Prof. Dr. Jürgen Dassow
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik

Codierungstheorie
und
Kryptographie

Sommersemester 2010

Inhaltsverzeichnis

1 Definition und Charakterisierung von Codes **5**
1.1 Definition von Codes 5
1.2 Codierung und Decodierung durch Automaten 10
1.3 Entscheidbarkeit der Eigenschaft, Code zu sein 13
1.4 Codeindikator und Konstruktion von Codes 23

2 Optimale Codes **27**

3 Fehlerkorrigierende Codes **37**
3.1 Fehlertypen und Fehlerkorrektur 37
3.2 Beispiele für fehlerkorrigierende Codes 43
3.3 Abschätzungen für fehlerkorrigierende Codes 47

4 Lineare Codes **55**

5 Klassische Verschlüsselungen **65**
5.1 Monoalphabetische Substitutionschiffren 66
5.2 Polyalphabetische Substitutionschiffren 69
5.3 Der Data Encryption Standard 77
5.4 Steganographie 83

6 Perfekte Sicherheit **85**

7 Öffentliche Schlüsselsysteme **91**
7.1 Die Idee öffentlicher Schlüssel 91
7.2 Ein System auf der Basis des Rucksack-Problems 93
7.3 Systeme auf der Basis von formalen Sprachen 103
7.3.1 Iterierte Morphismen 103
7.3.2 Ein kryptographisches System auf der Basis des Mitgliedsproblems . 109
7.4 Chiffrierungen auf zahlentheoretischer Basis 110
7.4.1 Einiges aus der Zahlentheorie 111
7.4.2 Das Schlüsselsystem von Rivest, Shamir und Adleman 118
7.4.3 Ein Verfahren mit Hilfe des diskreten Logarithmus 121
7.4.4 Signaturen und Hashfunktionen 122

Literaturverzeichnis **127**

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 Verschlüsselung, der Kryptoanalyst beliebig viele Paare erzeugen kann, die aus einem Klar-

text 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* **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 **NP** enthalten. □

Um die Aufgabe des Kryptoanalysten wirklich schwer zu machen, ist es daher wünschenswert, dass sie ein **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* **NP***-vollständig ist, dann gilt* $\mathbf{NP} = co - \mathbf{NP}$.

Beweis. Analog zum Beweis von Satz 7.1 kann gezeigt werden, dass die Aufgabe C des Kryptoanalysten in $co - \mathbf{NP}$ liegt.

Da C nach Voraussetzung **NP**-vollständig ist, kann jede Sprache $L \in \mathbf{NP}$ 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 **NP** liegt, liegt damit auch das Komplement von L in **NP**. Damit ist L in $co - \mathbf{NP}$ und folglich $\mathbf{NP} \subseteq co - \mathbf{NP}$ gezeigt.

Sei nun L in $co - \mathbf{NP}$. Dann ist das Komplement von L in **NP** und nach dem Vorstehenden damit in $co - \mathbf{NP}$. Somit ist L in **NP**. Daher gilt auch $co - \mathbf{NP} \subseteq \mathbf{NP}$. □

Da die Gleichheit $co-\mathbf{NP} = \mathbf{NP}$ äquivalent zu der Gleichheit $\mathbf{P} = \mathbf{NP}$ ist, ist davon auszugehen, dass $co-\mathbf{NP} = \mathbf{NP}$ nicht gilt. Daher ist nicht anzunehmen, dass die Aufgabe des Kryptoanalysten **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 **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 **NP**-vollständige, unentscheidbare Probleme bzw. ein Problem wählen, das nach bisheriger Kenntnis sehr schwer 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 $\alpha_1 a_1 + \alpha_2 a_2 + \ldots + \alpha_n a_n = a_{n+1}$ gilt (oder man treffe die Aussage, dass derartige Zahlen nicht existieren).

Dies 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 **NP**-vollständig ist. Da die allgemeine Erwartung ist, dass die Lösung **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

$$b = \beta_1 a_1 + \beta_2 a_2 + \ldots + \beta_n a_n$$

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.

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 $A' = (a_1, a_2, \dots, a_n)$ 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. □

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 monotoner 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) besitzt höchstens eine Lösung.

ii) Wenn das Rucksack-Problem (B, b) eine Lösung besitzt, dann stimmt sie mit der Lösung von $(A, t^{-1}b)$ 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 **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

$$\sum_{j=1}^{i-1} \lfloor ta_j/m \rfloor \leq \lfloor \frac{t \cdot \sum_{j=1}^{i-1} a_j}{m} \rfloor < \lfloor ta_i/m \rfloor$$

□

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 $\lfloor ta_n/m \rfloor > 0$ 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

$$y \geq y' = \frac{(\sum_{i=1}^{n} a_i) - m}{t - \sum_{i=1}^{n} \lfloor ta_i/m \rfloor} + 1$$

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

$$x_i \geq x_i' = \frac{(\sum_{j=1}^{i-1} a_j) - a_i}{\lfloor ta_i/m \rfloor - \sum_{i=1}^{n} \lfloor ta_i/m \rfloor} + 1$$

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 superwachsendes 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 $t^{-1} \bmod m$. Dann bestimmt er $A = t^{-1}B \bmod m$. 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 $(A', (t')^{-1}b \bmod m)$, 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 **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 $a_1a_2 \dots a_n$ mit $a_i \in X$

$$\sigma(a_1a_2 \dots a_n) = \sigma(a_1)\sigma(a_2) \dots \sigma(a_n)$$

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 $x_1x_2 \dots x_n \in \{0,1\}^*$ ordnen wir die Menge

$$\sigma_G(x_1x_2 \dots x_{n-1}x_n) = \sigma_{x_n}(\sigma_{x_{n-1}}(\dots(\sigma_{x_2}(\sigma_{x_1}(w)))\dots))$$

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

$$G_1 = (\{a, b, c\}, \sigma_0, \sigma_1, abc)$$

[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.

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

$$\sigma_1(a^r b^s c^t) = \{a^r b^{2s} c^{2t}\}\,.$$

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 $z = \#_1(x)$

$$\sigma_{G_1}(x) = \{a^u b^{2^z} c^{2^n} \mid 1 \le u \le 2^y\}\,.$$

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

$$\sigma_{G_2}(x) = (ab)^{2^n}\,.$$

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.

Um die Verschlüsselung des 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 **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*

$$h_{i_n}(h_{i_{n-1}}(\ldots(h_{i_1}(w))\ldots)) = h_{j_m}(h_{j_{m-1}}(\ldots(h_{j_1}(w))\ldots))$$

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 $n \geq 1$ 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

$$\begin{aligned}
\sigma_{G'_3}(0) &= \{kllmlllpl, kllmllml, kllmllpl, kllmlml, lnpllpl, lnpllml,\\
&\qquad lnpllpl, lnplml, kpllllpl, kplllml, kplllpl, kpllml\},\\
\sigma_{G'_3}(1) &= \{llnlllmlk, llnlllmln\},\\
\sigma_{G'_3}(11) &= \{l^6kl^{11}mlkl^5n, l^6kl^{11}mlkl^3kl^2, l^6kl^{11}mlnl^5n, l^6kl^{11}mlnl^3kl^2\}\,.
\end{aligned}$$

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.

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 **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

$$V = W \cup \{z, 0, 1, 2, \ldots, t, 1', 2', \ldots t'\} \cup \{[i] \mid 1 \le i \le t\}$$

und

$h_0(0) = 1$,	$h_1(0) = 1'$,
$h_0(i) = i + 1$ für $1 \le i < t$,	$h_1(i) = u_i 1[i]$ für $1 \le i \le t$,
$h_0(i') = (i+1)'$ für $1 \le i < t$,	$h_1(i') = u_i 1'[i]$ für $1 \le i \le t$,
$h_0(t) = h_0(t') = z$,	
$h_0(z) = [1]z$,	$h_1(z) = [2]z$,
$h_0(a) = a$ für $a \in W \cup \{[i] \mid 1 \le i \le t\}$,	$h_1(a) = a$ für $a \in W \cup \{[i] \mid 1 \le i \le t\}$,

Es ist leicht zu sehen, dass für

$$1 \le i \le t,\ s \ge 0,\ 1 \le j \le s\ \ 1 \le k_j \le t,\ m \ge 0$$

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

$$u_{k_1} u_{k_2} \dots u_{k_s} = u_{k_1} v_{k_2} \dots v_{k_s} \tag{7.6}$$

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 $x_1 x_2 \dots x_n$ wird durch das Wort $w_1 \# w_2 \# \dots \# w_n$ 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 **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

$$b = b_k 2^k + b_{k-1} 2^{k-1} + \ldots + b_2 2^2 + b_1 2 + b_0 \,,$$

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

$$a^b = a^{b_k 2^k + b_{k-1} 2^{k-1} + \ldots + b_2 2^2 + b_1 2 + b_0} = a^{b_k 2^k} a^{b_{k-1} 2^{k-1}} \ldots a^{b_2 2^2} a^{b_1 2} a^{a_0} \,.$$

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 $17^{73} \bmod 133$ 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, …, 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

$$\begin{aligned} r_{k-1} &= r_{k-3} - q_{k-1}r_{k-2} \\ &= r_{k-3} - q_{k-1}(r_{k-4} - q_{k-2}r_{k-3} \\ &= (1 - q_{k-2})r_{k_3} + q_{k_1}r_{k-4} \\ &= (1 - q_{k-2})(r_{k-5} - q_{k-3}r_{k-4}) + q_{k_1}r_{k-4} \\ &= (1 - q_{k-2})r_{k-5} + (q_{k+1} - (1 - q_{k-2})q_{k-3})r_{k_4} \\ &\dots \\ &= \alpha n + \beta m \end{aligned}$$

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 und machen eine Aussage zur Komplexität der Berechnung des größten gemeinsamen Teilers zweier Zahlen bzw. gewisser inverser Elemente.

Lemma 7.18 *Für eine natürliche Zahl n sei $p(n)$ die Anzahl der Primzahlen, die $\leq n$ sind. Dann gilt*

$$\lim_{n\to\infty} (p(n) - \frac{n}{\ln n}) = 0\,,$$

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* $\varphi : \mathbf{N} \to \mathbf{N}$ 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 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 $\bmod p$.

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

$$2^2 = 4, 2^3 = 8, 2^4 = 16 = 5, 2^5 = 10, 2^6 = 20 = 9, 2^7 = 18 = 7, 2^8 = 14 = 3, 2^9 = 6, 2^{10} = 12 = 1 \tag{7.9}$$

und

$$3^2 = 9, 3^3 = 27 = 5, 3^4 = 15 = 4, 3^5 = 12 = 1, \tag{7.10}$$

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 gibt es 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 $g^x = 1 \bmod p$ 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 $m < p-1$, ist ein Primfaktor von $p-1$ auch Teiler von n. 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 einen 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, 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 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 zu bilden. Die Berechnung dieser Potenzen ist nach Lemma 7.16 in $O(\log_2(\frac{p-1}{q}) \leq 0(\log_2(p))$ Schritten möglich. Die Aussage folgt nun weil es höchstens $O(\log_2(p))$ verschiedene Primteiler von p gibt. □

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 als 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 $y = g^{mj} =^b g^{-i} \bmod p$ 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 bzw. 7	$O(1)$	$O(1)$
2 bzw. 4	$O(m \cdot \log_2(m))$	$O(m)$
3 bzw. 5	$O(m \cdot \log_2(m))$	$O(m)$
6	$O(m)$	$O(m)$

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 $a = \Big(\frac{(p+q)}{2}\Big)^2$ eine Quadratzahl, die nicht viel größer als n ist. Da man aus $n = pq$ und $a = \Big(\frac{(p+q)}{2}\Big)^2$ 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 $0 \leq M \leq p-1$ 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

$$\begin{aligned} D(a,b) &= a^{p-1-x}b = (g^k)^{p-1-x}My^k = (g^k)^{p-1-x}M(g^x)^k \\ &= Mg^{k(p-1)-kx+kx} = M(g^{p-1})^k = M \bmod p \end{aligned}$$

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

$$1 - x \approx 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} \ldots = e^{-x}$$

(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 Geburtstage 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. □