Prof. Dr. Jürgen Dassow
und
Dr. Bianca Truthe

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik

C O D I E R U N G S T H E O R I E

und

K R Y P T O G R A P H I E

Vorlesungsmanuskript

Magdeburg, April – Juli 2011

Inhaltsverzeichnis

Kapitel 1

Definition und Charakterisierung von Codes

1.1 Definition von Codes

Gegeben sei eine Menge A von zu codierenden Objekten. Um eine Codierung vorzunehmen, ist sicher erst einmal jedem Element aus A ein Element eines Codes C zuzuordnen. Offensichtlich muss die dadurch definierte Funktion eineindeutig sein, wenn wir eine eindeutige Decodierung erwarten.

Diese Forderung reicht aber nicht aus, wenn wir Nachrichten übermitteln wollen, die aus Folgen der zu codierenden Objekte aus A bestehen. Um dies zu sehen, betrachten wir die Mengen

$$A = \{A_1, A_2, A_3\} \quad \text{und} \quad C_0 = \{a, ba, ab\},$$

deren Elemente vermöge

$$A_1 \leftrightarrow a, \quad A_2 \leftrightarrow ba, \quad A_3 \leftrightarrow ab$$

einander eineindeutig zugeordnet werden können. Empfangen wir das Wort aba, so ist nicht zu erkennen, ob die Nachricht A_1A_2 oder A_3A_1 gesendet wurde.

Die folgende Definition berücksichtigt die beiden genannten Aspekte.

Definition 1.1 *Eine eineindeutige Funktion $\varphi : A \to C$ ist eine Codierung der Menge A durch die nichtleere Sprache C über einem Alphabet X, wenn die homomorphe Erweiterung[1] φ^* von φ auf A^* eine injektive Funktion von A^* in X^* ist.*

Eine nichtleere Sprache C (über X) heißt Code, wenn C Wertevorrat einer Codierung ist.

Die obige Definition bindet den Begriff Code an den einer Codierung. Eine Feststellung, ob eine Sprache C ein Code ist, erfordert daher das Auffinden einer zu codierenden Menge A. Hierfür eignet sich aber jede zu C gleichmächtige Menge, so dass die Wahl von A

[1] Für eine Abbildung $\alpha : A \to X^*$, die eine Menge A auf Wörter über einem Alphabet X abbildet, ist die homomorphe Erweiterung $\alpha^* : A^* \to X^*$ durch $\alpha(\lambda) = \lambda$ und $\alpha(a_1a_2 \dots a_n) = \alpha(a_1)\alpha(a_2) \dots \alpha(a_n)$ für $a_i \in A$, $1 \leq i \leq n$, definiert.

intuitiv bedeutungslos ist. Der nachfolgende Satz gibt eine Bedingung an, die äquivalent zur Codedefinition ist, aber nur die Menge C selbst betrifft.

Satz 1.1 *Eine nichtleere Sprache C ist genau dann ein Code, wenn für beliebige Wörter*

$$x_{i_1}, x_{i_2}, \ldots, x_{i_n}, x_{j_1}, x_{j_2}, \ldots, x_{j_m} \in C,\ n \geq 1, m \geq 1$$

die Beziehung

$$x_{i_1}x_{i_2}\ldots x_{i_n} = x_{j_1}x_{j_2}\ldots x_{j_m} \quad \textit{impliziert} \quad x_{i_1} = x_{j_1} \tag{1.1}$$

gilt.

Beweis. Wir nehmen zuerst an, C sei ein Code. Dann gibt es eine Menge A und eine Codierung $\varphi : A \to C$. Es seien

$$a_{i_t} = \varphi^{-1}(x_{i_t}) \quad \text{und} \quad a_{j_s} = \varphi^{-1}(x_{j_s})$$

für $1 \leq t \leq n$ und $1 \leq s \leq m$. Dann gilt

$$\varphi^*(a_{i_1}a_{i_2}\ldots a_{i_n}) = x_{i_1}x_{i_2}\ldots x_{i_n} = x_{j_1}x_{j_2}\ldots x_{j_m} = \varphi^*(a_{j_1}a_{j_2}\ldots a_{j_m}).$$

Aus der Injektivität von φ^* folgt nun

$$a_{i_1}a_{i_2}\ldots a_{i_n} = a_{j_1}a_{j_2}\ldots a_{j_m}.$$

Damit erhalten wir $a_{i_1} = a_{j_1}$ und folglich

$$x_{i_1} = \varphi(a_{i_1}) = \varphi(a_{j_1}) = x_{j_1},$$

womit Bedingung (1.1) nachgewiesen ist.

Erfülle nun die Menge C die Bedingung (1.1). Es sei A eine zu C gleichmächtige Menge. Dann gibt es eine eineindeutige Funktion $\varphi : A \to C$. Wir zeigen nun indirekt, dass auch die Erweiterung von φ auf A^* injektiv ist.

Angenommen, die Erweiterung ist nicht injektiv. Dann gibt es Elemente $a_{i_1}, a_{i_2}, \ldots, a_{i_n}$, $a_{j_1}, a_{j_2}, \ldots, a_{j_m}$ so, dass

$$a_{i_1}a_{i_2}\ldots a_{i_n} \neq a_{j_1}a_{j_2}\ldots a_{j_m} \tag{1.2}$$

und

$$\varphi^*(a_{i_1}a_{i_2}\ldots a_{i_n}) = \varphi^*(a_{j_1}a_{j_2}\ldots a_{j_m}) \tag{1.3}$$

gelten. Wir wählen nun diese Elemente mit diesen Eigenschaften so, dass m minimal ausfällt. Weiterhin setzen wir

$$x_{i_t} = \varphi(a_{i_t}) \quad \text{und} \quad x_{j_s} = \varphi(a_{j_s})$$

für $1 \leq t \leq n$ und $1 \leq s \leq m$. Dann gilt

$$x_{i_1}x_{i_2}\ldots x_{i_n} = \varphi^*(a_{i_1}a_{i_2}\ldots a_{i_n}) = \varphi^*(a_{j_1}a_{j_2}\ldots a_{j_m}) = x_{j_1}x_{j_2}\ldots x_{j_m}.$$

Wegen Bedingung (1.1) erhalten wir $x_{i_1} = x_{j_1}$. Damit gilt $\varphi(a_{i_1}) = \varphi(a_{j_1})$, woraus $a_{i_1} = a_{j_1}$ folgt. Somit ergeben sich

$$a_{i_2}a_{i_3}\dots a_{i_n} \neq a_{j_2}a_{j_3}\dots a_{j_m}$$

(die Gleichheit dieser Wörter würde auch die Gleichheit von $a_{i_1}a_{i_2}\dots a_{i_n}$ und $a_{j_1}a_{j_2}\dots a_{j_m}$ nach sich ziehen, womit ein Widerspruch zu (1.2) gegeben wäre) und

$$\varphi^*(a_{i_2}a_{i_3}\dots a_{i_n}) = \varphi^*(a_{j_2}a_{j_3}\dots a_{j_m})$$

(dies folgt aus (1.3) mittels Kürzen von $\varphi(a_{i_1}) = \varphi(a_{j_1})$). Die beiden letzten Relationen ergeben einen Widerspruch zur Minimalität von m. □

Beispiel 1.1 Die Mengen

$$C_1 = \{a, bb, aab, bab\},$$
$$C_2 = \{aa, bb, aba, baa\},$$
$$C_3 = \{aaa, aba, bab, bbb\},$$
$$C_4 = \{a, ab, bb\}$$

über $X = \{a, b\}$ sind Codes. Wir zeigen dies nur für C_1; die Beweise für C_2, C_3 und C_4 können analog geführt werden, jedoch kann mittels der weiter unten gegebenen Definitionen für C_2 und C_3 direkt ein Nachweis geführt werden. Wir zeigen, dass Bedingung (1.1) erfüllt und damit C_1 nach Satz 1.1 ein Code ist.

Es seien $x_{i_1}, x_{i_2}, \dots, x_{i_n}, x_{j_1}, x_{j_2}, \dots, x_{j_m}$ Elemente aus C_1 mit der Eigenschaft

$$x_{i_1}x_{i_2}\dots x_{i_n} = x_{j_1}x_{j_2}\dots x_{j_m}. \tag{1.4}$$

Das in (1.4) gegebene Wort sei w. Wir diskutieren nun die möglichen Fälle für x_{i_1}.
Fall 1. $x_{i_1} \in \{bb, bab\}$. Dann muss offenbar $x_{i_1} = x_{j_1}$ gelten, womit (1.1) nachgewiesen ist.
Fall 2. $x_{i_1} = aab$. Dann folgt $x_{i_1} = x_{j_1} = aab$ oder $x_{j_1} = a$. Im ersten Fall ist (1.1) erfüllt. Im zweiten Fall muss dann auch $x_{j_2} = a$ gelten und und für x_{j_3} gibt es zwei Möglichkeiten.
Fall 2.1. $x_{j_3} = bab$, d. h.

$$w = yx_{i_2}\dots x_{i_n} = yabx_{j_4}\dots x_{j_m} \text{ mit } y = aab.$$

Folglich ist $x_{i_2} = a$, und es ergibt sich

$$w = zx_{i_3}\dots x_{i_n} = zbx_{j_4}\dots x_{j_m} \text{ mit } z = aaba,$$

womit wir im wesentlichen die Situation aus Fall 2.2 erhalten.
Fall 2.2. $x_{j_3} = bb$, d. h.

$$w = yx_{i_2}\dots x_{i_n} = ybx_{j_4}\dots x_{j_m} \text{ mit } y = aab..$$

Folglich ist $x_{i_2} = bb$ oder $x_{i_2} = bab$, und es ergibt sich

$$w = z'bx_{i_3}\dots x_{i_n} = z'x_{j_4}\dots x_{j_m} \text{ mit } z' = aabb.$$

oder

$$w = z'abx_{i_3} \dots x_{i_n} = z'x_{j_4} \dots x_{j_m} \text{ mit } z' = aabb,$$

womit wir erneut im wesentlichen die Situation aus Fall 2.2 oder Fall 2.1 erhalten.

Die Situation der beiden Unterfälle wird also stets erneuert, womit gezeigt ist, dass die Gleichheit (1.4) im Widerspruch zur Annahme nicht gilt.
Fall 3. $x_{i_1} = a$. Wir können wie in Fall 2 zeigen, dass (1.1) erfüllt sein muss. $\Diamond$

Wir merken an, dass ein Code mit mindestens zwei Wörtern das Leerwort nicht enthalten kann, denn wegen

$$\lambda x = x\lambda$$

für jedes Wort x des Codes ist Bedingung (1.1) nicht erfüllt.

Wir geben nun eine leichte Verallgemeinerung von Satz 1.1.

Satz 1.2 *Eine Sprache C ist genau dann ein Code, wenn für beliebige Wörter*

$$x_{i_1}, x_{i_2}, \dots, x_{i_n}, x_{j_1}, x_{j_2}, \dots, x_{j_m} \in C,\ n \geq 1, m \geq 1,$$

mit

$$x_{i_1}x_{i_2} \dots x_{i_n} = x_{j_1}x_{j_2} \dots x_{j_m}$$

die Gleichheiten

$$n = m \quad \textit{und} \quad x_{i_t} = x_{j_t} \textit{ für } 1 \leq t \leq n$$

gelten.

Beweis. Es sei zuerst C ein Code, und es gelte $x_{i_1}x_{i_2} \dots x_{i_n} = x_{j_1}x_{j_2} \dots x_{j_m}$ für gewisse Wörter $x_{i_1}, x_{i_2}, \dots, x_{i_n}, x_{j_1}, x_{j_2}, \dots, x_{j_m} \in C$, $n \geq 1$, $m \geq 1$. Nach Satz 1.1 gilt $x_{i_1} = x_{j_1}$. Damit haben wir auch die Gleichheit

$$x_{i_2}x_{i_3} \dots x_{i_n} = x_{j_2}x_{j_3} \dots x_{j_m}.$$

Erneut erhalten wir aus Satz 1.1 $x_{i_2} = x_{j_2}$. So fortfahrend erhalten wir alle gewünschten Gleichheiten.

Gelten umgekehrt die Gleichheiten, so folgt, dass C ein Code ist wegen $x_{i_1} = x_{j_1}$ aus Satz 1.1. $\square$

Ausgehend von Satz 1.1 definieren wir eine spezielle Klasse von Codes.

Definition 1.2 *Ein Code C heißt strenger Code, wenn für beliebige $x_{i_k} \in C$ und $x_{j_k} \in C$, $k \geq 1$, für die für alle $n \geq 1$*

$$x_{i_1}x_{i_2} \dots x_{i_n} \textit{ ein Präfix von } x_{j_1}x_{j_2} \dots x_{j_n}$$

oder

$$x_{j_1}x_{j_2}\dots x_{j_n} \textit{ ein Präfix von } x_{i_1}x_{i_2}\dots x_{i_n}$$

ist, die Gleichheit $x_{i_1} = x_{j_1}$ *gilt.*

Offensichtlich gilt für strenge Codes die Satz 1.2 entsprechende Charakterisierung.

Bemerkung *Ein Code* C *ist ein strenger Code, wenn für beliebige* $x_{i_k} \in C$ *und* $x_{j_k} \in C$, $k \geq 1$, *für die für alle* $n \geq 1$

$$x_{i_1}x_{i_2}\dots x_{i_n} \textit{ ein Präfix von } x_{j_1}x_{j_2}\dots x_{j_n}$$

oder

$$x_{j_1}x_{j_2}\dots x_{j_n} \textit{ ein Präfix von } x_{i_1}x_{i_2}\dots x_{i_n}$$

ist, die Gleichheiten $x_{i_k} = x_{j_k}$ *für* $k \geq 1$ *gelten.*

Wir bemerken, dass die Forderung, dass

$$x_{i_1}x_{i_2}\dots x_{i_n} \text{ ein Präfix von } x_{j_1}x_{j_2}\dots x_{j_n}$$

oder

$$x_{j_1}x_{j_2}\dots x_{j_n} \text{ ein Präfix von } x_{i_1}x_{i_2}\dots x_{i_n}$$

ist, kürzer dadurch ausgedrückt werden kann, dass die Gleichheit

$$x_{i_1}x_{i_2}\dots x_{i_n}\dots = x_{j_1}x_{j_2}\dots x_{j_n}\dots$$

der „unendlichen Wörter“ $x_{i_1}x_{i_2}\dots$ und $x_{j_1}x_{j_2}\dots$ gilt.

Offensichtlich ist der Code C_3 aus Beispiel 1.1 ein strenger Code, denn da alle Wörter aus C_3 die Länge 3 haben und verschieden voneinander sind, müssen bei gleichem Präfix die ersten Wörter übereinstimmen.

Die Sprache C_4 ist dagegen kein strenger Code, denn mit $x_1 = a$, $x_2 = ab$ und $x_3 = bb$ gilt die Gleichheit

$$x_1x_3x_3x_3\dots = x_2x_3x_3x_3\dots = abbbbbbb\dots$$

Wir definieren nun Klassen von Codes.

Definition 1.3 *Eine nichtleere Sprache* C *heißt* Präfixcode, *wenn kein Wort aus* C *Präfix eines anderen Wortes aus* C *ist.*

Wir zeigen, dass ein Präfixcode C ein Code ist. Es seien $x_{i_1}, x_{i_2}, \dots, x_{i_n}, x_{j_1}, x_{j_2}, \dots, x_{j_m}$ beliebige Elemente aus C mit der Eigenschaft

$$x_{i_1}x_{i_2}\dots x_{i_n} = x_{j_1}x_{j_2}\dots x_{j_m}.$$

Ohne Beschränkung der Allgemeinheit sei $|x_{i_1}| \leq |x_{j_1}|$. Dann gilt $x_{i_1}v = x_{j_1}$ für ein Wort $v \in X^*$. Wegen der Eigenschaft von Präfixcodes müssen $v = \lambda$ und $x_{i_1} = x_{j_1}$ gelten, womit Bedingung (1.1) als gültig nachgewiesen ist. Nach Satz 1.1 ist deshalb gezeigt, dass C ein Code ist.

Die Sprache C_1 aus Beispiel 1.1 ist ein Code, aber kein Präfixcode. Die Sprachen C_2 und C_3 sind Präfixcodes.

Definition 1.4 *Es sei $n \geq 1$ eine natürliche Zahl. Eine Teilmenge C von X^n heißt Blockcode der Länge n über X.*

Offensichtlich sind Blockcodes Präfixcodes und folglich Codes.

Die Sprache C_2 ist ein Präfixcode, aber kein Blockcode. Die Sprache C_3 ist ein Blockcode.

Satz 1.3 *Für einen Code C und eine natürliche Zahl $k \geq 1$ ist auch C^k ein Code.*

Beweis. Wir betrachten beliebige Elemente $y_{i_1}, y_{i_2}, \ldots, y_{i_n}, y_{j_1}, y_{j_2}, \ldots, y_{j_m} \in C^k$ mit

$$y_{i_1} y_{i_2} \ldots y_{i_n} = y_{j_1} y_{j_2} \ldots y_{j_m}. \tag{1.5}$$

Da jedes Wort y_{i_t}, $1 \leq t \leq n$, und jedes Wort y_{j_s}, $1 \leq s \leq m$, ein Produkt von jeweils k Elementen aus C ist, kann (1.5) als eine Gleichheit zwischen Elementen aus C^* aufgefasst werden (die linke Seite ist ein Produkt aus $n \cdot k$ Faktoren aus C, die rechte Seite enthält $m \cdot k$ Faktoren). Nach Satz 1.2 gilt $nk = mk$, d. h. $n = m$ und die Faktoren aus C sind entsprechend ihrer Reihenfolge gleich. Insbesondere bedeutet dies die Gleichheit der Produkte aus den ersten k Faktoren der linken und rechten Seite. Das bedeutet, dass $y_{i_1} = y_{j_1}$ gilt. Nach Satz 1.1 ist damit bewiesen, dass C^k ein Code ist. □

1.2 Codierung und Decodierung durch Automaten

Wir wollen nun zeigen, dass für beliebige Codes die Codierung und für strenge Codes auch die Decodierung durch Automaten realisiert werden kann. Dafür benötigen wir Automaten, die ein Eingabe-Ausgabe-Verhalten beschreiben können.

Definition 1.5 *Ein Mealy-Automat ist ein 6-Tupel $\mathcal{A} = (X, Y, Z, f, g, z_0)$, wobei*
- *X, Y, Z Alphabete (endliche nichtleere Mengen) sind,*
- *$f : Z \times X \to Z$ und $g : Z \times X \to Y^*$ Funktionen sind und*
- *z_0 ein Element von Z ist.*

Dabei bezeichnen X, Y und Z das Eingabe- bzw. Ausgabealphabet bzw. die Menge der Zustände. Die Funktionen f und g werden Überführungs- bzw. Ausgabefunktion genannt. Befindet sich der Automat in einem Zustand z und liest ein Symbol x, so geht er in den Zustand $f(z, x)$ und gibt $g(z, x)$ aus. Der ausgewiesene Zustand z_0 heißt Anfangszustand.

Da wir bei Codierungen Folgen von Buchstaben, d. h. Wörter über dem Eingabealphabet, verarbeiten wollen, setzen wir die Funktionen f und g auf $Z \times X^*$ fort. Die entsprechenden Funktionen $f^* : Z \times X^* \to Z$ und $g^* : Z \times X^* \to Y^*$ definieren wir durch

$$f^*(z, \lambda) = z,\ g^*(z, \lambda) = \lambda,$$
$$f^*(z, wa) = f(f^*(z, w), a),\ g^*(z, wa) = g^*(z, w) g(f^*(z, w), a) \text{ für } w \in X^* \text{ und } a \in X.$$

Dabei gibt $f^*(z, w)$ den Zustand an, den der Automat vom Zustand z durch Abarbeitung des Eingabewortes $w \in X^*$ erreicht; $g^*(z, w)$ ist das dabei produzierte Ausgabewort.

Wir betrachten zwei Beispiele.

Beispiel 1.2 Es sei ein Automat $\mathcal{A} = (X, Y, Z, f, g, z_0)$ durch

$$X = \{A, B, C\},\ Y = \{a, b\} \text{ und } Z = \{z_0\},$$
$$f(z_0, A) = f(z_0, B) = f(z_0, C) = z_0,$$
$$g(z_0, A) = aa,\ g(z_0, B) = ab,\ g(z_0, C) = bb$$

gegeben. Dann erhalten wir $f(z_0, w) = z_0$ für jedes Wort $w \in X^*$ und beispielsweise $g^*(z_0, CA) = bb\,aa$ und $g^*(z_0, ABBA) = aa\,ab\,ab\,aa$. ◊

Beispiel 1.3 Wir wollen nun einen Mealy-Automaten angegeben, der (zumindest partiell) einen Automaten beschreibt, an dem Scheine zum Parken gezogen werden können. Die möglichen Eingaben für einen solchen Automaten seien 50-Cent-, 1-Euro- und 2-Euro-Münzen. Die möglichen Parkzeiten sind eine halbe Stunde, eine Stunde, anderthalb Stunden oder zwei Stunden, wobei für jeweils eine halbe Stunde 50 Cent zu zahlen sind. Längeres Parken ist nicht erlaubt. Der Parkschein wird aber nur ausgegeben, wenn durch Knopfdruck ein Parkschein angefordert wurde. Hieraus resultiert, dass wir als Eingabe- und Ausgabemenge

$$X = \{50, 1, 2, A\} \quad \text{und} \quad Y = \{30, 60, 90, 120\}$$

(A für Anfordern; Parkzeit in Minuten) verwenden können. Damit der Automat die richtige Parkdauer ausgibt, muss er sich den in den Automaten gezahlten Betrag merken. Folglich verwenden wir die Zustandsmenge

$$Z = \{0, 50, 100, 150, 200\}$$

(eingezahlter Betrag in Cent). Zwar dürfen größere Beträge in den Automaten gesteckt werden, aber die Parkdauer wird dadurch nicht erhöht, stimmt also mit der von 200 Cent überein. Es ergeben sich die folgende Überführungs- und Ausgabefunktion, die wir durch eine Tabelle angeben, bei der die Zeilen den Eingabesymbolen und die Spalten den Zuständen entsprechen und im Schnittpunkt der Zeile zu x und der Spalte zu z das Paar $(f(z, x), g(z, x))$ angegeben ist:

	0	50	100	150	200
50	$(50, \lambda)$	$(100, \lambda)$	$(150, \lambda)$	$(200, \lambda)$	$(200, \lambda)$
1	$(100, \lambda)$	$(150, \lambda)$	$(200, \lambda)$	$(200, \lambda)$	$(200, \lambda)$
2	$(200, \lambda)$	$(200, \lambda)$	$(200, \lambda)$	$(200, \lambda)$	$(200, \lambda)$
A	$(0, \lambda)$	$(0, 30)$	$(0, 60)$	$(0, 90)$	$(0, 120)$

Als Anfangszustand verwenden wir 0, da zu Beginn kein Betrag gezahlt worden ist.

Für die sinnvollen Eingabefolgen, d. h. es werden nur Beträge von 50, 100, 150 oder 200 Cent gezahlt und am Ende von jedem Nutzer die Anforderung A ausgelöst, ergibt sich unter Verwendung von Wörtern w_{x_i} (der i-te Nutzer hat den Betrag x_i durch seine Eingabefolge gezahlt) und $y_i = \frac{3}{5}x_i$ die Werte

$$f^*(0, w_{x_1} A w_{x_2} A \ldots w_{x_k} A) = 0 \text{ und } g^*(0, w_{x_1} A w_{x_2} A \ldots w_{x_k} A) = y_1 y_2 \ldots y_k$$

für den erreichten Zustand und die gesamte Ausgabe. ◊

Es seien zwei Alphabete $A = \{a_1, a_2, \ldots, a_n\}$ und X gegeben, außerdem eine Sprache $C = \{c_1, c_2, \ldots, c_n\} \subseteq X^*$ sowie eine Codierung $\varphi : A \to C$ mit $\varphi(a_i) = c_i$, $1 \leq i \leq n$. Dann gilt für die homomorphe Erweiterung φ^* offenbar

$$\varphi^*(a_{i_1} a_{i_2} \ldots a_{i_m}) = c_{i_1} c_{i_2} \ldots c_{i_m}.$$

Wenn wir diese Abbildung durch einen Automaten erzeugen wollen, so benötigen wir einen Mealy-Automaten, der auf die Eingabe a_i, $1 \leq i \leq n$, die Ausgabe c_i erzeugt. Eine Abhängigkeit von einem Zustand ist hier nicht erforderlich. Formal ergibt sich der Automat

$$\mathcal{A}_{\text{cod}} = (\{a_1, a_2, \ldots, a_n\}, X, \{z\}, f, g, z)$$

mit

$$f(z, a_i) = z \text{ und } g(z, a_i) = c_i \quad \text{für} \quad 1 \leq i \leq n.$$

Offensichtlich ergibt sich dann

$$g^*(z, a_{i_1} a_{i_2} \ldots a_{i_m}) = c_{i_1} c_{i_2} \ldots c_{i_m} = \varphi^*(a_{i_1} a_{i_2} \ldots a_{i_m}).$$

Der Automat $\mathcal{A}_{\text{cod}}$ realisiert also gerade die durch φ gegebene Codierung.

Der in Beispiel 1.2 angegebene Automat ist der codierende Automat für die Codierung $\varphi : \{A, B, C\} \to \{aa, ab, bb\}$.

Wir betrachten nun das umgekehrte Problem. Wir wollen eine Folge $c_{i_1} c_{i_2} \ldots c_{i_m}$, die durch die Codierung φ entstanden ist, zurückübersetzen in die eingegebene Folge $a_{i_1} a_{i_2} \ldots a_{i_m}$. Dieser Vorgang wird Decodierung genannt.

Wir wollen auch hier einen Automaten $\mathcal{A}_{\text{dec}} = (X, Y, Z, f, g, z_0)$ konstruieren, für den

$$g^*(z_0, c_{i_1} c_{i_2} \ldots c_{i_m}) = a_{i_1} a_{i_2} \ldots a_{i_m} = (\varphi^*)^{-1}(c_{i_1} c_{i_2} \ldots c_{i_m})$$

gilt. Wir geben die Konstruktion für Präfixcodes an und verwenden die Charakterisierung von Codes entsprechend Satz 1.1. Intuitiv bedeutet dies: Wir verarbeiten das Eingabewort ohne Ausgabe (genauer gesagt, mit Ausgabe des Leerwortes λ), bis wir ein Codewort c gelesen haben. Dann geben wir $\varphi^{-1}(c)$ aus. Das verbleibende Eingabewort wird nun genauso verarbeitet. Das Testen, ob ein Codewort vorliegt, wird dadurch realisiert, dass wir uns den schon gelesenen Teil merken und mit den Codewörtern vergleichen. Formal ergibt sich der endliche Mealy-Automat

$$\mathcal{A}_{\text{dec}} = (X, Y, Z, f, g, z_0)$$

mit der Menge X von Eingabesymbolen, dem Ausgabealphabet

$$Y = A \cup \{\text{Fehler}\}$$

(neben den Elementen aus A, die als Ergebnis der Decodierung auftreten, enthält Y noch eine Fehlermeldung), der Zustandsmenge

$$Z = \{[w] \mid w \in \mathit{Präf}(C)\} \cup \{z_{\text{Fehler}}\}$$

(wobei $Präf(U) = \{w \mid \text{es existiert ein Wort } x \text{ mit } wx \in U\}$ die Menge der Präfixe von Wörtern aus U ist), dem Anfangszustand

$$z_0 = [\lambda]$$

(zu Beginn hat der Automat noch nichts gelesen), der Zustandsüberführungsfunktion $f : Z \times X \to Z$ und der Ausgabefunktion $g : Z \times X \to Y$, die durch

$$f(z,x) = \begin{cases} [wx], & \text{falls } z = [w] \text{ und } wx \text{ echtes Präfix eines Wortes aus } C \text{ ist,} \\ [\lambda], & \text{falls } z = [w] \text{ und } wx \in C, \\ z_{\text{Fehler}}, & \text{sonst} \end{cases}$$

und

$$g(z,x) = \begin{cases} \lambda, & \text{falls } z = [w] \text{ und } wx \text{ echtes Präfix eines Wortes aus } C \text{ ist,} \\ a_i, & \text{falls } z = [w] \text{ und } wx = c_i \in C, \\ \lambda, & \text{falls } z = z_{\text{Fehler}}, \\ \text{Fehler}, & \text{sonst} \end{cases}$$

gegeben sind. Der Automat $\mathcal{A}$ liest die Eingabe und merkt sich das bereits gelesene Wort w als Zustand $[w]$ und gibt nichts (d. h. das Leerwort) aus, solange dies der Anfang eines Codewortes ist. Ist ein Codewort c_i vollständig gelesen worden, so vergisst $\mathcal{A}$ den bereits gelesenen Teil und gibt das Symbol a_i aus, das durch c_i codiert wird. Ist der gelesene Teil kein Anfang eines Codewortes, so geht $\mathcal{A}$ in einen speziellen Fehlerzustand z_{Fehler} und gibt eine Fehlermeldung aus. Liegt bereits der Fehlerzustand vor, so bleibt $\mathcal{A}$ in diesem Zustand und gibt nichts mehr aus. Daher überführt der Automat $\mathcal{A}$ ein Eingabewort $c_{i_1}c_{i_2}\ldots c_{i_m}$ mit $c_{i_j} \in C$, $1 \le j \le m$, in das Ausgabewort $a_{i_1}a_{i_2}\ldots a_{i_m}$, womit eine korrekte Decodierung vorgenommen wird. Ist dagegen das Eingabewort kein Produkt von Codewörtern, wird ein Wort der Form yFehler ausgegeben, womit ausgesagt wird, dass die Eingabe nicht decodierbar ist.

Offenbar liefert der oben konstruierte Automat $\mathcal{A}_{\text{dec}}$ nur für Präfixcodes ein richtiges Ergebnis. Dies ist wie folgt zu sehen: Enthält der Code zwei Wörter x und xy für ein gewisses $y \in X^*$, so weiß der Automat nach Lesen von x nicht, ob er das Codewort x oder nur den Anfang von xy gelesen hat.

Wir bemerken, dass sich der Gedanke aber auch für strenge Codes nutzen lässt, indem man die Entscheidung, ob der Automat ein Codewort oder ein echtes Präfix eines Codewortes gelesen hat, erst etwas später trifft. Für die entsprechende Konstruktion des Automaten verweisen wir auf [9].

Diese Ausführung lassen sich zu folgendem Satz zusammenfassen.

Satz 1.4 *Es seien X ein Alphabet und $C \subseteq X^+$ ein strenger Code. Dann gibt es einen Algorithmus, der zu jeder Codierung $\varphi : A \to C$ und jedem Wort $x \in X^+$ in linearer Zeit $(\varphi^*)^{-1}(x)$ berechnet oder feststellt, dass $(\varphi^*)^{-1}(x)$ nicht definiert ist.* □

1.3 Entscheidbarkeit der Eigenschaft, Code zu sein

Wir geben zuerst eine weitere Charakterisierung von Codes an.

Definition 1.6 *Eine Sprache L heißt produktunabhängig, falls kein Wort in L als Produkt von mindestens zwei Wörtern aus L dargestellt werden kann.*

Bei einer produktunabhängigen Sprache L gilt nach Definition 1.6 für jedes Produkt $w = x_1x_2 \dots x_n$ mit $n \geq 2$ und $x_i \in L$ für $1 \leq i \leq n$ die Relation $w \notin L$.

Offenbar ist jeder Code wegen Bedingung (1.1) und Satz 1.1 produktunabhängig. Andererseits ist die zu Anfang des Abschnitts betrachtete Menge $\{a, ab, ba\}$ offensichtlich produktunabhängig, aber kein Code.

Satz 1.5 *Es sei C eine produktunabhängige Menge über einem Alphabet X. Dann ist C genau dann ein Code, wenn für jedes Wort $w \in X^*$ die folgende Bedingung gilt:*

$$wC^* \cap C^* \neq \emptyset \text{ und } C^*w \cap C^* \neq \emptyset \quad \text{implizieren} \quad w \in C^*. \tag{1.6}$$

Beweis. Wir nehmen zuerst an, dass die Bedingung (1.6) für jedes Wort $w \in X^*$ gilt, und zeigen, dass C ein Code ist.

Angenommen, C wäre kein Code. Dann gibt es wegen Satz 1.1 Wörter $x_{i_1}, x_{i_2}, \dots, x_{i_n}$, $x_{j_1}, x_{j_2}, \dots, x_{j_m} \in C$ mit

$$x_{i_1}x_{i_2} \dots x_{i_n} = x_{j_1}x_{j_2} \dots x_{j_m} \quad \text{und} \quad x_{i_1} \neq x_{j_1}.$$

Dann ist x_{i_1} ein echtes Präfix von x_{j_1} oder umgekehrt. Ohne Beschränkung der Allgemeinheit gelte $|x_{i_1}| < |x_{j_1}|$. Dann folgt die Existenz eines Wortes $y \in X^+$ mit

$$x_{j_1} = x_{i_1}y \quad \text{und} \quad x_{i_2}x_{i_3} \dots x_{i_n} = yx_{j_2}, x_{j_3} \dots x_{j_m}.$$

Damit gelten

$$x_{j_1} \in C^* \cap C^*y \quad \text{und} \quad x_{i_2}x_{i_3} \dots x_{i_n} \in C^* \cap yC^*.$$

Somit erhalten wir $y \in C^*$. Damit erhalten wir einen Widerspruch zur Produktunabhängigkeit von C, da x_{j_1} wegen $x_{j_1} = x_{i_1}y$ eine Produktdarstellung aus zwei Faktoren aus C besitzt.

Es sei nun C ein Code. Wir zeigen, dass die Aussage (1.6) für jedes Wort $w \in X^*$ folgt.

Angenommen, dies wäre nicht der Fall. Dann muss es Elemente

$$x_{i_1}, x_{i_2}, \dots, x_{i_n}, x_{j_1}, x_{j_2}, \dots, x_{j_m}, y_{k_1}, y_{k_2}, \dots, y_{k_t}, y_{l_1}, y_{l_2}, \dots, y_{l_s} \in C$$

für $n \geq 0$, $m \geq 0$, $t \geq 0$, $s \geq 0$ und

$$w \notin C^*$$

so geben, dass die Beziehungen

$$x_{i_1}x_{i_2} \dots x_{i_n}w = x_{j_1}x_{j_2} \dots x_{j_m} \in C^*w \cap C^* \text{ und } wy_{k_1}y_{k_2} \dots y_{k_t} = y_{l_1}y_{l_2} \dots y_{l_s} \in wC^* \cap C^*$$

gelten. Es seien die Elemente so gewählt, dass m minimal ausfällt. Dann gilt $x_{i_1} \neq x_{j_1}$. Damit erhalten wir

$$\begin{aligned} x_{j_1}x_{j_2}\dots x_{j_m}y_{k_1}y_{k_2}\dots y_{k_t} &= x_{i_1}x_{i_2}\dots x_{i_n}wy_{k_1}y_{k_2}\dots y_{k_t} \\ &= x_{i_1}x_{i_2}\dots x_{i_n}y_{l_1}y_{l_2}\dots y_{l_s}. \end{aligned}$$

Wegen $x_{i_1} \neq x_{j_1}$ ist somit die Bedingung (1.1) verletzt, woraus wegen Satz 1.1 ein Widerspruch dazu resultiert, dass C ein Code ist. □

Die Bedingung (1.6) aus Satz 1.5 kann noch verschärft werden:

$$C^*w \cap wC^* \cap C^* \neq \emptyset \quad \text{impliziert} \quad w \in C^* \tag{1.7}$$

gilt für jedes $w \in X^*$. Dies kann wie folgt eingesehen werden.

(1.6) $\Rightarrow$ (1.7) . Es sei $C^*w \cap wC^* \cap C^* \neq \emptyset$. Dann sind auch $wC^* \cap C^*$ und $C^*w \cap C^*$ nichtleere Mengen. Damit gilt $w \in C^*$ wegen (1.6), womit (1.7) bewiesen ist.

(1.7) $\Rightarrow$ (1.6). Es gelte

$$wx_1 = x_2 \text{ und } x_3w = x_4 \quad \text{für gewisse } x_1, x_2, x_3, x_4 \in C^*.$$

Dann gilt auch

$$wx_1x_4 = x_2x_4 = x_2x_3w,$$

womit gezeigt ist, dass es ein Element in $wC^* \cap C^* \cap C^*w$ gibt. Somit erhalten wir $w \in C^*$ wegen (1.7). Daher ist (1.6) bewiesen.

Wir merken noch an, dass (1.6) in der Gruppentheorie ein Kriterium für Untergruppen darstellt, d. h. für eine Gruppe G und eine nichtleere Teilmenge $H \subseteq G$ gilt, dass H genau dann eine Untergruppe von G ist, wenn

$$fH \cap H \neq \emptyset \text{ und } Hf \cap H \neq \emptyset \quad \text{implizieren} \quad f \in H \tag{1.8}$$

für beliebige $f \in G$ gilt. Somit können Codes innerhalb der freien Halbgruppe X^* als Gegenstücke zu Untergruppen betrachtet werden.

Wir beweisen nun das Untergruppenkriterium.

Es sei H eine Untergruppe von G. Ferner erfülle $f \in G$ die Beziehung $Hf \cap H \neq \emptyset$. Folglich gilt $h_1f = h_2$ für gewisse $h_1, h_2 \in H$. Somit ergibt sich $f = h_1^{-1}h_2 \in H$, und (1.8) ist bewiesen.

Gilt umgekehrt für eine nichtleere Menge $H \subseteq G$ und alle $f \in G$ die Aussage (1.8), so ergeben sich

- $e \in H$ für das neutrale Element e von G wegen $He \cap H = H \cap H = H \neq \emptyset$ und $eH \cap H \neq \emptyset$,
- $h^{-1} \in H$ für $h \in H$ wegen $h^{-1}h = e \in h^{-1}H \cap H \neq \emptyset$ und $Hh^{-1} \cap H \neq \emptyset$,
- $h_1h_2 \in H$ für $h_1, h_2 \in H$ auf Grund von $h_1^{-1}h_1h_2 = h_2 \in Hh_1h_2 \cap H \neq \emptyset$ und $h_1h_2h_2^{-1} = h_1 \in h_1h_2H \cap H \neq \emptyset$.

Damit sind die Bedingungen des klassischen Kriteriums für Untergruppen erfüllt und H ist als Untergruppe nachgewiesen.

Die bisherigen Charakterisierungen von Codes sind leider nicht effektiv in dem Sinn, dass aus ihnen ein Algorithmus gewonnen werden kann, der entscheidet ob die gegebene Menge ein Code ist. Wir geben nun zwei Kriterien an, die effektiv sind.

Satz 1.6 *Die Menge $C = \{x, y\}$ bestehe aus zwei nichtleeren Wörtern x und y über X. Dann ist C genau dann ein Code, wenn $xy \neq yx$ gilt.*

Beweis. Für einen Code mit den Elementen x und y muss wegen Satz 1.1 die Ungleichheit $xy \neq yx$ gelten.

Um die umgekehrte Implikation zu beweisen, betrachten wir die Menge

$$A = \{\{x, y\} \mid xy \neq yx,\ \{x, y\} \text{ ist kein Code}\}.$$

Wir haben zu zeigen, dass A leer ist (und werden dies indirekt beweisen).

Dazu nehmen wir an, dass $A \neq \emptyset$ gilt. Wir wählen $\{r, s\} \in A$ so, dass $|rs|$ minimal ausfällt, d. h.

$$|rs| = \min\{|xy| \mid \{x, y\} \in A\}.$$

Da $\{r, s\}$ kein Code ist, muss es wegen Satz 1.1 ein Wort w geben, dass auf zwei verschiedene Arten als Produkt über $\{r, s\}$ dargestellt werden kann. Wir wählen w minimal unter allen Wörtern mit zwei Darstellungen als Produkt. Dann gilt

$$rxr = sx's \quad \text{für gewisse} \quad x, x' \in \{r, s\}^* \tag{1.9}$$

oder

$$rys = sy'r \quad \text{für gewisse} \quad y, y' \in \{r, s\}^*. \tag{1.10}$$

Da bei $|r| = |s|$ die Menge $\{r, s\}$ ein Blockcode wäre, haben r und s verschiedene Längen. Ohne Beschränkung der Allgemeinheit nehmen wir $|s| > |r|$ an. Sowohl aus (1.9) als auch (1.10) damit folgt

$$s = ur \quad \text{für ein gewisses} \quad u \in X^+.$$

Wegen $\lambda s = s\lambda$ folgt aus der Wahl von r und s, dass r nicht leer ist. Folglich gilt

$$|ur| = |s| < |rs|. \tag{1.11}$$

Falls $ur = ru$ gilt, so erhalten wir $sr = urr = rur = rs$ im Widerspruch zur Wahl von r und s. Folglich gilt

$$ur \neq ru. \tag{1.12}$$

Wegen (1.12), (1.11) und der Minimalität von $|rs|$, muss $\{r, u\}$ ein Code sein.

Andererseits erhalten wir durch Einsetzen von ur für s in (1.9) bzw. (1.10)

$$rxr = urx'ur \quad \text{bzw.} \quad ryur = ury'r$$

mit $x, x', y, y' \in \{r, u\}^*$, woraus wegen Satz 1.1 resultiert, dass $\{r, u\}$ kein Code sein kann. Damit haben wir den gewünschten Widerspruch. □

Die Bedingung aus Satz 1.6 ist sehr einfach zu testen, gilt aber nur für sehr spezielle Codes. Wir wollen nun ein Kriterium angeben, das für beliebige Codes gilt und für endliche Mengen effektiv ist.

Dazu definieren wir für eine nichtleere Sprache C über X induktiv die folgenden Mengen:

$$K_0(C) = C,$$
$$K_{i+1}(C) = \{w \in X^+ \mid yw = x \text{ oder } xw = y \text{ für gewisse } x \in C, y \in K_i(C)\}.$$

Nach Definition ergibt sich damit $K_{i+1}(C)$ als die Menge aller Suffixe von Wörtern aus C bzw. $K_i(C)$, deren Präfixe in $K_i(C)$ bzw. C liegen.

Zur Illustration der Konstruktion geben wir das folgende Beispiel.

Beispiel 1.4 Zuerst betrachten wir die Menge $C_0 = \{a, ab, ba\}$ (siehe Abschnitt 1.1). Es ergeben sich die folgenden Mengen:

- $K_0(C_0) = C_0 = \{a, ab, ba\}$ nach Definition.
- $K_1(C_0) = \{b\}$, da nur das Produkt aus $a \in C_0 = K_0(C_0)$ mit b ein Element aus $C_0 = K_0(C_0)$ ergibt.
- $K_2(C_0) = \{a\}$, denn $b \in K_1(C_0)$ ist nicht als Produkt darstellbar und nur $ba \in C_0$ ergibt sich als Produkt von $b \in K_1(C_0)$ und a.
- $K_3(C_0) = \{b\}$ ergibt sich analog zu $K_1(C_0)$.

Daher erhalten wir

$$K_i(C_0) = \begin{cases} \{a, ab, ba\} & \text{für } i = 0, \\ \{a\} & \text{für gerade Zahlen } i \geq 1, \\ \{b\} & \text{für ungerade Zahlen } i \geq 1. \end{cases}$$

Für den Code $C_1 = \{a, bb, aab, bab\}$ aus Beispiel 1.1 ergeben sich die Mengen

- $K_1(C_1) = \{ab\}$, da aab das einzige Wort aus der Menge $C_1 = K_0(C)$ mit einem Präfix aus $C_1 = K_0(C_1)$ ist, wobei das Suffix ab ist,
- $K_2(C_1) = \{b\}$, denn $ab \in K_1(C_1)$ hat nur das Präfix $a \in C_1$, wobei b Suffix ist, und kein Element aus C_1 hat das Präfix ab, das einzige Wort aus $K_1(C_1)$,
- $K_3(C_1) = \{b, ab\}$, da die einzigen zulässigen Zerlegungen von Elementen aus C_1 und $K_2(C_1)$ durch $b \cdot b$ und $b \cdot ab$ gegeben sind,
- $K_4(C_1) = \{b, ab\}$ in Analogie zu $K_3(C_1)$.

Somit erhalten wir

$$K_i(C) = \begin{cases} \{a, bb, aab, bab\} & \text{für } i = 0, \\ \{ab\} & \text{für } i = 1, \\ \{b\} & \text{für } i = 2, \\ \{b, ab\} & \text{für } i \geq 3. \end{cases}$$

◊

Wir beweisen zuerst zwei Lemmata, die im Wesentlichen einen Schritt (von i nach $i+1$) auf den Übergang von i nach $j > i$ vollziehen. Dazu definieren wir noch für eine Menge U von Wörtern über X die Menge

$$\mathit{Suff}(U) = \{w \mid \text{es existiert ein Wort } x \in X^* \text{ mit } xw \in U\}$$

aller Suffixe von Wörtern aus U.

Lemma 1.7 *Für jeden Code C, jedes $n \geq 0$ und jedes $w \in K_n(C)$ gilt $w \in \mathit{Suff}(C)$.*

Beweis. Wir beweisen die Aussage durch vollständige Induktion über n.

$n = 0$: Nach Definition gilt $w \in C$. Wegen $C \subseteq \mathit{Suff}(C)$ ist der Induktionsanfang gezeigt.

$n \to n+1$: Es sei $w \in K_{n+1}(C)$. Gelten $yw = x$, $y \in K_n(C)$ und $x \in C$, so folgt sofort $w \in \mathit{Suff}(x) \subseteq \mathit{Suff}(C)$. Gelten dagegen $xw = y$, $y \in K_n(C)$ und $x \in C$, so folgt mittels Induktionsvoraussetzung $w \in \mathit{Suff}(y) \subseteq \mathit{Suff}(\mathit{Suff}(C)) = \mathit{Suff}(C)$. □

Lemma 1.8 *Ein Wort v_n liegt genau dann in $K_n(C)$, $n \geq 1$, wenn es für jede Zahl $i < n$ Wörter $v_i \in K_i(C)$ und $x_{i_1}, x_{i_2}, \ldots, x_{i_k}, x_{j_1}, x_{j_2}, \ldots, x_{j_l} \in C$ mit $k + l = n - i$ derart gibt, dass entweder*

$$v_i x_{i_1} x_{i_2} \ldots x_{i_k} v_n = x_{j_1} x_{j_2} \ldots x_{j_l} \quad \mathit{mit} \quad |v_n| < |x_{j_l}|$$

oder

$$v_i x_{i_1} x_{i_2} \ldots x_{i_k} = x_{j_1} x_{j_2} \ldots x_{j_l} v_n \quad \mathit{mit} \quad |v_n| < |x_{i_k}| \text{ für } k \neq 0$$

gilt.

Beweis. Für $n - i = 1$ also $i = n - 1$ entsprechen die Bedingungen gerade der Definition von Wörtern in $K_n(C)$.

Es sei $n - i = 2$. Zu einem Wort $v_n \in K_n(C)$ gibt es nach Definition Wörter $v_{n-1} \in K_{n-1}(C)$ und $x' \in C$ derart, dass entweder

$$v_{n-1} v_n = x' \tag{1.13}$$

oder

$$x' v_n = v_{n-1} \tag{1.14}$$

gilt. Weiterhin gibt es wegen $v_{n-1} \in K_{n-1}(C)$ Wörter $v_{n-2} \in K_{n-2}(C)$ und $x \in C$ derart, dass entweder

$$v_{n-2} v_{n-1} = x \tag{1.15}$$

oder

$$x v_{n-1} = v_{n-2} \tag{1.16}$$

gültig ist. Wir betrachten nun die vier Kombinationsmöglichkeiten:

– *Fall 1:* Es gelten die Gleichungen (1.13) und (1.15). Wir erhalten einerseits $v_{n-2}v_{n-1}v_n = v_{n-2}x'$ und andererseits $v_{n-2}v_{n-1}v_n = xv_n$, woraus mit $v_{n-2}x' = xv_n$ eine Gleichheit der gewünschten Art resultiert.
– *Fall 2:* Es gelten die Gleichungen (1.13) und (1.16). Dann gilt einerseits $xv_{n-1}v_n = v_{n-2}v_n$ und andereseits $xv_{n-1}v_n = xx'$. Hieraus ergibt sich die Gleichheit $v_{n-2}v_n = xx'$ der gewünschten Art.
– *Fall 3:* Es gelten die Gleichungen (1.14) und (1.15). Wir erhalten $v_{n-2}x'v_n = v_{n-2}v_{n-1} = x$. Die letzte Gleichheit ist von der geforderten Art.
– *Fall 4:* Es gelten die Gleichungen (1.14) und (1.16). Wir erhalten $v_{n-2} = xv_{n-1} = xx'v_n$, woraus die Gleichheit $v_{n-2} = xx'v_n$ der geforderten Art resultiert.

Damit haben wir gezeigt, dass für $v_n \in K_n(C)$ und $n - i = 2$ Wörter der gewünschten Art existieren.

Für die umgekehrte Richtung sei zuerst $v_{n-2}x'v_n = x$ für $v_{n-2} \in K_{n-2}(C)$ und $x, x' \in C$ gültig. Dann setzen wir $v_{n-1} = x'v_n$. Wegen $v_{n-2}v_{n-1} = x$ ist dann $v_{n-1} \in K_{n-1}(C)$ und wegen $v_{n-1} = x'v_n$ folglich $v_n \in K_n(C)$. Dies entspricht dem obigen Fall 3. In analoger Weise behandeln wir die anderen drei möglichen Fälle $v_{n-2}v_n = xx'$, $v_{n-2}x' = xv_n$ und $v_{n-2} = xx'v_n$.

Damit ist die Aussage für $n - i = 2$ beweisen.

Mittels vollständiger Induktion lässt sich die Aussage nun für alle natürlichen Zahlen n und $i < n$ beweisen. □

Wir verwenden nun die Mengen $K_n(C)$ für eine weitere Charakterisierung von (strengen) Codes.

Satz 1.9 *Eine nichtleere Sprache C über X ist genau dann ein Code, wenn $K_i(C) \cap C = \emptyset$ für $i \geq 1$ gilt.*

Beweis. Zuerst zeigen wir (indirekt), dass aus der Gültigkeit von $K_i(C) \cap C = \emptyset$ für alle $i \geq 1$ folgt, dass C ein Code ist.

Nehmen wir dazu an, dass C kein Code wäre. Dann gibt es nach Satz 1.1 Elemente $x_{i_1}, x_{i_2}, \ldots, x_{i_n}, x_{j_1}, x_{j_2}, \ldots, x_{j_m} \in C$ mit

$$x_{i_1}x_{i_2}\ldots x_{i_n} = x_{j_1}x_{j_2}\ldots x_{j_m}$$

und $x_{i_1} \neq x_{j_1}$. Wenn wir $n + m$ minimal wählen, so erhalten wir, dass

$$x_{i_1}x_{i_2}\ldots x_{i_t} \neq x_{j_1}x_{j_2}\ldots x_{j_s}$$

und sogar noch schärfer

$$|x_{i_1}x_{i_2}\ldots x_{i_t}| \neq |x_{j_1}x_{j_2}\ldots x_{j_s}|$$

für beliebige $1 \leq t \leq n$ und $1 \leq s \leq m$ gelten.

Für $1 \leq t \leq n$ sei t' die minimale natürliche Zahl mit

$$|x_{i_1}x_{i_2}\ldots x_{i_t}| < |x_{j_1}x_{j_2}\ldots x_{j_{t'}}|.$$

Offenbar gilt dann

$$x_{j_1}x_{j_2}\ldots x_{j_{t'}} = x_{i_1}x_{i_2}\ldots x_{i_t}v_t$$

für ein gewisses Wort v_t. In völlig analoger Weise definieren wir für $1 \leq s \leq m$ die minimale Zahl s' und z_s mit

$$x_{i_1}x_{i_2}\dots x_{i_{s'}} = x_{j_1}x_{j_2}\dots x_{j_s}z_s.$$

Wir zeigen nun mittels Induktion über die Länge von $x_{i_1}x_{i_2}\dots x_{i_t}$ und $x_{j_1}x_{j_2}\dots x_{j_s}$, dass die Wörter v_t bzw. z_s in einer der Mengen $K_i(C)$ mit $i \geq 1$ liegen.

Ohne Beschränkung der Allgemeinheit nehmen wir an, dass $|x_{i_1}| < |x_{j_1}|$ gilt. Dann erhalten wir $x_{i_1}v_1 = x_{j_1}$. Wegen $x_{i_1}, x_{j_1} \in C = K_0(C)$, ergibt sich, dass v_1 in $K_1(C)$ liegt, womit der Induktionsanfang bewiesen ist.

Wir betrachten nun v_t. Wir unterscheiden zwei Fälle:

Fall 1. $|x_{i_1}x_{i_2}\dots x_{i_{t-1}}| < |x_{j_1}x_{j_2}\dots x_{j_{t'-1}}|$. Aufgrund der Minimalität von t' gilt

$$|x_{j_1}x_{j_2}\dots x_{j_{t'-1}}| < |x_{i_1}x_{i_2}\dots x_{i_t}|\,.$$

Damit ist $x_{i_1}x_{i_2}\dots x_{i_t}$ das Wort minimaler Länge, das länger als $x_{j_1}x_{j_2}\dots x_{j_{t'-1}}$ ist. Folglich erhalten wir

$$x_{i_1}x_{i_2}\dots x_{i_t}v_t = x_{j_1}x_{j_2}\dots x_{j_{t'}} = x_{j_1}x_{j_2}\dots x_{j_{t'-1}}z_{t'-1}v_t.$$

Damit erfüllt $z_{t'-1}$ folgende Gleichungen:

$$x_{j_1}x_{j_2}\dots x_{j_{t'-1}}z_{t'-1} = x_{i_1}x_{i_2}\dots x_{i_t} \tag{1.17}$$

und

$$z_{t'-1}v_t = x_{j_{t'}}. \tag{1.18}$$

Nach (1.17 und Lemma 1.8 gilt $z_{t'-1} \in K_i(C)$ für ein gewisses $i \geq 1$. Damit ergibt sich aus (1.18) dann $v_t \in K_{i+1}(C)$, womit die Induktionsbehauptung gezeigt ist.

Fall 2. $|x_{i_1}x_{i_2}\dots x_{i_{t-1}}| > |x_{j_1}x_{j_2}\dots x_{j_{t'-1}}|$. Dann gilt $(t-1)' = t'$ und folglich

$$x_{i_{t-1}}v_t = v_{t-1}.$$

Da nach Induktionsbehauptung $v_{t-1} \in K_i(C)$ für ein gewisses $i \geq 1$ ist und $x_{i_{t-1}} \in C$ gilt, erhalten wir $v_t \in K_{i+1}(C)$, womit auch in diesem Fall die Induktionsbehauptung bewiesen ist.

Analog beweist man, dass jedes z_s in einer der Mengen aus $K(C)$ liegt.

Ohne Beschränkung der Allgemeinheit sei $|x_{i_n}| < |x_{j_m}|$. Dann folgt $(n-1)' = m$ und $v_{n-1} = x_{j_m}$. Wegen $v_{n-1} \in K_j(C)$ für ein gewisses $j \geq 1$ und $x_{j_m} \in C$ erhalten wir, dass es ein $j \geq 1$ gibt, für das $K_j(C) \cap C$ nicht leer ist. Dies widerspricht unserer Voraussetzung.

Es sei nun C ein Code. Wir zeigen zuerst, dass

$$C^*w \cap C^* \neq \emptyset \quad \text{für} \quad w \in K_i(C),\ i \geq 0 \tag{1.19}$$

gilt.

Für $i = 0$ folgt dies direkt aus der Definition von $K_0(C) = C$.

Es sei nun $w \in K_i(C)$. Dann gibt es $x \in C$ und $y \in K_{i-1}(C)$ derart, dass

$$xw = y \quad \text{oder} \quad yw = x$$

gilt. Ferner ist nach Induktionsannahme

$$x_1 y = x_2$$

für gewisse $x_1, x_2 \in C^*$ gültig. Damit erhalten wir

$$x_1 xw = x_1 y = x_2 \quad \text{oder} \quad x_1 x = x_1 yw = x_2 w$$

und damit in beiden Fällen (1.19).

Wir nehmen nun an, dass $K_i(C) \cap C$ für ein $i \geq 1$ nicht leer ist. Es sei $x \in K_i(C) \cap C$. Dann gibt es nach Definition von $K_i(C)$ Elemente $z \in C$ und $y \in K_{i-1}(C)$ mit

$$yx = z \quad \text{oder} \quad zx = y. \tag{1.20}$$

Gilt $yx = z$, so folgt $yC^* \cap C^* \neq \emptyset$. Außerdem haben wir $C^* y \cap C^* \neq \emptyset$ wegen (1.19). Damit folgt aus Satz 1.5 (beachte, dass jeder Code produktunabhängig ist), dass y in C^* liegt, d. h. es gilt $y = y_1 y_2 \dots y_l$ für gewisse $y_k \in C$, $1 \leq k \leq l$. Somit erhalten wir

$$yx = y_1 y_2 \dots y_l x = z$$

mit $y_1 \neq z$. Dies ist wegen Satz 1.1 ein Widerspruch zur Voraussetzung, dass C ein Code ist.

Folglich muss von den Gleichheiten in (1.20) die zweite gelten, d. h. $zx = y$. Wir bemerken zuerst, dass dann $i \geq 2$ gilt, denn für $i = 1$ würde $y \in K_0(C) = C$ als Produkt von Elementen $z \in C$ und $x \in C$ darstellbar sein, womit sich erneut ein Widerspruch zu Satz 1.1 ergeben würde. Daher gibt es nach Definition von $K_{i-1}(C)$ Elemente $y_1 \in K_{i-2}(C)$ und $z_1 \in C$ so, dass

$$y_1 y = z_1 \quad \text{oder} \quad z_1 y = y_1$$

gilt. Die erste dieser Möglichkeiten führt zu $y_1 zx = z_1$ und dann analog zur ersten der beiden Gleichheiten in (1.20) zu einem Widerspruch. Aus der zweiten möglichen Gleichheit erhalten wir

$$z_1 zx = z_1 y = y_1.$$

Ist $i = 2$, so ergibt sich wegen $y_1 \in K_{i-2}(C) = K_0(C) = C$ aus der vorstehenden Gleichheit ein Widerspruch zu Satz 1.1. Damit muss $i \geq 3$ gelten.

Analog fahren wir fort und erhalten, dass $i \geq n_0$ für jede Schranke n_0 gilt. Das bedeutet aber gerade, dass für alle $i \geq 1$ die Menge $K_i(C) \cap C$ leer ist, was zu zeigen war. □

Satz 1.10 *Ein Code C über X ist genau dann ein strenger Code, wenn $K_n(C) = \emptyset$ für $n \geq \#(C)(\max\{|c| \mid c \in C\} - 1) + 1$ gilt.*

Beweis. Hinlänglichkeit: Wir setzen

$$m = \#(C)(\max\{|c| \mid c \in C\} - 1) + 1\,.$$

Wir nehmen an, dass $K_n(C) \neq \emptyset$ für ein $n \geq m$ gilt. Dann ist auch $K_m(C) \neq \emptyset$. Es sei nun $v_m \in K_m(C)$. Dann gibt es ein $v_{m-1} \in K_{m-1}(C)$ derart, dass $xv_m = v_{m-1}$ oder

$v_{m-1}v_m = x$ für ein $x \in C$ gilt. Zu v_{m-1} gibt es wieder ein $v_{m-2} \in K_{m-2}(C)$ usw. Es sei $v_0, v_1, \ldots, v_{m-1}, v_m$ die so erzeugte Folge von Elementen. Nach Lemma 1.7 liegt jedes Element in $\mathit{Suff}(C)$. Da jedes Wort x aus C höchstens $|x|$ verschiedene nichtleere Suffixe hat, gibt es insgesamt höchstens m nichtleere Wörter in $\mathit{Suff}(C)$. Daher müssen zwei Wörter der Folge $v_0, v_1, \ldots v_m$ gleich sein. Es sei $v_{h_1} = v_{h_2}$.

Mit Blick auf den Beweis von Lemma 1.8 erkennen wir, dass v_i gerade dass nach Lemma 1.8 existierende Element aus $K_i(C)$ ist. Folglich erhalten wir aus Lemma 1.8, dass entweder

$$v_{h_1}x_{i_1}x_{i_2}\ldots x_{i_k}v_{h_2} = x_{j_1}x_{j_2}\ldots x_{j_l} \text{ mit } l \geq 1 \text{ und } |v_{h_2}| < |x_{j_l}|$$

oder

$$v_{h_1}x_{i_1}x_{i_2}\ldots x_{i_k} = x_{j_1}x_{j_2}\ldots x_{j_l}v_{h_2} \text{ mit } k \geq 1,\ l \geq 1 \text{ und } |v_{h_2}| < |x_{i_k}|$$

und damit unter Verwendung von $v = v_{h_1} = v_{h_2}$

$$vx_{i_1}x_{i_2}\ldots x_{i_k}v = x_{j_1}x_{j_2}\ldots x_{j_l} \text{ mit } l \geq 1 \text{ und } |v| < |x_{j_l}| \tag{1.21}$$

oder

$$vx_{i_1}x_{i_2}\ldots x_{i_k} = x_{j_1}x_{j_2}\ldots x_{j_l}v \text{ mit } k \geq 1,\ l \geq 1 \text{ und } |v| < |x_{i_k}| \tag{1.22}$$

für geeignete Wörter $x_{i_1}, \ldots, x_{i_k}, x_{j_1}, \ldots, x_{j_l} \in C$ gilt. Im ersten Fall erhalten wir durch Multiplikation mit $x_{i_1}x_{i_2}\ldots x_{i_k}v$

$$vx_{i_1}x_{i_2}\ldots x_{i_k}vx_{i_1}x_{i_2}\ldots x_{i_k}v = x_{j_1}x_{j_2}\ldots x_{j_l}x_{i_1}x_{i_2}\ldots x_{i_k}v$$

und

$$vx_{i_1}x_{i_2}\ldots x_{i_k}vx_{i_1}x_{i_2}\ldots x_{i_k}v = vx_{i_1}x_{i_2}\ldots x_{i_k}x_{j_1}x_{j_2}\ldots x_{j_l}\,,$$

woraus

$$vx_{i_1}x_{i_2}\ldots x_{i_k}x_{j_1}x_{j_2}\ldots x_{j_l} = x_{j_1}x_{j_2}\ldots x_{j_l}x_{i_1}x_{i_2}\ldots x_{i_k}v$$

folgt, also eine Gleichheit der Art aus (1.22). Daher brauchen wir im Folgenden nur noch den Fall aus (1.22) zu diskutieren.

Durch wiederholte Anwendung von 1.22 erhalten wir

$$\begin{aligned} v(x_{i_1}x_{i_2}\ldots x_{i_k})^r &= vx_{i_1}x_{i_2}\ldots x_{i_k}(x_{i_1}x_{i_2}\ldots x_{i_k})^{r-1} \\ &= x_{j_1}x_{j_2}\ldots x_{j_l}v(x_{i_1}x_{i_2}\ldots x_{i_k})^{r-1} \\ &= (x_{j_1}x_{j_2}\ldots x_{j_l})^2v(x_{i_1}x_{i_2}\ldots x_{i_k})^{r-2} \\ &\vdots \\ &= (x_{j_1}x_{j_2}\ldots x_{j_l})^rv\,. \end{aligned} \tag{1.23}$$

Außerdem gilt nach Lemma 1.8 unter Beachtung von $v = v_{h_1} \in K_{h_1}(C)$

$$v_0y_{f_1}y_{f_2}\ldots y_{f_s}v = y_{g_1}y_{g_2}\ldots y_{g_t} \text{ und } |v| < |y_{g_t}|$$

oder

$$v_0 y_{f_1} y_{f_2} \dots y_{f_s} = y_{g_1} y_{g_2} \dots y_{g_t} v \text{ und } |v| < |y_{f_s}|$$

für gewisse Codewörter $y_{f_1}, y_{f_2}, \dots, y_{f_s}, y_{g_1}, y_{g_2}, \dots, y_{g_t}$. Unter Berücksichtigung der Beziehung $v_0 \in K_0(C) = C$ ergibt sich

$$y_{p_1} y_{p_2} \dots y_{p_u} v = y_{q_1} y_{q_2} \dots y_{q_w} \text{ und } |v| < |y_{q_w}| \tag{1.24}$$

für gewisse Codewörter $y_{p_1}, y_{p_2}, \dots y_{p_u}, y_{q_1}, y_{q_2}, \dots, y_{q_w}$. Es gibt einen Index i mit $1 \leq i \leq u$ derart, dass $y_{p_i} \neq y_{g_i}$ aber für alle Indices $j < i$ die Gleichheit $y_{p_j} = y_{g_j}$ gilt (wenn es so einen Index i nicht gäbe, wäre $v = y_{q_{u+1}} y_{q_{u+2}} \dots y_{q_w}$, was ein Widerspruch zu $|v| < |y_{q_w}|$ ist).

Aus den Gleichheiten (1.23) und (1.24) erhalten wir

$$y_{p_i} y_{p_{i+1}} \dots y_{p_u} v (x_{i_1} x_{i_2} \dots x_{i_k})^r = y_{p_i} y_{p_{i+1}} \dots y_{p_u} (x_{j_1} x_{j_2} \dots x_{j_l})^r v$$

und

$$y_{p_i} y_{p_{i+1}} \dots y_{p_u} v (x_{i_1} x_{i_2} \dots x_{i_k})^r = y_{q_i} y_{q_{i+1}} \dots y_{q_w} (x_{i_1} x_{i_2} \dots x_{i_k})^r$$

und damit

$$y_{p_i} y_{p_{i+1}} \dots y_{p_u} (x_{j_1} x_{j_2} \dots x_{j_l})^r v = y_{q_i} y_{q_{i+1}} \dots y_{q_w} (x_{i_1} x_{i_2} \dots x_{i_k})^r \,.$$

Da diese Gleichung für jede natürliche Zahl $r \geq 1$ gilt und $y_{p_i} \neq y_{q_i}$ ist, erhalten wir einen Widerspruch dazu, dass C ein strenger Code ist.

Notwendigkeit: Wir nehmen an, dass C kein strenger Code ist. Dann gibt es Codewörter x_{i_k} und x_{j_k}, $k \geq 1$, so dass

$$x_{i_1} x_{i_2} \dots = x_{j_1} x_{j_2} \dots \quad \text{mit} \quad i_1 \neq j_1$$

gilt. Folglich besteht für beliebiges k eine Gleichheit

$$x_{i_1} x_{i_2} \dots x_{i_k} v_k = x_{j_1} x_{j_2} \dots x_{j_{l(k)}} \quad \text{mit} \quad |v_k| < |x_{j_{l(k)}}| \,.$$

Aus Lemma 1.8 (mit $v_0 = x_{i_1}$ oder $v_0 = x_{j_1}$) folgt nun $v_k \in K_{k+l(k)-1}(C)$. Da k beliebig groß werden kann, ist $K_m(C) \neq \emptyset$ und damit ein Widerspruch zur Voraussetzung hergeleitet. □

Es sei C eine endliche Sprache. Wir setzen

$$k = \max\{|v| \mid v \in C\}.$$

Dann folgt aus Lemma 1.7, dass jede Menge $K_i(C)$, $i \geq 0$, eine Teilmenge aller Wörter der Länge $l \leq k$ ist. Folglich ist die Menge

$$K(C) = \{K_i(C) \mid i \geq 0\}$$

endlich. Daher gibt es Zahlen i und j mit $i < j$ und $K_i(C) = K_j(C)$. Es ist leicht zu sehen, dass dann $K_{j+k}(C) = K_{i+k}(C)$ für $k \geq 0$ gilt. Wählen wir j minimal mit dieser Eigenschaft, so gilt

$$K(C) = \{K_0(C), K_1(C), \ldots, K_{j-1}(C)\}.$$

Hieraus folgt, dass es für endliche Sprachen C über endlichen Alphabeten einen Algorithmus gibt, der die Menge $K(C)$ bestimmt. Wir ermitteln einfach der Reihe nach die Mengen $K_0(C), K_1(C), K_2(C), \ldots$ und brechen ab, wenn wir eine Menge $K_j(C)$ erhalten, die bereits unter den Mengen $K_i(C)$ mit $i < j$ vorkommt.

Die Menge aller Wörter der Länge l (für eine natürliche Zahl $l \geq 0$) über einem m-elementigen Alphabet X (für eine natürliche Zahl $m \geq 1$) besteht aus m^l Wörtern. Daraus ergibt sich, dass das für jene Zahl j, bei der sich zum ersten Mal eine Menge in $K(C)$ wiederholt, die Abschätzung

$$j \leq 2^{m+m^2+\cdots+m^k} = 2^{\frac{m^{k+1}-1}{m-1}-1}$$

gilt.

Aus Satz 1.9 und Satz 1.10 ergibt sich sofort der folgende Satz.

Satz 1.11 *Es gibt einen Algorithmus, der für jede endliche Sprache C über einem endlichen Alphabet X entscheidet, ob C ein (strenger) Code ist.*

Beweis. Unter den gegebenen Voraussetzungen kann – wie oben gezeigt – $K(C)$ algorithmisch bestimmt werden. Wir haben nun nur noch zu testen, ob für die endlich vielen Mengen $K_i(C)$ mit $i \geq 1$ auch $K_i(C) \cap C$ leer ist. Fällt dieser Test positiv aus, so ist C entsprechend Satz 1.9 ein Code; andernfalls ist C kein Code.

Um herauszufinden, ob C ein strenger Code ist, ermitteln wir zunächst, ob C ein Code ist. Falls die Sprache C kein Code ist, ist sie auch kein strenger Code. Falls C ein Code ist, berechnen wir die Menge $K_m(C)$ für $m = \#(C)(\max\{|c| \mid c \in C\} - 1) + 1$ (was wegen der Periodizität der Mengen $K_i(C)$, $i \geq 1$, nach Obigem möglich ist) und untersuchen, ob $K_m(C)$ die leere Menge ist. Der Code C ist genau dann streng, wenn $K_m(C) = \emptyset$ gilt. □

Beispiel 1.5 Wenden wir den Algorithmus auf

$$C_0 = \{a, ab, ba\} \quad \text{und} \quad C_1 = \{a, bb, aab, bab\}$$

aus Beispiel 1.4 an, so erhalten wir wegen der in Beispiel 1.4 jeweils bestimmten Mengen $K_i(C_0)$ und $K_i(C_1)$, dass C_0 kein Code ist, während C_1 ein Code, aber kein strenger Code ist. ◊

1.4 Code-Indikator und Konstruktion von Codes

Entsprechend den bisher angegebenen Resultaten können wir für eine gegebene Menge feststellen, ob sie ein Code ist. Es bleibt aber noch die Aufgabe, einen Algorithmus zu finden, der einen Code konstruiert. Ein solches Verfahren wird sich aus der nachfolgenden hinreichenden Charakterisierung von Codes durch den Code-Indikator ergeben.

Definition 1.7 *Es sei X ein Alphabet der Kardinalität $n \geq 2$. Der Code-Indikator $ci_n(w)$ eines Wortes $w \in X^*$ ist durch*

$$ci_n(w) = n^{-|w|}$$

definiert. Zu einer Sprache L sei n_L die Anzahl aller vorkommenden Buchstaben (das bezüglich der Inklusion kleinste Alphabet X mit $L \subseteq X^*$ hat die Kardinalität n_L). Wir setzen

$$ci(L) = \sum_{w \in L} ci_{n_L}(w).$$

Beispiel 1.6 Für die zu Beginn eingeführte Menge C_0, die Mengen C_1 und C_3 aus Beispiel 1.1 und die Menge $C_5 = \{a, ba, bb, aab\}$ ergeben sich die folgenden Code-Indikatoren:

$$\begin{aligned}
C_0 &= \{a, ab, ba\} && \rightsquigarrow ci(C_0) = \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{4} = 1,\\
C_1 &= \{a, bb, aab, bab\} && \rightsquigarrow ci(C_1) = \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{8} = 1,\\
C_3 &= \{aaa, aba, bab, bbb\} && \rightsquigarrow ci(C_3) = \tfrac{1}{8} + \tfrac{1}{8} + \tfrac{1}{8} + \tfrac{1}{8} = \tfrac{1}{2},\\
C_5 &= \{a, ba, bb, aab\} && \rightsquigarrow ci(C_5) = \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{4} + \tfrac{1}{8} = \tfrac{9}{8}.
\end{aligned}$$

$\Diamond$

Satz 1.12 *Es seien L_1 und L_2 zwei Sprachen über einem Alphabet X, das aus n Buchstaben bestehe. Dann gilt*

$$ci(L_1 \cdot L_2) \leq ci(L_1) \cdot ci(L_2),$$

und die Gleichheit tritt genau dann ein, wenn für je vier Wörter w_1, w_2, w_3, w_4 mit $w_1, w_2 \in L_1$ und $w_3, w_4 \in L_2$ aus $w_1 w_3 = w_2 w_4$ folgt, dass $w_1 = w_2$ gilt.

Beweis. Die behauptete Ungleichung ergibt sich aus

$$\begin{aligned}
ci(L_1 \cdot L_2) &= ci(\{v_1 v_2 \mid v_1 \in L_1,\ v_2 \in L_2\})\\
&\leq \sum_{v_1 \in L_1, v_2 \in L_2} n^{-|v_1 v_2|}\\
&= \sum_{v_1 \in L_1, v_2 \in L_2} n^{-(|v_1| + |v_2|)}\\
&= \sum_{v_1 \in L_1, v_2 \in L_2} n^{-|v_1|} \cdot n^{-|v_2|}\\
&= \sum_{v_1 \in L_1} n^{-|v_1|} \cdot \sum_{v_2 \in L_2} n^{-|v_2|}\\
&= ci(L_1) \cdot ci(L_2).
\end{aligned}$$

Dabei gilt die Gleichheit genau dann, wenn kein Element w aus $L_1 L_2$ zwei verschiedene Darstellungen als Produkt hat, da sonst w bei der Berechnung des Code-Indikators von $L_1 L_2$ nur einmal betrachtet wird, während bei der Berechnung von $ci(L_1)ci(L_2)$ beide Darstellungen berücksichtigt werden. $\square$

Wir geben nun den Satz an, der die Bezeichnung Code-Indikator rechtfertigt.

Satz 1.13 *Für jeden Code C gilt $ci(C) \leq 1$.*

Beweis. Wir zeigen zuerst mittels Induktion, dass

$$ci(C^i) = (ci(C))^i$$

für alle natürlichen Zahlen $i \geq 1$ gilt.

Für $i = 1$ ist dies offensichtlich.

Es sei $w \in C^i$. Dann gibt es wegen Satz 1.2 genau eine Darstellung von w als Produkt von i Elementen aus C. Insbesondere ist damit w in genau einer Weise als Produkt von einem Element aus C und einem Element aus C^{i-1} darstellbar. Somit erhalten wir aus Satz 1.12 und der Induktionsvoraussetzung

$$ci(C^i) = ci(C \cdot C^{i-1}) = ci(C) \cdot ci(C^{i-1}) = ci(C) \cdot (ci(C))^{i-1} = (ci(C))^i.$$

Es seien nun

$$k = \min\{|w| \mid w \in C\} \quad \text{und} \quad K = \max\{|w| \mid w \in C\},$$

so gilt für jedes Wort $v \in C^j$, $j \geq 1$,

$$|v| \geq jk \quad \text{und} \quad |v| \leq jK.$$

Es sei n die Anzahl der in der Sprache vorkommenden Buchstaben (die Kardinalität des kleinsten Alphabets, über dem C definiert ist). Dann folgt

$$\begin{aligned} ci(C^j) &= \sum_{i=jk}^{jK} \sum_{v \in C^j, |v|=i} n^{-|v|} \\ &\leq \sum_{i=jk}^{jK} \sum_{|v|=i} n^{-|v|} \\ &= \sum_{i=jk}^{jK} 1 = jK - jk + 1 = (K-k)j + 1. \end{aligned}$$

Gilt nun $ci(C) > 1$, so wächst $(ci(C))^j$ exponentiell in j, und somit gibt es eine natürliche Zahl j mit

$$ci(C^j) = (ci(C))^j > (K-k)j + 1,$$

womit wir einen Widerspruch zur Ungleichung davor erhalten. Daher muss $ci(C) \leq 1$ erfüllt sein. □

Die Umkehrung von Satz 1.13 gilt nicht, denn die Menge C_0 ist kein Code, wie zu Beginn dieses Kapitels nachgewiesen wurde, erfüllt aber die Bedingung $ci(C_0) \leq 1$ (siehe Beispiel 1.6).

Aus Satz 1.13 folgt aber sofort, dass C_5 aus Beispiel 1.6 kein Code ist.

Satz 1.14 *Es seien $n \geq 2$ und $l_1, l_2, \ldots, l_m$, $m \geq 1$, natürliche positive Zahlen, die der Bedingung*

$$\sum_{i=1}^{m} n^{-l_i} \leq 1$$

genügen. Dann gibt es einen Code (Präfixcode)

$$C = \{c_0, c_1, \ldots, c_{m-1}\}$$

über einem n-elementigen Alphabet X mit

$$|c_{i-1}| = l_i \quad \textit{für} \quad 1 \leq i \leq m.$$

Beweis. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass die gegebenen Zahlen geordnet sind, d. h.

$$l_1 \leq l_2 \leq \cdots \leq l_m.$$

Wir definieren zuerst Zahlen q_i, $0 \leq i \leq m-1$, wie folgt:

$$q_0 = 0,$$

$$q_i = q_{i-1} + n^{-l_i} = \sum_{j=1}^{i} n^{-l_j} \quad \text{für} \quad 1 \leq i \leq m-1.$$

Für $0 \leq i \leq m-1$ gelten dann folgende Aussagen:

- $0 \leq q_i = \sum_{j=1}^{i} n^{-l_j} < \sum_{j=1}^{m} n^{-l_j} \leq 1$.
- Die n-äre Darstellung von q_i hat offensichtlich höchstens l_i Stellen hinter dem Komma (da n^{-l_i} der minimale Wert der Summanden bei der Bildung von q_i ist).

Es sei nun $0, a_i^{(1)} a_i^{(2)} \ldots a_i^{(l_{i+1})}$ die n-äre Darstellung von q_i (sollte $l_{i+1} > l_i$ sein, so ergänzen wir am Ende einige Nullen). Dann setzen wir

$$c_i = a_i^{(1)} a_i^{(2)} \ldots a_i^{(l_{i+1})}$$

Offenbar gilt damit $|c_i| = l_{i+1}$.

Wir zeigen nun, dass die Menge $C = \{c_0, c_1, \ldots, c_{m-1}\}$ ein Präfixcode ist. Dazu nehmen wir an, dass c_r Präfix von c_s für gewisse r und s gilt, und führen dies zu einem Widerspruch. Es sei

$$c_s = c_r b_r^{(l_{r+1}+1)} b_r^{(l_r+2)} \ldots b_r^{(l_{s+1})}.$$

Dann folgt

$$q_s = q_r + \sum_{j=1}^{l_{s+1}-l_{r+1}} b_r^{(l_{r+1}+j)} n^{-(l_{r+1}+j)} < q_r + n^{-l_{r+1}}.$$

Andererseits gilt

$$q_s \geq q_{r+1} = q_r + n^{-l_{r+1}}.$$

Die beiden letzten Ungleichungen widersprechen sich. □

Wir illustrieren die im Beweis von Satz 1.14 gegebene Methode durch ein Beispiel.

Beispiel 1.7 Wir setzen $n = 2$ und $X = \{0, 1\}$. Ferner seien

$$l_1 = l_2 = l_3 = 3, \quad l_4 = l_5 = l_6 = l_7 = 4.$$

Dann ist die Bedingung

$$\sum_{i=1}^{7} 2^{-l_i} = \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{16} + \frac{1}{16} + \frac{1}{16} + \frac{1}{16} = \frac{5}{8} \leq 1$$

erfüllt. Entsprechend dem Beweis von Satz 1.14 ergeben sich die Werte der folgenden Tabelle:

i	l_i	q_{i-1}	Dualdarst. von q_{i-1}	c_{i-1}
1	3	$q_0 = 0$	0,000	000
2	3	$q_1 = q_0 + \frac{1}{8} = 0 + \frac{1}{8} = \frac{1}{8}$	0,001	001
3	3	$q_2 = q_1 + \frac{1}{8} = \frac{1}{8} + \frac{1}{8} = \frac{1}{4}$	0,010	010
4	4	$q_3 = q_2 + \frac{1}{8} = \frac{1}{4} + \frac{1}{8} = \frac{3}{8}$	0,0110	0110
5	4	$q_4 = q_3 + \frac{1}{16} = \frac{3}{8} + \frac{1}{16} = \frac{7}{16}$	0,0111	0111
6	4	$q_5 = q_4 + \frac{1}{16} = \frac{7}{16} + \frac{1}{16} = \frac{1}{2}$	0,1000	1000
7	4	$q_6 = q_5 + \frac{1}{16} = \frac{1}{2} + \frac{1}{16} = \frac{9}{16}$	0,1001	1001

Daher ergibt sich für die vorgegebenen Längen die Menge

$$C = \{000, 001, 010, 0110, 0111, 1000, 1001\}$$

als Präfixcode. ◊

Definition 1.8 *Ein Code C heißt maximal, wenn für jedes Wort $w \notin C$ die Menge $C \cup \{w\}$ kein Code ist.*

Zu einem maximalen Code lässt sich nach Definition 1.8 kein Wort hinzufügen, ohne dass die Eigenschaft, Code zu sein, verloren geht.

Satz 1.15 *Ein Code C mit $ci(C) = 1$ ist ein maximaler Code.*

Beweis. Es sei $ci(C) = 1$ für einen Code C. Wenn C nicht maximal ist, gibt es ein Wort $w \notin C$ derart, dass auch $C' = C \cup \{w\}$ ein Code ist. Wegen

$$ci(C') = ci(C) + ci(w) > 1$$

ergibt sich ein Widerspruch zu Satz 1.13. □

Der folgende Satz, den wir ohne Beweis angeben, verschärft Satz 1.15 für endliche Codes.

Satz 1.16 *Ein endlicher Code C ist genau dann maximal, wenn $ci(C) = 1$ gilt.* □

Kapitel 2

Optimale Codes

Im vorhergehenden Abschnitt haben wir eine Methode angegeben, mittels derer entschieden werden kann, ob eine vorgebene Menge von Wörtern ein Code ist, bzw. mittels derer ein Code konstruiert werden kann. Dabei wurde aber nicht darauf geachtet, ob der erzeugte Code noch weitere Eigenschaften hat, die aus praktischen oder theoretischen Gründen für eine Codierung wichtig sein können. In diesem und dem folgenden Abschnitt werden wir daher klären, ob Codes mit gewissen Eigenschaften existieren und wie diese dann erzeugt werden können.

Als erstes sind wir daran interessiert, Codes zu konstruieren, bei denen die zu übermittelnde Nachricht nach der Codierung im Durchschnitt möglichst kurz ist.

Es sei eine Quelle gegeben, die in zufälliger Weise nacheinander Buchstaben eines Alphabets $A = \{a_1, a_2, \ldots, a_m\}$ erzeugt, wobei das Erzeugen eines einzelnen Buchstaben unabhängig von den bereits erzeugten Buchstaben erfolgt und der Buchstabe a_i, $1 \leq i \leq m$, mit der Wahrscheinlichkeit p_i erzeugt wird. Dann müssen $p_i \geq 0$ für $1 \leq i \leq m$ und $\sum_{i=1}^{m} p_i = 1$ gelten. Der dadurch erzeugte Text $T = a_{i_1} a_{i_2} \ldots a_{i_n}$ der Länge n werde entsprechend der Codierung $\varphi : A = \{a_1, a_2, \ldots, a_m\} \to C \subseteq X^+$ vermöge $\varphi(a_i) = c_i$, $1 \leq i \leq m$, codiert. Für die Länge des Textes nach der Codierung ergibt sich

$$|\varphi^*(T)| = |c_{i_1}||c_{i_2}| \ldots |c_{i_n}| = \sum_{i=1}^{m} \#_{a_i}(T) \cdot |c_i|.$$

Wenn der Text T hinreichend lang ist, können wir annehmen, dass jeder Buchstabe a_i, $1 \leq i \leq m$, entsprechend seiner Wahrscheinlichkeit p_i in T vorkommt. Dann gilt

$$\#_{a_i}(T) = p_i \cdot |T| = p_i \cdot n.$$

Folglich ergibt sich

$$|\varphi^*(T)| = \sum_{i=1}^{m} p_i \cdot n \cdot |c_i| = n \cdot \sum_{i=1}^{m} p_i \cdot |c_i|.$$

Da n durch $|T|$ festgelegt ist, können wir die Länge von $\varphi^*(T)$ nur dadurch optimieren, dass wir φ so wählen, dass $\sum_{i=1}^{m} p_i |c_i|$ möglichst klein wird.

Die folgenden Definitionen formalisieren die vorstehenden Überlegungen.

Definition 2.1 *i) Für einen Code $C = \{c_1, c_2, \ldots, c_m\}$ und eine Wahrscheinlichkeitsverteilung $P = \{p_1, p_2, \ldots, p_m\}$, $p_i \geq 0$ für $1 \leq i \leq m$, $\sum_{i=1}^{m} p_i = 1$, definieren wir die Kosten von C unter P durch*

$$\mathcal{L}(C, P) = \sum_{i=1}^{m} p_i |c_i|.$$

ii) Für eine Wahrscheinlichkeitsverteilung $P = \{p_1, p_2, \ldots, p_m\}$, $p_i \geq 0$ für $1 \leq i \leq m$, $\sum_{i=1}^{m} p_i = 1$, und ein Alphabet X setzen wir

$$\mathcal{L}_X(P) = \inf \mathcal{L}(C, P),$$

wobei das Infimum über alle m-elementigen Codes über X zu nehmen ist. Ein Code C' über X heißt optimal für P, wenn

$$\mathcal{L}(C', P) = \mathcal{L}_X(P)$$

gilt.

Nach der Definition von $\mathcal{L}_X(P)$ haben wir das Infimum über alle m-elementigen Codes über X zu nehmen. Es reicht aber das Infimum über alle m-elementigen Präfixcodes zu nehmen. Dies folgt daraus, dass die Größe $\mathcal{L}_X(P)$ nur von den Längen der Codewörter abhängt (da die Wahrscheinlichkeiten als fest gegeben angesehen werden können) und zu vorgegebenen Längen der Codewörter stets ein Präfixcode konstruiert werden kann (siehe Beweis von Satz 1.14). Aus gleichem Grund ist auch die Existenz eines für die Verteilung P optimalen Präfixcodes gesichert, falls es einen für P optimalen Code gibt.

Im Folgenden nehmen wir stets an, dass alle Wahrscheinlichkeiten p_i, $1 \leq i \leq m$, der Verteilung P positiv sind. [1]

Wir zeigen zuerst, dass unter dieser Voraussetzung für jede Verteilung und jedes Alphabet ein optimaler Code existiert.

Satz 2.1 *Für jede Verteilung P, deren Wahrscheinlichkeiten alle positiv sind, und jedes Alphabet X existiert ein (Präfix)-Code über X, der optimal für P ist.*

Beweis. Es habe P genau $m \geq 2$ und X genau $n \geq 2$ Elemente. Dann setzen wir

$$l_i = \lceil m + \log_n(m) \rceil \quad \text{für} \quad 1 \leq i \leq m.$$

Offensichtlich gilt dann

$$\sum_{i=1}^{m} n^{-l_i} \leq \sum_{i=1}^{m} n^{-m-\log_n(m)} = m \cdot n^{-m-\log_n(m)} = m \cdot \frac{n^{-m}}{n^{\log_n(m)}} = m \cdot \frac{n^{-m}}{m} = n^{-m} < 1.$$

Entsprechend Satz 1.14 können wir nun einen Präfixcode $C = \{c_1, c_2, \ldots, c_m\}$ mit $|c_i| = l_i$ für $1 \leq i \leq m$ erzeugen.

[1] Die meisten der folgenden Betrachtungen gelten auch für den Fall, dass einige der Wahrscheinlichkeiten 0 sind, jedoch verkomplizieren sich sich dann die Beweise. Andererseits kann auf Buchstaben, die mit der Wahrscheinlichkeit 0 vorkommen, in der Praxis meist verzichtet werden.

Es sei p die minimale der Wahrscheinlichkeiten von P. Dann setzen wir

$$k = \frac{\mathcal{L}(C,P)}{p}.$$

Hat ein Code C' ein Wort einer Länge $l > k$, so gilt für seine Kosten

$$\mathcal{L}(C',P) \geq p \cdot l > p \cdot k = p \cdot \frac{\mathcal{L}(C,P)}{p} = \mathcal{L}(C,P).$$

Folglich kann C' nicht optimal für P sein, da seine Kosten die von C übersteigen. Daher muss ein optimaler Code für P nach den obigen Ausführungen ein (Präfix)-Code sein, der aus m Wörtern mit einer Länge $\leq k$ besteht. Offensichtlich gibt es nur eine endliche Anzahl von Mengen, die aus m Wörtern der Länge $\leq k$ bestehen. Unter diesen bestimmen wir alle (Präfix)-Codes und erhalten einen optimalen Code durch Minimierung der Kosten über dieser endlichen Menge. □

Diese Methode ist aber sehr aufwendig, da wir maximal $\binom{n^k}{m}$ Mengen betrachten müssen. Daher sind auch Verfahren von Interesse, mittels derer kostengünstige, aber unter Umständen nicht optimale Codes gewonnen werden können. Deshalb geben wir jetzt eine Abschätzung für die Größe $\mathcal{L}_X(P)$ an.

Satz 2.2 *Für jede Verteilung $P = \{p_1, p_2, \ldots, p_m\}$ und jedes Alphabet X mit $\#(X) = n$ gilt*

$$\sum_{i=1}^{m} p_i \log_n \left(\frac{1}{p_i}\right) \leq \mathcal{L}_X(P) \leq 1 + \sum_{i=1}^{m} p_i \log_n \left(\frac{1}{p_i}\right),$$

wobei die Gleichheit

$$\mathcal{L}_X(P) = \sum_{i=1}^{m} p_i \log_n \left(\frac{1}{p_i}\right)$$

genau dann gilt, wenn $\log_n(p_i)$ für $1 \leq i \leq m$ ganze Zahlen sind.

Beweis. Es sei $C = \{c_1, c_2, \ldots, c_m\}$ ein beliebiger Code über X. Mit l_i bezeichnen wir die Länge des Wortes c_i, $1 \leq i \leq m$. Unter Beachtung der üblichen Regeln für das Rechnen mit Logarithmen, den Beziehungen $\log_n(x) = \log_n(e)\ln(x)$ (wobei ln den Logarithmus zur Basis e bezeichnet), $\ln(x) \leq x - 1$ und Satz 1.13 erhalten wir

$$\begin{aligned}
\sum_{i=1}^{m} p_i \log_n \left(\frac{1}{p_i}\right) - \sum_{i=1}^{m} p_i l_i &= \sum_{i=1}^{m} p_i (\log_n \left(\frac{1}{p_i}\right) - l_i) = \sum_{i=1}^{m} p_i (\log_n \left(\frac{1}{p_i}\right) - \log_n(n^{l_i})) \\
&= \sum_{i=1}^{m} p_i \log_n \left(\frac{n^{-l_i}}{p_i}\right) = \sum_{i=1}^{m} p_i \log_n(e) \ln \left(\frac{n^{-l_i}}{p_i}\right) \\
&= \log_n(e) \sum_{i=1}^{m} p_i \ln \left(\frac{n^{-l_i}}{p_i}\right) \leq \log_n(e) \sum_{i=1}^{m} p_i \left(\frac{n^{-l_i}}{p_i} - 1\right) \\
&= \log_n(e)(\sum_{i=1}^{m} n^{-l_i} - \sum_{i=1}^{m} p_i) = \log_n(e)(\sum_{i=1}^{m} n^{-l_i} - 1) \leq 0
\end{aligned}$$

und damit

$$\mathcal{L}(C,P) = \sum_{i=1}^{m} p_i l_i \geq \sum_{i=1}^{m} p_i \log_n \left(\frac{1}{p_i}\right).$$

Da diese Abschätzung für alle Codes C gilt, gilt sie auch für das Infimum der Kosten, womit

$$\mathcal{L}_X(P) \geq \sum_{i=1}^{m} p_i \log_n \left(\frac{1}{p_i}\right)$$

bewiesen ist.

Da $\ln(x) = x - 1$ nur für $x = 1$ gilt, folgt aus der vorstehenden Abschätzung sofort, dass die Gleichheit nur gilt, wenn $n^{-l_i} = p_i$ für $1 \leq i \leq m$ erfüllt ist.

Zum Beweis der oberen Abschätzung im Satz konstruieren wir zuerst wie folgt einen Code, wobei wir ohne Beschränkung der Allgemeinheit annehmen, dass die Wahrscheinlichkeiten in absteigender Folge geordnet sind, dass also

$$p_1 \geq p_2 \geq \cdots \geq p_m$$

gilt. Wir setzen

- $l_i = \lceil \log_n \left(\frac{1}{p_i}\right) \rceil$,
- $q_0 = 0$ und $q_i = \sum_{j=1}^{i} p_j$ für $1 \leq i \leq m-1$

und bestimmen die Codewörter $c_0, c_1, \ldots, c_{m-1}$ dann entsprechend der im Beweis von Satz 1.14 angegebenen Methode aus den n-ären Darstellungen der Zahlen $q_0, q_1, \ldots, q_{m-1}$. Wie im Beweis von Satz 1.14 kann gezeigt werden, dass die so konstruierten Wörter einen Präfixcode bilden.

Nach Konstruktion gilt für $C = \{c_0, c_1, \ldots, c_{m-1}\}$ die Abschätzung

$$\begin{aligned}\mathcal{L}(C,P) &= \sum_{i=1}^{m} p_i |c_{i-1}| = \sum_{i=1}^{m} p_i l_i \\ &\leq \sum_{i=1}^{m} p_i \left(\log_n(\frac{1}{p_i}) + 1\right) = \sum_{i=1}^{m} p_i \log_n \left(\frac{1}{p_i}\right) + \sum_{i=1}^{m} p_i = \sum_{i=1}^{m} p_i \log_n \left(\frac{1}{p_i}\right) + 1.\end{aligned}$$

Da nach Definition für jeden Code über X auch $\mathcal{L}_X(P) \leq \mathcal{L}(C,P)$ gültig ist, folgt

$$\mathcal{L}_X(P) \leq 1 + \sum_{i=1}^{m} p_i \log_n \left(\frac{1}{p_i}\right).$$

□

Beispiel 2.1. Wir betrachten die Verteilung P mit den Wahrscheinlichkeiten

$$p_1 = p_2 = 0.20, \quad p_3 = 0.19, \quad p_4 = 0.12, \quad p_5 = 0.11, \quad p_6 = p_7 = 0.09$$

und das Alphabet $X = \{0, 1\}$. Dann ergibt sich bei einer Rechnung mit fünf Stellen hinter dem Komma

$$\sum_{i=1}^{m} p_i \log\left(\frac{1}{p_i}\right) = 2.72666$$

und somit nach Satz 2.2

$$2.72666 \leq \mathcal{L}_X(P) \leq 3.72666 \ .$$

Wir konstruieren nun nach der im Beweis der oberen Abschätzung angegebenen Methode einen Code C_S, dessen Kosten höchstens 3.72666 sind. Die notwendigen Angaben können der folgenden Tabelle entnommen werden:

i	p_i	$\frac{1}{p_i}$	l_i	q_{i-1}	Dualdarst. q_{i-1}	c_{i-1}
1	0.20	5	3	0	0.0000000	000
2	0.20	5	3	0.20	0.0011001...	001
3	0.19	5.26...	3	0.40	0.0110011...	011
4	0.12	8.33...	4	0.59	0.1001011...	1001
5	0.11	9.09...	4	0.71	0.1011010...	1011
6	0.09	11.1...	4	0.82	0.1101001...	1101
7	0.09	11.1...	4	0.91	0.1110100...	1110

Die Kosten des so erhaltenen Präfixcodes

$$C_S = \{\ 000,\ 001,\ 011,\ 1001,\ 1011,\ 1101,\ 1110\ \}$$

betragen

$$\mathcal{L}(C_S, P) = \sum_{i=1}^{m} p_i l_i = (0.20 + 0.20 + 0.19) \cdot 3 + (0.12 + 0.11 + 0.09 + 0.09) \cdot 4 = 3.41 \ .$$

Jedoch ist einfach zu sehen, dass sich die Codewörter aus C_S verkürzen lassen, ohne dass die Eigenschaft Präfixcode zu sein, verlorengeht. So sind die Anfänge 01, 100, 101, 110, 111 der letzten fünf Codewörter nicht Präfix der anderen Codewörter. Daher können wir den Code C_S zum Code

$$C'_S = \{\ 000,\ 001,\ 01,\ 100,\ 101,\ 110,\ 111\ \}$$

verkürzen. Für diesen Code ergeben sich die Kosten

$$\mathcal{L}(C'_S, P) = (0.20 + 0.20) \cdot 3 + 0.19 \cdot 2 + (0.12 + 0.11 + 0.09 + 0.09) \cdot 3 = 2.81 \ ,$$

die deutlich unter denen des Codes C_S liegen. Es lässt sich noch eine weitere Verbesserung erreichen, wenn wir das kürzeste Codewort 01 der höchsten Wahrscheinlichkeit zuordnen. Durch diese Umordnung erhalten wir den Code

$$C''_S = \{\ 01,\ 000,\ 001,\ 100,\ 101,\ 110,\ 111\ \}$$

mit den Kosten

$$\mathcal{L}(C''_S, P) = 2.80\,.$$

Die Methode aus dem Beweis von Satz 2.2 zur Konstruktion eines kostengünstigen Codes geht auf C.E. SHANNON zurück. Sie erfordert einigen Rechenaufwand. Ein erheblich einfacheres Verfahren wurde von R.M. FANO für Codes über $\{0,1\}$ vorgeschlagen. Hierbei wird wie folgt vorgegangen:

Wir ordnen die Wahrscheinlichkeiten so, dass

$$p_1 \geq p_2 \geq \cdots \geq p_m$$

gilt. Wir beginnen mit

$$E_\lambda = \{p_1, p_2, \ldots, p_m\}$$

und konstruieren aus einer Menge

$$E_x = \{p_s, p_{s+1}, \ldots, p_t\}$$

mit $x \in \{0,1\}^*$ und $t - s \geq 1$ zwei Mengen E_{x0} und E_{x1} entsprechend der folgenden Vorschrift bis alle Mengen einelementig sind:

1. Für k, $s \leq k \leq t$ setzen wir

$$s_{k0} = \sum_{j=s}^{k} p_j \quad \text{und} \quad s_{k1} = \sum_{j=k+1}^{t} p_j$$

2. Wir bestimmen r so, dass

$$|s_{r1} - s_{r0}| = \min\{|s_{k1} - s_{k0}| \;\mid s \leq k \leq t\}$$

gilt.

3. Wir setzen

$$E_{x0} = \{p_s, p_{s+1}, \ldots, p_r\} \quad \text{und} \quad E_{x1} = \{p_{r+1}, p_{r+2}, \ldots, p_t\}.$$

Dadurch haben wir E_x in zwei Teile aufeinanderfolgender Wahrscheinlichkeiten geteilt, bei denen sich die Summen der Wahrscheinlichkeiten eines jeden Teiles möglichst wenig unterscheiden.

Gilt $E_x = \{p_j\}$, so nehmen wir x als j-tes Element des Codes, d. h. den Buchstaben, der mit der Wahrscheinlichkeit p_j auftritt, codieren wir durch x.

Wir haben zu zeigen, dass diese Konstruktion einen Code liefert. Dies folgt sofort aus der Tatsache, dass für zwei einelementige Mengen E_x und E_y weder x ein Präfix von y noch y ein Präfix von x ist, denn es gibt eine Menge E_z mit $x = z0z_1$ und $y = z1z_2$ oder

$y = z0z_1$ und $x = z1z_2$. Daraus resultiert dann auch, dass die mittels der Methode von FANO gewonnenen Codes Präfixcodes sind.

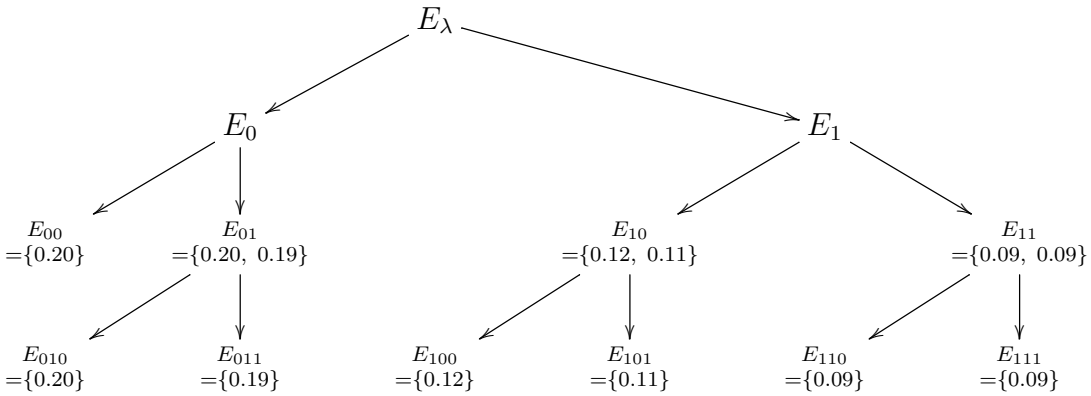
Beispiel 2.1. (Fortsetzung) Wir illustrieren die Methode von FANO anhand der Verteilung

$$P = \{0.20,\ 0.20,\ 0.19,\ 0.12,\ 0.11,\ 0.09,\ 0.09\}\ .$$

Es ergibt sich mit

$$E_\lambda = \{0.20,\ 0.20,\ 0.19,\ 0.12,\ 0.11,\ 0.09,\ 0.09\},$$
$$E_0 = \{\ 0.20,\ 0.20,\ 0.19\}, \quad E_1 = \{0.12,\ 0.11,\ 0.09,\ 0.09\},$$

der folgende Baum von Mengen.



und damit der Code

$$C_F = \{00,\ 010,\ 011,\ 100,\ 101,\ 110,\ 111\}$$

mit den Kosten

$$\mathcal{L}(C_F, P) = 0.20 \cdot 2 + (0.20 + 0.19 + 0.12 + 0.11 + 0.09 + 0.09) \cdot 3 = 2.80.$$

Daher ist der nach dem Verfahren von FANO konstruierte Code C_F in unserem Beispiel noch kostengünstiger als der mittels der Methode von SHANNON mit anschließender Kürzung gewonnene Code C'_S.

Die nach den Verfahren von SHANNON und FANO konstruierten Codes müssen nicht optimal sein (Beispiel 2.1 zeigt dies bereits für die Methode von SHANNON). Wir wollen nun eine Methode angeben, die auf D.A. HUFFMAN zurückgeht und mittels derer optimale Codes über $\{0, 1\}$ erzeugt werden. Sie basiert auf dem folgenden Satz.

Satz 2.3 *Es sei $C = \{c_1, c_2, \ldots, c_m\} \subseteq \{0,1\}^+$ ein optimaler Präfixcode für die Verteilung $P = \{p_1, p_2, \ldots, p_m\}$. Ferner gelte*

$$p_j = q_0 + q_1$$

und

$$p_1 \geq p_2 \geq \cdots \geq p_{j-1} \geq p_j \geq p_{j+1} \geq \cdots \geq p_m \geq q_0 \geq q_1.$$

Dann ist

$$C' = \{c_1, c_2, \ldots, c_{j-1}, c_{j+1}, \ldots, c_m, c_j 0, c_j 1\}$$

ein optimaler Präfixcode für die Verteilung

$$P' = \{p_1, p_2, \ldots, p_{j-1}, p_{j+1}, \ldots, p_m, q_0, q_1\}.$$

Beweis. Da C ein Präfixcode ist, bilden auch die Elemente von C' einen Präfixcode. Außerdem gilt

$$\begin{aligned}
\mathcal{L}(C', P') &= \sum_{i=1}^{j-1} p_i|c_i| + \sum_{i=j+1}^{m} p_i|c_i| + q_0|c_j 0| + q_1|c_j 1| \\
&= \sum_{i=1}^{j-1} p_i|c_i| + \sum_{i=j+1}^{m} p_i|c_i| + q_0(|c_j| + 1) + q_1(|c_j| + 1) \\
&= \sum_{i=1}^{j-1} p_i|c_i| + \sum_{i=j+1}^{m} p_i|c_i| + (q_0 + q_1)(|c_j| + 1) \\
&= \sum_{i=1}^{j-1} p_i|c_i| + \sum_{i=j+1}^{m} p_i|c_i| + p_j|c_j| + p_j \\
&= \sum_{i=1}^{m} p_i|c_i| + p_j \\
&= \mathcal{L}(C, P) + p_j.
\end{aligned}$$

Wir haben zu zeigen, dass C' optimal für P' ist.

Angenommen,

$$D' = \{d_1, d_2, \ldots, d_{j-1}, d_{j+1}, \ldots, d_{m-1}, d_m, d_{m+1}, d_{m+2}\}$$

ist optimal für die Verteilung P'. Ohne Beschränkung der Allgemeinheit können wir annehmen, dass D' ein Präfixcode ist. Es sei l die maximale Länge eines Wortes aus D'. Wenn es genau ein Wort w der Länge l in D' gibt, so ist auch

$$D'' = (D \setminus \{w\}) \cup \{v\},$$

wobei v aus w durch Streichen des letzten Buchstaben entsteht, ein Präfixcode. Offenbar gilt $\mathcal{L}(D'', P') < \mathcal{L}(D', P)$, da die Wörter von D'' höchstens die gleiche Länge und in einem Fall eine kleinere Länge haben. Dies widerspricht aber der Annahme, dass D' optimal für P' ist.

Daher enthält D' mindestens zwei Wörter w_1 und w_2 der Länge l. Falls keine zwei Codewörter der Länge l ein gemeinsames Präfix der Länge $l-1$ haben, so definieren wir v_1 und v_2 als die Präfixe der Länge $l-1$ von w_1 bzw. w_2. Dann ist

$$D''' = (D' \setminus \{w_1, w_2\}) \cup \{v_1, v_2\}$$

erneut ein Präfixcode mit geringeren Kosten als D'.

Folglich gibt es ein Wort w so, dass $w0$ und $w1$ die Wörter der Länge l in D' sind. Wir können annehmen, dass $w0$ und $w1$ den minimalen Wahrscheinlichkeiten q_0 und q_1 zugeordnet sind, da sonst durch Umordnung der Codewörter ein Code erreicht werden kann, dessen Kosten nicht größer sind. Also gelten $d_{m+1} = w0$ und $d_{m+2} = w1$.

Wir betrachten nun den Code

$$D = \{d_1, d_2, \ldots, d_{j-1}, w, d_{j+1}, d_{j+2}, \ldots d_{m-1}, d_m\}.$$

Analog zur Rechnung zu Beginn des Beweises erhalten wir

$$\mathcal{L}(D', P') = \mathcal{L}(D, P) + p_j$$

und damit wegen der vorausgesetzten Optimalität von C für P

$$\mathcal{L}(C', P') = \mathcal{L}(C, P) + p_j \leq \mathcal{L}(D, P) + p_j = \mathcal{L}(D', P').$$

Da D' optimal für P' ist, muss daher auch C' optimal für P' sein. □

Aus Satz 2.3 folgt die folgende Methode zur Erzeugung eines optimalen Codes über $\{0, 1\}$ für die Verteilung

$$P = \{p_1, p_2, \ldots, p_m\}.$$

1. Wir setzen $P_1 = P$.
2. Für $2 \leq i \leq m-1$ konstruieren wir die Verteilung P_i wie folgt: Ist

   $$P_{i-1} = \{r_1, r_2, \ldots, r_{m-i}, r_{m-i+1}\}$$

   mit

   $$r_1 \geq r_2 \geq \cdots \geq r_{m-i} \geq r_{m-i+1},$$

   so setzen wir

   $$P_i = \{r_1, r_2, \ldots, r_{m-i-1}, r_{m-i} + r_{m-i+1}\}.$$

3. Für $P_{m-1} = \{t_1, t_2\}$, $t_1 \geq t_2$, setzen wir $C_{m-1} = \{0, 1\}$. (Dies ist der optimale Code für eine zweielementige Verteilung.)
4. Für $1 \leq j \leq m-2$ konstruieren wir den optimalen Code C_i für P_i aus dem optimalen Code C_{i+1} für P_{i+1} entsprechend dem Verfahren aus Satz 2.3.
5. Wir setzen $C = C_1$, welches der optimale Code für $P = P_1$ ist.

Beispiel 2.1. (Fortsetzung) Wir illustrieren die Methode anhand unseres Beispiels mit der Verteilung

$$P = \{0.20,\ 0.20,\ 0.19,\ 0.12,\ 0.11,\ 0.09,\ 0.09\}\ .$$

Die nachstehenden Tabellen geben die Konstruktion der Verteilungen P_i, $1 \leq i \leq m-1$, wobei wir stets die Wahrscheinlichkeiten von oben nach unten der Größe nach ordnen (da dies bei Satz 2.3 gefordert ist) und der Codes C_i, $1 \leq i \leq m-1$.

P_1	P_2	P_3	P_4	P_5	P_6
0.20	0.20	$\rightarrow$ 0.23	$\rightarrow$ 0.37	$\rightarrow$ 0.40	$\rightarrow$ 0.60
0.20	0.20	0.20	0.23	0.37 —	0.40
0.19	0.19	0.20	0.20 —	0.23 —	
0.12	$\rightarrow$ 0.18	0.19 —	0.20 —		
0.11	0.12 —	0.18 —			
0.09 —	0.11 —				
0.09 —					

C_6	C_5	C_4	C_3	C_2	C_1
0 —	1 —	00 —	01 —	10	10
1	$\rightarrow$ 00	01	10	11	11
	$\rightarrow$ 01	$\rightarrow$ 10	11	000	000
		$\rightarrow$ 11	$\rightarrow$ 000	001 —	010
			$\rightarrow$ 001	$\rightarrow$ 010	011
				$\rightarrow$ 011	$\rightarrow$ 0010
					$\rightarrow$ 0010

Für den so gewonnen Code

$$C_H = \{\ 10,\ 11,\ 000,\ 010,\ 011,\ 0010,\ 0011\ \}$$

ergeben sich die Kosten

$$\mathcal{L}(C_H, P) = (0.20 + 0.20) \cdot 2 + (0.19 + 0.12 + 0.11) \cdot 3 + (0.09 + 0.09) \cdot 4 = 2.78.$$

Da C_H optimal ist, gilt $\mathcal{L}_{\{0,1\}}(P) = 2.78$, womit auch gezeigt ist, dass die oben nach den Methoden von SHANNON bzw. FANO gewonnenen Codes C'_S und C''_S bzw. C_F relativ kostengünstig für P sind.

Bei allen bisher betrachteten Konstruktionen von kostengünstigen Codes geht man davon aus, oft auftretenden Buchstaben relativ kurze Codewörter und seltener auftretenden Buchstaben längere Wörter zuzuordnen. Bei gewissen Aufgaben sind aber Blockcodes besonders günstig. Sind dabei k Symbole über $\{0, 1\}$ zu codieren, so wird mit Codewörter der Länge $\lceil log_2(k) \rceil$ eine kostenoptimale Variante erreicht. Für Texte der deutschen Sprache (ohne Berücksichtigung der Großschreibung) sind die 26 Buchstaben (ohne ä,ö,ü und ß), ein Leerzeichen _ (zur Trennung der Wörter) und die Satzzeichen Punkt, Ausrufezeichen, Fragezeichen, Semikolon, Doppelpunkt, Gedankenstrich, Apostroph und zwei Arten Anführungsstriche zu codieren. Mit $\mathcal{S}$ bezeichnen wir die Menge der Satzzeichen. Damit sind mehr als 32 und weniger als 64 Symbole zu codieren. Daher sind Codewörter der Länge 6 erforderlich.

Es ist aber offensichtlich, dass die Satzzeichen relativ selten im Vergleich zu den Buchstaben und dem Leerzeichen auftreten. In der deutschen Sprache steht im Mittel nach 30 Buchstaben bzw. Leerzeichen ein Satzzeichen. Deshalb kann man mit dem folgenden Trick eine Codierung verwenden, bei der nur Codewörter der Länge 5 benutzt werden. Wir betrachten Codierungen

$$\varphi : \{a, b, c, \ldots, x, y, z, \_\} \rightarrow \{0, 1\}^5 \text{ und } \psi : \mathcal{S} \rightarrow \{0, 1\}^5.$$

Da sowohl das eigentliche Alphabet als auch die Menge der Satzzeichen keine 32 Symbole enthalten, nutzen wir bei den Codierungen φ und ψ nicht das Wort 11111 und verwenden dieses als Markierung, die den Wechsel von φ zu ψ und umgekehrt bezeichnet. Mit

$$\varphi(a) = 00001, \varphi(i) = 01001, \varphi(m) = 01101, \psi(.) = 01001$$

wird durch

01101 00001 01101 00001 01101 01001 00001 11111 01001 11111 01001 00001

die Folge

$m\ a\ m\ a\ m\ i\ a\ .\ i\ a$

codiert. Geht man davon aus dass durchschnittlich auf 30 Buchstaben und Leerzeichen ein Satzzeichen folgt, so benötigt man zur Codierung von diesen 30 Buchstaben und dem Satzzeichen 165 Elemente aus $\{0, 1\}$, da zusätzlich noch zweimal die Markierung 11111 auftritt. Dies entspricht nur durchschnittlich 5,32 Ziffern 0 bzw. 1 je codiertem Symbol. Daher ist diese Art der Codierung (mit Wechsel der Codes) kostengünstiger als die mit einer Folge aus 6 Ziffern bei Verwendung einer direkten Codierung der Gesamtmenge $\{a, b, \ldots, y, z\} \cup \mathcal{S}$.

Kapitel 3

Fehlerkorrigierende Codes

3.1 Fehlertypen und Fehlerkorrektur

Beim Übertragen codierter Nachrichten können aufgrund technischer Defekte oder Störungen im Übertragungskanal Fehler auftreten. Daher sind in der Codierungstheorie die beiden folgenden Probleme zu lösen.

- Ein Code ist so zu konstruieren, dass der Empfänger einer Nachricht in der Lage ist, beim Decodieren gewisse Übertragungsfehler zu bemerken. Codes dieser Art heißen fehlererkennend.
- Ein Code ist so zu konstruieren, dass der Empfänger einer Nachricht in der Lage ist, Korrekturen so vorzunehmen, dass eine korrekte Decodierung möglich ist. Codes mit dieser Eigenschaft heißen fehlerkorrigierend.

Bei den bisher betrachteten Codierungen von Objekten über (endlichen) Alphabeten ist die Erkennung von Übertragungsfehlern mittels der im Beweis von Satz 1.4 gegebenen Decodierung gesichert. Der decodierende Automat gibt uns eine Fehlermeldung, falls dass empfangene Wort nicht in C^+ liegt.

Schwieriger gestaltet sich die Fehlererkennung, falls die zugrundeliegende Menge von zu codierenden Objekten potentiell unendlich und nicht als Menge von Wörtern angesehen wird. Ein solches Beispiel ist die zur Identifikation von Büchern verwendete ISBN-Kennzeichnung (Internationale Standard-Buch-Nummer). Eine ISBN besteht aus fünf Teilen:

– einem Präfix 978 oder 979 (kennzeichnet den Artikel als Buch, aus Kompatibilitätsgrunden zum EAN/GTIN-System),
– einer Kennzahl für eine nationale, geographische, sprachliche oder sonstige geeignete Gruppe,
– einer Ziffernfolge, die den Verlag widerspiegelt,
– einer Ziffernfolge (Titelnummer), die vom Verlag vergeben wird, und
– einer Prüfziffer.

Die Gesamtlänge einer ISBN ist 13. Die einzelnen Teile dürfen durch Bindestriche verbunden werden; aber auch ohne Bindestriche sind die Nummern eindeutig. Die Prüfziffer wird aus den ersten 12 Ziffern $z_1, z_2, \ldots, z_{12}$ wie folgt berechnet. Es sei

$$s = z_1 + z_3 + z_5 + z_7 + z_9 + z_{11} + 3(z_2 + z_4 + z_6 + z_8 + z_{10} + z_{12})$$

eine gewichtete Quersumme, wobei jede Ziffer an einer ungeraden Stelle mit 1 und jede Ziffer an einer geraden Stelle mit 3 gewichtet wird. Die Prüfziffer z_{13} gibt den kleinsten Abstand einer durch 10 teilbaren Zahl, die mindestens so groß wie s ist, von s an:

$$z_{13} = (10 - (s \bmod 10)) \bmod 10.$$

Daraus ergibt sich, dass die wie oben gewichtete Quersumme $s + z_{13}$ einer ISBN stets durch 10 teilbar ist.

Beispielsweise ist die ISBN-Kennzeichnung des Buches [19] von Peter Sweeney zur Fehlererkennung und -korrektur 978-3-446-16439-0, wobei 3 für Deutschland, 446 für den Hanser-Verlag und 16439 für das angegebene Buch stehen, während die in den USA (entspricht 0) bei Prentice Hall International (entspricht 13) erschienene Originalausgabe durch 978-0-13-278706-2 gekennzeichnet ist (genau genommen hat die Ausgabe die Nummer 0-13-278706-7 nach dem im Jahre 2007 abgeschafften ISBN-10-System, aber die Nummer im ISBN-13-System entsteht aus alten Nummern durch Voranstellen von 978 und Anpassen der Prüfziffer).

Wird nun eine ISBN übermittelt, so kann der Empfänger anhand der Folgenlänge und der gewichteten Quersumme überprüfen, ob die empfangene Ziffernfolge einer ISBN entspricht. Stellt der Empfänger die Folgenlänge 13 (Bindestriche sind zu ignorieren) und eine durch 10 teilbare gewichtete Quersumme fest, so ist die Folge – mit hoher Wahrscheinlichkeit – korrekt übermittelt worden. Ist dagegen beispielsweise die Folge zu kurz, so ist beim Übertragen mindestens ein Symbol verlorengegangen. Ist die gewichtete Quersumme nicht durch 10 teilbar, so ist mindestens ein Symbol nicht korrekt übertragen worden.

Durch die Einführung des Prüfsymbols sind wir zwar in der Lage zu erkennen, ob das übermittelte Wort eine ISBN-Kennzeichnung sein kann, aber wir können bei Auftreten eines Übertragungsfehlers nicht ermitteln, welcher Fehler vorliegt. Beispielsweise entsteht die Folge 978-3-446-16449-0 aus der oben gegebenen ISBN von [19], indem als z_{11} nicht 3 sondern 4 übermittelt wurde. Diese Folge stellt keine ISBN dar, da die Prüfziffer 9 sein müsste. Um die gesendete Folge zu erhalten, müssen wir z_{11} um 1 verringern. Verringern wir aber z_9 um 1, erhalten wir die Folge 978-3-446-15449-0, deren gewichtete Quersumme auch durch 10 teilbar ist und daher die gesendete ISBN sein könnte. Somit ist nicht klar, wie die empfangene Folge richtig zu decodieren ist.

Als Zweites betrachten wir den Blockcode

$$C = \{11000, 10110, 01101\}$$

und nehmen an, dass beim Übertragen höchstens ein Fehler auftritt, wobei außerdem gilt, dass anstelle einer 1 eine 0 oder anstelle einer 0 eine 1 übermittelt wird. Die Länge des Wortes wird daher nicht verändert und durch stückweises Vergleichen mit den Codewörtern (siehe die Ausführungen vor von Satz 1.4) ist leicht feststellbar, ob die Übertragung korrekt erfolgte.

Wir wollen nun annehmen, dass beim Übertragen des Codewortes 11000 ein Fehler an der dritten Stelle eingetreten ist, also 11100 empfangen wurde. Es ist nun leicht zu sehen, dass durch Ändern der anderen Codewörter von C an *einer* beliebigen Stelle immer ein Wort entsteht, das von 11100 verschieden ist. Gehen wir davon aus, dass beim Übertragen maximal ein Fehler gemacht wurde, so ist klar, dass das Wort 11000 gesendet worden ist. Hieraus folgt, dass mit der Wahl von C der Empfänger die Möglichkeit hat, diesen Fehler

zu korrigieren. Wir werden unten nachweisen, dass C ein Code mit der Möglichkeit zur Korrektur eines jeden Fehlers der eben behandelten Art ist.

In diesem Abschnitt wollen wir unsere Betrachtungen auf Blockcodes über dem Alphabet $\{0,1\}$ und auf die Korrektur von Fehlern bei der Übertragung von Codewörtern beschränken (bei den hauptsächlich betrachteten Fehlern lässt sich die Methode zur Korrektur auch auf beliebige Nachrichten übertragen, da die Länge des empfangenen Codewortes durch den Blockcode vorgegeben ist; ansonsten verwenden wir ein Trennzeichen zur Trennung der Codewörter).

Wir betrachten die folgenden Typen von Fehlern.

Definition 3.1 *Unter einem Austauschfehler verstehen wir die Übertragung einer* 0 *anstelle einer* 1 *oder die Übertragung einer* 1 *anstelle einer* 0.

Unter einem Ausfallfehler verstehen wir den Ausfall eines Symbols während der Übertragung; das übertragene Wort wird durch Löschen eines Buchstabens gekürzt.

Unter einem Einschubfehler verstehen wir die Übertragung eines zusätzlichen Symbols; das empfangene Wort wird durch den Einschub eines Symbols an einer Stelle im übertragenen Wort verlängert.

Wir bezeichnen diese Typen von Fehlern durch

$$1 \to 0,\ 0 \to 1,\ 0 \to \lambda,\ 1 \to \lambda,\ \lambda \to 0,\ \lambda \to 1.$$

Es sei G die Menge aller dieser Fehler.

Definition 3.2 *Eine Teilmenge von G bezeichnen wir als Fehlertyp. Ein Fehlertyp F heißt symmetrisch, falls F durch Vereinigung aus den folgenden Mengen $\{0 \to 1, 1 \to 0\}$, $\{\lambda \to 0, 0 \to \lambda\}$ und $\{\lambda \to 1, 1 \to \lambda\}$ gewonnen werden kann.*

Bei einem symmetrischen Fehlertyp enthält F mit einem Fehler f auch den Fehler g, durch den das ursprüngliche Wort wieder gewonnen werden kann.

Für einen Fehlertyp F sowie Wörter w und v über dem Alphabet $\{0,1\}$ setzen wir

$$w \overset{F,t}{\Longrightarrow} v,$$

falls bei der Übertragung durch das simultane Auftreten von t Fehlern aus F aus dem Wort w das Wort v entsteht. Offenbar gilt $w \overset{F,t}{\Longrightarrow} v$ genau dann, wenn es Wörter $w_1, w_2, \ldots, w_{t-1}$ so gibt, dass

$$w = w_0 \overset{F,1}{\Longrightarrow} w_1 \overset{F,1}{\Longrightarrow} w_2 \overset{F,1}{\Longrightarrow} \cdots \overset{F,1}{\Longrightarrow} w_{t-1} \overset{F,1}{\Longrightarrow} w_t = v \tag{3.1}$$

gilt.

Beispiel 3.1 Es sei

$$F = \{0 \to 1, 1 \to 0, \lambda \to 1, 0 \to \lambda\}.$$

Dann gelten

- $0011 \overset{F,1}{\Longrightarrow} 0111$
  (Auftreten des Fehlers $0 \to 1$ an zweiter Stelle),

- $0011 \overset{F,2}{\Longrightarrow} 1001$
  (bei Auftreten des Fehlers $0 \to 1$ an erster und $1 \to 0$ an dritter Stelle oder durch Ausfall einer 1 und Einschub einer 1 an erster Stelle),

- $0011 \overset{F,1}{\Longrightarrow} 00111$
  (bei Auftreten des Fehlers $\lambda \to 1$ zwischen zweiter und dritter Stelle),

- $0011 \overset{F,3}{\Longrightarrow} 10$
  (bei Ausfall des ersten und zweiten Buchstaben entsprechend Fehler $0 \to \lambda$ und Austausch des Symbols an vierter Stelle entsprechend Fehler $1 \to 0$).

Es gilt jedoch nicht $0 \overset{F,1}{\Longrightarrow} 111$. Mehr noch, es gibt kein Wort v mit einer Länge größer als zwei, für das $0 \overset{F,1}{\Longrightarrow} v$ gilt. $\Diamond$

Wir führen nun den zentralen Begriff dieses Abschnitts ein.

Definition 3.3 *Es sei F ein Fehlertyp und C ein Blockcode über dem Alphabet $\{0,1\}$. Dann heißt der Code C Code mit Korrektur von s Fehlern aus F, falls zu jedem Wort $v \in \{0,1\}^*$ höchstens ein Wort $w \in C$ mit $w \overset{F,t}{\Longrightarrow} v$ und $t \leq s$ existiert.*

Wir bemerken, dass bei einer Übertragung von Wörtern eines Codes C, bei der höchstens s Fehler auftreten, nur Wörter v empfangen werden können, für die $w \overset{F,t}{\Longrightarrow} v$ mit $w \in C$ und $t \leq s$ gilt. Ist C ein Code mit Korrektur von s Fehlern, so kann jedem empfangenen Wort v eindeutig ein Codewort w zugeordnet werden, das übertragen werden sollte.

Besteht F nur aus Austauschfehlern, so kann offensichtlich bei der Übertragung die Länge der Wörter nicht verändert werden. Daher gibt es zu Wörtern v, deren Länge von der des Blockcodes verschieden ist, kein Wort $w \in C$ mit $w \overset{F,t}{\Longrightarrow} v$. Somit ist gerechtfertigt, dass in der Definition eines fehlerkorrigierenden Codes die Existenz von *höchstens* einem $w \in C$ zu v gefordert wurde.

Die Definition fehlerkorrigierender Codes ist nicht effektiv in dem Sinn, dass aus ihr direkt ein Algorithmus folgt, mittels dessen es sich entscheiden lässt, ob ein Code s Fehler korrigieren kann. Wir streben nun ein solches Kriterium an. Dazu führen wir folgende Begriffe ein.

Für einen Fehlertyp F und Wörter $w, v \in \{0,1\}^*$ definieren wir

$$d_F(w,v) = \begin{cases} \min\{t \mid w \overset{F,t}{\Longrightarrow} v\}, & \text{falls dies existiert,} \\ \infty & \text{sonst.} \end{cases}$$

Offenbar gilt $d_F(w,v) = \infty$ genau dann, wenn w mittels Fehlern aus F nicht in v überführt werden kann.

Es sei

$$H = \{0 \to 1,\ 1 \to 0\}$$

der Fehlertyp, der aus den beiden Austauschfehlern besteht. Für H und zwei Wörter

$$w = x_1 x_2 \dots x_n \quad \text{und} \quad v = y_1 y_2 \dots y_m$$

ergibt sich

$$d_H(w,v) = \begin{cases} \#(\{i \mid x_i \neq y_i\}), & \text{falls } n = m, \\ \infty & \text{sonst.} \end{cases} \tag{3.2}$$

Die Funktion d_H wird Hamming-Abstand genannt.

Satz 3.1 *Für einen symmetrischen Fehlertyp F ist durch d_F eine Abstandsfunktion auf der Menge $\{0,1\}^*$ definiert.*

Beweis. Wir zeigen die drei Eigenschaften einer Abstandsfunktion:

1. Es gilt $d_F(w,v) = 0$ genau dann, wenn $w = v$ ist.

   Es gilt $d_F(w,v) = 0$ genau dann, wenn kein Fehler vorliegt. Dies ist offensichtlich gleichwertig zu $w = v$.

2. Es gilt $d_F(w,v) = d_F(v,w)$ für beliebige Wörter $w, v \in \{0,1\}^*$.

   Da F symmetrisch ist, folgt, dass entweder beide Werte unendlich oder beide Werte endlich sind.

   Für den Fall, dass beide unendlich sind, gilt die Behauptung offenbar.

   Es sei $d_F(w,v) = t$. Dann gilt

   $$w = w_0 \overset{F,1}{\Longrightarrow} w_1 \overset{F,1}{\Longrightarrow} w_2 \overset{F,1}{\Longrightarrow} \cdots \overset{F,1}{\Longrightarrow} w_{t-1} \overset{F,1}{\Longrightarrow} w_t = v$$

   für gewisse Wörter $w_1, w_2, \dots, w_{t-1}$. Da F ein symmetrischer Fehlertyp ist, erhalten wir auch die Beziehung

   $$v = w_t \overset{F,1}{\Longrightarrow} w_{t-1} \overset{F,1}{\Longrightarrow} w_{t-2} \overset{F,1}{\Longrightarrow} \cdots \overset{F,1}{\Longrightarrow} w_1 \overset{F,1}{\Longrightarrow} w_0 = w,$$

   woraus nach Definition

   $$d_F(v,w) \leq t = d_F(w,v)$$

   folgt. Analog kann man auch $d_F(w,v) \leq d_F(v,w)$ zeigen, womit die Gleichheit folgt.

3. Es gilt $d_F(w,v) + d_F(v,z) \geq d_F(w,z)$ für alle Wörter $w, v, z \in \{0,1\}^*$.

   Ist einer der Werte $d_F(w,v)$ oder $d_F(v,z)$ unendlich, so ist die Behauptung offenbar gültig.

   Es seien daher $d_F(w,v) = t$ und $d_F(v,z) = s$. Dann gibt es Wörter $w_1, w_2, \dots, w_{t-1}$, $v_1, v_2, \dots, v_{s-1}$ mit

   $$w \overset{F,1}{\Longrightarrow} w_1 \overset{F,1}{\Longrightarrow} w_2 \overset{F,1}{\Longrightarrow} \cdots \overset{F,1}{\Longrightarrow} w_{t-1} \overset{F,1}{\Longrightarrow} v$$

und

$$v \overset{F,1}{\Longrightarrow} v_1 \overset{F,1}{\Longrightarrow} v_2 \overset{F,1}{\Longrightarrow} \cdots \overset{F,1}{\Longrightarrow} v_{s-1} \overset{F,1}{\Longrightarrow} z.$$

Somit erhalten wir

$$w \overset{F,1}{\Longrightarrow} w_1 \overset{F,1}{\Longrightarrow} w_2 \overset{F,1}{\Longrightarrow} \cdots \overset{F,1}{\Longrightarrow} w_{t-1} \overset{F,1}{\Longrightarrow} v \overset{F,1}{\Longrightarrow} v_1 \overset{F,1}{\Longrightarrow} v_2 \overset{F,1}{\Longrightarrow} \cdots \overset{F,1}{\Longrightarrow} v_{s-1} \overset{F,1}{\Longrightarrow} z,$$

woraus

$$d_F(w,z) \leq t + s = d_F(w,v) + d_F(v,z)$$

resultiert. □

Definition 3.4 *Für einen symmetrischen Fehlertyp F und einen endlichen Code C definieren wir den Codeabstand $d_F(C)$ als*

$$d_F(C) = \min\{d_F(x,y) \mid x,y \in C, x \neq y\}.$$

Wir charakterisieren nun die Fähigkeit, s Fehler zu korrigieren, durch den Codeabstand.

Satz 3.2 *Es sei F ein symmetrischer Fehlertyp. Dann ist ein endlicher Code C genau dann ein Code mit Korrektur von s Fehlern aus F, wenn*

$$d_F(C) \geq 2s + 1$$

gilt.

Beweis. Es sei C ein Code mit $d_F(C) \leq 2s$. Dann gibt es eine natürliche Zahl $t \leq 2s$, Codewörter $x, y \in C$ mit $x \neq y$ und Wörter $w_1, w_2, \ldots, w_{t-1}$ so, dass

$$x \overset{F,1}{\Longrightarrow} w_1 \overset{F,1}{\Longrightarrow} w_2 \overset{F,1}{\Longrightarrow} \cdots \overset{F,1}{\Longrightarrow} w_{t-1} \overset{F,1}{\Longrightarrow} y$$

gilt. Für $w = w_{\lceil t/2 \rceil}$ erhalten wir

$$d_F(x,w) = r_1 \leq s \quad \text{und} \quad d_F(y,w) = r_2 \leq s$$

und damit

$$x \overset{F,r_1}{\Longrightarrow} w \quad \text{und} \quad y \overset{F,r_2}{\Longrightarrow} w \tag{3.3}$$

mit $r_1 \leq s$ und $r_2 \leq s$, womit C kein Code mit Korrektur von s Fehlern aus F sein kann.

Ist umgekehrt C kein Code mit Korrektur von s Fehlern aus F, so gibt es ein Wort $w \in \{0,1\}^*$, Codewörter $x, y \in C$ mit $x \neq y$ und Zahlen $r_1 \leq s$ und $r_2 \leq s$ derart, dass (3.3) erfüllt ist. Deshalb gilt

$$d_F(x,y) \leq d_F(x,w) + d_F(w,y) \leq r_1 + r_2 \leq 2s,$$

womit

$$d_F(C) \leq 2s$$

nachgewiesen ist. □

Beispiel 3.2 Für den oben betrachteten Blockcode

$$C = \{11000, 10110, 01101\}$$

und den Fehlertyp H, der aus den beiden Austauschfehlern $0 \to 1$ und $1 \to 0$ besteht, ergibt sich wegen (3.2) für den Hamming-Abstand $d_H(w, v) = 3$ für je zwei Wörter $w, v \in C$ und somit

$$d_H(C) = 3.$$

Daher ist C ein Code mit Korrektur eines Austauschfehlers. ◊

3.2 Beispiele für fehlerkorrigierende Codes

Mittels des Satzes 3.2 kann zu einem gegebenen Code die Anzahl s der korrigierbaren symmetrischen Fehler bestimmt werden. Der Satz liefert aber keine Möglichkeit, den oder die Fehler zu korrigieren. Hierfür steht uns entsprechend den Definitionen bisher nur der folgende Algorithmus zur Verfügung. Wir bilden den Abstand zwischen dem empfangenen Wort und den Codewörtern und wissen dann, dass das Codewort gesendet wurde, bei dem dieser Abstand höchstens s beträgt. Wir wollen nun einige spezielle Codes betrachten, bei denen die Stellen, an denen der Fehler aufgetreten ist, direkt ermittelt werden können.

a) HAMMING-*Codes zur Korrektur eines Austauschfehlers*

Es sei n eine beliebige positive natürliche Zahl. Wir setzen $l = \lfloor log_2(n) \rfloor + 1$. Damit gilt $2^{l-1} \leq n < 2^l$. Zu einer Zahl i mit $1 \leq i \leq n$ sei $u_{i,1}u_{i,2}\dots u_{i,l}$ die Binärdarstellung von i. Mit $e_l(i)$ bezeichnen den Vektor $(u_{i,1}, u_{i,2}, \dots, u_{i,l}) \in \{0,1\}^l$ für $1 \leq i \leq n$. Für ein beliebiges Wort $X = x_1x_2\dots x_n \in \{0,1\}^n$ setzen wir

$$H(X) = \sum_{i=1}^{n} x_i \cdot e_l(i)\,,$$

wobei die Addition der Vektoren komponentenweise modulo 2 erfolgt. Damit ist $H(X)$ also ein Vektor aus $\{0,1\}^l$. Wir definieren nun den HAMMING-Code H_n durch

$$H_n = \{X \mid X \in \{0,1\}^n, H(X) = (0,0,\dots,0)\}\,.$$

Nach dieser Definition können die Elemente von H_n als Lösungen eines homogenen Gleichungssystems mit n Unbekannten und l Gleichungen aufgefasst werden. Beachten wir noch, dass die Vektoren $e_l(2^i)$, $0 \leq i \leq l-1$, jeweils genau eine Eins enthalten und paarweise verschieden sind, so ist klar, dass die von ihnen gebildete Matrix die Einheitsmatrix $E_{l,l}$ ist und dass damit das Gleichungssystem den Rang l besitzt. Damit ergibt sich, dass die Elemente des HAMMING-Codes einen $(n-l)$-dimensionalen Vektorraum über dem Körper $\{0,1\}$ bilden. Folglich hat H_n genau 2^{n-l} Elemente. Wegen $l = \lfloor log_2(n) \rfloor + 1$ erhalten wir

$$\frac{2^{n-1}}{n} = \frac{2^{n-1}}{2^{\log_2(n)}} = 2^{n-1-log_2(n)} \leq 2^{n-l} \leq 2^{n-log_2(n+1)} = \frac{2^n}{2^{\log_2(n+1)}} = \frac{2^n}{n+1}$$

und damit

$$\frac{2^{n-1}}{n} \leq \#(H_n) = 2^{n-(\lfloor log_2(n) \rfloor + 1)} \leq \frac{2^n}{n+1}\,.$$

Wir geben nun eine Methode zur Berechnung der Elemente von H_n an. Wir setzen

$$M = \{1, 2, \ldots, n\} \setminus \{2^i \mid 0 \leq i \leq l-1\}\,.$$

Dann können wir den HAMMING-Code dadurch bestimmen, dass wir die Werte x_j mit $j \in M$, beliebig in $\{0,1\}$ wählen, und dann die Werte x_{2^i}, $0 \leq i \leq l-1$, entsprechend dem Gleichungssystem berechnen. Jedoch ist das Lösen des Gleichungssystems sehr einfach, denn wegen

$$(0, 0, \ldots, 0) = \sum_{k=1}^{n} x_k e_k(l) = \sum_{i=0}^{l-1} x_{2^i} e_l(2^i) + \sum_{j \in M} x_j e_l(j)$$

ergibt sich

$$\sum_{i=0}^{l-1} x_{2^i} e_l(2^i) = \sum_{j \in M} x_j e_l(j)\,.$$

Setzen wir

$$(u_l, u_{l-1}, \ldots, u_1) = \sum_{i \in M} x_i e_l(i)\,.$$

und beachten, dass die Vektoren $e_l(2^i)$ die Einheitsmatrix bilden, so muss

$$x_{2^i} = u_{i+1}$$

gelten.

Beispiel 3.3 Es sei $n = 6$. Dann ergibt sich $l = 3$. Der HAMMING-Code H_6 besteht also aus $2^{n-l} = 2^3 = 8$ Elementen. Um ein Wort $X = x_1x_2x_3x_4x_5x_6$ aus H_6 zu berechnen, können wir die Werte x_3, x_5 und x_6 frei wählen und bestimmen dann x_1, x_2 und x_4. Es ergibt sich die folgende Tabelle

x_3	x_5	x_6	$x_3e_3(3) + x_5e_3(5) + x_6e_3(6)$	x_1	x_2	x_4	X
0	0	0	$(0,0,0)$	0	0	0	000000
0	0	1	$(1,1,0)$	0	1	1	010101
0	1	0	$(1,0,1)$	1	0	1	100110
0	1	1	$(0,1,1)$	1	1	0	110011
1	0	0	$(0,1,1)$	1	1	0	111000
1	0	1	$(1,0,1)$	1	0	1	101101
1	1	0	$(1,1,0)$	0	1	1	011110
1	1	1	$(0,0,0)$	0	0	0	001011

und damit

$$H_6 = \{000000, 010101, 100110, 110011, 111000, 101101, 011110, 001011\}$$

als HAMMING-Code. ◊

Es sei bei der Übertragung von $X = x_1x_2\dots x_n$, $x_i \in \{0,1\}$ für $1 \le i \le n$ ein Austauschfehler – sagen wir an der j-ten Stelle – aufgetreten. Dann ist das von uns empfangene Wort $Y = x_1x_2\dots x_{j-1}(x_j \oplus 1)x_{j+1}x_{j+2}\dots x_n$. Weiterhin gilt

$$H(Y) = \sum_{i=1}^{j-1} x_i e_l(i) \oplus (x_j \oplus 1)e_l(j) \oplus \sum_{i=j+1}^{n} x_i e_l(i) = e_l(j) \oplus \sum_{i=1}^{n} x_i e_i(l) = e_l(j) \oplus H(X)\,.$$

Wenn wir nun annehmen, dass $X \in H_n$ gilt, so ist $H(X)$ der l-dimensionale Nullvektor, und es ergibt sich

$$H(Y) = e_l(j)\,.$$

Interpretieren wir $e_l(j)$ als eine Dualzahl, so erhalten wir nach Definition j und damit die Stelle, an der der Austauschfehler vorliegt.

Beispiel 3.3 (Fortsetzung) Es sei $Y = 010111$ das empfangene Wort. Wegen $Y \notin H_6$, kann nicht Y gesendet worden sein. Wir nehmen nun an, dass Y durch einen Austauschfehler während der Übertragung entstanden ist. Wegen

$$H(Y) = (0,1,0) \oplus (1,0,0) \oplus (1,0,1) \oplus (1,1,0) = (1,0,1)$$

muss der Fehler an der fünften Stelle vorliegen, da 101 die Dualdarstellung von 5 ist. Es muss also $X = 010101$ gesendet worden sein.

Wir können natürlich keine Aussage über das gesendete Wort treffen, falls sogar zwei (oder mehr) Fehler zugelassen sind. Das Wort Y kann dann sowohl durch einen Fehler an der fünften Stelle aus $010101 \in H_6$ als auch durch Fehler an der dritten und sechsten Stelle aus $011110 \in H_6$ entstanden sein.

b) Codes zur Korrektur eines Fehlers vom Typ $\{0 \to 1\}$

Es seien n und k zwei beliebige positive natürliche Zahlen. Für ein Wort $X = x_1x_2\dots x_n$ aus $\{0,1\}^n$ setzen wir

$$W(X) = \sum_{i=1}^{n} x_i \cdot i = x_1 + 2x_2 + 3x_3 + \dots + nx_n\,.$$

Nach Definition ist $W(X)$ die Summe der Indizes, bei denen der Buchstabe im Wort eine 1 ist. Ferner setzen wir

$$W_{n,k} = \{X \mid X \in \{0,1\}^n,\ W(X) = 0 \bmod k\}.$$

Beispiel 3.4 Es sei $n = 6$. Wir betrachten die Wörter

$$X = 110100, \quad Y = 010101, \quad Z = 010010, \quad Z' = 110011\,.$$

Dann gelten wegen

$$W(X) = 1 + 2 + 4 = 7,\ W(Y) = 2 + 4 + 6 = 12,$$
$$W(Z) = 2 + 5 = 7,\ W(Z') = 1 + 2 + 5 + 6 = 14$$

die Beziehungen $X \in W_{6,7}$, $Y \notin W_{6,7}$, $Z \in W_{6,7}$, $Z' \in W_{6,7}$ und $X \notin W_{6,12}$, $Y \in W_{6,12}$, $Z \notin W_{6,12}$, $Z' \notin W_{6,12}$. $\Diamond$

Wir zeigen nun, dass jeder Code $W_{n,k}$ mit $k \geq n + 1$ ein Code mit Korrektur von einem Fehler vom Typ $\{0 \to 1\}$ ist. Es seien $X = x_1x_2 \ldots x_n$ ein Wort aus $W_{n,k}$ und j eine Zahl mit $1 \leq j \leq n$. Außerdem entstehe ein Wort Y aus X durch einen Fehler vom Typ $\{0 \to 1\}$ an der Stelle j. Dann gelten $x_j = 0$ und $Y = x_1x_2 \ldots x_{j-1}1x_{j+1}x_{j+2} \ldots x_n$ und daher auch

$$W(Y) = W(X) + j\,.$$

Wegen $k \geq n + 1$ und $j \leq n$ folgt

$$W(Y) = W(X) + j = (0 + j) \bmod k = j \bmod k\,.$$

Somit gibt der Wert $W(Y)$ direkt die Stelle an, an der der Fehler aufgetreten ist.

Wir bemerken, dass $W_{n,k}$ für $k \geq n + 1$ nicht unbedingt auch ein Code mit Korrektur eines Fehlers vom Typ $\{1 \to 0\}$ ist. Dies ist wie folgt einzusehen: ersetzt man an der j-ten Stelle im Wort $X = x_1x_2 \ldots x_n$ den Buchstaben $x_j = 1$ durch eine Null, so ergibt sich $W(X) - W(Y) = -j$. Ist X ein Element des Codes $W_{n,k}$, so ergibt sich

$$W(X) - W(Y) = k - j \bmod k\,.$$

Wir betrachten nun den Fall $n = 6$ und $k = 7$ und das empfangene Wort $V = 110010$. Dann gilt $W(V) = 1 \bmod 7$. Entsprechend obigem bedeutet dies, dass dies durch die Änderung $0 \to 1$ an der ersten Stelle oder durch die Änderung $1 \to 0$ an der Stelle $k - 1 = 6$ entstanden sein kann. Im ersten Fall wurde $Z = 010010 \in W_{6,7}$ und im zweiten Fall $Z' = 110011 \in W_{6,7}$ (siehe Beispiel 3.4) gesendet.

Es sei nun $k \geq 2n$. Für $1 \leq l \leq n$ gilt dann $l \leq n < k - l$, Haben wir das Wort Y mit $W(Y) = j \bmod k$ empfangen, so liegt bei $j \leq n$ ein Fehler vom Typ $\{0 \to 1\}$ an der j-ten Stelle vor und bei $n < j = k - l$ liegt ein Fehler vom Typ $\{1 \to 0\}$ an der l-ten Stelle vor. Folglich ist $W_{n,k}$ für $k \geq 2n$ sogar ein Code mit Korrektur eines Austauschfehlers.

c) Codes zur Korrektur eines Ausfallfehlers

Wir wollen jetzt zeigen, dass die Codes $W_{n,k}$ mit $k \geq n + 1$ auch zur Korrektur eines Ausfallfehlers geeignet sind.

Es sei $X = x_1x_2 \ldots x_n$ ein Wort aus $W_{n,k}$ und entstehe Y aus X durch einen Ausfallfehler an der Stelle j. Dann gilt $Y = x_1x_2 \ldots x_{j-1}x_{j+1}x_{j+2} \ldots x_n$. Wir bezeichnen mit

n_1 bzw n_0 die Anzahl der Einsen bzw. Nullen, die rechts vom ausgefallenen Symbol x_j stehen, d. h. $n_i = \#_i(x_{j+1}x_{j+2}\dots x_n)$, $i \in \{0,1\}$. Offenbar gilt

$$n_0 + n_1 = n - j\,. \tag{3.4}$$

Wir betrachten zuerst den Fall $x_j = 0$. Da die Werte x_k, $j+1 \le k \le n$, in Y gegenüber X um eine Stelle nach vorn gerückt wurden, erhalten wir

$$W(Y) = \sum_{i=1}^{j-1} x_i \cdot i + \sum_{i=j+1}^{n} x_i(i-1)\,.$$

Aus $x_j = 0$ und der Definition von $W(X)$ folgt nun

$$W(X) - W(Y) = \sum_{i=j+1}^{n} x_i = n_1\,.$$

Offenbar gelten $\#_1(x_{j+1}x_{j+2}\dots x_n) = n_1 \le \#_1(X)$. Da eine Null ausgefallen ist, haben wir auch noch $\#_1(X) = \#_1(Y)$. Somit erhalten wir

$$W(X) - W(Y) = n_1 \le \#_1(Y)\,.$$

Es sei nun $x_j = 1$. Dann trägt x_j bei der Berechnung von $W(X)$ den Wert j bei, der bei Y entfällt. Daher ergibt sich

$$W(X) - W(Y) = j + n_1 = n - n_0\,,$$

wobei die letzte Beziehung aus (3.4) folgt. Weiterhin gilt $\#_1(X) \le j + n_1 = n - n_0$. Da eine 1 ausgefallen ist, folgt $\#_1(Y) = \#_1(X) - 1$ und damit

$$W(X) - W(Y) = n - n_0 > \#_1(Y)\,.$$

Wegen $W(X) = 0 \bmod k$ gilt $W(X) - W(Y) = -W(Y) \bmod k$. Damit können wir bei gegebenem Y den Wert $W(X) - W(Y)$ berechnen. Diesen vergleichen wir mit $\#_1(Y)$.

Gilt $W(X) - W(Y) \le \#_1(Y)$, so muss nach obigem eine Null ausgefallen sein, und wir fügen eine Null derart ein, dass hinter ihr noch $n_1 = W(X) - W(Y)$ Einsen stehen.

Gilt dagegen $W(X) - W(Y) > \#_1(Y)$, so muss nach obigem eine Eins ausgefallen sein, und wir fügen eine Eins derart ein, dass hinter ihr noch n_0 Nullen stehen, wobei sich n_0 aus $W(X) - W(Y) = n - n_0$ ergibt.

Beispiel 3.5 Wir betrachten den Code $W_{6,7}$, d. h. $n = 6$ und $k = 7$. Es sei das Wort $Y = 10100$ empfangen worden. Da Y nur die Länge 5 hat, muss ein Symbol bei der Übertragung ausgefallen sein. Wir berechnen nun zuerst $W(Y) = 1 + 3 = 4$. Damit ergibt sich $W(X) - W(Y) = 3$. Ferner haben wir $\#_1(Y) = 2$. Wegen $3 > 2$, muss also eine Eins ausgefallen sein. Ferner gilt $n_0 = n - (W(X) - W(Y)) = 6 - 3 = 3$. Daher muss die Eins so eingefügt werden, dass hinter ihr noch drei Nullen stehen. Damit ergibt sich $X = 110100$ (und nach Beispiel 3.4 liegt X in $W_{6,7}$). Dabei ist es egal, ob wir die zusätzlich Eins in Y vor oder hinter der Eins am Beginn von Y einfügen. ◊

Wir bemerken, dass mit analogen Betrachtungen gezeigt werden kann, dass $W_{n,k}$ für $k \geq n+1$ auch ein Code mit Korrektur eines Einschubfehlers ist.

d) Codes zur Korrektur eines Fehlers vom Typ $\{1 \to 0,\ 0 \to 1,\ 0 \to \lambda,\ 1 \to \lambda,\ \lambda \to 0,\ \lambda \to 1\}$

Wir betrachten den Code $W_{n,k}$ mit $k \geq 2n$. Für das empfangene Wort Y unterscheiden wir drei Fälle.

Fall 1. Y hat die Länge n. Gilt $W(Y) = 0 \bmod k$, so ist Y ein Element von $W_{n,k}$ und es ist kein Fehler bei der Übertragung aufgetreten. Ist dagegen $W(Y) \neq 0 \bmod k$, so ist ein Austauschfehler aufgetreten (da sich die Länge des Wortes bei der Übertragung nicht geändert hat). Daher sind wir in der Lage das gesendete Wort zu ermitteln, wenn genau ein Austauschfehler aufgetreten ist, wie am Ende des Abschnitts b) gezeigt wurde.

Fall 2. Y habe die Länge $n-1$. Dann muss ein Ausfallfehler aufgetreten sein, und wir können das gesendete Wort wie in Abschnitt c) gezeigt ermitteln.

Fall 3. Y habe die Länge $n+1$. Dann muss ein Einschubfehler aufgetreten sein, den wir korrigieren können, da nach der Bemerkung am Ende von Abschnitt c) $W_{n,k}$ ein Code mit Korrektur eines Einschubfehlers ist.

Ist also genau ein Fehler aufgetreten, so sind wir in der Lage zuerst festzustellen, ob es ein Austausch-, Ausfall- oder Einschubfehler ist und diesen dann zu korrigieren. Damit ist $W_{n,k}$ für $k \geq 2n$ ein Code mit Korrektur eines Fehlers vom Typ $\{1 \to 0,\ 0 \to 1,\ 0 \to \lambda,\ 1 \to \lambda,\ \lambda \to 0,\ \lambda \to 1\}$

3.3 Abschätzungen für fehlerkorrigierende Codes

In diesem Abschnitt betrachten wir nur Austauschfehler. Folglich gilt stets

$$F = \{0 \to 1,\ 1 \to 0\}.$$

Der Einfachheit halber verwenden wir deshalb zur Bezeichnung des Abstandes d anstelle von $d_{\{0\to1,1\to0\}}$.

Wir betrachten den HAMMING-Code H_7. Aus den Betrachtungen des vorigen Abschnitts wissen wir, dass H_7 wegen $l = \lfloor log_2(7) \rfloor + 1 = 3$ aus $2^{7-3} = 2^4 = 16$ Elementen besteht. Folglich sind wir bei Verwendung von H_7 nur in der Lage 16 Symbole durch Wörter der Länge 7 über $\{0,1\}$ zu codieren. Es erhebt sich die Frage, ob dies optimal ist, d. h. ob es einen Blockcode $C \subseteq \{0,1\}^7$ mit mindestens 17 Elementen gibt, der ebenfalls einen Austauschfehler korrigieren kann.

Wir wollen zeigen, dass dies nicht der Fall ist. Es sei deshalb $C \subseteq \{0,1\}^7$ ein Code mit Korrektur eines Austauschfehlers. Für ein Element $X \in C$ setzen wir

$$U_1(X) = \{Y \mid d(Y,X) \leq 1\}$$

und betrachten die Vereinigung V dieser Mengen, d. h.

$$V = \bigcup_{X \in C} U_1(X)\,.$$

Da C einen Austauschfehler korrigieren kann, gilt für den Codeabstand $d(C) = 3$ und folglich sind die Mengen $U_1(X)$ und $U_1(X')$ für $X, X' \in C$ disjunkt. Da jede der Mengen $U_1(X)$ genau 8 Elemente enthält, nämlich X selbst und die 7 Wörter, die durch Änderung von X an genau einer Stelle entstehen, gilt

$$\#(V) = \#(\bigcup_{X \in C} U_1(X)) = \#(C) \cdot 8 \leq 128\,,$$

da es genau $2^7 = 128$ verschiedene Wörter der Länge 7 gibt. Damit ergibt sich

$$\#(C) \leq \frac{128}{8} = 16\,.$$

Folglich ist 16 die maximale Zahl von Symbolen, die durch Codes in $\{0,1\}^7$ mit Korrektur von einem Austauschfehler codiert werden können.

In diesem Abschnitt wollen wir der Frage nach der maximalen Mächtigkeit von Blockcodes mit einer gegebenen Länge und mit einer gegebenen Anzahl von korrigierbaren Austauschfehlern nachgehen. Für natürliche Zahlen $n \geq 1$ und $d \geq 1$ setzen wir dazu

$$m(n,d) = \max\{\#(C) \mid C \subseteq \{0,1\}^n, d(C) \geq d\}\,.$$

Wegen Satz 3.2 gibt $m(n,d)$ die maximale Mächtigkeit von Codes aus Wörtern der Länge n und mit Korrektur von $\frac{d-1}{2}$ Austauschfehlern an.

Als erstes geben wir eine Abschätzung für $m(n,d)$ an, die dem am Beispiel demonstriertem Vorgehen entspricht.

Satz 3.3 *Für $n \geq 3$ und $s \geq 1$ gilt*

$$\frac{2^n}{\sum_{k=0}^{2s} \binom{n}{k}} \leq m(n, 2s+1) \leq \frac{2^n}{\sum_{k=0}^{s} \binom{n}{k}}\,.$$

Beweis. Es sei $C \subseteq \{0,1\}^n$ ein Blockcode mit Korrektur von s Austauschfehlern, d. h. mit Codeabstand $2s+1$ nach Satz 3.2, mit maximaler Mächtigkeit, d. h. $\#(C) = m(n, 2s+1)$. Für $X \in C$ und eine positive natürliche Zahl r setzen wir

$$U_r(X) = \{Y \mid Y \in \{0,1\}^n,\ d(Y,X) \leq r\}\,.$$

Wenn $d(Y,X) = k$ gilt, so unterscheiden sich X und Y an genau k Stellen. Umgekehrt erhalten wir bei beliebiger Wahl von k Stellen in X und der Änderung der Buchstaben an diesen Stellen ein Wort Y mit $d(Y,X) = k$. Da wir $\binom{n}{k}$ Möglichkeiten zur Wahl der k Stellen in X haben, gibt es genau $\binom{n}{k}$ Wörter Y der Länge n mit $d(Y,X) = k$. Somit ergibt sich wegen

$$U_r(X) = \bigcup_{k=0}^{r} \{Y \mid d(Y,X) = k\}$$

für die Anzahl der Elemente in $U_r(X)$

$$\#(U_r(X)) = \sum_{k=0}^{r} \binom{n}{k}\,.$$

Man beachte, dass diese Zahl für alle Elemente X gleich ist. Da die Mengen $U_s(X)$ und $U_s(X')$ für $X, X' \in C$ disjunkt sind (siehe Beweis von Satz 3.2), erhalten wir

$$m(n, 2s+1) \cdot \sum_{k=0}^{s} \binom{n}{k} = \#(C) \cdot \sum_{k=0}^{s} \binom{n}{k} = \#(\bigcup_{X \in C} U_s(X)) \leq \#(\{0,1\}^n) = 2^n \,,$$

woraus sich sofort die zweite Ungleichung der Behauptung ergibt.

Wir betrachten nun

$$V = \bigcup_{X \in C} U_{2s}(X) \,.$$

Da die Mengen $U_{2s}(X)$ und $U_{2s}(X')$ für $X, X' \in C$ nicht notwendig disjunkt sein müssen, erhalten wir

$$\#(V) \leq \#(C) \cdot \sum_{k=0}^{2s} \binom{n}{k} = m(n, 2s+1) \cdot \sum_{k=0}^{2s} \binom{n}{k} \,.$$

Nehmen wir nun an, dass

$$m(n, 2s+1) \cdot \sum_{k=0}^{2s} \binom{n}{k} < 2^n \tag{3.5}$$

gilt, so haben wir auch $\#(V) < 2^n$. Daher muss es dann ein Wort Z der Länge n geben, das nicht in V liegt. Damit gilt dann auch $Z \notin U_{2s}(X)$ für alle $X \in C$. Folglich gilt $d(Z, X) \geq 2s+1$ für alle $X \in C$. Folglich hat auch der Code $C \cup \{Z\}$ den Codeabstand $2s+1$ und ist in $\{0,1\}^n$ enthalten. Dies widerspricht aber der vorausgesetzten Maximalität von C hinsichtlich der Mächtigkeit. Deshalb kann (3.5) nicht gelten, womit

$$2^n \leq m(n, 2s+1) \cdot \sum_{k=0}^{2s} \binom{n}{k} \,,$$

gültig ist. Hieraus ergibt sich die erste Ungleichung der Behauptung sofort. □

Wir bemerken, dass aus dem letzten Teil des Beweises eine induktive Methode zur Konstruktion von hinsichtlich der Mächtigkeit maximalen Blockcodes vorgegebener Länge folgt. Diese ist durch das folgende „Programm“ gegeben, das einen Code C mit maximaler Mächtigkeit ermittelt:

$C := \emptyset$;
A: $V := \bigcup_{X \in C} U_{2s}(X)$;
if $V = \{0,1\}^n$ **then goto** B ;
Wähle $X \in \{0,1\}^n \setminus V$;
$C := C \cup \{X\}$;
goto A ;
B: **stop**

Wir geben jetzt einige Beziehungen zwischen den Werten $m(n, d)$ und $m(n', d')$ für gewisse Parameter n, n', d, d'.

Satz 3.4 *Für zwei beliebige positive natürliche Zahlen n und d (mit $n \geq d$) gilt*

$$m(n,d) \leq 2 \cdot m(n-1,d)\,.$$

Beweis. Es sei C ein maximaler Blockcode der Länge n mit Codeabstand d. Folglich gilt $\#(C) = m(n,d)$. Es seien C_0 bzw. C_1 die Teilmengen von C, die aus allen Wörtern bestehen, die mit 0 bzw. 1 beginnen. Dann gilt offenbar $\#(C_0) + \#(C_1) = \#(C)$. Daher gibt es ein $i \in \{0,1\}$ mit

$$\#(C_i) \geq \frac{\#(C)}{2} = \frac{m(n,d)}{2}\,. \tag{3.6}$$

Wir betrachten nun den Blockcode C_i', der aus C_i entsteht, indem wir in jedem Wort den ersten Buchstaben (das ist i) streichen. Dann gilt sicher $C_i' \in \{0,1\}^{n-1}$, da jeweils ein Buchstabe gestrichen wurde. Für ein Wort $X \in C_i'$ setzen wir $X' = iX$. Offenbar gilt $X' \in C_i$. Außerdem gilt für zwei Wörter X und Y aus C_i' die Beziehung $d(X,Y) = d(X',Y')$, da X' und Y' den gleichen ersten Buchstaben haben. Wegen $d(C) = d$ und $X', Y' \in C_i \subseteq C$ erhalten wir $d \leq d(X',Y') = d(X,Y)$, woraus $d(C_i') \geq d$ folgt. Ein maximaler Code $C' \subseteq \{0,1\}^{n-1}$ mit Codeabstand d enthält mindestens soviel Elemente wie C_i' enthalten. Mit (3.6) ergibt sich daraus

$$m(n-1,d) \geq \#(C_i') = \#(C_i) \geq \frac{m(n,d)}{2}$$

und damit die Behauptung. □

Satz 3.5 *Es seien n und d zwei beliebige positive natürliche Zahlen (mit $n \geq d$).*

i) Dann gilt $m(n,d) \geq m(n+1,d+1)$.
ii) Ist d ungerade, so gilt sogar $m(n,d) = m(n+1,d+1)$.

Beweis. i) Es sei C ein Code mit $C \subseteq \{0,1\}^{n+1}$, $d(C) = d+1$ und $\#(C) = m(n+1,d+1)$. Wir betrachten den Code C', der aus C entsteht, indem wir in jedem Wort aus C den ersten Buchstaben streichen. Offenbar gelten dann $C' \subseteq \{0,1\}^n$, $\#(C') = \#(C)$ und $d(C') \geq d$, da bei einem Unterschied im ersten Buchstaben der Abstand beim Übergang von Wörtern aus C zu Wörtern aus C' um 1 sinken kann. Für einen maximalen Blockcode B der Länge n und Codeabstand d erhalten wir somit

$$m(n,d) \geq \#(C') = \#(C) = m(n+1,d+1)$$

und damit Teil i) der Behauptung.

ii) Wegen i) reicht es, die Ungleichung $m(n+1,d+1) \geq m(n,d)$ für ungerade Zahlen d zu zeigen. Es sei dazu $B \subseteq \{0,1\}^n$ ein Blockcode mit $d(B) = d$ und $\#(B) = m(n,d)$. Jedem Wort $X = x_1x_2\ldots x_n \in B$ ordnen wir das Wort

$$X' = x_1x_2\ldots x_nx_{n+1} \quad \text{mit} \quad x_{n+1} = x_1 \oplus x_2 \oplus \cdots \oplus x_n$$

zu und betrachten den Code

$$B' = \{X' \mid X \in B\}\,.$$

Offenbar ist B' ein Blockcode aus Wörtern der Länge $n+1$ und enthält die gleiche Anzahl von Elementen wie B.

Wir zeigen nun, dass B' bei ungeradem d den Codeabstand $d+1$ hat. Dazu betrachten wir zwei Wörter X und Y aus B und die zugeordneten Wörter X' und Y' aus B'. Ohne Beschränkung der Allgemeinheit können wir annehmen, dass X und Y die Form

$$X = x_1 x_2 \dots x_r \ \underbrace{11\dots1}_{s}\underbrace{00\dots0}_{t},$$
$$Y = x_1 x_2 \dots x_r \ \underbrace{00\dots0}_{s}\underbrace{11\dots1}_{t}$$

haben (durch Umsortieren der Komponenten kann dies erreicht werden) und $s \le t$ gilt. Hieraus ergibt sich $d(X,Y) = s + t \ge d(B) = d$. Ferner haben X' und Y' die Form

$$X' = x_1 x_2 \dots x_r \ \underbrace{11\dots1}_{s}\underbrace{00\dots0}_{t} \ x_{n+1}\,,$$
$$Y' = x_1 x_2 \dots x_r \ \underbrace{00\dots0}_{s}\underbrace{11\dots1}_{t} \ y_{n+1}$$

mit

$$x_{n+1} = x_1 \oplus x_2 \oplus \dots \oplus x_r \oplus \underbrace{1 \oplus 1 \oplus \dots \oplus 1}_{s},$$
$$y_{n+1} = x_1 \oplus x_2 \oplus \dots \oplus x_r \oplus \underbrace{1 \oplus 1 \oplus \dots \oplus 1}_{t}\,.$$

Gilt $s + t \ge d + 1$, so ist auch $d(X', Y') \ge s + t \ge d + 1$ gültig.

Wir betrachten nun den Fall $s + t = d$. Da d ungerade ist, impliziert dies, dass genau eine der Zahlen s und t gerade ist und die andere ungerade ist. Folglich sind die Werte x_{n+1} und y_{n+1} verschieden. Somit ergibt sich $d(X', Y') = d + 1$.

Damit gilt in jedem Fall $d(X', Y') \ge d+1$ für $X', Y' \in B'$, woraus $d(B') \ge d+1$ folgt. Für einen maximalen Blockcode C der Länge $n + 1$ mit Codeabstand $d + 1$ gilt folglich

$$m(n + 1, d + 1) = \#(C) \ge \#(B') = \#(B) = m(n, d)\,.$$

□

Für zwei Wörter $X = x_1 x_2 \dots x_n$ und $Y = y_1 y_2 \dots y_n$ der Länge n definieren wir ihre Summe $X \oplus Y$ durch

$$X \oplus Y = (x_1 \oplus y_1)(x_2 \oplus y_2) \dots (x_n \oplus y_n)\,.$$

Offenbar gilt

$$d(X, Y) = \#_1(X \oplus Y)\,. \tag{3.7}$$

Satz 3.6 *Für zwei beliebige positive natürliche Zahlen n und d (mit $n \ge d$) gilt*

$$m(2n, 2d) \ge m(n, d) \cdot m(n, 2d)\,.$$

Beweis. Es seien $C_1 \subseteq \{0,1\}^n$ ein Code mit $d(C_1) = d$ und $\#(C_1) = m(n,d)$ und $C_2 \subseteq \{0,1\}^n$ ein Code mit $d(C_2) = 2d$ und $\#(C_1) = m(n,2d)$. Zwei Wörtern $X \in C_1$ und $Y \in C_2$ ordnen wir das Wort

$$w(X,Y) = XX \oplus Y0^n$$

zu und betrachten den Code

$$C = \{w(X,Y) \mid X \in C_1, Y \in C_2\}\,.$$

Nach Definition ist C ein Blockcode mit Wörtern der Länge $2n$. Außerdem besteht C aus $\#(C_1) \cdot \#(C_2) = m(n,d) \cdot m(n,2d)$ Elementen. Wir zeigen nun noch $d(C) \geq 2d$. Hieraus folgt dann

$$m(2n,2d) \geq \#(C) = m(n,d) \cdot m(n,2d)$$

und damit die Behauptung.

Daher reicht es, für beliebige $X', X'' \in C_1$ und $Y', Y'' \in C_2$ zu zeigen, dass

$$2d \leq d(w(X',Y'), w(X'',Y'')) = d(X'X' \oplus Y'0^n, X''X'' \oplus Y''0^n)$$

für $w(X',Y') \neq w(X'',Y'')$, d. h. für $X' \neq X''$ oder $Y' \neq Y''$, gilt. Durch getrennte Betrachtung der ersten und letzten n Buchstaben erhalten wir aus (3.7)

$$\begin{aligned} d(w(X',Y'), w(X'',Y'')) &= \#_1((X' \oplus Y') \oplus (X'' \oplus Y'')) + \#_1((X' \oplus 0^n) \oplus (X'' \oplus 0^n)) \\ &= \#_1((X' \oplus X'') \oplus (Y' \oplus Y'')) + \#_1(X' \oplus X'')\,. \end{aligned} \tag{3.8}$$

Falls $Y' = Y''$ (und damit $Y' \oplus Y'' = 0^n$) und $X' \neq X''$ gelten, ergibt sich aus (3.8) sofort

$$d(w(X',Y'), w(X'',Y'')) = 2 \cdot \#_1(X' \oplus X'') = 2 \cdot d(X',X'') \geq 2 \cdot d(C_1) = 2 \cdot d\,.$$

Es sei daher $Y' \neq Y''$. Aus (3.7), (3.8) und den Eigenschaften der Abstandsfunktion d folgt dann

$$\begin{aligned} d(w(X',Y'), w(X'',Y'')) &= \#_1((X' \oplus X'') \oplus (Y' \oplus Y'')) + \#_1((X' \oplus X'') \oplus 0^n) \\ &= d(X' \oplus X'', Y' \oplus Y'') + d(X' \oplus X'', 0^n) \\ &= d(Y' \oplus Y'', X' \oplus X'') + d(X' \oplus X'', 0^n) \\ &\geq d(Y' \oplus Y'', 0^n) = \#_1(Y' \oplus Y'') = d(Y',Y'') \geq d(C_2) \\ &= 2d. \end{aligned}$$

□

Satz 3.7 *Es seien n und d zwei beliebige positive natürliche Zahlen n und d (mit $n \geq d$).*
i) Falls d eine gerade Zahl ist, gilt

$$\begin{aligned} &m(n,d) \leq 2 \cdot \lfloor \tfrac{d}{2d-n} \rfloor \quad \textit{für } 2d > n, \\ &m(n,d) \leq 2n \qquad\qquad\quad \textit{für } 2d = n\,. \end{aligned}$$

ii) Falls d eine ungerade Zahl ist, gilt

$$\begin{array}{ll} m(n,d) \leq 2 \cdot \lfloor \frac{d+1}{2d+1-n} \rfloor & \textit{für } 2d+1 > n, \\ m(n,d) \leq 2(n+1) & \textit{für } 2d+1 = n\,. \end{array}$$

iii) Es gelten

$$\begin{array}{ll} m(n,d) \leq d \cdot 2^{n-2d+2} & \textit{für gerade Zahlen } d \textit{ und } n \geq 2d\,, \\ m(n,d) \leq (d+1) \cdot 2^{n-2d+1} & \textit{für ungerade Zahlen } d \textit{ und } n \geq 2d+1\,. \end{array}$$

Beweis. Es sei C ein beliebiger Code. Wir definieren $R(C)$ als die Summe aller Abstände zwischen Wörtern aus C. Für $1 \leq i \leq n$ bezeichnen wir mit h_i die Anzahl der Wörter aus C, deren i-ter Buchstabe eine 1 ist. Folglich ist $\#(C) - h_i$ die Anzahl der Codewörter mit 0 als i-tem Buchstaben. Jedes Paar von Wörtern X und Y aus C mit 1 bzw. 0 an der i-ten Stelle trägt durch diesen Unterschied 1 zu $R(C)$ bei. Daher gilt

$$R(C) = \sum_{i=1}^{n} h_i(\#(C) - h_i)\,. \tag{3.9}$$

Weiterhin gilt für je zwei Wörter $X, Y \in C$ mit $X \neq Y$ die Beziehung $d(C) \leq d(X,Y)$. Da es $\frac{\#(C)(\#(C)-1)}{2}$ verschiedene ungeordnete Paare von verschiedenen Wörtern aus C gibt, gilt

$$\frac{\#(C)(\#(C)-1)}{2} \cdot d(C) \leq R(C)\,. \tag{3.10}$$

Es sei nun C ein Code mit Wörtern der Länge n, $d(C) = d$ und $\#(C) = m(n,d)$. Wir setzen zur Abkürzung $m = m(n,d)$.

Es sei zuerst m eine gerade Zahl. Dann nimmt der Ausdruck $h(m-h)$ den maximalen Wert bei $h = \frac{m}{2}$ an. Folglich ergibt sich aus (3.9) und (3.10)

$$\frac{m(m-1)}{2} \cdot d \leq R(C) \leq n \cdot \frac{m}{2}(m - \frac{m}{2}) = n \cdot \frac{m^2}{4}\,.$$

Ist m ungerade, so liegt das Maximum der Funktion bei $\frac{m-1}{2}$. Damit ergibt aus (3.9) und (3.10)

$$\frac{m(m-1)}{2} \cdot d \leq R(C) \leq n \cdot \frac{m-1}{2}(m - \frac{m-1}{2}) = n \cdot \frac{m-1}{2} \cdot \frac{m+1}{2} = n \cdot \frac{m^2-1}{4} \leq n \cdot \frac{m^2}{4}\,.$$

In beiden Fällen gilt also

$$\frac{m(m-1)}{2} \cdot d \leq n \cdot \frac{m^2}{4}\,.$$

Durch einfaches algebraisches Umformen erhalten wir hieraus

$$m^2 \cdot \frac{2d-n}{2} \leq md\,.$$

Für $2d \geq n$ ergibt sich

$$m \leq \frac{2d}{2d-n}\,,$$

woraus wegen der Ganzzahligkeit von m die schärfere Aussage

$$m \leq 2 \cdot \lfloor \frac{d}{2d-n} \rfloor$$

folgt. Wegen $m = m(n,d)$ ist damit der erste Teil von i) bereits gezeigt.

Es sei nun $2d = n$. Dann folgt unter Verwendung von Satz 3.4 und der gerade bewiesenen Ungleichung aus i) (für $2d \geq 2d-1 = n'$)

$$m(n,d) = m(2d,d) = 2 \cdot m(2d-1,d) \leq 2 \cdot 2 \cdot \lfloor \frac{d}{2d-(2d-1)} \rfloor = 4d = 2n\,.$$

Damit ist auch der zweite Teil von i) bewiesen.

Wir verschärfen[1] nun die beiden Aussagen aus i) für ungerades d unter Verwendung von Satz 3.6. Wir erhalten

$$m(n,d) = m(n+1,d+1) \leq 2 \cdot \lfloor \frac{d+1}{2(d+1)-(n+1)} \rfloor = 2 \cdot \lfloor \frac{d+1}{2d+1-n} \rfloor \quad \text{für} \quad 2d+1 \geq n$$

und

$$m(2d+1,d) = m(2d+2,d+1) \leq 4(d+1) \quad \text{für} \quad 2d+1 = n\,.$$

Dies sind gerade die Beziehungen aus ii).

Wir beweisen nun iii) durch vollständige Induktion über n.

Es sei zuerst d gerade. Für $n = 2d$ erhalten wir aus der zweiten Aussage von Teil i)

$$m(n,d) \leq 2n = 4d = d \cdot 2^2 = d \cdot 2^{n-2d+2}$$

und somit den Induktionsanfang für $n = 2d$. Ist die Aussage schon für $n \geq 2d$ bewiesen, so folgt sie für $n+1$ wegen Satz 3.4 aus

$$m(n+1,d) \leq 2 \cdot m(n,d) \leq 2 \cdot d \cdot 2^{n-2d+2} = d \cdot 2^{(n+1)-2d+2}\,.$$

Für ungerades d folgt der Induktionsanfang für $n = 2d+1$ aus der zweiten Aussage von ii) und einem Induktionsschritt wie für den geraden Fall. □

[1] Der Leser möge sich überlegen, dass tatsächlich $\frac{d+1}{2d+1-n} \leq \frac{d}{2d-n}$ für $2d \geq n$ gilt.

Kapitel 4

Lineare Codes

Bisher haben wir Codes als Mengen von Wörtern aufgefasst. Um Codeeigenschaften zu ermitteln oder zu untersuchen, haben wir im Wesentlichen kombinatorische Eigenschaften der Wortmenge bzw. der Wörter selbst betrachtet. In Abschnitt 3.2 haben wir bei den Hamming-Codes festgestellt, dass die Menge der Codewörter sogar einen linearen Vektorraum bildet. Daher können neben kombinatorischen Eigenschaften auch die algebraischen Eigenschaften der Wortmenge beim Studium der Hamming-Codes verwendet werden. Diese Möglichkeit soll in diesem Kapitel für eine ganze Klasse von Codes genutzt werden.

Jedem Wort $w = a_1a_2 \dots a_n$ der Länge n kann in eineindeutiger Weise der n-dimensionale Zeilenvektor $v_w = (a_1, a_2, \dots, a_n)$ zugeordnet werden. In diesem Abschnitt werden wir oft nicht zwischen dem Wort und seinem zugeordneten Vektor unterscheiden. Daher erhalten wir auch eine Addition von Wörtern der Länge n, da im Vektorraum aller n-dimensionalen Vektoren eine Addition definiert ist. Dies liefert

$$a_1a_2 \dots a_n \oplus b_1b_2 \dots b_n = c_1c_2 \dots c_n \text{ mit } c_i = a_i \oplus b_i \text{ für } 1 \leq i \leq n\,.$$

Definition 4.1 *Ein Blockcode $C \subseteq \{0,1\}^n$ heißt linearer Code, wenn die Elemente aus C einen linearen Vektorraum über dem Körper $\{0,1\}$ bilden.*[1]

Aus den Eigenschaften eines linearen Vektorraumes folgt sofort, dass das Wort 0^n in jedem linearen Code $C \subseteq \{0,1\}^n$ ist.

Ein linearer Code $C \subseteq \{0,1\}^n$ hat als Vektorraum eine Dimension, die wir mit $dim(C)$ bezeichnen. Wir sagen dann auch, dass C ein $[n, dim(C)]$-Code ist.

Definition 4.2 *Es sei C ein $[n,k]$-Code.*

i) Eine Matrix G vom Typ (k,n) heißt Erzeugendenmatrix für C, falls die k Zeilen von G ein Erzeugendensystem für C (als Vektorraum) bilden.

ii) Eine Matrix H vom Typ $(n-k,n)$ heißt Kontrollmatrix für C, falls

$$C = \{c \mid c \in \{0,1\}^n, Hc^T = (0^{n-k})^T\}$$

gilt.

[1] Wie schon im vorhergehenden Kapitel beschränken wir uns auch in diesem Kapitel auf Codes über $\{0,1\}$. Einige unserer Konzepte und Resultate können aber auch für den Fall formuliert bzw. bewiesen werden, wenn man anstelle des Körpers $\{0,1\}$ einen anderen endlichen Körper K und Codes über K (genauer Blockcodes, die in K^* enthalten sind) betrachtet.

Da jeder lineare Vektorraum eine Basis hat und die Elemente der Erzeugenmatrix eine Basis bilden, gibt es zu jedem linearen Code eine Erzeugendenmatrix.

Es sei C ein $[n,k]$-Code. Dann bilden die n-dimensionalen Vektoren, die senkrecht auf allen Vektoren aus C stehen, d. h. die Menge aller v mit $vc^T = 0$ für alle $c \in C$, einen linearen Vektorraum C' der Dimension $n-k$. Wählen wir nun eine Basis von C' und nehmen deren Elemente als Zeilen einer Matrix, so bilden diese eine Kontrollmatrix H für C. Dies folgt daraus, dass die Menge C'' aller Vektoren x mit $Hx^T = (0^{n-k})^T$ als Lösungsmenge eines linearen homogenen Gleichungssystems einen linearen Vektorraum der Dimension $n-(n-k) = k$ bildet und nach Definition alle Elemente aus C in C'' liegen, woraus sich $C'' = C$ ergibt. Damit hat auch jeder lineare Code eine Kontrollmatrix.

Für den Hamming-Code aus Abschnitt 3.2. ergeben sich als Erzeugendenmatrix

$$G = \begin{pmatrix} 0\,1\,0\,1\,0\,1 \\ 1\,0\,0\,1\,1\,0 \\ 0\,0\,1\,0\,1\,1 \end{pmatrix},$$

indem wir die Codewörter so wählen, dass die ersten drei Komponenten eine Permutation der Einheitsmatrix bilden, und als Kontrollmatrix

$$H = \begin{pmatrix} 0\,1\,0\,1\,0\,1 \\ 1\,0\,0\,1\,1\,0 \\ 1\,1\,1\,0\,0\,0 \end{pmatrix},$$

indem wir aus der Tabelle in Abschnitt 3.2 die Wahlen von x_3, x_5 und x_6 auswählen, bei denen diese Komponenten erneut im Wesentlichen die Einheitsmatrix bilden.

Definition 4.3 *i) Unter dem Gewicht $w(c)$ eines Wortes $c \in \{0,1\}^*$ verstehen wir die Anzahl der in c vorkommenden Einsen.*

ii) Das Gewicht $w(C)$ eines Blockcodes $C \subseteq \{0,1\}^n$ wird durch

$$w(C) = \min\{w(c) \mid w \in C \setminus \{0^n\}\}$$

definiert.

Offensichtlich gelten die Gleichungen $w(c) = \#_1(c)$ und $d(c_1,c_2) = w(c_1 \oplus c_2)$ für alle Wörter $c_1, c_2 \in \{0,1\}^n$. Für $c = c_1c_2\ldots c_n$ setzen wir

$$Tr(c) = \{i \mid c_i = 1\}.$$

Dann gilt offenbar $w(c) = \#(Tr(c))$.

Es seien c_1 und c_2 zwei Wörter der Länge n. Ohne Beschränkung der Allgemeinheit können wir annehmen, dass

$$c_1 = 1^t0^s1^r0^{n-t-s-r} \quad \text{und} \quad c_2 = 0^t1^s1^r0^{n-t-s-r}$$

gilt (durch Umordnen kann dies stets erreicht werden). Es gilt dann

$$w(c_1 \oplus c_2) = t+s = (t+r)+(s+r)-2r = w(c_1)+w(c_2)-2\cdot\#(Tr(c_1)\cap Tr(c_2)). \quad (4.1)$$

Wir geben nun eine Charakterisierung des Gewichtes eines Codes durch die Kontrollmatrix an.

Satz 4.1 *Es sei C ein linearer $[n,k]$-Code und H eine Kontrollmatrix für C. Dann gilt*

$$\begin{aligned} w(C) &= \min\{r \mid \textit{es gibt } r \textit{ linear abhängige Spalten von } H\} \\ &= \max\{r \mid \textit{je } r-1 \textit{ Spalten von } H \textit{ sind linear unabhängig}\}. \end{aligned}$$

Beweis. Es seien $h_1, h_2, \ldots, h_n$ die Spalten von H. Da H nur $n-k$ Zeilen hat, sind die n Spalten linear abhängig. Es sei nun p die minimale Anzahl linear abhängiger Spalten von H. Außerdem seien $h_{i_1}, h_{i_2}, \ldots, h_{i_p}$ linear abhängige Spalten. Dann gilt wegen der Minimalität von p die Beziehung $\sum_{j=1}^{p} h_{i_j} = (0^{n-k})^T$. Wir definieren nun den Vektor $v = (v_1, v_2, \ldots v_n)$ dadurch, dass wir genau dann $v_i = 1$ setzen, wenn $i \in \{i_1, i_2, \ldots, i_p\}$. Dann gilt

$$Hv^T = \sum_{i=1}^{n} v_i h_i = \sum_{j=1}^{p} h_{i_j} = (0^{n-k})^T .$$

Damit ist nach Definition der Kontrollmatrix $v \in C$. Da $w(v) = p$ ist, erhalten wir $p \geq w(C)$.

Angenommen, es gibt ein $c \in C$ mit $w(c) = t < p$. Es seien $c_{k_1}, c_{k_2}, \ldots, c_{k_t}$ die von Null verschiedenen Komponenten von c. Da $Hc^T = (0^{n-k})^T$ gilt, ist

$$(0^{n-k})^T = \sum_{i=1}^{n} c_i h_i = \sum_{j=1}^{t} h_{k_j} .$$

Damit sind die t Spalten $h_{k_1}, h_{k_2}, \ldots, h_{k_t}$ linear abhängig, was wegen der Minimalität von p nicht möglich ist. Folglich gilt

$$w(C) = \min\{r \mid \text{es gibt } r \text{ linear abhängige Spalten von } H\} .$$

Die andere Gleichheit folgt sofort. □

Wir wollen nun zeigen, dass die Berechnung des Codeabstandes bei linearen Codes (erheblich) einfacher ist als bei beliebigen Blockcodes.

Satz 4.2 *Für einen linearen Code C gilt $d(C) = w(C)$.*

Beweis. Es sei zuerst c ein Codewort aus $C \subseteq \{0,1\}^n$, für das $w(C) = w(c)$ gilt. Da C ein linearer Code ist, ist $0^n \in C$. Offenbar gilt $w(c) = d(c, 0^n) \geq d(C)$. Folglich haben wir $w(C) \geq d(C)$.

Es seien nun c_1 und c_2 zwei (verschiedene) Codewörter aus C mit $d(c_1, c_2) = d(C)$. Dann erhalten wir $d(C) = d(c_1, c_2) = w(c_1 \oplus c_2) \geq w(C)$.

Somit folgt $d(C) = w(C)$. □

Zur Bestimmung des Codeabstandes müssen wir im allgemeinen Fall alle Abstände zwischen zwei Codewörtern betrachten und dann das Minimum bestimmen. Dies erfordert einen quadratischen Aufwand in der Anzahl der Codewörter. Bei linearen Codes haben wir dagegen nur das Minimum der Gewichte zu ermitteln, was mit linearen Aufwand erfolgen kann.

Als zweites Beispiel für die Effizienz von linearen Codes betrachten wir die Decodierung. Nehmen wir an, dass wir ein Wort v empfangen haben. Falls es kein Codewort ist, so ist es sehr natürlich anzunehmen, dass jenes Codewort x gesendet wurde, für das

$$d(v,x) = \min\{d(v,c) \mid c \in C\} \tag{4.2}$$

erfüllt ist. Dies erfordert im Allgemeinen einen Aufwand $k' \cdot n \cdot \#(C)$, wobei k' eine Konstante ist (für jedes Codewort erfordert die Berechnung des Abstandes n Vergleiche).

Es sei nun C ein linearer $[n,k]$-Code. Wir definieren eine Äquivalenzrelation ϱ in $\{0,1\}^n$ durch

$$(x,y) \in \varrho \quad \text{genau dann, wenn} \quad Hx^T = Hy^T\,.$$

(Es ist leicht zu sehen, dass ϱ tatsächlich eine Äquivalenzrelation ist.) Die Nebenklasse bez. ϱ, die 0^n enthält, besteht dann genau aus den Vektoren x mit

$$Hx^T = H(0^n)^T = (0^{n-k})^T.$$

Damit besteht diese Nebenklasse genau aus den Elementen aus C. Es sei nun f ein Repräsentant einer Nebenklasse N von ϱ. Dann gilt $N = f \oplus C$. Somit gibt es $2^n/\#(C)$ Nebenklassen.

Für jede Nebenklasse N bestimmen wir nun ein Element f_N mit

$$w(f_N) = \min\{w(y) \mid y \in N\}\,.$$

Das empfangene Wort v liegt in einer Äquivalenzklasse, sagen wir in N. Für das Codewort x mit (4.2) und den Vektor $f = v \ominus x$ gilt

$$Hf^T = H(v \ominus x)^T = Hv^T \ominus Hx^T = Hv^T\,,$$

d. h., dass v und f in der gleichen Nebenklasse, also in N liegen. Weiterhin haben wir aber auch $f_N = v \oplus c'$ für ein Codewort c'. Damit gilt $w(f_N) = d(v,c')$. Wegen der Wahl von f gilt daher $w(f) \le w(f_N)$. Nach Wahl von f_N gilt aber auch $w(f_N) \le w(f)$. Folglich ist $w(f_N) = w(f)$. Somit können wir zur Decodierung von v, das Codewort $c' = v \ominus f_N$ verwenden.

Um das gesendete Codewort zu ermitteln, reicht es also, die Elemente f_N, wobei N eine Nebenklasse ist, durchzumustern und festzustellen, welches von diesen $Hf_N^T = Hv^T$ erfüllt. Da die Vektoren Hf_N^T vorab berechnet werden können, muss also nur Hv^T berechnet werden und mit den Hf_N^T verglichen werden. Die Berechnung von Hv^T kann in $k'' \cdot n^2$ Schritten erfolgen, wobei k'' eine Konstante ist (man gehe entsprechend der Definition des Produktes vor), jeder der Vergleiche erfordert n Schritte. Folglich haben wir höchstens

$$k'' \cdot n^2 + n \cdot \frac{2^n}{\#(C)}$$

Schritte auszuführen. Falls $dim(C) = k > \frac{n}{2}$ ist, so ist der Aufwand kleiner als $k''n^2 + n2^{n/2}$. und damit geringer als der des obigen allgemeinen Verfahrens, das $k'n2^k$ Schritte erfordert.

Wir wollen nun ein paar Aussagen über die maximale Mächtigkeit linearer Codes treffen. Da die Anzahl der Elemente eines linearen Codes der Dimension k gerade 2^k ist, reicht es, die Dimension zu maximieren.

Für $n \geq 1$ und $d \geq 1$ definieren wir

$$k(n,d) = \max\{dim(C) \mid C \subseteq \{0,1\}^n \text{ ist linearer Code mit } d(C) \geq d\}\,.$$

Aus den Aussagen der Sätze 3.4 – 3.6 und der Tatsache, dass einer Multiplikation bei der Mächtigkeit des Codes eine Addition der Dimension entspricht erhalten wir sofort

$$\begin{aligned} k(n,d) &\leq k(n-1,d)+1\,,\\ k(n,d) &= k(n+1,d-1) \text{ für ungerades } d\,,\\ k(2n,2d) &\geq k(n,d)+k(n,2d)\,. \end{aligned}$$

Weiterhin definieren wir $n(k,d)$ als die minimale Zahl n, so dass ein linearer Code C mit $C \subseteq \{0,1\}^n$, $d(C) = d$ und $dim(C) = k$ existiert.

Ohne Beweis bemerken wir, dass n eine sowohl in k als auch in d wachsende Funktion ist, d. h. es gelten für beliebiges $k \geq 1$ und $d \geq 1$ die Ungleichungen

$$n(k+1,d) > n(k,d) \quad \text{und} \quad n(k,d+1) > n(k,d)\,.$$

Satz 4.3 *Für $k > 1$ gilt*

$$n(k,d) \geq n\big(k-1, \lceil \frac{d}{2} \rceil\big) + d\,.$$

Beweis. Es sei C ein linearer $[n,k]$-Code mit $n = n(k,d)$ und Codeabstand d. Ferner sei G eine Erzeugendenmatrix von C. Ohne Beschränkung der Allgemeinheit können wir annehmen, dass eine Zeile von G durch das Codewort $c_1 = 0^{n-d}1^d$ gebildet wird (wir wählen als eines der erzeugenden Elemente von C ein Wort c_1' mit $d = d(c_1', 0^n)$ und vertauschen notfalls die Reihenfolge (d. h. die Spalten der Matrix), um c zu erhalten). Die Zeilen von G seien $c_1, c_2, \ldots, c_k$. Für $2 \leq i \leq k$ sei d_i die Zeile aus G, die aus c_i durch Streichen der letzten d Komponenten hervorgeht. Es sei G' die Matrix mit den Zeilen $d_2, d_3, \ldots, d_k$.

Angenommen, die Zeilen von G' wären linear abhängig. Dann hätten wir

$$\sum_{i=2}^{k} \alpha_i d_i = 0^{n-d},$$

wobei $\alpha_i = 1$ für mindestens ein i mit $2 \leq i \leq d$ gilt. Wir betrachten nun

$$c = \sum_{i=2}^{k} \alpha_i c_i.$$

Offensichtlich ist $c \in C$ wegen der Linearität von C. Außerdem gilt $c = 0^{n-d}c'$. Falls $c \neq c_1$, so gilt $d(c, 0^n) < d$, womit ein Widerspruch zu $d = d(C)$ besteht. Somit muss $c = c_1$ sein. Dann gilt aber (wegen $1 = -1$) $c_1 \oplus \sum_{i=2}^{k} \alpha_i c_i = 0^n$ im Widerspruch zur linearen Unabhängigkeit der Zeilen von G. Daher ist unsere Annahme falsch, und folglich sind die Zeilen $d_2, d_3, \ldots, d_k$ linear unabhängig und bilden einen $[n-d, k-1]$-Code C'.

Es sei $d' = d(C')$. Dann gibt es eine Zeile b' in G' mit $w(b') = d'$. In G gibt es dann eine Zeile $b = b'b''$ mit $b'' \in \{0,1\}^d$. Damit erhalten wir $c_1 \oplus b \in C$ und

$$w(c_1 \oplus b) = d' + d - w(b'') \geq d \text{ und } w(b) = d' + w(b'') \geq d\,,$$

woraus $2d' \geq d$ resultiert. Folglich ist $d(C') \geq \lceil d/2 \rceil$ und $n - d \geq n(k-1, \lceil d/2 \rceil)$, was zu beweisen war. □

Unter Berücksichtigung von

$$n(1,d) = d \quad \text{und} \quad \left\lceil \frac{\lceil d/2 \rceil}{2^i} \right\rceil = \lceil \frac{d}{2^{i+1}} \rceil$$

ergibt sich aus Satz 4.3 durch Induktion sofort die folgende Aussage, die als Griesmer-Schranke für lineare Codes bekannt ist.

Folgerung 4.4 *Für $k \geq 1$ gilt*

$$n(k,d) \geq \sum_{i=0}^{k-1} \lceil \frac{d}{2^i} \rceil\,.$$

Aus der Griesmer-Schranke ergibt sich auch eine Abschätzung für $k(n,d)$.

Folgerung 4.5 *Es gilt*

$$k(n,d) \leq \max\{k \mid \sum_{i=1}^{k-1} \lceil \frac{d}{2^i} \rceil \leq n\}\,.$$

Beweis. Es sei

$$l = \max\{k \mid \sum_{i=0}^{k-1} \lceil \frac{d}{2^i} \rceil \leq n\}\,.$$

Dann gilt wegen Folgerung 4.4

$$n(l+1,d) \geq \sum_{i=0}^{l} \lceil \frac{d}{2^i} \rceil > n\,.$$

Wäre nun $k(n,d) > l$, so wäre wegen der Monotonie von $n(k,d)$ auch

$$n = n(k(n,d),d) \geq n(l+1,d) > n,$$

woraus ein Widerspruch resultiert. □

Wir wollen nun lineare Codes konstruieren, die beweisen, dass die Griesmer-Schranke optimal ist. Dazu geben wir zuerst eine Methode an, mit der aus linearen Codes neue lineare Codes gewonnen werden können und die Parameter des neuen Codes sich aus denen der gegebenen Codes einfach errechnen lassen.

Lemma 4.6 *Es seien zwei linearen Codes C_1 und C_2 mit Dimensionen k_1 bzw. k_2 und Codeabständen d_1 bzw. d_2 gegeben. Dann ist*

$$C = C_1 \alpha C_2 = \{(c_1, c_1 \oplus c_2) \mid c_1 \in C_1, c_2 \in C_2\}$$

ein linearer Code mit

$$C \subseteq \{0,1\}^{2n}, \quad dim(C) = k_1 + k_2 \quad und \quad d(C) = \min\{2d_1, d_2\}\,.$$

Beweis. Nach Definition gilt $C \subseteq \{0,1\}^{2n}$. Damit ist C ein Blockcode. Es bleibt daher zu zeigen, dass C ein linearer Vektorraum ist, um C als linearen Code nachzuweisen. Dafür reicht es nach den Kriterien für Vektorräume zu beweisen, dass für beliebige Elemente x und y aus C und $\gamma \in \{0,1\}$ auch $x \oplus y$ und γx in C liegen. Es seien $x = (c_1, c_1 \oplus c_2)$ und $y = (c_1', c_1' \oplus c_2')$ mit $c_1, c_1' \in C_1$ und $c_2, c_2' \in C_2$. Dann ergibt sich

$$\begin{aligned} x \oplus y &= (c_1, c_1 \oplus c_2) \oplus (c_1', c_1' \oplus c_2') = (c_1 \oplus c_1', (c_1 \oplus c_2) \oplus (c_1' \oplus c_2')) \\ &= (c_1 \oplus c_1', (c_1 \oplus c_1') \oplus (c_2 \oplus c_2'))\,, \end{aligned}$$

woraus mit $c_1 \oplus c_1' \in C_1$ und $c_2 \oplus c_2' \in C_2$ folgt, dass $x \oplus y \in C$ gilt. Wegen

$$0 \cdot x = 0^{2n} = (0^n, 0^n \oplus 0^n) \in C$$

und $1 \cdot x = x \in C$ ist auch die zweite Forderung erfüllt.

Offensichtlich ist die Abbildung $\tau : (C_1 \times C_2) \to C$ vermöge $(c_1, c_2) \to (c_1, c_1 \oplus c_2)$ eine Isomorphie zwischen den Vektorräumen $(C_1 \times C_2)$ (mit komponentenweiser Addition) und C (denn es gelten

$$\begin{aligned} \tau((c_1, c_1') \oplus (c_2, c_2')) &= \tau((c_1 \oplus c_2, c_1' \oplus c_2')) \\ &= (c_1 \oplus c_2, (c_1 \oplus c_2) \oplus (c_1' \oplus c_2')) \\ &= (c_1, c_1 \oplus c_1') \oplus (c_2, c_2 \oplus c_2') \\ &= \tau((c_1, c_1')) \oplus \tau((c_2, c_2')) \end{aligned}$$

und

$$\tau(\gamma(c_1, c_2)) = \tau((\gamma c_1, \gamma c_2)) = (\gamma c_1, \gamma c_1 \oplus \gamma c_2) = (\gamma c_1, \gamma(c_1 \oplus c_2)) = \gamma\tau((c_1, c_2))\,.$$

Damit gilt $dim(C) = dim(C_1 \times C_2) = k_1 + k_2$.

Es sei $c = (c_1, c_1 \oplus c_2)$. Zuerst betrachten wir den Fall, dass $c_2 = 0^n$ gilt. Dann ergibt sich

$$w(c) = w(c_1) + w(c_1) = 2 \cdot w(c_1) \geq 2d_1\,.$$

Für $c_2 \neq 0^n$ erhalten wir

$$\begin{aligned} w(c) &= w(c_1) + w(c_1 \oplus c_2) \\ &= w(c_1) + w(c_1) + w(c_2) - 2 \cdot \#(Tr(c_1) \cap Tr(c_2)) \quad \text{(wegen (4.1))} \\ &\geq w(c_2) \qquad \text{(wegen } w(c_1) = \#(Tr(c_1)) \geq \#(Tr(c_1) \cap Tr(c_2))) \\ &\geq d_2\,. \end{aligned}$$

Somit haben wir $w(c) \geq \min\{2d_1, d_2\}$ für alle $c \in C$. Damit gilt

$$d(C) = w(C) \geq \min\{2d_1, d_2\}.$$

Es seien c_1 und c_2 Codewörter aus C_1 bzw. C_2 so, dass $w(c_1) = d(C_1) = d_1$ bzw. $w(c_2) = d(C_2) = d_2$ gelten (also von minimalen Gewicht in den Codes). Dann erhalten wir wegen $0^n \in C_1 \cap C_2$

$$w((c_1, c_1 \oplus 0^n)) = 2w(c_1) = 2d_1 \quad \text{und} \quad w((0^n, 0^n \oplus c_2)) = w(c_2) = d_2\,.$$

Daher gilt

$$d(C) = w(C) = \min\{w(c) \mid c \in C\} \leq \min\{2d_1, d_2\}\,,$$

woraus mit Obigem sofort $d(C) = \min\{2d_1, d_2\}$ folgt. □

Wir nutzen die Konstruktion von linearen Codes aus linearen Codes, die in Lemma 4.6 eingeführt wurde, um Reed-Muller-Codes $RM(r, m)$ für $r, m \in \mathbb{N}$ und $0 \leq r \leq m$ zu definieren. Dazu definieren wir zuerst

$$RM(0, m) = \{0^{2^m}, 1^{2^m}\} \quad \text{und} \quad RM(m, m) = \{0, 1\}^{2^m}$$

und setzen für $1 \leq r \leq m - 1$

$$RM(r, m) = RM(r, m-1)\alpha RM(r-1, m-1)\,.$$

Wir bestimmen die Reed-Muller-Codes für kleine Werte von r und m. Für $m = 0$ erhalten wir den Code

$$RM(0, 0) = \{0, 1\}\,.$$

Für $m = 1$ erhalten wir die Codes

$$RM(0, 1) = \{00, 11\} \quad \text{und} \quad RM(1, 1) = \{00, 01, 10, 11\}\,.$$

Für $m = 2$ und $m = 3$ ergeben sich die Codes

$$\begin{aligned}
RM(0, 2) &= \{0000, 1111\}\,,\\
RM(1, 2) &= RM(1, 1)\alpha RM(0, 1) = \{0000, 0101, 1010, 1111, 0011, 0110, 1001, 1100\},\\
RM(2, 2) &= \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000,\\
&\qquad 1001, 1010, 1011, 1100, 1101, 1110, 1111\}\,.
\end{aligned}$$

und

$$\begin{aligned}
RM(0, 3) &= \{00000000, 11111111\}\,,\\
RM(1, 3) &= \{00000000, 01010101, 10101010, 11111111, 00110011, 01100110,\\
&\qquad 10011001, 11001100, 00001111, 01011010, 10100101,\\
&\qquad 11110000, 00111100, 01101001, 10010110, 11000011\}\,,\\
RM(2, 3) &= \{w_1 w_2 \mid w_1 \in \{0, 1\}^4,\ w_2 = w_1 \oplus v,\ v \in RM(1, 2)\}\,,\\
RM(3, 3) &= \{0, 1\}^8\,.
\end{aligned}$$

Reed-Muller-Codes wurden in den siebziger Jahren des vorigen Jahrhundert bei der Weltraumfahrt genutzt.

Wir wollen nun zeigen, dass die Reed-Muller-Codes $RM(r,m)$ lineare $[2^m, \sum_{i=0}^{r}\binom{m}{i}]$-Codes mit dem Codeabstand 2^{m-r} sind.

Wir beweisen dies mittels vollständiger Induktion über m. Für $m=0$ ergibt sich die Aussage sofort aus der Definition von $RM(0,0)$, der ein linearer Code in $\{0,1\}^{2^0}$ mit der Dimension $1(=\sum_{i=0}^{0}\binom{0}{i})$ und dem Codeabstand 2^0 ist. Für $m>0$ und $r=0$ erhalten wir aus der Definition von $RM(0,m)$, dass dies ein linearer Code in $\{0,1\}^{2^m}$ mit der Dimension $1(=\sum_{i=0}^{0}\binom{m}{i})$ und dem Codeabstand 2^m ist. Für $m>0$ und $r=m$ erhalten wir aus der Definition von $RM(m,m)$, dass dies ein linearer Code in $\{0,1\}^{2^m}$ mit der Dimension $2^m(=\sum_{i=0}^{m}\binom{m}{i})$ und dem Codeabstand 2^0 ist.

Wegen Lemma 4.6 erhalten wir, dass $RM(r,m)$ für $m>0$ und $1\leq r\leq m-1$ ein linearer Code mit

$$
\begin{aligned}
RM(r,m) &= RM(r,m-1)\alpha RM(r-1,m-1) \subseteq \{0,1\}^{2^{m-1}}\cdot\{0,1\}^{2^{m-1}} = \{0,1\}^{2^m}\,,\\
d(RM(r,m)) &= \min\{2\cdot d(RM(r,m-1)), d(RM(r-1,m-1))\}\\
&= \min\{2\cdot 2^{m-1-r}, 2^{m-1-(r-1)}\}\\
&= 2^{m-r}\,,\\
dim(RM(r,m)) &= dim(RM(r,m-1)) + dim(RM(r-1,m-1)\\
&= \sum_{i=0}^{r}\binom{m-1}{i} + \sum_{i=0}^{r-1}\binom{m-1}{i}\\
&= \big(1+\sum_{i=1}^{r}\binom{m-1}{i}\big) + \sum_{i=0}^{r-1}\binom{m-1}{i}\\
&= \big(1+\sum_{i=0}^{r-1}\binom{m-1}{i+1}\big) + \sum_{i=0}^{r-1}\binom{m-1}{i}\\
&= 1+\sum_{i=0}^{r-1}\big(\binom{m-1}{i+1}+\binom{m-1}{i}\big)\\
&= 1+\sum_{i=0}^{r-1}\binom{m}{i+1} = 1+\sum_{i=1}^{r}\binom{m}{i}\\
&= \sum_{i=0}^{r}\binom{m}{i}
\end{aligned}
$$

ist.

Für die Reed-Muller-Codes $RM(1,m)\subseteq\{0,1\}^{2^m}$ haben wir

$$dim(RM(1,m)) = \sum_{i=0}^{1}\binom{m}{i} = m+1 \text{ und } d(RM(1,m)) = 2^{m-1}.$$

Damit ergibt sich aus Folgerung 4.4

$$
\begin{aligned}
n(k,d) &\geq \sum_{i=0}^{m}\lceil\frac{d}{2^i}\rceil = \sum_{i=0}^{m}\frac{2^{m-1}}{2^i}\\
&= 2^{m-1}+2^{m-2}+\cdots+2+1+1 = 2^m\,.
\end{aligned}
$$

Da andererseits $RM(1,m) \subseteq \{0,1\}^{2^m}$ gilt, ist damit die Griesmer-Schranke nicht zu verbessern.

In Folgerung 4.5 haben wir eine obere Schranke für die maximale Dimension $k(n,d)$ angegeben. Wir geben abschließend noch eine untere Schranke an, die auf Gilbert und Varshamov zurückgeht.

Satz 4.7 *Wenn drei natürliche Zahlen n, k und d die Bedingungen*

$$k \leq n \quad \textit{und} \quad 2^{n-k} > \sum_{i=0}^{d-2} \binom{n-1}{i}$$

erfüllen, so gibt es einen linearen Code C mit $C \subseteq \{0,1\}^n$, $dim(C) = k$ und $d(C) \geq d$ (es gilt also $k(n,d) \geq k$).

Beweis. Ist $k = n$, also $2^{n-k} = 2^0 = 1$, so muss wegen der zweiten vorausgesetzten Ungleichung $d = 1$ sein. Für die Parameter $n = k$ und $d = 1$ ist $\{0,1\}^n$ ein Code der gewünschten Art.

Es sei also $k < n$. Es sei $v_1, v_2, \ldots, v_{n-k}$ eine Basis von $\{0,1\}^{n-k}$. Ferner seien v_{n-k+1}, $v_{n-k+2}, \ldots, v_{n-k+s}$ weitere Elemente aus $\{0,1\}^{n-k}$ derart, dass je $d-1$ Vektoren linear unabhängig sind. Die Anzahl der Vektoren, die sich als Linearkombination von höchstens $d-2$ Elementen aus $\{v_1, v_2, \ldots, v_{n-k+s}\}$ bilden lassen ist

$$\sum_{i=0}^{d-2} \binom{n-k+s}{i}.$$

Ist diese Summe echt kleiner als 2^{n-k}, so läßt sich in $\{0,1\}^{n-k} \setminus \{v_1, v_2, \ldots, v_{n-k+s}\}$ ein Element $v_{n-k+s+1}$ finden, so dass je $d-1$ Vektoren aus $\{v_1, v_2, \ldots, v_{n-k+s}, v_{n-k-s+1}\}$ linear unabhängig sind. Nach Voraussetzung erhalten wir die Existenz einer Menge $\{v_1, v_2, \ldots, v_n\}$. Aus $v_1, v_2, \ldots, v_n$ bilden wir nun eine Matrix K vom Typ $(n-k, n)$ und verwenden diese als Kontrollmatrix eines Codes C. Dann ist C ein linearer $[n,k]$-Code, dessen Codeabstand nach den Sätzen 4.1 und 4.2 mindestens d ist. □

Kapitel 5

Klassische Verschlüsselungen

Im Rahmen der Codierungstheorie werden vor allem Fragen betrachtet, die daraus resultieren, dass bei der Übertragung ein Kanal benutzt wird. So werden u. a. eindeutige Decodierbarkeit, eine schnelle Übertragung bzw. die Möglichkeiten zur Korrektur von Übertragungsfehlern angestrebt. Eine Geheimhaltung, d. h. die übermittelte Nachricht soll nicht von unbefugten Personen decodiert werden können, ist dagegen kein angestrebtes Ziel.

Die Kryptologie (oder Kryptographie) stellt sich nun gerade dieser Aufgabe. Man ist daran interessiert, eine Codierung einer Nachricht so vorzunehmen, dass folgende Bedingungen erfüllt sind:

- Die Codierung/Verschlüsselung der Nachricht durch den Sender soll einfach sein.
- Die Decodierung/Entschlüsselung der verschlüsselten Nachricht durch den befugten Empfänger soll einfach sein.
- Die Decodierung der verschlüsselten Nachricht durch unbefugte Personen soll unmöglich oder zumindest sehr schwierig sein.

Die Kryptologie hat daher zwei Aspekte. Der eine Aspekt ist der des Entwurfs von Verschlüsselungen, die die oben genannten Bedingungen erfüllen, und der andere Aspekt betrifft die Situation des Kryptoanalysten, der versucht, als unbefugte Person die Entschlüsselung vorzunehmen. Natürlich sind diese Aspekte nicht unabhängig voneinander. So hat man sich schon beim Entwurf zu überlegen, welche Möglichkeiten ein (sehr intelligenter und fähiger) Kryptoanalyst bei der entworfenen Verschlüsselung besitzt, um sein Ziel zu erreichen.

Wir werden uns im Wesentlichen auf Chiffrierungen beschränken, bei denen Wörter über dem Alphabet

$$alph = \{\text{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z}\}$$

der lateinischen Buchstaben wieder auf Wörter über $alph$ abgebildet werden. In einigen Fällen werden wir auch Wörter über dem Alphabet $\{0, 1\}$, d. h. Binärfolgen oder Bit-Folgen, betrachten. In der Regel basiert die Chiffrierung auf einer Abbildungsvorschrift für einzelne Buchstaben, Paare von Buchstaben oder relativ kurze Wörter, die dann auf beliebig lange Wörter (homomorph) fortgesetzt werden kann. Diese Abbildungsvorschrift

ist in den meisten Fällen einfach zu realisieren, wenn man ein hierzu gehöriges Grundelement, den sogenannten Schlüssel, kennt. Auch die Dechiffrierung ist bei Kenntnis des Schlüssels einfach. Ist dagegen der Schlüssel nicht bekannt, so sind sehr viele mögliche Abbildungsvorschriften als Grundlage der Chiffrierung möglich, wodurch der Kryptoanalyst vor einer schwierigen Aufgabe steht.

Da die Schlüssel nur den befugten Personen bekannt sein sollen, müssen sie „merkbar“ sein. Insbesondere müssen sie eine sehr viel kürzer Beschreibung haben, als die Nachricht selbst (sonst wäre die Geheimhaltung des Schlüssels nicht einfacher als die der unverschlüsselten Nachricht). Besonders in der Vergangenheit bestanden die Schlüssel aus einer bzw. wenigen Zahlen oder einem Wort bzw. einem leicht merkbaren kurzen Text.

In der Folge werden wir den unverschlüsselten Text, der vom Sender an den Empfänger geschickt wird als Klartext bezeichnen. Unter dem Kryptotext verstehen wir den Text, der aus dem Klartext durch die Verschlüsselung/Chiffrierung entsteht.

Wir behandeln als erstes einige klasssische Chiffren, die teilweise schon seit Zeiten des Altertums benutzt wurden, aber teiweise bis die Neuzeit noch angewendet wurden.

5.1 Monoalphabetische Substitutionschiffren

Als erstes Beispiel für eine Chiffrierung wollen wir Verschiebungschiffren behandeln, die schon im Altertum benutzt wurden. In den Schriften von Sueton wird berichtet, dass Caesar (100 - 44 v.Chr.) diese Methode zur Verschlüsselung seiner geheimen Nachrichten benutzt hat. Daher werden sie manchmal auch als Caesar-Chiffren bezeichnet. Verschiebungschiffren werden durch zyklische Verschiebung der Buchstaben des Alphabets entsprechend ihrer Ordnung erzeugt. Formal ergibt sich folgendes Vorgehen.

Wir definieren die Funktion φ, die die Buchstaben des Alphabets in eineindeutiger Weise Zahlen zwischen 0 und 25 zuordnet, entsprechend der Tabelle in Abbildung 5.1. Als

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

Abbildung 5.1: Funktion φ

Schlüssel wählen wir eine Zahl i mit $0 \leq i \leq 25$. Wir definieren dann die Chiffriervorschrift $v_i : alph \to alph$ der i-ten Verschiebungschiffre durch

$$v_i(\alpha) = \varphi^{-1}(\varphi(\alpha) + i \bmod 26)) \,.$$

Bei Wahl von $i = 5$ erhalten wir aus dem Klartext MAGDEBURG den Kryptotext RFLIJGZWL.

Für den Schlüssel i, $0 \leq i \leq 25$, ist die Vorschrift für das Dechiffrieren dann offensichtlich durch die Funktion dv_i zu bewerkstelligen, die durch

$$dv_i(\alpha) = \varphi^{-1}(\varphi(\alpha) - i \bmod 26) = \varphi^{-1}(\varphi(\alpha) + (26 - i) \bmod 26)$$

(wir benutzen auch zur Bezeichnung der Äquivalenz $\bmod 26$ das Gieichheitszeichen) definiert ist.

Offenbar gibt es 26 verschiedene Verschiebungschiffren.

Wir betrachten nun den Kryptoanalysten. Liegt ein Paar *(Klartext, Kryptotext)* vor, so ist seine Aufgabe sehr leicht zu lösen. Er betrachtet nur den ersten Buchstaben α des Klartexts und den ersten Buchstaben β des Kryptotextes. Dann ergibt sich der verwendete Schlüssel i der Wert $\varphi(\beta) - \varphi(\alpha)$.

Ist nur ein Kryptotext gegeben, so könnte der Kryptoanalyst alle 26 möglichen Schlüssel und die zugehörigen Dechiffrierungen durchtesten und dabei sicher auch den Klartext finden. Es ist jedoch im Allgemeinen nicht notwendig alle 26 Möglichkeiten zu testen, wenn zusätzlich die Häufigkeit des Vorkommen der Buchstaben im Text berücksichtigt wird.

Für deutsche Texte gilt die in Abbildung 5.2 angegebene Wahrscheinlichkeitsverteilung der Buchstaben.[1]

α	A	B	C	D	E	F	G	H	I
$p(\alpha)$	6,51	1,89	3,06	5,08	17,40	1,66	3,01	4,76	7,55
α	J	K	L	M	N	O	P	Q	R
$p(\alpha)$	0,27	1,21	3,44	2,53	9,78	2,51	0,79	0,02	7,00
α	S	T	U	V	W	X	Y	Z	
$p(\alpha)$	7,27	6,15	4,35	0,67	1,89	0,03	0,04	1,13	

Abbildung 5.2: Wahrscheinlichkeitsverteilung der Buchstaben in deutschen Texten

Der Kryptoanalyst ermittelt zuerst die Wahrscheinlichkeiten des Auftretens der einzelnen Buchstaben im Text. Er geht dann davon aus, dass der im Kryptotext am häufigsten auftretende Buchstabe α_1 dem Buchstaben E entspricht. Er nimmt also die Entschlüsselung mit $i = \varphi(\alpha) - 4$ vor. Ergibt sich hierbei als vermuteter Klartext kein echter deutscher Text, so wird als nächstes angenommen, dass der zweithäufigste Buchstabe α_2 dem Buchstaben E entspricht. Ergibt sich hier kein brauchbares Resultat, so wird mit dem dritthäufigsten Buchstaben des Textes getestet usw.

Als Beispiel betrachten wir den Kryptotext

C T J C J C S C T J C O X V A J U I Q P A A D C H K D C C C T C P X H I V J I

Die Häufigkeit des Vorkommens von Buchstaben im Kryptotext ergibt sich wie folgt:

[1] Die Wahrscheinlichkeitsverteilung ist nicht eindeutig zu ermitteln; sie hängt entscheidend davon ab, welche Texte analysiert wurden. Daher differieren auch die in den Büchern angegebenen Verteilungen leicht. Jedoch ist die Reihenfolge hinsichtlich der Häufigkeit auf den ersten sechs Plätzen stets gleich.

9-mal : C
5-mal : J
3-mal : A, I, T
2-mal : D, H, P, V, X
1-mal : K, N, O, Q, S
0-mal : B, E, F, G, L, M, R, W, Y, Z

Die Annahme, dass C als Originalbuchstaben E hat, liefert den Text

E V L E L E U E V L E Q Z X C L W K S R C C F E J M F E E V E R Z J K X L K

und die Annahme, dass J dem E entspricht führt zu

X O E X E X N X O E X J S Q V E P D L K V V Y X C F Y X X O X K S C D Q E D

Erst wenn der Kryptologe T als den E entsprechenden Buchstaben testet, erhält er den brauchbaren Text

N E U N U N D N E U N Z I G L U F T B A L L O N S V O N N E N A I S T G U T

(oder lesbarer NEUNUNDNEUNZIG LUFTBALLONS VON NENA IST GUT).

Jede Verschiebungschiffre v_i, $0 \leq i \leq 25$, stellt eine Permutation der Buchstaben dar und bei der Chiffrierung wird jeder Buchstabe immer durch den gleichen Buchstaben ersetzt. Chiffren mit dieser Eigenschaft nennen wir *monoalphabetische Subsitutionschiffren*. Bei der Substitution muss es sich selbstverständlich um eine Permutation handeln, da sonst keine eindeutige Dechiffrierung möglich wäre.

Die Verschiebungschiffren haben den Vorteil, dass man sich als Schlüssel nur eine Zahl merken muss (wir gehen natürlich davon aus, dass man die alphabetische Reihenfolge der Buchstaben kennt). Eine komplizierte Permutation kann man sich dagegen kaum merken, d. h. sie muss irgendwo notiert werden, wodurch Unbefugten ein Zugriff unter Umständen möglich wäre. Jedoch gibt es Permutationen, die man sich sehr leicht merken kann. Ein derartiges Beispiel kann wie folgt gebildet werden. Wir wählen als Schlüssel ein (möglichst langes Wort, in dem kein Buchstabe doppelt vorkommt. Dieses Wort schreiben wir dann zuerst hin und lassen ihm in umgekehrter alphabetischer Reihenfolge die Buchstaben folgen, die im Wort nicht vorkommen. Ausgehend vom Schlüsselwort GREIFSWALD erhalten wir die Permutation

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

Aus dem Klartext MAGDEBURG entsteht hierbei der Kryptotext XGWIFRMPW.

Eine weitere Methode, komplizierte Permutationen zu erhalten, sind *affine* Chiffren. Hierbei werden als Schlüssel zwei natürliche Zahlen a und b so gewählt, dass $0 \leq a \leq 25$ und $0 \leq b \leq 25$ gelten und a und 26 den größten gemeinsamen Teiler 1 haben. Dann wird die Funktion $v_{(a,b)} : alph \to alph$ durch

$$v_{(a,b)}(\alpha) = \varphi^{-1}(a \cdot \varphi(\alpha) + b \bmod 26)$$

definiert. Die Verschiebungschiffren sind damit die affinen Chiffren, bei denen $a = 1$ ist.

α	A	B	C	D	E	F	G	H	I	J	K	L	M
$\varphi(\alpha)$	0	1	2	3	4	5	6	7	8	9	10	11	12
$a \cdot \varphi(\alpha) + b \bmod 26$	5	8	11	14	17	20	23	0	3	6	9	12	15
$v_{(3,5)}(\alpha)$	F	I	L	O	R	U	X	A	D	G	J	M	P
α	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
$\varphi(\alpha)$	13	14	15	16	17	18	19	20	21	22	23	24	25
$a \cdot \varphi(\alpha) + b \bmod 26$	18	21	24	1	4	7	10	13	16	19	22	25	2
$v_{(3,5)}(\alpha)$	S	V	Y	B	E	H	K	N	Q	T	W	Z	C

Abbildung 5.3: Beispiel einer affinen Chiffrierung mit $a = 3$ und $b = 5$

Für die Werte $a = 3$ und $b = 5$ erhalten wir die Zuordnungen aus Abbildung 5.3. Daraus ergibt sich für den Klartext MAGDEBURG der Kryptotext PFXORINEX.

Die Forderung, dass a und 26 den größten Teiler 1 haben sollen, wird aus drei Gründen erhoben:

- Wir erreichen dadurch eine eineindeutige Funktion. Zum Beispiel besteht bei Verwendung von $a = 10$ und $b = 1$ die Gleichheit $10 \cdot 0 + 1 = 10 \cdot 13 + 1 = 1 \bmod 26$, was bedeutet, dass A und N beide durch den gleichen Buchstaben B verschlüsselt werden müssten.

- Es wird abgesichert, dass $v_{(a,b)}$ eine Abbildung auf *alph* ist. Bei $a = 10$ und $b = 1$ würde kein Buchstabe durch O chiffriert werden, da es keine Zahl z mit $10z + 1 = 16 \bmod 26$ gibt, wie man leicht nachrechnet.

- Für a existiert ein inverses Element, d. h. es gibt ein a^{-1}, $0 \leq a^{-1} \leq 25$, mit $a \cdot a^{-1} = 1 \bmod 26$. Für $a = 3$ ist dieser Wert z.B. 9, da $3 \cdot 9 = 27 = 1 \bmod 26$ ist. Damit ergibt sich für die Dechiffrierung eines Buchstabens β der Buchstabe

  $$\varphi^{-1}(a^{-1}(\varphi(\beta) - b)) .$$

Alle monoalphabetischen Subsitutionschiffren lassen sich aber vom Kryptoanalysten unter Verwendung der Häufigkeitsverteilung von Buchstaben und von Paaren (und Tripeln) von Buchstaben (die ebenfalls bestimmt wurden) ermitteln. Er hat jetzt im Wesentlichen nur mehrere der häufig auftretenden Buchstaben gleichzeitig zu betrachten, da die Kenntnis der richtigen Zuordnung eines Buchstaben jetzt nicht mehr ausreicht, um die anderen Zuordnungen zu ermitteln. Daher kann der Kryptoanalyst mittels der Trial-and-error-Methode den Klartext zu (langen) Kryptotexten ermitteln, ohne alle 26! Permutationen auszuprobieren.

5.2 Polyalphabetische Substitutionschiffren

Um dem Kryptoanalysten die Arbeit zu erschweren ist es also erforderlich, die Häufigkeitsverteilung zu verschleiern. Dies kann zum Beispiel dadurch geschehen, dass bei der

Substitution einem Buchstaben nicht stets der gleiche Wert zugeordnet wird. Die Substitutionen heißen daher *polyalphabetische Chiffren*. Wir betrachten hier drei Beispiele.

Unser erstes Beispiel sind HILL-Chiffren. Bei diesen wird als Schlüssel eine Matrix M vom Typ (2,2) gewählt, deren Elemente a_{ij}, $i,j \in \{1,2\}$, die Bedingung $0 \leq a_{ij} \leq 25$ erfüllen und für die es eine inverse Matrix M^{-1} gibt (für die

$$M \cdot M^{-1} = M^{-1} \cdot M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \bmod 26$$

gilt). Wir unterteilen den Text in Teilwörter der Länge 2 und verschlüsseln ein Wort $\alpha\beta$ durch $\alpha'\beta'$, wobei folgende Bedingungen gelten:

$$M \cdot \begin{pmatrix} \varphi(\alpha) \\ \varphi(\beta) \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \bmod 26\,, \quad \alpha' = \varphi^{-1}(x)\,, \quad \beta' == \varphi^{-1}(y)\,,$$

d. h. wir bilden zuerst einen Spaltenvektor mit den zu α und β gehörenden Zahlen, multiplizieren diesen Vektor mit der Schlüsselmatrix M und überführen die beiden erhaltenen Komponenten des Vektors wieder in Buchstaben. Die Dechiffrierung erfolgt jetzt genauso, nur dass anstelle von M die Matrix M^{-1} verwendet wird.

Als Beipiel betrachten wir die Matrix

$$M = \begin{pmatrix} 3 & 5 \\ 3 & 24 \end{pmatrix}$$

mit der inversen Matrix $M^{-1} = \begin{pmatrix} 10 & 25 \\ 15 & 11 \end{pmatrix}$. Wir wollen das Wort SAHARA verschlüsseln. Den zugehörigen Teilwörtern SA, HA und RA entsprechen die Vektoren $(18,0)^T$, $(7,0)^T$ und $(17,0)^T$, woraus sich wegen

$$\begin{pmatrix} 3 & 5 \\ 3 & 24 \end{pmatrix} \begin{pmatrix} 18 \\ 0 \end{pmatrix} = \begin{pmatrix} 54 \\ 54 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \bmod 26\,,$$
$$\begin{pmatrix} 3 & 5 \\ 3 & 24 \end{pmatrix} \begin{pmatrix} 7 \\ 0 \end{pmatrix} = \begin{pmatrix} 21 \\ 21 \end{pmatrix}\,,$$
$$\begin{pmatrix} 3 & 5 \\ 3 & 24 \end{pmatrix} \begin{pmatrix} 17 \\ 0 \end{pmatrix} = \begin{pmatrix} 51 \\ 51 \end{pmatrix} = \begin{pmatrix} 25 \\ 25 \end{pmatrix} \bmod 26$$

als Kryptotext das Wort CCVVZZ ergibt. Man erkennt, dass sich zum einen der Buchstabe A bei allen drei Vorkommen durch verschiedene Buchstaben verschlüsselt wird und zum anderen verschiedene Buchstaben wie z.B. S und A durch den gleichen Buchstaben verschlüsselt werden. Dadurch wird die Häufigkeit des Auftretens von Buchstaben stark verschleiert und der Kryptoanalyst kann nicht wie bei monoalphabetischen Substitutionen aus der Häufigkeit der Buchstaben Rückschlüsse auf die Chiffre ziehen.

Weiterhin wollen wir den Kryptotext HHTQ entschlüsseln. Wir erhalten die Vektoren $(7,7)^T$ zu HH und $(19,15)^T$ zu TQ und unter Verwendung von M^{-1}

$$\begin{pmatrix} 10 & 25 \\ 15 & 11 \end{pmatrix} \begin{pmatrix} 7 \\ 7 \end{pmatrix} = \begin{pmatrix} 11 \\ 0 \end{pmatrix} \bmod 26 \text{ und } \begin{pmatrix} 10 & 25 \\ 15 & 11 \end{pmatrix} \begin{pmatrix} 19 \\ 16 \end{pmatrix} = \begin{pmatrix} 18 \\ 19 \end{pmatrix} \bmod 26\,,$$

woraus der Klartext LAST resultiert.

Wir betrachten nun die Situation des Kryptoanalysten. Wenn er einen Klartext wählen und zu diesem den Kryptotext erhalten kann, so ist die Aufgabe für ihn einfach. Er wählt den kurzen Klartext HELP mit den zugehörigen Vektoren $(7,4)^T$ und $(11,15)^T$ und erhält den Kryptotext $\alpha\beta\gamma\delta$. Daraus erhält er die Gleichungen

$$N \cdot \begin{pmatrix} 7 \\ 4 \end{pmatrix} = \begin{pmatrix} \varphi(\alpha) \\ \varphi(\beta) \end{pmatrix} \quad \text{und} \quad N \cdot \begin{pmatrix} 11 \\ 15 \end{pmatrix} = \begin{pmatrix} \varphi(\gamma) \\ \varphi(\delta) \end{pmatrix}$$

oder in anderer Schreibweise

$$N \cdot \begin{pmatrix} 7 & 11 \\ 4 & 15 \end{pmatrix} = \begin{pmatrix} \varphi(\alpha) & \varphi(\gamma) \\ \varphi(\beta) & \varphi(\delta) \end{pmatrix}$$

für die Schlüsselmatrix N. Die Wahl HELP wurde deshalb vorgenommen, weil die entstandene Matrix $U = \begin{pmatrix} 7 & 11 \\ 4 & 15 \end{pmatrix}$ eine inverse Matrix U^{-1} besitzt, für die sich $U^{-1} = \begin{pmatrix} 19 & 19 \\ 14 & 21 \end{pmatrix}$ ergibt. Damit erhalten wir

$$N = N \cdot (U \cdot U^{-1}) = (N \cdot U) \cdot U^{-1} = \begin{pmatrix} \varphi(\alpha) & \varphi(\gamma) \\ \varphi(\beta) & \varphi(\delta) \end{pmatrix} \cdot \begin{pmatrix} 19 & 19 \\ 14 & 21 \end{pmatrix} .$$

Der Kryptoanalyst kann also die Schlüsselmatrix N berechnen.

Die Situation ändert sich schon, wenn er nur ein Paar *(Klartext, Kryptotext)* hat. Nehmen wir an, man gibt ihm das oben bestimmte Paar (SAHARA,CCVVZZ) und HHTQ als weiteren Kryptotext. Dem Kryptoanalysten ist die Schlüsselmatrix $N = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ nicht bekannt. Da das ersten Teilwort SA der Länge 2 durch BB verschlüsselt wird, ergibt sich

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 18 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} ,$$

woraus die Gleichungen

$$18 \cdot a = 2 \quad \text{und} \quad 18 \cdot c = 2$$

resultieren. Die möglichen Lösungen müssen $a, c \in \{3, 16\}$ erfüllen. Betrachten wir nun das Paar HA, das durch VV verschlüsselt wird. Der Vektor $(7,0)^T$ muss also in $(21,21)^T$ überführt werden. Da $3 \cdot 7 = 21$ und $16 \cdot 7 = 112 = 8 \bmod 26$ gelten, bleiben nur noch die Möglichkeiten $a = c = 3$ übrig. Das Paar RA liefert keine weiteren Informationen, da $\begin{pmatrix} 3 & b \\ 3 & d \end{pmatrix} \begin{pmatrix} 17 \\ 0 \end{pmatrix} = \begin{pmatrix} 25 \\ 25 \end{pmatrix} \bmod 26$ gilt. Somit bleiben für die Entschlüsselung des Kryptotext HHTQ mit den Vektoren $(7,7)^T$ zu HH und $(19,16)^T$ zu TQ die Gleichungen

$$\begin{pmatrix} 3 & b \\ 3 & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 7 \\ 7 \end{pmatrix} \text{ und } \begin{pmatrix} 3 & b \\ 3 & d \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 19 \\ 16 \end{pmatrix}$$

zu lösen. Der Kryptoanalyst muss also aus vier Gleichungen sechs Unbekannte bestimmen; dabei hat er noch die Zusatzinformation, dass $\begin{pmatrix} 3 & b \\ 3 & d \end{pmatrix}$ eine inverse Matrix besitzen muss.

Letzte Bedingung ist erfüllt, wenn er $b = 2$ und $d = 1$ wählt, da die Zeilen der Matrix offensichtlich linear unabhängig sind. Damit ergeben sich die Gleichungen

$$\begin{pmatrix} 3 & 2 \\ 3 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 7 \\ 7 \end{pmatrix} \text{ und } \begin{pmatrix} 3 & 2 \\ 3 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 19 \\ 16 \end{pmatrix},$$

deren Lösungen

$$x = 11 \text{ und } y = 0 \quad \text{bzw.} \quad x' = 13 \text{ und } y' = 3$$

sind. Damit ergibt sich der Klartext LAND. Da dies ein richtiges deutsches Wort ist, vermutet der Kryptograph, dass der Klartext LAND ist. Er irrt aber, denn die obige Entschlüsselung mit der richtigen Schlüsselmatrix M ergibt LAST.

Unser zweites Beispiel für polyalphabetische Chiffrierungen sind die *Fairplay-Chiffren*. Bei diesen Chiffren wählen wir 25 Buchstaben des Alphabets, d. h. wir lassen einen selten vorkommenden Buchstaben fort, und ordnen diese in einer Matrix mit 5 Zeilen und 5 Spalten an. Um sich diese Anordnung zu merken, kann man wie bei der Konstruktion merkbarer Permutationen vorgehen. Man merkt sich ein Schlüsselwort und ordnet die verbleibenden Buchstaben in alphabetischer oder umgekehrter alphabetischer Reihenfolge an.

Bei Fortlassung von J, erneuter Wahl von GREIFSWALD und alphabetischer Anordnung der restlichen Buchstaben ergibt sich die Fairplay-Matrix

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

Die Verschlüsselung wird wie folgt vorgenommen. Wir unterteilen den Text erneut in Paare und verlangen, dass keines dieser Paare aus zwei gleichen Buchstaben besteht. Sollte dies der Fall sein, so fügen wir ein Vorkommen von Q ein. Sollte der dadurch entstehende Text nicht gerade Länge haben, fügen wir ein Q am Ende hinzu. Nun verschlüsseln wir den Text, indem wir die Paare des Klartextes durch Buchstabenpaare verschlüsseln. Sei $\alpha\beta$ ein Buchstabenpaar. Wir ersetzen es durch das Paar α', β', das wie folgt ermittelt wird:

1. Falls α und β in einer Zeile zu finden sind, so sind α' und β' die Buchstaben, die auf α bzw. β zyklisch in der Zeile folgen.

2. Falls α und β in einer Spalte zu finden sind, so sind α' und β' die Buchstaben, die auf α bzw. β zyklisch in der Spalte folgen.

3. Falls sich α und β sowohl in verschiedenen Zeilen als auch verschiedenen Spalten befinden, so wählen wir für α' und β' die Buchstaben so, dass

$$\begin{pmatrix} \alpha & \dots & \alpha' \\ \vdots & & \vdots \\ \beta' & \dots & \beta \end{pmatrix} \text{ oder } \begin{pmatrix} \alpha' & \dots & \alpha \\ \vdots & & \vdots \\ \beta & \dots & \beta' \end{pmatrix} \text{ oder } \begin{pmatrix} \beta & \dots & \beta' \\ \vdots & & \vdots \\ \alpha' & \dots & \alpha \end{pmatrix} \text{ oder } \begin{pmatrix} \beta' & \dots & \beta \\ \vdots & & \vdots \\ \alpha & \dots & \alpha' \end{pmatrix}$$

eine Teilmatrix der Fairplay-Matrix bilden.

Auf diese Art verschlüsseln wir unter Verwendung obiger Fairplay-Matrix das Wort SAHARAWIND durch WLPHEWLPTS. Man erkennt sofort, dass bei der Verschlüsselung für A drei verschiedene Buchstaben (L,H und W) benutzt werden.

Der befugte Empfänger kann mit der Kenntnis der Fairplay-Matrix leicht die Dechiffrierung vornehmen. Stehen die empfangene Buchstaben $\alpha'\beta'$ in einer Zeile, so stehen die Originalbuchstaben α und β in der Fairplay-Matrix in der Zeile zyklisch vor α' bzw. β'. Stehen beide Buchstaben in einer Spalte, so gehen wir analog in der Spalte vor, um α und β zu ermitteln. Stehen α' und β' weder in einer Spalte noch in einer Zeile, so finden wir α und β entsprechend Punkt 3 der Verschlüsselung.

Die Aufgabe des Kryptoanalysten ist nicht einfach. Während des Zweiten Weltkrieges hat das britische Militär Fairplay-Chiffrierungen verwendet, und diese wurden von der gegnerischen Seite nicht geknackt.

Als letztes Beispiel für polyalphabetische Substitutionschiffren behandeln wir die Vigenère-Chiffren.[2] Sie bestehen im Wesentlichen in einer periodischen Anwendung von Verschiebungschiffren.

Zuerst wird ein Schlüsselwort $\alpha_1\alpha_2\ldots\alpha_n$ gewählt. Ferner sei der Klartext $t_1t_2\ldots t_m$ gegeben. Dann verschlüsseln wir den Buchstaben t_i durch die Verschiebungschiffre $v_{\varphi(\alpha_j)}$ mit

$$j = \begin{cases} i \bmod n & i \text{ ist kein Vielfaches von } n \\ n & \text{sonst.} \end{cases}$$

Wenn wir die Tabelle aus Abbildung 5.4 verwenden, so bedeutet dies, dass wir den Buchstaben t_i durch sein Bild in der Zeile zum Buchstaben α_j mit $1 \leq j \leq n$ und $i = j + kn$ für ein gewisses $k \geq 0$ verschlüsseln. Wir wenden also die Verschiebungschiffren, die zu den Buchstaben des Schlüsselwortes gehören, periodisch an.

Unter Verwendung des Schlüsselwortes ABEND erhalten wir für den Klartext SAHARAWIND den Kryptotext SBLNUAXMAG. Erneut wird der Buchstabe A bei jedem seiner Vorkommen anders verschlüsselt.

Um den empfangenen Kryptotext zu dechiffrieren, hat der (befugte) Empfänger nur die Entschlüsselungen zu den Verschiebungschiffren ebenfalls periodisch anzuwenden. Bei Kenntnis des Schlüsselwortes ist dies eine leichte Aufgabe.

Dem Kryptoanalalysten möge ein Kryptotext $\beta_1\beta_2\ldots\beta_m$ vorliegen. Der Kryptoanalyst hat für die Entschlüsselung eigentlich das Schlüsselwort zu ermitteln. Ihm reicht es aber, die Länge des Schlüsselwortes zu bestimmen. Hat er diese mit n ermittelt, so weiß er, dass die Buchstaben $s_is_{i+n}s_{i+2n}\ldots s_{i+un}$, wobei $1 \leq i \leq n$ und $i + un \leq m < i + (u+1)n$ gelten, durch die gleiche Verschiebungschiffre gewonnen wurden. Durch eine Analyse der Häufigkeiten der Buchstaben, kann er nun sehr wahrscheinliche Kandidaten $v_{i,1}v_{i,2},\ldots v_{i,k_i}$ für diese Chiffre bestimmen. Durch Durchtesten dieser Kandidaten für $1 \leq i \leq n$ gelingt es ihm dann das Schlüsselwort zu erhalten und damit kann er dann in gleicher Weise wie

[2]Sie wurden nach dem französischen Diplomaten Blaise de Vigenère (1523–1596) benannt, der diese Verschlüsselungen erstmals 1586 benutzte.

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

Abbildung 5.4: Vigenère-Tabelle

der Empfänger die Entschlüsselung (auch neuer Kryptotexte) vornehmen. Daher ist das zentrale Problem für den Kryptoanalysten die Bestimmung der Länge des Schlüsselwortes.

Durch den deutschen Kryptoanalysten F.W. Kasinski wurde um 1860 vorgeschlagen, Teilwörter der Länge ≥ 3 zu suchen, die im Kryptotext mehrfach vorkommen. Dabei wird davon ausgegangen, dass bei einer Länge ≥ 3 eine gleiche Chiffrierung mit großer Wahrscheinlichkeit nur dann vorliegt, wenn das gleiche Wort des Klartext verschlüsselt wurde. Folglich ist die Differenz des Auftretens der gleichen Wörter ein Vielfaches der Schlüsselwortlänge.

Als Beispiel betrachten wir den Kryptotext aus Abbildung 5.5, in dem die mehrfach auftretende Teilwörter der Länge ≥ 3 unterstrichen sind.. Man erhält daraus die folgende Tabelle:

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

Abbildung 5.5: Ein Kryptotext mit mehrfach auftretenden Teilwörtern der Länge ≥ 3

Folge	Abstand
JTD	$50 = 2 \cdot 5^2$
VIQM	$265 = 5 \cdot 53$
TDMHZGNMWK	$90 = 2 \cdot 3^2 \cdot 5$
MWK	$75 = 3 \cdot 5^2$
ZHUM	$40 = 2^3 \cdot 5$
KAH	$128 = 2^7$

Nimmt man den größten gemeinsamen Teiler der Abstände so ergibt sich 1. Wenn das Schlüsselwort aber die Länge 1 hat, so erfolgt die Verschlüsselung durch eine einfache Verschiebechiffre. Dies ist aber nicht anzunehmen, da eine Vigenère-Chiffre vorliegen soll. Es ist also anzunehmen, dass eines der Teilwörter nicht durch Verschlüsselung des gleichen Wortes sondern zufällig entstanden ist. Hierfür ist KAH der erste Kandidat, da die Primfaktoren des zu diesem Wort gehörenden Abstands beide nicht Primfaktoren anderer Abstände sind. Ausgehend von den verbleibenden Wörtern ist ein Schlüsselwort der Länge 5 zu erwarten, da nun 5 der größte gemeinsame Teiler ist.

Eine andere Methode zur Berechnung der Länge des Schlüsselwortes stammt von William Friedman, der sie 1925 vorschlug. Hierbei wird die Länge nur approximiert, aber dadurch werden gewisse Werte weitgehend ausgeschlossen.

Gegeben sei ein Kryptotext der Länge n. Für einen Buchstaben α, sei n_α die Anzahl seines Auftretens im gegebenen Kryptotext. Wir bestimmen nun die Anzahl des Auftretens von Zweiermengen nicht notwendig benachbarter gleicher Buchstaben. Für einen Buchstaben α ist diese Zahl durch $n_\alpha(n_\alpha - 1)/2$ gegeben, da der erste Buchstabe an n_α Stellen und der zweite Buchstabe an $n_\alpha - 1$ Stellen auftauchen kann und die Reihenfolge keine Rolle spielt. Für die Anzahl des Auftretens von Paaren gleicher Buchstaben im

Kryptotext erhalten wir daher

$$I = \sum_{\alpha \in alph} \frac{n_\alpha(n_\alpha - 1)}{2}, .$$

Nehmen wir nun den Klartext. In ihm taucht der Buchstabe α mit der Häufigkeit p_α entsprechend der Abbildung 5.2 auf. Tritt ein Buchstabe α darin mit der Häufigkeit p_α auf (man beachte, dass wir diese Häufigkeit nicht kennen, da wir den Text nicht kennen) so hat die Wahrscheinlichkeit für das Auftretens eines Paares des Buchstaben α den Wert p_α^2 (dies ist eigentlich nur bei Beachtung der Reihenfolge in den Paaren richtig, aber bei großem n kann dieser Unterschied vernachlässigt werden) und damit für das Auftreten eines Paares $\sum_{\alpha \in alph} p_\alpha^2$. Falls die Buchstaben so verteilt sind, wie dies nach Abbildung 5.2 bei deutschen Texten der Fall ist, so ergibt sich für die Wahrscheinlichkeit $\sum_{\alpha \in alph} p_\alpha^2 = 0,0762$. Sind die Buchstaben dagegen im Klartext gleichverteilt, so wäre p_α =1/26 für alle α und damit $\sum_{\alpha \in alph} p_\alpha^2 = 1/26$=0,0385.

Nehmen wir nun an, dass das Schlüsselwort der Vigenère-Chiffre die Länge l hat. Wir schreiben den Text nun so, dass jede Zeile (vielleicht bis auf die letzte) genau l Buchstaben enthält. In jeder der so entstehenden Spalten stehen daher n/l Buchstaben (wir nehmen hierfür Ganzzahligkeit an, was bei hinreichender Länge n gemacht werden kann). Wir betrachten nun die Häufigkeit für das Auftreten eines Paaren von Buchstaben. Wir wählen dazu eine Position aus. Damit ist auch eine Spalte gewählt. Innerhalb der Spalte können wir noch $\frac{n}{l} - 1$ Positionen für den zweiten Buchstaben auswählen. Da es auf die Reihenfolge nicht ankommt, liefert dies

$$\frac{n \cdot (\frac{n}{l} - 1)}{2} = \frac{n(n-l)}{2l}$$

Möglichkeiten. Für die Anzahl der Paare mit dem zweiten Buchstaben in einer anderen Spalte ergeben sich (da nun die n/l Positionen der ersten Spalte entfallen)

$$\frac{n \cdot (n - \frac{n}{l})}{2} = \frac{n^2(l-1)}{2l}$$

Möglichkeiten.

Die Buchstaben einer Spalten werden durch die gleiche Verschiebechiffre geliefert. Die Buchstabenverteilung der Spalten des Kryptotextes entspricht daher der des Klartextes, da bei einer Verschiebechiffre die Zuordnung der Buchstaben eineindeutig ist. Für das Auftreten eines Paares gleicher Buchstaben in einer Spalte erhalten wir daher (bei einem deutschen Text) $0,0762 \cdot \frac{n(n-l)}{2l}$ Möglichkeiten. Entstammen die Buchstaben dagegen verschiedenen Spalten, so können wir davon ausgehen, dass sie relativ gleichverteilt sind (und bei Gleichverteilung der Buchstaben im Schlüsselwort wäre Gleichverteilung im Kryptotext auch gegeben). Daher gibt $0,0385 \cdot \frac{n^2(l-1)}{2l}$ die Anzahl der Möglichkeiten für das Auftreten eines Paares gleicher Buchstaben in verschiedenen Spalten. Damit ist die Gesamtzahl der zu erwartenden Paare gleicher Buchstaben im Kryptotext

$$I' = 0,0762 \cdot \frac{n(n-l)}{2l} + 0,0385 \cdot \frac{n^2(l-1)}{2l} .$$

Offensichtlich ist zumindest angenähert $I = I'$ zu erwarten. Damit erhalten wir

$$I = 0,0762 \cdot \frac{n(n-l)}{2l} + 0,0385 \cdot \frac{n^2(l-1)}{2l}$$

und daraus

$$l = \frac{0,0377n}{2I - 0,0385 \cdot n^2 + 0,0762 \cdot n}\,.$$

Da I durch Auszählen am Kryptotext ermittelt werden kann, ist somit l berechenbar.

Für unseren Beispieltext aus Abbildung 5.5 erhalten wir $I = 5924$ und damit $l = 6,5$. Somit ist die Länge des Schlüsselwortes in der Nähe von $6,5$ zu suchen. Unter Berücksichtigung unserer Ergebnisse aus der Methode von KASISKI liegt damit 5 als Länge des Schlüsselwortes nahe. Wir bestimmen daher für i mit $0 \leq i \leq 4$, die am häufigsten auftauchenden Buchstaben in allen Spalten mit einer Nummer k mit $k = i \bmod 5$. Hierfür ergeben sich V, E, H, M und S. Damit entsprechen diese Buchstaben bei den fünf Verschiebechiffren jeweils dem Buchstaben E. Hieraus resultiert das Schlüsselwort RADIO. Nimmt man nun hiermit die Dechiffrierung vor, so ergibt sich der Text

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

oder in lesbarerer Form

> DEN HÖCHSTEN ORGANISATIONSSTAND ERFUHR DIE KRYPTOLOGIE IN VENEDIG, WO SIE IN FORM EINER STAATLICHEN BÜROTÄTIGKEIT AUSGEÜBT WURDE. ES GAB SCHLÜSSELSEKRETÄRE, DIE IHR BÜRO IM DOGENPALAST HATTEN UND FÜR IHRE TÄTIGKEIT RUND ZEHN DUKATEN IM MONAT BEKAMEN. ES WURDE DAFÜR GESORGT, DASS SIE WÄHREND IHRER ARBEIT NICHT GESTÖRT WURDEN. SIE DURFTEN IHRE BÜROS ABER AUCH NICHT VERLASSEN, BEVOR SIE EINE GESTELLTE AUFGABE GELÖST HATTEN.

5.3 Der Data Encryption Standard

In diesem Abschnitt behandeln wir den Data Encryption Standard, der 1977 vom National Bureau of Standards in den USA angekündigt und von IBM realisiert wurde. Hierbei geht es darum 64-Bit-Folgen (Wörter der Länge 64 über $\{0,1\}$) zu übertragen und als Schlüssel dient dabei eine (geheimzuhaltende) 56-Bit-Folge. Die Verschlüsselung ist dann standardisiert. Dafür wurden sehr effektive Hardware- und Softwarerealisierungen geschaffen. Die Methode erfuhr seit den siebziger Jahren einige Veränderungen. Wir werden hier aber als Prototyp die Originalvariante behandeln.

Die Chiffrierung ist ein iterativer Prozess mit 16 Iterationsschritten. Innerhalb dieser Schritte werden zwei wesentliche Operationen benutzt. Die erste Operation ist die Überführung der Wörter $a_1a_2\dots a_k$ und $b_1b_2\dots b_k$ in das Wort $(a_1 \oplus b_1)(a_2 \oplus b_2)\dots(a_k \oplus b_k)$. Die andere besteht in der Transformation des Wortes $a_1a_2\dots a_k$ in das Wort $a_{i_1}a_{i_2}\dots a_{i_l}$, wobei wir dann stets nur die Folge $i_1, i_2, \dots, i_l$ angeben, die eine Auswahl (meist eine Permutation) von der Folge $1, 2, \dots, k$ ist.

Der Algorithmus zur Verschlüsselung des Klartextes in den Kryptotext ist in Abbildung 5.6 dargestellt. Dabei bedeuten die umkreisten Knoten jeweils eine Transformation, während die durch ein Rechteck angegeben Knoten nur das aktuelle Zwischenergebnis enthalten.

Wir haben nun die einzelnen angewendeten Operationen zu erklären. Bei IP handelt es sich um die initiale Permutation der Bits. Sie ist durch

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

gegeben. Das Ergebnis sei $t_1t_2\dots t_{64}$. Bei der nachfolgenden Aufspaltung erhält der linke Strang die ersten 32 Bits, also das Wort $t_1t_2\dots t_{32}$, während an den linken Strang das Wort $t_{33}t_{34}\dots t_{64}$ der letzten 32 Bits übergeben wird. Vor Realisierung der IP inversen Transformation werden die beiden Wörter der Länge 32 wieder zu einem Wort der Länge 64 zusammengefügt (dabei ist die Vertauschung davor zu beachten).

Die Funktion f wird durch das Schema in Abbildung 5.7 berechnet. Als Eingabe werden eine 32-Bit-Folge R_{i-1} und eine 48-Bit-Folge K_i verarbeitet (i bezieht sich dabei auf den Iterationsschritt im Algorithmus). Zuerst wird die Operation E auf R_{i-1} angewendet, wodurch die 32-Bit-Folge R_{i-1} auf eine 48-Bit-Folge A erweitert wird.. Welches Element aus R_{i-1} an welcher Stelle in A steht, zeigt das folgende Schema:

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

Danach erfolgt die bitweise Addition von K_i. Das dadurch entstehende Resultat ist eine 48-Bit-Folge $f_1f_2\dots f_{48}$, die nun in acht Teilfolgen $U_i = f_{6(i-1)+1}f_{6(i-1)+2}\dots f_{6i}$ der Länge 6 unterteilt wird, $1 \le i \le 8$. Die Folge U_i wird an S_i übergeben und wie folgt verarbeitet. Zuerst werden $f_{6(i-1)+1}f_{6i}$ und $f_{6(i-1)+2}f_{6(i-1)+3}f_{6(i-1)+4}f_{6i-1}$ (man beachte die Gleichheit $6i-1 = 6(i-1)+5$) als binäre Darstellungen von Zahlen a und b mit $0 \le a \le 3$ bzw. $0 \le b \le 15$ interpretiert. Für das Paar (a,b) wird entsprechend der Tabelle aus

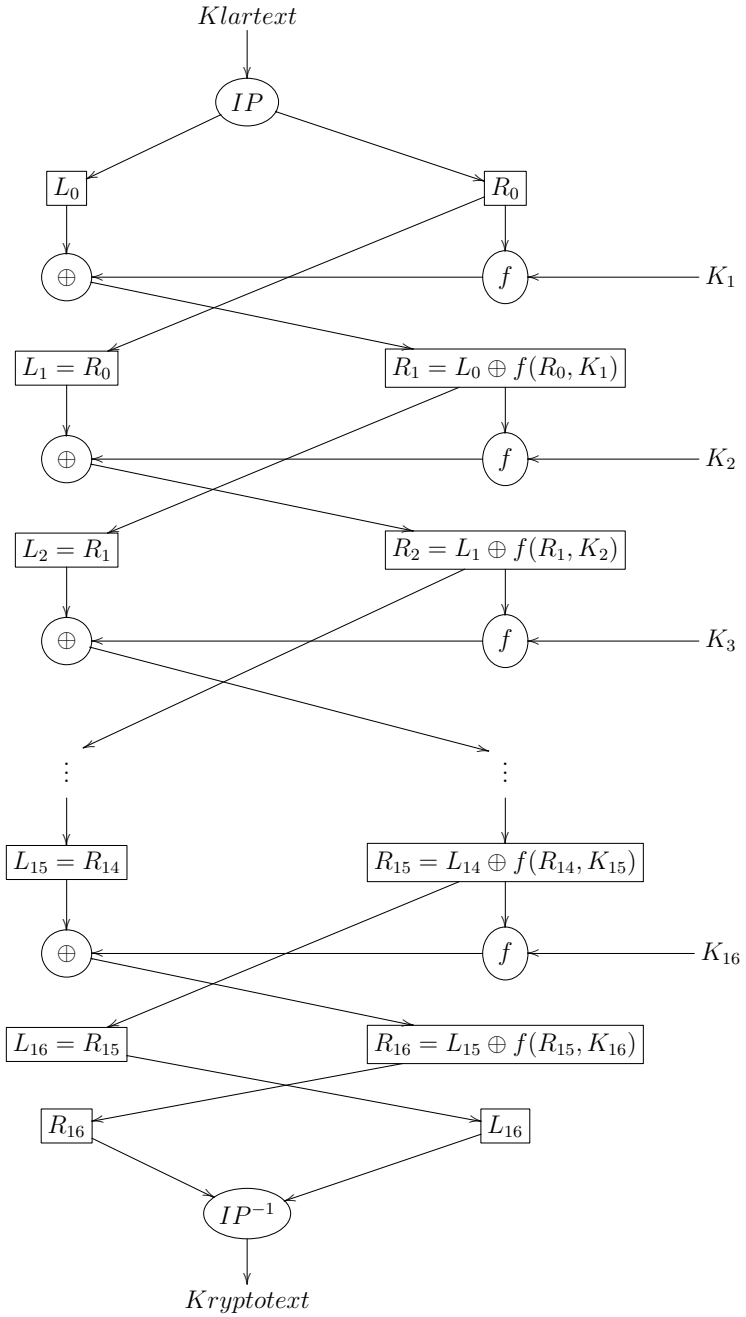


Abbildung 5.6: DES-Verschlüsselungsalgorithmus

Abbildung 5.8 eine Zahl c mit $0 \leq c \leq 15$ ermittelt, deren zugehörige Binärfolge U_i' der Länge 4 als Ausgabe von S_i dient. Die Verkettung $U_1'U_2'\ldots U_8'$ mit der Länge 32 wird durch P wie folgt permutiert:

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

Es bleibt zu klären, wie die 16 Eingaben K_i, $1 \leq i \leq 16$, in den 16 Iterationsschritten ermittelt werden. Das Verfahren ist in Abbildung 5.9 dargestellt. Als Eingabe dient der Schlüssel K. Er ist eine 56-Bit-Folge, die zuerst entsprechend dem Schema

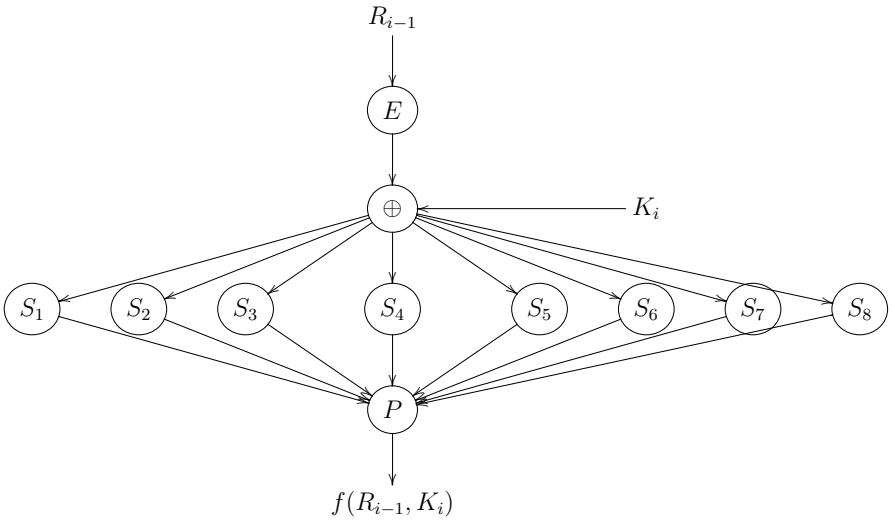


Abbildung 5.7: Berechnungsschema der Funktion f

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

permutiert wird. Danach wird sie in zwei Teilwörter der Länge 28 aufgeteilt. Es schließen sich 16 gleichartige Berechnungen für die K_i, $1 \leq i \leq 16$ an. Zuerst erfolgt eine zyklische Verschiebung innerhalb der beide Wörter der Länge 28 um jeweils $j(i)$ Elemente nach links. Die folgende Tabelle gibt die Größen $j(i)$ an.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$j(i)$	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Dann erfolgt eine Verkettung der so entstandenen Wörter. Aus dem entstandenen Wort der Länge 56 streicht PC_2 die Buchstaben an den Positionen 9, 18, 22, 25, 38, 43 und 54 und permutiert die verbleibenden Buchstaben. Es erfolgt damit eine Auswahl entsprechend dem folgenden Schema:

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

Dadurch entsteht das Wort K_i der Länge 48 im i-ten Schritt.

Wir kommen nun zur Dechiffrierung. Dazu wird das gleiche Schema verwendet, jedoch werden die Wörter K_i, $1 \leq i \leq 16$ in umgekehrter Reihenfolge angewendet. Zuerst wenden wir auf einen Buchstaben a_K des Kryptotextes, d. h. einem 64-Bit-Wort, die Permutation IP an. Da die Verschlüsselung mit der Anwendung von IP^{-1} abschloss, liefert dies die Teilwörter R_{16} und L_{16}, die nun entsprechend dem Verschlüsselungsalgorithmus (mit K_i in umgekehrter Reihenfolge) umgewandelt werden. Es ergeben sich dann mit Induktion von 16 nach 1 für i, $1 \leq i \leq 16$,

$$L_i = R_{i-1}$$

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

Abbildung 5.8: Die Transformationen S_i, $1 \leq i \leq 8$

und

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

Abschließend wird auf L_0R_0 noch IP^{-1} angewendet, wodurch der Buchstabe des Eingabetextes entsteht, der in a_K überführt wurde.

Damit gibt es auch für die Dechiffrierung eine gute Methode.

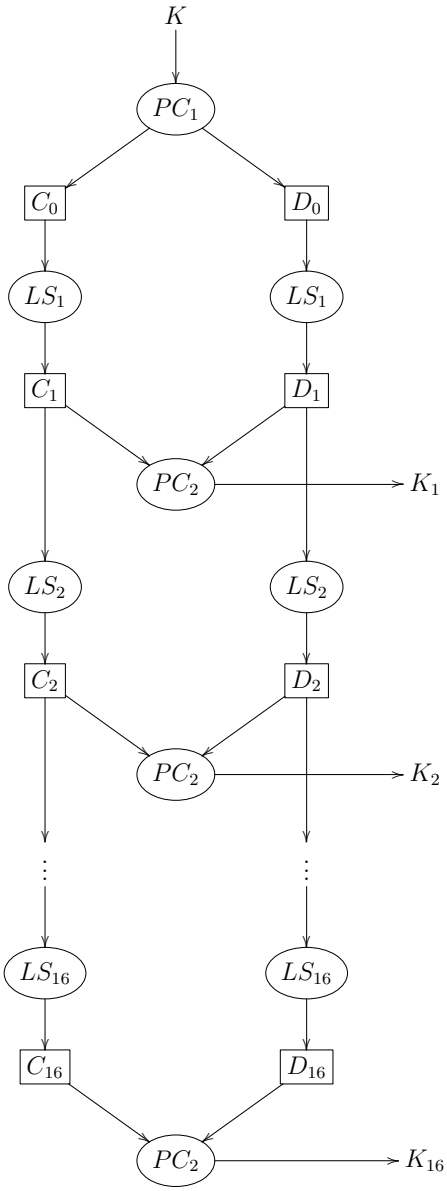


Abbildung 5.9: Berechnung der K_i, $1 \leq i \leq 16$

Der Kryptoanalyst hat kaum andere Möglichkeiten als alle denkbaren Schlüsselwörter K der Länge 56 zu testen, um bei der Entschlüsselung einen sinnvollen (dann hoffentlich richtigen) Klartext zu erhalten. Dies erfordert einen enorm hohen Aufwand.

5.4 Steganographie

Steganographie ist eine Methode, bei der der Klartext in einem umfangreichen Text versteckt wird und nur durch den Schlüssel lesbar gemacht werden kann. Da hierbei keine Verschlüsselung vorgenommen wird, kann dieses Verfahren nur bedingt der Kryptologie zugeordnet werden. Wir wollen hier nur ein solches Beispiel behandeln, das vom französischen Kardinal Richelieu (1585 – 1642) vielfach angewendet wurde. Wir betrachten

den folgenden Text:

D E R I N D E R K A M V O R D E M

L E T T E N E R S A H D E N U N G

A R N U N D B E Z A H L T E D A S

G E L D I M K L E I N E N R A U M

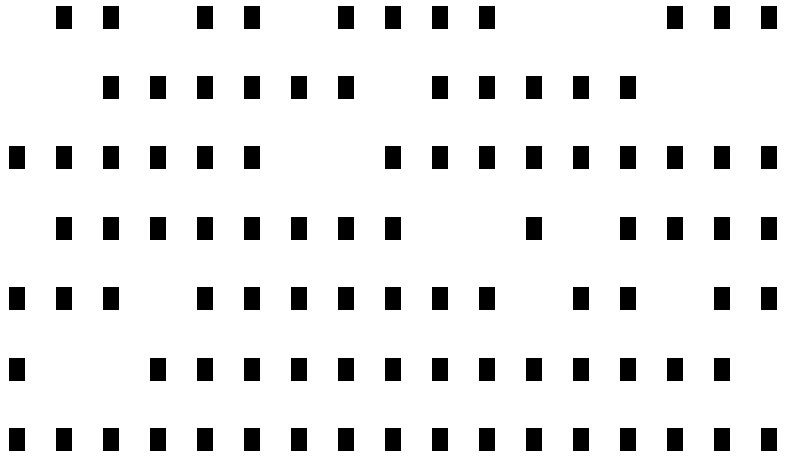
D E S T U R M E S A U F D E R B L

A U E N L I E G E S A S S D E R H

E S S E G E L A N G W E I L T

Die damit übermittelte Nachricht wird nur dann lesbar, wenn man im Besitz einer Schablone ist, die alle überflüssigen Buchstaben abdeckt. Die verbleibenden sichtbaren Buchstaben ergeben dann den Text.

In unserem Beispiel ist folgende Schablone zu benutzen. Dabei haben wir die abzudeckenden Buchstaben durch ■ markiert.



Es ergibt sich dann folgende Situation

D ■ ■ I ■ ■ E ■ ■ ■ ■ V O R ■ ■ ■

L E ■ ■ ■ ■ ■ ■ S ■ ■ ■ ■ ■ U N G

■ ■ ■ ■ ■ ■ B E ■ ■ ■ ■ ■ ■ ■ ■ ■

G ■ ■ ■ ■ ■ ■ ■ ■ I N ■ N ■ ■ ■ ■

■ ■ ■ T ■ ■ ■ ■ ■ ■ ■ F ■ ■ R ■ ■

■ U E ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ H

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

und damit der Klartext DIE VORLESUNG BEGINNT FRÜH.

Kapitel 6

Perfekte Sicherheit

Wir haben bei den monoalphabetischen Chiffren gesehen, dass der Kryptoanalyst wichtige Informationen, wie z. B. die Häufigkeit der Buchstaben im Kryptotext ausnutzen kann, um die Schlüsselinformation zu erhalten und den Kryptotext zu entschlüsseln. In diesem Abschnitt wollen wir solche Chiffriersysteme behandeln, bei denen der Kryptograph keine Erkenntnisse aus der Kenntnis des Kryptotextes gewinnen kann.

Wir führen dazu folgende Bezeichnungen ein. Mit $\mathcal{T}$ bezeichnen wir die Menge der (möglichen) Klartexte. $\mathcal{S}$ sei eine Menge von Verschlüsselungsfunktionen, bei denen jede Verschlüsselungsfunktion durch einen Schlüssel bestimmt sei. Mit $\mathcal{K}$ bezeichnen wir die Menge der Kryptotexte, die aus Klartexten durch Verschlüsselungen aus $\mathcal{S}$ entstehen können. Es gilt also

$$\mathcal{K} = \{\tau(t) \mid t \in \mathcal{T}, \tau \in \mathcal{S}\}\,.$$

Wir wollen davon ausgehen, dass es für jede Verschlüsselung $\tau \in \mathcal{S}$ und jeden Kryptotext $k \in \mathcal{K}$ höchstens einen Klartext $t \in \mathcal{T}$ mit $\tau(t) = k$ gibt (ansonsten könnte auch der befugte Empfänger den Klartext nicht eindeutig aus dem Kryptotext rekonstruieren). Damit gilt offensichtlich

$$\#(\mathcal{T}) \leq \#(\mathcal{K}). \tag{6.1}$$

Wir bezeichnen noch mit $p(t)$ die Wahrscheinlichkeit für das Auftreten des Klartextes $t \in \mathcal{T}$. Außerdem sei $p_k(t)$ die Wahrscheinlichkeit dafür, dass der Kryptotext $k \in \mathcal{K}$ durch Chiffrierung des Textes $t \in \mathcal{T}$ entstanden ist.

Definition 6.1 *Wir sagen, dass die Verschlüsselungen aus $\mathcal{S}$ (bzw. die Schlüssel zu $\mathcal{S}$) hinsichtlich $\mathcal{T}$ und $\mathcal{K}$ perfekte Sicherheit bieten, falls*

$$p_k(t) = p(t) \text{ für jeden Kryptotext } k \in \mathcal{K} \text{ und jeden Klartext } t \in \mathcal{T}$$

gilt.

Intuitiv bedeutet dies, dass die Kenntnis des Textes k dem Kryptoanalysten nichts hilft, da sich dadurch sein Kenntnisstand hinsichtlich der Frage, ob t das Urbild des Textes ist, nicht ändert.

Ein einfaches Beispiel für ein System mit perfekter Sicherheit bilden die Verschiebechiffren hinsichtlich der Mengen $\mathcal{T}$ und $\mathcal{K}$, die beide jeweils nur aus den Buchstaben aus *alph* bestehen. Offensichtlich gilt dabei $p(t) = \frac{1}{26}$ für jeden Text aus $t \in \mathcal{T}$ (Buchstaben t aus *alph*). Liegt dem Kryptoanalysten ein Text $k \in \mathcal{K}$ vor, so ist jeder Klarbuchstabe gleichwahrscheinlich als Urbild von k. Somit gilt auch $p_k(t) = \frac{1}{26}$.

Sei nun $\mathcal{S}$ ein Schlüsselsystem mit perfekter Sicherheit bez. gewisser Mengen $\mathcal{T}$ und $\mathcal{K}$ von Klar- und Kryptotexten. Dann gilt für jeden Kryptotext $k \in \mathcal{K}$ und für jeden Klartext $t \in \mathcal{T}$, $p_k(t) = p(t)$. Weiterhin ist $p(t) > 0$ für jeden Text $t \in T$ (wäre die Wahrscheinlichkeit $p(t) = 0$, so könnten wir diesen Text einfach aus $\mathcal{T}$ streichen). Damit ist $p_k(t) > 0$. Dies bedeutet, dass folgende Aussage gilt.

Fakt 1 *Bietet das Schlüsselsystem $\mathcal{S}$ perfekte Sicherheit bez. $\mathcal{T}$ und $\mathcal{K}$, so gibt es zu jedem Klartext $t \in \mathcal{T}$ und jedem Kryptotext $k \in \mathcal{K}$ eine Verschlüsselung $\tau \in \mathcal{S}$ so, dass $\tau(t) = k$ gilt.*

Wir betrachten nun zu einem Klartext t alle Verschlüsselungen, d. h. die Menge $U(t) = \{\tau(t) \mid \tau \in \mathcal{S}\}$. Nach Definition gilt $\#(U(t)) = \#(\mathcal{S})$. Außerdem gilt wegen Fakt 1 aber noch $\#(U(t)) \geq \#(\mathcal{K})$. Damit erhalten wir unter Beachtung von (6.1) den nächsten Fakt.

Fakt 2 *$\#(\mathcal{S}) \geq \#(\mathcal{K}) \geq \#(\mathcal{T})$ gilt für jedes Schlüsselsystem $\mathcal{S}$ mit perfekter Sicherheit bez. $\mathcal{T}$ und $\mathcal{K}$.*

Der nächste Satz kann als Umkehrung der beiden Fakten aufgefasst werden.

Satz 6.1 *Es sei $\mathcal{S}$ ein Schlüsselsystem mit $\#(\mathcal{T}) = \#(\mathcal{K}) = \#(\mathcal{S})$, in dem alle Schlüssel mit der gleichen Wahrscheinlichkeit vorkommen und in dem es zu jedem Klartext t und jedem Kryptotext k genau eine Transformation $\tau \in \mathcal{S}$ gibt, für die $\tau(t) = k$ gilt. Dann bietet $\mathcal{S}$ perfekte Sicherheit bez. $\mathcal{T}$ und $\mathcal{K}$.*

Beweis. Aufgrund des Bayesschen Satzes gilt

$$p_k(t) = \frac{p(t) \cdot p_t(k)}{p(k)}\,, \tag{6.2}$$

wobei $p(k)$ die Wahrscheinlichkeit dafür ist, dass ein Kryptotext mit k übereinstimmt, und $p_t(k)$ die Wahrscheinlichkeit dafür ist, dass t in k überführt wird. Aufgrund unserer Voraussetzung ist jeder Schlüssel gleichwahrscheinlich, womit jeder Kryptotext aus einem Klartext mit gleicher Wahrscheinlichkeit entsteht. Dies liefert $p_t(k) = \frac{1}{\#(\mathcal{S})}$. Außerdem ist jeder Kryptotext gleichwahrscheinlich, denn jeder Text entsteht mit gleicher Wahrscheinlichkeit aus einem Klartext unter Verwendung gleichwahrscheinlicher Schlüssel. Folglich ist $p(k) = \frac{1}{\#(\mathcal{K})}$. Beachten wir nun die Voraussetzung, dass $\#(\mathcal{S}) = \#(\mathcal{K})$ gilt, so erhalten wir $p_t(k) = p(k)$ und damit aus (6.2) die gewünschte Gleichheit $p_k(t) = p(t)$ für alle $k \in \mathcal{K}$ und alle $t \in \mathcal{T}$. □

Als ein Beispiel behandeln wir das *one-time Pad*. Dabei bestehen die Klartexte aus allen Bit-Folgen der Länge n und als Menge der Schlüssel verwenden wir die gleiche Menge. Die Verschlüsselung erfolgt durch bitweises Addieren modulo 2. Aus dem Klartext $t = t_1 t_2 \dots t_n \in \{0,1\}^n$ und dem Schlüssel $s_1 s_2 \dots s_n \in \{0,1\}^n$ entsteht der Kryptotext $(t_1 \oplus s_1)(t_2 \oplus s_2) \dots (t_n \oplus s_n)$. Damit liegt aber auch jeder Kryptotext in $\{0,1\}^n$. Weiterhin

ergibt sich, dass zu gegebenen $t \in \{0,1\}^n$ und gegebenem $k \in \{0,1\}^n$ ein Schlüssel $s = t \oplus k \in \{0,1\}^n$ mit $t \oplus s = k$ existiert. Hieraus ergibt sich als erstes, dass jede Bit-Folge aus $\{0,1\}^n$ als Kryptotext auftaucht, womit ebenfalls $\mathcal{K} = \{0,1\}^n$ gilt. Da die Mengen $\mathcal{T}$, $\mathcal{S}$ und $\mathcal{K}$ damit identisch sind, stimmen insbesondere ihre Kardinalitäten überein. Auch die anderen Voraussetzungen von Satz 6.1 sind erfüllt. Folglich handelt es sich bei dem System mit $\mathcal{T} = \mathcal{S} = \mathcal{K} = \{0,1\}^n$ um ein System mit perfekter Sicherheit.

Wir merken an, dass wegen $(t \oplus s) \oplus s = t$ die Dechiffrierung mit der Chiffrierung übereinstimmt.

Auf den ersten Blick ist dies natürlich kein brauchbares Kryptosystem, denn der geheimzuhaltende Schlüssel ist von gleicher Länge wie der Klartext. Wenn der Schlüssel nun problemlos vom Sender an den Empfänger übermittelt werden kann, so wäre dies ja auch für den Klartext zutreffend, und man könnte gleich diesen sicher senden. Jedoch gibt es einen kleinen Unterschied. Der Klartext ist in der Regel zu einem bestimmten Moment zu verschicken, während der Schlüsselaustausch früher zu einem passenden Moment oder auch über einem anderen Kanal verschickt werden kann. So könnten z.B. die Schlüssel durch einen Diplomaten überbracht werden, während der Klartext eine wichtige Information zu einem bestimmten Zeitpunkt innerhalb von Verhandlungen sein kann. Dabei kann dann der früher übergebene Schlüssel benutzt werden.

Für die perfekte Sicherheit ist es erforderlich, dass die Schlüssel gleichwahrscheinlich sind. Dies bedeutet in der Praxis, dass es sich beim Schlüssel um eine möglichst zufällige Folge handelt. Dies erfordert dann aber bei der Schlüsselübermittlung die Übergabe des gesamten Wortes der Länge n. Einfacher wäre es Folgen zu finden, die den Eindruck einer zufällig erzeugten Folge machen, aber durch einen (einfach zu merkenden und einfach zu realisierenden) Algorithmus gewonnen werden können.
Wir behandeln hier eine derartige Möglichkeit, bei der wir die Folge durch ein Schieberegister erzeugen.

Definition 6.2 *i) Unter einem Schieberegister der Länge m verstehen wir*

- *m Speicherelemente k_1, k_2, $\ldots k_m$, von denen jedes zu einem Zeitpunkt t, $t \geq 0$, genau ein Element $k_i(t) \in \{0,1\}$ enthält,*
- *m Konstanten $c_1, c_2, \ldots, c_m$ aus $\{0,1\}$ und*
- *m Werte $x_1, x_2, \ldots, x_m$ aus $\{0,1\}$.*

ii) Die Speicherelemente sind initial mit den Werten belegt, d. h. für $1 \leq i \leq m$ gilt $k_i(0) = x_i$.
Die Veränderung in einem Speicherelement erfolgt taktweise entsprechend den folgenden Formeln:

$$k_1(t+1) = c_1 k_1(t) \oplus c_2 k_2(t) \oplus \cdots \oplus c_m k_m(t)\,,$$
$$k_i(t+1) = k_{i-1}(t) \text{ für } 2 \leq i \leq m,\ t \geq 0\,.$$

iii) Die von einem Schieberegister ausgegebene Folge ist $k_t(0)k_t(1)k_t(2)\ldots$

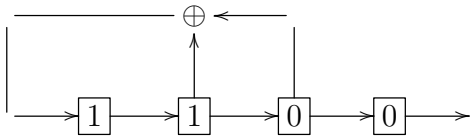


Abbildung 6.1: Veranschaulichung eines Schieberegisters

Die Speicherelemente werden auch Register genannt.

Als Beispiel betrachten wir das Schieberegister der Länge 4 mit

$$c_1 = 0,\ c_2 = 1,\ c_3 = 1,\ c_4 = 0 \quad \text{und} \quad x_1 = x_2 = 1,\ x_3 = x_4 = 0\,.$$

Das Schieberegister lässt sich wie in Abbildung 6.1 veranschaulichen. Dabei sind in der unteren Zeile die Register mit der Anfangsbelegung angegeben. Ferner gehen wir davon aus, dass die Addition (im Allgemeinen die Additionen) ohne Verzögerung arbeiten. Wir haben für die Berechnung des Wertes im Register k_1 nur die Addition von den Inhalten der Register k_2 und k_3 benutzt, da nach der Wahl der c_i nur diese Register einen Beitrag zur Summe $c_1k_1(t) \oplus c_2k_2(t) \oplus c_3k_3(t) \oplus c_4k_4(t)$ leisten. Es ergeben sich dann in den Takten die folgenden Werte:

Takt t	$k_1(t)$	$k_2(t)$	$k_3(t)$	$k_4(t)$	Ausgabe
0	1	1	0	0	0
1	1	1	1	0	0
2	0	1	1	1	1
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	0	0	0
6	1	0	1	0	0
7	1	1	0	1	1
8	1	1	1	0	0
9	0	1	1	1	1
10	0	0	1	1	1

Wir erkennen, dass die Register in den Takten 1 und 8 die gleichen Inhalte haben. Folglich stimmen auch die Inhalte in den Takten 2 und 9 bzw. 3 und 10 überein, da die Berechnung der Inhalte im nächsten Takt eindeutig ist. Somit haben wir ein periodisches Verhalten, nach jeweils 7 Takten erhalten wir das gleiche Resultat, wobei der Ausgangstakt ≥ 1 sein muss. Dieses Verhalten überträgt sich selbstverständlich auch auf die Ausgabefolge. Folglich erzeugt unser Schieberegister die Ausgabefolge

$$0(0111001)^*\,.$$

Dieses periodische Verhalten muss sich bei jedem Schieberegister der Länge m einstellen, denn die Tupel der Registerinhalte sind in $\{0,1\}^n$ enthalten, und diese Menge hat nur 2^m Elemente. Es ist beim Entwurf des Kryptosystems, das eine Bit-Folge verwendet, die von einem Schieberegister erzeugt wird, die Länge m möglichst groß und die Koeffizienten c_i und die Anfangsbelegung x_i, $1 \leq i \leq m$ so zu wählen, dass eine möglichst große Periode entsteht. Wir geben hier ohne Beweis an, dass man bei gegebenem m die Parameter stets

so wählen kann, dass sich eine Periode der Länge $2^m - 1$ ergibt. Dadurch wirkt die erzeugte Folge relativ zufällig.

Kommen wir nun zur Situation des Kryptoanalysten. Kennt er ein Paar *(Klartext, Kryptotext)* der Länge $2m$, wobei m die Länge des Schieberegisters ist, so kann er die Gleichungen, die das Schieberegister beschreiben, in einfacher Weise umformen. Sei $(t_1 t_2 \ldots t_{2m}, v_1 v_2 \ldots v_{2m})$ das gegebene Paar. Dann wird daraus zuerst die Ausgabefolge $y_1 y_2 \ldots y_{2m}$ des Schieberegisters ermittelt. Offensichtlich muss $y_i = t_i \oplus v_i$ für $1 \leq i \leq 2m$ gelten. Als Erstes stellt der Kryptoanalyst fest, dass die Ausgabe in den ersten m Takten, ihm gerade die Anfangsbelegung liefert, d. h. $y_i = d_{m-i}$, da in den ersten m Takten der Reihe nach die Werte $k_m(0), k_{m-1}(0), \ldots, k_1(0)$ ausgegeben werden. Ferner gilt noch $y_j = k_{m-s}(j+s+1))$. Damit erhalten wir aus der Gleichung

$$k_1(t+1) = c_1 k_1(t) \oplus c_2 k_2(t) \oplus \cdots \oplus c_m k_m(t)\,,$$

die das Verhalten des ersten Registers beschreibt, die Gleichung

$$y(m+t+1) = c_1 y(m+t) \oplus c_2 y(m+t-1) \oplus \cdots \oplus c_m y(t+1)$$

für $0 \leq t \leq m-1$. Der Kryptoanalyst erhält somit ein lineares Gleichungssystem zur Bestimmung der c_i, $1 \leq i \leq m$. Die Lösungen bilden einen Vektorraum der Dimension $m - r$, wobei r der Rang der Koeffizientenmatrix ist. Ist $r = m$, so hat das Gleichungssystem eine eindeutige Lösung, aus der er die volle Kenntnis des Schieberegisters gewinnt und damit in der Lage ist, jeden Kryptotext zu entschlüsseln. Ist das Gleichungssystem nicht eindeutig lösbar, so gibt es 2^{n-r} Lösungen, d. h. der Kryptoanalyst hat in der Regel durch sein Vorgehen die Zahl der möglichen Schieberegister stark eingeschränkt.

Kapitel 7

Öffentliche Schlüsselsysteme

7.1 Die Idee öffentlicher Schlüssel

Bei allen bisher behandelten Verschlüsselungen war es äußerst wichtig, den Schlüssel geheim zu halten, da dessen Kenntnis nicht nur das Verschlüsseln sondern stets auch das Entschlüsseln gestattete. In diesem Abschnitt behandeln wird kryptographische Systeme, bei denen die Verschlüsselungsmethode öffentlich bekanntgegeben wird, womit jede Person einen verschlüsselten Text schicken. Trotzdem soll das Entschlüsseln für unbefugte Personen nicht möglich oder zumindest sehr schwer sein. Auf den ersten Blick scheint es, dass derartige Systeme nicht existieren können, aber es zeigte sich, dass sie konstruierbar sind.

Als einführendes Beispiel betrachten wir folgende Verschlüsselung. Um den Buchstaben α zu verschlüsseln, wählen wir aus dem Telefonbuch einer größeren Stadt, z.B. Magdeburg, nichtdeterministisch eine Person, deren Anfangsbuchstaben des Nachnamens gerade α ist (es gibt tatsächlich in Magdeburg zu jedem Buchstaben eine solche Person) und übermitteln deren Telefonnummer anstelle des Buchstaben. Hat man nur ein übliches Telefonbuch, so ist die Entschlüsselung in der Tat schwer, da man zu der Telefonnummer die Person – genauer den Anfangsbuchstaben ihres Nachnamens – finden muss. Wer aber im Besitz der umgekehrten Zuordnung ist, wo die Telefonnummern in geordneter Reihenfolge mit den zugehörigen Personen stehen (derartige Telefonbücher sind mindestens bei den Telefongesellschaften vorhanden), kann sehr einfach entschlüsseln. Für befugte Personen gibt es daher ein einfaches Entschlüsselungsverfahren, während unbefugte Personen unter Verwendung von üblichen Telefonbüchern den Kryptotext kaum entschlüsseln können. Jedoch gibt es einen Ausweg für den Kryptoanalysten: Er ruft einfach die Nummer an und fragt nach dem Namen des Anschlussbesitzers, womit jeweils der richtige Buchstabe gefunden werden kann. Es muss also gesichert werden, dass es keine einfache Methode zur Entschlüsselung gibt.

Dies wiederum bedeutet aber, dass auch die befugten Personen die Entschlüsselung nicht vornehmen können. Es ist daher notwendig, den befugten Empfängern Zusatzinformationen zu geben, die ihnen eine Entschlüsselung gestattet. Diese Zusatzinformation ist selbstverständlich geheim zu halten, und sie darf nicht aus der Verschlüsselungsmethode ablesbar sein.

Desweiteren hat man zu bedenken, dass bei öffentlich bekannter Methode der Ver-

schlüsselung, der Kryptoanalyst beliebig viele Paare erzeugen kann, die aus einem Klartext und dem zugehörigen Kryptotext bestehen, denn er kann die Verschlüsselung ja selbst vornehmen. Wir wissen aus den Beispielen der vorhergehenden Kapitel, dass bei Kenntnis beliebig vieler Paare *(Klartext,Kryptotext)* der Kryptotext oft leicht eine Entschlüsselungsmethode finden kann.

Wir wollen zuerst den bisher nur umgangssprachlich benutzten Begriff der „Schwierigkeit“ etwas präzisieren. Wir werden „Schwierigkeit“ im Sinne der Komplexitätstheorie verwenden. Unter Berücksichtigung praktischer Belange bedeutet dies, dass „einfach“ mit „polynomialer Berechenbarkeit“ (wobei der Grad der Polynome möglichst höchstens drei ist) und „schwierig“ mit „nichtpolynomialer Berechenbarkeit“ gleichgesetzt wird.

Wir wollen nun einige Ausführung zur Komplexität der Aufgabe des Kryptoanalysten machen. Dabei wollen wir annehmen, dass die öffentlich bekannte Verschlüsselung „einfach“ ist, also in einer Zeit, die polynomial in der Länge des Klartextes ist, ausgeführt werden kann. Gleiches soll auch für die Entschlüsselung gelten, d. h. der Klartext kann in einer Zeit, die polynomial in der Länge des Kryptotextes ist, ermittelt werden.

Satz 7.1 *Die Aufgabe des Kryptoanalysten, zu einem gegeben Kryptotext k den Klartext t zu finden, liegt in der Komplexitätsklasse $\mathbb{NP}$ (der nichtdeterministisch in polynomialer Zeit lösbaren Probleme).*

Beweis. Da die Entschlüsselung entsprechend unserer Annahme höchstens polynomiale Zeit erfordert, hat t höchstens eine in $|k|$ polynomiale Länge l. Der Kryptoanalyst erzeugt nun nichtdeterministisch alle möglichen Wörter w der Länge $\leq l$, berechnet für jedes Wort w den zugehörigen Kryptotext $k(w)$ entsprechend der öffentlich bekannten Verschlüsselungsmethode und vergleicht $k(w)$ mit k. Fällt einer dieser Tests positiv ausfällt, so liefert das zugehörige Wort w den Klartext, da es genau einen Klartext gibt zu k gibt (sonst hätte der legale Empfänger auch zwei mögliche Klartexte). Da jeder Schritt in diesem Verfahren in polynomialer Zeit erledigt werden kann, ist die Aufgabe des Kryptoanalysten in $\mathbb{NP}$ enthalten. □

Um die Aufgabe des Kryptoanalysten wirklich schwer zu machen, ist es daher wünschenswert, dass sie ein $\mathbb{NP}$-vollständiges Problem ist (da allgemein angenommen wird, dass derartige Probleme nicht in polynomialer Zeit gelöst werden können).

Satz 7.2 *Wenn die Aufgabe des Kryptoanalysten $\mathbb{NP}$-vollständig ist, dann gilt $\mathbb{NP} = co\text{-}\mathbb{NP}$.*

Beweis. Analog zum Beweis von Satz 7.1 kann gezeigt werden, dass die Aufgabe C des Kryptoanalysten in co-$\mathbb{NP}$ liegt.

Da C nach Voraussetzung $\mathbb{NP}$-vollständig ist, kann jede Sprache $L \in \mathbb{NP}$ in polynomialer Zeit auf C reduziert werden. Daher kann das Komplement von L in polynomialer Zeit auf das Komplement von C reduziert werden. Da das Komplement von C in $\mathbb{NP}$ liegt, liegt damit auch das Komplement von L in $\mathbb{NP}$. Damit ist L in co-$\mathbb{NP}$ und folglich $\mathbb{NP} \subseteq co\text{-}\mathbb{NP}$ gezeigt.

Sei nun L in co-$\mathbb{NP}$. Dann ist das Komplement von L in $\mathbb{NP}$ und nach dem Vorstehenden damit in co-$\mathbb{NP}$. Somit ist L in $\mathbb{NP}$. Daher gilt auch $co\text{-}\mathbb{NP} \subseteq \mathbb{NP}$. □

Da die Gleichheit *co*-$\mathbb{NP} = \mathbb{NP}$ äquivalent zu der Gleichheit $\mathbb{P} = \mathbb{NP}$ ist, ist davon auszugehen, dass *co*-$\mathbb{NP} = \mathbb{NP}$ nicht gilt. Daher ist nicht anzunehmen, dass die Aufgabe des Kryptoanalysten $\mathbb{NP}$-vollständig ist.

Die Idee für den Entwurf eines kryptographischen Systems mit öffentlichen Schlüsseln folgt im Wesentlichen dem folgenden Plan.

1. Man wähle ein schwieriges Problem P (z.B. könnte P ein unentscheidbares oder ein $\mathbb{NP}$-vollständiges Problem sein; es sollte aber nicht nur die Komplexität des ungünstigsten Falls sondern auch die durchschnittliche Komplexität hoch sein).

2. Man wähle ein einfaches Teilproblem P_l von P (d. h. P_l sollte in polynomialer Zeit möglichst in linearer Zeit lösbar sein; der Grad des Polynoms sollte zumindest nicht größer als sein).

3. Man transformiere das Problem P_l derartig in ein Problem Q,

   - dass nicht zu sehen ist, dass P_l Ausgangspunkt war, und
   - dass Q als genauso schwer aussieht wie P.

4. Man gebe öffentlich bekannt, wie man mittels des Problems Q eine Nachricht verschlüsselt. Die Entschlüsselung wird dann für den Kryptoanalysten in der Lösung von Q (und damit scheinbar von P) bestehen.

   Dagegen wird die Methode, wie man aus Q das einfache Problem P_l gewinnt, geheim gehalten. Die Entschlüsselung erfolgt dann mittels P_l.

In den folgenden Abschnitten werden wir dieses Vorgehen anhand einiger Beispiele diskutieren, wobei wir für P $\mathbb{NP}$-vollständige, unentscheidbare Probleme bzw. ein Problem wählen, das nach bisheriger Kenntnis sehr schwierig zu sein scheint.

7.2 Ein System auf der Basis des Rucksack-Problems

Wir stellen in diesem Abschnitt ein System mit öffentlichem Schlüssel, d. h. einer öffentlich bekannten Verschlüsselung vor, das in [11] eingeführt wurde und dessen Grundlage das Rucksack-Problem bildet. Dabei wollen wir schrittweise dem in Abschnitt 7.1 angegebenen Verfahren folgen.

Schritt 1. Ausgangspunkt ist für uns das Rucksack-Problem

Gegeben:	$n+1$ natürliche Zahlen $a_1, a_2, \ldots, a_n, a_{n+1}$
Gesucht:	Zahlen $\alpha_1, \alpha_2, \ldots, \alpha_n \in \{0,1\}$ derart, dass $\alpha_1 a_1 + \alpha_2 a_2 + \cdots + \alpha_n a_n = a_{n+1}$ gilt (oder man treffe die Aussage, dass derartige Zahlen nicht existieren).

Dieses Problem lässt auch wie folgt beschreiben. Es sei $A = (a_1, a_2, \ldots, a_n)$ ein Vektor. Gesucht ist dann eine Lösung $(\alpha_1, \alpha_2, \ldots, \alpha_n) \in \{0,1\}$ des Gleichungssystems

$$A(x_1, x_2, \ldots, x_n)^T = a_{n+1}.$$

Wir beschreiben daher das Rucksack-Problem auch durch das Paar (A, a_{n+1})

Außerdem kann das Rucksack-Problem als die Frage interpretiert werden, ob es eine Auswahl aus den Elementen $a_1, a_2, \ldots, a_n$ derart gibt, dass die Summe der ausgewählten Elemente gerade a_{n+1} gibt.

Es ist bekannt (siehe z.B. [5]), dass das Rucksack-Problem $\mathbb{NP}$-vollständig ist. Da die allgemeine Erwartung ist, dass die Lösung $\mathbb{NP}$-vollständiger Probleme nicht in polynomialer Zeit möglich ist, betrachten wir das Rucksack-Problem als ein schwieriges Problem.

Wir beschreiben nun die Verschlüsselung von Objekten mittels des Rucksack-Problems. Dabei gehen wir davon aus, dass die Objekte durch Binärzahlen der Länge n gegeben sind. Ist $\beta_1\beta_2\ldots\beta_n$ die Darstellung des Objektes o, so verschlüsseln wir o einfach als Zahl b, die durch

$$b = \beta_1 a_1 + \beta_2 a_2 + \cdots + \beta_n a_n$$

definiert ist.

Beispiel 7.1 Als Beispiel verwenden wir die Darstellung eines Buchstaben a durch die Binärdarstellung der Zahl $\varphi(a) + 1$. Da wir nur 26 Buchstaben haben, reicht es offensichtlich Binärzahlen der Länge 5 zu betrachten. Um stets die gleiche Länge zu erreichen, benutzen wir führende Nullen.

Wir erhalten dann zum Beispiel für die Buchstaben A, L, P, U die Darstellung aus der folgenden Tabelle:

Buchstabe	$\varphi(a)+1$	Binärdarstellung
A	1	00001
L	12	01100
P	16	10000
U	21	10101

Für die Verschlüsselung müssen wir ein Rucksack-Problem mit $n = 5$ benutzen. Wir setzen

$$a_1 = 1,\ a_2 = 3,\ a_3 = 5,\ a_4 = 10,\ a_5 = 20\,.$$

Dann ergeben sich für unsere Buchstaben die Werte

Buchstabe	Verschlüsselungswert
A	$20 = 0\cdot 1 + 0\cdot 3 + 0\cdot 5 + 0\cdot 10 + 1\cdot 20$
L	$8 = 0\cdot 1 + 1\cdot 3 + 1\cdot 5 + 0\cdot 10 + 0\cdot 20$
P	$1 = 1\cdot 1 + 0\cdot 3 + 0\cdot 5 + 0\cdot 10 + 0\cdot 20$
U	$26 = 1\cdot 1 + 0\cdot 3 + 1\cdot 5 + 0\cdot 10 + 1\cdot 20$

Somit könnten wir das Wort PAUL durch die Folge (1, 20, 26, 8) verschlüsseln.

Bei diesem Vorgehen hat der Kryptoanalyst zur Bestimmung des durch b verschlüsselten Buchstabens a das Rucksack-Problem mit den Parametern

$$a_1 = 1,\ a_2 = 3,\ a_3 = 5,\ a_4 = 10,\ a_5 = 20 \text{ und } a_6 = b$$

zu lösen.

In der Praxis wird man natürlich nicht das Rucksack-Problem mit $n = 5$ wählen, denn durch Testen aller 2^n Möglichkeiten für die Koeffizienten α_i zu a_i, $1 \leq i \leq 5$, ist die Dualdarstellung des Buchstaben leicht zu gewinnen. Verschlüsseln wir ein Wort aus k Buchstaben $a_1 a_2 \dots a_k$ durch das Binärwort, das durch Konkatenation (Hintereinanderschreiben) der Binärwörter zu a_i entsteht, so wird ein Rucksack-Probleme mit $n = 5k$ erforderlich. Für hinreichend großes k ist dann ein Durchtesten nicht mehr möglich. $\Diamond$

Schritt 2. Wir führen in diesem Schritt super-wachsende Folgen ein, die eine einfache Variante des Rucksack-Problems ermöglichen.

Definition 7.1

i) *Ein Vektor $(a_1, a_2, \dots, a_n)$ heißt monoton wachsend, falls $a_i < a_{i+1}$ für $1 \leq i < n$ gilt.*

ii) *Ein Vektor $(a_1, a_2, \dots, a_n)$ heißt super-wachsend, falls*

$$\sum_{j=1}^{i-1} a_j < a_i$$

für $2 \leq i \leq n$ gilt.

Super-Wachstum impliziert offensichtlich monotones Wachstum.

Falls ein Rucksack-Problem einen super-wachsenden Vektor $A = (a_1, a_2, \dots, a_n)$ hat, so nennen wir auch das Rucksack-Problem *super-wachsend*.

Wir bemerken, dass der zu unserem Rucksack-Problem aus Beispiel 7.1 gehörende Vektor $A = (1, 3, 5, 10, 20)$ super-wachsend ist.

Satz 7.3 *Ist das Rucksack-Problem durch einen super-wachsenden Vektor A und a_{n+1} gegeben, so ist eine Lösung des Problems in linearer Zeit bestimmbar.*

Beweis. Es sei $A = (a_1, a_2, \dots, a_n)$.

Für $n = 1$ ist die Aussage klar. Falls $a_{n+1} = a_2 = a_1$ gilt, so ist die Lösung $\alpha_1 = 1$. Im anderen Fall $a_2 \neq a_1$ gibt es keine Lösung.

Es sei nun schon gezeigt, dass ein super-wachsendes Rucksack-Problem mit dem Parameter n in linearer Zeit lösbar ist, und sei das super-wachsende Rucksack-Problem mit $A = (a_1, a_2, \dots, a_{n+1})$ und a_{n+2} gegeben.

Falls $a_{n+2} < a_{n+1}$ gilt, so muss offensichtlich $\alpha_{n+1} = 0$ gelten. Damit hat das gegebene Rucksack-Problem genau dann eine Lösung, wenn es eine Lösung für das ebenfalls super-wachsende Rucksack-Problem mit $A' = (a_1, a_2, \dots, a_n)$ und $a'_{n+1} = a_{n+2}$ gibt. Letzteres ist in linearer Zeit lösbar. Damit ist auch das Ausgangsproblem in linearer Zeit lösbar.

Es sei nun $a_{n+2} \geq a_{n+1}$. Dann muss $\alpha_{n+1} = 1$ gelten, da die Summe der verbleibende Werte a_i, $1 \leq i \leq n$ wegen des strengen Wachstums nicht a_{n+2} erreicht. Wir betrachten nun das super-wachsende Rucksack-Problem mit

$$A' = (a_1, a_2, \dots, a_n) \text{ und } a'_{n+1} = a_{n+2} - a_{n+1}.$$

Hat dieses Problem die in linearer Zeit ermittelbare Lösung $(\alpha_1, \alpha_2, \dots, \alpha_n)$, so hat das Ausgangsproblem die Lösung $(\alpha_1, \alpha_2, \dots, \alpha_n, 1)$, die dann auch in linearer Zeit ermittelt wurde. Hat das neu konstruierte Problem aber keine Lösung, so hat auch das Ausgangsproblem keine Lösung. $\square$

Beispiel 7.2 Wir illustrieren die im Beweis gegebene Methode zur Lösung super-wachsender Rucksack-Probleme anhand unseres Beispiels 7.1.

Sei der Wert $a_6 = 24$ gegeben. Da $20 < 24$ ist, muss $\alpha_5 = 1$ gelten.

Wir gehen zum Rucksack-Problem mit den Parametern $1, 3, 5, 10$ und 4 über. Wegen $10 > 4$ ergibt sich auch $\alpha_4 = 0$.

Nun haben wir für das neue Rucksack-Problem die Parameter $1, 3, 5$ und 4. Wegen $4 < 5$, ist $\alpha_3 = 0$.

Das neue Rucksack-Problem hat daher die Parameter $1, 3$ und 4. Offenbar muss $\alpha_2 = 1$ gelten.

Das verbleibende ist dann durch 1 und 1 bestimmt, woraus $\alpha_1 = 1$ folgt.

Wir erhalten aus der Lösung folglich die Binärdarstellung 10011 der Zahl 19, die dem Buchstaben S entspricht. Wir würden also den Kryptotext 24 (entsprechend der Verschlüsselung aus Beispiel 7.1) durch S entschlüsseln. ◊

Schritt 3. Wir transformieren nun ein super-wachsendes Rucksack-Problem derart, dass das vorhandene Super-Wachstum der Komponenten verloren geht, so dass für den Kryptoanalysten scheinbar ein gewöhnliches Rucksack-Problem vorliegt.

Definition 7.2 *Es seien der Vektor $A = (a_1, a_2, \ldots, a_n)$ gegeben. Ferner seien (t, m) ein Paar natürlicher Zahlen mit*

$$ggT(t, m) = 1 \quad \textit{und} \quad m > \max\{a_1, a_2, \ldots, a_n\}\,.$$

i) *Wir sagen, dass der Vektor*

$$B = (ta_1 \bmod m,\ ta_2 \bmod m,\ \ldots, ta_n \bmod m)$$

aus A durch modulare Multiplikation mit (t, m) entsteht. t und m heißen Faktor bzw. Modulus der modularen Multiplikation.

ii) *Falls sogar $m > \sum_{i=1}^{n} a_i$ gilt, so sprechen wir von strenger modularer Multiplikation.*

Beispiel 7.3 Wir betrachten den Vektor $A = (1, 3, 5, 10, 20)$ des Rucksack-Problems aus Beispiel 7.1. Wir wählen $t = 33$ und $m = 100$. Offenbar gilt dann

$$ggT(33, 100) = 1 \quad \text{und} \quad 1 + 3 + 5 + 10 + 20 = 29 < 100\,.$$

Folglich entsteht aus A durch (strenge) modulare Multiplikation der Vektor

$$\begin{aligned} B &= (33 \cdot 1 \bmod 100, 33 \cdot 3 \bmod 100, 33 \cdot 5 \bmod 100, 33 \cdot 10 \bmod 100, 33 \cdot 20 \bmod 100) \\ &= (33 \bmod 100, 99 \bmod 100, 165 \bmod 100, 330 \bmod 100, 660 \bmod 100) \\ &= (33, 99, 65, 30, 60) \end{aligned}$$

Wir sehen sofort, dass B kein super-wachsender Vektor, ja nicht einmal ein wachsender Vektor ist.

Wenn wir nun mittels des neuen Vektors B die Verschlüsselung von PAUL vornehmen wollen, so ergeben sich bei Rechnung $\bmod m$ die Werte

Buchstabe	Verschlüsselungswert
A	$0 \cdot 33 + 0 \cdot 99 + 0 \cdot 65 + 0 \cdot 30 + 1 \cdot 60 = \ \ 60 = 60 \bmod 100$
L	$0 \cdot 33 + 1 \cdot 99 + 1 \cdot 65 + 0 \cdot 30 + 0 \cdot 60 = 164 = 64 \bmod 100$
P	$1 \cdot 33 + 0 \cdot 99 + 0 \cdot 65 + 0 \cdot 30 + 0 \cdot 60 = \ \ 33 = 33 \bmod 100$
U	$1 \cdot 33 + 0 \cdot 99 + 1 \cdot 65 + 0 \cdot 30 + 1 \cdot 60 = 158 = 58 \bmod 100$

und damit zum Klartext PAUL der Kryptotext $(60, 64, 33, 58)$. ◊

Bevor wir die Entschlüsselung diskutieren, bemerken wir, dass wegen der Voraussetzung $ggt(t, m) = 1$ eine natürliche Zahl t^{-1} $(1 \leq t^{-1} < m)$ existiert, für die $t \cdot t^{-1} = t^{-1} \cdot t = 1 \bmod m$ gilt. t^{-1} heißt (modulare) Inverse zu t.

Der folgende Satz gibt Auskunft über die Lösbarkeit eines Rucksack-Problems (B, b), bei dem B durch modulare Multiplikation aus einem super-wachsenden A entstanden ist, d. h. über die Lösbarkeit in der Situation, die gerade bei der von uns vorgenommenen Transformation zur Verschleierung des Super-Wachstums auftritt.

Satz 7.4 *Es seien A ein super-wachsender Vektor und b eine natürliche Zahl. Der Vektor B entstehe aus A durch strenge modulare Multiplikation mit (t, m).*

i) *Das Rucksack-Problem (B, b) hat höchstens eine Lösung.*
ii) *Wenn das Rucksack-Problem (B, b) eine Lösung hat, dann stimmt sie mit der Lösung von $(A, t^{-1}b \bmod m)$ überein.*

Beweis. Es sei $A = (a_1, a_2, \ldots, a_n)$ und $B = (b_1, b_2, \ldots, b_n)$. Wegen $b_i = ta_i \bmod m$ gilt auch $t^{-1}b_i = t^{-1}ta_i = 1 \cdot a_i = a_i \bmod m$ für $1 \leq i \leq n$.

Wenn $D = (\beta_1, \beta_2, \ldots, \beta_n)$ eine Lösung von (B, b) ist, so gilt $B \cdot D^T = b$. Damit gilt auch

$$t^{-1}b = t^{-1} \cdot (B \cdot D^T) = (t^{-1} \cdot B) \cdot D^T = A \cdot D^T \bmod m\,.$$

Da B durch strenge modulare Multiplikation aus A entsteht, ist $m > \sum_{i=1}^{n} a_i$. Daher gilt sogar $t^{-1}b \bmod m = A \cdot D^T$, womit D Lösung des Rucksack-Problems $(A, t^{-1}b \bmod m)$ ist. Da A ein super-wachsender Vektor ist, hat $(A, t^{-1}b \bmod m)$ genau eine Lösung. Folglich hat auch (B, b) genau eine Lösung, wenn es überhaupt eine Lösung gibt. Damit sind alle Aussagen des Satzes gezeigt. □

Aufgrund dieses Satzes kann ein befugter Empfänger mit der Kenntnis des Paares (t, m) zuerst das Inverse t^{-1} ausrechnen, und dann aus (B, b) das System $(A, t^{-1}b \bmod m)$ (man beachte $a_i = t^{-1}b_i \bmod m$) gewinnen, dessen Lösung sich nach Satz 7.3 in linearer Zeit gewinnen lässt.

Für den Kryptoanalysten stellt sich die Aufgabe anders. Er verfügt nicht über die Kenntnis von (t, m) und muss daher das scheinbar $\mathbb{NP}$-vollständige Rucksack-Problem (B, b) lösen.

Wir wollen nun zeigen, dass der Kryptoanalyst auch ganz anders vorgehen kann, um eine Lösung zu finden. Eine Idee wäre es, das Paar (t, m) der (strengen) modularen Multiplikation, mit der das originale streng-wachsende Rucksack-Problem transformiert wurde, zu finden. Dann hätte er nur noch die Entschlüsselung wie ein legaler Empfänger vorzunehmen. Dieser Ansatz lässt sich aber noch vereinfachen. Der Kryptoanalyst muss nicht

das zur Verschleierung benutzte Paar (t, m) bestimmen. Es reicht ihm ein Paar (t', m') mit $ggT(t', m') = 1$ und $m' > \sum_{i=1}^{n} a_i$ und einen streng-wachsenden Vektor A' so zu bestimmen, dass B aus A' mittels strenger modularer Multiplikation entsteht. Aufgrund von Satz 7.3 haben dann die Rucksack-Probleme (B, b) und $(A', (t')^{-1}b)$ die gleiche Lösung (man beachte, dass (B, b) eine Lösung hat, da es durch Verschlüsselung entstanden ist). Die Lösung von $(A', (t')^{-1}b)$ kann der Kryptoanalyst in linearer Zeit bestimmen, da A' super-wachsend ist. Deshalb geben wir die folgende Definition.

Definition 7.3 *Ein Vektor B heißt super-erreichbar, wenn es einen super-wachsenden Vektor A gibt, aus dem B mittels strenger modularer Multiplikation gewonnen werden kann.*

Definition 7.4 *Für einen Vektor $A = (a_1, a_2, \ldots a_n)$, natürliche Zahlen m und t mit $m \geq \max\{a_1, a_2, \ldots a_n\}$ und $ggT(t, m) = 1$ und eine natürliche Zahl $k \geq 0$ definieren den Vektor*

$$A(k) = (a_1 + k\lfloor ta_1/m \rfloor, a_2 + k\lfloor ta_2/m \rfloor, \ldots, a_n + k\lfloor ta_n/m \rfloor)\,.$$

Die Folge der Vektoren $A(k)$, $k \geq 0$, heißt wachsende Folge zu A bez. (t, m).

Lemma 7.5 *Es seien A, m und t wie in Definition 7.4 gegeben. Ist A ein (super-)wachsender Vektor, so ist auch jeder Vektor $A(k)$, $k \geq 0$, der wachsenden Folge bezüglich (m, t) (super-)wachsend.*

Beweis. Wir geben den Beweis nur für das Superwachstum.
Aus der Relation

$$\sum_{j=1}^{i-1} a_j < a_i$$

folgt sofort

$$\sum_{j=1}^{i-1} \lfloor ta_j/m \rfloor \leq \lfloor \frac{t \cdot \sum_{j=1}^{i-1} a_j}{m} \rfloor \leq \lfloor ta_i/m \rfloor$$

□

Lemma 7.6 *Es seien A, m und t wie in Definition 7.4 gegeben. Falls der Vektor B durch eine modulare Multiplikation mit (t, m) aus dem Vektor A gewonnen werden kann. so entsteht B für jede natürliche Zahl $k \geq 0$ durch modulare Multiplikation mit $(t, mk + t)$ aus dem Vektor $A(k)$ der wachsenden Folge bez. (t, m). Diese Aussage gilt auch, wenn modulare Multiplikation durch strenge modulare Multiplikation ersetzt wird.*

Beweis. Als Erstes stellen wir fest, dass auch $ggT(t, m + kt) = 1$ gilt, denn jeder Teiler von t und $m + kt$ ist auch ein Teiler von m. Damit ist $(t, m + kt)$ für eine modulare Multiplikation geeignet.

Nach Voraussetzung gilt $b_i = ta_i \bmod m$. Daher ist $ta_i = \lfloor ta_i/m \rfloor \cdot m + b_i$. Damit erhalten wir

$$\begin{aligned} t(a_i + k \cdot \lfloor ta_i/m \rfloor) &= b_i + \lfloor ta_i/m \rfloor \cdot m + \lfloor ta_i/m \rfloor \cdot kt \\ &= b_i + \lfloor ta_i/m \rfloor \cdot (m + kt), \end{aligned}$$

woraus wegen $b_i < m < m + kt$ folgt, dass

$$b_i = t(a_i + k \cdot \lfloor ta_i/m \rfloor) \bmod (m + kt)$$

gilt. Damit ist gezeigt, dass B durch modulare Multiplikation mit $(t, m + kt)$ aus $A(k)$ entsteht.

Entsteht B aus A durch strenge modulare Multiplikation mit (t, m), so gilt

$$\sum_{i=1}^{n} a_i < m\,.$$

Daraus ergibt sich

$$\begin{aligned} \sum_{i=1}^{n}(a_i + k \cdot \lfloor ta_i/m \rfloor) &< m + \sum_{i=1}^{n} k \cdot \lfloor ta_i/m \rfloor \\ &\leq m + k \cdot \lfloor \frac{t \cdot \sum_{j=1}^{i-1} a_j}{m} \rfloor \\ &\leq m + k \cdot \lfloor t \rfloor \\ &= m + kt\,, \end{aligned}$$

womit die modulare Multiplikation mit $(t, m + kt)$ ebenfalls streng ist. □

Es sei B ein super-erreichbarer Vektor. Dann gibt es einen super-wachsenden Vektor A und natürliche Zahlen t und m derart, dass B aus A durch strenge modulare Multiplikation entsteht. Dabei können wir annehmen, dass $\lfloor ta_n/m \rfloor > 0$ (oder gleichwertig $ta_n \geq m$) gilt, da sonst B auch super-wachsend ist und die Entschlüsselung für jeden Empfänger einfach wäre. Entsprechend den Lemmata 7.5 und 7.6 gibt es dann für B unendlich viele super-wachsende Vektoren, aus denen B mittels strenger modularer Multiplikation gewonnen werden kann, nämlich alle Vektoren $A(k)$ der wachsenden Folge zu A bez. (t, m).

Wir nehmen nun an, dass B aus A durch modulare Multiplikation mit (t, m) entsteht, die nicht streng ist. Dann gilt

$$m \leq \sum_{i=1}^{n} a_i\,. \tag{7.1}$$

Falls nun

$$\sum_{i=1}^{n} \lfloor ta_i/m \rfloor < t \tag{7.2}$$

gilt, so erfüllt jede natürliche Zahl

$$y \geq y' = \frac{\left(\sum_{i=1}^{n} a_i\right) - m}{t - \sum_{i=1}^{n} \lfloor ta_i/m \rfloor} + 1$$

die Beziehung

$$\sum_{i=1}^{n} a_i + y \sum_{i=1}^{n} \lfloor ta_i/m \rfloor < m + yt\,.$$

Durch die Wahl $k = y$ erhalten wir dann, dass B durch strenge modulare Multiplikation aus $A(k)$ entsteht.

Ist dagegen (7.1) erfüllt und (7.2) nicht erfüllt, so gilt für jedes y offensichtlich

$$\sum_{i=1}^{n} a_i + y \sum_{i=1}^{n} \lfloor ta_i/m \rfloor \geq m + yt\,.$$

Folglich ist bei Gültigkeit von (7.1) das Entstehen von B durch strenge modulare Multiplikation aus einem Element der wachsenden Folge von A bez. (t, m) genau dann möglich, wenn auch (7.2) gilt.

Wir setzen nun

$$z = \begin{cases} 0 & \text{falls (7.1) nicht gilt} \\ \lceil y' \rceil & \text{falls (7.1) und (7.2) gelten} \end{cases}$$

Dann ist nach Obigem und Lemma 7.6 gesichert, dass B von jedem $A(k)$ mit $k \geq z$ mittels strenger modularer Multiplikation erreicht werden kann.

Wir wollen uns nun überlegen, ob auch ein Superwachstum von $A(k)$ erreicht werden kann, wenn dies bei A noch nicht gegeben ist. Wenn A nicht super-wachsend ist, so gibt es ein i mit

$$a_i \leq \sum_{j=1}^{i-1} a_j\,. \tag{7.3}$$

Gilt nun

$$\sum_{j=1}^{i-1} \lfloor ta_j/m \rfloor < \lfloor ta_i/m \rfloor\,, \tag{7.4}$$

so erfüllt jedes

$$x_i \geq x_i' = \frac{\left(\sum_{j=1}^{i-1} a_j\right) - a_i}{\lfloor ta_i/m \rfloor - \sum_{i=1}^{n} \lfloor ta_i/m \rfloor} + 1$$

die Beziehung

$$\sum_{i=1}^{n} a_i + x_i \sum_{i=1}^{n} \lfloor ta_i/m \rfloor < a_i + x_i \lfloor ta_i/m \rfloor\,.$$

Damit ist für i die Forderung des Superwachstums bei $A(x)$ erfüllt. Analog zu Obigem überlegt man sich, dass Superwachstum genau dann erreicht werden kann, wenn für jedes i, $1 \leq i \leq n$, bei Gültigkeit von (7.3) auch (7.4) gelten muss.

Wir setzen jetzt für $1 \leq i \leq n$

$$z_i = \begin{cases} 0 & \text{falls (7.3) nicht gilt} \\ \lceil x_i' \rceil & \text{falls (7.3) und (7.4) gelten} \end{cases}$$

Hierdurch sichern wir nach Lemma 7.6 das Superwachstum hinsichtlich i für $A(k)$ mit $k \geq z_i$.

Wir fassen die vorstehenden Überlegungen in dem folgenden Lemma zusammen.

Lemma 7.7 *Es seien A, m und t wie in Definition 7.4 gegeben. Ferner sei B aus A mittels modularer Multiplikation mit (t, m) entstanden.*

i) Dann gibt es genau dann ein $k \geq 0$ derart, dass B aus dem super-wachsenden Vektor $A(k)$ der wachsenden Folge von A bez. (t, m) durch strenge modulare Multiplikation entsteht, wenn

mit (7.1) auch (7.2) gilt

und

für jedes i, $1 \leq i \leq n$, mit (7.3) auch (7.4) gilt.

ii) Falls B aus einem super-wachsenden Vektor $A(k)$ der wachsenden Folge von A bez. (t, m) durch strenge modulare Multiplikation entsteht, so sind alle $A(l)$ mit $l \geq \max\{z, z_1, z_2, \ldots, z_n\}$ super-wachsend und B entsteht aus jedem $A(l)$ durch strenge modulare Multiplikation. □

Lemma 7.8 *Der Vektor $B = (b_1, b_2, \ldots, b_n)$ entstehe aus dem Vektor $A = (a_1, a_2, \ldots, a_n)$ durch (strenge) modulare Multiplikation mit (t, m). Ferner seien*

$$A_1 = (\lfloor ta_1/m \rfloor, \lfloor ta_2/m \rfloor, \ldots, \lfloor ta_n/m \rfloor),\ t_1 = -m \bmod t \text{ und } m_1 = t\,.$$

Falls $t < m$ und $t > \max\{b_1, b_2, \ldots, b_n\}$ gilt, so entsteht B auch durch (strenge) modulare Multiplikation mit (t_1, m_1) aus A_1. □

Wir verzichten auf einen Beweis, da er ähnlich zu dem von Lemma 7.5 und 7.6 geführt werden kann.

Wir merken aber an, dass aus den Voraussetzungen und Setzungen $ggT(t_1, m_1) = 1$ und $t_1 < t$ folgen. Daher kann dieser Prozess iteriert werden, bis der Faktor kleiner als $\max\{b_1, b_2, \ldots, b_n\}$ ist. Daher haben wir folgende Folgerung.

Folgerung 7.9 *Wenn $B = (b_1, b_2, \ldots, b_n)$ super-erreichbar ist, dann entsteht B aus einem super-wachsenden Vektor A durch strenge modulare Multiplikator mit einem Faktor, der kleiner als $\max\{b_1, b_2, \ldots, b_n\}$.* □

Wir wollen nun noch den Modulus einschränken. Dazu definieren für einen Vektor eine Folge weiterer Vektoren, die in gewisser Weise als dual zu der wachsende Folge angesehen werden kann.

Definition 7.5 *Es seien A ein Vektor und t und m natürliche Zahlen mit $ggT(t, m) = 1$. Wir definieren nun induktiv eine Folge von Vektoren $A(-k)$ durch folgende Setzungen:*

- *Es gilt* $A(-0) = A$.
- *Ist* $A(-k) = (d_1, d_2, \ldots, d_n)$ *bereits definiert und gilt* $m - kt > \max\{d_1, d_2, \ldots, d_n\}$, *so setzen wir*

$$A(-k-1) = (d_1 - \lfloor td_1/(m-kt) \rfloor, d_2 - \lfloor td_2/(m-kt) \rfloor, \ldots, d_n - \lfloor td_n/(m-kt) \rfloor)\,.$$

Die Folge der $A(-k)$ *heißt fallende Folge zu* A *bez.* (t, m).

Im Gegensatz zur wachsende Folge zu A bez. (t, m) ist die fallende Folge endlich, da die Komponenten stets verkleinert werden bzw. einmal die Situation eintritt, dass $m - kt$ kleiner ist als das Maximum der Komponenten von $A(-k)$

Lemma 7.10 *i) Ist der Vektor* $A = (a_1, a_2, \ldots, a_n)$ *monoton wachsend, dann sind auch die Vektoren* $A(-k)$ *jeder fallenden Folge zu* A *monoton wachsend.*

ii) Ist $A(-k)$ *ein Element der fallende Folge von* A *bez.* (t, m). *Dann ist* A *das* k*-te Elemente der wachsenden Folge von* $A(-k)$ *bez.* $(t, m - kt)$. □

Lemma 7.11 *Der Vektor* $B = (b_1, b_2, \ldots, b_n)$ *resultiere aus* A *durch modulare Multiplikation mit* (t, m), *wobei*

$$m > 2 \cdot \max\{b_1, b_2, \ldots, b_n\} \text{ und } t \leq \max\{b_1, b_2, \ldots, b_n\}$$

gelte. Dann resultiert B *auch aus dem Vektor* $A(-1)$ *durch modulare Multiplikation mit* $(t, m - t)$. □

Erneut können wir dieses Lemma iterieren und erhalten unter Beachtung von Folgerung 7.9 die folgende Aussage.

Folgerung 7.12 *Wenn* $B = (b_1, b_2, \ldots, b_n)$ *durch modulare Multiplikation aus einem Vektor entsteht, dann gibt es einen Vektor* A *aus dem* B *durch modulare Multiplikation mit* (t, m) *entsteht, wobei*

$$t \leq \max\{b_1, b_2, \ldots, b_n\} \text{ und } m \leq 2 \cdot \max\{b_1, b_2, \ldots, b_n\}$$

gelten. □

Wir kombinieren nun unsere Resultate und erhalten den folgenden Satz.

Satz 7.13 *Der Vektor* $B = (b_1, b_2, \ldots, b_n)$ *ist genau dann super-erreichbar, wenn es einen monoton wachsenden Vektor* A, *natürliche Zahlen* t *und* m *mit*

$$t \leq \max\{b_1, b_2, \ldots, b_n\},\ m \leq 2 \cdot \max\{b_1, b_2, \ldots, b_n\} \text{ und } ggT(t, m) = 1$$

derart gibt, dass B *aus* A *durch modulare Multiplikation mit* (t, m) *entsteht und* A *die Bedingung erfüllt, dass*

mit (7.1) auch (7.2) gilt

und

für jedes i, $1 \leq i \leq n$, mit (7.3) auch (7.4) gilt. □

Aus Satz 7.13 ergibt sich direkt ein Algorithmus, wie der Kryptoanalyst ein super-wachsendes Rucksack-Problem gewinnen kann, dass die gleiche Lösung besitzt wie das ihm vorliegende Rucksack-Problem (B, b). Sei $B = (b_1, b_2, \ldots, b_n)$. Zuerst bestimmt der Kryptoanalyst zwei Zahlen t und m mit

$$t \leq \max\{b_1, b_2, \ldots, b_n\},\ m \leq 2 \cdot \max\{b_1, b_2, \ldots, b_n\} \text{ und } ggT(t, m) = 1$$

und berechnet t^{-1} mod m. Dann bestimmt er $A = t^{-1}B$ mod m. Falls A nicht monoton wachsend ist, wählt er ein neues Paar mit obigen Eigenschaften. Falls A wachsend ist, überprüft der Kryptoanalyst, ob für A gilt, dass mit (7.1) auch (7.2) gilt und für jedes i, $1 \leq i \leq n$, mit (7.3) auch (7.4) gilt. Entsprechend Lemma 7.7 ii) kann er dann ein super-wachsendes A', den Faktor t' und den Modulus m' ermitteln, aus dem B durch strenge modulare Multiplikation mit (t', m') hervorgeht. Anstelle von (B, b) löst er nun in linearer Zeit $(A', (t')^{-1}b \text{ mod } m)$, das die gleiche Lösung wie (B, b) besitzt. Da nach Annahme des Kryptoanalysten B durch strenge modulare Multiplikation entstanden ist und der Algorithmus daher ein A' in polynomialer Zeit liefert, kann er das scheinbar $\mathbb{NP}$-vollständige Problem (B, b) in polynomialer Zeit lösen.

7.3 Systeme auf der Basis von formalen Sprachen

7.3.1 Iterierte Morphismen

Wir geben nun ein kryptographisches System mit öffentlichen Schlüssel an, das in [17] eingeführt wurde und das auf Lindenmayer-Systemen beruht, die 1968 von Aristid Lindenmayer als formalsprachliches Modell zur Beschreibung der Entwicklung niederer Organismen eingeführt wurden. Wir geben zuerst die notwendigen Definitionen, wobei wir nur die Spezialfälle betrachten, die für das kryptographische System von Interesse sind. Für weitere Details der Theorie der Lindenmayer-Systeme verweisen wir auf [14].

Seien X und Y zwei Alphabete. Eine endliche Substitution σ von X^* in Y^* ist eine Abbildung, die jedem Wort über X eine endliche Menge von Wörtern über Y zuordnet und $\sigma(xy) = \sigma(x)\sigma(y)$ für beliebige Wörter $x \in X^*$ und $y \in X^*$ erfüllt. Offensichtlich reicht es zur Angabe einer endlichen Substitution σ die Mengen $\sigma(a)$ mit $a \in X$ anzugeben, da sich dann für ein Wort $a_1 a_2 \ldots a_n$ mit $a_i \in X$

$$\sigma(a_1 a_2 \ldots a_n) = \sigma(a_1)\sigma(a_2)\ldots\sigma(a_n)$$

ergibt. Eine Substitution σ heißt Morphismus, falls $\sigma(a)$ für jedes $a \in X$ einelementig ist. In dem Fall schreiben wir einfach $\sigma(a) = v$ anstelle von $\sigma(a) = \{v\}$.

Definition 7.6 *Ein T0L-System ist ein Quadrupel $G = (V, \sigma_0, \sigma_1, w)$, wobei*
- *V ein Alphabet ist,*
- *σ_0 und σ_1 endliche Substitutionen von V^* in V^* sind, und*
- *w ein nichtleeres Wort über V ist.*

Ein T0L-System heißt DT0L-System, falls σ_0 und σ_1 Morphismen sind.[1]

Im Fall von DT0L-Systemen benutzen wir anstelle von σ_0 und σ_1 zur Bezeichnung der Morphismen h_0 und h_1.

Es sei ein T0L-System $G = (V, \sigma_0, \sigma_1, w)$ gegeben. Einem Binärwort

$$x_1 x_2 \dots x_n \in \{0,1\}^*$$

ordnen wir die Menge

$$\sigma_G(x_1 x_2 \dots x_{n-1} x_n) = \sigma_{x_n}(\sigma_{x_{n-1}}(\dots(\sigma_{x_2}(\sigma_{x_1}(w)))\dots))$$

zu, d. h. wir wenden die Substitutionen in der Reihefolge iteriert auf w an, die durch das Binärwort gegeben ist. Offensichtlich gilt stets $\sigma_G(\lambda) = \{w\}$.

Bei einem DT0L-System G ist offenbar für jedes $x \in \{0,1\}^*$ die Menge $\sigma_G(x)$ einelementig. Daher schreiben wir hier auch nur $\sigma_G(x) = v$ anstelle von $\sigma_G(x) = \{v\}$.

Beispiel 7.4 Wir betrachten das T0L-System

$$G_1 = (\{a, b, c\}, \sigma_0, \sigma_1, abc)$$

mit

$$\sigma_0(a) = \{a, aa\},\ \sigma_0(b) = \{b\},\ \sigma_0(c) = \{c^2\},$$
$$\sigma_1(a) = \{a\},\ \sigma_1(b) = \{b^2\},\ \sigma_1(c) = \{c^2\}.$$

Da für jeden Buchstabe $x \in V$ und jedes $i \in \{0,1\}$ die Menge $\sigma_i(x)$ eine (endliche) Teilmenge von $\{x\}^*$ ist, wird die Reihenfolge der Buchstaben des Anfangswortes w nicht verändert, d. h. zuerst kommen alle a's, dann alle b's und schließlich alle c's.

Da bei σ_0 jedes Vorkommen von a entweder durch ein a oder durch zwei a ersetzt wird, entstehen aus k Buchstaben a mindestens k Vorkommen von a und höchstens $2k$ Vorkommen von a. Ferner bleibt bei Anwendung von σ_0 die Anzahl der Buchstaben b unverändert, da jedes b durch genau ein b ersetzt wird. Die Anzahl der Buchstaben c wird dagegen bei Anwendung von σ_0 verdoppelt, da jeder Buchstabe c durch das Wort cc ersetzt wird. Folglich erhalten wir

$$\sigma_0(a^r b^s c^t) = \{a^u b^s c^{2t} \mid r \le u \le 2r\}.$$

Analog überlegt man sich

$$\sigma_1(a^r b^s c^t) = \{a^r b^{2s} c^{2t}\}.$$

[1] Die Buchstaben bei der Bezeichnung der Systeme haben in der Theorie formaler sprachen folgende Bedeutung: L erinnert an den A. Lindenmayer; 0 (Null) steht dafür, dass die Substitution eines Buchstaben unabhängig von den ihn umgebenden Buchstaben erfolgt (es gibt in der Theorie auch Systeme, bei denen eine Abhängigkeit der substitution von den benachbarter Buchstaben besteht; T steht für tabelliert, da wir mehrere Substitutionen (auch Tabelle genannt) betrachten; D steht für deterministisch, da jeder Buchstabe in eindeutiger weise ersetzt wird.

Damit ergeben sich folgende Mengen:

$$\begin{aligned}
\sigma_{G_1}(0) &= \{abc^2, a^2bc^2\},\\
\sigma_{G_1}(1) &= \{ab^2c^2\},\\
\sigma_{G_1}(00) &= \{abc^4, a^2bc^4, a^3bc^4, a^4bc^4\},\\
\sigma_{G_1}(01) &= \{ab2c^4, a^2b^2c^4\},\\
\sigma_{G_1}(10) &= \{ab^2c^4, a^2b^2c^4\},\\
\sigma_{G_1}(11) &= \{ab^4c^4\}
\end{aligned}$$

und allgemein für ein Wort $x \in \{0,1\}^n$ der Länge n mit $y = \#_0(x)$ und $z = \#_1(x)$

$$\sigma_{G_1}(x) = \{a^u b^{2^z} c^{2^n} \mid 1 \leq u \leq 2^y\}\,.$$

$\Diamond$

Beispiel 7.5 Wir betrachten das DT0L-System $G_2 = (\{a,b\}, h_0, h_1, ab)$ mit

$$h_0(a) = \lambda,\ h_0(b) = (ab)^2, \quad \text{und} \quad h_1(a) = (ab)^2,\ h_1(b) = \lambda\,.$$

Bei Anwendung von h_0 wird jedes Vorkommen von a gestrichen und jeder Buchstabe b durch $abab$ ersetzt. Folglich ist $h_0(x) = (ab)^r$ mit $r = \#_b(v)$. Analog ergibt sich $h_1(v) = (ab)^s$ mit $s = \#_a(v)$. Da außerdem jedes Wort $h_i(x)$ die gleiche Anzahl von a und b enthält und dies auch auf das Startwort w zutrifft, erhalten wir für jedes Wort $x \in \{0,1\}^*$ der Länge n

$$\sigma_{G_2}(x) = (ab)^{2^n}\,.$$

$\Diamond$

Beispiel 7.6 Wir betrachten das DT0L-System $G_3 = (\{a,b\}, h_0, h_1, ab)$ mit

$$h_0(a) = ab,\ h_0(b) = b, \quad \text{und} \quad h_1(a) = a,\ h_1(b) = ba\,.$$

Man rechnet leicht nach, dass folgende Beziehungen gelten:

$$\begin{aligned}
\sigma_{G_3}(0) &= abb,\\
\sigma_{G_3}(1) &= aba,\\
\sigma_{G_3}(00) &= abbb,\\
\sigma_{G_3}(01) &= ababa,\\
\sigma_{G_3}(10) &= abbab,\\
\sigma_{G_3}(11) &= abaa
\end{aligned}$$

usw. $\Diamond$

Um die Verschlüsselung eines Wortes $x_1x_2\ldots x_n \in \{0,1\}^*$ durch ein T0L-System $G = (V, \sigma_0, \sigma_1, w)$ vorzunehmen, wählen wir ein Wort aus $\sigma_G(x_1x_2\ldots x_{n-1}x_n)$ aus.

Bei der Entschlüsselung bzw. durch den Kryptanalysten ist dann zu gegebenen Wort $v \in V^*$ ein Binärwort x derart zu finden, dass $v \in \sigma_G(x)$ gilt. Dies ist das klassische Parsing aus der Theorie formaler Sprachen. Hierfür gilt folgende Aussage (siehe [14]), die wir ohne Beweis angeben.

Satz 7.14

i) *Das Parsing-Problem*

Gegeben: *T0L-System* $G = (V, \sigma_0, \sigma_2, w)$ *und* $v \in V^*$
Gesucht: *Binärwort* $x \in \{0,1\}^*$ *mit* $v \in \sigma_G(x)$
(oder man treffe die Aussage, dass ein derartiges Wort nicht existiert)

ist $\mathbb{NP}$*-vollständig.*

ii) *Das Parsing-Problem ist für DT0L-Systeme in polynomialer Zeit lösbar.*

Wegen Satz 7.14 scheinen T0L- bzw. DT0L-Systeme auf den ersten Blick für ein kryptographisches System mit öffentlicher Verschlüsselungsmethode geeignet. Dazu wählen wir entsprechend dem Vorgehen in Abschnitt 7.1 im ersten Schritt das Parsing-Problem für T0L-Systeme und nehmen die Verschlüsselung wie oben beschrieben vor. Im zweiten Schritt entscheiden wir uns dann für das Parsing-Problem für DT0L-Systeme als einfacher Variante. Jedoch gibt es dabei noch ein Problem: Zu einem $v \in V^*$ darf es höchstens ein Binärwort x mit $v \in \sigma_G(x)$ geben, da sonst die Entschlüsselung kein eindeutiges Ergebnis liefert. Daher schränken wir die Systeme weiter ein.

Definition 7.7 *Ein DT0L-System* $G = (V, h_0, h_1, w)$ *heißt rückwärts eindeutig, wenn aus*

$$h_{i_n}(h_{i_{n-1}}(\dots(h_{i_1}(w))\dots)) = h_{j_m}(h_{j_{m-1}}(\dots(h_{j_1}(w))\dots))$$

folgt, dass

$$m = n \quad \textit{und} \quad i_k = j_k \textit{ für } 1 \le k \le n$$

gelten.

Intuitiv ist ein DT0L-System rückwärts eindeutig, wenn zu jedem Wort $v \in V^*$ höchstens ein x mit $v \in \sigma_G(x)$ gibt, denn die Forderung, dass aus $v \in \sigma_G(x)$ und $v \in \sigma_G(y)$ die Gleichheit $x = y$ folgt, ist offenbar gleichwertig zu der aus Definition 7.7.

Von den oben angegebenen DT0L-Systemen ist G_2 (Beispiel 7.5) nicht rückwärts eindeutig, denn für $v = (ab)^{2^n}$ mit $n \ge 1$ gilt $v \in \sigma_{G_2}(x)$ für jedes Wort $x \in \{0,1\}^*$ der Länge n. Dagegen ist G_3 rückwärts eindeutig. Dies ist wie folgt zu sehen: Da sowohl $h_0(a)$ als auch $h_0(b)$ auf b enden, endet auch $h_0(z)$ für jedes Wort $z \in \{a,b\}^*$ auf b. Analog ergibt sich, dass $h_1(z)$ für jedes Wort $z \in \{a,b\}^*$ auf a endet. Daher kann am letzten Buchstaben eindeutig erkannt werden, welcher Morphismus als letzter angewendet wurde. Ferner kann auch das Wort, auf das dieser Morphismus angewendet wurde, eindeutig rekonstruiert werden. Man geht von hinten immer wieder zu dem vorhergehenden Vorkommen des letzten Buchstaben. Wir demonstrieren dieses Vorgehen an einem Beispiel, wobei wir die Wörter immer einmal wiederholen und dabei die Zerlegung entsprechend dem letzten Buchstaben vornehmen. Es ergibt sich für das Wort *ababaaba*

$$\begin{aligned} ababaaba &= a\ ba\ ba\ a\ ba = h_1(abbab) \\ &= h_1(ab\ b\ ab) = h_1(h_0(aba)) \\ &= h_1(h_0(a\ ba)) = h_1(h_0(h_1(ab)) \end{aligned}$$

Daher gilt $ababaaba = \sigma_{G_3}(101)$.

Bei Verwendung von rückwärts eindeutigen DT0L-Systemen ergibt sich eine polynomiale Entschlüsselung.

Wir haben nun entsprechend dem dritten Schritt des Verfahrens aus Abschnitt 7.1 eine Transformation des DT0L-Systems $G = (V, h_0, h_1, w)$ vorzunehmen, so dass ein T0L-System Q entsteht, das wie ein beliebiges T0L-System aussieht und aus dem G nur schwer zu rekonstruieren ist.

Dazu sei X ein weiteres Alphabet (mit viel mehr Buchstaben als V) und g ein Morphismus von X^* in V^* mit folgenden Eigenschaften:

- Für jedes $a \in X$ gilt $g(a) \in V \cup \{\lambda\}$.
- Für jedes $b \in V$ gibt es mindestens ein $a_b \in X$ mit $g(a_b) = b$.

Wir konstruieren nun aus $G = (V, h_0, h_1, w)$ ein T0L-System $G' = (X, \sigma_0, \sigma_1, w')$, wobei folgende Bedingungen erfüllt sein müssen:

- $\sigma_i(a) \subseteq g^{-1}(h_i(g(a)))$ für $a \in X$ und $i \in \{0, 1\}$,
- $w' \in g^{-1}(w)$.

Während durch die Angabe von G' eine öffentlich bekannte Verschlüsselung erreicht wird, bleibt g geheim.

Beispiel 7.7 Wir gehen von dem DT0L-System $G_3 = (\{a, b\}, h_0, h_1, ab)$ mit

$$h_0(a) = ab,\ h_0(b) = b, \quad \text{und} \quad h_1(a) = a,\ h_1(b) = ba$$

aus Beispiel 7.6 aus. Wir wählen

$$X = \{k, l, m, n, p\} \text{ und } g(k) = g(n) = a,\ g(m) = g(p) = b,\ g(l) = \lambda\,.$$

Für praktische Belange ist die Anzahl der Elemente in X viel zu klein, aber zur Demonstration der Idee reicht diese Transformation. Entsprechend der Forderung an σ_0 haben wir für $\sigma_0(k)$ eine endliche Teilmenge von

$$g^{-1}(h_0(g(k))) = g^{-1}(ab) = \bigcup_{r\geq 0}\bigcup_{s\geq 0}\bigcup_{t\geq 0}\{l^r\}\{k, n\}\{l^s\}\{m, p\}\{l^t\}$$

zu wählen und analog für die anderen Fälle vorzugehen. Wir wählen

$$\begin{array}{ll}
\sigma_0(k) = \{kllm, lnp, kpl\}, & \sigma_1(k) = \{lln\}, \\
\sigma_0(l) = \{ll, l\}, & \sigma_1(l) = \{lll\}, \\
\sigma_0(m) = \{lpl, ml\}, & \sigma_1(m) = \{mlk, mln\}, \\
\sigma_0(n) = \{lnp, kpl\}, & \sigma_1(n) = \{kll\}, \\
\sigma_0(p) = \{lml, pl\}, & \sigma_1(k) = \{lmk, pln\}.
\end{array}$$

Ferner wählen wir $w' = klm$ aus $g^{-1}(ab)$. Für das so entstehende T0L-System $G'_3 = (X, \sigma_0, \sigma_1, klm)$ ergeben sich u. a.

$$\begin{aligned}
\sigma_{G'_3}(0) = \{&kllmlllpl, kllmllml, kllmllpl, kllmlml, lnplllpl, lnpllml,\\
&lnpllpl, lnplml, kpllllpl, kplllml, kplllpl, kpllml\},\\
\sigma_{G'_3}(1) = \{&llnlllmlk, llnlllmln\},\\
\sigma_{G'_3}(11) = \{&l^6kl^{11}mlkl^5n, l^6kl^{11}mlkl^3kl^2, l^6kl^{11}mlnl^5n, l^6kl^{11}mlnl^3kl^2\}\,.
\end{aligned}$$

Auf die Angabe weiterer Mengen verzichten wir hier, da z.B. $\sigma_{G_3'}(00)$ und $\sigma_{G_3'}(01)$ bereits $3(2^8 + 3 \cdot 2^7 + 2 \cdot 2^6 + 2^5) + 2^8 + 2 \cdot 2^7 + 2^6$ bzw. 24 Wörter enthalten. Schon am großen Unterschied der Mengen $\sigma_{G_3}(x)$ und $\sigma_{G_3'}(x)$ erkennt man, dass die Verschleierung der Eigenschaften von G_3 durch den Übergang zu G_3' als gelungen angesehen werden kann. $\Diamond$

Es sei $G = (V, h_0, h_1, w)$ ein DT0L-System und $G' = (X, \sigma_0, \sigma_1, w)$ sei das zugehörige T0L-System entsprechend obiger Konstruktion. Wir geben nun eine Beziehung zwischen den Mengen $\sigma_G(x)$ und $\sigma_{G'}(x)$ an.

Es sei $i \in \{0, 1\}$. Wir betrachten $u \in \sigma_i(b)$ für ein $b \in X$. Dann gilt

$$g(u) = g(g^{-1}(h_i(g(b)))) \subseteq h_i(g(b))\,.$$

Da $h_i(g(b))$ einelementig ist, ergibt sich $g(u) = h_i(g(b))$. Dies setzt sich homomorph auf Wörter fort, d. h. für $u \in \sigma_i(v)$ gilt ebenfalls $g(u) = h_i(g(v))$.

Damit erhalten wir für $u \in \sigma_i(\sigma_j(v))$, $i, j \in \{0, 1\}$,

$$g(u) = h_i(g(\sigma_j(v))) = h_i(h_j(g(v)))\,.$$

Induktiv erhalten wir daraus, dass aus $u \in \sigma_{G'}(x)$ folgt, dass $g(u) = \sigma_G(x)$ ist.

Wie löst nun der befugte Empfänger die Dechiffrierung. Er nutzt einfach die Kenntnis von g und $G = (V, h_1, h_2, w)$ aus. Er berechnet zu einem gegeben Wort u einfach $g(u)$ und durch Lösung des Parsing-Problems für $g(u)$ bezüglich des DT0L-Systems G ermittelt er die Folge x.

Der Kryptoanalyst kennt g und G nicht. Er hat also das eigentlich $\mathbb{NP}$-vollständige Parsing-Problem für u bezüglich G' zu lösen. Eine anderes Vorgehen – analog zu dem beim öffentlichen Schlüsselsystem auf der Basis des Rucksack-Problems – würde darin bestehen, ein $g'' : X \to V''$ und $G'' = (V'', h_0'', h_1'', w'')$ so zu finden, dass G' aus G'' mittels g'' erzeugt werden kann und G'' rückwärts eindeutig ist. Mit Hilfe von G'' könnte er dann – analog zum befugten Empfänger – die gesuchte Folge x zu u ermitteln.

Ein G'' kann dadurch gewonnen werden, indem eine Teilmenge von X von V'', eine Abbildung g'' mit $g''(a) \in V^*$ für $a \in X$ und für $h_i''(a)$, $a \in V''$, einfach ein Element aus $\sigma_i(a)$ gewählt werden. Man beachte, dass hier sehr viele Wahlen bei großen Mengen V'', $\sigma_0(a)$ und $\sigma_1(a)$ für $a \in V''$ bestehen.

Noch schwieriger gestaltet sich aber der zweite Teil, d. h. die Entscheidung, ob G'' rückwärts eindeutig ist, denn es gilt der folgende Satz.

Satz 7.15 ([3]) *Es ist algorithmisch nicht entscheidbar, ob ein gegebenes DT0L-System rückwärts eindeutig ist.*

Beweis. Wir nehmen eine Reduktion auf das algorithmisch unentscheidbare Postsche Korrespondenzproblem (siehe [7])

Gegeben:	Alphabet W mit mindestens 2 Buchstaben, Wortpaare $(u_1, v_1), (u_2, v_2), \ldots, (u_t, v_t)$ mit $u_i, v_i \in V^+$,
Frage:	Gibt es eine Folge $(i_1, i_2, \ldots, i_k)$ derart, dass $u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$ gilt.

vor. Dazu betrachten wir das DT0L-System $G = (V, h_0, h_1, 0)$ mit

$$V = W \cup \{z, 0, 1, 2, \ldots, t, 1', 2', \ldots t'\} \cup \{[i] \mid 1 \leq i \leq t\}$$

und

$$\begin{array}{ll}
h_0(0) = 1, & h_1(0) = 1', \\
h_0(i) = i + 1 \text{ für } 1 \leq i < t, & h_1(i) = u_i 1[i] \text{ für } 1 \leq i \leq t, \\
h_0(i') = (i+1)' \text{ für } 1 \leq i < t, & h_1(i') = v_i 1'[i] \text{ für } 1 \leq i \leq t, \\
h_0(t) = h_0(t') = z, & \\
h_0(z) = [1]z, & h_1(z) = [2]z, \\
h_0(a) = a \text{ für } a \in W \cup \{[i] \mid 1 \leq i \leq t\}, & h_1(a) = a \text{ für } a \in W \cup \{[i] \mid 1 \leq i \leq t\},
\end{array}$$

Es ist leicht zu sehen, dass für

$$1 \leq i \leq t,\ s \geq 0,\ 1 \leq j \leq s\ \ 1 \leq k_j \leq t,\ m \geq 0$$

folgende Relationen bestehen:

$$\begin{aligned}
\sigma_G(00^{k_1-1}10^{k_2-1}\ldots 10^{k_s-1}10^{i-1}) &= u_{k_1}u_{k_2}\ldots u_{k_s}i[k_s]\ldots[k_2][k_1]\,, \\
\sigma_G(10^{k_1-1}10^{k_2-1}\ldots 10^{k_s-1}10^{i-1}) &= v_{k_1}v_{k_2}\ldots v_{k_s}i'[k_s]\ldots[k_2][k_1]\,, \\
\sigma_G(00^{k_1-1}10^{k_2-1}\ldots 10^{k_s-1}10^{t}i_1\ldots i_m) &= u_{k_1}u_{k_2}\ldots u_{k_s}[i_1+1]\ldots[i_m+1]z[k_s]\ldots[k_1]\,, \\
\sigma_G(10^{k_1-1}10^{k_2-1}\ldots 10^{k_s-1}10^{t}i_1\ldots i_m) &= v_{k_1}v_{k_2}\ldots v_{k_s}[i_1+1]\ldots[i_m+1]z[k_s]\ldots[k_1]\,.
\end{aligned}$$

Hieraus ist sofort zu sehen, dass ein Wort höchstens dann in zwei verschiedenen Mengen $\sigma_G(x)$ und $\sigma_G(y)$ gilt, wenn sowohl

$$x = 00^{k_1-1}10^{k_2-1}\ldots 10^{k_s-1}10^{t}i_1\ldots i_m \text{ und } y = 10^{k_1-1}10^{k_2-1}\ldots 10^{k_s-1}10^{t}i_1\ldots i_m \quad (7.5)$$

als auch

$$u_{k_1}u_{k_2}\ldots u_{k_s} = u_{k_1}v_{k_2}\ldots v_{k_s} \quad (7.6)$$

gelten. Folglich hat dann das Postsche Korrespondenzproblem eine Lösung. Umgekehrt folgt aus einer Gleichheit (7.6), dass für die Binärwörter x und y aus (7.5) die Gleichheit $\sigma_G(x) = \sigma_G(y)$ besteht. Damit ist G genau dann rückwärts eindeutig, wenn das Postsche Korrespondenzproblem keine Lösung hat. Somit ist die Unentscheidbarkeit der Frage, ob G rückwärts eindeutig ist, gezeigt. □

Es ist natürlich anzumerken, dass das aus dem Postschen Korrespondenzproblem konstruierte DT0L-System nicht als Ausgangspunkt für ein öffentliches Schlüsselsystem genommen wird, so dass für die vom Kryptoanalysten konstruierten Systeme G'' das Problem, ob G'' rückwärts eindeutig ist, noch entscheidbar sein kann.

In den Arbeiten [8] hat Jarkko Kari gezeigt, dass der Kryptoanalysten unter der Kenntnis, dass das öffentliche Schlüsselsystem G' mittels obiger Konstruktion aus einem rückwärts eindeutigen DT0L-System gewonnen wurde, in polynomialer Zeit zu einem u die zugehörige Binärfolge x mit $u \in \sigma_{G'}(x)$ gewinnen kann.

7.3.2 Ein kryptographisches System auf der Basis des Mitgliedsproblems

In diesem Abschnitt gehen wir davon aus, dass der Leser mit den Grundbegriffen der Theorie formaler Sprachen und der Automatentheorie, wie sie üblicherweise in einer Grundvorlesung zur Theoretischen Informatik vermittelt werden vertraut ist. Für Einzelheiten verweisen wir auf [7]. Wir geben hier nur einige Bezeichnungen.

Eine Regelgrammatik $G = (N, T, P, S)$ spezifizieren wir durch die disjunkten Mengen N und T der Nichtterminale bzw. Terminale, die Menge P der Regeln (für die Ableitung) und das Axiom (oder Startelement) $S \in N$. Ein endlicher Automat $\mathcal{A} = (X, Z, z_0, F, \delta)$ wird durch die Menge X der Eingabesymbole, die Menge Z der Zustände, den Anfangszustand $z_0 \in Z$, die Menge $F \subseteq Z$ der akzeptierende Zustände und die Überführungsfunktion $\delta : Z \times X \to Z$ gegeben. Die von $\mathcal{A}$ akzeptierte Wortmenge ist durch

$$T(\mathcal{A}) = \{w \mid \delta^*(z_0, w) \in F\}$$

definiert.

Wir gehen wieder wie beim Verfahren zur Gewinnung eines kryptographischen Systems mit öffentlichem Schlüssel aus Abschnitt 7.1 vor.

Schritt 1. Wir gehen vom algorithmisch unentscheidbaren Mitgliedsproblem

Gegeben:	Regelgrammatik $G = (N, T, P, S)$ und Wort $w \in T^*$
Frage:	Ist w in der von G erzeugten Sprache $L(G)$ enthalten, d. h. gilt $w \in L(G)$?

Wir verwenden zur Verschlüsselung zwei Grammatiken G_0 und G_1 mit gleichem Terminalalphabet T. Die Binärfolge $x_1x_2 \ldots x_n$ wird durch das Wort $w_1\#w_2\# \ldots \#w_n$ verschlüsselt, wobei für die Teilwörter $w_i \in T^*$ jeweils $w_i \in L(G_{x_i})$ gilt. Um einen Buchstaben x_i zu verschlüsseln bestimmen wir also einfach ein Wort aus $L(G_{x_i})$ und wählen dies als w_i.

Schritt 2. Um ein einfaches Teilproblem zu erhalten, wählen wir eine reguläre Sprache L, die als akzeptierte Wortmenge $T(\mathcal{A})$ eines endlichen Automaten $\mathcal{A} = (X, Z, z_0, F, \delta)$ gegeben sei. Das Komplement $\overline{L}$ von L bez. T^* ist ebenfalls eine reguläre Sprache (und wird von $\mathcal{B} = (X, Z, z_0, Z \setminus F, \delta)$ akzeptiert). Wir konstruieren nun aus G_0 und $\mathcal{A}$ bzw. G_1 und $\mathcal{B}$ Grammatiken G_0' bzw. G_1' für die Sprachen $L(G_0) \cap L$ bzw. $L(G_1) \cap \overline{L}$ und verwenden diese Grammatiken als öffentliche Schlüssel. Die Sprache L bleibt geheim.

Ist dann ein Wort $w \in L(G_0') \cup L(G_1')$ gegeben, so hat man bei der Entschlüsselung zu entscheiden, ob $w \in L(G_0')$ oder $w \in L(G_1')$ richtig ist, d. h. man hat scheinbar das eigentlich unentscheidbare Mitgliedsproblem für zwei Regelgrammatiken zu entscheiden.

Da $L \cap \overline{L} = \emptyset$ gilt, ist auch $L(G_0') \cap L(G_1') = \emptyset$. Daher reicht es für ein Wort $w \in L(G_0') \cup L(G_1')$ anstelle von $w \in L(G_0')$ einfach $w \in L$ zu entscheiden bzw. anstelle von $w \in L(G_1')$ einfach $w \in \overline{L}$ zu entscheiden. Der befugte Empfänger testet also einfach ob $w \in L$ gilt (dies ist mittels $\mathcal{A}$ in linearer Zeit in $|w|$ möglich). Bei positiver Antwort liegt w auch in $L(G_0')$ und verschlüsselt folglich eine 0, bei negativer Antwort liegt w in $L(G_1')$ und verschlüsselt daher eine 1.

Schritt 3 wurde bei der Erklärung der vorhergehenden Schritte schon erläutert.

Die Aufgabe kann für den Kryptoanalysten noch erschwert werden, indem – in Analogie zu dem Vorgehen bei iterierten Morphismen – noch eine weitere Verschleierung durch einen

Morphismus vornimmt. Es sei die Grammatik $G' = (N, T, P, S)$ und ein Morphismus $g : X^* \to (N \cup T \cup \{\lambda\})^*$ mit großer Menge X und $g^{-1}(N \cup T \cup \{\lambda\}) = X$ und $g^{-1}(a) \neq \emptyset$ für $a \in N \cup T$ gegeben. Dann konstruieren wir die Grammatik

$$G'' = (g^{-1}(N), g^{-1}(T \cup \{\lambda\}), P'', S'')\,,$$

wobei S'' ein Element aus $g^{-1}(S)$ ist und P'' eine Teilmenge von

$$\{\alpha' \to \beta' \mid \alpha' \in g^{-1}(\alpha), \beta' \in g^{-1}(\beta), \alpha \to \beta \in P\}$$

ist, wobei für jede Regel $\alpha \to \beta \in P$ mindestens eine Regel $\alpha' \to \beta'$ mit $\alpha' \in g^{-1}(\alpha)$ und $\beta' \in g^{-1}(\beta)$ in P'' enthalten sein muss. g bleibt geheim.

Anstelle von G'_1 und G'_2 von oben werden dann die nach dieser Konstruktion entstehenden Grammatiken G''_1 und G''_2 verwendet. Der befugte Empfänger wendet auf ein Wort v einfach g an und untersucht, ob $g(v)$ in L liegt oder nicht.

7.4 Chiffrierungen auf zahlentheoretischer Basis

In den vorhergehenden Abschnitten haben wir als Ausgangspunkt zur Konstruktion von Chiffrierungen mit einem öffentlichen Schlüssel sowohl unentscheidbare Probleme als auch $\mathbb{NP}$-vollständige Probleme verwendet, da diese als schwierig zu lösen angesehen werden. Eine weitere Möglichkeit wäre die Wahl eines Problems, dessen Komplexität nicht genau bekannt ist, aber allgemein als schwierig angesehen wird. In diesem Abschnitt werden wir derartige Probleme als Basis wählen. Es handelt sich dabei um Probleme aus der Zahlentheorie: die Zerlegung einer Zahl in ihre Primfaktoren und die Bestimmung des diskreten Logarithmus (das ist das Finden einer Zahl e mit $x^e = y \bmod n$ für gegebene Zahlen x, y und n.

7.4.1 Einiges aus der Zahlentheorie

Wir benötigen einige grundlegende Kenntnisse der Zahlentheorie, die wir jetzt kurz (in einigen Fällen ohne Beweis bzw. nur mit Beweisen für Spezialfälle) zusammenstellen wollen.

Wir beginnen mit einigen Aussagen zur Komplexität der Berechnung einiger Zahlen.

Lemma 7.16 *Für zwei natürliche Zahlen a und b (und eine natürliche Zahl m) erfordert die Berechnung von a^b ($a^b \bmod m$) einen zeitlichen Aufwand $O(\log_2(b))$.*

Beweis. Es sei

$$b = b_k 2^k + b_{k-1} 2^{k-1} + \cdots + b_2 2^2 + b_1 2 + b_0\,,$$

d. h. $b_k b_{k-1} \ldots b_2 b_1 b_0$ ist die Dualdarstellung von b. Dann gilt $k = \lceil \log_2(b+1) \rceil$. Ferner gilt

$$a^b = a^{b_k 2^k + b_{k-1} 2^{k-1} + \cdots + b_2 2^2 + b_1 2 + b_0} = a^{b_k 2^k} a^{b_{k-1} 2^{k-1}} \ldots a^{b_2 2^2} a^{b_1 2} a^{a_0}\,.$$

Damit kann a^b dadurch berechnet werden, dass wir zuerst der Reihe nach durch fortgesetztes Quadrieren die Werte a, a^2, a^4, a^8 usw. bestimmen und dann das Produkt der

entsprechend der Dualdarstellung von b erforderlichen a^{2^i} bilden. Dies erfordert nur logarithmischen Aufwand in b, da dies für das Gewinnen der Dualdarstellung und die Anzahl der Quadrieroperationen und Produktbildung (und Divisionen zur Berechnung des Restes) gilt. □

Wir demonstrieren dieses Vorgehen an einem Beispiel. Wir wollen 17^{73} mod 133 berechnen. Zuerst stellen wir fest, dass $73 = 64 + 8 + 1 = 2^6 + 2^3 + 2^0$ gilt. Nun berechnen wir der Reihe nach

$$\begin{aligned}
17^1 &= 17 \bmod 133,\\
17^2 &= 289 = 23 \bmod 133,\\
17^4 &= (17^2)^2 = 23^2 = 529 = 130 \bmod 133,\\
17^8 &= (17^4)^2 = 130^2 = 16900 = 9 \bmod 133,\\
17^{16} &= (17^8)^2 = 9^2 = 81 \bmod 133,\\
17^{32} &= (17^{16})^2 = 81^2 = 6561 = 44 \bmod 133,\\
17^{64} &= (17^{32})^2 = 44^2 = 1936 = 74 \bmod 133.
\end{aligned}$$

und dann

$$17^{73} = 17^{64} \cdot 17^8 \cdot 17^1 = 74 \cdot 9 \cdot 17 = 11322 = 17 \bmod 133\,.$$

Lemma 7.17 *i) Für zwei natürliche Zahlen n und m mit $n \geq m \geq 1$ lässt sich ihr größter gemeinsamer Teiler $ggT(n,m)$ in logarithmischer Zeit in n berechnen.*

ii) Für natürliche Zahlen n und m mit $n \geq m \geq 1$ und $ggT(m,n) = 1$ lässt sich in logarithmischer Zeit in n eine Zahl a mit $ma = 1 \bmod n$ berechnen.

Beweis. i) Ohne Beschränkung der Allgemeinheit sei $n \geq m$. Wir betrachten das „Programm“

$r_{-1} = n;\ r_0 = m;\ i = 0$
WHILE $r_i \neq 0$ BEGIN $r_{i+1} = r_{i-1} \bmod r_i;\ i = i + 1$ END
$ggT = r_{i-1}$

(man beachte, dass hier mod die Funktion ist, die zu a und b den Rest $r = a \bmod b$ bei der ganzzahligen Division von a durch b gibt, d. h. $a = qb + r$).

Es ist sofort zu sehen, dass die Zahlen r_i bei jedem Durchlauf der Schleife echt kleiner werden. Folglich tritt die Abbruchbedingung nach einer gewissen Anzahl von Durchläufen ein, und daher terminiert das „Programm“.

Bei einem Abbruch nach k Durchläufen der WHILE-Schleife erhalten wir der Reihe nach die Gleichungen

$$\begin{aligned}
n &= q_1 m + r_1,\\
m &= q_2 r_1 + r_2,\\
r_1 &= q_3 r_2 + r_3,\\
&\dots\\
r_{k-4} &= q_{k-2} r_{k-3} + r_{k-2},\\
r_{k-3} &= q_{k-1} r_{k-2} + r_{k-1},\\
r_{k-2} &= q_k r_{k-1} + r_k = q_k r_{k-1} \quad (r_k = 0).
\end{aligned} \tag{7.7}$$

Sei nun d ein Teiler von n und m. Dann ist d wegen (7.7) der Reihe nach auch ein Teiler von $r_1, r_2, \ldots, r_{k-1}$. Für uns ist bedeutend, dass jeder Teiler von m und n auch ein Teiler von r_{k-1} ist.

Außerdem ist wegen der letzten Gleichung von (7.7) r_{k-1} ein Teiler von r_{k-2}. Aus den beiden letzten Gleichungen von (7.7)

$$r_{k-3} = q_{k-1}r_{k_2} + r_{k-1} = q_{k_1}q_k r_{k-1} + r_{k-1} = (q_{k_1}q_k + 1)r_{k-1} = \alpha_{k-3}r_{k-1}$$

und dann

$$r_{k-4} = q_{k-2}r_{k-3} + r_{k-2} = q_{k-2}\alpha_{k_3}r_{k-1} + q_k r_{k-1} = (q_{k-2}\alpha_{k_3} + q_k)r_{k-1} = \alpha_{k-4}r_{k-1}$$

usw. Dabei erhalten wir zum Schluss

$$m = \alpha_0 r_{k-1} \text{ und } n = \alpha_{-1}r_{k-1}.$$

Damit ist r_{k-1} ein Teiler von m und n. Folglich ist r_{k-1} sogar der größte gemeinsame Teiler von m und n.

Bei jeder Gleichung von $r_{j-2} = q_j r_{j-1} + r_j$ aus (7.7) erfolgt eine Halbierung des Wertes. Denn wenn $r_j \geq r_{j-2}/2$ wäre, so wäre wegen $r_{j-1} > r_j$ auch $r_{j-1} > r_{j-2}/2$. Weiterhin ist $q_j = 1$, da $r_{j-2} > r_{j-1} > r_{j-2}/2$ gilt. Damit ergibt sich

$$r_{j-2} = r_{j-1} + r_j > r_{j-2}/2 + r_{j-2}/2 = r_{j-2},$$

was unmöglich ist. Damit wird die WHILE-Schleife höchstens $2\log_2(n)$ durchlaufen ($r_1 \leq n$, $r_3 \leq r_1/2 \leq n/4$, $r_5 \leq r_3/2 \leq n/8$ usw.). Folglich ist die Gesamtzahl der Schritte logarithmisch in n.

ii) Durch Rückrechnung von (7.7) ergeben sich die Gleichungen

$$\begin{aligned}
r_{k-1} &= r_{k-3} - q_{k-1}r_{k-2} \\
&= r_{k-3} - q_{k-1}(r_{k-4} - q_{k-2}r_{k-3} \\
&= (1 - q_{k-2})r_{k-3} + q_{k-1}r_{k-4} \\
&= (1 - q_{k-2})(r_{k-5} - q_{k-3}r_{k-4}) + q_{k-1}r_{k-4} \\
&= (1 - q_{k-2})r_{k-5} + (q_{k-1} - (1 - q_{k-2})q_{k-3})r_{k-4} \\
&\quad \ldots \\
&= \alpha n + \beta m
\end{aligned}$$

mit gewissen ganzen Zahlen α und β.

Da nach Voraussetzung $ggT(n, m) = 1$ ist und r_{k-1} der größte gemeinsame Teiler von n und m ist, erhalten wir

$$1 = \alpha n + \beta m = \beta m \bmod n.$$

Damit ist $\beta \bmod n$ der gesuchte Wert a. Da wir zur Berechnung von β nur die Rückrechnung bei logarithmisch vielen Gleichungen vornehmen müssen (siehe Teil i), ist die Berechnung von a in logarithmischer Zeit möglich. □

Ohne Beweis geben wir die nachfolgende Abschätzung für die Anzahl der Primzahlen einer vorgegebenen Größe an.

Lemma 7.18 *Für eine natürliche Zahl n sei $p(n)$ die Anzahl der Primzahlen, die $\leq n$ sind. Dann gilt*

$$\lim_{n\to\infty} \big(p(n) - \frac{n}{\ln n}\big) = 0\,,$$

wobei ln *den natürlich Logarithmus bedeutet.* □

Lemma 7.19 *Für zwei Primzahlen p und q möge $x = a \bmod p$ und $x = a \bmod q$ gelten. Dann gilt auch $x = a \bmod pq$.*

Beweis. Die vorausgesetzten Gleichheiten lassen sich zu $x - a = tp$ und $x - a = sq$ für gewisse Zahlen t und s umformulieren. Durch Multiplikation mit q bzw. p und Subtraktion erhalten wir $(q-p)(x-a) = (t-s)pq$. Da weder p noch q ein Teiler von $q-p$ sind, muss $x-a$ sowohl p als auch q und damit pq als Teiler haben. Somit gilt $x = a \bmod pq$. □

Die *Eulersche Funktion* $\varphi : \mathbb{N} \to \mathbb{N}$ ist dadurch definiert, dass $\varphi(n)$ die Anzahl der Zahlen a mit $1 \leq a \leq n$ und $ggT(a,n) = 1$ angibt.

Es sei $n = 28$. Da $28 = 7 \cdot 2^2$ gilt, sind alle Vielfachen von 2 bzw. 7 nicht teilerfremd zu 28. Damit sind die Zahlen $1, 3, 5, 9, 11, 13, 15, 17, 19, 23, 25, 27$ die zu 28 teilerfremden Zahlen, deren Anzahl $\varphi(28)$ ist. Also ist $\varphi(28) = 12$

Es seien p und q Primzahlen. Dann gelten

$$\varphi(p) = p - 1,\ \varphi(p^2) = p^2 - p \text{ und } \varphi(pq) = (p-1)(q-1)\,. \tag{7.8}$$

Die erste Gleichheit aus (7.8) gilt offenbar, da alle Zahlen a mit $1 \leq a \leq p-1$ teilerfremd zu p sind.

Nur die Vielfachen von p sind nicht teilerfremd zu p^2. Da wir uns nur für Zahlen $\leq p^2$ interessieren, sind dies gerade die Zahlen $p, 2p, 3p, \ldots (p-1)p, p^2$. Daher entfallen p Zahlen, und es verbleiben $p^2 - p$ teilerfremde Zahlen. Damit gilt die zweite Gleichheit aus (7.8).

Die dritte Gleichheit aus (7.8) folgt daraus, dass nur die Vielfachen von p und q, also die Zahlen $p, 2p, 3p, \ldots, (q-1)p, q, 2q, \ldots, (p-1)q$ und pq nicht teilerfremd zu pq sind, woraus durch Abzählen $\varphi(pq) = pq - (q-1) - (p-1) - 1 = (p-1)(q-1)$ folgt.

Ohne Beweis geben wir die verallgemeinerten Versionen der Aussagen aus (7.8) an.

Lemma 7.20

i) *Für die Zahlen n und m gelte $ggT(n,m) = 1$. Dann ist $\varphi(nm) = \varphi(n)\varphi(m)$.*
ii) *Für eine Primzahl p und eine beliebige natürliche Zahl $n \geq 1$ gilt $\varphi(p^n) = p^n - p^{n-1}$.*

□

Damit sind wir in der Lage aus der Primzahlzerlegung einer Zahl n den Wert von $\varphi(n)$ zu berechnen. Für unser Beispiel $28 = 7 \cdot 2^2$ erhalten wir $\varphi(28) = \varphi(7) \cdot \varphi(2^2) = 6 \cdot (2^2 - 2) = 12$, was unser obiges Ergebnis bestätigt.

Es sei p eine Primzahl. Wir betrachten die Menge $M_p = \{1, 2, \ldots, p-1\}$ der von Null verschiedenen Reste bei Division durch p. Zuerst bemerken wir, dass für $a \in M_p$ und $b \in M_p$ auch $a \cdot b \bmod p$ in M_p liegt. Wäre das nämlich nicht der Fall, so wäre $ab = 0 \bmod p$. Folglich wäre $ab = tp$ für eine gewisse natürliche Zahl t. Folglich müsste p ein Teiler von a oder ein Teiler von b sein, was aber beides unmöglich ist. Also kann $ab = 0 \bmod p$ nicht eintreten. Damit ist M_p bezüglich der Multiplikation $\bmod p$ abgeschlossen. In dieser

Struktur gelten selbstverständlich das Assoziativ- und das Kommutativgesetz, da sie sich von den ganzen Zahlen auf M_p übertragen. Weiterhin ist 1 das neutrale Element bez. der Multiplikation mod p.

Es sei a eine beliebige Zahl aus M_p. Die Zahlen a, $2a \bmod p$, $3a \bmod p$, $\ldots$, $(p-1)a \bmod p$ sind alle paarweise verschieden. Wäre nämlich $xa = ya \bmod p$ für gewisse x und y mit $1 \leq y < x \leq p-1$, so würde $(x-y)a$ ein Vielfaches von p sein. Da p sowohl kein Teiler von a als auch kein Teiler von $x-y$ ist, erhalten wir einen Widerspruch. Unter den $p-1$ Zahlen a, $2a \bmod p$, $3a \bmod p$, $\ldots (p-1)a \bmod p$ muss daher auch die 1 sein. Folglich gibt es ein $b \in M_p$ so, dass $ba = 1 \bmod p$. wegen der Kommutativität gilt auch $ab = 1 \bmod p$. Somit hat jedes $a \in M_p$ ein inverses Element in M_p. Somit ist M_p eine Gruppe der Ordnung $p-1$. Aus der bekannten Tatsache, dass in einer endlichen Gruppe $(G, \circ)$ mit t Elementen $a^t = 1$ für alle Elemente a aus G gilt, folgt für unseren Spezialfall die folgende Aussage, die auch Kleiner Satz von Fermat genannt wird.

Lemma 7.21 *Für jede Primzahl p und jedes $a \in M_p$ gilt $a^{p-1} = 1 \bmod p$.* □

Die folgende Aussage, der sogenannte Eulersche Satz, ist eine Verallgemeinerung von Lemma 7.21, die daraus resultiert, dass die zu n teilerfremden Zahlen bez. der Multiplikation mod n eine Gruppe bilden.

Lemma 7.22 *Für beliebige natürliche Zahlen $n \geq 2$ und $a \geq 1$ mit $ggT(a,n) = 1$ gilt die Beziehung $a^{\varphi(n)} = 1 \bmod n$.* □

In einer beliebigen endlichen Gruppe $(G, \cdot)$ mit dem neutralen Element 1 erzeugt jedes Element $a \in G$ eine zyklische Untergruppe, die aus den Elementen $1, a, a^2, a^3, \ldots, a^{m-1}$ (man beachte $a^0 = 1$ und $a^1 = a$) besteht, wobei m die Ordnung von m ist, die durch die folgenden drei Eigenschaften festgelegt ist:

- $m > 0$,
- $a^m = 1$,
- $a^r \neq 1$ für $1 \leq r \leq m-1$.

Es ist bekannt, dass die Ordnung m von a ein Teiler der Anzahl der Elemente von G ist.

Es sei p eine Primzahl. Wir betrachten den Spezialfall der Gruppe M_p. Für jedes Element $a \in M_p$ betrachten wir die von a erzeugte Untergruppe

$$M_p(a) = \{1, a, a^2, a^3, \ldots, a^{m-1}\}.$$

Die Ordnung m von a ist ein Teiler von $p-1$, da M_p genau $p-1$ Elemente enthält.

Es sei $p = 11$. Dann gilt

$$M_p = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}.$$

Wegen der Gleichheiten

$$2^2 = 4, 2^3 = 8, 2^4 = 16 = 5, 2^5 = 10, 2^6 = 20 = 9, 2^7 = 18 = 7, 2^8 = 14 = 3,$$
$$2^9 = 6, 2^{10} = 12 = 1 \quad (7.9)$$

und

$$3^2 = 9, 3^3 = 27 = 5, 3^4 = 15 = 4, 3^5 = 12 = 1, \quad (7.10)$$

die alle modulo 11 zu verstehen sind, erhalten wir das 2 bzw. 3 in M_{11} die Ordnung 10 bzw. 5 haben und

$$M_{11}(2) = M_p \text{ und } M_{11}(3) = \{1, 3, 4, 5, 9\}$$

erfüllt sind. Man beachte, dass wir in (7.9) bei $2^5 \neq 1$ hätten aufhören können, um $M_{11}(2)$ und die Ordnung von 2 zu bestimmen, da die Ordnung ein Teiler von $p-1 = 10$ sein muss; wenn die Ordnung aber dann >5 ist, so muss sie 10 sein, woraus $M_{11}(2) = M_p$ resultiert.

Definition 7.8 *Für eine Primzahl $p > 2$ heißt $a \in M_p$ primitive Wurzel von p, falls a in M_p die Ordnung $p-1$ hat.*

Die Voraussetzung $p > 2$ wurde gemacht, da im Fall $p = 2$ die triviale Situation $M_2 = \{1\}$ eintritt.

Nach Obigem ist 2 eine primitive Wurzel von 11, während 3 keine primitive Wurzel von 11 ist.

Offenbar gilt, dass a eine primitive Wurzel von p ist, wenn $M_p = M_p(a)$ erfüllt ist. Damit gibt es genau dann eine primitive Wurzel von p, wenn M_p eine zyklische Gruppe ist.

Ohne Beweis geben wir das folgende Lemma.

Lemma 7.23 *Für jede Primzahl p ist M_p (mit der Multiplikation modulo p als Operation) eine zyklische Gruppe.* □

Damit ist für jede Primzahl $p > 2$ die Existenz einer primitiven Wurzel von p gesichert.

Lemma 7.24 *Für eine Primzahl $p > 2$ gibt es $\varphi(p-1)$ primitive Wurzeln von p.*

Beweis. Es sei a eine primitive Wurzel von p. Dann gilt $a^x = 1 \bmod p$ genau dann, wenn x ein Vielfaches von $p-1$ ist.

Es sei $b \in M_b$. Dann gibt es eine Zahl n, $0 \leq n \leq p-2$ mit $b = a^n$. Nach der Definition der primitiven Wurzel ist $b \in M_p$ keine primitive Wurzel von p, wenn $b^m = 1 \bmod p$ für eine Zahl m mit $m < p-1$ gilt. Da $b^m = (a^n)^m = a^{nm} = 1 \bmod p$ gilt, ist nm ein Vielfaches von $p-1$. Wegen $m < p-1$, gibt es einen Primfaktor von $p-1$, der auch Teiler von n ist. Folglich gilt $ggt(p-1, n) > 1$. Ist umgekehrt der größte gemeinsame Teiler t von $p-1$ und n echt größer als 1, so ergeben sich für $m = \frac{p-1}{t}$ offenbar $m < p-1$ und auch

$$b^m = (a^n)^m = a^{(p-1)\frac{n}{t}} = 1 \bmod p$$

(da $\frac{n}{t}$ ganzzahlig ist, weil t ein Teiler von n ist). Damit ist b keine primitive Wurzel.

Damit haben wir bewiesen, dass b genau dann eine primitive Wurzel von p ist, wenn $ggt(n, p-1) = 1$ gilt. Folglich gibt es $\varphi(p-1)$ primitive Wurzeln. □

Lemma 7.25 *Es sei $p > 2$ eine Primzahl.*

i) *Eine Zahl $a \in M_p$ ist genau dann eine primitive Wurzel von p, wenn $a^{\frac{p-1}{q}} \neq 1 \bmod p$ für alle Primteiler q von $p-1$ gilt.*

ii) *Es ist in* $O(\log_2(p)\log_2(p))$ *Schritten feststellbar, ob* $a \in M_p$ *eine primitive Wurzel von* p *ist oder nicht.*

Beweis.

i) Es sei b keine primitive Wurzel von p. Dann ist die Ordnung m von b von $p-1$ verschieden und ein Teiler von $p-1$. Folglich gibt es eine Primzahl q und eine Zahl $r \geq 1$ mit $m \cdot q \cdot r = p-1$. Damit gilt

$$b^{\frac{p-1}{q}} = b^{mr} = (b^m)^r = 1^r = 1 \bmod p.$$

Gilt umgekehrt $b^{\frac{p-1}{q}} = 1 \bmod p$ für einen Primteiler q von $p-1$, so ist wegen $\frac{p-1}{q} < p-1$ die Zahl b keine primitive Wurzel von p.
Somit ist $b \in M_p$ genau dann keine primitive Wurzel von p, wenn es einen Primteiler q von $p-1$ mit $b^{\frac{p-1}{q}} = 1 \bmod p$ gibt.
Durch Negation erhalten wir, dass $a \in M_p$ genau dann primitive Wurzel von p ist, wenn $a^{\frac{p-1}{q}} \neq 1 \bmod p$ für alle Primteiler q von $p-1$ gilt.

ii) Wegen i) reicht es für die Überprüfung, ob $a \in M_p$ eine primitive Wurzel von p ist, die Potenzen $a^{\frac{p-1}{q}} \bmod p$ für alle Primteiler q von $p-1$ zu bilden. Die Berechnung dieser Potenzen ist nach Lemma 7.16 in $O(\log_2(\frac{p-1}{q})) \leq O(\log_2(p))$ Schritten möglich. Die Aussage folgt nun weil es höchstens $O(\log_2(p))$ verschiedene Primteiler von $p-1$ gibt. Letzteres ist wie folgt einzusehen. Da $p-1$ gerade ist, hat $p-1$ eine Primzahlzerlegung $p-1 = 2^{n_1}p_2^{n_2}\ldots p_r^{n_r}$. Daher gilt

$$\begin{aligned}\log_2(p-1) &= n_1\log_(2) + n_2\log_2(p_2) + \cdots + n_r\log_2(p_r)\\ &\geq \log_(2) + \log_2(p_2) + \cdots + \log_2(p_r) \leq r\log_2(2) = r.\end{aligned}$$

□

Lemma 7.26 *Für eine primitive Wurzel* a *von* p *gilt* $a^{\frac{p-1}{2}} = -1 \bmod p$.

Beweis. Nach dem Kleinen Satz von Fermat (Lemma 7.21) gilt $(a^{\frac{p-1}{2}})^2 = a^{p-1} = 1 \bmod p$. Damit ist $a^{\frac{p-1}{2}}$ eine Wurzel aus 1, wofür nur die Werte 1 und -1 in Frage kommen. Falls $a^{\frac{p-1}{2}} = 1 \bmod p$ gilt, so ist a keine primitive Wurzel von p im Widerspruch zur Voraussetzung. Folglich gilt $a^{\frac{p-1}{2}} = -1 \bmod p$. □

Wir kommen nun zum Begriff des diskreten Logarithmus.

Definition 7.9 *Es seien* $p > 2$ *eine Primzahl und* a *eine primitive Wurzel von* p. *Der diskrete Logarithmus von* $b \in M_p$ *bez.* g *ist die Zahl* x *mit* $a^x = b \bmod p$. *Wir schreiben dann* $x = \log_a(b)$.

Offensichtlich wird durch diese Definition der übliche Logarithmus $\log_a(b)$ mit positiven reellen Zahlen a und b auf die Menge M_p übertragen. Die Voraussetzung, dass a eine primitive Wurzel ist, ist erforderlich, weil ansonsten der Logarithmus nicht für jedes b definiert ist. Entsprechend (7.10) ist z. B. kein Wert x vorhanden, so dass $3^x = 6$ ist, womit $\log_3(6)$ nicht definiert wäre. Für eine primitive Wurzel a ist dagegen wegen $M_p(a) = M_p$ gesichert, dass der Logarithmus bez. a von jeder Zahl aus M_p gebildet werden kann.

Wenn wir den Wert $\log_a(b)$ für eine gegebene primitive Wurzel a und einen gegebenen Wert $b \in M_p$ berechnen wollen, gibt es dafür entsprechend der Definition zwei elementare Möglichkeiten:

- Wir können der Reihe nach die Potenzen a^1, a^2, a^3, usw. bilden, bis wir x mit $a^x = b$ gefunden haben. Da nach Lemma 7.16 die Bildung der Potenzen a^y in $O(\log_2(y)) \leq O(\log_2(p))$ erfolgen kann und höchstens $p-2$ Potenzen zu bilden sind, erfordert dieses Verfahren einen zeitlichen Aufwand von $O(p \cdot \log_2(p))$ und der Speicherbedarf beträgt $O(\log_2(p))$, da dieser Platzbedarf für das Berechnen einer Potenz besteht und ansonsten nur der Vergleich mit b eine endliche Anzahl zusätzlicher Speicherzellen erfordert.

- Wir können auch vorab die Paare $(x, a^x \bmod p)$, $0 \leq x \leq p-2$, bestimmen (Zeitbedarf und Platzbedarf $O(p \cdot \log_2(p))$), und nach der zweiten Komponente sortieren (Zeitbedarf und Platzbedarf $O(p \cdot \log_2(p))$). Dann ist bei gegebenem b dieser Wert b nur in der zweiten Komponente zu suchen (bei binärer Suche sind $O(\log_2(p))$ Schritte bei einer Liste der Länge $O(p)$ erforderlich), und die erste Komponente gibt dann den gesuchten Logarithmus. Hierfür brauchen wir Vorberechnungen mit Bedarfen $O(p \cdot \log_2(p))$ und die eigentliche Berechnung erfordert den zeitlichen bzw. speichermäßigen Aufwand von $O(\log_2(p))$ bzw. $O(p)$.

Beide Möglichkeiten sind also nicht in erträglicher Zeit bzw. mit erträglichem Speicheraufwand durchführbar, wenn p eine sehr große Zahl ist.

Der folgende Algorithmus gibt eine Verbesserung auf $O(\sqrt{p} \cdot \log_2(\sqrt{p}))$.

Eingabe: primitive Wurzel g von p, $b \in M_p$
Ausgabe: diskrete Logarithmus $\log_g(b)$

1. Berechne $m = \sqrt{p-1}$.
2. Berechne $g^{m \cdot j} \bmod p$ für $0 \leq j \leq m-1$.
3. Sortiere die m geordneten Paare $(j, g^{m \cdot j} \bmod p)$ nach der zweiten Komponente. Hierbei entstehe die Liste L_1 geordneter Paare.
4. Berechne $bg^{-i} \bmod p = bg^{p-1-i} \bmod p$ für $0 \leq i \leq m-1$.
5. Sortiere die m geordneten Paare $(i, bg^i \bmod p)$ nach der zweiten Komponente. Hierbei entstehe die Liste L_2 geordneter Paare.
6. Durch Vergleiche der zweiten Komponente finde Paare $(j, y) \in L_1$ and $(i, y) \in L_2$.
7. Setze $\log_g(b) = (mj + i) \bmod (p-1)$.

Wir zeigen zuerst, dass dieser Algorithmus den diskreten Logarithmus korrekt ermittelt.

Falls $(j, y) \in L_1$ und $(i, y) \in L_2$ sind, so erhalten wir $y = g^{mj} = bg^{-i} \bmod p$ und somit $g^{mj+i} = b \bmod p$, woraus $\log_g(b) = mj + i \bmod (p-1)$ folgt.

Außerdem haben wir wegen $0 \leq j \leq m-1$ und $0 \leq i \leq m-1$ für alle Zahlen x mit $0 \leq x \leq m^2 - 1$ eine Darstellung $x = mj + i$. Somit gibt es $m^2 - 1$ verschiedene Zahlen

g^{mj+i}, womit für jedes b eine Darstellung $g^{mj+i} = b$ existiert. Somit wird für jedes b auch ein diskreter Logarithmus gefunden.

Hinsichtlich der Komplexität ergeben sich bei den einzelnen Schritten folgende Bedarfe:

Schritt	Zeitbedarf	Platzbedarf
1	$O(m)$	$O(1)$
2 bzw. 4	$O(m \cdot \log_2(m))$	$O(m)$
3 bzw. 5	$O(m \cdot \log_2(m))$	$O(m)$
6	$O(m)$	$O(m)$
7	$O(1)$	$O(1)$

Insgesamt erhalten wir daher den Zeitaufwand $O(m \cdot \log_2(m)) = O(\sqrt{p}\log_2(\sqrt{x}))$ und den Speicherplatz $O(m) = O(\sqrt{p})$.

Es gibt noch weitere Verbesserungen, aber hinsichtlich der Größenordnung bleibt es im Wesentlichen bei der Komplexität des obigen Algorithmus. Es ist daher also schwer, den diskreten Logarithmus zu berechnen.

7.4.2 Das Schlüsselsystem von Rivest, Shamir und Adleman

Wir beschreiben nun das RSA-System, das nach ihren Begründern R.L. Rivest, A. Shamir und L. Adleman benannt wurde.

1. Wir wählen zuerst zwei sehr große Primzahlen p und q und berechnen ihr Produkt $n = pq$.
   Dann ergibt sich $\varphi(n) = (p-1)(q-1)$ (siehe Lemma 7.20).

2. Als nächstes wählen wir eine große zufällige Zahl d mit $1 \leq d \leq n$ und $ggT(d, n) = 1$. Danach bestimmen wir eine Zahl e mit $1 \leq e \leq n$ und $ed = 1 \bmod \varphi(n)$.

3. Die Zahlen n und e werden öffentlich gemacht, während p, q, $\varphi(n)$ und d geheim gehalten werden.

4. Jede Binärfolge kann als Dualdarstellung einer Zahl aufgefasst werden, und diese Zahl kann einfach ermittelt werden. Auch jede Buchstabenfolge kann als Zahl in einem 26-ären Zahlensystem interpretiert werden. Daher reicht es, eine Verschlüsselung für Zahlen anzugeben, um Binärfolgen oder Texte zu verschlüsseln.

   Die Verschlüsselung einer Zahl w erfolgt nun beim RSA-System durch $w^e \bmod n$. Da w gegeben ist, n und e öffentlich sind, kann jeder diese Verschlüsselung vornehmen.

Wir wollen zuerst etwas zur Entschlüsselung sagen. Wir nehmen die Entschlüsselung der Kryptozahl $c = w^e \bmod n$ einfach dadurch vor, dass wir den Wert $c^d \bmod n$ berechnet. Um zu zeigen, dass hierdurch eine Entschlüsselung erfolgt, müssen wir zeigen, dass $c^d = w \bmod n$ ist.

Dazu bemerken wir erst einmal, dass nach Voraussetzung $ed = t\varphi(n) + 1$ für eine gewisse Zahl t gilt.

Wir nehmen nun als Erstes an, dass weder p noch q ein Teiler von w ist. Dann gilt $ggT(w, n) = 1$. Damit ist nach dem Eulerschen Satz (Lemma 7.22) $w^{\varphi(n)} = 1 \bmod n$. Damit erhalten wir

$$c^d = (w^e)^d = w^{ed} = w^{t\varphi(n)+1} = (w^{\varphi(n)})^t \cdot w = 1 \cdot w = w \bmod n\,.$$

Sei nun p ein Teiler von w und q kein Teiler von w. Dann gilt sicher $w = 0 \bmod p$ und damit auch $w^{ed} = 0 \bmod p$ und damit $w^{ed} = w \bmod p$. Wegen Lemma 7.21 erhalten wir noch $w^{q-1} = 1 \bmod q$. Hieraus ergeben sich

$$w^{\varphi(n)} = w^{(p-1)(q-1)} = (w^{q-1})^{p-1} = 1^{p-1} = 1 \bmod q \text{ und } w^{ed} = w^{t\varphi(n)+1} = w \bmod q\,.$$

Nach Lemma 7.19 liefert dies $c^d = w^{ed} = w \bmod pq = w \bmod n$.

Sind sowohl p als auch q Teiler von w, so erhalten wir $w^{ed} = w = 0 \bmod p$ und $w^{ed} = w = 0 \bmod q$, woraus nach Lemma 7.19 folgt, dass $c^d = w^{ed} = w \bmod n$ gilt.

Wir kommentieren nun die Wahlen, die für ein RSA-System erforderlich sind, und zeigen, dass diese „einfach“ realisierbar sind.

Wir beginnen mit der Verschlüsselung. Hier ist die Potenz $w^e \bmod n$ zu bilden. Nach Lemma 7.16 ist dies mit logarithmischem Aufwand in e zu leisten. Selbst bei einer Zahl e mit 100 Stellen (in der Dezimaldarstellung erfordert dies nur ca. 600 Operationen.

Da die Entschlüsselung auch das Potenzieren einer Zahl – nämlich der Kryptozahl c – erfordert, ist auch dieses „einfach“ zu bewerkstelligen.

Wir kommen nun zur Wahl der beiden Primzahlen p und q. Diese müssen beide sehr groß sein. Sind beide Primzahlen klein, so ist auch n klein, und durch Durchtesten aller Zahlen bis $\sqrt{n}$ kann mindestens eine der Primzahlen gefunden werden, woraus sich dann die andere einfach berechnen lässt. Ist nur eine der beiden Primzahlen klein, so kann man durch Durchtesten aller Primzahlen entsprechend ihrer Größe auf der Basis einer Primzahltabelle die kleine Primzahl als Faktor von n finden und dann erneut die andere ausrechnen. Wir gehen davon aus, dass die Primzahlen in ihrer Dezimaldarstellung mindestens 100 Ziffern haben sollen, womit $n \geq 10^{200}$ ist. Wie findet man solche großen Primzahlen? Hierzu stellen wir zuerst fest, dass es wegen Lemma 7.18 annähernd

$$A = \frac{10^{100}}{\ln(10^{100})} - \frac{10^{99}}{\ln(10^{99})}$$

Primzahlen mit 100 Ziffern gibt (wir führen die Überlegungen hier für 100 durch, sie lassen sich aber leicht auf andere Werte übertragen). Mit 100 Ziffern gibt es

$$B = (10^{100} - 10^{99})/2$$

ungerade Zahlen. Wir wählen nun zufällig eine Zahl mit 100 Dezimalziffern (indem wir 100-mal zufällig eine Ziffer wählen). x sei die kleinste ungerade Zahl, die größer oder gleich dieser Zufallszahl ist. Nun überprüfen wir, ob x eine Primzahl ist. Falls diese Überprüfung negativ ausfällt, setzen wir mit $x + 2$ fort. Fällt der Test auf Primzahl erneut negativ aus, setzen wir mit $x + 4$ fort, usw. Die Wahrscheinlichkeit eine Primzahl zu finden beträgt offensichtlich $A/B = 0,00868...$, so dass nach 1000 Versuchen eine Primzahl zu erwarten ist.

Es bleibt zu klären, wie kompliziert es ist, den Primzahltest durchzuführen. Agrawal hat im Jahre 2001 gezeigt, dass es einen Primzahltest gibt, der für x in in x polynomialer Zeit entscheidet, ob x eine Primzahl ist. Der Grad des Polynoms ist aber zu hoch, um für unsere Belange genutzt zu werden. Für praktische Belange reicht es aber einen Primzahltest zu haben, der mit hoher Wahrscheinlichkeit die richtige Antwort gibt.

Wir erläutern jetzt einen solchen Primzahltest. Wenn k eine Primzahl ist, so gilt nach Lemma 7.21 für jede Zahl a mit $1 \leq a \leq k-1$ die Beziehung $a^{k-1} = 1 \bmod k$. Daher wissen wir, dass k keine Primzahl ist, wenn $a^{k-1} \neq 1 \bmod k$ gilt. Wir nennen daher eine Zahl u einen *Zeugen* dafür, dass eine Zahl m eine Primzahl ist, falls u die Bedingungen $ggT(u,m) = 1$ und $u^{m-1} = 1 \bmod m$ erfüllt.

Lemma 7.27 *Für eine gegebene Zahl $m \geq 2$ sind entweder alle Zahlen oder höchstens die Hälfte aller Zahlen u mit $1 \leq u \leq m-1$ und $ggt(u,m) = 1$ ein Zeuge dafür, dass m eine Primzahl ist.*

Beweis. Es seien nicht alle Zahlen u mit $1 \leq u \leq m-1$ und $ggt(u,m) = 1$ ein Zeuge dafür, dass m eine Primzahl ist. Dann gibt es eine Zahl z mit $1 \leq z \leq m-1$, $ggt(z,m) = 1$ und $z^{m-1} \neq 1 \bmod m$. Seien nun u_i, $1 \leq i \leq r$, die Zeugen dafür, dass m eine Primzahl ist. Wir bilden die Zahlen $z_i = u_i z \bmod m$, $1 \leq i \leq r$. Für alle diese Zahlen z_i gilt

$$z_i^{m-1} = (u_i z)^{m-1} = u_i^{m-1} z^{m-1} = 1 \cdot z^{m-1} = z^{m-1} \neq 1 \bmod m\,.$$

Daher sind alle Zahlen z_i auch keine Zeugen dafür, dass m Primzahl ist. Folglich ist die Zahl der Zeugen höchstens so groß wie die Zahl der Nichtzeugen, woraus die Behauptung sofort folgt. □

Aus diesem Lemma ergibt sich folgender Algorithmus zum Testen, ob m eine Primzahl ist. Wir wählen eine beliebige Zahl u mit $2 \leq u \leq m-1$. Zuerst testen wir, ob $ggT(u,m) = 1$ ist. Fällt der Test negativ aus, so ist m keine Primzahl. Dann bestimmen wir $u^{m-1} \bmod m$. Ist dieser Wert von 1 verschieden, so ist m keine Primzahl. Im anderen Fall ist u ein Zeuge dafür, dass m ein Primzahl ist, und wegen Lemma 7.27 ist m mit Wahrscheinlichkeit $1/2$ keine Primzahl. Nach k Wahlen von Zahlen u erhalten wir, für die sich stets u als Zeuge erweist, ist m nur noch mit der Wahrscheinlichkeit $1/2^k$ keine Primzahl.

Bei der Wahl von p und q ist zu beachten, dass die Differenz $p-q$ (bei $p > q$) nicht sehr klein ausfallen darf. Wegen

$$n = pq = \Big(\frac{(p+q)}{2}\Big)^2 - \Big(\frac{(p+q)}{2}\Big)^2$$

wäre dann $a = \left(\frac{(p+q)}{2}\right)^2$ eine Quadratzahl, die nicht viel größer als n ist. Da man aus $n = pq$ und $a = \left(\frac{(p+q)}{2}\right)^2$ für p^2 die quadratische Gleichung $(p^2)^2 + (2n - 4a)p^2 + n^2 = 0$ ermittelt, lässt sich p aus n und a berechnen. Durch Durchtesten aller Quadratzahlen $> n$ könnte man dann p bestimmen.

Wir kommen nun zur Wahl von d. Wir wählen zufällig eine große Zahl d als Kandidaten (indem wir wieder die Dezimalziffern der Reihe nach zufällig wählen. Dann testen wir, ob $ggT(d,n) = 1$ gilt. Falls der Test negativ ausfällt, addieren wir 1, testen erneut usw., bis die Zahl teilerfremd zu n ist.

Die Berechnung von e ist dann nach Lemma 7.17 i) einfach zu bewerkstelligen.

Da die zu $\varphi(n)$ teilerfremden Zahlen hinsichtlich der Multiplikation $\bmod \varphi(n)$ eine Gruppe bilden, gibt es eine Zahl t mit $e^t = 1 \bmod \varphi(n)$. Offensichtlich gilt dann

$$d = e^{t-1} \bmod \varphi(n).$$

Bei der Wahl von d (und dadurch e) ist darauf zu achten, dass t mit $e^t = 1 \bmod \varphi(n)$ möglichst groß ausfällt. Dies ist erforderlich, da sonst der Kryptoanalyst wie folgt vorgehen kann. Er ermittelt zu der Kryptozahl $c = w^e \bmod n$ eine Zahl v mit $c^{e^v} = c$. Da dann auch $c^{e^v-1} = 1 \bmod n$ gilt, ist wegen der Teilerfremdheit von e und $\varphi(n)$ und dem Satz von Euler (Lemma 7.22) $e^v - 1 = 0 \bmod \varphi(n)$. Somit ist $e^v = 1 \bmod \varphi(n)$, woraus dann $d = e^{v-1} \bmod \varphi(n)$ folgt, d. h. der Kryptoanalyst ist in der Lage d zu berechnen, womit er die Entschlüsselung vornehmen kann.

Abschließend bemerken wir noch, dass die geheim zu haltenden Größen nicht unabhängig voneinander sind. So reicht es, sich p zu merken. Daraus resultiert q als n/p und dann $\varphi(n) = (p-1)(q-1)$. Da e öffentlich bekannt ist, kann nun auch d bestimmt werden.

Es ist aber auch ausreichend, sich $\varphi(n)$ zu merken, denn aus den Beziehungen

$$n = p \cdot q \text{ und } \varphi(n) = (p-1)(q-1)$$

ergibt sich

$$p^2 + p(\varphi(n) - n - 1) + n = 0\,,$$

woraus p ermittelt werden kann. Dann lassen wie zuvor die anderen Größen bestimmen.

7.4.3 Ein Verfahren mit Hilfe des diskreten Logarithmus

Wir haben oben gesehen, dass die Berechnung des diskreten Logarithmus schwer ist. Diesen Umstand nutzen wir im folgenden öffentlichen Schlüsselsystem.

Wir wählen die Parameter des Verfahrens wie folgt.

1. Wähle eine große Primzahl p und eine primitive Wurzel g von p.
2. Wähle zufällig eine Zahl $x \in \{1, 2, \ldots, p-2\}$ und berechne $y = g^x \bmod p$.
3. Gebe öffentlich die Parameter p, g und y bekannt, während x geheim bleibt.

Der Kryptoanalyst hat um Kenntnis von x zu erhalten, den diskreten Logarithmus $x = \log_g(b)$ zu berechnen, was nach unserer Annahme schwer ist.

Die Verschlüsselung wird wie folgt vorgenommen. Wir stellen die Nachricht M wieder als Zahl mit $0 \le M \le p-1$ dar. Dann wählen wir zufällig eine Zahl $k \in \{1, 2, \ldots, p-2\}$ und berechnen die Zahlen

$$a = g^k \bmod p \quad \text{und} \quad b = M \cdot y^k \bmod p$$

unter Verwendung der öffentlich bekannten Parameter p, g, und y. Als Ergebnis der Verschlüsselung liegt dann das Paar $C(M) = (a, b)$ vor.

Bei der Decodierung ermitteln wir $D(a, b) = a^{p-1-x} b \bmod p$.

Dieses Verfahren ist korrekt, weil

$$\begin{aligned} D(a,b) &= a^{p-1-x} b = (g^k)^{p-1-x} M y^k = (g^k)^{p-1-x} M (g^x)^k \\ &= M g^{k(p-1)-kx+kx} = M(g^{p-1})^k = M \bmod p \end{aligned}$$

gilt.

7.4.4 Signaturen und Hashfunktionen

Bisher haben wir die Schlüsselsysteme dazu benutzt, dass ein Sender eine Nachricht M in codierter/verschlüsselter Form abschickt, die nur von befugten Empfängern decodiert/entschlüsselt werden kann. Bei öffentlichen Schlüsselsystemen kann jeder die Verschlüsselung vornehmen und damit kann jede Person die geheime Nachricht an den/die befugten Empfänger senden. Jedoch sind öffentliche Schlüsselsysteme auch dazu geeignet, sicher zu stellen, dass die Nachricht vom angegebenen Sender stammt.

Dazu sei ein öffentliches Schlüsselsystem mit einer öffentlichen Verschlüsselung C und einer geheimer Entschlüsselung E gegeben, für die $E(C(M)) = C(E(M)) = M$ gelte. Dies ist beispielsweise bei der RSA-Verschlüsselung deer Fall. Dann sendet die befugte Person, die im Besitz von E ist eine Nachricht M und zusätzlich $E(M)$, die sogenannte Signatur zu M. Der Empfänger kann nun $C(E(M))$ bilden. Stimmt dieser Text mit M überein, so weiß er, dass der Sender über E verfügt. Daher kann er sicher sein, dass die Nachricht M vom richtigen Sender stammt.

Der Nachteil dieser Methode besteht darin, dass bei einem langen Text M auch der (unter Umständen noch längere) Text $E(M)$ mit versendet werden muss. Zur Beseitigung dieses Umstandes werden Hashfunktionen benutzt.

Definition 7.10 *Eine Funktion $h : X \to Y$ heißt Hashfunktion, wenn $\#(X)$ viel größer als $\#(Z)$ ist.*

In der Regel wird $X = \{0,1\}^+$ oder $X = \{0,1\}^m$ und $Z = \{0,1\}^n$ verwendet, wobei m viel größer als n ist. Im Folgenden setzen wir oft voraus, dass $\#(X) \geq 2 \cdot \#(Z)$ gilt. Diese Voraussetzung sichert bei $X = \{0,1\}^m$ und $Z = \{0,1\}^n$, dass mindestens $m \geq n+1$ gilt. Entsprechend Definition 7.10 ist m viel größer als n, so dass die gemachte Voraussetzung eigentlich stets erfüllt ist.

Es sei h eine Hashfunktion. Um nun eine Nachricht M zu signieren, wird $E(h(M))$. Der Empfänger erhält also M und die Signatur $E(h(M))$. Er bildet aus M den Text $h(M)$ und aus $E(h(M))$ den Text $C(E(h(M)))$. Stimmen die beiden so gewonnenen Texte überein, so ist der Sender wieder als befugt anzusehen.

Hierbei können jetzt folgende Probleme auftreten.

Wenn eine unbefugte Person zu M einen Text M' ermitteln kann, der $M' \neq M$ und $h(M) = h(M')$ erfüllt, so kann er das Paar $(M', E(h(M)))$ senden, wobei er den Text in der zweiten Komponente von der Versendung von $(M, E(h(M)))$ durch einen befugten Sender übernimmt. Der Empfänger ermittelt $h(M') = h(M)$ und $C(E(h(M))) = h(M)$ und glaubt daher, dass M' von einem befugten Sender stammt.

Diese Situation ist sicher gegeben, wenn der Sender über eine Methode verfügt, aus einem Wert z einen Text M mit $h(M) = z$ zu ermitteln.

Hat der unbefugte Sender ein zwei Texte M und M' mit $M \neq M'$ und $h(M) = h(M')$ ermittelt, so kann er einen befugten Empfänger überreden $E(h(M))$ als Signatur zu verwenden und dann M' versenden, die durch den Empfänger wieder als Nachricht von einem befugten Empfänger interpretiert wird.

Definition 7.11 *Wir sagen, dass M und M' bez. einer Hashfunktion h eine Kollision bilden, wenn $M \neq M'$ und $h(M) = h(M')$ gelten.*

Um die oben angegebenen Situationen zu vermeiden, benötigt man Hashfunktionen, bei denen Kollisionen praktisch unmöglich sind.

Definition 7.12

i) *Eine Hashfunktion h heißt schwach kollisionfrei für M, wenn es praktisch unmöglich ist, einen Text M' mit $M' \neq M$ und $h(M) = h(M')$ durch einen Algorithmus finden.*

ii) *Eine Hashfunktion h heißt stark kollisionfrei, wenn es praktisch unmöglich ist, eine Kollision bez. h durch einen Algorithmus finden.*

iii) *Eine Hashfunktion heißt Ein-Weg-Funktion, wenn es praktisch unmöglich ist, zu einem Wert z einen Text M mit $h(M) = z$ zu durch einen Algorithmus finden.*

Die starke Kollisionsfreiheit ist gegeben, wenn es keinen Algorithmus gibt, mit dem ein M gefunden wird, für das h nicht schwach kollisionsfrei ist. In diesem Sinn folgt die schwache Kollisionsfreiheit aus der starken Kollisionsfreiheit.

Wir zeigen nun, dass bei Ein-Weg-Funktionen mit großer Wahrscheinlichkeit keine Kollisionen gefunden werden können.

Satz 7.28 *Es sei $h : X \to Z$ eine Hashfunktion, bei der $\#(X) \geq 2 \cdot \#(Z)$ erfüllt ist. Ferner gebe es einen Algorithmus, der zu gegebenem z ein M mit $h(M) = z$ findet. Dann gibt es einen probabilistischen Algorithmus, der mit einer Wahrscheinlichkeit $\geq \frac{1}{2}$ eine Kollision findet.*

Beweis. WEs sei A der Algorithmus, der zu gegebenem z ein M mit $h(M) = z$ findet. Dann betrachten wir den folgenden Algorithmus B:

1. Wähle zufällig ein $M \in X$.
2. Berechne $z = h(M)$.
3. Mittels A berechne ein $M' \in X$ mit $h(M') = z$.
4. Falls $M' \neq M$, so ist eine Kollision gefunden; B stoppt erfolgreich.
   Anderenfalls liegt ein Misserfolg vor.

Wie berechnen nun die Wahrscheinlichkeit, dass mit B eine Kollision gefunden wird. Dazu definieren die Relation $\sim$ auf X dadurch, dass $M \sim M'$ genau dann gilt, wenn $h(M) = h(M')$. Es ist leicht zu sehen, dass $\sim$ eine Äquivalenzrelation ist. Daher wird X durch $\sim$ in disjunkte Äquivalenzklassen zerlegt. Es sei C die Menge der Äquivalenzklassen von $\sim$. Mit $[M]$ bezeichnen wir die Äquivalenzklasse von M. Da $[M] = \{M' \mid h(M) = h(M')\}$ gilt, gibt es zu jedem $z \in Z$ genau eine Äquivalenzklasse $c \in C$, die aus allen M mit $h(M) = z$ besteht. Daher gilt $\#(C) \leq \#(Z)$.

Für das in Schritt 1 des Algorithmus B gewählte M gibt es $\#([M])$ mögliche M', die in Schritt 3 ermittelt werden. Davon führen $\#([M]) - 1$ Elemente M' in Schritt 4 zu einer

Kollision. Folglich ergibt sich für die Wahrscheinlichkeit p des Erfolges

$$\begin{aligned}
p &= \frac{1}{\#(X)} \sum_{M \in X} \frac{\#([M]) - 1}{\#([M])} \\
&= \frac{1}{\#(X)} \sum_{c \in C} \sum_{M \in c} \frac{\#(c) - 1)}{\#(c)} \qquad \text{(da jedes } M \text{ in einem } c \in C \text{ liegt)} \\
&= \frac{1}{\#(X)} \sum_{c \in C} (\#(c) - 1) \qquad \text{(da } \#([M]) = \#(c) \text{ für } M \in c \text{ gilt)} \\
&= \frac{1}{\#(X)} (\sum_{c \in C} \#(c) - \sum_{c \in C} 1) \\
&= \frac{1}{\#(X)} (\#(X) - \#(C)) \qquad \text{(da } X \text{ die disjunkte Vereinigung der } c \in C \text{ ist)} \\
&= \frac{\#(X) - \#(C)}{\#(X)} \geq \frac{\#(X) - \#(Z)}{\#(X)} \geq \frac{\#(X) - \frac{\#(X)}{2}}{\#(X)} \\
&= \frac{1}{2}.
\end{aligned}$$

□

Satz 7.29 *Es sei $h : X \to Z$ eine Hashfunktion, bei der $\#(X) \geq 2 \cdot \#(Z)$ erfüllt ist und bei der die Werte von h gleichverteilt in Z sind. Dann lässt sich durch k zufällige Wahlen von Elementen aus X eine Kollision mit der Wahrscheinlichkeit finden, die annähernd*

$$1 - \frac{1}{e^{\frac{k(k-1)}{2\#(Z)}}}$$

ist.

Beweis. Es seien $m = \#(X)$ und $n = \#(Z)$. Aufgrund der Gleichverteilung der Werte von h in Z und der zufälligen Wahl der der k Elemente $M_1, M_2, \ldots, M_k$, sind die Werte $z_i = h)M_i)$ auch zufällige Größen aus Z. Die Wahrscheinlichkeit für $z_1 \neq z_2$ beträgt $\frac{n-1}{n}$, also $1 - \frac{1}{n}$. Wenn $z_1 \neq z_2$ gilt, so ist die Wahrscheinlichkeit dafür, dass $z_3 \neq z_1$ und $z_3 \neq z_2$ gelten, $\frac{n-2}{n}$, also $1 - \frac{2}{n}$. Damit ist die Wahrscheinlichkeit $(1 - \frac{1}{n})(1 - \frac{2}{n})$ für eine Wahl mit paarweise verschiedenen Elemente z_1, z_2, z_3. Wir fahren so fort und erhalten nach k Wahlen als Wahrscheinlichkeit dafür, dass $z_1, z_2, \ldots, z_k$ paarweise verschieden sind, d. h. dafür, dass keine Kollision vorliegt, den Wert

$$q = (1 - \frac{1}{n})(1 - \frac{2}{n}) \ldots (1 - \frac{k-1}{n}) = \prod_{i=1}^{k} -1(1 - \frac{i}{n}).$$

Für große n sind die Zahlen $\frac{i}{n}$ klein und können daher wegen

$$1 - x \approx 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} \cdots = e^{-x}$$

(letztere Gleichheit ist die Reihenentwicklung der Exponentialfunktion) durch $e^{-\frac{i}{n}}$ abgeschätzt werden. Somit ergibt sich

$$q = \prod_{i=1}^{k} -1(1 - \frac{i}{n}) \approx \prod_{i=1}^{k-1} e^{-\frac{i}{n}} = e^{-\frac{k(k-1)}{2n}}.$$

Die Wahrscheinlichkeit für das Auftreten einer Kollision ist somit

$$p = 1 - q \approx 1 - e^{-\frac{k(k-1)}{2n}} = 1 - \frac{1}{e^{\frac{k(k-1)}{2\#(Z)}}}. \tag{7.11}$$

□

Wenn wir uns eine Wahrscheinlichkeit p vorgeben, so erhalten wir aus (7.11) durch Logarithmieren und Multiplikation mit $2n$ die Beziehung $-k(k-1) \approx 2n \ln(1-p)$. Dies ist eine quadratische Gleichung für k, die als Lösung

$$k \approx \frac{1}{2} + \sqrt{\frac{1}{4} + 2n \ln(\frac{1}{1-p})} \approx \sqrt{(2 \ln(\frac{1}{1-p}) \cdot \sqrt{n} = c_p \cdot \sqrt{n}}$$

liefert. Damit kann zu einer Wahrscheinlichkeit p ausgerechnet werden, wie viele zufällige Wahlen vorgenommen werden müssen, um eine Kollision zu finden. Dies bedeutet, dass wir n so groß wählen müssen, dass $\sqrt{n}$ Wahlen praktisch nicht durchgeführt werden können.[2]

Abschließend zeigen wir nun, dass es stark kollisionsfreie Hashfunktionen gibt. Wir wählen eine große sichere Primzahl p, d. h. $p = 2q+1$ für eine Primzahl q und zwei primitive Wurzel g_1 und g_2 von p. Dann ist es praktisch unmöglich, den diskreten Logarithmus $\log_{g_1}(g_2)$ zu berechnen. Die Hashfunktion h definieren wir nun durch

$$h : \{0, 1, \ldots, q-1\} \times \{0, 1, \ldots, q-1\} \to M_p \text{ vermöge } h(x_1, x_2) = g_1^{x_1} g_2^{x_2}. \tag{7.12}$$

Für h gilt, dass der Definitionsbereich und der Wertevorrat $(q-1)^2$ bzw. $p - 1 = 2q$ Elemente enthalten. Für praktische Bedürfnisse ist damit der Definitionsbereich nicht viel größer als der Wertevorrat, und daher diese Hashfunktion nur bedingt geeignet. Jedoch können wir nachweisen, dass h stark kollisionsfrei ist. Dazu reicht es zu zeigen, dass die Berechenbarkeit einer Kollision die Berechenbarkeit von $\log_{g_1}(g_2)$ nach sich zieht.

Satz 7.30 *Ist eine Kollision für h aus (7.12) bekannt, so kann $\log_{g_1}(g_2)$ berechnet werden.*

Beweis. Es sei $(x_1, x_2) \neq (x_3, x_4)$ mit $h(x_1, x_2) = h(x_3, x_4)$ gegeben. Wir nehmen dabei ohne Beschränkung der Allgemeinheit an, dass $x_4 \geq x_2$ gilt. Wegen $0 \leq x_2 \leq q-1$ und $0 \leq x_4 \leq q-1$ ergibt sich auch $0 \leq x_4 - x_2 \leq q-1$. Aus der Gleichheit $h(x_1, x_2) = h(x_3, x_4)$ erhalten wir

$$g_1^{x_1} g_2^{x_2} = g_1^{x_3} g_2^{x_4} \bmod p, \tag{7.13}$$

[2]Für den Wert $p = \frac{1}{2}$ ergibt sich $c_p \approx 1,17$. Betrachtet man nun den Fall, dass wir für X die Menge aller Menschen und für Z die Tage des Jahres und für h die Funktion, die jedem Menschen seinen Geburtstag zuordnet, so ergibt sich, dass bei zufälliger Wahl von 23 Menschen die Wahrscheinlichkeit von zwei gewählten Personen mit gleichem Geburtstag schon größer als $\frac{1}{2}$ ist, da $1,17 \cdot \sqrt{365} = 22,3$ gilt.

woraus

$$g_1^{x_1-x_3} = g_2^{x_4-x_2} \bmod p \tag{7.14}$$

folgt.

Wir betrachten nun

$$d = ggt(x_4 - x_2, p - 1).$$

Wegen $p-1 = 2q$ ergeben sich nur die Möglichkeiten $d \in \{1, 2, q, p-1\}$. Da $x_4 - x_2 \leq q-1$ ist, entfällt die Möglichkeit $d = q$. Wir diskutieren jetzt die verbleibenden Fälle.

Fall 1. $d = p-1$. Wegen $x_4 - x_2 \leq q-1$ ist $d = p-1$ nur für $x_4 - x_2 = 0$ möglich. Damit erhalten wir zuerst $x_4 = x_2$, aus (7.13) dann $g_1^{x_1} g_2^{x_2} = g_1^{x_3} g_2^{x_2} \bmod p$, daraus schließlich $x_1 = x_3$ und damit letztlich $(x_1, x_2) = (x_3, x_4)$ im Widerspruch zu Voraussetzung, dass (x_1, x_2) und (x_3, x_4) eine Kollision bilden.

Fall 2. $d = 1$. Dann gibt es eine Zahl $y = (x_4 - x_2)^{-1} \bmod (p-1)$, und y ist nach Lemma 7.17 einfach zu berechnen. Wir erhalten

$$g_2 = g_2^{(x_4-x_2)y} = (g_2^{x_4-x_2})^y = (g_1^{x_1-x_3})^y = g_1^{(x_1-x_3)y} \bmod p, \tag{7.15}$$

woraus sich nach definition sofort $\log_{g_1}(g_2) = (x_1 - x_3)y \bmod (p-1)$ ergibt. Damit ist der diskrete Logarithmus $\log_{g_1}(g_2)$ aus den gegebenen Zahlen x_1 und x_3 und dem einfach berechenbaren y einfach zu ermitteln.

Fall 3. $d = 2$. Wegen $p - 1 = 2q$ ergibt sich $ggt(x_4 - x_2, q) = 1$. Damit existiert $z = (x_4 - x_2)^{-1} \bmod q$ und ist nach Lemma 7.17 einfach zu berechnen. Nach Definition erhalten wir $(x_4 - x_2)z = kq + 1$ für eine Zahl k. Nach Lemma 7.26 ergibt sich

$$g_2^q = g_2^{\frac{p-1}{2}} = -1 \bmod p$$

und somit

$$g_2^{x_4-x_2)y} = g_2^{kq+1} = (g_2^q)^k g_2 = (-1)^k g_2 \bmod p. \tag{7.16}$$

Falls k eine gerade Zahl ist (und damit $(-1)^k = 1$), erhalten wir erneut (7.15) und damit $\log_{g_1}(g_2) = (x_1 - x_3)y \bmod (p-1)$ als einfach zu ermittelnde Zahl.

Falls k ungerade und damit $(-1)^k = -1$ ist, so haben wir noch $g_1^q = -1 \bmod p$ nach Lemma 7.26. Unter Verwendung von (7.14) und (7.16) ergibt sich nun

$$g_1^{(x_1-x_3)y+q} = g_1^q (g_1^{x_1-x_3})^y = (-1)(g_2^{x_4-x_2})^k = (-1) g_2^{(x_4-x_2)y} = (-1)(-1) g_2 = g_2 \bmod p$$

und damit $\log_{g_1}(g_2) = (x_1 - x_3)y + q$. Somit ist der diskrete Logarithmus $\log_{g_1}(g_2)$ auch in diesem Fall einfach zu berechnen. □

Literaturverzeichnis

[1] J. Berstel / D. Perrin, *Theorie of Codes*. Academic Press, 1985.

[2] A. Beutelspacher, Kryptologie. Vieweg, 1991.

[3] J. Dassow, A note on DT0L Systems. *Bull. EATCS* **22** (1984) 11–14.

[4] J. Duske / H. Jürgensen, *Kodierungstheorie*. BI-Taschenbuch 25, Mannheim, 1977.

[5] M.R. Garey / D.S. Johnson, *Computers and Intractability / A Guide to NP-Completeness.* Freeman & Company, 1978.

[6] T. Grams, *Codierungsverfahren*. BI-Hochschultaschenbuch 625, Mannheim, 1986.

[7] J.E. Hopcroft / J.D. Ullman, Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie. Addison-Wesley, 1990.

[8] J. Kari, Observations concerning a public-key cryptosystem based on iterated morphisms. *Theor. Comp. Sci.* **66**(1989) 45–53.

[9] W.I. Löwenstein, Kodierungstheorie. In: *Diskrete Mathematik und mathematische Fragen der Kybernetik*, Herausg.: S.W.Jablonski / O.B.Lupanov, Akademie-Verlag, 1980.

[10] B. Martin, *Codage, cryptologie et applications*. Presses Polytechniques et Universitaires Romandes, 2004.

[11] R. Merkle / M. Hellman, Hiding informations and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory* **IT-24** (1978) 525–530.

[12] W.W. Peterson / E.J. Weldon, *Error-Correcting Codes.* MIT Press, Cambridge, 1972.

[13] R.L. Rivest / A. Shamir / L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM* **21** (1978) 120–126.

[14] G. Rozenberg / A. Salomaa, *Mathematical Theory of L Systems*. Academic Press, 1980.

[15] A. Salomaa, *Jewels of Formal Language Theory*. Computer Science Press, 1981.

[16] A. Salomaa, *Public-Key Cryptography*. Springer-Verlag, 1996.

[17] A. Salomaa / E. Welzl, On a public-key cryptosystems based on iterated morphisms and substitutions. Manuskript, 1983.

[18] H.J. Shyr, *Free Monoids and Languages*. Hon Min Book Co., Taichung, Taiwan, 1991.

[19] P. Sweeney, *Codierung zur Fehlererkennung und Fehlerkorrektur*. Hanser-Verlag, 1992.

[20] D. Wätjen, Kryptographie. Grundlagen, Algorithmen, Protokolle. Spektrum-Verlag, 2003.

[21] W. Willems, *Codierungstheorie*. Walter de Gruyter, Berlin, 1999.