zeichen darstellbar.

8-Bit-Code nach ISO/IEC. Er enthält 64 Plätze für *Steuerzeichen*. Da die Codetabelle aus zwei 7-Bit-Codetabellen zusammengesetzt ist, gibt es je zwei Bereiche für Schrift- und für Steuerzeichen.

Die eine Hälfte des *Schriftzeichensatzes* ist der der IRV entnommen. Nationale Versionen existieren hier nicht.

Die andere Hälfte erfüllt die Anforderungen verschiedener Sprachgruppen, Regionen oder Schriften. Es gibt einen Teil für alle westeuropäischen Sprachen, *Latein 1* genannt, sowie einen Teil für die osteuropäischen Sprachen mit lateinischer Schrift, *Latein 2*. Weiters gibt es z.B. noch die Schriftzeichen für die slawischen Sprachen mit kyrillischer Schrift oder die Schriftzeichen für Griechisch, Arabisch, Hebräisch usw.

Der Schriftzeichensatz *Latein 1* ist weiters in EBCDIC und anders codiert auch in der Codetabelle des PC unter DOS – dem *PC-ASCII*, sowie als *ANSI-Code* unter MS-Windows enthalten.

8-Bit-Code nach EBCDIC. Dieser 8-Bit-Code wird vor allem in Großrechnern verwendet. Er wurde für die Datenverarbeitung, insbesondere für die gepackte Darstellung numerischer Daten und das erleichterte Übersetzen von Daten aus Lochkarten, entwickelt.

Er enthält 64 Plätze für *Steuerzeichen*.

Im Unterschied zum Code nach ISO/IEC, der keine national verschiedenen Codierungen desselben Zeichensatzes enthält, stellt EBCDIC für *Latein 1* mehrere Codetabellen, die sogenannten *CECP (country extended code pages)* zur Verfügung.

Im EBCDIC gibt es auch Codetabellen, die anderen Schriftzeichensätzen der ISO/IEC entsprechen, z.B. denen für Latein 2, Kyrillisch, Griechisch oder Arabisch. Allerdings gibt es für diese Fassungen keine CECP.

8-Bit-Code des PC unter DOS(PC-ASCII). Ein Teil des *Schriftzeichensatzes* (Reihen 2-7 in der Codetabelle) stimmt mit dem der IRV nach ISO/IEC oder ASCII überein. Das Betriebssystem des PC benötigt allerdings keine der nach ISO/IEC möglichen Steuerzeichen (Reihen 8,9). Deshalb wurden diese Reihen für die Codierung von zusätzlichen Schriftzeichen genutzt.

In der rechten Hälfte der Codetabelle gibt es also:

- Drei Spalten mit Buchstaben, die für nationale Sprachen benötigt werden
- Drei Spalten mit Zeichen für Linien und Tabellen
- Zwei Spalten mit Formelzeichen und Buchstaben des griechischen Alphabets

Anfangs war der Datenaustausch mit nach ISO/IEC oder EBCDIC codierten Systemen nur bedingt möglich. Nach der Übernahme von Latein 1 in die EBCDIC-Codetabellen wurde auch der Schriftzeichensatz des PC entsprechend angepaßt. Dieser enthält dazu noch drei Schriftzeichen aus dem Satz des ursprünglichen PC: das Währungszeichen für Gulden (Florin), den doppelten Unterstreichungsstrich und das Eurozeichen.

8-Bit-Code des PC unter Windows(PC-ANSI). Dieser enthält ebenfalls zusätzliche Schriftzeichen anstelle der Steuerzeichen in den Reihen 8 und 9 der Codetabelle nach ISO/IEC 8859.

1.3.4 16-Bit-Code (Unicode)

Mit einem 16-Bit-Code sind $2^{16} = 65536$ Zeichen darstellbar.

Es ist somit erstmals möglich, jedem der ausgewählten Schriftzeichen oder den Bausteinen von Schriftzeichen immer *dieselbe Bitkombination* zuzuordnen, und umgekehrt.

Der einzige in Verwendung stehende 16-Bit-Code ist der von einem privaten Konsortium entwickelte Unicode. Seine ersten 256 Plätze enthalten den Zeichensatz Latein I des 8-Bit-Codes nach ISO/IEC 8859.

Da der Platz nur zur Abdeckung von Schriftzeichen der wichtigsten lebenden Sprachen ausreicht, werden Erweiterungen des Codes um zumindest eine zusätzliche 16-Bit-Codetabelle notwendig sein.

1.3.5 32-Bit-Code (UCS)

Mit einem 32-Bit-Code sind $2^{32} = 4294967296$ Zeichen darstellbar.

Es existiert nur ein 32-Bit-Code, der UCS-4-Oktett-Code[2] (*UCS = universal character set*).

Der UCS besteht aus *128 Gruppen zu 256 Ebenen zu 256 Reihen zu 256 Zellen.* Damit können 2147483648 Zeichen codiert werden, die letzten 128 Gruppen werden nicht belegt.

Eine Ebene des UCS-Codes besteht aus 65536 Zellen. Die erste Ebene, die sogenannte *Grundebene (basic multilingual plane, BMP)* ist mit nur zwei Oktetts darstellbar. In den 256 Zellen der ersten Reihe der Grundebene ist wieder der Latein 1-Zeichensatz des 8-Bit-Codes nach ISO/IEC 8859 enthalten. Die Grundebene des UCS wurde zwecks Verträglichkeit der Codierung dem *Unicode angeglichen.*

Im UCS liegt die Zukunft der Codierung, denn mit ihm ist es möglich, jedem Schriftzeichen, ob aus lebenden oder toten Sprachen, eine einzige, immer gleiche Bitkombination umkehrbar eindeutig zuzuordnen.

Abschließend noch eine Zusammenfassung der bedeutendsten Codes:

1.3.6 Bezeichnungsweisen

Die Nummerierung der Bitpositionen erfolgt in allen Codes von links nach rechts:

[2] Der Begriff *Oktett* steht für eine Guppe von 8 Bits

Bitanzahl	Codename	Internationale Norm	Deutsche Norm	Schriftzeichensatz
7	ASCII	ISO/IEC 646	DIN 66003	IRV
8	Latein 1	ISO/IEC 8859 Teil 1	DIN 66303	IRV + Erweiterung
8	Latein 2	ISO/IEC 8859 Teil 2	—	IRV + Erweiterung
8	EBCDIC	—	—	IRV + Erweiterung
8	PC-ASCII	—	—	IRV + Erweiterung
16	Unicode	—	—	IRV + Erweiterung
32	UCS	ISO/IEC 10646	—	IRV + Erweiterung

Tabelle 1.3.5.1: Namen, Normierung und Schriftzeichensätze wichtiger Codes

Bit i = Bit der Wertigkeit 2^{i-1}

Die *vier Oktetts des UCS* haben bestimmte Namen:

32 25	24 17	16 9	8 1
8	8	8	8
Gruppe	Ebene	Reihe	Zelle
G-octet	P-octet	R-octet	C-octet
group	plane	row	cell

Abbildung 1.3.6.1: Bitpositionen und Oktett-Bezeichnungen des UCS

Beim Bezug auf ein Zeichen einer vorbestimmten Ebene kann auf die Angabe des G- und P-Oktetts verzichtet werden.

1.4 Ganze Zahlen

Zur Repräsentation von Informationen in einer binären Rechenanlage wird jeder in Frage kommende Wert durch ein bestimmtes **Bitmuster** dargestellt bzw. verschlüsselt (*codiert*, engl. *encoded*).

Dazu kann eine *Code-Tabelle* verwendet werden, die für jede direkt verständliche Darstellung eines Wertes in einer natürlichen Sprache oder einer anderen direkt verständlichen Repräsentation ein bestimmtes Bitmuster enthält.

Bei arithmetischen und logischen Operationen unterscheidet man zwei Datentypen für ganze Zahlen: *vorzeichenlos* (unsigned) und *vorzeichenbehaftet* (signed). Sie unterscheiden sich in der *Zahlcodierung* sowie im *darstellbaren Zahlbereich*.

1.4.1 Vorzeichenlose Dualzahlen

Positive ganze Zahlen sind ein sehr häufiger Datentyp in Anwendungsprogrammen (z. B. als Feldindex, Laufvariable in Schleifen), aber auch maschinenintern als Speicheradresse usw.

Die Darstellung erfolgt als *Dualzahl* (Zahl zur Zahlenbasis 2). Dabei entspricht jede Ziffernposition einer Zweierpotenz (wie bei Dezimalzahlen jede Ziffernposition einer Zehnerpotenz entspricht).

$$n = \sum_{i=0}^{N} b_i 2^i \qquad \text{mit } b_i = \{0, 1\}$$

Beispiel. Dezimal$\Longrightarrow$Dual: fortgesetzte Division durch 2, Notieren der Reste

30_{10} als Dualzahl

$$\begin{array}{rll} 30:2=15 & \text{Rest } 0 & \\ 17:2=7 & \text{Rest } 1 & \\ 7:2=3 & \text{Rest } 1 & \\ 3:2=1 & \text{Rest } 1 & \\ 1:2=0 & \text{Rest } 1 & \Rightarrow 30_{10}=11110_2 \end{array}$$

1.4.2 Vorzeichenbehaftete Dualzahlen

Während positive Zahlen direkt durch das ihrer Binärdarstellung entsprechende Bitmuster repräsentiert werden, gibt es für negative Zahlen mehrere Möglichkeiten der Codierung. Die geläufigsten sind nachfolgend beschrieben.

1. *n-1 Bits* für den Betrag als ganze Zahl und *ein Bit* für das Vorzeichen.

   Nachteile:

   - Die Null gibt es zweimal $(+0, -0)$
   - Die Arithmetik wird durch Fallunterscheidungen komplizierter.

2. *Einer-Komplement*: Beim Einer-Komplement erhält man die Darstellung einer negativen Zahl aus der Darstellung der entsprechenden positiven Zahl, indem man diese *bitweise invertiert*.

   Der darstellbare Zahlenbereich ist bei Verwendung eines 8 Bit breiten Datenformats $[-127; +127]$.

Dezimal	Einer-Komplement							
38_{10}	0	0	1	0	0	1	1	0
-38_{10}	1	1	0	1	1	0	0	1

Abbildung 1.4.2.1: Einer-Komplement-Darstellung im Prozessor

   Nachteil:

   - Die Null gibt es zweimal $(+0, -0)$

3. *Zweier-Komplement*: Das Zweier-Komplement erhält man durch *Addition von Eins* zum Einer-Komplement. Im obigen Beispiel erhält man also:

Dezimal	Zweier-Komplement							
-38_{10}	1	1	0	1	1	0	1	0

Abbildung 1.4.2.2: Zweier-Komplement-Darstellung im Prozessor

   Vorteil:

   - Vorzeichen am ersten Bit erkennbar.

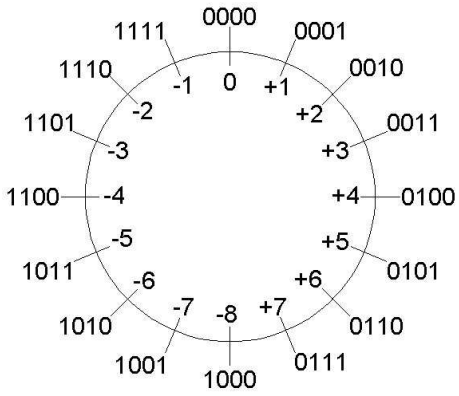

Abbildung 1.4.2.3: 4-Bit-Darstellung von Dezimalzahlen im Zweier-Komplement

Beispiel. 3-Bit Zahlen: $2_{10} = 010_2$

$$-2_{10} = 101_2 + 1 = 110_2$$

Vorteile:

- Vorzeichen ebenfalls am ersten Bit erkennbar.
- Die Addition negativer Zahlen geschieht wie die Addition positiver Zahlen, beim Überschreiten der Null geht der Übertrag verloren.

  Deshalb braucht man nur noch eine Hardware-Schaltung.
- Subtraktion $i - j$ durch Bildung des Zweierkomplements von j und Addition (hardwaretechnisch einfach umzusetzen):

$$i - j = i + (-j)$$

1.4.3 Computerarithmetik mit Dualzahlen

Arithmetische Operationen (Addition, Subtraktion, Multiplikation, Division) werden im Dualsystem analog zum Dezimalsystem ausgeführt.

Die *arithmetisch-logische Einheit (ALU)* erzeugt nämlich einen Ergebnisoperanden, der, sofern es keine Zahlenbereichsüberschreitung gegeben hat, für beide Zahlendarstellungen gültig ist. Zahlenbereichsüberschreitungen werden in den von der ALU zusätzlich erzeugten sogenannten *Bedingungsbits (condition codes)* angezeigt. Diese werden im Prozessorstatusregister gespeichert und können durch einen nachfolgenden Befehl, z.B. einen bedingten Sprungbefehl, ausgewertet werden. Gängige Bezeichnungen dieser Bits und deren Bedeutung sind:

- Übertragsbit (*carry bit*) C: Es zeigt an, ob ein Übertrag (*carry*) an höchster Bitposition des Ergebnisses aufgetreten ist.

  Für eine Operation mit vorzeichenlosen Dualzahlen bedeutet ein Übertrag (zu 1) gleichzeitig das Überschreiten des Zahlenbereichs, d.h., das Ergebnis ist im Datenformat nicht darstellbar, und die Bitfolge stellt einen falschen Wert dar.
- Überlaufbit (*overflow bit*) V: Es zeigt für 2-Komplement-Zahlen an, ob ein Überlauf (*overflow*) aufgetreten ist.

In diesem Fall ist das Ergebnis nicht im Datenformat darstellbar, d.h., der Zahlenbereich wurde überschritten, und die Bitfolge stellt einen falschen Wert dar.

Das Überlaufbit wird als *Exklusiv-Oder-Verknüpfung* der Überträge an höchster und zweithöchster Bitposition des Ergebnisses gebildet, vorausgesetzt, die Subtraktion wird, wie bei heutigen arithmetisch-logischen Einheiten üblich, als eigenständige Operation und nicht als Addition des 2-Komplements eines der Summanden durchgeführt.

- Negativbit (*negative bit*) N: Es zeigt für 2-Komplement-Zahlen an, ob das Ergebnis negativ ist (Kopie des höchstwerteigen Ergebnisbits).
- Nullbit (*zero bit*) Z: Es zeigt an, ob das Ergebnis Null ist.

Für die **Multiplikation** und die **Division** von Dualzahlen sind, abhängig von der Zahlendarstellung *vorzeichenlos* oder *2-Komplement-Zahl*, unterschiedliche Rechenvorgänge erforderlich, die durch je zwei Befehlscodes unterschieden werden (Multiplikation bzw. Division *unsigned* sowie *signed*).

Die Multiplikation führt bei einfacher Länge der Faktoren (Wort) üblicherweise auf ein Produkt doppelter Länge (*Doppelwort*), sodass eine Bereichsüberschreitung ausgeschlossen ist.

Bei der Division ist dementsprechend der Dividend mit doppelter Länge vorzugeben.

1.5 Gleitpunktzahlen

Das *Festkomma-Format* hat den Nachteil, dass sehr kleine (d.h. nahe bei 0 liegende) und sehr große Zahlen nicht mehr dargestellt werden können. So ist es bei einem Format mit zwei Dezimalstellen vor und zwei Dezimalstellen nach dem Komma weder möglich, die Zahl $0,001$, noch die Zahl 123 darzustellen.

Man verwendet in Rechnern daher heute praktisch ausschließlich das *Gleitkomma-Format*, das der sogenannten wissenschaftlichen Zahldarstellung entspricht. Statt z.B. $0,001$ schreibt man $1 \cdot 10^{-3}$, statt 123 schreibt man $1,23 \cdot 10^2$.

Auch das lässt sich wieder in die Welt der Binärdarstellung übertragen. Der *Exponent* wird dabei ebenfalls *zur Basis 2* geschrieben, statt zur Basis 10.

Die Zahl $\frac{1}{8}$ kann nun binär nicht nur als $0,001^2$ dargestellt werden, sondern auch als $1 \cdot 2^{-3}$ (Der besseren Lesbarkeit halber ist hier und in den folgenden Beispielen der Exponent nicht binär, sondern dezimal aufgeschrieben!). Ebenso kann die Zahl 192 binär nicht nur als 11000000_2 geschrieben werden, sondern auch als $1,1_2 \cdot 2^7$.

Eine *positive Zahl* hat in diesem Format also immer die Form $m \cdot 2^e$, eine negative entsprechend $-m \cdot 2^e$.

Die Zahldarstellung besteht damit aus drei Teilen:

1. dem **Vorzeichen** (+ oder -)
2. dem **Exponenten** (e)
3. der **Mantisse** (m)

Die *Mantisse* ist dabei eine *Festkomma-Zahl*. Damit unsere Zahldarstellung eindeutig wird, genügt es nicht, nur die *Komma-Position* festzulegen, vielmehr muß der *Zahlbereich* der Mantisse festgelegt werden. Zum Beispiel könnte sonst die Zahl 1 als $1,0_2 \cdot 2^0$, aber auch als $0,1_2 \cdot 2^1$ dargestellt werden.

Man legt üblicherweise fest, dass die Mantisse m zwischen 1 und 2 liegen muß, also $1 < m < 2$. Da sie jetzt immer mit einer (binären) 1 beginnt, braucht diese nicht einmal mehr explizit angegeben zu werden; man lässt die führende 1 in der Zahldarstellung weg und gewinnt ein Bit.

Nun muss nur noch festgelegt werden, wie viele Bits wir für die Mantisse "spendieren" und wie viele für den Exponenten. Der *IEEE-754 Standard*, der heute praktisch überall eingesetzt wird, legt für einfach genaue Zahlen (insgesamt 32 Bit) folgendes Format fest:

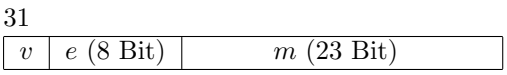

Abbildung 1.5.0.1: v=Vorzeichen (0: positiv, 1: negativ), e=Exponent, m=Mantisse

Um sowohl *positive als auch negative Exponenten* darstellen zu können, wird bei der Ermittlung des Zahlenwerts von e zunächst die Zahl $127 (= 2^7 - 1)$ abgezogen.

Der Zahlenwert ist somit definiert als: $a = (-1)^v \cdot m \cdot 2^{e-127}$

Die Zahl $\frac{1}{8}$ wird damit also wie folgt dargestellt:

0	011 1110	000 0000 0000 0000 0000 0000

Abbildung 1.5.0.2: Darstellung von $\frac{1}{8}$

Nachrechnen ergibt mit $m = 1$ (das ist die unterdrückte führende 1; die Nachkommastellen sind alle 0) und $e = 124$:

$$1 \cdot 2^{124-127} = 2^{-3} = \frac{1}{8}$$

Der *IEEE-754 Standard* legt auch noch verschiedene andere, spezielle Codierungen fest:

Mantisse	Exponent	Bedeutung
0	0	Null; das Vorzeichenbit ist egal
$\neq 0$	255	NaN (*Not a Number*); der Wert ist keine Zahl, ist undefiniert; z.B. $\frac{0}{0}$
0	255	je nach Vorzeichenbit: $+\infty$ oder $-\infty$
$\neq 0$	0	Zahlen der Form $(-1)^v \cdot m \cdot 2^{-126}$, wobei $0 < m < 1$ gilt und die erste führende 0 weggelassen wird

Tabelle 1.5.0.1: IEEE-754 Standard-Codierungen

Die letzte Codierung dient dazu, den Zahlbereich in Richtung 0 noch ein bisschen zu erweitern.

2 Informations- und Codierungstheorie

2.1 Geschichte und Anwendungsbereiche

Die Geburtsstunde der Informations- und Codierungstheorie schlug 1948, als Claude E. Shannon seine Arbeit *A Mathematical Theory of Communication* (The Bell System Technical Journal, Oktober, 1948) veröffentlichte. Folgender Absatz zitiert die Einleitung seiner Arbeit:

> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one *selected from a set* of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.

Der Begriff *Informationstheorie* wurde jedoch nicht von Shannon selbst geschaffen. Als genialer, aber bescheidener Ingenieur wollte er nur gewisse praktische Kommunikationsprobleme lösen.

Shannon beschrieb in seiner Informationstheorie jedoch nicht was unter dem ursprünglich betrachteten Begriff "Information" eigentlich zu verstehen ist, sondern nur den Informationsgehalt eines Zeichen und den mittleren Informationsgehalt einer Quelle, die als Entropie bezeichnet wird.

Der Begriff der Information hat Ähnlichkeit zu dem der Energie. Genau so wenig wie die Physik versucht, das "Wesen" der Energie zu beschreiben, versucht die Informationstheorie zu erklären, was Information eigentlich ist. Man kann in beiden Fällen aber die Transformation zwischen verschiedenen Formen quantitativ exakt erfassen. Bei der Übertragung von Information über gestörte Kanäle treten Verluste auf, die den Verlusten bei der Energieübertragung durch Reibung oder Abwärme entsprechen.

2.2 Informationstheorie

2.2.1 Nachricht und Information

Information: ein Bedeutungsunterschied, der in digitalen Systemen durch Bits/Bytes gemessen wird (Nachricht). Einen Bedeutungsüberschuss bezeichnet man als *Redundanz*.

Bei jeglicher Kommunikation werden Nachrichten ausgetauscht. Die Nachricht wird von einer **Quelle** (Sender) erzeugt und über einen **Kanal** (Übertragungsmedium) zur **Senke** (Empfänger) geschickt, der diese Nachricht interpretiert und in eine Information umwandelt. Die selbe Nachricht kann jedoch bei verschiedenen Empfänger zu unterschiedlichen Informationen führen.

Abbildung 2.2.1.1: Übertragungsschema

Quellen können die menschliche Stimme, ein Speichermedium mit digitalen Daten, Messinstrumente in einem Satelliten oder Sensoren in Sinnesorganen sein. Beispiele für Kanäle sind Telefonverbindungen, Richtfunkstrecken, Nervenbahnen und Datenleitungen in Computern

oder Netzwerken. Es ist häufig aber nicht einfach, Quelle und Kanal gegeneinander abzugrenzen.

Fast alle Kanäle unterliegen Störungen, die man üblicherweise als *Rauschen* bezeichnet. Diese Bezeichnung erklärt sich von selbst, wenn man einmal alte Schallplatten hört oder versucht, Radiosender auf der Kurzwelle zu empfangen. Das Rauschen stört die Übertragung der Information und verfälscht diese. Häufig stellt man die Störungen als zusätzliche Quelle dar, die das Rauschen in den (zunächst idealen) Kanal einspeist.

Redundanz. Viele Quellen besitzen *Redundanz*, d.h. eine gewisse Weitschweifigkeit oder Überbestimmtheit.

Ein einfaches Beispiel sind Datumsangaben wie "Montag, der 15.10.2001".Wenn keine Zweifel über den verwendeten Kalender bestehen, ist der Zusatz "Montag" überflüssig. Er ist aber dennoch sinnvoll, weil er die Datumsangabe absichert und verlässlicher macht.

Redundanz in der menschlichen Sprache entsteht auch daraus, dass nur ein extrem geringer Anteil der aller möglichen Buchstabenfolgen Wörter der deutschen (oder einer anderen) Sprache sind, und wiederum nur ein geringer Anteil der Folgen von Wörtern syntaktisch korrekte und semantisch sinnvolle Sätze ergeben. Deshalb können wir Texte trotz gelegentlicher Druckfehler einwandfrei lesen. Kurzum, *Redundanz sichert vor Übertragungsfehlern.*

Die Redundanz vieler Quellen lässt sich technisch schwer nutzen. Darum ist es häufig sinnvoll, die Redundanz der Quelle durch eine *Datenkompression* oder *Quellencodierung* zu eliminieren. Datenkompression ist uns aus der Computerwelt durch Programme wie zip bekannt, mit deren Hilfe redundante Daten auf kleinem Raum gespeichert werden können. In einem zweiten Schritt, der *Kanalcodierung*, wird gezielt Redundanz hinzugefügt, um die Datenübertragung gegen Störungen zu sichern.

Natürlich müssen Kanal- und Quellencodierung nach der Übertragung wieder rückgängig gemacht werden, wobei die *Kanaldecodierung* ein schwieriges Problem ist.

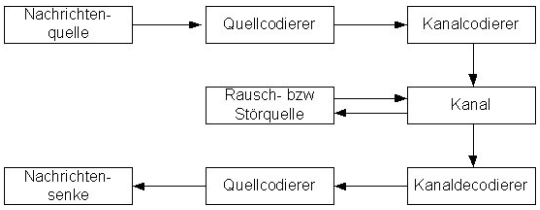

Abbildung 2.2.1.2: Realistisches Übertragungsschema

2.2.2 Entropie

Entropie als Begriff in der Informationstheorie ist in Analogie zur Entropie der Thermodynamik und Statischen Mechanik benannt. Beide Begriffe haben Gemeinsamkeiten, deren Erkennen allerdings Kenntnisse in beiden Fachgebieten vorrausetzt.

Die Entropie beschreibt den **mittleren Informationsgehalt** von Nachrichten aus Zeichen mit unterschiedlicher Häufigkeit wie folgt:

$$H(X) = \sum_{i=1}^{n} w(x_i) \cdot ld\frac{1}{w(x_i)} = \sum_{i=1}^{n} w(x_i) \cdot I(x_i)$$

$X = x_1, ..., x_N$ Zeichenvorrat einer Quelle
$w(x_1), ..., w(x_N)$ Zugehörige Auftrittswahrscheinlichkeit

$I(x_i)$ beschreibt den *Informationsgehalt eines einzelnen Zeichens* und hängt nur von der Auftrittswahrscheinlichkeit $w(x_i)$ab.

$$I(x_i) = ld\frac{1}{w(x_i)} \quad \text{in Bit}$$

Ein Maß für den Codieraufwand, der bei der Bildung einer Nachricht in einer Quelle entsteht, ist die *Anzahl der Binärentscheidungen* bis zur Auswahl eines bestimmten Zeichens. Ein Zeichenvorrat mit 2 Elementen führt auf eine binäre Entscheidung und hat das Maß von 1 Bit. Der **Entscheidungsgehalt** einer Quelle mit N Zeichen ist deshalb allgemein:

$$H_0 = ld(N) \quad \text{in Bit pro Zeichen}$$

Die Differenz zwischen dem Entscheidungsgehalt H_0 und der Entropie $H(X)$ wird als **Redundanz** R_Q der Quelle bezeichnet.

$$R_Q = H_0 - H(X)$$

Mit diesen Definitionen ist die Basis für eine Informationstheorie gelegt, die auch eine quantitative Betrachtung eines Systems zur Nachrichtenübertragung gestattet.

Die Informationstheorie nach Shannon orientiert sich an den *technischen Gegebenheiten* und lässt die Frage der **Interpretation** einer Nachricht und ihrer individuellen Bedeutung für die Senke völlig außer acht.

Die Quelle wird wie folgt beschrieben:
$X = x_1, ..., x_N$ Zeichenvorrat einer Quelle
$p(x_1), ..., p(x_N)$ Zugehörige Auftrittswahrscheinlichkeit

i	1	2	3	4	5	6	7	8
$P(x_i)$	0,4	0,15	0,12	0,1	0,08	0,06	0,05	0,04
$I(x_i)/Bit$	1,322	2,737	3,059	3,322	3,644	4,059	4,322	4,644
Entscheidungsgehalt der Quelle H_0							3 Bit/Zeichen	
Entropie der Quelle $H(X)$							2,58 Bit/Zeichen	
Redundanz der Quelle R_Q							0,42 Bit/Zeichen	

Tabelle 2.2.2.1: Beispiel für Quelle mit 8 Zeichen

Jedem Zeichen x_i wird ein Codewort zugeordnet und die jeweilige Codewortlänge wird mit $L(x_i)$ bezeichnet. Die **mittlere Codewortlänge** L wird in direkter Anlehnung an die Definition der Entropie wie folgt berechnet:

$$L = \sum_{i=1}^{N} p(x_i) \cdot L(x_i)$$

Die **Redundanz** R_C eines Quellcodierverfahrens ist definiert durch:

$$R_C = L - H(X)$$

Für das oben gezeigte Beispiel kann nun ein *Binärbaum* erstellt werden. Aus dieser Codierung ergeben sich folgende Werte:
mittlere Wortlänge $L = 2,62$ Bit pro Zeichen
Redundanz $R_C = 0,04$ Bit pro Zeichen

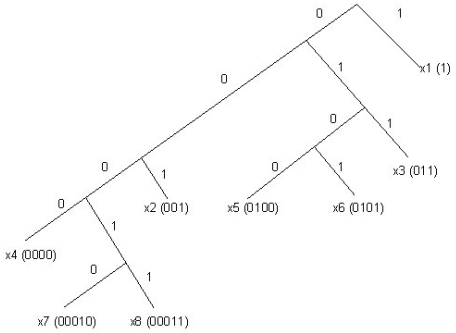

Abbildung 2.2.2.3: Binärbaum des vorherigen Beispiels

Die Entropie gibt also nicht nur den mittleren Informationsgehalt einer Quelle an, sondern gleichzeitig auch den *minimalen und ungefähren Codieraufwand* für die Zeichen einer Quelle.

Die effektivste Methode zur *Redundanz-Minimierung* gelang jedoch einem Schüler, *Huffman*. Es gibt keinen Code, der bei Codierung von Einzelzeichen eine geringere Redundanz aufweist als der nach Huffman erzeugte.

Als erster Schritt werden die Zeichen nach ihrer *Auftrittswahrscheinlichkeit* sortiert. Danach werden jene beiden Zeichen zu einem Knoten zusammengefasst, die die geringste Auftrittswahrscheinlichkeit besitzen. Dieser Vorgang wird dann wiederholt bis kein Knoten mehr übrig ist.

2.2.3 Kanalkapazität

Das Theorem von Claude Shannon [Shan48] liefert unabhängig vom Modulations- und Kodierverfahren eine Aussage über die maximale, fehlerfrei übertragbare Datenrate über einen gestörten Kanal. Diese *Kanalkapazität* stellt eine Obergrenze dar, die mit einem beliebig komplexen Modulations- und Codierverfahren erreicht werden kann. In der Praxis wird ein Verfahren immer unter der *Shannon-Grenze* liegen [Gapp99].

Die Kanalkapazität C [Bit/s] eines durch Gaußverteiltes Rauschen gestörten, bandbegrenzten Übertragungskanals mit der *Bandbreite* B, der empfangenen *Nutzsignalleistung* S_E und der *Störsignalleistung* S_N kann nach Shannon mit der Gleichung

$$C = B \cdot ld(1 + \tfrac{S_E}{S_N})$$

berechnet werden.

Ist der Kanal frequenzabhängig mit der Störleistungsdichte $S_{nn}(f)$ und der empfangenen Nutzleistungsdichte $S_{ee}(f)$, so berechnet sich die Kanalkapazität durch Integration über alle Frequenzen innerhalb der *Nutzbandbreite* B zu:

$$C = \int\limits_{f \in B} ld(1 + \frac{S_{ee}(f)}{S_{nn}(f)})df$$

Das Empfangssignal kann wahlweise auch aus dem *Sendeleistungsdichtespektrum* $S_{SS}(f)$ und dem Betrag der *Übertragungsfunktion* $|H(f)|$ berechnet werden:

$$C = \int ld(1 + \frac{S_{SS}(f) \cdot |H(f)|^2}{S_{NN}(f)})df$$

Hier wird ersichtlich, dass bei gegebener Übertragungsfunktion $H(f)$ und Störleistungsdichte $S_{nn}(f)$ die Kanalkapazität nur durch das Sendeleistungsdichtespektrums $S_{SS(f)}$ beeinflusst werden kann.

Ist die Sendeleistung SS beschränkt - das ist bei praktischen Systemen immer der Fall -, so wird die Kapazität *maximal* bei einer Verteilung der Sendeleistungsdichte nach der sogenannten *Water-pouring Methode* [Gall68]:

$$S_{SS}(f) = \begin{cases} S_{max} - \frac{S_{nn}(f)}{|H(f)|^2} & f \in B \\ 0 & f \notin B \end{cases}$$

wobei die Nebenbedingung:

$$S_S = \int\limits_{f \in B} \left[S_{max} - \frac{S_{nn}(f)}{|H(f)|^2} \right] df$$

erfüllt werden muss.

Bei gemessenen Kanaldaten liegen die Übertragungsfunktionen und Störspektren in Form von Abtastwerten $|H(f_i)|$ und $S_{nn}(f_i)$ im Abstand Δf mit $f_i = i * \Delta f$ vor. Die Berechnung der Kanalkapazität erfolgt dann über in eine Bestimmung der Kapazität von Schmalbandkanälen der Bandbreite Δf und Summation über alle Kanäle innerhalb der Nutzbandbreite B:

$$C = \Delta f \; ld\left(1 + \frac{S_{ee}(f_i)}{S_{nn}(f_i)}\right) = \Delta f \; ld\left(1 + \frac{S_{SS}(f_i) \cdot |H(f)|^2}{S_{nn}(f_i)}\right) \qquad \forall i \text{ mit } f_i \in B$$

Für die Berechnung der Kapazität in den folgenden Abschnitten wurde die Gleichung um den Faktor $SNRGap$ erweitert:

$$C = \Delta f \; ld\left(1 + \frac{S_{SS}(f_i) \cdot |H(f)|^2}{S_{nn}(f_i) \cdot SNRGap}\right) \qquad \forall i \text{ mit } f_i \in B$$

Dieser *frequenzunabhängige* Faktor SNRGap dient dazu, für realistischere Abschätzungen der tatsächlich erzielbaren Datenraten in der Praxis immer entstehende *Realisierungsverluste* oder *Sicherheitsfaktoren* zu integrieren [Frie98].

2.3 Codierungstheorie - Blockcodes

Die Codierungstheorie beschäftigt sich mit dem Entwurf und der Anwendung von *fehlererkennenden* und *fehlerkorrigierenden Codes* sowie den zugehörigen *(De-)Codierungsalgorithmen*.

Einfachste Beispiele fehlererkennender Codes sind die vielfach anzutreffenden *Prüfziffern*, fehlerkorrigierende Codes werden etwa bei Musik-CDs angewandt (*Reed-Solomon-Code*) oder bei D-Netz Telefonen (*Reed-Muller-Code*).

Da in der Praxis fast ausschließlich binäre Codes verwendet werden, d.h. Codes, die nur auf zwei Zeichen 0 und 1 beruhen, fasst man Codewörter im allgemeinen als *Elemente eines Vektorraums* über dem Körper mit zwei Elementen auf.

Falls die Menge aller Codewörter einen Untervektorraum bildet, spricht man von einem *linearen Code*. Solche Codes sind sehr einfach und schnell zu codieren und zu decodieren. Nichtlineare Codes haben demgegenüber den Vorteil, daß sie mit gleichem Aufwand an Prüfziffern im allgemeinen mehr Fehler erkennen und korrigieren können.

2.3.1 Hamming Verfahren

Das Hamming Verfahren ist eine Methode, um Fehler, die bei der Übertragung von Codewörter entstehen können, zu *erkennen* und falls möglich zu *korrigieren*.

Ein Code besteht aus einer fest vorgegebenen Menge von zulässigen Codewörtern einer konstanten Länge.

Der Hamming-Abstand zweier unterschiedlicher Codewörter ist die *Anzahl unterschiedlicher Bitpositionen* (d.h. die Anzahl 1 im Codewort, das durch bitweise XOR-Verknüpfung der beiden Wörter entsteht).

Der Hamming-Abstand des gesamten Codes wiederum ist der minimale Hamming-Abstand aller Kombinationen von Codewörtern und wird für einen gegebenen Code C mit $H(C)$ bezeichnet. Kennt man den Hamming-Abstand eines Codes, so kann man einfachste *Grundeigenschaften* des Codes betreffend Fehlererkennung und -korrektur finden.

$C = \{000000, 000111, 111000, 111111\}$
$H(x, y)$: Hamming-Abstand zwischen den Codewörtern x und y
$H(C)$: Hamming-Abstand des Codes C

$H(000000, 000111) = 3, \qquad H(111000, 000111) = 6$
$H(C) = min\{3, 6\} = 3$

Durch einfache Überlegungen kann man zeigen, dass für d und e, die die Gleichungen $H(C) = d + 1$ sowie $H(C) = 2e + 1$ erfüllen, gilt:

Für diesen Hamming-Code können bis zu d Fehler erkannt und bis zu e Fehler korrigiert werden. Ein Fehler ist in diesem Kontext ein *invertiertes* Bit des empfangenen Codewortes.

$$3 = H(C) = d + 1 = 2e + 1 \qquad \text{also } d = 2 \text{ und } e = 1$$

d.h. bis zu 2 Fehler sind zu erkennen und 1 Fehler kann korrigiert werden.

Fehlererkennungscodes hingegen können nur feststellen, dass Fehler aufgetreten sind, müssen aber den gesamten Datenblock neu anfordern.

gesendet	empfangen	korrigiert	Korrektur
000111	000011	000111	richtig (1-Bit-Fehler)
000000	000011	000111	falsch (2-Bit-Fehler)
000000	000111	Wort zulässig	(3-Bit-Fehler)

Tabelle 2.3.1.1: Beispiel für die Fehlererkennung von Hamming-Verfahren

Obige Beispiele zeigen, dass für gesendete Codewörter aus C tatsächlich gilt, dass 1 Fehler korrigierbar ist und bis zu 2 Fehler erkannt werden können. Treten 3 Fehler auf, so ist eine Erkennung nicht in jedem Fall möglich, denn ein 3-Bit-Fehler kann z.B. 000000 in 000111 überführen und so ein zulässiges Codewort zur Folge haben.

2.3.2 Algebraische Grundbegriffe

Definition eines *Galoisfelds (finite field)*:

Enthält die Menge K nur endlich viele (q) Elemente, so sprechen wir von einem endlichen Körper (finite field) oder auch von einem *Galoisfeld* $GF(q)$ zur Basis q. Galois-Felder existieren nur für $q = p^m$, wobei p eine *Primzahl* und m eine *natürliche Zahl* ist. Für ein Galoisfeld gelten ansonsten die gleichen Regeln wie für Körper. Ist $q = p$ eine Primzahl, sprechen wir auch von *Primkörpern*!

2.3.3 Binäre Gruppencodes

Ein Code C wird als binärer Gruppencode bezeichnet, wenn jede *modulo-2-Addition* von zwei gültigen Codewörtern wieder zu einem gültigen Codewort führt.

Der Vorteil dieser Gruppeneigenschaften liegt in den einfachen Berechnungsvorschriften, die bei der Codierung und bei der Decodierung angewendet werden können.

2.3.4 Zyklische Codes

Zyklische Codes sind eine wichtige Klasse im Bereich der Gruppencodes. Sie haben große praktische Bedeutung und werden z.B. eingesetzt für digitale Mobilfunksysteme, Compact Disc (*CD*), Digitale Kompaktkassette (*DCC*) und das Radio-Daten-System (*RDS*).

Ein Gruppencode heißt *zyklisch*, wenn sämtliche zyklischen Verschiebungen eines Codewortes wieder in dem Code sind. Codierung und Decodierung lassen sich technisch mit Hilfe von einfachen Schieberegisterschaltungen durchführen.

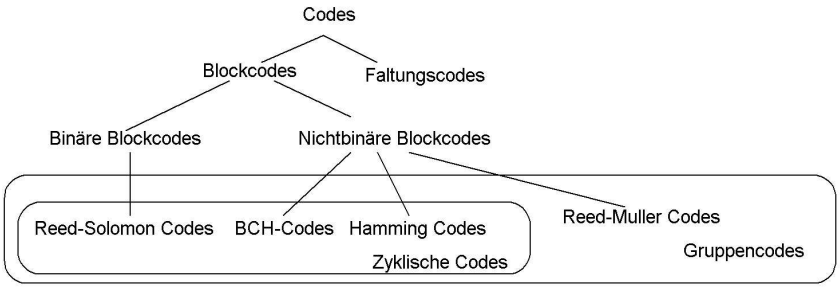

Abbildung 2.3.4.1: Codes im Überblick

2.4 Codierungstheorie - Faltungscodes

Im Gegensatz zu Blockcodierern, die jeweils Codewörter der festen Länge N bilden, werden bei Faltungscodierern die Nachrichtenbits *kontinuierlich* eingegeben und codiert.

Faltungscodes wurden erstmals 1955 vorgestellt, galten zunächst jedoch als nicht leistungsfähig genug. Heute kann die prinzipiell hinter Faltungscode stehende hohe Leistungsfähigkeit auch praktisch genutzt werden, und sie werden auch in vielen nachrichtentechnischen Systemen eingesetzt.

Beispiel für einen Faltungscodierer. Bei der Codierung wird die Datenbitfolge (im Beispiel 011010) schrittweise in ein *Schieberegister* mit $L(=3)$ Zellen eingegeben. In jedem Codierschritt werden $b = 1$ Datenbits in das Schieberegister eingegeben, und es werden $n = 2$ Ausgabebits durch modulo-2-Addition der Inhalte geeignet gewählter Zellen des Schieberegisters gebildet. Der Schieberegisterinhalt wird auch als *Zustand des Codierens* bezeichnet.

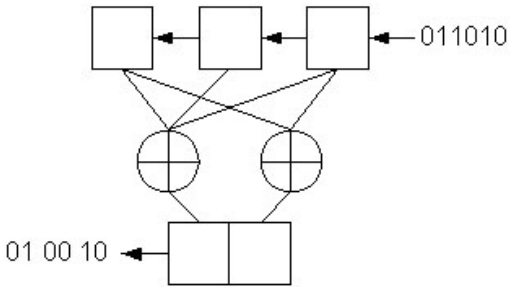

Abbildung 2.4.0.1: Schema eines Faltungscodierers

Die *Coderate* R des Faltungscodes beschreibt das Verhältnis $R = \frac{b}{n}$ zwischen der Anzahl der Ein- und Ausgabebits in der Schieberegisterschaltung. Die Coderate drückt das Verhältnis zwischen dem *Nutzdaten-* und dem *Redundanzanteil* innerhalb des Faltungscodes aus. In unserem Beispiel haben wir eine Coderate von $R = \frac{1}{2}$.

Wie bei Blockcodes können auch bei Faltungscodes um so mehr Übertragungsfehler korrigiert werden, je größer der *minimale Hamming-Abstand* zweier unterschiedlicher Codebitfolgen ist.

Bei der Decodierung wird nicht ein einzelnes empfangendes Symbol, sondern eine gesamte Sequenz bearbeitet. Dabei werden alle möglichen Sendsymbolfolgen mit der jeweiligen Empfangsfolge verglichen, und es wird die Symbolfolge ausgewählt, die mit höchster Wahrscheinlichkeit gesendet wurde.

Man spricht von einem *Maximum-Likelihood-Empfänger*.

3 Kryptologie und Datensicherheit

3.1 Einführung

Datensicherheit hat die Sicherung von Daten vor verschiedensten möglichen Bedrohungen zum Ziel, *Kryptologie* bietet die Techniken und Möglichkeiten zum Erreichen dieses Ziels.

Die Kryptologie (von griech. *kryptós* - geheim, verborgen), also ursprünglich die Lehre von den Geheimsprachen, kann in zwei Gebiete unterteilt werden: die *Kryptographie* beschäftigt sich damit, Kryptosysteme, das heißt Algorithmen zur Verschlüsselung von Information zu entwickeln, während man unter *Kryptoanalyse* - als Umkehrung dazu - das Brechen solcher Systeme, also das Dechiffrieren einer Nachricht ohne Kenntnis des Schlüssels, versteht.

Geschichte. Die älteste bekannte Verwendung einer Chiffre ist die *Skytala*, die vor ca. 2400 Jahren in Griechenland vor allem zur militärischen Nachrichtenübermittlung verwendet wurde. Das Verfahren beruhte auf *Transposition*, das heißt auf Veränderung der Anordnung der Schriftzeichen.

Bis in die 1970er Jahre blieb die Kryptologie eine Domäne von Geheimdienst und Militär, der Schutz der Vertraulichkeit stand im Vordergrund.

In der modernen Informations- und Kommunikationstechnik wird sie verwendet, um die drei wichtigsten Aspekte der **Datensicherheit** zu garantieren:

1. *Vertraulichkeit* : Unbefugte sollen die Nachricht nicht lesen können.
2. *Integrität* : Informationen dürfen nicht unerlaubt manipuliert werden.
3. *Authentizität* : Die Identität des Senders einer Nachricht muss nachprüfbar sein.

Je nach Art der verwendeten Schlüssel kann man dafür drei Klassen von **kryptographischen Mechanismen** unterscheiden:

1. *Schlüssellose* Mechanismen wie Einweg- und Hashfunktionen, die vorwiegend zum Schutz der Integrität verwendet werden.
2. *Symmetrische* Mechanismen, bei denen Sender und Empfänger den gleichen Schlüssel verwenden, können Vertraulichkeit und Integrität garantieren.
3. *Asymmetrische* Mechanismen können durch Verwendung unterschiedlicher Schlüssel bei Sender und Empfänger Integrität und Verbindlichkeit sicherstellen.

Neben ihrer Eignung für bestimmte Anwendungen lassen sich die verschiedenen Verschlüsselungsverfahren auch aufgrund der *Komplexität* des Schlüsselmanagements, des *Implementierungsaufwandes* oder der erreichbaren *Verschlüsselungsrate* unterscheiden.

Die Verfahren lassen sich über beliebigen Verfahren definieren, in der modernen Datentechnik verwendet man jedoch vornehmlich das *binäre Alphabet*. Klartexte, Chiffretexte und Schlüssel sind somit Bitfolgen.

Kryptoanalytische Angriffe. Sie haben die Ermittlung von Schlüssel und/oder Klartext zum Ziel. Man unterscheidet drei Angriffsarten:

1. *Brute-Force Angriff*: Durch rohe Gewalt"werden alle möglichen Schlüssel generiert und ausprobiert.

2. *Geheimtextangriff*: Mit Hilfe statistischer Auswertungen der Geheimtexte wird versucht, Informationen über Inhalt und Verschlüsselungsverfahren zu gewinnen.

3. *Klartextangriff*: Aufgrund bekannter Klartextteile und dazugehörigen Chiffren zieht man Rückschlüsse auf Verfahren und Schlüssel.

Perfekte Verschlüsselungssysteme. Ein Verschlüsselungsverfahren heißt perfekt, wenn die Kenntnis des Chiffretextes dem Kryptoanalytiker keine zusätzliche Information über den Klartext gibt. Der Schlüssel muss dann mindestens so lange sein wie der Klartext.

Hier ist genau ein Verfahren bekannt, der sogenannte *One Time Pad*, der schon 1917 entwickelt wurde. Dabei wird die Klartext-Bitfolge zeichenweise mit einer echt zufälligen Schlüssel-Bitfolge mittels XOR verknüpft.

3.2 Symmetrische Kryptographie

Der Empfänger benötigt für die Entschlüsselung den gleichen Schlüssel, der auch zur Verschlüsselung verwendet wurde. Aus diesem Grund ist es hier unbedingt erforderlich, dass der Schlüssel geheim bleibt.

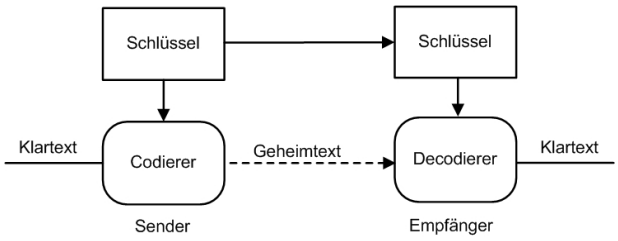

Abbildung 3.2.0.1: Symmetrische Kryptographie

Man unterscheidet hier block- und stromorientierte Verfahren. Eine *Blockchiffre* ordnet jeweils grösseren Klartextblöcken Chiffretextblöcke zu, bei *Stromchiffren* werden einzelne Zeichen des Klartextes durch XOR mit einem Schlüsselstrom verknüpft.

3.2.1 Entwurfskriterien

Ein Kryptosystem heißt *praktisch sicher*, wenn es kein Verfahren gibt, das dieses System mit aktuell verfügbaren Ressourcen und vertretbaren Kosten brechen kann.

Die Minimalanforderung ist daher aus der heute verfügbaren Rechenleistung ableitbar. Ein symmetrisches Kryptosystem sollte zumindest 64-Bit-, besser 80- oder 128-Bit-Schlüssel verwenden, um einen einfachen Brute-Force Angriff unmöglich zu machen.

3.2.2 Stromchiffren

Bei diesen Verfahren werden Klartextzeichen sequentiell - meist durch binäre Addition (XOR) mit einem Schlüsselstrom verknüpft.

Im Unterschied zum One Time Pad werden in der Praxis für die Schlüssel meist Pseudozufallsfolgen verwendet, die deterministisch aus einem zufälligen Initialisierungswert generiert werden.

3.2.3 Symmetrische Blockchiffren

Die Nachrichten werden hier nicht zeichenweise behandelt, sondern in Abschnitte fester Länge zerlegt, die nacheinander verschlüsselt werden.

Typische Blocklängen sind 64 oder 128 Bit. Zusätzliche kryptographische Sicherheit erzielt man durch die Iteration von Grundfunktionen.

Data Encryption Standard (DES). Bei dem 1976 als US-Bundesstandard anerkannte DES handelt es um eine auf 64-Bit-Blöcken arbeitende Blockchiffre.

Aus 64 Bit großen Blöcken des Klartextes erzeugt der symmetrische Algorithmus 64 Bit große Chiffre-Blöcke. Der verwendete Schlüssel ist ebenfalls 64 Bit lang. Da jedes Byte ein *Paritätsbit* aufweist, stehen allerdings effektiv nur 56 Bit zur Ver- und Entschlüsselung zur Verfügung.

Seine Sicherheit beruht auf einer Kombination von *Permutation* und *Substitution*. Die Ver- bzw. Entschlüsselung eines 64-Bit-Blocks kann man in drei Schritte unterteilen:

1. Der Eingabeblock wird einer festen Eingangspermutation unterworfen, die die Reihenfolge der Bits verändert (*Transposition*), das Ergebnis wird in zwei 32-Bit-Schieberegister geschrieben. Mit den 56 relevanten Schlüsselbits geschieht dasselbe (allerdings zweimal 28 Bit).

2. Nun werden die 32-Bit-Blöcke 16 gleichartigen Verschlüsselungsrunden unterzogen, wobei in jeder Runde ein neuer 48-Bit-Schlüssel bestimmt wird. Das rechte Datenregister wird auf 48 Bit expandiert und mit einem 48-Bit-Teilschlüssel XOR-verknüpft, dann werden je 6 Bits durch einen 4-Bit-Block ersetzt. Schließlich erfolgt eine Permutation der resultierenden 32 Bits und ihre XOR-Verknüpfung mit dem linken Register.

   Das Ergebnis bildet den neuen Inhalt des rechten Registers, der alte R-Block wird zuvor ins linke Register geschrieben.

3. Nach 16 Runden werden die beiden Register wieder zu einem 64-Bit-block zusammengefügt und eine zur Eingangspermutation inverse Ausgangspermutation angewendet.

Zur Entschlüsselung müssen lediglich die Teilschlüssel der 16 Runden in umgekehrter Reihenfolge angewandt werden.

Der große Schwachpunkt des DES liegt allerdings in der *geringen Mächtigkeit des Schlüsselraumes* von 2^{56}, da dieser einem Brute-Force Angriff bei Vernetzung genügend vieler Rechner keine 24 Stunden standhalten kann.

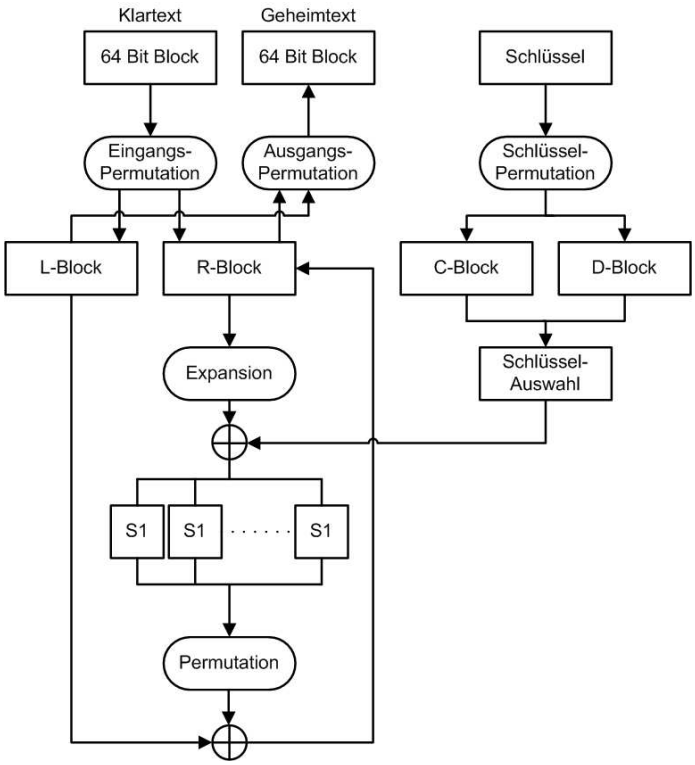

Abbildung 3.2.3.2: DES

International Data Encryption Algorithm (IDEA). Dieser 1990 als Alternative zum DES entwickelte Algorithmus bietet neben einen grösseren Schlüsselraum (er arbeitet mit einem *128-Bit Schlüssel*) auch eine leichtere Implementierbarkeit in Hard- und Software.

Es handelt sich hier ebenfalls um eine 64-Bit-Blockchiffre, bei der die Daten aber nur durch 8 Runden wiederholter Operationen sowie eine zusätzliche Output-Transformation verschlüsselt werden. Statt Substitutionen und Bit-Permutationen wird lediglich mit drei arithmetischen Elementaroperationen gearbeitet, wodurch sich eine geringere Rechenzeit bei gleichzeitig höherer Sicherheit ergibt:

1. *Bitweises XOR* - entspricht auch der Addition zweier Zahlen ohne (bitweisen) Übertrag.
2. *Addition zweier Zahlen modulo* 2^{16}, wobei das Ergebnis auf einen 16-Bit-Wert beschränkt bleibt.
3. *Multiplikation zweier Zahlen modulo* $2^{16}+1$, besondere Behandlung für die Randbereiche.

IDEA besitzt also eine interne 16-Bit-Struktur; alle Grundfunktionen arbeiten auf 16-Bit-Blöcken.

Advanced Encryption Standard (AES). 1996 wurde vom US-amerikanischen *National Institute of Standards and Technology (NIST)* ein Wettbewerb zur Ermittlung eines Nachfolgers für den DES ausgeschrieben. Beim Sieger des Wettbewerbs handelt es sich um eine symmetrische Blockchiffre mit variablen Block- und Schlüssellängen von 128 bis 256 Bit. Ein Klartextblock

wird in mehreren Runden verschlüsselt, deren Anzahl hängt von der jeweiligen Schlüssel- und Blocklänge ab.

Anzahl Runden	Blocklänge		
Schlüssellänge	128	192	256
128	**10**	**12**	**14**
192	**12**	**12**	**14**
256	**14**	**14**	**14**

Tabelle 3.2.3.1: Anzahl der Runden in Abhängigkeit von Block- und Schlüssellänge (Angaben in Bit)

Für die Ver- und Entschlüsselung wird für jede Runde ein neuer Rundenschlüssel erzeugt. Hierfür wird der geheime Schlüssel des Anwenders durch Anhängen rekursiv abgeleiteter 4-Byte-Wörter expandiert.

Beim Entwurf des AES wurden alle bekannten Attacken der Kryptoanalyse berücksichtigt, in den Prüfungen waren keine Sicherheitslücken mehr feststellbar. Gemäss NIST bräuchte ein Rechner, der DES in einer Sekunde knacken könnte, dür AES mit 128-Bit-Schlüssel ca.149 Billionen Jahre Rechenzeit.

3.2.4 Authentifizierungs-Codes

Zur elektronischen Prüfung der Datenintegrität wird eine Klartextnachricht M um redundante Information erweitert, die durch ein kryptographisches Verfahren aus M berechnet und zusammen mit ihr gespeichert oder übertragen wird. Dieser *Authentifizierungscode (message authentication code,* kurz: *MAC)* kann auf der Empfängerseite die Authentizität der Nachricht verifizieren.

Zur MAC-Berechnung gibt es standardisierte Verfahren, die Blockchiffren oder Hashfunktionen verwenden. Als Mindestlänge für einen MAC gelten 32 Bit.

Zur Erkennung aktiver Beeinträchtigungen wie Löschen, Verzögern oder Vertauschen abgehörter Nachrichten sind zusätzliche Massnahmen wie eindeutige Nummerierung der Nachrichten oder Zeitstempel erforderlich.

3.3 Schlüssellose kryptographische Mechanismen

Im Gegensatz zu symmetrischen Kryptoverfahren entfällt hier die Notwendigkeit der Geheimhaltung des Schlüssels bzw. aufwendiges Schlüsselmanagement.

Schlüssellose Verfahren sind für Anwendungen, die *keine Umkehrbarkeit* der kryptographischen Verschlüsselung erfordern, zum Beispiel die verschlüsselte Speicherung von Passwörtern, gut geeignet.

3.3.1 Einwegfunktionen

Bei einer *Einwegfunktion* f ist es unmöglich, aus einem gegebenen Funktionswert y ein Argument x mit der Eigenschaft $f(x) = y$ zu bestimmen, obwohl $f(x)$ bei gegebenem x ohne Probleme berechnet werden kann.

Einwegfunktionen können zum Beispiel mit Hilfe von symmetrischen Blockchiffren, die gegen Klartextangriffe resistent sind, konstruiert werden.

3.3.2 Kryptographische Hashfunktionen

Eine *kryptographische Hashfunktion* komprimiert beliebig lange Daten zu einem Hashwert fester Länge (meist 128 oder 160 Bit). Es wird praktisch unmöglich, ein Paar verschiedene Inputwerte mit übereinstimmenden Hashwerten zu finden.

Meist wird die Nachricht durch eine Folge gleichartiger Kompressionsschritte blockweise zu einem Hashwert umgewandelt.

Diese Methode findet hauptsächlich im Bereich *digitaler Signaturen* und der *Software-Versiegelung* Verwendung.

3.4 Asymmetrische Kryptographie

Als asymmetrische Verschlüsselungsverfahren werden Verfahren bezeichnet, bei denen zwischen den Schlüsseln zur Chiffrierung bzw. Dechiffrierung kein einfacher Zusammenhang besteht, sodass es unmöglich ist, ohne zusätzliches Wissen von einem Schlüssel auf den anderen zu schließen. Hier wird also immer mit einem geeigneten *Schlüsselpaar* gearbeitet.

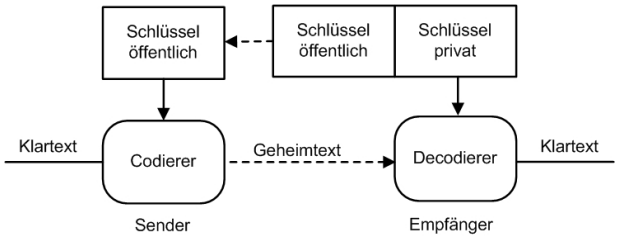

Abbildung 3.4.0.1: Asymmetrische Kryptographie

3.4.1 Öffentliche und private Schlüssel

Das Grundkonzept asymmetrischer Kryptographie beruht darauf, dass jeder Teilnehmer T einen öffentlichen Schlüssel K_T (*public key*), der veröffentlicht wird, und einen privaten Schlüssel ${K_T}'$ (*private key*), der nur ihm selbst bekannt ist, hat.

Wichtig dabei ist, dass es praktisch unmöglich sein muss, aus der Kenntnis von K_T den Wert des zugehörigen ${K_T}'$ zu berechnen.

Es gibt drei Arten asymmetrischer Verfahren:

1. *Asymmetrische Verschlüsselung* : Jeder, der den public key K_A von A kennt, kann A ohne vorherigen Kontakt eine (mit K_A) verschlüsselte Nachricht senden, die A dann mit seinem private key ${K_A}'$ entschlüsselt.

2. *Asymmetrische Signatur* : Wenn A zu einem Datensatz M mit ihrem private key ${K_A}'$ eine Signatur erstellt, so kann jeder, der den public key K_A besitzt, verifizieren, ob diese Signatur von A erstellt wurde.

3. *Asymmetrische Schlüsselvereinbarung* : Mit Hilfe einer speziellen Einwegfunktion $f({K_A}', K_B) = f({K_B}', K_A)$ können sich A und B jeweils gegenseitig ihre öffentlichen Schlüssel K_A oder K_B zusenden. Wenden dann beide auf den empfangenen Wert ihre privaten Schlüssel an, erhalten sie das gleiche Ergebnis $f({K_A}', K_B) = f({K_B}', K_A)$, das bei geeigneter Wahl von f niemand sonst kennt.

Vorteile asymmetrischer Kryptosysteme sind die Möglichkeit, eine zur herkömmlichen Unterschrift äquivalente elektronische Signatur zu erzeugen, sowie erleichtertes Schlüsselmanagement.

Der **Nachteil** liegt in der schlechten Performanz gegenüber symmetrischen Systemen, diese sind in der Regel tausendfach schneller.

Darum verwendet man meist *Hybridsysteme* mit asymmetrischer Schlüsselvereinbarung und symmetrischer Verschlüsselung der Klartextdaten.

3.4.2 RSA

Das 1977 von Rivest, Shamir und Adleman entwickelte RSA-Verfahren benutzt ein bekanntes Problem aus der Numerik: Es ist zwar einfach, zwei *große Primzahlen* zu finden und zu multiplizieren, aber praktisch unmöglich, diese Faktoren herauszufinden, wenn man nur das Produkt kennt.

Die Schlüsselerzeugung geschieht folgendermassen:

- Jeder Teilnehmer wählt zwei verschiedene, große Primzahlen p und q, wobei die Differenz nicht zu gering sein sollte, und berechnet das Produkt m.
- Ein zufälliger Wert e wird ermittelt, der kleiner m und teilerfremd zu $(p-1)*(q-1)$ ist. Dann wird die modulare Inverse d so berechnet, dass $(e*d)mod((p-1)*(q-1)) = 1$
- Als öffentlicher Schlüssel gilt e und m, als privater Schlüssel d und m. Die Primzahlen p und q dürfen nicht bekannt werden.

Die Chiffrierung und Dechiffrierung von Nachrichten ist dann vergleichsweise einfach: Der Klartext wird in eine Zahlenfolge übersetzt (*numerical encoding*) und dann in Blöcke der Grösse *Bitlänge(Block)* $\leq$ *Bitlänge*$(m) - 1$ zerlegt.
Verschlüsselt wird mit $C = M^e mod\ m$, entschlüsselt mit $M = C^d mod\ m$.

Das Verfahren ist sicher, solange m nicht faktorisiert werden kann.

3.4.3 Diskreter Logarithmus in endlichen Gruppen

Einigen asymmetrischen Kryptosystemen liegt das Problem des diskreten Logarithmus (DL) in endlichen Gruppen zugrunde.

Sei $G = \{1, g, g^2, ..., g^{n-1}\}$ eine endliche zyklische Gruppe mit n Elementen und g^x mit $x \in \mathbb{N}$ eine Einwegfunktion.

Das bedeutet, die Multiplikation $g * g$ ist einfach, die Berechnung des DL $\log_g y$ bei entsprechend großen Gruppen aber praktisch unmöglich.

Für die *Schlüsselgenerierung* benutzen alle Teilnehmer dieselbe Gruppe G und dasselbe erzeugende Element g. Jeder Teilnehmer T wählt dann eine zufällige Zahl $t < n$ als privaten und g^t als öffentlichen Schlüssel.

Diffie-Hellman Schlüsselvereinbarung. A sendet seinen public key g^a an B, B seinen g^b an A. Den empfangenen Wert potenzieren beide mit ihren private keys. Da $(g^a)^b = (g^b)^a = g^{ab}$ haben A und B dadurch einen gemeinsamen Schlüssel g^{ab} vereinbart, den sonst niemand kennt.

ElGamal Verschlüsselung. Hier handelt es sich um ein Hybridverfahren mit Diffie-Hellman-ähnlicher Schlüsselvereinbarung und darauffolgender symmetrischer Verschlüsselung.

A wählt eine zufällige natürliche Zahl $x < n$, berechnet einen temporären public key g^x und mit dem public key g^b von B einen gemeinsamen Schlüssel $K = (g^b)^x$. Dann verschlüsselt A den Klartext mit K und sendet ihn gemeinsam mit g^x an B. B kann $K = (g^x)^b$ ermitteln und die Nachricht dechiffrieren.

3.5 Authentifizierung

Sie ist die Grundlage aller Sicherheitssysteme. Zum Nachweis der Identität gibt es prinzipiell drei Methoden:

1. *Besitz*, z.B. einer Chipkarte
2. *Merkmale*, z.B. einen Fingerabdruck
3. *Wissen*, z.b. eines Kennwortes

In der Praxis werden meist zwei Methoden, z.B. Chipkarte und zugehöriger Zifferncode, kombiniert verwendet.

ad 3. Bei räumlicher Entfernung von authentifizierender Instanz A und verifizierender Instanz B findet meist Methode 3 Verwendung. Hier unterscheidet man wiederum zwei Möglichkeiten:

- *statische Authentifizierung*: A sendet jedesmal die gleiche Sequenz, z.B. ein bestimmtes Kennwort, an B
- *dynamische Authentifizierung*: Dies ist die sicherere Variante. Hier wird jedesmal eine andere *nonce* (*n*umber used only *once*) verschlüsselt. Dies kann eine von B zuvor generierte und an A gesendete Zufallszahl, ein Zeitstempel oder auch eine laufende Seriennummer sein.

3.6 Schlüsselmanagement

Die Aufgabe des Schlüsselmanagements liegt in der Bereitstellung und Kontrolle von Schlüsseln, was vor allem die Erzeugung, Verteilung, Speicherung und Zerstörung derselben beinhaltet.

3.6.1 Public-key-Infrastruktur

Zu den Funktionen des **PKI** zählen alle für den Einsatz von asymmetrischer Kryptographie in offenen Systemen notwendigen Massnahmen wie die *Nutzerregistrierung* oder das Ausstellen, Verwalten und Prüfen von *Zertifikaten*.

Letztere werden von *Zertifizierungsstellen* als Garantie für die Integrität von öffentlichen Schlüsseln und die Authentizität ihrer Eigentümer vergeben. In einem Zertifikat sind neben Benutzername und zugehörigem öffentlichen Schlüssel unter anderem auch ein Gültigkeitszeitraum, der Name der Zertifizierungsstelle sowie Angaben über das zur Zertifizierung verwendete Signaturverfahren enthalten.

3.6.2 Schlüsselverteilung

Die Punkt-zu-Punkt-Schlüsselverteilung oder -Schlüsselvereinbarung sind weitere wichtige Bestandteile des Schlüsselmanagements.

Bei *Schlüsselvereinbarungen* leisten Sender und Empfänger gleichermassen einen Beitrag zur Generierung des Schlüssels.

Ein für symmetrische Mechanismen notwendiger *Schlüsseltransport* setzt voraus, dass Sender und Empfänger bereits über einen gemeinsamen Schlüssel zur sicheren Übertragung des neuen Schlüssels verfügen.

Für die Schlüsselverteilung werden Mechanismen zur direkten Erzeugung eines geheimen Schlüssels zwischen zwei Partnern sowie Mechanismen zur geheimen Schlüsselübertragung mittels asymmetrischer Verfahren oder Mechanismen zur sicheren Verteilung öffentlicher Schlüssel mittels Zertifizierung verwendet.

3.6.3 Schlüsselhinterlegung/-offenlegung

Der einfache Zugang zu modernen Chiffrierverfahren führt zu einem Konfliktpotential zwischen dem Interesse der Nutzer einerseits und andererseits dem von Strafverfolgungsbehörden, da er das Abhören in gesetzlich zulässigen Fällen nutzlos machen kann.

Einen Lösungsansatz für dieses Problem bietet zum Beispiel die Verwendung *"schwacher" Verschlüsselungsverfahren* wie symmetrische Chiffren mit 40-Bit-Schlüsseln und die Hinterlegung bzw. Offenlegung von Dechiffrierschlüsseln (*key-escrow-* und *key-recovery-* Mechanismen).

A Literaturverzeichnis

A.1 Allgemeine Literatur

[1] Rechenberg, Peter; Pomberger, Gustav: Informatik-Handbuch, CARL HANSER VERLAG

[2] Haring, Günter; Kurz, Christian: Skriptum zur Vorlesung Grundzüge der Informatik I, Institut für Informatik und Wirtschaftsinformatik Abteilung Verteilte Systeme, Universität Wien

[3] Wismüller, Roland: Skriptum zur Vorlesung Einführung in die Technische Informatik I, Department of Software Science, Universität Wien

[4] Tavangarian, D.: Skriptum zur Übung Rechnersysteme I, WS 01/02, Institut für Technische Informatik, Universität Rostock

[5] Online-Recherchenkompass: Kapitel Kryptologie http://home.nwn.de/hgm/krypto/index.html

A.2 Spezielle Literatur

[Shan48] Shannon, Claude E.: A Mathematical Theory of Communications. Bell Syst. Techn. Journal, vol. 27 (1948); pp. 379-423 (July) und 623-656 (Oktober)

[Gapp99] Gappmair, Wilfried: The 50th Anniversary of Information Theory. IEEE Comunicatinos Magazine, vol. 37, No.9,Sept 1999, pp. 102-105.

[Gall68] Gallager, R. G.: Information Theory and Reliable Communications. John Wiley, New York, 1968

[Frie95] Friedrichs, Bernd: Kanalcodierung Springer, Berlin, 1995