Ausgewählte Kapitel aus der Kryptographie

Wolfgang M. Ruppert

Sommersemester 2001

25. Juli 2001[1]

[1]Im Sommersemester 2001 am Mathematischen Institut der Universität Erlangen abgehaltene Vorlesung

Inhaltsverzeichnis

Ankündigung

In der Kryptographie geht es darum, Daten oder Nachrichten so zu verschlüsseln, daß ein Unbefugter nichts damit anfangen kann. Während dies früher hauptsächlich im militärischen und politischen Bereich eine Rolle spielte, ist heutzutage die Kryptographie auch aus dem alltäglichen Leben nicht mehr wegzudenken, man denke etwa an abhörsicheres Telefonieren mit Mobiltelefonen, an Übertragung persönlicher Daten wie Kreditkartennummer, Kontonummer oder Paßwort via Internet, an den sicheren Einsatz von Chipkarten, etc.

Vor ungefähr 25 Jahren entstand mit dem Aufkommen der Public-Key-Kryptographie eine (sich gegenseitig befruchtende) Wechselwirkung zwischen Kryptographie und Computational Number Theory: Das Fehlen schneller und effektiver Verfahren zur Lösung gewisser mathematischer Probleme aus dem Bereich der Zahlentheorie (wie z.B. die Bestimmung der Primfaktorzerlegung einer natürlichen Zahl) nutzt man zur Konstruktion von Kryptosystemen, wobei die Sicherheit wesentlich vom aktuellen Forschungsstand in der Computational Number Theory abhängt.

Die Vorlesung findet parallel zu einem Seminar über Kryptographie statt und behandelt ausgewählte Themen.

Ort und Zeit: Montag, 8-10 Uhr, Übungsraum 2 (Beginn: 23. April 2001)

Anmerkungen

Das Skript und die Programme sind parallel zur Vorlesung entstanden, daher sind allerlei Mängel nicht auszuschließen.

Im Skript sind die folgenden Programme zu finden:

- otp.c, bema_ma, streamsh.c, blumgold.c, rc4.c, fips8.c, lfsr.c,
- aes_ma (aufgeteilt in Einzelfunktionen),
- kett_ma (aufgeteilt in Einzelfunktionen), kbfac_gmp.c squfof_gmp.c cfrac_gmp.c, rsa_ma,
- enigma_simlator.c.

Sind Rechenzeiten angegebenen, so beziehen sich diese auf einen Pentium-II-PC mit 128 MB Arbeitsspeicher, 400 MHz Taktfrequenz, Betriebssystem Linux.

KAPITEL 1

Zufallszahlen

1. Einführung

Was sind Zufallszahlen?
Spricht man von einer Folge von Zufallszahlen $x_i \in \{0, 1, \ldots, m-1\}$, so stellt man sich vor, daß x_n unabhängig von den Vorgängern $x_0, x_1, \ldots, x_{n-1}$ zufällig aus der Menge $\{0, 1, \ldots, m-1\}$ gewählt wurde.

Beispiel: Zufallszahlen aus $\{1, 2, 3, 4, 5, 6\}$ erhält man durch Würfeln eines (fehlerfreien) Würfels.

Beispiel: 1955 veröffentlichte die RAND Corporation ein Buch mit 1000000 Zufallszahlen (aus der Menge $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$).

Beispiel: Übers Internet kann man auch mit Zufallszahlen gefüllte CD-Roms kaufen (www.westphal-electronic.de).

Die Erzeugung von Zufallszahlen scheint also nicht ganz trivial zu sein. Oft verwendet man daher statt echten Zufallszahlen sogenannte Pseudozufallszahlen.

Was sind Pseudozufallszahlen?
Bei einer Folge von Pseudozufallszahlen x_i lassen sich die Folgenglieder aus dem Anfang der Folge $x_0, x_1, \ldots, x_{k-1}$, dem sogenannten Samen - Seed, nach einem festgelegten Verfahren/Algorithmus berechnen. Die Folge x_i sollte man (in ihrem statistischen Verhalten) nicht von einer echten Zufallsfolge unterscheiden können. (Darauf wird später ausführlicher eingegangen.)

Bemerkung: Im Prinzip könnte man sich bei den Zufallszahlen auf 0-1-Folgen beschränken, da sich die Binärentwicklung jeder natürlichen Zahl aus 0 und 1 zusammensetzt.

Wozu braucht man (Pseudo-)Zufallszahlen? Wir geben im folgenden ein paar Beispiele.

Zahlentheoretische Anwendungen: In der Zahlentheorie gibt es sogenannte probabilistische Algorithmen, deren Erfolg man zwar nicht garantieren, die aber praktisch schneller und einfacher als vergleichbare deterministische Algorithmen sind.

Beispiel: Ist $x_1, x_1, x_2, \ldots$ eine Zufallsfolge in $\{0, 1, \ldots, m-1\}$, so ist die Wahrscheinlichkeit, daß Indizes $1 \le k < \ell \le 3.1\sqrt{m}$ mit $x_k = x_\ell$ existieren, $> 99\%$.
Eine Anwendung ist die Pollardsche Faktorisierungsmethode: Will man die zusammengesetzte Zahl n mit Primteiler p faktorisieren, so bildet man $\mathrm{ggT}(x_k - x_\ell, n)$. Die Wahrscheinlichkeit, daß man unter den Zahlen $0 \le k < \ell \le 3.1\sqrt{p}$ das Resultat $p | \mathrm{ggT}(x_k - x_\ell, n)$ trifft, ist $> 99\%$. (In der Praxis kann man dies noch etwas einfacher machen.)

Kryptographische Anwendungen: Hier hat man in der Praxis noch eine weitere Forderung: Ein Außenstehender sollte die 'Zufallszahlen' nicht erraten können, da sonst die Verschlüsselung unsicher ist.

(1) Schlüsselerzeugung: Hier werden Zahlen benötigt, die man nicht durch irgendeine Systematik erraten kann.
(2) Bei vielen weiteren Verfahren werden Zufallszahlen benötigt. Kenntnis der Zufallszahl führt zur Entschlüsselung. (Beispiele: ElGamal-Verschlüsselung, ElGamal-Signatur, Fiat-Shamir-Identifikationsprotokoll)

Version vom 28.5.2001 (Anhang 25.7.2001)

Beispiel: Bei der ElGamal-Verschlüsselung ist (p, g, e) der geheime Schlüssel, mit $f = g^e \bmod p$ ist (p, g, f) der öffentliche Schlüssel. Eine Plaintextzahl a ergibt nach Wahl einer Zufallszahl z das verschlüsselte Paar $(g^z \bmod p, af^z \bmod p)$. Kennt man z, kann man sofort a berechnen.

2. Eine Motivation aus der Kryptographie: Das One-Time-Pad

Wir wollen in diesem Abschnitt zur weiteren Motivation der Beschäftigung mit Zufallszahlen ein einfaches Verschlüsselungsverfahren vorstellen, dessen Sicherheit in erster Linie von einer Folge von zufälligen Bits abhängt.

One-Time-Pad:

(1) Gegeben sei eine Nachricht/Ausgangstext/Plaintext in Form einer Folge von n Bits: $m_1, m_2, m_3, \ldots, m_n$.
(2) Man benötigt einen Schlüssel, der aus einer Folge von n Bits besteht: $k_1, k_2, k_3, \ldots, k_n$.
(3) Man berechnet $c_i = m_i + k_i \bmod 2$ und erhält als verschlüsselte Nachricht (Ciphertext) die Bitfolge $c_1, c_2, c_3, \ldots, c_n$.
(4) Die Entschlüsselung erhält man durch Anwendung des gleichen Verfahrens auf die Folge (c_i) mit der Schlüsselfolge (k_i) wegen $c_i + k_i \equiv (m_i + k_i) + k_i \equiv m_i \bmod 2$.

Bemerkungen:

(1) One-Time-Pads wurden von Joseph Mauborgne und Gilbert Vernam 1917 erfunden.
(2) Die Bitoperation $(m, k) \mapsto m + k \bmod 2$ wird auch als XOR (exklusives Oder) bezeichnet. Interpretiert man eine natürliche Zahl m über die Binärdarstellung als Bitfolge, so wird in C die komponentenweise Addition modulo 2 durch $m \wedge k$ gebildet, z.B.

$$18 \wedge 11 = (10010)_2 \wedge (01011)_2 = (11001)_2 = 25.$$

Das folgende C-Programm otp.c interpretiert Dateien als Bitfolgen und verschlüsselt mit dem One-Time-Pad.

```
/* Programm otp.c - One Time Pad - 29.4.2001 */

#include <stdio.h>

main()
{
  int m, k;
  char name[20];
  FILE *ein, *ein_key, *aus;

  printf("One-Time-Pad\n");
  printf("Eingabefile: "); scanf("%s",name);
  ein=fopen(name,"rb");
  strcat(name,".otp");
  aus=fopen(name,"wb"); /* Ausgabefile = Eingabefile.otp */
  printf("Ausgabefile: %s\n",name);
  printf("Schluesselfile: "); scanf("%s",name);
  ein_key=fopen(name,"rb");

  while ((m=getc(ein))!=EOF)
  {
    k=getc(ein_key);
    /* Ist die Schluesselfolge zu kurz, wird von vorne begonnen */
    if (k==EOF) { rewind(ein_key); k=getc(ein_key); }
    putc(m^k,aus);
  }
}
```

Beispiel: Wir wollen eine Datei verschlüsseln, die folgenden Text enthält:

```
FRISCHE FAHRT

Laue Luft kommt blau geflossen,
Fruehling, Fruehling soll es sein!
Waldwaerts Hoernerklang geschossen,
Mut'ger Augen lichter Schein;
Und das Wirren bunt und bunter
Wird ein magisch wilder Fluss,
In die schoene Welt hinunter
Lockt dich dieses Stromes Gruss.

Und ich mag mich nicht bewahren!
Weit von euch treibt mich der Wind,
Auf dem Strome will ich fahren,
Von dem Glanze selig blind!
Tausend Stimmen lockend schlagen,
Hoch Aurora flammend weht,
Fahre zu! Ich mag nicht fragen,
Wo die Fahrt zu Ende geht!
```

In Hexadezimaldarstellung sieht die Datei so aus:

```
46524953434845204614852540a0a4c617565204c756674206b6f6d6d7420626c6175
206765666c6f7373656e2c0a46727565686c696e672c2046727565686c696e6720736f
6c6c206573207365696e210a57616c6477616572747320486f65726e65726b6c616e67
2067657363686f7373656e2c0a4d7574276765722041756765 6e206c69636874657220
53636865696e3b0a556e642064617320576972726e2062756e7420756e642062756e
7465720a576972642065696e206d6167697363682077696c64657220466c7573732c0a
496e2064696520736368f656e652057656c742068696e756e7465720a4c6f636b7420
646963682064696573657320537472f6d6573204772757373732e0a0a556e6420696368
206d6167206d696368206e6963687420626577616872656e210a5765697420766f6e20
657563682074726569627420 6d6963682064657220577696e642c0a4175662064656d20
5374726f6d652077696c6c20696368206661687265 6e2c0a566f6e2064656d20476c61
6e7a652073656c696720626c696e64210a54617573656e64205374696d6d656e206c6f
636b656e64207363686c6167656e2c0a486f6368204175726f726120666c616d6d656e
6420776568742c0a4661687265207a752120496368206d6167206e6963687420667261
67656e2c0a576f20646965204661687274207a7520456e64652067656874210a
```

(Das 1. Byte ist also 46, was die Bitfolge 01000110 ergibt.)
Als Schlüsselfile verwenden wir eine Datei, die folgenden Text enthält:

```
Viele Boten gehn und gingen
Zwischen Erd' und Himmelslust,
Solchen Gruss kann keiner bringen,
Als ein Lied aus frischer Brust.
```

In Hexadezimaldarstellung ergibt dies die Bytefolge:

```
5669656c6520426f74656e206765686e20756e642067696e67656e0a5a776973636865
6e204572642720756e642048696d6d656c736c7573742c0a536f6c6368656e20477275
7373206b616e6e206b65696e6572206272696e67656e2c0a416c732065696e204c6965
642061757320667269736368657220427273742e0a
```

Mit dem One-Time-Pad verschlüsselt ergibt sich die Bytefolge:

```
103b2c3f2668074f32242672336f622241000b446c120f1a470e0167370349110f0910
4e472014084853060b0a0c422f1f1800041f051b14580c4c211a090b040c004767011a
1f1f000e124e1d45020b486432134c0605080b15111d0c422e09014e001b054c2d0702
44470406104809011a160d446f3f5536551216060e4b230e0002454c2b0c1c110b5247
360b06451c005f2a32070a47010f797a200001110d0b4e42301c1007551b0a44423d07
```

```
190817662405071754496 33d4f01020f0c1d432f52021a1f440e134e284c1e161a426f
3b4e4216000b47160d446524021600320c02546c010c0a550f0116526c3e0610081c45
1649211a55171d4b79331a453f11522d0211164e6715101b1d0e7f64314e034907040d
4e673b10491e0a0b0d4e4e2c110c5300170b1341201b0803446624091c07545a653d4f
09160b0d4e5435171c11070006080d06000f001b4e321b4e065e632f12034e486f2c4c
2054170603456c1e0c084c411c10484614081b110d0b5e2a141d1b53104b67762e090d
0b5a274f0700024900450a02491b0a452a33081b1400006e7a241d1a0e05000000291d
074c451b0a44532b01010c02091d407f3b1b4f62732e191107170f00211e141e1e4505
054e19450311456423134810174914124 44e6569294c1e41024900492f011144461314
1445085e63240c48011b456234141b065a2a2c1c45290b44274f13000654466f
```

(Wir betrachten jeweils das 1. Byte: 46 der Ausgangsdatei ergibt die Bitfolge 01000110, 56 der Schlüsseldatei ergibt die Bitfolge 01010110, was verschlüsselt die Bitfolge 00010000, also in Hexadezimaldarstellung das Byte 10 liefert.)

Bemerkung: Schaut die Folge $k_1, k_2, \ldots, k_n$ zufällig aus, so auch $c_1, c_2, \ldots, c_n$. Daher kann man keine großen Erkenntnisse aus dem Studium von $c_1, c_2, \ldots, c_n$ gewinnen. (Extrembeispiele: $m_i = 0$ für alle i ergibt verschlüsselt die Folge k_i, $m_i = 1$ für alle i ergibt verschlüsselt die Folge $1 - k_i$, die natürlich dann auch wieder zufällig aussieht.)
Auf jeden Fall muß die Folge $k_1, k_2, \ldots, k_n$ geheim bleiben.

3. Konstruktion von Pseudozufallszahlen mit Hilfe einer Funktion $f : M \to M$

Wir wollen Zahlenfolgen x_i modulo m konstruieren. Eine Möglichkeit erhält man durch Vorgabe einer Funktion $f : M \to M$ mit $M = \mathbf{Z}/(m)$ und eines Punktes $x_0 \in M = \mathbf{Z}/(m)$. Rekursiv definiert man dann $x_i = f(x_{i-1})$ für $i \geq 1$. (Natürlich ist zunächst überhaupt nicht klar, inwieweit man die Folge x_i als Zufallsfolge bezeichnen kann.)

Beispiel: $f(x) = x^2 + 2$, $x_0 = 0$, $m = 15$, also

$$x_0 = 0. \quad x_1 = 2, \quad x_2 = 6, \quad x_3 = 8, \quad x_4 = 6, \quad x_5 = 8, \quad x_6 = 6, \quad x_7 = 8, \ldots$$

Folgen, die nach obigem Prinzip konstruiert sind, haben ein einfaches Periodizitätsverhalten:
Sei $k < \ell$ minimal mit $x_k = x_\ell$. Wir setzen $r = \ell - k$. Dann sieht man durch Induktion, daß für $i \geq 0$ gilt $x_{k+i} = x_{\ell+i} = x_{(k+i)+r}$, d.h. $x_j = x_{r+j}$ für $j \geq k$. Die Zahl r heißt Periodenlänge,

$$x_0, x_1, \ldots, x_{k-1} \text{ Vorperiode }, x_k, x_{k+1}, \ldots, x_{k+r-1} \text{ Periode }.$$

Wir geben jetzt einige Beispiele für Funktionen f an:

Beispiel: $f(x) = ax+b \bmod m$. Man spricht von linearen Kongruenzgeneratoren. Sie spielen auch deshalb eine besondere Rolle, da nur wenige Rechenoperationen benutzt werden.

Beispiel: $f(x) = g^x \bmod m$.

Beispiel: $f(x) = x^e \bmod m$.

Beispiel: Von Neumann hat die sogenannte middle-square-Methode vorgeschlagen: Man betrachtet $M = \{0, 1, \ldots, 10^{2k} - 1\}$, also die Zahlen, die sich mit $2k$ Dezimalstellen schreiben lassen. Für $x \in M$ kann man x^2 als $4k$-stellige Dezimalzahl schreiben. $f(x)$ seien die $2k$ mittleren Dezimalstellen. Z.B. mit $k = 1$ und $x_0 = 23$:

$$23^2 = 0529, \quad 52^2 = 2704, \quad 70^2 = 4900, \quad 90^2 = 8100, \quad 10^2 = 0100, \quad \ldots$$

(Die middle-square-Methode ist allerdings nicht besonders gut.)

Beispiel: $f(x) = x^2 + a \bmod m$. Diese Funktionen werden bei der Pollardschen Faktorisierungsmethode verwendet, wobei man $a = 0$ und $a = -2$ ausschließt.

Die mathematische Analyse ist oft nicht leicht. Im nächsten Abschnitt behandeln wir ein Beispiel, das gut analysiert werden kann.

4. Lineare Kongruenzgeneratoren

Wir wählen hier $f(x) \equiv ax+b \bmod m$, konstruieren also nach Vorgabe von $x_0 \in \{0,\ldots,m-1\}$ eine Folge rekursiv durch $x_{i+1} \equiv ax_i + b \bmod m$. Wegen $x_i \in \{0,1,\ldots,m-1\}$ ist die Periode der Folge sicher $\leq m$.

Wir betrachten zunächst den Fall, daß m eine Primzahl p ist.

SATZ. *Sei p eine Primzahl und eine Folge x_i rekursiv definiert durch $x_{i+1} \equiv ax_i + b \bmod p$ nach Vorgabe von $a, b, x_0 \in \{0,1,\ldots,p-1\}$.*

(1) *Ist $a = 0$, so ist die Folge x_0, $x_i = b$ für $i \geq 1$.*
(2) *Ist $a = 1$, $b = 0$, so ist die Folge konstant: $x_i = x_0$.*
(3) *Ist $a = 1$, $b \neq 0$, so ist $x_i \equiv x_0 + ib \bmod p$ periodisch mit Periode p.*
(4) *Ist $a \neq 0, 1$ und $x_0 \equiv \frac{b}{1-a} \bmod p$, so ist $x_i = x_0$ konstant.*
(5) *Ist $a \neq 0, 1$, $x_0 \not\equiv \frac{b}{1-a} \bmod p$ und n minimal mit $a^n \equiv 1 \bmod p$, so ist die Folge x_i periodisch mit Periodenlänge n. Die Zahl n teilt $p-1$. (Anders ausgedrückt: Die Periodenlänge ist die Ordnung von a in der multiplikativen Gruppe von $\mathbf{Z}/(p)$.)*

Beweis: Für $a = 0$ ist $f(x) = b$ und damit erhalten wir die Folge

$$x_0, \quad x_1 = b, \quad x_2 = b, \quad x_3 = b, \ldots$$

Für $a = 1$, $b = 0$ ist $f(x) = x$, die entstehende Folge ist konstant.
Für $a = 1$, $b \neq 0$ ist $f(x) = x + b$ und daher

$$x_0, \quad x_1 \equiv x_0 + b, \quad x_2 \equiv x_0 + 2b, \quad \ldots, x_i \equiv ib \bmod p.$$

$x_i = x_j$ ist äquivalent mit $i \equiv j \bmod p$, die Folge hat also Periode p.
Sei nun $a \neq 0, 1$. Wann gilt $x_1 = x_0$? Genau dann, wenn $ax_0+b = x_0$, was gleichwertig mit $x_0 \equiv \frac{b}{1-a} \bmod p$ ist. Im Fall $x_1 = x_0$ ist die entstehende Folge konstant.
Sei jetzt $a \neq 0, 1$ und $x_1 \neq x_0$ vorausgesetzt. Aus $x_i = ax_{i-1} + b \bmod p$ und $x_{i+1} = ax_i + b \bmod p$ folgt $x_{i+1} - x_i = a(x_i - x_{i-1}) \bmod p$, was induktiv $x_{i+1} - x_i = a^i(x_1 - x_0) \bmod p$ und damit

$$\begin{aligned} x_n - x_0 &= (x_n - x_{n-1}) + (x_{n-1} - x_{n-2}) + \cdots + (x_2 - x_1) + (x_1 - x_0) = \\ &= a^{n-1}(x_1 - x_0) + a^{n-2}(x_1 - x_0) + \cdots + a(x_1 - x_0) + (x_1 - x_0) = \\ &= (a^{n-1} + a^{n-2} + \cdots + a + 1)(x_1 - x_0) = \\ &= \frac{a^n - 1}{a - 1}(x_1 - x_0) \bmod p \end{aligned}$$

ergibt. Es gilt $x_0 = x_n$ genau dann, wenn $a^n \equiv 1 \bmod p$ gilt. Diese n's sind alle Vielfache der Ordnung von a in der multiplikativen Gruppe von $\mathbf{Z}/(p)$, außerdem gilt für das minimale n dann $n|p-1$. ■

Bemerkung: Die Ordnung von a in der multiplikativen Gruppe von $\mathbf{Z}/(p)$ erhält man durch die Maple-Funktion numtheory[order](a, p).

Beispiel: $a = 102$, $b = 103$, $p = 1009$. Die Ordnung von 102 in der multiplikativen Gruppe von $\mathbf{Z}/(p)$ ist 1008. Für $x_0 = 988 \equiv \frac{b}{1-a} \bmod p$ wird die Folge konstant. Für $x_0 \neq 988$ hat die entstehende Folge Periode 1008. Bei Wahl von $x_0 = 987$ erhält man die Folge

987, 886, 674, 239, 265, 899, 991, 285, 921, 208, 130, 246, 979, 70, 180, ...

(mit Periode 1008).

Bemerkung: Maple liefert beim Aufruf von rand() Pseudozufallszahlen x_i, die rekursiv mit $x_i = f(x_{i-1})$ aus der Funktion $f(x) = ax \bmod p$ konstruiert werden, wobei

$$x_0 = a = 427419669081 \quad \text{und} \quad p = 999999999989$$

ist. p ist eine Primzahl, a hat Ordnung $p-1$ in der multiplikativen Gruppe von $\mathbf{Z}/(p)$. Die Periode der Folge ist also $p-1$.
Durch f=rand(m) erhält man bei Aufruf von f() Pseudozufallszahlen modulo m, die durch $x_i \bmod m$ gebildet werden. (Welche Periode hat x_i modulo m?)

Wir betrachten jetzt wieder den allgemeinen Fall: $x_{i+1} \equiv ax_i + b \bmod m$. Wir interessieren uns dafür, wann die Periode maximal, also m ist. Dies wird in folgendem Satz einfach charakterisiert:

SATZ. *Gegeben seien m, a, b, x_0. Die Folge x_i wird rekursiv durch $x_{i+1} \equiv ax_i \bmod m$ für $i \geq 0$ definiert. Genau dann hat die Folge x_i die (maximale) Periode m, falls gilt*

$$a \equiv 1 \bmod p \text{ für } p|m, \quad a \equiv 1 \bmod 4, \text{ falls } 4|m, \quad \mathrm{ggT}(b, m) = 1.$$

Für den Beweis sind einige vorbereitende Betrachtungen nützlich:
Wir definieren für $a \in \mathbf{Z}$, $k \in \mathbf{N}_0$

$$S_k(a) = 1 + a + \cdots + a^{k-1} = \sum_{i=0}^{k-1} a^i,$$

insbesondere $S_0(a) = 0$, $S_1(a) = 1$.

LEMMA.

$$S_{mn}(a) = S_m(a^n) S_n(a).$$

Beweis:

$$S_m(a^n) S_n(a) = \sum_{i=0}^{m-1} a^{ni} \sum_{j=0}^{n-1} a^j = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a^{ni+j}.$$

Nun ist

$$\{ni + j : 0 \leq i \leq m-1, 0 \leq j \leq n-1\} = \{0, 1, \ldots, mn-1\},$$

also folgt $S_m(a^n) S_n(a) = S_{mn}(a)$. ■

LEMMA. *Sei p prim und $a \in \mathbf{Z}$ mit $a \equiv 1 \bmod p$ bzw. $a \equiv 1 \bmod 4$ im Fall $p = 2$. Dann gilt für alle $n \geq 1$:*

$$S_{p^n}(a) \equiv 0 \bmod p^n, \quad \text{aber} \quad S_{p^n}(a) \not\equiv 0 \bmod p^{n+1},$$

d.h. in $S_{p^n}(a)$ steckt p genau n mal.

Beweis: Sei $a = 1 + cp$. Wir wollen $S_p(a)$ berechnen modulo p^2:

$$\begin{aligned}
S_p(1+cp) &= \sum_{i=0}^{p-1} (1+cp)^i = \\
&= 1 + (1+cp) + (1 + 2cp + c^2p^2) + (1 + 3cp + 3c^2p^2 + c^3p^3) + \cdots + \\
&\quad + \cdots + (1 + (p-1)cp + \binom{p-1}{2} c^2p^2 + \ldots) \\
&\equiv 1 + (1+cp) + (1+2cp) + (1+3cp) + \cdots + (1 + (p-1)cp) \bmod p^2 \\
&= p + \sum_{i=0}^{p-1} i \cdot cp = p + \frac{p(p-1)}{2} cp \\
&\equiv p \bmod p^2.
\end{aligned}$$

Die allgemeine Behauptung folgt jetzt durch Induktion aus der Formel

$$S_{p^n}(1+cp) = S_p((1+cp)^{p^{n-1}}) S_{p^{n-1}}(1+cp) = S_p(1+c'p) S_{p^{n-1}}(1+cp). \quad ■$$

LEMMA. *Sei p prim und $a \equiv 1 \bmod p$ bzw. $a \equiv 1 \bmod 4$ im Fall $p = 2$. Dann gilt:*

$$p^n | S_m(a) \quad \Longleftrightarrow \quad p^n | m.$$

Beweis: Sei $m = p^k \ell$ mit $\ell \not\equiv 0 \bmod p$. Dann ist

$$S_m(a) = S_\ell(a^{p^k}) S_{p^k}(a)$$

und man erhält mit dem letzten Lemma:

$$p^n | S_m(a) \quad \Longleftrightarrow \quad p^n | S_{p^k}(a) \quad \Longleftrightarrow \quad p^n | p^k \quad \Longleftrightarrow \quad p^n | m,$$

was zu zeigen war. ■

Beweis des Satzes: $\Rightarrow$ Die Folge x_i habe maximale Periode m. Für jeden Teiler $t|m$ hat dann die Folge $x_i \bmod t$ (maximale) Periode t. Ist p eine Primzahl, so ist also $x_i \bmod p$ periodisch mit Periode p. Nach einem vorangegangenen Satz ist die einzige Möglichkeit dann $a \equiv 1 \bmod p$ und $b \not\equiv 0 \bmod p$. (Dies liefert sofort $\mathrm{ggT}(b, m) = 1$.)
Wir betrachten den Fall $4|m$. Gerade wurde gezeigt, daß $a \equiv 1 \bmod 2$ gilt. Angenommen $a \equiv 3 \bmod 4$. Dann ist $f(x) \equiv 3x + b \equiv b - x \bmod 4$, also

$$x_1 \equiv b - x_0 \bmod 4, \quad x_2 \equiv x_0 \bmod 4, \quad x_3 \equiv x_1 \bmod 4, \ldots$$

was nicht sein sollte.
$\Leftarrow$ Nun setzen wir umgekehrt $a \equiv 1 \bmod p$, falls $p|m$, $a \equiv 1 \bmod 4$, falls $4|m$ und $\mathrm{ggT}(b, m) = 1$ voraus. Wir wählen $x_0 = 0$. Dann ist

$$x_n = S_n(a)b.$$

Das kleinste $n > 0$ mit $x_n = 0$ ist $n = m$. Daher hat die Folge Periode m. ■

Beispiel: Will man im Fall $m = 2^k$ aus $x_{i+1} \equiv ax_i + b \bmod 2^k$ eine Folge mit der Periode 2^k konstruieren, erhält man aus dem Satz die notwendigen und hinreichenden Bedingungen $a \equiv 1 \bmod 4$ und $b \equiv 1 \bmod 2$. Als numerisches Beispiel betrachten wir

$$m = 2^{64} = 18446744073709551616, \quad a = 16526768014310081049, \quad b = 952885106517919979.$$

Wir starten mit $x_0 = 0$ und erhalten in Binärdarstellung die Folge

0
1101001110010101001111110100101011011100001011001000111010 11
1111100011000001000111111011101001011101110011101011110111011110
1111001110011000000010111100110000111101010001000000111110011001
1110011001001110111110101011111001010101100100100000000011011100
1011110111001001010011000010110101110100011111110010110011001111
1010011110111011111110111111111000100111000111100011011111010
1010011101010000001111110010010100000010011110001010110101010101
1110010010111011001100001111110100110100011101110010111000111000
1000101111110000111100000010110000110001101101011110001100011
1110010110001100001100001011110000010110010111000111010010010110
1111011101000101000011100110100111100111011110001010111100100 01
1001010010000000000111001010001110011100101011111100000010100
1101000010000001000011001101111011110011100110000101010110111111
1100101101001110111000101000100111101101100110110011011010110010
1011001110110101111000010010100110000000110111000010001001101
1111010101110101001001001001101100100001111110101100111001110000
1101111010101001111110010101011111011001100010111010000111011011
1100111100001010000011101100110110010000110101000111110101001110
1010010000001101111110001010100010000110111010101000001100010 01
1101111010111001000100011101111011010110101010010010000101001100
...

Lineare Kongruenzgeneratoren haben den (kryptographischen) Nachteil, daß sich aus der Kenntnis einiger Folgenglieder meist das Bildungsgesetz der Folge leicht erschließen läßt, wie folgendes Beispiel zeigen soll.

Beispiel: Wir finden 10 Glieder einer Zahlenfolge

$$x_0 = 759847716285171439016630, \quad x_1 = 1079155487292941551994261,$$
$$x_2 = 2200637712395766057604 96, \quad x_3 = 5714138101164827648685 35,$$
$$x_4 = 1747648883124553429633 54, \quad x_5 = 7227087141104480679785 05,$$
$$x_6 = 4023343622008815945792 52, \quad x_7 = 4249566727602071810944 11,$$
$$x_8 = 1980578387902206582840 94, \quad x_9 = 9902891939511032914204 77,$$

von der wir annehmen, daß sie durch ein Bildungsgesetz $x_i \equiv ax_{i-1}+b \bmod m$ entstanden ist. Wir wollen a, b und m bestimmen.
Mit dem Ansatz $x_1 = ax_0 + b$, $x_2 = ax_1 + b$ berechnen wir (rationale Zahlen) a und b:

$$\begin{aligned} a &= -\frac{85909171605336494623376 5}{31930777100777011297763 1}, \\ b &= \frac{99736161174097146048165799064671382823863588764 1}{3193077710077701129776 31}. \end{aligned}$$

Nun sei $g_i = x_i - (ax_{i-1} + b)$ für $i \geq 3$ und h_i sei der Zähler von g_i:

$$\begin{aligned} h_3 &= -625849778834237143933545753269915700892706799616, \\ h_4 &= -150220318031312881615615631784027488487592689664, \\ h_5 &= -616456035344250588097528040844994786213469618176, \\ h_6 &= -248020053934909564912831377035906232035577954304, \\ h_7 &= -516027526136639869223511219487854103876882923520, \\ h_8 &= -569043447456355199987857380963355182055829798912, \\ h_9 &= -511004727963251951327747094542914959807814303744. \end{aligned}$$

Geht alles gut, sollte $h_i \equiv 0 \bmod m$ gelten und somit auch $m | \mathrm{ggT}(h_i, h_j)$. Also berechnen (und finden) wir

$$\mathrm{ggT}(h_3, h_4) = \mathrm{ggT}(h_3, h_4, h_5, h_6, h_7, h_8, h_9) = 2^{80}.$$

Daher probieren wir $m = 2^{80}$, $a = a \bmod m$, $b = b \bmod m$ und erhalten

$$a = 37778931862957161709573, \quad b 75557863725914323419143, \quad m = 1208925819614629174706176,$$

was tatsächlich unsere Ausgangsfolge beschreibt.

5. Schieberegister mit linearer Rückkoppelung — Linear feedback shift registers (LFSR)

Ein Schieberegister (mit Rückkoppelung) der Länge n besteht aus n Einträgen $s_i \in \{0,1\} \simeq \mathbf{F}_2$, die wir als

s_{n-1}	s_{n-2}	$\ldots$	s_1	s_0

oder als Vektor $(s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$ skizzieren und als Zustand bezeichnen. Bei einem Durchgang wird der Inhalt des Registers um 1 nach rechts verschoben, s_0 ausgegeben und der Platz ganz links aufgefüllt mit $s_n = f(s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$, wo $f : \{0,1\}^n \to \{0,1\}$ eine vorgegebene Funktion ist:

s_{n-1}	s_{n-2}	$\ldots$	s_1	s_0

$\longrightarrow$

s_n	s_{n-1}	$\ldots$	s_2	s_1

mit Ausgabe s_0 und Eingabe $s_n = f(s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$.

(Man sagt auch: Das Schieberegister wird getaktet.)
Wir iterieren jetzt diesen Vorgang und erhalten eine Folge von Zuständen, die wir untereinander schreiben, wobei links das neu hinzukommende Eingabebit, rechts das Ausgabebit steht:

	s_{n-1}	s_{n-2}	$\ldots$	s_1	s_0	s_0
$s_n = f(s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$	s_n	s_{n-1}	$\ldots$	s_2	s_1	s_1
$s_{n+1} = f(s_n, s_{n-1}, \ldots, s_2, s_1)$	s_{n+1}	s_n	$\ldots$	s_3	s_2	s_2
$s_{n+2} = f(s_{n+1}, s_n, \ldots, s_3, s_2)$	s_{n+2}	s_{n+1}	$\ldots$	s_4	s_3	s_3
$s_{n+3} = f(s_{n+2}, s_{n+1}, \ldots, s_4, s_3)$	s_{n+3}	s_{n+2}	$\ldots$	s_5	s_4	s_4
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Durch Iteration erhält man eine Folge $(s_i)_{i \geq 0}$, die man mathematisch auch folgendermaßen beschreiben kann:

$$s_i = f(s_{i-1}, s_{i-2}, \ldots, s_{i-n}) \text{ für } i \geq n.$$

Die Folge $(s_i)_{i \geq 0}$ wird als Ausgabefolge des Schieberegisters bezeichnet. Der Zustand $(s_{n-1}, \ldots, s_1, s_0)$ wird auch als Anfangszustand bezeichnet.
(Im folgenden rechnen wir über dem Körper $\mathbf{F}_2$ mit den beiden Elementen 0 und 1.)

Beispiel: Wir wählen $n = 4$ und $f(a_3, a_2, a_1, a_0) = a_2 + a_0$, also haben wir als Übergangsoperation

$$(a_3, a_2, a_1, a_0) \longrightarrow (a_2 + a_0, a_3, a_2, a_1).$$

Wählen wir als Anfangszustand $(1, 0, 0, 0)$, so erhalten wir folgende Zustandsfolge:

1	0	0	0
0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0
0	0	0	1
1	0	0	0
⋮	⋮	⋮	⋮

Wir haben also Periodizität der Länge 6. Die zugehörige Ausgabefolge (s_i) hat die Periode $0, 0, 0, 1, 0, 1$ (mit Periodelänge 6).

Schieberegister mit linearer Rückkoppelung: Ist

$$f(s_{n-1}, s_{n-2}, \ldots, s_1, s_0) = c_1 s_{n-1} + c_2 s_{n-2} + \cdots + c_{n-1} s_1 + c_n s_0$$

mit $c_i \in \mathbf{F}_2$, so haben wir ein Schieberegister mit linearer Rückkoppelung (linear feedback shift register). Im folgenden werden wir nur solche betrachten. Die Konstanten c_i werden eindeutig durch das sogenannte Verbindungspolynom

$$C(t) = 1 + c_1 t + c_2 t^2 + \cdots + c_n t^n \in \mathbf{F}_2[t]$$

angegeben. Ein lineares Schieberegister wird also durch das Paar $(n, C(t))$ gegeben. Wegen $c_i \in \{0, 1\}$ gilt

$$f(s_{n-1}, s_{n-2}, \ldots, s_1, s_0) = \sum_{1 \le i \le n, c_i \neq 0} s_{n-i},$$

man addiert also nur die Einträge mit $c_i \neq 0$ (modulo 2) auf.

Eingabebit	c_1	c_2	$\ldots$	c_{n-1}	c_n	Ausgabebit
	s_{n-1}	s_{n-2}	$\ldots$	s_1	s_0	s_0
$s_n = c_1 s_{n-1} + c_2 s_{n-2} + \cdots + c_{n-1} s_1 + c_n s_0$	s_n	s_{n-1}	$\ldots$	s_2	s_1	s_1
$s_{n+1} = c_1 s_n + c_2 s_{n-1} + \cdots + c_{n-1} s_2 + c_n s_1$	s_{n+1}	s_n	$\ldots$	s_3	s_2	s_2
$s_{n+2} = c_1 s_{n+1} + c_2 s_n + \cdots + c_{n-1} s_3 + c_n s_2$	s_{n+2}	s_{n+1}	$\ldots$	s_4	s_3	s_3
$s_{n+3} = c_1 s_{n+2} + c_2 s_{n+1} + \cdots + c_{n-1} s_4 + c_n s_3$	s_{n+3}	s_{n+2}	$\ldots$	s_5	s_4	s_4
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Singuläre Schieberegister: Ein durch $(n, C(t))$ mit $C(t) = 1 + c_1 t + \cdots + c_n t^n$ gegebenes lineares Schieberegister nennt man singulär, falls $c_n = 0$ ist, andernfalls nichtsingulär. Im singulären Fall erhält man die Zustandsfolge

Eingabebit	c_1	c_2	$\ldots$	c_{n-1}	0	Ausgabebit
	s_{n-1}	s_{n-2}	$\ldots$	s_1	s_0	s_0
$s_n = c_1 s_{n-1} + c_2 s_{n-2} + \cdots + c_{n-1} s_1$	s_n	s_{n-1}	$\ldots$	s_2	s_1	s_1
$s_{n+1} = c_1 s_n + c_2 s_{n-1} + \cdots + c_{n-1} s_2$	s_{n+1}	s_n	$\ldots$	s_3	s_2	s_2
$s_{n+2} = c_1 s_{n+1} + c_2 s_n + \cdots + c_{n-1} s_3$	s_{n+2}	s_{n+1}	$\ldots$	s_4	s_3	s_3
$s_{n+3} = c_1 s_{n+2} + c_2 s_{n+1} + \cdots + c_{n-1} s_4$	s_{n+3}	s_{n+2}	$\ldots$	s_5	s_4	s_4
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Betrachtet man jetzt das Schieberegister $(n-1, C(t))$ mit dem Anfangszustand $(s_{n-1}, s_{n-2}, \ldots, s_2, s_1)$, so erhält man folgende Zustandsfolge:

Eingabebit	c_1	c_2	$\ldots$	c_{n-2}	c_{n-1}	Ausgabebit
	s_{n-1}	s_{n-2}	$\ldots$	s_2	s_1	s_1
$s_n = c_1 s_{n-1} + c_2 s_{n-2} + \cdots + c_{n-1} s_1$	s_n	s_{n-1}	$\ldots$	s_3	s_2	s_2
$s_{n+1} = c_1 s_n + c_2 s_{n-1} + \cdots + c_{n-1} s_2$	s_{n+1}	s_n	$\ldots$	s_4	s_3	s_3
$s_{n+2} = c_1 s_{n+1} + c_2 s_n + \cdots + c_{n-1} s_3$	s_{n+2}	s_{n+1}	$\ldots$	s_5	s_4	s_4
$s_{n+3} = c_1 s_{n+2} + c_2 s_{n+1} + \cdots + c_{n-1} s_4$	s_{n+3}	s_{n+2}	$\ldots$	s_6	s_5	s_5
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Bis auf s_0 liefert also das gekürzte Register $(n-1, C(t))$ die gleiche Ausgabefolge wie das ursprüngliche Register. Interessiert man sich nur dafür, wie die Ausgabefolge schließlich aussieht, kann man sich folglich auf nichtsinguläre Schieberegister beschränken. Wir werden das im folgenden tun.

Beschreibung von linearen Schieberegistern durch Matrizen: Ein Schieberegister sei durch $(n, C(t))$ gegeben mit $C(t) = 1 + c_1 t + \cdots + c_n t^n$. Der Übergang von einem Zustand in den nächsten kann nun leicht durch Matrizenmultiplikation beschrieben werden:

$$(s_n, s_{n-1}, \ldots, s_2, s_1) = (s_{n-1}, s_{n-2}, \ldots, s_1, s_0) \begin{pmatrix} c_1 & 1 & 0 & \ldots & 0 \\ c_2 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n-1} & 0 & 0 & \ldots & 1 \\ c_n & 0 & 0 & \ldots & 0 \end{pmatrix}.$$

Sei $A = \begin{pmatrix} c_1 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1} & 0 & \ldots & 1 \\ c_n & 0 & \ldots & 0 \end{pmatrix}$ diese Übergangsmatrix. Das charakteristische Polynom der Matrix A ist

$$c(t) = t^n + c_1 t^{n-1} + c_2 t^{n-2} + \cdots + c_{n-1} t + c_n = t^n C(\frac{1}{t}).$$

(Man beachte, daß es in Charakteristik 2 keine Vorzeichen gibt.) Ist $s = (s_{n-1}, \ldots, s_0)$ der Anfangszustand, so ist also die Folge der Zustände

$$s, \quad sA, \quad sA^2, \quad sA^3, \quad sA^4, \quad \ldots$$

Wegen $\det A = c_n$ ist das Schieberegister genau dann singulär, wenn die Übergangsmatrix A singulär ist.

Periodizität: Sei $(n, C(t))$ ein Schieberegister mit Übergangsmatrix A und Anfangszustand $s = (s_{n-1}, \ldots, s_0) \in \mathbf{F}_2^n \setminus \{0\}$. Da es nur 2^n mögliche Zustände gibt, gibt es Indizes $0 \le i < j \le 2^n$ mit $sA^i = sA^j$. Es folgt $sA^{i+1} = sA^{j+1}$, $sA^{i+2} = sA^{j+2}$, $\ldots$, also wird die Zustandsfolge schließlich periodisch.
Wir setzen jetzt voraus, daß das Schieberegister nichtsingulär ist. Aus $sA^i = sA^j$ folgt dann durch Multiplikation mit A^{-1} die Beziehung $sA^{i-1} = sA^{j-1}$. Daher können wir o.E. $i = 0$ voraussetzen. Wählen wir $j > 0$ minimal mit $s = sA^j$, so folgt $j \le 2^n - 1$, da es nur $2^n - 1$ Zustände $\neq 0$ gibt.
Jeder Anfangszustand $\neq 0$ liefert also eine periodische Folge mit Periodenlänge $\le 2^n - 1$. Die Menge der Zustände $\neq 0$ zerlegt sich dann in disjunkte Zykel:

$$\mathbf{F}_2^n \setminus \{0\} = \{s_1 = s_1 A^{r_1}, s_1 A, s_1 A^2, \ldots, s_1 A^{r_1 - 1}\} \cup \{s_2 = s_2 A^{r_2}, s_2 A, s_2 A^2, \ldots, s_2 A^{r_2 - 1}\} \cup \ldots$$

Beispiel: Wir betrachten das Schieberegister mit $(4, 1 + t^2 + t^4)$. Die Übergangsfunktion lautet also $(s_3, s_2, s_1, s_0) \mapsto (s_2 + s_0, s_3, s_2, s_1)$. Die Menge der Zustände $\neq 0$ zerfällt in 3 Zykel (mit Periodenlängen

6, 6, 3):

0	1	0	1
0	0	0	1
1	0	0	0
0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0
0	0	0	1
⋮	⋮	⋮	⋮

0	1	0	1
0	0	1	1
1	0	0	1
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
⋮	⋮	⋮	⋮

0	1	0	1
0	1	1	0
1	0	1	1
1	1	0	1
0	1	1	0
1	0	1	1
1	1	0	1
0	1	1	0
⋮	⋮	⋮	⋮

(Die oberste Zeile bezeichnet jeweils $(c_1, c_2, c_3, c_4) = (0, 1, 0, 1)$.)

Bemerkung: Während nichtsinguläre Schieberegister immer periodische Ausgabefolgen liefern, gibt es bei singulären Schieberegistern auch Ausgabefolgen mit einer Vorperiode. Wählt man nämlich bei einem singulären Schieberegister $(0, 0, \ldots, 0, 0, 1)$ als Anfangszustand, so wird die Ausgabefolge $1, 0, 0, 0, 0, \ldots$.

SATZ. *Durch $(n, C(t))$ werde ein nichtsinguläres lineares Schieberegister gegeben. Das Polynom $C(t) \in \mathbf{F}_2[t]$ sei irreduzibel. Sei m die kleinste natürliche Zahl mit $C(t)|t^m - 1$ in $\mathbf{F}_2[t]$. Dann gilt:*

(1) *Jeder Anfangszustand $s = (s_{n-1}, s_{n-2}, \ldots, s_1, s_0) \neq 0$ liefert eine periodische Folge mit Periodenlänge m.*
(2) *m teilt $2^n - 1$.*

Beweis: Sei $c(t) = t^n + c_1 t^{n-1} + \cdots + c_n$ das charakteristische Polynom der Übergangsmatrix A. Die Voraussetzungen des Satzes gelten dann auch für das Polynom $c(t)$, d.h. $c(t)$ ist irreduzibel und m ist minimal mit $c(t)|t^m - 1$. (Aus $t^m - 1 = C(t)D(t)$ folgt nämlich $(\frac{1}{t})^m - 1 = C(\frac{1}{t})D(\frac{1}{t})$, indem man t durch $\frac{1}{t}$ ersetzt, dann durch Multiplikation mit t^m die Gleichung $1 - t^m = t^n C(\frac{1}{t}) \cdot t^{m-n} D(\frac{1}{t}) = c(t) \cdot t^{m-n} D(\frac{1}{t})$, also teilt $c(t)$ das Polynom $t^m - 1$. Die anderen Aussagen zeigt man analog.) Es gibt zwei Fälle:

(1) Für $1 \leq k < m$ ist $c(t)$ kein Teiler von $t^k - 1$, also ist $\mathrm{ggT}(t^k - 1, c(t)) = 1$ in $\mathbf{F}_2[t]$, da $c(t)$ irreduzibel vorausgesetzt war. Mit dem erweiterten euklidischen Algorithmus findet man Polynome $g(t), h(t) \in \mathbf{F}_2[t]$ mit $g(t)(t^k - 1) + h(t)g(t) = 1$. Setzt man $t = A$ ein, erhält man wegen $c(A) = 0$ die Matrizengleichung $g(A)(A^k - 1) = 1$, also ist $A^k - 1$ eine invertierbare Matrix, was für $s \neq 0$ sofort $s(A^k - 1) \neq 0$ und somit $sA^k \neq s$ liefert. Der Anfangszustand s hat also keine Periode k.
(2) Es gibt ein Polynom $g(t) \in \mathbf{F}_2[t]$ mit $t^m - 1 = c(t)g(t)$, was dann $A^m - 1 = c(A)g(A) = 0$ und somit $A^m = 1$ liefert. Damit erhält man $sA^m = s$ für alle Zustände s. Insbesondere haben alle Anfangszustände $\neq 0$ Periode m.

Durch Übergang $s \to sA$ wird die ($2^n - 1$-elementige) Menge der Zustände $\neq 0$ in Zykel der Länge m zerlegt, was $m|2^n - 1$ liefert und damit die letzte Behauptung zeigt. ■

Bemerkung: Teilt das Polynom $g(t) \in \mathbf{F}_2[t]$ das Polynom $h(t) \in \mathbf{F}_2[t]$ in $\mathbf{F}_2[t]$, so liefert die Maple-Funktion 'Rem(h,g,t) mod 2' den Wert 0. Auf diese Weise kann man obigen Satz schnell auf kleinere Beispiele anwenden.

Beispiel: Es gibt nur 3 irreduzible Polynome $C(t) \in \mathbf{F}_2[t]$ vom Grad 4, nämlich

$$1 + t + t^4, \quad 1 + t^3 + t^4, \quad 1 + t + t^2 + t^3 + t^4.$$

In den ersten beiden Fällen ist das minimale $m \geq 1$ mit $C(t)|t^{2^m - 1}$ die Zahl $m = 15$, im letzten Fall $m = 5$.

Primitive Polynome: Ist $C(t) \in \mathbf{F}_2[t]$ irreduzibel vom Grad n und ist das minimale m mit $C(t)|t^m - 1$ die (größt mögliche) Zahl $m = 2^n - 1$, so nennt man $C(t)$ ein primitives Polynom.
Jeder Anfangszustand $\neq 0$ des zugehörigen Schieberegisters durchläuft dann alle Zustände $\neq 0$ und hat maximale Periode $2^n - 1$.

Beispiel: $1+t^3+t^{31}$, $1+t^6+t^{31}$, $1+t^7+t^{31}$ sind primitive Polynome modulo 2. (Bei den zugehörigen Schieberegistern liefert dies Periodenlänge $2^{31}-1 = 2147483647$.)

Algebraische Interpretation: Ein irreduzibles Polynom $C(t) \in \mathbf{F}_2[t]$ vom Grad n definiert eine Körpererweiterung von $\mathbf{F}_2$ vom Grad n, also einen endlichen Körper $\mathbf{F}_{2^n}$ mit 2^n Elementen. Die Elemente dieses Körpers kann man sich durch Polynome vom Grad $< n$ repräsentiert denken:

$$\mathbf{F}_{2^n} \simeq \{a_0 + a_1 t + a_2 t^2 + \cdots + a_{n-1}t^{n-1} : a_i \in \mathbf{F}_2\}.$$

Die Addition ist klar, die Multiplikation ist durch $(g(t), h(t)) \mapsto g(t)h(t) \bmod C(t)$ gegeben. Die multiplikative Gruppe des Körpers $\mathbf{F}_{2^n}$ ist zyklisch von der Ordnung $2^n - 1$. Die Ordnung eines Elements $g(t)$ ist die kleinste Zahl $k \geq 1$ mit $g(t)^k \equiv 1 \bmod C(t)$. Außerdem gilt $k | 2^n - 1$. Das Polynom $C(t)$ nennt man dann primitiv, wenn t maximale Ordnung hat, also die multiplikative Gruppe des Körpers erzeugt, d.h. wenn sich jedes Element $\neq 0$ als Potenz von t darstellen läßt:

$$\mathbf{F}_{2^n} \simeq \{0, 1, t, t^2, t^3, \ldots, t^{2^n-2} \bmod C(t)\}.$$

Wir können jetzt die Schieberegister mit Ausgabefolgen maximaler Periodenlänge charakterisieren:

SATZ. *Genau dann liefert ein lineares Schieberegister $(n, C(t))$ eine Ausgabefolge maximaler Periodenlänge $2^n - 1$, wenn $C(t)$ primitiv vom Grad n ist.*

Beweis: Sei A die Übergangsmatrix und $c(t)$ das charakteristische Polynom des Schieberegisters, also $c(t) = t^n C(\frac{1}{t})$.

(1) Ist $C(t)$ primitiv, so wissen wir, daß das Schieberegister Ausgabefolgen maximaler Periodenlänge liefert. Wir können also umgekehrt jetzt annehmen, daß maximale Periodenlänge vorliegt, d.h. wir haben einen Anfangszustand $s = (s_{n-1}, \ldots, s_1, s_0) \in \mathbf{F}_2^n \setminus \{0\}$ und

$$\mathbf{F}_2^n = \{0, s, sA, sA^2, sA^3, \ldots, \}.$$

Eine Folgerung ist, daß jeder A-invariante lineare Teilraum von $\mathbf{F}_2^n$ entweder $\{0\}$ oder $\mathbf{F}_2^n$ ist.

(2) Wir nehmen nun an, $c(t)$ ist reduzibel, also $c(t) = g(t)h(t)$ mit normierten Polynomen vom Grad ≥ 1. Aus $c(A) = 0$ folgt $xg(A)h(A) = 0$ für alle $x \in \mathbf{F}_2^n$. Die Menge $U = \{x \in \mathbf{F}_2^n : xg(A) = 0\}$ ist ein A-invarianter linearer Teilraum. Also gibt es zwei Möglichkeiten:
   (a) $U = 0$: Dann ist $g(A)$ invertierbar, was wegen $c(A) = 0$ sofort $h(A) = 0$ ergibt. Schreibt man $h(t) = t^l + b_{l-1}t^{l-1} + \cdots + b_0$, so folgt

$$A^l = b_0 \cdot 1 + b_1 A + \cdots + b_{l-1}A^{l-1}.$$

   Daher ist $\mathbf{F}_2 s + \mathbf{F}_2 sA + \cdots + \mathbf{F}_2 sA^{l-1}$ ein A-invarianter linearer Teilraum, verschieden von 0 und $\mathbf{F}_2^n$, was nicht sein kann. Also bleibt nur der Fall $U = \mathbf{F}_2^n$.
   (b) $U = \mathbf{F}_2^n$: Dann ist $g(A) = 0$. Dies führt aber mit der gleichen Überlegung wie eben zu einem Widerspruch.
   Die Annahme, daß $c(t)$ reduzibel ist, ist also falsch.

(3) Nun ist $c(t)$ irreduzibel. Nach einem vorangegangenen Satz ist die Periodenlänge dann das kleinste m mit $C(t) | t^m - 1$. Da wir Periodenlänge $m = 2^n - 1$ vorausgesetzt haben, ist $C(t)$ dann nach Definition primitiv. ■

Bemerkung: Lineare Schieberegister sind recht nützlich, da sie sich einfach beschreiben lassen und bei Wahl eines geeigneten Verbindungspolynoms Aussagefolgen großer Periodenlänge liefern.

6. Lineare Komplexität

Wir betrachten weiterhin binäre Folgen $s = s_0, s_1, s_2, s_3, \ldots$. Eine endliche Teilfolge $s_0, s_1, s_2, \ldots, s_{m-1}$ der Länge m bezeichnen wir mit s^m.
Wir sagen, ein Schieberegister erzeugt eine Folge s, wenn s die Ausgabefolge eines Schieberegisters ist.

SATZ. *Ist $s = s_0, s_1, s_2, \ldots$ eine periodische binäre Folge mit Periodenlänge n, so ist s Ausgabefolge des Schieberegisters $(n, 1 + t^n)$ mit Anfangszustand $(s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$.*

Beweis: Wir betrachten das Schieberegister:

0	0	...	0	1	Ausgabebit
s_{n-1}	s_{n-2}	...	s_1	s_0	s_0
s_0	s_{n-1}	...	s_2	s_1	s_1
s_1	s_0	...	s_3	s_2	s_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_{n-2}	s_{n-3}	...	s_0	s_{n-1}	s_{n-1}
s_{n-1}	s_{n-2}	...	s_1	s_0	s_0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

was sofort die Behauptung zeigt. ■

DEFINITION. Die lineare Komplexität $L(s)$ einer Folge s wird wie folgt definiert:

(1) Ist s die 0-Folge, so sei $L(s) = 0$.
(2) Erzeugt kein Schieberegister s, so sei $L(s) = \infty$,
(3) andernfalls sei $L(s)$ die Länge eines kürzesten Schieberegisters, das s erzeugt.

Die lineare Komplexität einer endlichen Folge s^m ist die Länge eines kürzesten Schieberegisters, das s^m als die ersten m Folgenglieder ausgibt.

Beispiele:

(1) Wir betrachten die Folge s mit Periode 3

$$1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, \ldots$$

Das Schieberegister $(2, 1+t+t^2)$ mit dem Anfangszustand $(0, 1)$ hat die Folge als Ausgabefolge, denn die Zustandsfolge ist

1	1	Ausgabefolge
0	1	1
1	0	0
1	1	1
0	1	1
$\vdots$	$\vdots$	$\vdots$

Also ist die lineare Komplexität $L(s) = 2$, da Schieberegister der Länge 1 nur Folgen mit Periodenlänge $2^1 - 1 = 1$ liefern können.

(2) Die Folge s mit Periodenlänge 3

$$0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, \ldots$$

wird vom Schieberegister $(3, 1+t^3)$ mit Anfangszustand $(0, 0, 1)$ erzeugt, wie wir oben allgemein gesehen haben. Ein Schieberegister der Länge ≤ 2 mit Anfangszustand $(0, 0)$ oder (0) kommt nicht in Frage, also folgt für die lineare Komplexität $L(s) = 3$.

SATZ. *Sei $s = (s_0, s_1, s_2, s_3, \ldots)$ eine periodische binäre Folge mit Periodenlänge m. Dann gilt für die lineare Komplexität $L(s)$*

$$L(s) \leq m \leq 2^{L(s)-1}.$$

Beweis: Das Schieberegister $(m, 1+t^m)$ mit dem Anfangszustand $(s_{m-1}, s_{m-2}, \ldots, s_1, s_0)$ hat s als Ausgabefolge, also folgt $L(s) \leq m$. Ist umgekehrt $(n, C(t))$ ein minimales Schieberegister mit s als Ausgabefolge, also $n = L(s)$, so gilt für die Periodenlänge $m \leq 2^n - 1 = 2^{L(s)} - 1$, was die zweite Ungleichung beweist. ■

Die Abschätzungen des letzten Satzes können nicht verbessert werden, wie folgende Beispiele zeigen:

Beispiele:

(1) Sei s die Folge mit Periode $(0, 0, \ldots, 0, 1)$ und Periodenlänge m. Dann hat das Schieberegister $(m, 1 + t^m)$ die Folge als Ausgabefolge, also $L(s) \leq m$. Da aber kürzere Schieberegister mit $(0, \ldots, 0)$ als Anfangszustand nur die Nullfolge ausgeben würden, folgt $L(s) = m$.

(2) Sei s die Ausgabefolge eines Schieberegisters $(n, C(t))$ mit primitivem Polynom $C(t)$. Dann hat s Periodenlänge $2^n - 1$ und nach obiger Abschätzung muß $L(s) = n$ gelten.

Überlegung: $s^n = s_0, s_1, \ldots, s_{n-1}$ sei eine endliche binäre Folge. Wir wollen ein erzeugendes Schieberegister $(m, 1 + c_1 t + \cdots + c_m t^m)$ konstruieren, wobei m, c_1, $\ldots$, c_m noch unbekannt sind. Der Anfangszustand muß $(s_{m-1}, s_{m-2}, \ldots, s_1, s_0)$ sein, außerdem müssen die $n - m$ Gleichungen

$$s_k = c_1 s_{k-1} + c_2 s_{k-2} + \cdots + c_m s_{k-m} \text{ für } m \leq k \leq n-1$$

erfüllt sein. Man hat also m Unbekannte und $n - m$ Gleichungen. So erhält man für $n = 4$ und $m = 2$ die Gleichungen

$$\begin{aligned} s_2 &= c_1 s_1 + c_2 s_0, \\ s_3 &= c_1^2 s_1 + c_1 c_2 s_0 + c_2 s_1. \end{aligned}$$

Ist das Gleichungssystem lösbar, so gilt für die lineare Komplexität $L(s^n) \leq m$. Durch Variation von m kann man so im Prinzip $L(s)$ bestimmen.
(Ist man in einer hinreichend 'allgemeinen' Situation, wird man für $m \approx \frac{n}{2}$ Lösungen erwarten, also $L(s^n) \approx \frac{n}{2}$.)
Wir werden allerdings gleich ein wesentlich besseres Verfahren zur Berechnung von $L(s^n)$ angeben.

DEFINITION. Sei $s = s_0, s_1, s_2, \ldots$ eine binäre Folge. Sei L_n die lineare Komplexität der Teilfolge $s^n = s_0, s_1, s_2, \ldots, s_{n-1}$. Die Folge $L_1, L_2, L_3, \ldots$ heißt dann das lineare Komplexitätsprofil von s.

Die lineare Komplexität einer endlichen Folge, das Komplexitätsprofil, sowie erzeugende Schieberegister lassen sich gut mit dem folgenden Algorithmus berechnen. (Wir verzichten auf einen Beweis.)

Berlekamp-Massey-Algorithmus: Sei eine binäre Folge $s^n = s_0, s_1, \ldots, s_{n-1}$ gegeben.

(1) Setze $C(t) := 1$, $L := 0$, $m := -1$, $B(t) := 1$, $i := 0$.
(2) Führe folgende Schritte aus, solange $i < n$ ist:
  (a) Berechne $d := (s_i + \sum_{j=1}^{L} c_j s_{i-j}) \bmod 2$.
  (b) Ist $d = 1$, dann: $T(t) := C(t)$, $C(t) := C(t) + B(t) t^{i-m}$.
  Ist $L \leq i/2$, setze $L := i + 1 - L$, $m := i$, $B(t) := T(t)$.
  (c) $i := i + 1$.
  (d) (L ist jetzt die lineare Komplexität von $s_i = s_0, s_1, \ldots, s_{i-1}$, $(L, C(t))$ ein erzeugendes Schieberegister.)
(3) L ist nun die lineare Komplexität, $(L, C(t))$ ein erzeugendes Schieberegister.

Der Algorithmus läßt sich leicht in ein Maple-Programm üebersetzen:

```
linear_complexity:=proc()
  n:=nops(args[1]);
  s:=array(0..n-1);
  for j from 0 to n-1 do s[j]:=args[1][j+1]; od;
  C:=1; L:=0; m:=-1; B:=1; i:=0;
  while i<n do
    d:=s[i];
    for j from 1 to L do
      d:=(d+coeff(C,t,j)*s[i-j]) mod 2;
    od;
    if d=1 then
      T:=C; C:=(C+expand(B*t^(i-m))) mod 2;
      if L<=i/2 then
        L:=i+1-L; m:=i; B:=T;
      fi;
    fi;
    i:=i+1;
    printf("L(s^%d)=%d und C(t)=%a\n",i,L,C);
```

```
  od;
  [L,C];
end;
```

Beispiel:

```
> linear_complexity([1,0,0,1,1,0,1,1,1]);
L(s^1)=1 und C(t)=1+t
L(s^2)=1 und C(t)=1
L(s^3)=1 und C(t)=1
L(s^4)=3 und C(t)=1+t^3
L(s^5)=3 und C(t)=1+t^3+t
L(s^6)=3 und C(t)=1+t^3+t+t^2
L(s^7)=4 und C(t)=1+t+t^2
L(s^8)=4 und C(t)=1+t+t^2
L(s^9)=5 und C(t)=1+t+t^5+t^3+t^4
                                      5    3    4
                          [5, 1 + t + t  + t  + t ]
```

Das Beispiel zeigt auch, daß ein s^n erzeugendes Schieberegister nicht eindeutig bestimmt ist. So wird die Folge $s^4 = 1, 0, 0, 1$ von $(3, 1 + t^3)$, $(3, 1 + t + t^3)$ oder $(3, 1 + t + t^2 + t^3)$ erzeugt.

Der folgende Satz gibt eine einfache Charakterisierung, wann ein erzeugendes Schieberegister einer endlichen binären Folge s^n eindeutig bestimmt ist.

SATZ. *Sei s^n eine binäre Folge der Länge n mit linearer Komplexität L.*

(1) *Ist $n \geq 2L$, so gibt es genau ein Schieberegister der Länge L, das s^n erzeugt.*
(2) *Ist $n < 2L$, so gibt es mehrere Schieberegister der Länge L, die s^n erzeugen.*

Beweis: Sei $(L, C(t))$ ein die Folge s^n erzeugendes Schieberegister und A die zugehörige Übergangsmatrix. Wir schreiben

$$a_i = (s_{i+L-1}, s_{i+L-2}, \ldots, s_{i+1}, s_i) \text{ für } 0 \leq i \leq n - L.$$

Die Matrix A erfüllt die Gleichungen

$$a_{i+1} = a_i A \text{ für } 0 \leq i \leq n - L - 1.$$

Um andere Schieberegister der Länge L zu finden, die s^n erzeugen, brauchen wir nur Übergangsmatrizen B zu finden mit

$$a_{i+1} = a_i B \text{ für } 0 \leq i \leq n - L - 1.$$

Wir können ansetzen

$$B = A + \begin{pmatrix} e_1 & 0 & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ e_L & 0 & \ldots & 0 \end{pmatrix}$$

und erhalten dann mit $e = (e_1, \ldots, e_n)$ die Bedingungen

$$a_0 e^t = a_1 e^t = \cdots = a_{n-L-1} e^t = 0.$$

Man hat also $n - L$ lineare Gleichungen mit L Unbekannten $e_1, \ldots, e_L$.
Fall $n < 2L$: Dann hat man $\leq L - 1$ lineare Gleichungen mit L Unbekannten. Es gibt also sicher eine nichttriviale Lösung, was die Behauptung in diesem Fall zeigt.
Fall $n \geq 2L$: Wir werden zeigen, daß schon die L Vektoren $a_0, \ldots, a_{L-1}$ linear unabhängig sind. Dann ist klar, daß die L Gleichungen $a_i e^t = 0$, $0 \leq i \leq L - 1$ nur die triviale Lösung haben, was die Eindeutigkeit des Schieberegisters beweist. Wir machen einen Widerspruchsbeweis und nehmen an, daß $a_0, a_1, \ldots, a_{L-1}$ linear abhängig sind. Dann gibt es ein $m \leq L - 1$ und $d_i \in \mathbf{F}_2$ mit

$$a_m = d_1 a_{m-1} + d_2 a_{m-2} + \cdots + d_m a_0,$$

was auch als

$$sA^m = d_1 sA^{m-1} + d_2 sA^{m-2} + \cdots + d_m s$$

geschrieben werden kann. Multiplikation mit A^{j-m} für $j \geq m$ liefert

$$sA^j = d_1 sA^{j-1} + d_2 sA^{j-2} + \cdots + d_m sA^{m-j} \text{ für alle } j \geq m.$$

Da der letzte Eintrag von sA^l jetzt s_l ist, folgt

$$s_j = d_1 s_{j-1} + d_2 s_{j-2} + \cdots + d_m s_{j-m} \text{ für } j \geq m.$$

Die Folge läßt sich also durch ein Schieberegister der Länge $m \leq L-1$ beschrieben, was der Definition von L als linearer Komplexität von s^n widerspricht. Daher ist die Annahme falsch und die Behauptung folgt. ■

Bemerkung: Hat eine binäre Folge s lineare Komplexität L, so braucht man nur $2L$ Elemente $s_0, s_1, \ldots, s_{2L-1}$ zu kennen um mit ihnen und dem Berlekamp-Massey-Algorithmus das erzeugende Schieberegister zu bestimmen. Dies macht die kryptographische Verwendung von linearer Schieberegister unsicher.

Beispiel: Wir betrachten die endliche binäre Folge

```
 s  = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,
0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0
```

mit 1000 Folgenglieder. Mit dem Berlekamp-Massey-Algorithmus findet man schnell die lineare Komplexität und ein erzeugendes Schieberegister: $(100, t^{100}+t^{37}+1)$. Das Verbindungspolynom ist primitiv, also ist die Periodenlänge der zugehörigen Folge $2^{100}-1 = 1267650600228229401496703205375$.

Beispiel: Werner schickt an Heinrich eine verschlüsselte Nachricht, die wir abfangen. Wir vermuten, daß sie mit einem One-Time-Pad verschlüsselt wurde und daß die ersten 17 Zeichen 'Lieber Heinrich, ' lauten, was $17 \cdot 8 = 136$ Bits ergibt. Die ersten 136 Elemente der mutmaßlichen Schlüsselfolge berechnen sich zu:

```
ss := [1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0,
1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
0, 0, 0, 0, 1, 0, 1, 0];
```

Wir bestimmen mit dem Massey-Berlekamp-Algorithmus die lineare Komplexität: $(31, t^{31}+t^3+1)$. Daher vermuten wir, daß die 'Zufallsfolge' für den Schlüssel mit dem entsprechenden linearen Schieberegister der Länge 31 erzeugt wurde.

7. Kombination von linearen Schieberegistern

Auf der einen Seite sind lineare Schieberegister interessant, da sie einfach herzustellen sind. Auf der anderen Seite lassen sie sich aber bei Kenntnis einiger Folgenglieder mit dem Massey-Berlekamp-Algorithmus leicht rekonstruieren, was sie für kryptographische Zwecke untauglich macht.
Eine Möglichkeit besteht nun darin, mehrere lineare Schieberegister (nichtlinear) zu kombinieren.

Der Shrinking-Generator Gegeben sind zwei lineare Schieberegister R_1 und R_2. Die folgenden Schritte werden wiederholt, bis die Ausgabefolge gewünschte Länge hat.

(1) R_1 und R_2 werden getaktet (weitergeschoben).
(2) Gibt R_1 das Bit 1 aus, ist das Ausgabebit von R_2 das Ausgabebit.
(3) Gibt R_1 das Bit 0 aus, wird nichts ausgegeben.

Beispiel: Wir wählen $R_1 = (2, 1+t+t^2)$ und $R_2 = (3, 1+t+t^3)$ und die Anfängszustände $(0,1)$ bzw. $(0,0,1)$. Die Ausgabefolge von R_1 hat Periode $0,1,1$ (Periodenlänge 3), die Ausgabefolge von R_2 hat Periode $0,0,1,1,1,0,1$ (Periodenlänge 7). In der folgenden Tabelle stehen in den ersten beiden Zeilen die Ausgabefolgen von R_1 und R_2, in der dritten Zeile die Ausgabe des Shrinking-Generators:

0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1
0	0	1	1	1	0	1	0	0	1	1	1	0	1	0	0	1	1	1	0	1	0	0	1	1	1	0	1	0	0
	0	1		1	0		0	0		1	1		1	0		1	1		0	1		0	1		1	0		0	0

Dies ergibt die Folge

$$0,1,1,0,0,0,1,1,1,0,1,1,0,1,0,1,1,0,0,0,1,1,1,0,1,1,0,1,0,1,1,0,0,0,1,1,1,0,1,1,0,1,\ldots$$

Die lineare Komplexität der Folge ist 6 mit dem Verbindungspolynom $1+t^4+t^6$ und Periode 14.

Der Shrinking-Generator wurde 1993 vorgeschlagen. Man kann folgenden Satz beweisen:

SATZ. *Seien $R_1 = (L_1, C_1(t))$ und $R_2 = (L_2, C_2(t))$ lineare Schieberegister mit primitiven Polynomen $C_1(t)$ und $C_2(t)$. Sei x die Ausgabefolge des Shrinking-Generators.*

(1) *Ist* $\mathrm{ggT}(L_1, L_2) = 1$, *dann hat x Periode* $(2^{L_2}-1)2^{L_1-1}$.
(2) *Für die lineare Komplexität $L(x)$ von x gilt:*

$$L_2 \cdot 2^{L_1-2} < L(x) \leq L_2 \cdot 2^{L_1-1}.$$

Wir geben eine kryptographische Anwendung:

Stromchiffrierung mit dem Shrinking-Generator:

(1) Man wählt Schieberegister $R_1 = (L_1, C_1(t))$ und $R_2 = (L_2, C_2(t))$.
(2) Als Schlüssel wählt man Anfangszustände $s_1 \in \mathbf{F}_2^{L_1}$ und $s_2 \in \mathbf{F}_2^{L_2}$ der Schieberegister R_1 und R_2.

(3) Die zu verschlüsselnde Nachricht wird in eine Bitfolge $m = m_1, m_2, m_3, \ldots$ zerlegt. Der Shrinking-Generator mit den Schieberegistern R_1 und R_2 sowie den Anfangszuständen s_1 und s_2 liefert eine Bitfolge $k = k_1, k_2, k_3, \ldots$. Die verschlüsselte Nachricht ist die Bitfolge $c_1, c_2, c_3, \ldots$ mit $c_i = m_i + k_i (\mathrm{mod} 2)$.
(4) Die Entschlüsselung funktioniert genau wie die Verschlüsselung (mit dem gleichen Schlüssel).

Bemerkungen:

(1) In der angegebenen Form hat obige Stromchiffrierung $2^{L_1+L_2}$ verschiedene Schlüssel.
(2) Zur Erhöhung der Sicherheit kann auch noch die Verbindungspolynome $C_1(t)$ und $C_2(t)$ geheimhalten, also als Teil des Schlüssels wählen. Weiter sollte man dann $\mathrm{ggT}(L_1, L_2) = 1$ und primitive Polynome $C_1(t)$, $C_2(t)$ wählen. Bei Wahl von $L_1, L_2 \approx 64$ scheint das Verfahren dann heutzutage sicher zu sein.

Beispiel: Das nachfolgende Programm streamsh.c realisiert die Stromchiffrierung mit dem Shrinking-Generator. Allerdings sind bereits Schieberegister der Länge 32 gewählt, die Schlüssel sind damit 64 Bits lang. Die lineare Komplexität des Schlüsselstroms sollte dann zwischen $2^{35} \approx 3 \cdot 10^{10}$ und $2^{36} \approx 7 \cdot 10^{10}$ liegen.

```
/* streamshrink.c - Version vom 19.5.2001
   Stromchiffrierung mit dem Shrinking-Generator.
   Der Einfachkeit halber koennen nur lineare Schieberegister
   mit Laenge <=32 benutzt werden.
   Maximalwert unsigned: 2^32-1 = 4294967295 = 0.4*10^10
   Uebersetzung mit gcc streamshrink.c -o streamshrink */

#include <stdio.h>

/* Ausgabe einer Zahl x in Binaerdarstellung mit den n letzten Zeichen
 */
void bitausgabe( unsigned x, int n)
{
  int a[32], i;
  for (i=0;i<n;i++) { a[i]=x&1; x>>=1; }
  for (i=n-1;i>=0;i--) printf("%d",a[i]);
}

/* Summe der Bits in x mod 2 */
int subi2( unsigned x)
{
  int az=0;
  while (x!=0)
  {
    x&=(x-1);   /* x&=(x-1) loescht das am weitesten rechts stehende */
                /* 1-Bit in x */
    az^=1;      /* az^1 = (az+1) mod 2 fuer az=0,1 */
  }
  return az;
}

/* Takten eines linearen Schieberegisters [s_(n-1),...,s_1,s_0]
   der Laenge n<=32 mit Verbindungspolynom C(t)=1+c_1*t+...+c_n*t^n.
   Dabei ist c=c_1*2^(n-1)+c_2*2^(n-2)+...c_(n-1)*2+c_n und
   zustand=s_(n-1)*2^(n-1)+...s_1*2+s_0 */
unsigned clock( unsigned zustand, unsigned c, int n)
{
```

```
  if (subi2(zustand&c)==1) { zustand>>=1; zustand|=(1<<(n-1)); }
  else zustand>>=1;
  return zustand;
}

/* Der Shrinking-Generator */
int shrink( unsigned *r1, unsigned c1, int n1,
            unsigned *r2, unsigned c2, int n2)
{
  for (;;)
  {
    *r1=clock(*r1,c1,n1); *r2=clock(*r2,c2,n2);
    if ((*r1)&1==1) return (*r2)&1;
  }
}

main( int argc, char *argv[])
{
  int n1, n2, m, k, c, i, s;
  unsigned c1, c2, r1, r2;
  FILE *ein, *aus;

  printf("Stromchiffrierung mit dem Shrinking-Generator\n");
  if (argc!=2)
  {
    printf("Aufruf: 'streamsh datei'\n");
    return;
  }

  /* Die benutzten zwei Schieberegister:
     C_1=1+t^2+t^6+t^7+t^32 und  C_2=1+t+t^27+t^28+t^32 */
  n1=32; c1=(1<<(32-2))|(1<<(32-6))|(1<<(32-7))|1;
  n2=32; c2=(1<<(32-1))|(1<<(32-27))|(1<<(32-28))|1;
  printf("Die benutzten Schieberegister:\n");
  printf("R1: "); bitausgabe(c1,n1); printf("\n");
  printf("R2: "); bitausgabe(c2,n2); printf("\n");

  /* Schluessel */
  printf("Schluessel: 8-stellige Hexadezimalzahlen r1 und r2\n");
  printf("r1="); scanf("%x",&r1);
  printf("r2="); scanf("%x",&r2);

  printf("Schluessel: ");
  bitausgabe(r1,n1); printf(" "); bitausgabe(r2,n2); printf("\n");

  ein=fopen(argv[1],"rb");
  strcat(argv[1],".str"); aus=fopen(argv[1],"wb");

  while ((m=getc(ein))!=EOF)
  {
    k=0; for (i=0;i<8;i++)
    {
      k<<=1;
      k|=shrink(&r1,c1,n1,&r2,c2,n2);
```

```
    }
    c=m^k;
    putc(c,aus);
  }
  printf("Ausgabefile: %s\n",argv[1]);
}
```

Wir geben noch ein weiteres Beispiel für die Kombination von linearen Schieberegistern:

Der alternierende Stop-and-Go-Generator: Wir haben 3 Schieberegister R_1, R_2, R_3. Die folgenden Schritte werden wiederholt, bis man eine Ausgabefolge gewünschter Länge erhalten hat.

(1) R_1 wird weitergeschoben.
(2) Ist die Ausgabe von R_1 1, dann wird R_2 weitergeschoben, R_3 aber nicht.
(3) Ist die Ausgabe von R_1 0, dann wird R_3 weitergeschoben, R_2 aber nicht.
(4) Ausgegeben wird die Summe der Ausgabebits von R_2 und R_3 (modulo 2).

Beispiel: Wir wählen $R_1 = (4, t^4 + t + 1)$ mit dem Anfangszustand $(1, 1, 1, 1)$. Die Ausgabefolge hat Periodenlänge 15 mit der Periode $1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0$. Weiter wählen wir $R_2 = (2, t^2 + t + 1)$, $R_3 = (3, t^3 + t + 1)$ mit Anfangszuständen $(0, 1)$ bzw. $(0, 0, 1)$ und den Perioden $1, 0, 1$ und $1, 0, 0, 1, 1, 1, 0$. Man erhält:

1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0
1	0	1	1		0		1	1			0				1	1	0	1		1		0	1			1			
1				0		0			1	1		1	0	1					0		0			1	1		1	0	1
0	1	0	0	1	0	0	1	1	0	0	1	1	0	1	0	0	1	0	1	1	1	0	1	0	0	0	0	1	0

Die lineare Komplexität der Ausgabefolge ist $(45, t^{45} + 1)$, die Periodenlänge ist 45.

8. Kryptographisch sichere Pseudozufallsgeneratoren

Bei den unten angegebenen sogenannten 'kryptographisch sicheren Pseudozufallsgeneratoren' liegt jedesmal eine große natürliche Zahl n zugrunde, deren Primfaktorzerlegung man aber praktisch nicht bestimmen können sollte. Hätte man ein Verfahren, mit dem man aus der Kenntnis von $z_1, z_2, \ldots, z_l$ das nächste Bit z_{l+1} mit einer Wahrscheinlichkeit $> \frac{1}{2}$ berechnen könnte, dann könnte man auch n faktorisieren, was der Wahl von n widerspricht.

Der RSA-Pseudozufallszahlengenerator:

(1) Man wähle geheim Primzahlen p, q, setze $n = pq$, wähle $1 < e < \phi(n)$ mit $\mathrm{ggT}(e, \phi(n)) = 1$. ($n$ sollte praktisch nicht zu faktorisieren sein.)
(2) Man wähle eine zufällige Zahl x_0 mit $1 < x_0 < n - 1$ (seed).
(3) Man berechne rekursiv:
  (a) $x_i \equiv x_{i-1}^e \bmod n$,
  (b) $z_i \equiv x_i \bmod 2$.
(4) Die Ausgabefolge ist $z_1, z_2, z_3, \ldots, z_l$.

Beispiel: Wir wählen $n = 2911 = 41 \cdot 71$, als Startwert $x_0 = 2$ und erhalten periodische Folgen mit Periodenlänge 12:

$$\begin{aligned} x &= 2, 8, 512, 251, 699, 1935, 1004, 1071, 1068, 2796, 1578, 333, 2, 8, \ldots \\ z &= 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, \ldots \end{aligned}$$

Mit dem Berlekamp-Massey-Algorithmus erhalten wir die lineare Komplexität 12 und das erzeugende Schieberegister $R = (12, 1 + t^{12})$.

Der Blum-Blum-Shub-Pseudozufallszahlenbitgenerator:

(1) Man wähle geheim Primzahlen p, q mit $p \equiv q \equiv 3 \bmod 4$ und setze $n = pq$. (n sollte praktisch nicht zu faktorisieren sein.)

(2) Man wähle eine zufällige Zahl r (seed) mit $1 < r < n-1$ und $\mathrm{ggT}(r,n) = 1$. Man setze $x_0 \equiv r^2 \bmod n$.
(3) Man berechne rekursiv:
  (a) $x_i \equiv x_{i-1}^2 \bmod n$,
  (b) $z_i \equiv x_i \bmod 2$, d.h. z_i ist das niedrigste Bit von x_i.
(4) Die Ausgabefolge ist $z_1, z_2, z_3, z_4, \ldots$.

Wir konstruieren mit diesem Pseudozufallszahlengenerator ein Public-Key-Kryptosystem:

Blum-Goldwasser-Verschlüsselung:

(1) Jeder Teilnehmer A wählt sich zwei Primzahlen $p \equiv 3 \bmod 4$ und $q \equiv 3 \bmod 4$, so daß für einen Außenstehenden $n = pq$ praktisch nicht zu faktorisieren ist. Der öffentliche Schlüssel von A ist n, der private Schlüssel (p,q).
(2) B will an A verschlüsselt eine Nachricht m schicken.
  (a) B besorgt sich den öffentlichen Schlüssel n von A.
  (b) B berechnet $h = \lfloor \log_2 \lfloor \log_2 n \rfloor \rfloor$ (oder wählt z.B. $h = 8$, wenn entsprechend vereinbart) und teilt die Nachricht in Blöcke von je h Bits ein: $m = m_1 m_2 m_3 \ldots m_t$, wo also m_i aus h Bits besteht.
  (c) B wählt zufällig r mit $\mathrm{ggT}(r,n) = 1$ und setzt $x_0 \equiv r^2 \bmod n$.
  (d) B führt nacheinander für $1 \le i \le t$ folgende Schritte aus:
    (i) $x_i \equiv x_{i-1}^2 \bmod n$,
    (ii) k_i seien die h niedrigsten Bits von x_i,
    (iii) $c_i = m_i$ XOR k_i, oder anders geschrieben $c_i = m_i + k_i \bmod 2$, wo $+$ für komponentenweise Addition steht.
  (e) $x_{t+1} \equiv x_t^2 \bmod n$.
  (f) B schickt als verschlüsselten Text
$$(c_1, c_2, \ldots, c_t, x_{t+1})$$
  an A.
(3) Wie entschlüsselt A die verschlüsselte Nachricht $(c_1, c_2, \ldots, c_t, x_{t+1})$?
  (a) A berechnet nacheinander
$$d_p \equiv (\frac{p+1}{4})^{t+1} \bmod p-1, \quad d_q \equiv (\frac{q+1}{4})^{t+1} \bmod q-1,$$
$$u_p \equiv x_{t+1}^{d_p} \bmod p, \quad u_q \equiv x_{t+1}^{d_q} \bmod q.$$
  Mit dem chinesischen Restsatz berechnet A eine natürliche Zahl u mit
$$u \equiv u_p \bmod p, \quad u \equiv u_q \bmod q.$$
  (b) Nun sei $x_0 \equiv u \bmod n$ und für $1 \le i \le t$ dann

$x_i \equiv x_{i-1}^2 \bmod n$, k_i seinen die h niedrigsten Bits von x_i, $m_i = k_i + c_i = k_i$ XOR c_i.

  Die Ausgangsnachricht ist dann $(m_1, m_2, \ldots, m_t)$.

Beweis für die Richtigkeit der Entschlüsselung: Wir müssen nur zeigen, daß $x_0 \equiv u \bmod n$ gilt, denn dann hat man bei der Entschlüsselung die gleichen Folgen x_i und k_i und die Behauptung folgt aus
$$c_i \text{ XOR } k_i = (m_i \text{ XOR } k_i) \text{ XOR } k_i = m_i \text{ XOR } (k_i \text{ XOR } k_i) = m_i.$$
Mit der Notation $x_{-1} \equiv r \bmod n$ gilt $x_i \equiv x_{i-1}^2 \bmod p$ für $0 \le i \le t+1$ und somit für $1 \le i \le t$ modulo p:
$$x_i^{\frac{p+1}{4}} \equiv x_{i-1}^{\frac{p+1}{2}} \equiv x_{i-1} \cdot x_{i-1}^{\frac{p-1}{2}} \equiv x_{i-1} \cdot x_{i-2}^{p-1} \equiv x_{i-1} \bmod p,$$
was durch Induktion sofort
$$x_{t+1}^{(\frac{p+1}{4})^{t+1}} \equiv x_0 \bmod p \quad \text{und analog} \quad x_{t+1}^{(\frac{q+1}{4})^{t+1}} \equiv x_0 \bmod q$$

liefert. Man erhält wieder mit Verwendung des kleines Satzes von Fermat ($x_{t+1}^{p-1} \equiv 1 \bmod p$ und $x_{t+1}^{q-1} \equiv 1 \bmod q$)

$$\begin{aligned} u &\equiv u_p \equiv x_{t+1}^{d_p} \equiv x_{t+1}^{(\frac{p+1}{4})^{t+1}} \equiv x_0 \bmod p, \\ u &\equiv u_q \equiv x_{t+1}^{d_q} \equiv x_{t+1}^{(\frac{q+1}{4})^{t+1}} \equiv x_0 \bmod q, \end{aligned}$$

was sofort $x_0 \equiv u \bmod n$ und damit die Behauptung liefert. ■

Beispiel: Wir wählen als öffentlichen Schlüssel $n = 4295622677$ und bilden jeweils 5-Bit-Blöcke. Wir identifizieren die Kleinbuchstaben mit den Zahlen zwischen 0 und 25 und stellen sie als 5-Bit-Zahlen dar. Wir wollen 'michael' verschlüsseln. Dazu wählen wir zufällig $r = 3041882126$, bilden $x_0 \equiv r^2 \bmod n$, also $x_0 = 2182455525$ und dann rekursiv $x_i \equiv x_{i-1}^2 \bmod n$, $k_i \equiv x_i \bmod 32$ und erhalten

i	x_i	k_i	Text	m_i	m_i binär	verschlüsselt
1	130273928	01000	m	12	01100	00100
2	4193086566	00110	i	8	01000	01110
3	1575234542	01110	c	2	00010	01100
4	314505964	01100	h	7	00111	01011
5	3877584104	01000	a	0	00000	01000
6	4225468852	10100	e	4	00100	10000
7	4122374247	00111	l	11	01011	01100

$x_8 = 3680951627$. Die verschlüsselte Nachricht ist also

$$(00100011100110001011010001000001100, 3680951627).$$

Zur Entschlüsselung: Der private Schlüssel sind die Primfaktoren von n: $p = 65539$, $q = 65543$. Wir bilden

$$\begin{gathered} (\frac{p+1}{4})^8 \equiv 32641 \bmod p-1, \quad (\frac{q+1}{4})^8 \equiv 43652 \bmod q-1, \\ u = x_8^{32641} \equiv 6825 \bmod p, \quad v = x_8^{43652} \equiv 4711 \bmod q, \\ 2182455525 \equiv u \bmod p, 2182455525 \equiv v \bmod q, \end{gathered}$$

was tatsächlich x_0 ergibt.

Bemerkung: Die Sicherheit des Blum-Goldwasser-Kryptosystems beruht wesentlich darauf, daß man modulo $n = pq$ keine Quadratwurzeln ziehen kann ohne die Faktorzerlegung von n zu kennen. Ansonsten könnte man nämlich rekursiv mit der Gleichung $x_i \equiv x_{i-1}^2 \bmod n$ aus x_{t+1} den Startwert x_0 berechnen. Das folgende Lemma behandelt das Wurzelziehen modulo n.

LEMMA. *Seien p und q verschiedene Primzahlen und $n = pq$. Sei $b \in \mathbf{Z}$ mit $\mathrm{ggT}(b, n) = 1$ und $a \equiv b^2 \bmod n$. Mit dem chinesischen Restsatz findet man (modulo n eindeutig bestimmte) Zahlen b_1, b_2, b_3, b_4 mit*

$$\begin{aligned} b_1 &\equiv b \bmod p, & b_1 &\equiv b \bmod q, \\ b_2 &\equiv b \bmod p, & b_2 &\equiv -b \bmod q, \\ b_3 &\equiv -b \bmod p, & b_3 &\equiv b \bmod q, \\ b_4 &\equiv -b \bmod p, & b_4 &\equiv -b \bmod q. \end{aligned}$$

Dann hat die Gleichung $x^2 \equiv a \bmod n$ genau 4 Lösungen (modulo n), nämlich $x = b_1, b_2, b_3, b_4$, und

$$\mathrm{ggT}(b_1 - b_1, n) = n, \quad \mathrm{ggT}(b_1 - b_2, n) = p, \quad \mathrm{ggT}(b_1 - b_3, n) = q, \quad \mathrm{ggT}(b_1 - b_4, n) = 1.$$

Beweis: Da $\mathbf{Z}/(p)$ und $\mathbf{Z}/(q)$ Körper sind, hat die Gleichung $x^2 \equiv b^2 \bmod p$ genau die Lösungen $x \equiv \pm b \bmod p$, analog die Gleichung $x^2 \equiv b^2 \bmod q$ genau die Lösungen $x \equiv \pm b \bmod q$. Mit dem chinesischen Restsatz ergeben sich dann folgende Äquivalenzen:

$$\begin{aligned} x^2 \equiv a \bmod n \quad &\Longleftrightarrow \quad x^2 \equiv b^2 \bmod n \quad \Longleftrightarrow \quad x^2 \equiv b^2 \bmod p \text{ und } x^2 \equiv b^2 \bmod q \quad \Longleftrightarrow \\ &\Longleftrightarrow \quad x \equiv \pm b \bmod p \text{ und } x \equiv \pm b \bmod q \quad \Longleftrightarrow \\ &\Longleftrightarrow \quad x \equiv b_i \bmod n \text{ für ein } i \in \{1, 2, 3, 4\}, \end{aligned}$$

was die erste Behauptung beweist. Nun gilt:

$$b_1 - b_2 \equiv b - b \equiv 0 \bmod p \quad \text{und} \quad b_1 - b_2 \equiv -b - b \equiv -2b \not\equiv 0 \bmod q,$$

was $p|b_1 - b_2$ und $q \not| b_1 - b_2$, also $\mathrm{ggT}(b_1 - b_2, n) = p$ liefert. Die anderen Behauptungen ergeben sich ganz genauso. ■

Bemerkungen:

(1) Aus dem Lemma folgt direkt: Hat man ein Verfahren, um alle Lösungen einer Gleichung $x^2 \equiv a \bmod n$ zu bestimmen, kann man n faktorisieren.

(2) Angenommen, wir haben ein Verfahren, um zu einem Quadrat modulo n eine Wurzel zu bestimmen, also eine Funktion W mit $W(b^2)^2 \equiv b^2 \bmod n$ für alle b mit $\mathrm{ggT}(b, n) = 1$. Dann wählen wir zufällig b mit $\mathrm{ggT}(b, n) = 1$ und berechnen $W(b^2)$ und damit

$$\mathrm{ggT}(b - W(b^2), n).$$

Nach dem letzten Lemma ist die Wahrscheinlichkeit $\frac{1}{2}$, daß dieser ggT ein nichttrivialer Teiler von n ist. Wir wählen solange b's, bis wir n faktorisiert haben.

Das nachfolgende Programm blumgold.c realisiert das Blum-Goldwasser-Verschlüsselungssystem.

```
/* blumgold.c - Version vom 19.5.2001
   Blum-Goldwasser-Public-Key-Verschluesselungsverfahren
   Von der Zufallsfolge x_i=x_(i-1)^2 mod n werden jeweils die letzten
   8 Bits k_i=x_i mod 256 zur Verschluesselung benutzt.
   Uebersetzung mit gmp blumgold.c -o blumgold -lgmp */

#include <stdio.h>
#include <gmp.h>

main( int argc, char *argv[])
{
  int len, fall, i, m, k, c, bytn, t;
  mpz_t p, q, n, x, r, dp, dq, up, uq, a, b;
  char name[1000], benutzer[30];
  FILE *ein, *aus;

  if (argc!=2)
  {
    printf("Aufruf: 'blumgold datei' (Verschluesselung)\n");
    printf("        'blumgold datei.bgw' (Entschluesselung)\n");
    return;
  }

  /* Bei Dateiendung '.bgw' Entschluesselung, sonst Verschluesselung */
  len=strlen(argv[1]);
  fall=( argv[1][len-4]=='.' && argv[1][len-3]=='b' &&
         argv[1][len-2]=='g' && argv[1][len-1]=='w'   ) ? 2 : 1;

  switch(fall)
  {
    case 1: /* Verschluesselung */

      printf("Blum-Goldwasser-Verschluesselung\n");

      if ((ein=fopen("blumgold.key.pub","rb"))==NULL)
      {
        printf("Schluesselfile blumgold.key.pub existiert nicht!\n");
```

```
    return;
  }
  printf("Zielperson: "); scanf("%s",benutzer);

  while (strcmp(benutzer,name)!=0) fscanf(ein,"%s",name);
  mpz_init(n); mpz_inp_str(n,ein,10); fclose(ein);
  printf("Oeffentlicher Schluessel von %s:\n",benutzer);
  printf("n="); mpz_out_str(stdout,10,n); printf("\n");

  ein=fopen(argv[1],"rb");
  strcat(argv[1],".bgw"); aus=fopen(argv[1],"wb");

  /* r wird eingegeben, dann ist x0 = r^2 mod n */
  printf("Zufallszahl: ");
  mpz_init(x); mpz_inp_str(x,stdin,10);
  mpz_mul(x,x,x); mpz_mod(x,x,n);

  while ((m=getc(ein))!=EOF)
  {
    mpz_mul(x,x,x); mpz_mod(x,x,n); /* x=x^2 mod n */
    k=mpz_fdiv_ui(x,256); /* k sind die letzten 8 Bits von x */
    c=m^k; /* Verschluesselung c=m+k mod 2 komponentenweise */
    putc(c,aus);
  }

  /* x_(t+1) muss berechnet und gespeichert werden */
  mpz_mul(x,x,x); mpz_mod(x,x,n);

  bytn=0; /* bytn ist die Anzahl der Bytes von n */
  while (mpz_sgn(n)!=0) { bytn++; mpz_fdiv_q_2exp(n,n,8); }

  /* x_(t+1) wird in Bytes zerlegt und in die Datei geschrieben */
  mpz_init(r);
  for (i=bytn-1;i>=0;i--)
  {
    mpz_fdiv_q_2exp(r,x,8*i);
    putc(mpz_fdiv_ui(r,256),aus);
  }

  printf("Verschluesselte Datei: %s\n",argv[1]);
  break;

case 2: /* Entschluesselung */

  printf("Blum-Goldwasser-Entschluesselung\n");

  /* Ansatz: name = Eingabefile ohne Endung ".bgw" */
  for (i=0;i<len-4;i++) name[i]=argv[1][i]; name[len-4]='\0';
  if ((aus=fopen(name,"rb"))!=NULL)
  {
    printf("Die Datei %s existiert bereits.\n",name); fclose(aus);
    printf("Neuer Dateiname: ",name); scanf("%s",name);
  }
  aus=fopen(name,"wb");
```

```
if ((ein=fopen("blumgold.key.priv","rb"))==NULL)
{
   printf("Schluesselfile blumgold.key.priv existiert nicht!\n");
   return;
}
mpz_init(p); mpz_inp_str(p,ein,10);
mpz_init(q); mpz_inp_str(q,ein,10);
fclose(ein);

printf("Privater Schluessel:\n");
printf("p="); mpz_out_str(stdout,10,p); printf("\n");
printf("q="); mpz_out_str(stdout,10,q); printf("\n");

/* n=pq, bytn ist die Byte-Anzahl von n */
mpz_init(n); mpz_mul(n,p,q); bytn=0;
while (mpz_sgn(n)!=0) { bytn++; mpz_fdiv_q_2exp(n,n,8); }
mpz_mul(n,p,q);

ein=fopen(argv[1],"rb");
t=0; while (getc(ein)!=EOF) t++;
t-=bytn; /* t = Anzahl der verschluesselten Bytes */

/* Die letzten bytn Bytes ergeben x_(t+1) */
fseek(ein,-bytn,SEEK_CUR);
mpz_init(x);
for (i=0;i<bytn;i++)
{
   mpz_mul_ui(x,x,256); mpz_add_ui(x,x,getc(ein));
}

/* Nun wird x0 aus x_(t+1), p, q und t bestimmt:
   dp=((p+1)/4)^(t+1) mod (p-1), up=x_(t+1)^dp mod p
   dq=((q+1)/4)^(t+1) mod (q-1), uq=x_(t+1)^dq mod q
   Gesucht wird x0 mit x0=up mod p und x0=uq mod q.
   Dazu werden a,b mit ap+bq=1 bestimmt.
   Dann ist x0=uq*a*p+up*b*q */
mpz_init(r); mpz_init(a); mpz_init(b);
mpz_init(dp); mpz_init(dq); mpz_init(up); mpz_init(uq);

mpz_add_ui(dp,p,1); mpz_fdiv_q_2exp(dp,dp,2);
mpz_sub_ui(r,p,1); mpz_powm_ui(dp,dp,t+1,r); mpz_powm(up,x,dp,p);

mpz_add_ui(dq,q,1); mpz_fdiv_q_2exp(dq,dq,2);
mpz_sub_ui(r,q,1); mpz_powm_ui(dq,dq,t+1,r); mpz_powm(uq,x,dq,q);

mpz_gcdext(r,a,b,p,q); mpz_mul(r,up,b); mpz_mul(r,r,q);
mpz_mul(x,uq,a); mpz_mul(x,x,p); mpz_add(x,x,r); mpz_mod(x,x,n);

/* Nun muss nur der Verschluesselungsvorgang wiederholt werden */
rewind(ein);
for (i=1;i<=t;i++)
{
  mpz_mul(x,x,x); mpz_mod(x,x,n); /* x_i=x_(i-1)^2 mod n */
```

```
        k=mpz_fdiv_ui(x,256); c=getc(ein); m=c^k; putc(m,aus);
      }

      printf("Entschluesselte Datei: %s\n",name);
      break;
  }
}
```

9. Die Stromchiffrierung RC4

RC4 ist eine Stromchiffrierung mit variabler Schlüssellänge, die 1987 von Rivest für RSA Data Security (für kommerzielle Zwecke) entwickelt wurde und zunächst geheimgehalten wurde. Inzwischen ist aber der einfache Algorithmus allgemein bekannt.

RC4-Chiffrierung:

(1) Ein Schlüssel besteht aus 256 Bytes K_i, $0 \le i \le 255$, also $0 \le K_i \le 255$. (Gibt man weniger Bytes an, werden die vorhandenen periodisch wiederholt.)
(2) Man hat 256 Bytes S_i, $0 \le i \le 255$, und setzt zunächst $S_i = i$, $0 \le i \le 255$.
(3) Man setzt $j = 0$ und führt für $i = 0, \ldots, 255$ folgende Schritte aus:
  (a) $j := (j + S_i + K_i) \bmod 256$,
  (b) vertausche S_i und S_j.
(4) Nun wird die Eingabedatei wie folgt verschlüsselt.
  (a) $i := (i + 1) \bmod 256$,
  (b) $j := (j + S_i) \bmod 256$,
  (c) vertausche S_i und S_j,
  (d) $t := (S_i + S_j) \bmod 256$,
  (e) $k = S_t$.
  (f) Sei m das nächste Byte der Eingabedatei.
  (g) Das verschlüsselte Byte $c = m$ XOR k wird in die Ausgabedatei geschrieben.

```
#include <stdio.h>

main( int argc, char *argv[])
{
  int i, j, S[256], K[256], t, bz, l, m, k, c;
  FILE *ein, *aus;

  ein=fopen(argv[1],"rb");
  strcat(argv[1],".rc4");
  aus=fopen(argv[1],"wb");

  /* Schluesseleingabe */
  printf("Wieviele Schluesselbytes sollen eingeben werden? ");
  scanf("%d",&bz);
  for (l=0;l<bz;l++)
  {
    printf("K[%d]=",l); scanf("%d",&K[l]);
  }
  for (l=bz;l<256;l++) K[l]=K[l-bz];

  /* Initialisierung */
  for (l=0;l<256;l++) S[l]=l;

  j=0;
  for (i=0;i<256;i++)
```

```
  {
    j=(j+S[i]+K[i])%256;
    l=S[i]; S[i]=S[j]; S[j]=l;
  }

  while ((m=getc(ein))!=EOF)
  {
    i=(i+1)%256;
    j=(j+S[i])%256;
    l=S[i]; S[i]=S[j]; S[j]=l;
    t=(S[i]+S[j])%256;
    k=S[t];
    c=m^k;
    putc(c,aus);
  }
}
```

10. Wann ist eine 0-1-Folge zufällig? — Statistische Tests

Produziert ein Pseudozufallszahlengenerator eine 0-1-Folge $s_0, s_1, s_2, \ldots$, so wollen wir wissen, ob die Folge wie eine Zufallsfolge aussieht. Leider gibt es hier keine mathematischen Beweise, es gibt aber eine Reihe von statistischen Tests, bei denen bestimmte Eigenschaften getestet werden, die eine Zufallsfolge haben sollte. Besteht eine Zahlenfolge einen solchen Test nicht, kann man sie als Pseudozufallszahlenfolge ablehnen, da sie zu wenig 'zufällig' aussieht.

Beispiele: Gegeben sei eine 0-1-Folge $s = s_0, s_1, s_2, \ldots$. Wir betrachten die n ersten Folgenglieder $s_0, s_1, s_2, \ldots, s_{n-1}$, wobei wir uns n groß vorstellen.

(1) Sei n_0 die Anzahl der Nullen, n_1 die Anzahl der Einsen in $s_0, s_1, \ldots, s_{n-1}$. Es ist $n = n_0 + n_1$. Ist die Folge s eine Zufallsfolge, sollte $n_0 \approx n_1 \approx \frac{1}{2}n$ gelten. Wie weit darf aber in der Praxis n_1 von $\frac{1}{2}n$ abweichen, so daß wir die Folge noch als 'zufällig' akzeptieren?

(2) Wir teilen die Folge s in Blöcke $s_{4l}, s_{4l+1}, s_{4l+2}, s_{4l+3}$ der Länge 4 auf $(l = 0, 1, 2, \ldots, \lfloor \frac{n-4}{4} \rfloor)$. Für einen Block gibt es dann die 16 Möglichkeiten: 0000, 0001, 0010,..., 1111. Bei einer Zufallsfolge sollte jede der 16 Möglichkeiten ungefähr gleich oft realisiert werden. Praktisch stellt sich aber die Frage, welche Abweichungen wir zulassen, so daß wir die Folge (in diesem Punkt) noch als 'zufällig' akzeptieren.

Die in den Beispielen angesprochenen Probleme können mit dem sogenannten χ^2-Test behandelt werden.

χ^2-Test: Wir nehmen an, es gibt k mögliche Ausgänge eines Experiments mit (theoretischen) Wahrscheinlichkeiten $p_1, \ldots, p_k$, insbesondere $p_1 + \cdots + p_k = 1$. Wir machen n Versuche und erhalten die Häufigkeiten $n_1, \ldots, n_k$. Wir erwarten $n_i \approx np_i$. Wir setzen an

$$X = \sum_{i=1}^{k} \frac{(n_i - np_i)^2}{np_i}.$$

X folgt (für große n ungefähr) einer χ^2-Verteilung mit $\nu = k - 1$ Freiheitsgraden. Es gibt die Wahrscheinlichkeitsaussage

$$P(X > \chi^2_{\nu,\alpha}) = \alpha,$$

wo α eine Wahrscheinlichkeit bezeichnet und $\chi^2_{\nu,\alpha}$ einer Tabelle entnommen oder mit der Formel

$$\alpha = P(X > x) = \int_x^\infty \frac{1}{\Gamma(\frac{\nu}{2})2^{\frac{\nu}{2}}} t^{\frac{\nu}{2}-1} e^{-\frac{t}{2}} dt$$

berechnet werden kann.
Man kann die Formel für X noch etwas anders schreiben: Es gilt mit $\sum_{i=1}^k n_i = n$ und $\sum_{i=1}^k p_i = 1$

$$\begin{aligned} X &= \sum_{i=1}^k \frac{(n_i - np_i)^2}{np_i} = \sum_{i=1}^k \frac{n_i^2 - 2n_i np_i + n^2 p_i^2}{np_i} = \sum_{i=1}^k (\frac{n_i^2}{np_i} - 2n_i + np_i) = \sum_{i=1}^k \frac{n_i^2}{np_i} - 2n + n = \\ &= \frac{1}{n} \sum_{i=1}^k \frac{n_i^2}{p_i} - n \end{aligned}$$

Ist $p_1 = \cdots = p_k = \frac{1}{k}$, so erhält man

$$X = \frac{k}{n} \sum_{i=1}^k n_i^2 - n.$$

Beispiel: Wir haben n Folgenglieder einer 0-1-Folge. Wir finden n_0 Nullen, n_1 Einsen. Unter der Annahme, daß die Folge 'zufällig' ist, gilt $p_0 = p_1 = \frac{1}{2}$ und damit für die χ^2-Statistik:

$$X = \frac{2}{n}(n_0^2 + n_1^2) - n = \frac{2(n_0^2 + n_1^2) - (n_0 + n_1)^2}{n} = \frac{(n_0 - n_1)^2}{n}.$$

Die Anzahl der Freiheitsgrade ist 1. In einer Tabelle finden wir z.B. $\chi^2_{1,1\%} = 6.635$. In 99% aller Fälle wird also

$$\frac{(n_0 - n_1)^2}{n} < 6.635$$

gelten. Mit obiger Formel und Maple finden wir $P(X > 15) = 0.0001$, d.h. in 99.99% aller Fälle gilt

$$\frac{(n_0 - n_1)^2}{n} < 15.$$

Im folgenden geben wir Tests an, die für US-Regierungsbehörden entwickelt wurden (FIPS PUB: Federal Information Processing Standards Publication):

FIPS PUB 140-1 - statistischer Test für Zufälligkeit: Gegeben sei ein $n = 20000$-Bit-langes Stück einer 0-1-Folge.

(1) Der Monobit-Test: Die Anzahl n_1 der Einsen sollte im Bereich

$$9654 < n_1 < 10346$$

liegen.

(2) Der Poker-Test: Die Folge wird in 5000 Blöcke der Länge 4 aufgeteilt. 4 Bits $s_{4l}s_{4l+1}s_{4l+2}s_{4l+3}$ entsprechen (in der Binärdarstellung) einer Zahl zwischen 0 und 15, nämlich $i = 8s_{4l} + 4s_{4l+1} + 2s_{4l+2} + s_{4l+3}$. Sei $f(i)$ die Anzahl der i entsprechenden Blöcke, $0 \le i \le 15$. Nun bildet man

$$X = \frac{16}{5000} \sum_{i=0}^{15} f(i)^2 - 5000.$$

Der Test gilt als bestanden, wenn

$$1.03 < X < 57.4$$

gilt.

(3) Der Runs-Test: Ein Block ist eine Teilfolge aus lauter Einsen, wo davor und dahinter 0 oder nichts kommt. Ein Gap ist entsprechend eine Teilfolge aus lauter Nullen, wo davor und dahinter 1 oder nichts kommt. Sei B_i die Anzahl der Blöcke der Länge i, G_i die Anzahl der Gaps der

Länge i. (Run ist Block oder Gap.) Der Test ist bestanden, wenn alle folgenden Ungleichungen erfüllt sind:

$$\begin{aligned} 2267 \le B_1 \le 2733 \qquad & \qquad 2267 \le G_1 \le 2733 \\ 1079 \le B_2 \le 1421 \qquad & \qquad 1079 \le G_2 \le 1421 \\ 502 \le B_3 \le 748 \qquad & \qquad 502 \le G_3 \le 748 \\ 223 \le B_4 \le 402 \qquad & \qquad 223 \le G_4 \le 402 \\ 90 \le B_5 \le 223 \qquad & \qquad 90 \le G_5 \le 223 \\ 90 \le \sum_{i \ge 6} B_i \le 223 \qquad & \qquad 90 \le \sum_{i \ge 6} G_i \le 223 \end{aligned}$$

(4) Der Long-Runs-Test: Er ist bestanden, falls

$$B_i = G_i = 0 \quad \text{für alle } i \ge 34$$

gilt.

Das folgende Programm fips8.c führt die entsprechenden statistischen Tests durch:

```
/* fips8.c - Version vom 22.5.2001
   Statistischer Zufallszahlentest nach FIPS PUB 140-1
   Die ersten 2500 Bytes (20000 Bits) einer Datei werden getestet.
   Eventuell wird die Datei zurueckgespult. */

#include <stdio.h>

/* Ausgabe von x in Binaerdarstellung mit den letzten n Ziffern */
void bitausgabe( unsigned x, int n)
{
  int a[32], i;
  for (i=0;i<n;i++) { a[i]=x&1; x>>=1; }
  for (i=n-1;i>=0;i--) printf("%d",a[i]);
}

main( int argc, char *argv[])
{
  int i, j, c, s, n=0, n1=0, m=0, z=0, bg=0, l=0;
  int np[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, B[1000], G[1000];
  float X;
  FILE *ein;

  for (i=0;i<1000;i++) { B[i]=0; G[i]=0; }

  if (argc!=2)
  {
    printf("Aufruf: fips8 ‘datei’\n");
    printf("2500 Bytes (20000 Bits) von ‘datei’ werden statistisch ");
    printf("getestet.\nIst ‘datei’ zu klein, wird von vorne ");
    printf("begonnen.\n");
    return;
  }

  if ((ein=fopen(argv[1],"rb"))==NULL)
  {
    printf("‘%s’ existiert nicht!\n",argv[1]);
```

```
    return;
  }

  j=8; /* j nummeriert die Bits in Byte c */
  while (n<20000)
  {
    n++;
    if ((++j)==9)
    {
      while ((c=getc(ein))==EOF) rewind(ein);
      j=1; /* Begonnen wird mit Bit Nr. 1 */
    }
    s=(c>>(8-j))&1; /* s ist das j. Bit von Byte c */

    /* monobit test: Die Anzahl n1 der Einsen wird gezaehlt */
    if (s==1) n1++;

    /* poker test: Jeweils 4 Bits werden zu z mit 0<=z<=15
     * zusammengefasst. np[z] zaehlt, wie oft Typ z auftritt. */
    m++; z=2*z+s;
    if (m==4)
    {
      np[z]++;
      m=0; z=0;
    }

    /* runs test: Bloecke (aus Einsen) und Luecken (aus Nullen) werden
     * gezaehlt. l ist die aktuelle Laenge, bg die aktuelle 'Farbe'. */
    if ((bg==0 && s==0) || (bg==1 && s==1)) l++;
    else
    {
      if (l>999) l=999;
      if (bg==0) G[l]++; else B[l]++;
      bg=s; l=1;
    }
  }
  /* Abschluss des runs tests */
  if (l>999) l=999;
  if (bg==0) G[l]++; else B[l]++;


  printf("Ergebnisse:\n");
  printf("Von %d getesteten Bits hatten %d den Wert 1.\n",n,n1);

  printf("Wird die Folge in Bloecke der Laenge 4 aufgeteilt, ");
  printf("treten folgende Typen auf:\n");
  for (i=0;i<16;i++)
  {
    printf("Typ "); bitausgabe(i,4); printf(": %d\n",np[i]);
  }
  X=0; for (i=0;i<16;i++) X+=np[i]*np[i]; X=16*X/5000-5000;
  printf("Poker-Test-Statistik-Wert: %f\n",X);

  printf("Anzahl der Bloecke (aus Einsen) und Luecken (aus Nullen):\n");
```

```
  for (i=1;i<998;i++)
  {
    if (B[i]>0 || G[i]>0) printf("Laenge=%d: (%d,%d)\n",i,B[i],G[i]);
  }
  if (B[999]>0 || G[999]>0)
    printf("Laenge>998: (%d,%d)\n",B[999],G[999]);

  printf("\nTestergebnisse:\n");

  printf("Monobit-Test: ");
  if (9654<n1 && n1<10346) printf("o.k.\n");
  else printf("nicht bestanden!\n");

  printf("Poker-Test: ");
  if (1.03<X && X<57.4) printf("o.k.\n");
  else printf("nicht bestanden!\n");

  /* Long-Run-Test */
  l=1;
  for (i=34;i<1000;i++)
    if (B[i]>0 || G[i]>0) l=0;

  for (i=7;i<1000;i++) { B[6]+=B[i]; G[6]+=G[i]; }
  printf("Runs-Test: ");
  if (2267<=B[1] && B[1]<=2733 && 2267<=G[1] && G[1]<=2733 &&
      1079<=B[2] && B[2]<=1421 && 1079<=G[2] && G[2]<=1421 &&
       502<=B[3] && B[3]<=748  &&  502<=G[3] && G[3]<=748 &&
       223<=B[4] && B[4]<=402  &&  223<=G[4] && G[4]<=402 &&
        90<=B[5] && B[5]<=223  &&   90<=G[5] && G[5]<=223 &&
        90<=B[6] && B[6]<=223  &&   90<=G[6] && G[6]<=223)
    printf("o.k.\n");
  else printf("nicht bestanden!\n");

  printf("Long-Runs-Test: ");
  if (l==1) printf("o.k.\n"); else printf("nicht bestanden!\n");
}
```

Beispiel: Schieberegister mit primitivem Verbindungspolynom $1 + t^2 + t^5$ und Anfangszustand 11111.

```
Ergebnisse:
Von 20000 getesteten Bits hatten 10325 den Wert 1.
Wird die Folge in Bloecke der Laenge 4 aufgeteilt, treten folgende Typen auf:
Typ 0000: 161
Typ 0001: 323
Typ 0010: 323
Typ 0011: 322
Typ 0100: 323
Typ 0101: 322
Typ 0110: 322
Typ 0111: 322
Typ 1000: 322
Typ 1001: 323
Typ 1010: 323
Typ 1011: 324
Typ 1100: 322
```

```
Typ 1101: 322
Typ 1110: 322
Typ 1111: 324
Poker-Test-Statistik-Wert: 78.367996
Anzahl der Bloecke (aus Einsen) und Luecken (aus Nullen):
Laenge=1: (2580,2580)
Laenge=2: (1290,1290)
Laenge=3: (645,645)
Laenge=4: (0,645)
Laenge=5: (646,0)

Testergebnisse:
Monobit-Test: o.k.
Poker-Test: nicht bestanden!
Runs-Test: nicht bestanden!
Long-Runs-Test: o.k.
```

(Die Ausgabefolge hat Periodenlänge $2^5 - 1 = 31$, die 4-Blöcke wiederholen sich also jeweils nach 124 Folgengliedern.)

Beispiel: Schieberegister mit Verbindungspolynom $C(t) = 1 + t^2 + t^6 + t^7 + t^{32}$ und Anfangszustand $111\ldots111$. (Die Periodenlänge ist hier bereits $2^{32} - 1 = 4294967295$.)

```
Ergebnisse:
Von 20000 getesteten Bits hatten 9922 den Wert 1.
Wird die Folge in Bloecke der Laenge 4 aufgeteilt, treten folgende Typen auf:
Typ 0000: 300
Typ 0001: 347
Typ 0010: 300
Typ 0011: 320
Typ 0100: 319
Typ 0101: 284
Typ 0110: 316
Typ 0111: 282
Typ 1000: 335
Typ 1001: 300
Typ 1010: 325
Typ 1011: 331
Typ 1100: 337
Typ 1101: 310
Typ 1110: 288
Typ 1111: 306
Poker-Test-Statistik-Wert: 18.451200
Anzahl der Bloecke (aus Einsen) und Luecken (aus Nullen):
Laenge=1: (2464,2454)
Laenge=2: (1290,1229)
Laenge=3: (630,660)
Laenge=4: (319,321)
Laenge=5: (154,170)
Laenge=6: (63,77)
Laenge=7: (37,44)
Laenge=8: (14,21)
Laenge=9: (9,8)
Laenge=10: (0,3)
Laenge=11: (6,0)
Laenge=12: (0,1)
```

```
Laenge=14: (1,0)
Laenge=32: (1,0)

Testergebnisse:
Monobit-Test: o.k.
Poker-Test: o.k.
Runs-Test: o.k.
Long-Runs-Test: o.k.
```

Beispiel: Die getestete Folge wurde mit dem Shrinking-Generator erzeugt (bzw. mit streamsh.c, dem Schluessel 01234567 89abcdef und einer Datei mit lauter 0-Bits):

```
Ergebnisse:
Von 20000 getesteten Bits hatten 9944 den Wert 1.
Wird die Folge in Bloecke der Laenge 4 aufgeteilt, treten folgende Typen auf:
Typ 0000: 317
Typ 0001: 320
Typ 0010: 298
Typ 0011: 309
Typ 0100: 311
Typ 0101: 297
Typ 0110: 328
Typ 0111: 296
Typ 1000: 320
Typ 1001: 323
Typ 1010: 298
Typ 1011: 285
Typ 1100: 345
Typ 1101: 317
Typ 1110: 343
Typ 1111: 293
Poker-Test-Statistik-Wert: 14.636800
Anzahl der Bloecke (aus Einsen) und Luecken (aus Nullen):
Laenge=1: (2447,2433)
Laenge=2: (1263,1265)
Laenge=3: (658,651)
Laenge=4: (305,297)
Laenge=5: (138,147)
Laenge=6: (83,86)
Laenge=7: (40,40)
Laenge=8: (14,25)
Laenge=9: (11,13)
Laenge=10: (4,7)
Laenge=11: (2,2)
Laenge=12: (3,1)

Testergebnisse:
Monobit-Test: o.k.
Poker-Test: o.k.
Runs-Test: o.k.
Long-Runs-Test: o.k.
```

Beispiel: Die Folge entstand durch Verschlüsselung einer Datei aus lauter 0-Bits mit RC4 und dem Schlüssel 24052001:

```
Ergebnisse:
```

```
Von 20000 getesteten Bits hatten 9887 den Wert 1.
Wird die Folge in Bloecke der Laenge 4 aufgeteilt, treten folgende Typen auf:
Typ 0000: 321
Typ 0001: 312
Typ 0010: 327
Typ 0011: 309
Typ 0100: 357
Typ 0101: 313
Typ 0110: 327
Typ 0111: 343
Typ 1000: 296
Typ 1001: 312
Typ 1010: 271
Typ 1011: 311
Typ 1100: 304
Typ 1101: 310
Typ 1110: 317
Typ 1111: 270
Poker-Test-Statistik-Wert: 23.417601
Anzahl der Bloecke (aus Einsen) und Luecken (aus Nullen):
Laenge=1: (2503,2448)
Laenge=2: (1274,1296)
Laenge=3: (619,651)
Laenge=4: (311,272)
Laenge=5: (169,156)
Laenge=6: (68,95)
Laenge=7: (37,51)
Laenge=8: (15,17)
Laenge=9: (8,13)
Laenge=10: (2,6)
Laenge=11: (1,0)
Laenge=12: (0,1)

Testergebnisse:
Monobit-Test: o.k.
Poker-Test: o.k.
Runs-Test: o.k.
Long-Runs-Test: o.k.
```

Beispiel: Wir haben bei den ersten 20000 Folgengliedern der mit der Maple-Funktion rand() produzierten Folge jeweis das niedrigste Bit gewählt:

```
Ergebnisse:
Von 20000 getesteten Bits hatten 10049 den Wert 1.
Wird die Folge in Bloecke der Laenge 4 aufgeteilt, treten folgende Typen auf:
Typ 0000: 306
Typ 0001: 317
Typ 0010: 327
Typ 0011: 310
Typ 0100: 298
Typ 0101: 312
Typ 0110: 301
Typ 0111: 328
Typ 1000: 331
Typ 1001: 316
```

```
Typ 1010: 271
Typ 1011: 316
Typ 1100: 300
Typ 1101: 300
Typ 1110: 344
Typ 1111: 323
Poker-Test-Statistik-Wert: 13.971200
Anzahl der Bloecke (aus Einsen) und Luecken (aus Nullen):
Laenge=1: (2461,2447)
Laenge=2: (1206,1245)
Laenge=3: (635,644)
Laenge=4: (316,311)
Laenge=5: (169,159)
Laenge=6: (81,62)
Laenge=7: (47,48)
Laenge=8: (22,17)
Laenge=9: (7,9)
Laenge=10: (5,5)
Laenge=11: (3,4)
Laenge=12: (1,2)
Laenge=13: (1,0)

Testergebnisse:
Monobit-Test: o.k.
Poker-Test: o.k.
Runs-Test: o.k.
Long-Runs-Test: o.k.
```

Die statistischen Tests in FIPS PUB 140-1 sollen für die Nachfolgeversion **FIPS PUB 140-2** wie folgt verschärft werden: Gegeben sei ein $n = 20000$-Bit-langes Stück einer 0-1-Folge.

(1) Der Monobit-Test: Die Anzahl n_1 der Einsen sollte im Bereich

$$9725 < n_1 < 10275 \quad \text{(statt früher } 9654 < n_1 < 10346\text{)}$$

liegen.

(2) Beim Poker-Test sollte

$$2.16 < X < 46.17 \quad \text{(statt früher } 1.03 < X < 57.4\text{)}$$

gelten.

(3) Der Runs-Test: Für die Anzahl der Blöcke und Gaps der Länge i sollten folgende Intervallbedingungen gelten

i	B_i, G_i im Intervall	früher
1	2343 – 2657	2267 – 2733
2	1135 – 1365	1079 – 1421
3	542 – 708	502 – 748
4	251 – 373	223 – 402
5	111 – 201	90 – 223
6+	111 – 201	90 – 223

(4) Der Long-Runs-Test: Er ist bestanden, falls

$$B_i = G_i = 0 \quad \text{für alle } i \geq 26 \quad \text{(statt früher } i \geq 34\text{)}$$

gilt.

11. Anhang

Das folgende Programm lfsr.c wurde zur Konstruktion von Schieberegisterausgabefolgen benutzt:

```
/* lfsr.c - 23.02.2001 */
/* Maximalwert unsigned: 2^32-1 = 4294967295 = 0.4*10^10 */

#include <stdio.h>

int umw2( int c)
{
  if ('0'<=c && c<='9') return c-'0';
  if ('a'<=c && c<='f') return c-'a'+10;
  if ('A'<=c && c<='F') return c-'A'+10;
}

/* Ausgabe von x in Binaerdarstellung mit den letzten n Ziffern */
void bitausgabe( unsigned x, int n)
{
  int a[32], i;
  for (i=0;i<n;i++) { a[i]=x&1; x>>=1; }
  for (i=n-1;i>=0;i--) printf("%d",a[i]);
}

/* Summe der Bits mod 2 */
int subi2( unsigned x)
{
  int az=0;
  while (x!=0) { az^=1; x&=(x-1); }
  return az;
}

unsigned takt( unsigned zustand, unsigned verpol, int n)
{
  if (subi2(zustand&verpol)==1) { zustand>>=1; zustand|=(1<<(n-1)); }
  else zustand>>=1;
  return zustand;
}

main()
{
  int i, j, n, b, az;
  unsigned c, r;
  char name[50], ver[50], zu[50];
  FILE *aus, *ausbin;

  printf("Ausgabefolge eines Schieberegisters:\n");

  printf("Ausgabedatei: "); scanf("%s",name); aus=fopen(name,"wb");
  strcat(name,".bin"); printf("Byte-Ausgabe: %s\n",name);
  ausbin=fopen(name,"wb");

  printf("Laenge des Schieberegisters: "); scanf("%d",&n);
  if (n>32) { printf("Zu grosse Laenge!\n"); return; }

  printf("Koeffizienten c_1,...,c_%d des Verbindungspolnoms:\n",n);
```

```
  scanf("%s",ver);
  if (strlen(ver)!=n) { printf("Falsche Eingabe!\n"); return;}

  printf("Anfangszustand s_%d,...,s_0 des Schieberegisters:\n",n-1);
  scanf("%s",zu);
  if (strlen(zu)!=n) { printf("Falsche Eingabe!\n"); return;}

  c=0; r=0;
  for (i=0;i<n;i++)
  {
    if ((b=ver[i])=='0' || b=='1') c=2*c+(b-'0');
    else { printf("Falsches Verbindungspolynom!\n"); return; }

    if ((b=zu[i])=='0' || b=='1') r=2*r+(b-'0');
    else { printf("Falscher Anfangszustand!\n"); return; }
  }
  printf("c=%d r=%d\n",c,r);

  printf("Wieviele Bits sollen erzeugt werden? "); scanf("%d",&az);

  j=0; b=0;
  printf("Ausgabefolge:\n");
  for (i=0;i<az;i++)
  {
    printf("%d ",r&1); fprintf(aus,"%d",r&1);
    j++; b=2*b+(r&1);
    if (j==8) { putc(b,ausbin); j=0; }
    r=takt(r,c,n);
  }
  printf("\n");
}
```

Literatur

- A. J. Menezes, P. C. van Oorschot, S. C. Vanstone, Handbook of Applied Cryptography, CRC Press 1996.
- O. Forster, Algorithmische Zahlentheorie, Vieweg 1996.
- D. E. Knuth, The Art of Computer Programming, Second Edition, Vol. 2, Seminumerical Algorithms, Addison-Wesley 1981.
- U.S. Department of Commerce / National Institute of Standards and Technology, FIPS PUB 140-1, Security Requirements for Cryptographic Modules, 1994.
- U.S. Department of Commerce / National Institute of Standards and Technology, FIPS PUB 140-2, Security Requirements for Cryptographic Modules, 2001.

KAPITEL 2

Advanced Encryption Standard (AES)

1. Einführung

1997 begann das NIST (National Institute of Standards and Technology) mit der Entwicklung eines neuen Verschlüsselungsstandards – Advanced Encryption Standard (AES), um den seit 1976 benutzten Data Encryption Standard (DES) abzulösen. Es sollte sich dabei um eine (symmetrische) Blockchiffrierung mit 128-Bit-Blocklänge und möglichen Schlüssellängen von 128, 192 oder 256 Bits handeln. Die kryptographische Gemeinschaft wurde angeregt, entsprechende Algorithmen zu entwickeln und Vorschläge einzureichen.

Am 20. August 1998 wurde dann nachfolgende Liste mit 15 AES-Kandidaten bekanntgegeben.

Name des Verfahrens	eingereicht von
CAST-256	Entrust Technologies, Inc.
CRYPTON	Future Systems, Inc.
DEAL	Richard Outerbridge, Lars Knudson
DFC	CNRS - Centre National pour la Recherche Scientifique - Ecole Normale Superieure
E2	NTT - Nippon Telegraph and Telephone Corporation
FROG	TecApro Internacional S.A.
HPC	Rich Schroeppel
LOKI97	Lawrie Brown, Josef Pieprzyk, Jennifer Seberry
MAGENTA	Deutsche Telekom AG
MARS	IBM
RC6	RSA Laboratories
RIJNDAEL	Joan Daemen, Vincent Rijmen
SAFER+	Cylink Corporation
SERPENT	Ross Anderson, Eli Biham, Lars Knudson
TWOFISH	Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson

Nach entsprechenden Diskussionen wurden im April 1999 aus dieser Liste zur weiteren Begutachtung fünf Kandidaten ausgewählt, nämlich MARS, RC6, Rijndael, Serpent und Twofish.

Am 2. Oktober 2000 wurde schließlich bekanntgegeben, daß die Wahl für den neuen Standard AES auf Rijndael gefallen ist.

Wir werden im nachfolgenden die Version AES-128 vorstellen. Dabei handelt es sich um eine Abbildung

$$\text{Cipher}(\text{Eingabeblock}, \text{Schlüssel}) = \text{Ausgabeblock},$$

wo Eingabeblock, Schlüssel und Ausgabeblock jeweils 16 Bytes (128 Bits) haben.

2. Bytes und Zustände

Ein Byte kann man sich als ganze Zahl b zwischen 0 und $255 = 2^8 - 1$ vorstellen. Nimmt man die Binärdarstellung

$$b = (b_7b_6b_5b_4b_3b_2b_1b_0)_2 = b_7 \cdot 128 + b_6 \cdot 64 + b_5 \cdot 32 + b_4 \cdot 16 + b_3 \cdot 8 + b_2 \cdot 4 + b_1 \cdot 2 + b_0, \quad b_i \in \{0, 1\},$$

so ergibt $b_7b_6b_5b_4b_3b_2b_1b_0$ die zugehörige Bitfolge. In der Praxis gibt man ein Byte meist als 2-stellige Hexadezimalzahl (mit Ziffern 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f) an.

Für die Zwecke des vorliegenden Verschlüsselungsverfahrens identifiziert man Bytes vermöge der Zuordnung

$$b = (b_7b_6b_5b_4b_3b_2b_1b_0)_2 \quad \longleftrightarrow \quad b_7\alpha^7 + b_6\alpha^6 + b_5\alpha^5 + b_4\alpha^4 + b_3\alpha^3 + b_2\alpha^2 + b_1\alpha + b_0$$

mit Polynomen vom Grad ≤ 7 über $\mathbf{F}_2 \simeq \{0,1\}$ in der Unbestimmten α. Die zugehörige Menge sei

$$F = \{b_7\alpha^7 + b_6\alpha^6 + b_5\alpha^5 + b_4\alpha^4 + b_3\alpha^3 + b_2\alpha^2 + b_1\alpha + b_0 : b_i \in \mathbf{F}_2\}.$$

Die Elemente aus F kann man addieren, was der XOR-Operation auf (der gewöhnlichen) Byte-Ebene entspricht.

Beispiele: Die nachfolgende Tabelle gibt verschiedene Darstellungsmöglichkeiten einiger Bytes an.

Byte	in Hexadazimaldarstellung	als Bitfolge	als Polynom
0	00	00000000	0
1	01	00000001	1
2	02	00000010	α
3	03	00000011	$\alpha + 1$
10	0a	00001010	$\alpha^3 + \alpha$
99	63	01100011	$\alpha^6 + \alpha^5 + \alpha + 1$
255	ff	11111111	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1$

Hier sind zwei Maple-Funktionen, die Bytes in Polynome und umgekehrt umwandeln:

```
byte2poly:=proc() # wandelt Bytes in Polynome vom Grad <=7 um
  local b, p, i;
  b:=args[1]; p:=0;
  for i from 0 to 7 do
    p:=p+(b mod 2)*alpha^i; b:=iquo(b,2);
  od;
  p;
end;

poly2byte:=proc() # invers zu byte2poly
  subs(alpha=2,args[1]);
end;
```

Die Menge F ist ein $\mathbf{F}_2$-Vektorraum der Dimension 8, also bzgl. Addition eine abelsche Gruppe mit $2^8 = 256$ Elementen. Ist $f \in \mathbf{F}_2[\alpha]$ ein irreduzibles Polynom vom Grad 8, so wird F mit der Multiplikation

$$F \times F \to F, \quad (b,c) \mapsto bc \bmod f$$

zu einem Körper ($\simeq \mathbf{F}_{256}$). Für spätere Zwecke wählen wir das (modulo 2) irreduzible Polynom

$$f_\alpha = \alpha^8 + \alpha^4 + \alpha^3 + 1.$$

Wie findet man zu $b \in F \setminus \{0\}$ das Inverse modulo f_α? Mit dem erweiterten euklidischen Algorithmus bestimmt man $\widetilde{b}, c \in F$ mit $\widetilde{b}b + cf_\alpha = 1$. Dann ist $\widetilde{b}b \equiv 1 \bmod f_\alpha$, d.h. $\widetilde{b}$ ist (multiplikativ) invers zu b modulo f_α. Wir schreiben auch $\widetilde{b} \equiv \frac{1}{b} \bmod f_\alpha$.

Die nachfolgende Maple-Funktion führt das Verfahren praktisch durch, wobei für spätere Zwecke bei Eingabe von 0 auch 0 zurückgegeben wird:

```
falpha:=alpha^8+alpha^4+alpha^3+alpha+1;

invalpha:=proc() # berechnet 1/b mod falpha zu gegebenem b
  local i, j;
  global alpha, falpha;
  if Gcdex(args[1],falpha,alpha,`i`,`j`) mod 2<>1 then i:=0; fi;
```

```
  i;
end;
```

Die vorliegende Blockchiffrierung behandelt jeweils Blöcke aus 16 Bytes, die als *Zustand* bezeichnet werden. Genauer besteht ein Zustand S aus 16 Bytes s_{ij}, $0 \le i,j \le 3$, die als 4×4-Matrix angeordnet werden:

$$S = \begin{array}{|c|c|c|c|}\hline s_{00} & s_{01} & s_{02} & s_{03} \\ \hline s_{10} & s_{11} & s_{12} & s_{13} \\ \hline s_{20} & s_{21} & s_{22} & s_{23} \\ \hline s_{30} & s_{31} & s_{32} & s_{33} \\ \hline \end{array}$$

3. Die Verschlüsselungsabbildung Cipher

Die Verschlüsselung setzt voraus, daß man zunächst den 16-Byte-langen Schlüssel $k_0, k_1, \ldots, k_{15}$ mit der weiter unten beschriebenen Operation KeyExpansion zu einem 176-Byte-langen Schlüssel $k_0, k_1, \ldots, k_{175}$ erweitert hat.
Weiter werden die Zustandsabbildungen SubBytes, ShiftRows, MixColumns und AddRoundKey benötigt, die später definiert werden.

Die Verschlüsselungsabbildung Cipher**:** Eingegeben werden 16 Bytes $x_0, x_1, \ldots, x_{15}$ und ein 176-Byte-langer Schlüssel $k_0, k_1, \ldots, k_{175}$. Ausgegeben werden 16 Bytes $y_0, y_1, \ldots, y_{15}$.

- Die 16 Eingangsbytes $x_0, x_1, \ldots, x_{15}$ definieren über

$$S_0 = \begin{array}{|c|c|c|c|}\hline x_0 & x_4 & x_8 & x_{12} \\ \hline x_1 & x_5 & x_9 & x_{13} \\ \hline x_2 & x_6 & x_{10} & x_{14} \\ \hline x_3 & x_7 & x_{11} & x_{15} \\ \hline \end{array} \quad \text{bzw.} \quad (S_0)_{ij} = x_{i+4j} \quad (0 \le i,j \le 3)$$

  den Anfangszustand S_0. (Man beachte die Reihenfolge der Einträge!) Dann berechnet man

$$S_1 = \text{AddRoundKey}(S_0, k, 0).$$

- Für r von 1 bis 9 berechnet man rekursiv:

$$S_{r+1} = \text{AddRoundKey}(\text{MixColumns}(\text{ShiftRows}(\text{SubBytes}(S_r))), k, r)$$

- Der Zustand S_{11} ergibt sich aus

$$S_{11} = \text{AddRoundKey}(\text{ShiftRows}(\text{SubBytes}(S_{10})), k, 10).$$

- Die 16 Ausgabebytes $y_0, y_1, \ldots, y_{15}$ erhält man nun aus

$$\begin{array}{|c|c|c|c|}\hline y_0 & y_4 & y_8 & y_{12} \\ \hline y_1 & y_5 & y_9 & y_{13} \\ \hline y_2 & y_6 & y_{10} & y_{14} \\ \hline y_3 & y_7 & y_{11} & y_{15} \\ \hline \end{array} = S_{11} \quad \text{bzw.} \quad y_l = (S_{11})_{l \bmod 4, \lfloor \frac{l}{4} \rfloor} \text{ für } 0 \le l \le 15.$$

Die folgende Maple-Funktion Cipher realisiert das dargestellte Verschlüsselungsverfahren, wobei praktisch darauf zu achten ist, ob man sich ein Byte b als Zahl aus $\{0, \ldots, 255\}$ oder als Polynom aus F denkt:

```
Cipher:=proc() # Eingabe: Liste mit 16 Bytes, Liste mit 16 Bytes (Schluessel)
  local zustand, k, r, i, j;
  zustand:=array(0..3,0..3);
  for i from 0 to 3 do
    for j from 0 to 3 do
      zustand[i,j]:=byte2poly(args[1][i+4*j+1]);
    od;
  od;
  k:=args[2];

  zustand:=AddRoundKey(zustand,k,0);
  for r from 1 to 9 do
```

```
    zustand:=AddRoundKey(MixColumns(ShiftRows(map(SubBytes,zustand))),k,r);
  od;
  zustand:=AddRoundKey(ShiftRows(map(SubBytes,zustand)),k,10);
  map(poly2byte,[zustand[0,0],zustand[1,0],zustand[2,0],zustand[3,0],
                 zustand[0,1],zustand[1,1],zustand[2,1],zustand[3,1],
                 zustand[0,2],zustand[1,2],zustand[2,2],zustand[3,2],
                 zustand[0,3],zustand[1,3],zustand[2,3],zustand[3,3]]);
end;
```

Als nächstes müssen die Zustandsabbildungen SubBytes, ShiftRows, MixColumns und AddRoundKey definiert werden.

4. Die Abbildung SubBytes

Mit dem zuvor eingeführten Polynom $f_\alpha = \alpha^8 + \alpha^4 + \alpha^3 + 1$ definieren wir zunächst eine Abbildung SubBytes : $F \to F$ wie folgt:

$$\text{SubBytes}(b) = b'' \quad \text{mit} \quad b' = \begin{cases} \frac{1}{b} \bmod f_\alpha & \text{für } b \neq 0, \\ 0 & \text{für } b = 0 \end{cases}$$
$$\text{und} \quad b'' \equiv b'(1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4) + (1 + \alpha + \alpha^5 + \alpha^6) \bmod \alpha^8 + 1.$$

Schreibt man

$$\begin{aligned} b &= b_7\alpha^7 + b_6\alpha^6 + b_5\alpha^5 + b_4\alpha^4 + b_3\alpha^3 + b_2\alpha^2 + b_1\alpha + b_0, \\ b' &= b'_7\alpha^7 + b'_6\alpha^6 + b'_5\alpha^5 + b'_4\alpha^4 + b'_3\alpha^3 + b'_2\alpha^2 + b'_1\alpha + b'_0, \\ b'' &= b''_7\alpha^7 + b''_6\alpha^6 + b''_5\alpha^5 + b''_4\alpha^4 + b''_3\alpha^3 + b''_2\alpha^2 + b''_1\alpha + b''_0, \end{aligned}$$

so kann man den Übergang von b' zu b'' explizit auch so ausdrücken:

$$\begin{aligned} b''_7 &= b'_5 + b'_6 + b'_7 + b'_4 + b'_3 \\ b''_6 &= b'_5 + b'_6 + b'_4 + b'_3 + b'_2 + 1 \\ b''_5 &= b'_5 + b'_4 + b'_3 + b'_2 + b'_1 + 1 \\ b''_4 &= b'_4 + b'_3 + b'_2 + b'_1 + b'_0 \\ b''_3 &= b'_7 + b'_3 + b'_2 + b'_1 + b'_0 \\ b''_2 &= b'_6 + b'_7 + b'_2 + b'_1 + b'_0 \\ b''_1 &= b'_5 + b'_6 + b'_7 + b'_1 + b'_0 + 1 \\ b''_0 &= b'_5 + b'_6 + b'_7 + b'_4 + b'_0 + 1 \end{aligned}$$

Beispiele:

Byte	als Polynom b	b'	SubBytes(b)	als Byte
0	0	0	$\alpha^6 + \alpha^5 + \alpha + 1$	99
1	1	1	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2$	124
2	α	$\alpha^7 + \alpha^3 + \alpha^2 + 1$	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha + 1$	119
3	$\alpha + 1$	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha$	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha + 1$	123

Hier ist eine mögliche Maple-Funktion:

```
SubBytes:=proc()
  local b;
  b:=invalpha(args[1]);
  Rem(b*(1+alpha+alpha^2+alpha^3+alpha^4)+byte2poly(6*16+3),
      alpha^8-1,alpha) mod 2;
end;
```

Für die Praxis ist es sicher nützlicher, sich die Funktion SubBytes zu tabellieren:

```
SubBytes:=proc()
  local S;
  S:=[99,124,119,123,242,107,111,197,48,1,103,43,254,215,171,118,202,
      130,201,125,250,89,71,240,173,212,162,175,156,164,114,192,183,
      253,147,38,54,63,247,204,52,165,229,241,113,216,49,21,4,199,35,
      195,24,150,5,154,7,18,128,226,235,39,178,117,9,131,44,26,27,110,
      90,160,82,59,214,179,41,227,47,132,83,209,0,237,32,252,177,91,
      106,203,190,57,74,76,88,207,208,239,170,251,67,77,51,133,69,249,
      2,127,80,60,159,168,81,163,64,143,146,157,56,245,188,182,218,33,
      16,255,243,210,205,12,19,236,95,151,68,23,196,167,126,61,100,93,
      25,115,96,129,79,220,34,42,144,136,70,238,184,20,222,94,11,219,
      224,50,58,10,73,6,36,92,194,211,172,98,145,149,228,121,231,200,
      55,109,141,213,78,169,108,86,244,234,101,122,174,8,186,120,37,
      46,28,166,180,198,232,221,116,31,75,189,139,138,112,62,181,102,
      72,3,246,14,97,53,87,185,134,193,29,158,225,248,152,17,105,217,
      142,148,155,30,135,233,206,85,40,223,140,161,137,13,191,230,66,
      104,65,153,45,15,176,84,187,22];
  byte2poly(S[poly2byte(args[1])+1]);
end;
```

Die Abbildung SubBytes : $F \to F$ setzen wir zu einer (gleichbezeichneten) Abbildung SubBytes zwischen Zuständen fort:

$$\text{SubBytes}(S) = S', \text{ wenn } S = (s_{ij}) \text{ und } S' = (\text{SubBytes}(s_{ij})).$$

5. Die Abbildung ShiftRows

Die ShiftRows-Funktion operiert auf den Zuständen wie folgt:

$$\text{ShiftRows}\begin{array}{|c|c|c|c|}\hline s_{00} & s_{01} & s_{02} & s_{03} \\ \hline s_{10} & s_{11} & s_{12} & s_{13} \\ \hline s_{20} & s_{21} & s_{22} & s_{23} \\ \hline s_{30} & s_{31} & s_{32} & s_{33} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline s_{00} & s_{01} & s_{02} & s_{03} \\ \hline s_{11} & s_{12} & s_{13} & s_{10} \\ \hline s_{22} & s_{23} & s_{20} & s_{21} \\ \hline s_{33} & s_{30} & s_{31} & s_{32} \\ \hline \end{array}$$

Anders geschrieben: ShiftRows$(S) = S'$ mit $s'_{ij} = s_{i,i+j \bmod 4}$. (Die 2. Zeile wird also (zirkulär) um 1 nach links verschoben, die 3. Zeile um 2, die 4. Zeile um 3, die 1. Zeile bleibt unverändert.)

Eine Maple-Funktion:

```
ShiftRows:=proc()
  local z_ein, z_aus, i, j;
  z_ein:=args[1];
  z_aus:=array(0..3,0..3);
  for i from 0 to 3 do
    for j from 0 to 3 do
      z_aus[i,j]:=z_ein[i,i+j mod 4];S
    od;
  od;
  z_aus;
end;
```

6. Die Abbildung MixColumns

Ist $S = (s_{ij})$ ein Zustand, so wird $S' = \text{MixColumns}(S)$ durch $S' = (s'_{ij})$ mit

$$\begin{aligned} s'_{0j} &= (s_{1j} + s_{2j} + s_{3j}) + (s_{0j} + s_{1j})\alpha \bmod f_\alpha \\ s'_{1j} &= (s_{0j} + s_{2j} + s_{3j}) + (s_{1j} + s_{2j})\alpha \bmod f_\alpha \\ s'_{2j} &= (s_{0j} + s_{1j} + s_{3j}) + (s_{2j} + s_{3j})\alpha \bmod f_\alpha \\ s'_{3j} &= (s_{0j} + s_{1j} + s_{2j}) + (s_{0j} + s_{3j})\alpha \bmod f_\alpha \end{aligned}$$

definiert. Jede Spalte wird also einzeln transformiert.
In Matrizenschreibweise sieht dies so aus:

$$\begin{pmatrix} s'_{0j} \\ s'_{1j} \\ s'_{2j} \\ s'_{3j} \end{pmatrix} = \begin{pmatrix} \alpha & 1+\alpha & 1 & 1 \\ 1 & \alpha & 1+\alpha & 1 \\ 1 & 1 & \alpha & 1+\alpha \\ 1+\alpha & 1 & 1 & \alpha \end{pmatrix} \begin{pmatrix} s_{0j} \\ s_{1j} \\ s_{2j} \\ s_{3j} \end{pmatrix} \bmod f_\alpha$$

Beispiel: Wir betrachten nur die erste Spalte des folgenden Zustands:

$$S = \begin{array}{|c|c|c|c|} \hline 212 & & & \\ \hline 191 & & & \\ \hline 93 & & & \\ \hline 48 & & & \\ \hline \end{array} \longleftrightarrow \begin{array}{|c|c|c|c|} \hline \alpha^7+\alpha^6+\alpha^4+\alpha^2 & & & \\ \hline \alpha^7+\alpha^5+\alpha^4+\alpha^3+\alpha^2+\alpha+1 & & & \\ \hline \alpha^6+\alpha^4+\alpha^3+\alpha^2+1 & & & \\ \hline \alpha^5+\alpha^4 & & & \\ \hline \end{array}$$

Wir haben also

$$\begin{aligned} s_{00} &= \alpha^6+\alpha^4+\alpha^2+\alpha^7 \\ s_{10} &= 1+\alpha^7+\alpha^5+\alpha^4+\alpha+\alpha^3+\alpha^2 \\ s_{20} &= \alpha^4+\alpha^3+\alpha^2+1+\alpha^6 \\ s_{30} &= \alpha^5+\alpha^4 \end{aligned}$$

und erhalten mit obigen Formeln

$$\begin{aligned} s'_{00} &= \alpha^2 \\ s'_{10} &\equiv \alpha^6+\alpha^2+\alpha^4+1+\alpha^3+\alpha^5+\alpha^8 \equiv \alpha+\alpha^5+\alpha^2+\alpha^6 \bmod f_\alpha \\ s'_{20} &= 1+\alpha^7 \\ s'_{30} &\equiv \alpha^7+\alpha^6+\alpha^2+\alpha^4+\alpha+\alpha^3+\alpha^5+\alpha^8 = 1+\alpha^2+\alpha^6+\alpha^7+\alpha^5 \bmod f_\alpha \end{aligned}$$

Damit wird

$$\text{MixColumns}(S) = \begin{array}{|c|c|c|c|} \hline \alpha^2 & & & \\ \hline \alpha^2+\alpha+\alpha^6+\alpha^5 & & & \\ \hline 1+\alpha^7 & & & \\ \hline \alpha^2+\alpha^6+1+\alpha^5+\alpha^7 & & & \\ \hline \end{array} \longleftrightarrow \begin{array}{|c|c|c|c|} \hline 4 & & & \\ \hline 102 & & & \\ \hline 129 & & & \\ \hline 229 & & & \\ \hline \end{array}$$

Beispiel: Wir nehmen an, in jeder Zeile der Spalte j eines Zustands S steht b, d.h. $s_{ij} = b$ für $0 \le i \le 3$. Dann ist $s'_{ij} = b$ für $0 \le i \le 3$, so daß auch in jeder Zeile der Spalte j von $S' = \text{MixColumns}(S)$ der Eintrag b steht. Also:

$$S = \begin{array}{|c|c|c|c|} \hline & b & & \\ \hline & b & & \\ \hline & b & & \\ \hline & b & & \\ \hline \end{array} \text{ und } S' = \text{MixColumns}(S) = \begin{array}{|c|c|c|c|} \hline & b & & \\ \hline & b & & \\ \hline & b & & \\ \hline & b & & \\ \hline \end{array}$$

Bemerkung: Die obige Transformationsmatrix hat das charakteristische Polynom $x^4 + 1 = (x-1)^4$ (modulo 2), 1 ist der einzige Eigenwert, die Eigenvektoren sind Vielfache von $(1,1,1,1)^t$. Wird also eine Spalte eines Zustands durch MixColumns in sich selbst überführt, hat sie die Gestalt $(b,b,b,b)^t$ wie in obigem Beispiel.

Die folgende Maple-Funktion führt MixColumns aus:

```
MixColumns:=proc()
  local s, z_aus, i, j, p;
  s:=args[1];
  z_aus:=array(0..3,0..3);
  for j from 0 to 3 do
    z_aus[0,j]:=Rem(s[1,j]+s[2,j]+s[3,j]+(s[0,j]+s[1,j])*alpha,falpha,alpha) mod 2;
    z_aus[1,j]:=Rem(s[0,j]+s[2,j]+s[3,j]+(s[1,j]+s[2,j])*alpha,falpha,alpha) mod 2;
    z_aus[2,j]:=Rem(s[0,j]+s[1,j]+s[3,j]+(s[2,j]+s[3,j])*alpha,falpha,alpha) mod 2;
    z_aus[3,j]:=Rem(s[0,j]+s[1,j]+s[2,j]+(s[0,j]+s[3,j])*alpha,falpha,alpha) mod 2;
  od;
  z_aus;
end;
```

Bemerkung: Eine alternative Beschreibung von MixColumns sieht so aus: Wir bilden aus den Spalten eines Zustands $S = (s_{ij})_{0 \le i,j \le 3}$ Polynome

$$p_j = s_{0j} + s_{1j}x + s_{2j}x^2 + s_{3j}x^3$$

(in einer Unbestimmten x). Ist dann

$$p'_j = \left(p_j \cdot [(1+\alpha)x^3 + x^2 + x + \alpha] \bmod x^4 + 1\right) \bmod f_\alpha,$$

so ergeben sich die Koeffizienten des Zustands $S' = \mathrm{MixColumns}(S)$ durch Koeffizientenvergleich aus

$$p'_j = s'_{0j} + s'_{1j}x + s'_{2j}x^2 + s'_{3j}x^3.$$

7. Die Operation KeyExpansion

Eingegeben werden 16 Schlüsselbytes $k_0, k_1, k_2, \ldots, k_{15}$. Für $i \ge 16$ werden dann mit folgender Rekursionsformel die weiteren Schlüsselbytes $k_{16}, \ldots, k_{175}$ erzeugt. (Wir wählen die Darstellung $k_i \in F$.)

$$k_i = \begin{cases} k_{i-16} + \mathrm{SubBytes}(k_{i-3}) + \alpha^{\frac{i}{16}-1} \bmod f_\alpha & i \equiv 0 \bmod 16 \\ k_{i-16} + \mathrm{SubBytes}(k_{i-3}) & i \equiv 1, 2 \bmod 16 \\ k_{i-16} + \mathrm{SubBytes}(k_{i-7}) & i \equiv 3 \bmod 16 \\ k_{i-16} + k_{i-4} & i \not\equiv 0, 1, 2, 3 \bmod 16 \end{cases}$$

Alternativ kann man die 176 Schlüsselbytes $k_0, \ldots, k_{175}$ aus den gegebenen 16 Bytes $k_0, \ldots, k_{15}$ auch nach folgendem Verfahren gewinnen: Man setzt

$$w_i = (k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3})$$

und hat dann durch die Vorgabe von $k_0, \ldots, k_{15}$ bereits w_0, w_1, w_2, w_3 definiert.
Für $4 \le i \le 43$ definiert man rekursiv

$$w_i = w_{i-4} + \begin{cases} \mathrm{SubBytes}(k_{4i-3}, k_{4i-2}, k_{4i-1}, k_{4i-4}) + (\alpha^{\frac{i}{4}-1}, 0, 0, 0) \bmod f_\alpha & i \equiv 0 \bmod 4 \\ w_{i-1} & i \not\equiv 0 \bmod 4 \end{cases}$$

(Dies ist die ursprüngliche Darstellung. Man gewinnt daraus schnell die zuerst gegebene Definition.)

Beispiel: Wir beginnen mit folgendem Schlüssel in Hexadezimaldarstellung

2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

Daraus erhalten wir $k_0, \ldots, k_{15} \in F$:

$$k_0 = 1 + \alpha + \alpha^3 + \alpha^5 \qquad k_1 = \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6$$
$$k_2 = 1 + \alpha^2 + \alpha^4 \qquad k_3 = \alpha + \alpha^2 + \alpha^4$$
$$k_4 = \alpha^3 + \alpha^5 \qquad k_5 = \alpha + \alpha^2 + \alpha^3 + \alpha^5 + \alpha^7$$
$$k_6 = \alpha + \alpha^4 + \alpha^6 + \alpha^7 \qquad k_7 = \alpha + \alpha^2 + \alpha^5 + \alpha^7$$
$$k_8 = 1 + \alpha + \alpha^3 + \alpha^5 + \alpha^7 \qquad k_9 = 1 + \alpha + \alpha^2 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7$$
$$k_{10} = 1 + \alpha^2 + \alpha^4 \qquad k_{11} = \alpha^3 + \alpha^7$$
$$k_{12} = 1 + \alpha^3 \qquad k_{13} = 1 + \alpha + \alpha^2 + \alpha^3 + \alpha^6 + \alpha^7$$
$$k_{14} = 1 + \alpha + \alpha^2 + \alpha^3 + \alpha^6 \qquad k_{15} = \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5$$

Mit KeyExpansion berechnen wir $k_{16}, \ldots, k_{21}$:

$$\begin{aligned}
k_{16} &= k_0 + \text{SubBytes}(k_{13}) + 1 = (1 + \alpha + \alpha^3 + \alpha^5) + \text{SubBytes}(1 + \alpha + \alpha^2 + \alpha^3 + \alpha^6 + \alpha^7) + 1 = \\
&= (1 + \alpha + \alpha^3 + \alpha^5) + (\alpha + \alpha^3 + \alpha^7) + 1 = \alpha^5 + \alpha^7 \\
k_{17} &= k_1 + \text{SubBytes}(k_{14}) = (\alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6) + \text{SubBytes}(1 + \alpha + \alpha^2 + \alpha^3 + \alpha^6) = \\
&= (\alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6) + (\alpha^2 + \alpha^7) = \alpha + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7 \\
k_{18} &= k_2 + \text{SubBytes}(k_{15}) = (1 + \alpha^2 + \alpha^4) + \text{SubBytes}(\alpha^2 + \alpha^3 + \alpha^4 + \alpha^5) = \\
&= (1 + \alpha^2 + \alpha^4) + (1 + \alpha + \alpha^3 + \alpha^5 + \alpha^6 + \alpha^7) = \alpha + \alpha^2 + \alpha^4 + \alpha^7 + \alpha^6 + \alpha^5 + \alpha^3 \\
k_{19} &= k_3 + \text{SubBytes}(k_{12}) = (\alpha + \alpha^2 + \alpha^4) + \text{SubBytes}(1 + \alpha^3) = (\alpha + \alpha^2 + \alpha^4) + 1 = \\
&= 1 + \alpha + \alpha^2 + \alpha^4 \\
k_{20} &= k_4 + k_{16} = (\alpha^3 + \alpha^5) + (\alpha^5 + \alpha^7) = \alpha^3 + \alpha^7 \\
k_{21} &= k_5 + k_{17} = (\alpha + \alpha^2 + \alpha^3 + \alpha^5 + \alpha^7) + (\alpha + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7) = \alpha^2 + \alpha^4 + \alpha^6
\end{aligned}$$

Alle 176 Schlüsselbytes lassen sich kürzer in der Hexadezimalschreibweise angeben:

2b	7e	15	16	28	ae	d2	a6	ab	f7	15	88	09	cf	4f	3c
a0	fa	fe	17	88	54	2c	b1	23	a3	39	39	2a	6c	76	05
f2	c2	95	f2	7a	96	b9	43	59	35	80	7a	73	59	f6	7f
3d	80	47	7d	47	16	fe	3e	1e	23	7e	44	6d	7a	88	3b
ef	44	a5	41	a8	52	5b	7f	b6	71	25	3b	db	0b	ad	00
d4	d1	c6	f8	7c	83	9d	87	ca	f2	b8	bc	11	f9	15	bc
6d	88	a3	7a	11	0b	3e	fd	db	f9	86	41	ca	00	93	fd
4e	54	f7	0e	5f	5f	c9	f3	84	a6	4f	b2	4e	a6	dc	4f
ea	d2	73	21	b5	8d	ba	d2	31	2b	f5	60	7f	8d	29	2f
ac	77	66	f3	19	fa	dc	21	28	d1	29	41	57	5c	00	6e
d0	14	f9	a8	c9	ee	25	89	e1	3f	0c	c8	b6	63	0c	a6

Eine zugehörige Maple-Funktion:

```
KeyExpansion:=proc() # Eingabe: Liste mit 16 Bytes
  local k, i;
  k:=array(0..175);
  for i from 0 to 15 do k[i]:=byte2poly(args[1][i+1]); od;
  for i from 16 to 175 do
    if i mod 16=0 then
      k[i]:=Rem(k[i-16]+SubBytes(k[i-3])+alpha^(i/16-1),falpha,alpha) mod 2;
    elif i mod 16=1 or i mod 16=2 then
      k[i]:=k[i-16]+SubBytes(k[i-3]) mod 2;
    elif i mod 16=3 then
      k[i]:=k[i-16]+SubBytes(k[i-7]) mod 2;
    else
      k[i]:=k[i-16]+k[i-4] mod 2;
```

```
    fi;
  od;
  k;
end;
```

8. Die Abbildung AddRoundKey

Die Abbildung AddRoundKey wird wie folgt definiert:

$$\text{AddRoundKey}(S,k,r) = S' \text{ mit } s'_{ij} = s_{ij} + k_{16r+i+4j},$$

oder ausgeschrieben:

$$\text{AddRoundKey}\left(\begin{array}{|c|c|c|c|}\hline s_{00} & s_{01} & s_{02} & s_{03} \\ \hline s_{10} & s_{11} & s_{12} & s_{13} \\ \hline s_{20} & s_{21} & s_{22} & s_{23} \\ \hline s_{30} & s_{31} & s_{32} & s_{33} \\ \hline \end{array}\right) = \begin{array}{|c|c|c|c|}\hline s_{00}+k_{16r} & s_{01}+k_{16r+4} & s_{02}+k_{16r+8} & s_{03}+k_{16r+12} \\ \hline s_{10}+k_{16r+1} & s_{11}+k_{16r+5} & s_{12}+k_{16r+9} & s_{13}+k_{16r+13} \\ \hline s_{20}+k_{16r+2} & s_{21}+k_{16r+6} & s_{22}+k_{16r+10} & s_{23}+k_{16r+14} \\ \hline s_{30}+k_{16r+3} & s_{31}+k_{16r+7} & s_{32}+k_{16r+11} & s_{33}+k_{16r+15} \\ \hline \end{array}$$

(Man beachte die Reihenfolge der Schlüsselbytes $k_{16r}, \ldots, k_{16r+15}$.)

Eine Maple-Funktion:

```
AddRoundKey:=proc()
  local z_ein, z_aus, k, r, i, j;
  z_ein:=args[1]; k:=args[2]; r:=args[3];
  z_aus:=array(0..3,0..3);
  for i from 0 to 3 do
    for j from 0 to 3 do
      z_aus[i,j]:=z_ein[i,j]+k[16*r+4*j+i] mod 2;
    od;
  od;
  z_aus;
end;
```

Bemerkung: Da aus $s' \equiv s + k \bmod 2$ sofort $s \equiv s' + k \bmod 2$ folgt, gilt auch:

$$\text{AddRoundKey}(S,k,r) = S' \quad \Longrightarrow \quad \text{AddRoundKey}(S',k,r) = S,$$

d.h. die Funktion $S \mapsto \text{AddRoundKey}(S,k,r)$ ist zu sich selbst invers.

9. Beispiele

Nun sind alle notwendigen Abbildungen beschrieben um die Verschlüsselung konkret auszuführen. Wir geben zwei Beispiele an. Der Kürze halber sind die Bytes in Hexadezimaldarstellung geschrieben, die Zustände (s_{ij}) sind linear in einer Zeile in der Reihenfolge $s_{00}, s_{10}, s_{20}, s_{30}, s_{01}, s_{11}, \ldots$ aufgelistet.

Beispiel:

Eingabeblock:	00	11	22	33	44	55	66	77	88	99	aa	bb	cc	dd	ee	ff
Schluessel:	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
Anfangszustand:	00	11	22	33	44	55	66	77	88	99	aa	bb	cc	dd	ee	ff
Rundenschluessel:	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
nach AddRoundKey:	00	10	20	30	40	50	60	70	80	90	a0	b0	c0	d0	e0	f0
1. Runde:																
Rundenstart:	00	10	20	30	40	50	60	70	80	90	a0	b0	c0	d0	e0	f0
nach SubBytes:	63	ca	b7	04	09	53	d0	51	cd	60	e0	e7	ba	70	e1	8c
nach ShiftRows:	63	53	e0	8c	09	60	e1	04	cd	70	b7	51	ba	ca	d0	e7
nach MixColumns:	5f	72	64	15	57	f5	bc	92	f7	be	3b	29	1d	b9	f9	1a
Rundenschluessel:	d6	aa	74	fd	d2	af	72	fa	da	a6	78	f1	d6	ab	76	fe
nach AddRoundKey:	89	d8	10	e8	85	5a	ce	68	2d	18	43	d8	cb	12	8f	e4
2. Runde:																
Rundenstart:	89	d8	10	e8	85	5a	ce	68	2d	18	43	d8	cb	12	8f	e4
nach SubBytes:	a7	61	ca	9b	97	be	8b	45	d8	ad	1a	61	1f	c9	73	69
nach ShiftRows:	a7	be	1a	69	97	ad	73	9b	d8	c9	ca	45	1f	61	8b	61
nach MixColumns:	ff	87	96	84	31	d8	6a	51	64	51	51	fa	77	3a	d0	09
Rundenschluessel:	b6	92	cf	0b	64	3d	bd	f1	be	9b	c5	00	68	30	b3	fe
nach AddRoundKey:	49	15	59	8f	55	e5	d7	a0	da	ca	94	fa	1f	0a	63	f7
3. Runde:																
Rundenstart:	49	15	59	8f	55	e5	d7	a0	da	ca	94	fa	1f	0a	63	f7
nach SubBytes:	3b	59	cb	73	fc	d9	0e	e0	57	74	22	2d	c0	67	fb	68
nach ShiftRows:	3b	d9	22	68	fc	74	fb	73	57	67	cb	e0	c0	59	0e	2d
nach MixColumns:	4c	9c	1e	66	f7	71	f0	76	2c	3f	86	8e	53	4d	f2	56
Rundenschluessel:	b6	ff	74	4e	d2	c2	c9	bf	6c	59	0c	bf	04	69	bf	41
nach AddRoundKey:	fa	63	6a	28	25	b3	39	c9	40	66	8a	31	57	24	4d	17
4. Runde:																
Rundenstart:	fa	63	6a	28	25	b3	39	c9	40	66	8a	31	57	24	4d	17
nach SubBytes:	2d	fb	02	34	3f	6d	12	dd	09	33	7e	c7	5b	36	e3	f0
nach ShiftRows:	2d	6d	7e	f0	3f	33	e3	34	09	36	02	dd	5b	fb	12	c7
nach MixColumns:	63	85	b7	9f	fc	53	8d	f9	97	be	47	8e	75	47	d6	91
Rundenschluessel:	47	f7	f7	bc	95	35	3e	03	f9	6c	32	bc	fd	05	8d	fd
nach AddRoundKey:	24	72	40	23	69	66	b3	fa	6e	d2	75	32	88	42	5b	6c
5. Runde:																
Rundenstart:	24	72	40	23	69	66	b3	fa	6e	d2	75	32	88	42	5b	6c
nach SubBytes:	36	40	09	26	f9	33	6d	2d	9f	b5	9d	23	c4	2c	39	50
nach ShiftRows:	36	33	9d	50	f9	b5	39	26	9f	2c	09	2d	c4	40	6d	23
nach MixColumns:	f4	bc	d4	54	32	e5	54	d0	75	f1	d6	c5	1d	d0	3b	3c
Rundenschluessel:	3c	aa	a3	e8	a9	9f	9d	eb	50	f3	af	57	ad	f6	22	aa
nach AddRoundKey:	c8	16	77	bc	9b	7a	c9	3b	25	02	79	92	b0	26	19	96
6. Runde:																
Rundenstart:	c8	16	77	bc	9b	7a	c9	3b	25	02	79	92	b0	26	19	96
nach SubBytes:	e8	47	f5	65	14	da	dd	e2	3f	77	b6	4f	e7	f7	d4	90
nach ShiftRows:	e8	da	b6	90	14	77	d4	65	3f	f7	f5	e2	e7	47	dd	4f
nach MixColumns:	98	16	ee	74	00	f8	7f	55	6b	2c	04	9c	8e	5a	d0	36
Rundenschluessel:	5e	39	0f	7d	f7	a6	92	96	a7	55	3d	c1	0a	a3	1f	6b
nach AddRoundKey:	c6	2f	e1	09	f7	5e	ed	c3	cc	79	39	5d	84	f9	cf	5d
7. Runde:																
Rundenstart:	c6	2f	e1	09	f7	5e	ed	c3	cc	79	39	5d	84	f9	cf	5d
nach SubBytes:	b4	15	f8	01	68	58	55	2e	4b	b6	12	4c	5f	99	8a	4c
nach ShiftRows:	b4	58	12	4c	68	b6	8a	01	4b	99	f8	2e	5f	15	55	4c
nach MixColumns:	c5	7e	1c	15	9a	9b	d2	86	f0	5f	4b	e0	98	c6	34	39
Rundenschluessel:	14	f9	70	1a	e3	5f	e2	8c	44	0a	df	4d	4e	a9	c0	26
nach AddRoundKey:	d1	87	6c	0f	79	c4	30	0a	b4	55	94	ad	d6	6f	f4	1f
8. Runde:																
Rundenstart:	d1	87	6c	0f	79	c4	30	0a	b4	55	94	ad	d6	6f	f4	1f
nach SubBytes:	3e	17	50	76	b6	1c	04	67	8d	fc	22	95	f6	a8	bf	c0
nach ShiftRows:	3e	1c	22	c0	b6	fc	bf	76	8d	a8	50	67	f6	17	04	95
nach MixColumns:	ba	a0	3d	e7	a1	f9	b5	6e	d5	51	2c	ba	5f	41	4d	23
Rundenschluessel:	47	43	87	35	a4	1c	65	b9	e0	16	ba	f4	ae	bf	7a	d2
nach AddRoundKey:	fd	e3	ba	d2	05	e5	d0	d7	35	47	96	4e	f1	fe	37	f1
9. Runde:																
Rundenstart:	fd	e3	ba	d2	05	e5	d0	d7	35	47	96	4e	f1	fe	37	f1
nach SubBytes:	54	11	f4	b5	6b	d9	70	0e	96	a0	90	2f	a1	bb	9a	a1
nach ShiftRows:	54	d9	90	a1	6b	a0	9a	b5	96	bb	f4	0e	a1	11	70	2f
nach MixColumns:	e9	f7	4e	ec	02	30	20	f6	1b	f2	cc	f2	35	3c	21	c7
Rundenschluessel:	54	99	32	d1	f0	85	57	68	10	93	ed	9c	be	2c	97	4e
nach AddRoundKey:	bd	6e	7c	3d	f2	b5	77	9e	0b	61	21	6e	8b	10	b6	89
10. Runde:																
Rundenstart:	bd	6e	7c	3d	f2	b5	77	9e	0b	61	21	6e	8b	10	b6	89
nach SubBytes:	7a	9f	10	27	89	d5	f5	0b	2b	ef	fd	9f	3d	ca	4e	a7
nach ShiftRows:	7a	d5	fd	a7	89	ef	4e	27	2b	ca	10	0b	3d	9f	f5	9f
Rundenschluessel:	13	11	1d	7f	e3	94	4a	17	f3	07	a7	8b	4d	2b	30	c5
nach AddRoundKey:	69	c4	e0	d8	6a	7b	04	30	d8	cd	b7	80	70	b4	c5	5a
Ausgabeblock:	69	c4	e0	d8	6a	7b	04	30	d8	cd	b7	80	70	b4	c5	5a

Beispiel:

Eingabeblock:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Schluessel:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Anfangszustand:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Rundenschluessel:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
nach AddRoundKey:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1. Runde:																
Rundenstart:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
nach SubBytes:	63	63	63	63	63	63	63	63	63	63	63	63	63	63	63	63
nach ShiftRows:	63	63	63	63	63	63	63	63	63	63	63	63	63	63	63	63
nach MixColumns:	63	63	63	63	63	63	63	63	63	63	63	63	63	63	63	63
Rundenschluessel:	62	63	63	63	62	63	63	63	62	63	63	63	62	63	63	63
nach AddRoundKey:	01	00	00	00	01	00	00	00	01	00	00	00	01	00	00	00
2. Runde:																
Rundenstart:	01	00	00	00	01	00	00	00	01	00	00	00	01	00	00	00
nach SubBytes:	7c	63	63	63	7c	63	63	63	7c	63	63	63	7c	63	63	63
nach ShiftRows:	7c	63	63	63	7c	63	63	63	7c	63	63	63	7c	63	63	63
nach MixColumns:	5d	7c	7c	42	5d	7c	7c	42	5d	7c	7c	42	5d	7c	7c	42
Rundenschluessel:	9b	98	98	c9	f9	fb	fb	aa	9b	98	98	c9	f9	fb	fb	aa
nach AddRoundKey:	c6	e4	e4	8b	a4	87	87	e8	c6	e4	e4	8b	a4	87	87	e8
3. Runde:																
Rundenstart:	c6	e4	e4	8b	a4	87	87	e8	c6	e4	e4	8b	a4	87	87	e8
nach SubBytes:	b4	69	69	3d	49	17	17	9b	b4	69	69	3d	49	17	17	9b
nach ShiftRows:	b4	17	69	9b	49	69	17	3d	b4	17	69	9b	49	69	17	3d
nach MixColumns:	b8	ba	c7	94	03	9f	49	df	b8	ba	c7	94	03	9f	49	df
Rundenschluessel:	90	97	34	50	69	6c	cf	fa	f2	f4	57	33	0b	0f	ac	99
nach AddRoundKey:	28	2d	f3	c4	6a	f3	86	25	4a	4e	90	a7	08	90	e5	46
4. Runde:																
Rundenstart:	28	2d	f3	c4	6a	f3	86	25	4a	4e	90	a7	08	90	e5	46
nach SubBytes:	34	d8	0d	1c	02	0d	44	3f	d6	2f	60	5c	30	60	d9	5a
nach ShiftRows:	34	0d	60	5a	02	2f	d9	1c	d6	60	0d	3f	30	d8	44	5c
nach MixColumns:	45	d4	17	85	b0	30	a0	c8	25	3e	ed	72	0b	0b	84	74
Rundenschluessel:	ee	06	da	7b	87	6a	15	81	75	9e	42	b2	7e	91	ee	2b
nach AddRoundKey:	ab	d2	cd	fe	37	5a	b5	49	50	a0	af	c0	75	9a	6a	5f
5. Runde:																
Rundenstart:	ab	d2	cd	fe	37	5a	b5	49	50	a0	af	c0	75	9a	6a	5f
nach SubBytes:	62	b5	bd	bb	9a	be	d5	3b	53	e0	79	ba	9d	b8	02	cf
nach ShiftRows:	62	be	79	cf	9a	e0	02	bb	53	b8	bd	3b	9d	b5	d5	ba
nach MixColumns:	ab	41	64	e4	ad	fc	a8	3a	f3	df	c7	86	8a	32	4c	b3
Rundenschluessel:	7f	2e	2b	88	f8	44	3e	09	8d	da	7c	bb	f3	4b	92	90
nach AddRoundKey:	d4	6f	4f	6c	55	b8	96	33	7e	05	bb	3d	79	79	de	23
6. Runde:																
Rundenstart:	d4	6f	4f	6c	55	b8	96	33	7e	05	bb	3d	79	79	de	23
nach SubBytes:	48	a8	84	50	fc	6c	90	c3	f3	6b	ea	27	b6	b6	1d	26
nach ShiftRows:	48	6c	ea	26	fc	6b	1d	50	f3	b6	84	c3	b6	a8	90	27
nach MixColumns:	e8	93	81	12	13	5d	5d	c9	7b	d0	08	a1	23	71	4c	b7
Rundenschluessel:	ec	61	4b	85	14	25	75	8c	99	ff	09	37	6a	b4	9b	a7
nach AddRoundKey:	04	f2	ca	97	07	78	28	45	e2	2f	01	96	49	c5	d7	10
7. Runde:																
Rundenstart:	04	f2	ca	97	07	78	28	45	e2	2f	01	96	49	c5	d7	10
nach SubBytes:	f2	89	74	88	c5	bc	34	6e	98	15	7c	90	3b	a6	0e	ca
nach ShiftRows:	f2	bc	7c	ca	c5	15	0e	88	98	a6	74	6e	3b	89	34	90
nach MixColumns:	96	df	f3	42	28	75	4f	44	c0	3d	64	bd	52	fe	71	cb
Rundenschluessel:	21	75	17	87	35	50	62	0b	ac	af	6b	3c	c6	1b	f0	9b
nach AddRoundKey:	b7	aa	e4	c5	1d	25	2d	4f	6c	92	0f	81	94	e5	81	50
8. Runde:																
Rundenstart:	b7	aa	e4	c5	1d	25	2d	4f	6c	92	0f	81	94	e5	81	50
nach SubBytes:	a9	ac	69	a6	a4	3f	d8	84	50	4f	76	0c	22	d9	0c	53
nach ShiftRows:	a9	3f	76	53	a4	4f	0c	a6	50	d9	69	84	22	ac	d8	0c
nach MixColumns:	2d	1e	8f	0f	28	88	02	e3	3d	c6	cc	53	7f	1e	31	0a
Rundenschluessel:	0e	f9	03	33	3b	a9	61	38	97	06	0a	04	51	1d	fa	9f
nach AddRoundKey:	23	e7	8c	3c	13	21	63	db	aa	c0	c6	57	2e	03	cb	95
9. Runde:																
Rundenstart:	23	e7	8c	3c	13	21	63	db	aa	c0	c6	57	2e	03	cb	95
nach SubBytes:	26	94	64	eb	7d	fd	fb	b9	ac	ba	b4	5b	31	7b	1f	2a
nach ShiftRows:	26	fd	b4	2a	7d	ba	1f	eb	ac	7b	64	b9	31	94	fb	5b
nach MixColumns:	ce	2a	d6	77	db	d8	df	ef	13	4f	cf	99	65	4f	a5	8a
Rundenschluessel:	b1	d4	d8	e2	8a	7d	b9	da	1d	7b	b3	de	4c	66	49	41
nach AddRoundKey:	7f	fe	0e	95	51	a5	66	35	0e	34	7c	47	29	29	ec	cb
10. Runde:																
Rundenstart:	7f	fe	0e	95	51	a5	66	35	0e	34	7c	47	29	29	ec	cb
nach SubBytes:	d2	bb	ab	2a	d1	06	33	96	ab	18	10	a0	a5	a5	ce	1f
nach ShiftRows:	d2	06	10	1f	d1	18	ce	2a	ab	a5	ab	96	a5	bb	33	a0
Rundenschluessel:	b4	ef	5b	cb	3e	92	e2	11	23	e9	51	cf	6f	8f	18	8e
nach AddRoundKey:	66	e9	4b	d4	ef	8a	2c	3b	88	4c	fa	59	ca	34	2b	2e
Ausgabeblock:	66	e9	4b	d4	ef	8a	2c	3b	88	4c	fa	59	ca	34	2b	2e

10. Die Entschlüsselungsabbildung InvCipher

Wir wiederholen die Verschlüsselung: Bei vorgegebenem Schlüssel $k_0, k_1, k_2, \ldots$ wird der Eingabezustand S_0 in den Ausgabezustand S_{11} nach folgendem Verfahren überführt, wenn man für AddRoundKey$(\ldots, k, r)$

einfach AddRoundKey_r schreibt:

$$\begin{aligned} S_1 &= \text{AddRoundKey}_0(S_0), \\ S_{r+1} &= \text{AddRoundKey}_r(\text{MixColumns}(\text{ShiftRows}(\text{SubBytes}(S_r)))), 1 \le r \le 9, \\ S_{11} &= \text{AddRoundKey}_{10}(\text{ShiftRows}(\text{SubBytes}(S_{10}))). \end{aligned}$$

Wir erhalten daraus:

$$\begin{aligned} S_{10} &= \text{SubBytes}^{-1}(\text{ShiftRows}^{-1}(\text{AddRoundKey}_{10}^{-1}(S_{11}))), \\ S_r &= \text{SubBytes}^{-1}(\text{ShiftRows}^{-1}(\text{MixColumns}^{-1}(\text{AddRoundKey}_r^{-1}(S_{r+1})))), 1 \le r \le 9, \\ S_0 &= \text{AddRoundKey}_0^{-1}(S_1). \end{aligned}$$

Es ist nicht schwer, sich daraus die Entschlüsselungsfunktion InvCipher explizit zu konstruieren.

Hier sind die zugehörigen Maple-Funktionen:

```
InvShiftRows:=proc()
  local z_ein, z_aus, i;
  z_ein:=args[1];
  z_aus:=array(0..3,0..3);
  for i from 0 to 3 do
    z_aus[0,i]:=z_ein[0,i];
    z_aus[1,i]:=z_ein[1,i+3 mod 4];
    z_aus[2,i]:=z_ein[2,i+2 mod 4];
    z_aus[3,i]:=z_ein[3,i+1 mod 4];
  od;
  z_aus;
end;

#InvSubBytes:=proc()
#   map(x->invalpha(Rem((x+byte2poly(6*16+3))*(alpha+alpha^3+alpha^6),
#                       alpha^8-1,alpha) mod 2),args[1]);
#end;

InvSubBytes:=proc()
  local S, x;
  S:=[82,9,106,213,48,54,165,56,191,64,163,158,129,243,215,251,
      124,227,57,130,155,47,255,135,52,142,67,68,196,222,233,203,
      84,123,148,50,166,194,35,61,238,76,149,11,66,250,195,78,
      8,46,161,102,40,217,36,178,118,91,162,73,109,139,209,37,
      114,248,246,100,134,104,152,22,212,164,92,204,93,101,182,146,
      108,112,72,80,253,237,185,218,94,21,70,87,167,141,157,132,
      144,216,171,0,140,188,211,10,247,228,88,5,184,179,69,6,
      208,44,30,143,202,63,15,2,193,175,189,3,1,19,138,107,
      58,145,17,65,79,103,220,234,151,242,207,206,240,180,230,115,
      150,172,116,34,231,173,53,133,226,249,55,232,28,117,223,110,
      71,241,26,113,29,41,197,137,111,183,98,14,170,24,190,27,
      252,86,62,75,198,210,121,32,154,219,192,254,120,205,90,244,
      31,221,168,51,136,7,199,49,177,18,16,89,39,128,236,95,
      96,81,127,169,25,181,74,13,45,229,122,159,147,201,156,239,
      160,224,59,77,174,42,245,176,200,235,187,60,131,83,153,97,
      23,43,4,126,186,119,214,38,225,105,20,99,85,33,12,125];
  map(x->byte2poly(S[poly2byte(x)+1]),args[1]);
end;

InvMixColumns:=proc()
```

```
  local s, z, i, j, p;
  s:=args[1];
  z:=array(0..3,0..3);
  for j from 0 to 3 do
    z[0,j]:=Rem((s[2,j]+s[1,j]+s[3,j])+(s[0,j]+s[1,j])*alpha
            +(s[2,j]+s[0,j])*alpha^2
            +(s[1,j]+s[0,j]+s[3,j]+s[2,j])*alpha^3,falpha,alpha) mod 2;
    z[1,j]:=Rem((s[3,j]+s[2,j]+s[0,j])+(s[1,j]+s[2,j])*alpha
            +(s[1,j]+s[3,j])*alpha^2
            +(s[1,j]+s[0,j]+s[3,j]+s[2,j])*alpha^3,falpha,alpha) mod 2;
    z[2,j]:=Rem((s[3,j]+s[1,j]+s[0,j])+(s[2,j]+s[3,j])*alpha
            +(s[2,j]+s[0,j])*alpha^2
            +(s[1,j]+s[0,j]+s[3,j]+s[2,j])*alpha^3,falpha,alpha) mod 2;
    z[3,j]:=Rem((s[1,j]+s[0,j]+s[2,j])+(s[3,j]+s[0,j])*alpha
            +(s[1,j]+s[3,j])*alpha^2
            +(s[1,j]+s[0,j]+s[3,j]+s[2,j])*alpha^3,falpha,alpha) mod 2;
  od;
  z;
end;

InvCipher:=proc()
  local zustand, k, r, i, j;
  zustand:=array(0..3,0..3);
  for i from 0 to 3 do
    for j from 0 to 3 do
      zustand[i,j]:=byte2poly(args[1][i+4*j+1]);
    od;
  od;
  k:=args[2];

  zustand:=InvSubBytes(InvShiftRows(AddRoundKey(zustand,k,10)));
  for r from 9 by -1 to 1 do
    zustand:=InvSubBytes(InvShiftRows(InvMixColumns(AddRoundKey(zustand,k,r))));
  od;
  zustand:=AddRoundKey(zustand,k,0);
  map(poly2byte,[zustand[0,0],zustand[1,0],zustand[2,0],zustand[3,0],
                 zustand[0,1],zustand[1,1],zustand[2,1],zustand[3,1],
                 zustand[0,2],zustand[1,2],zustand[2,2],zustand[3,2],
                 zustand[0,3],zustand[1,3],zustand[2,3],zustand[3,3]]);
end;
```

11. Bemerkungen zum Aufbau von AES

Wir beschränken uns auf Beobachtungen zur algebraischen Struktur.

Bemerkung zur Funktion SubBytes**:** Die Abbildung SubBytes : $F \to F$ war wie folgt definiert:

$$\begin{aligned} \text{SubBytes}(b) = b'' \quad &\text{mit} \quad b' = \frac{1}{b} \bmod f_\alpha \text{ (für } b \neq 0) \text{ bzw. } b' = 0 \text{ (für } b = 0), \\ &\text{und} \quad b'' \equiv b'(1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4) + (1 + \alpha + \alpha^5 + \alpha^6) \bmod \alpha^8 + 1. \end{aligned}$$

(1) Mit der Multiplikation $F \times F \to F$, $(b, c) \mapsto bc \bmod f_\alpha$ wird F zu einem endlichen Körper mit 256 Elementen. Die multiplikative Gruppe hat 255 Elemente, also gilt für $b \in F \setminus \{0\}$ die Beziehung $b^{255} \equiv 1 \bmod f_\alpha$ und damit $b^{254} \equiv \frac{1}{b} \bmod f_\alpha$. Wir können also in der Definition von SubBytes auch schreiben (ohne Fallunterscheidung)

$$b' \equiv b^{254} \bmod f_\alpha.$$

Mit der square&multiply-Methode berechnet man $b^{254} \bmod f_\alpha$ wegen $254 = (11111110)_2 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2$ über

$$b^{254} \equiv b^{2^7} \cdot b^{2^6} \cdot b^{2^5} \cdot b^{2^4} \cdot b^{2^3} \cdot b^{2^2} \cdot b^{2^1} \bmod f_\alpha$$

unter Verwendung der Rekursionsformel

$$b^{2^{i+1}} \equiv \left(b^{2^i}\right)^2 \bmod f_\alpha \text{ für } i \geq 0.$$

(2) Wir betrachten die Zuordnung $T : F \to F$, $b' \mapsto b''$ mit

$$b'' \equiv b'(1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4) + (1 + \alpha + \alpha^5 + \alpha^6) \bmod \alpha^8 + 1$$

und schreiben sie als

$$T : F \to F, \quad b \mapsto mb + t \bmod f$$

mit

$$f = 1 + \alpha^8 = (1 + \alpha)^8, \quad m = 1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4, \quad t = 1 + \alpha + \alpha^5 + \alpha^6 = (1 + \alpha)^2 m.$$

Dabei ist m irreduzibel und $1 + m = \alpha(1 + \alpha)^3$. Aus $\mathrm{ggT}(m, f) = 1$ folgt, daß T invertierbar ist. Explizit erhält man

$$T^{-1}(b) \equiv (\alpha + \alpha^3 + \alpha^6)b + (1 + \alpha^2) \bmod f.$$

T ist weiterhin so gemacht, daß es keinen Fixpunkt gibt: Der Ansatz $T(b) = b$ führt über

$$T(b) = b \quad \Longrightarrow \quad (1 + m)b \equiv t \bmod f \quad \Longrightarrow \quad \alpha(1 + \alpha)^3 b \equiv (1 + \alpha)^2 m \bmod (1 + \alpha)^8$$

zu einem Widerspruch, was die Teilbarkeit durch $(1+\alpha)^3$ betrifft. Ebensowenig gibt es ein $b \in F$ mit $T(b) = \overline{b}$, wo $\overline{b}$ das bitweise Komplement von b bezeichnet, also $\overline{b} = b + (1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7) = b + (1 + \alpha)^7$.

Wir wollen nun nochmals den Aufbau der Verschlüsselungsabbildung Cipher anschauen. Zur Abkürzung schreiben wir bei gegebenem Schlüssel k

$$\mathrm{AddRoundKey}(\ldots, k, r) = \lambda_{k[r]}, \quad \mathrm{MixColumns} = \mu, \quad \mathrm{ShiftRows} = \rho, \quad \mathrm{SubBytes} = \sigma.$$

Dann wird der Anfangszustand S_0 zum Endzustand S_{11} nach folgendem Verfahren verschlüsselt:

$$\begin{aligned} S_1 &= \lambda_{k[0]}(S_0) \\ S_{r+1} &= \lambda_{k[r]}(\mu(\rho(\sigma(S_r)))), 1 \leq r \leq 9 \\ S_{11} &= \lambda_{k[10]}(\rho(\sigma(S_{10}))) \end{aligned}$$

Wir schreiben nochmals die Abbildungen auf, die angewendet werden, um aus dem Anfangszustand S_0 den Endzustand S_{11} zu erhalten. Die Reihenfolge ist dabei zeilenweise von links nach rechts und dann von oben nach unten, also anders als bei der üblichen Abbildungsnotation:

$$\mathrm{Cipher} = \begin{array}{|ccc|c|} \hline & & & \lambda_{k[0]} \\ \sigma & \rho & \mu & \lambda_{k[1]} \\ \sigma & \rho & \mu & \lambda_{k[2]} \\ \sigma & \rho & \mu & \lambda_{k[3]} \\ \sigma & \rho & \mu & \lambda_{k[4]} \\ \sigma & \rho & \mu & \lambda_{k[5]} \\ \sigma & \rho & \mu & \lambda_{k[6]} \\ \sigma & \rho & \mu & \lambda_{k[7]} \\ \sigma & \rho & \mu & \lambda_{k[8]} \\ \sigma & \rho & \mu & \lambda_{k[9]} \\ \sigma & \rho & & \lambda_{k[10]} \\ \hline \end{array}$$

Bemerkung: Bei der Verschlüsselung $S_{11} = \mathrm{Cipher}(S_0)$

$$S_0 \xrightarrow{\lambda_{k[0]}} S_1 \longrightarrow \ldots \longrightarrow S_{10} \xrightarrow{\lambda_{k[10]}} S_{11}$$

ist die erste und die letzte Operation die Addition eines Rundenschlüssels. Der Grund dafür ist, daß die anderen Operationen σ, ρ, μ allgemein bekannt sind, also bis zur Addition eines Rundenschlüssels rückgängig gemacht werden können und daher zur Sicherheit nichts beitragen.

Bemerkung: Sieht man sich das Verschlüsselungsschema an, so stellt sich die Frage, weshalb in der letzten Runde die Abbildung MixColumns $= \mu$ wegfällt. Eine Antwort wird später gegeben.

Zum Aufbau der inversen Abbildung Cipher^{-1}**:** Die Verschlüsselung erfolgt durch Anwendung der Abbildungen

$$\lambda_{k[0]}, \quad \sigma, \quad \rho, \quad \mu, \quad \lambda_{k[1]}, \quad \ldots \quad \sigma, \quad \rho, \quad \lambda_{k[10]}.$$

Bei der inversen Abbildung muß man die Schritte rückgängig machen und die jeweils inversen Operationen ausführen:

$$\lambda^{-1}_{k[10]}, \quad \rho^{-1}, \quad \sigma^{-1}, \quad \ldots \quad \lambda^{-1}_{k[1]}, \quad \mu^{-1}, \quad \rho^{-1}, \quad \sigma^{-1}, \quad \lambda^{-1}_{k[0]}.$$

In unserem Schema sieht dies also so aus:

$$\text{Cipher}^{-1} = \boxed{\begin{array}{llll}
\lambda^{-1}_{k[10]} & & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[9]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[8]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[7]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[6]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[5]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[4]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[3]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[2]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[1]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\
\lambda^{-1}_{k[0]} & & &
\end{array}}$$

Wir wollen Cipher^{-1} etwas anders darstellen und bemerken dazu folgende Eigenschaften:

(1) Wir haben bereits früher festgestellt, daß

$$\lambda^{-1}_{k[r]} = \lambda_{k[r]}$$

gilt.

(2) SubBytes $= \sigma$ operiert auf einzelnen Bytes, ShiftRows $= \rho$ ändert nur die Anordnung der Bytes eines Zustands, also kommutieren die beiden Abbildungen:

$$\sigma \circ \rho = \rho \circ \sigma \quad \text{ und damit auch } \quad \rho^{-1} \circ \sigma^{-1} = \sigma^{-1} \circ \rho^{-1},$$

d.h. die Reihenfolge von ρ^{-1} und σ^{-1} kann vertauscht werden ohne die Abbildung zu ändern.

(3) Denken wir uns $k[r]$ als die r-ten 16 Bytes des Schlüssels als Zustand angeordnet (beginnend mit $r = 0$), so ist

$$\text{AddRoundKey}(S, k, r) = \lambda_{k[r]}(S) = S + k[r].$$

Sei $\widetilde{k}$ eine 176-Byte-lange Schlüsselfolge mit $\widetilde{k}[r] = \mu^{-1}k[r]$. Dann gilt:

$$\mu^{-1}\lambda_{k[r]}(S) = \mu^{-1}(k[r] + S) = \widetilde{k}[r] + \mu^{-1}S = \lambda_{\widetilde{k}[r]}\mu^{-1}(S),$$

also $\mu^{-1}\lambda_{k[r]} = \lambda_{\widetilde{k}[r]}\mu^{-1}$.

Mit diesen Eigenschaften und der neu eingeführten Schlüsselfolge $\widetilde{k}$ können wir also schreiben:

$$\text{Cipher}^{-1} = \begin{array}{|cccc|}\hline \lambda_{k[10]} & & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[9]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[8]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[7]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[6]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[5]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[4]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[3]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[2]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[1]} & \mu^{-1} & \rho^{-1} & \sigma^{-1} \\ \lambda_{k[0]} & & & \\ \hline \end{array} = \begin{array}{|cccc|}\hline \lambda_{k[10]} & & \sigma^{-1} & \rho^{-1} \\ \mu^{-1} & \lambda_{\widetilde{k}[9]} & \sigma^{-1} & \rho^{-1} \\ \mu^{-1} & \lambda_{\widetilde{k}[8]} & \sigma^{-1} & \rho^{-1} \\ \mu^{-1} & \lambda_{\widetilde{k}[7]} & \sigma^{-1} & \rho^{-1} \\ \mu^{-1} & \lambda_{\widetilde{k}[6]} & \sigma^{-1} & \rho^{-1} \\ \mu^{-1} & \lambda_{\widetilde{k}[5]} & \sigma^{-1} & \rho^{-1} \\ \mu^{-1} & \lambda_{\widetilde{k}[4]} & \sigma^{-1} & \rho^{-1} \\ \mu^{-1} & \lambda_{\widetilde{k}[3]} & \sigma^{-1} & \rho^{-1} \\ \mu^{-1} & \lambda_{\widetilde{k}[2]} & \sigma^{-1} & \rho^{-1} \\ \mu^{-1} & \lambda_{\widetilde{k}[1]} & \sigma^{-1} & \rho^{-1} \\ \lambda_{k[0]} & & & \\ \hline \end{array} = \begin{array}{|cccc|}\hline & & & \lambda_{k[10]} \\ \sigma^{-1} & \rho^{-1} & \mu^{-1} & \lambda_{\widetilde{k}[9]} \\ \sigma^{-1} & \rho^{-1} & \mu^{-1} & \lambda_{\widetilde{k}[8]} \\ \sigma^{-1} & \rho^{-1} & \mu^{-1} & \lambda_{\widetilde{k}[7]} \\ \sigma^{-1} & \rho^{-1} & \mu^{-1} & \lambda_{\widetilde{k}[6]} \\ \sigma^{-1} & \rho^{-1} & \mu^{-1} & \lambda_{\widetilde{k}[5]} \\ \sigma^{-1} & \rho^{-1} & \mu^{-1} & \lambda_{\widetilde{k}[4]} \\ \sigma^{-1} & \rho^{-1} & \mu^{-1} & \lambda_{\widetilde{k}[3]} \\ \sigma^{-1} & \rho^{-1} & \mu^{-1} & \lambda_{\widetilde{k}[2]} \\ \sigma^{-1} & \rho^{-1} & \mu^{-1} & \lambda_{\widetilde{k}[1]} \\ \sigma^{-1} & \rho^{-1} & & \lambda_{k[0]} \\ \hline \end{array}$$

Wir sehen also, daß Cipher^{-1} nach dem gleichen Muster wie Cipher aufgebaut ist, allerdings mit etwas veränderter Schlüsselfolge und σ, ρ, μ ersetzt durch σ^{-1}, ρ^{-1}, μ^{-1}. Der gleiche strukturelle Aufbau von Cipher und Cipher^{-1} war wohl auch ein Konstruktionsziel. Dies erklärt dann auch, weshalb MixColumns $= \mu$ in der letzten Runde der Verschlüsselung fehlt: Wäre MixColumns $= \mu$ in der letzten Runde vorhanden, könnte man Cipher^{-1} nicht wie Cipher anordnen.

12. Ein Miniatur-Modell von AES

Um die Sicherheit eines Kryptosystems zu untersuchen, kann man damit beginnen, verkleinerte Modelle des Systems zu untersuchen. Wir wollen hier ein Miniatur-Modell von AES betrachten: Wir beschränken uns auf eine Runde, statt Zustände mit 4×4-Matrizen wollen wir Zustände mit 2×2-Matrizen, Eingabe, Ausgabe und Schlüssel haben dann nur 4 statt 16 Bytes.

SubBytes $= \sigma$ bleibt unverändert. ShiftRows wird durch

$$\text{ShiftRows} \begin{pmatrix} s_{00} & s_{01} \\ s_{10} & s_{11} \end{pmatrix} = \begin{pmatrix} s_{00} & s_{01} \\ s_{11} & s_{10} \end{pmatrix}$$

definiert. MixColumns $= \mu$ operiert spaltenweise durch

$$\begin{pmatrix} s_{0j} \\ s_{1j} \end{pmatrix} \mapsto \begin{pmatrix} \alpha & 1+\alpha \\ 1+\alpha & \alpha \end{pmatrix} \begin{pmatrix} s_{0j} \\ s_{1j} \end{pmatrix} \bmod f_\alpha.$$

(Die Transformationsmatrix stimmt mit ihrem Inversen überein.)

Schlüsselerweiterung: Wir starten mit k_0, k_1, k_2, k_3. Wir bilden $w_i = (k_{2i}, k_{2i+1})$ und fordern für $i \geq 2$: Ist $i \equiv 0 \bmod 2$, so sei

$$w_i = w_{i-2} + \sigma((k_{2i-1}, k_{2i-2})) + (\alpha^{\frac{i}{2}-1}, 0) \bmod f_\alpha,$$

d.h.

$$(k_{2i}, k_{2i+1}) = (k_{2i-4} + \sigma(k_{2i-1}) + \alpha^{i-1}, k_{2i-3} + \sigma(k_{2i-2})) \bmod f_\alpha.$$

Ist $i \not\equiv 0 \bmod 2$, so sei

$$w_i = w_{i-2} + w_{i-1},$$

d.h.

$$(k_{2i}, k_{2i+1}) = (k_{2i-4} + k_{2i-2}, k_{2i-3} + k_{2i-1}).$$

Wir benötigen nur $k_1, k_2, \ldots, k_7$, was dann

$$\begin{aligned} k_4 &= k_0 + \sigma(k_3) + 1 \\ k_5 &= k_1 + \sigma(k_2) \\ k_6 &= k_2 + k_4 = k_2 + k_0 + \sigma(k_3) + 1 \\ k_7 &= k_3 + k_5 = k_3 + k_1 + \sigma(k_2) \end{aligned}$$

ergibt.

Verschlüsselung: Eingabebytes x_0, x_1, x_2, x_3, Ausgabebytes y_0, y_1, y_2, y_3, Schlüsselbytes k_0, k_1, k_2, k_3.

$$\begin{pmatrix} x_0 & x_2 \\ x_1 & x_3 \end{pmatrix} \xrightarrow{\lambda_{k[0]}} \begin{pmatrix} x_0+k_0 & x_2+k_2 \\ x_1+k_1 & x_3+k_3 \end{pmatrix} \xrightarrow{\sigma} \begin{pmatrix} \sigma(x_0+k_0) & \sigma(x_2+k_2) \\ \sigma(x_1+k_1) & \sigma(x_3+k_3) \end{pmatrix}$$

$$\xrightarrow{\rho} \begin{pmatrix} \sigma(x_0+k_0) & \sigma(x_2+k_2) \\ \sigma(x_3+k_3) & \sigma(x_1+k_1) \end{pmatrix}$$

$$\xrightarrow{\mu} \begin{pmatrix} \alpha\sigma(x_0+k_0)+(1+\alpha)\sigma(x_3+k_3) & \alpha\sigma(x_2+k_2)+(1+\alpha)\sigma(x_1+k_1) \\ (1+\alpha)\sigma(x_0+k_0)+\alpha\sigma(x_3+k_3) & (1+\alpha)\sigma(x_2+k_2)+\alpha\sigma(x_1+k_1) \end{pmatrix}$$

$$\xrightarrow{\lambda_{k[1]}} \begin{pmatrix} \alpha\sigma(x_0+k_0)+(1+\alpha)\sigma(x_3+k_3)+k_4 & \alpha\sigma(x_2+k_2)+(1+\alpha)\sigma(x_1+k_1)+k_6 \\ (1+\alpha)\sigma(x_0+k_0)+\alpha\sigma(x_3+k_3)+k_5 & (1+\alpha)\sigma(x_2+k_2)+\alpha\sigma(x_1+k_1)+k_7 \end{pmatrix}$$

Daher ist

$$\begin{aligned} y_0 &\equiv \alpha\sigma(x_0+k_0)+(1+\alpha)\sigma(x_3+k_3)+k_4 \bmod f_\alpha \\ y_1 &\equiv (1+\alpha)\sigma(x_0+k_0)+\alpha\sigma(x_3+k_3)+k_5 \bmod f_\alpha \\ y_2 &\equiv \alpha\sigma(x_2+k_2)+(1+\alpha)\sigma(x_1+k_1)+k_6 \bmod f_\alpha \\ y_3 &\equiv (1+\alpha)\sigma(x_2+k_2)+\alpha\sigma(x_1+k_1)+k_7 \bmod f_\alpha \end{aligned}$$

oder mit ausgeschriebenen Schlüsselbytes:

$$\begin{aligned} y_0 &\equiv \alpha\sigma(x_0+k_0)+(1+\alpha)\sigma(x_3+k_3)+k_0+\sigma(k_3)+1 \bmod f_\alpha \\ y_1 &\equiv (1+\alpha)\sigma(x_0+k_0)+\alpha\sigma(x_3+k_3)+k_1+\sigma(k_2) \bmod f_\alpha \\ y_2 &\equiv \alpha\sigma(x_2+k_2)+(1+\alpha)\sigma(x_1+k_1)+k_2+k_0+\sigma(k_3)+1 \bmod f_\alpha \\ y_3 &\equiv (1+\alpha)\sigma(x_2+k_2)+\alpha\sigma(x_1+k_1)+k_3+k_1+\sigma(k_2) \bmod f_\alpha \end{aligned}$$

Frage: Wie sicher ist dieses Verschlüsselungsverfahren?

Known-plaintext-attack: Wir nehmen an, wir kennen Eingabebytes x_0, x_1, x_2, x_3 und die zugehörigen verschlüsselten Bytes y_0, y_1, y_2, y_3. Wir wollen daraus die Schlüsselbytes k_0, k_1, k_2, k_3 berechnen.
Wir haben 4 Gleichungen und 4 Unbekannte k_0, k_1, k_2, k_3. Was kann man machen?

(1) Man kann alle möglichen $256^4 = 2^{32} = 4294967296$ Schlüssel durchprobieren.

(2) Wir betrachten obige Gleichungen. Wir wählen k_1, k_2 und berechnen damit (aus den beiden letzten Gleichungen)

$$\begin{aligned} k_3 &\equiv y_3+(1+\alpha)\sigma(x_2+k_2)+\alpha\sigma(x_1+k_1)+k_1+\sigma(k_2) \bmod f_\alpha \\ k_0 &\equiv y_2+\alpha\sigma(x_2+k_2)+(1+\alpha)\sigma(x_1+k_1)+k_2+\sigma(k_3)+1 \bmod f_\alpha \end{aligned}$$

Nun überprüfen wir, ob (die beiden ersten Gleichungen)

$$\begin{aligned} y_0 &\equiv \alpha\sigma(x_0+k_0)+(1+\alpha)\sigma(x_3+k_3)+k_0+\sigma(k_3)+1 \bmod f_\alpha \\ y_1 &\equiv (1+\alpha)\sigma(x_0+k_0)+\alpha\sigma(x_3+k_3)+k_1+\sigma(k_2) \bmod f_\alpha \end{aligned}$$

erfüllt sind. Wenn ja, haben wir einen möglichen Schlüssel gefunden. Wir müssen bei diesem Verfahren nur $256^2 = 2^{16} = 65536$ mögliche Werte durchprobieren.

(3) Frage: Kann man aus obigen Gleichungen noch mehr Information holen um weniger probieren zu müssen?

13. Anhang

Literatur:

- U.S. Department of Commerce / National Institute of Standards and Technology, Draft FIPS PUB Advanced Encryption Standard (AES), 2001.

KAPITEL 3

Kettenbrüche

1. Definition

Unter einem Kettenbruch versteht man einen Ausdruck der Form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \dots}}}}$$

Aus schreibtechnischen Gründen ist dafür auch die Bezeichnung

$$[a_0, a_1, a_2, a_3, a_4, \dots]$$

gebräuchlich.

Beispiel:

$$[2,3,5,7] = 2 + \cfrac{1}{3 + \cfrac{1}{5 + \cfrac{1}{7}}} = 2 + \cfrac{1}{3 + \cfrac{1}{\cfrac{36}{7}}} = 2 + \cfrac{1}{3 + \cfrac{7}{36}} = 2 + \cfrac{1}{\cfrac{115}{36}} = 2 + \frac{36}{115} = \frac{266}{115}$$

Kettenbruchalgorithmus zur Gewinnung der Kettenbruchentwicklung einer reellen Zahl: Ist $\alpha = \alpha_0 \in \mathbf{R}$ gegeben, so definiert man rekursiv

$$a_i = \lfloor \alpha_i \rfloor \quad \text{und} \quad \alpha_{i+1} = \frac{1}{\alpha_i - a_i}, \text{ solange } \alpha_i \notin \mathbf{Z}.$$

Dann gilt

$$\alpha_i = a_i + \frac{1}{\alpha_{i+1}}$$

und damit

$$\alpha = \alpha_0 = a_0 + \frac{1}{\alpha_1} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\alpha_2}} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\alpha_3}}} = \dots = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ddots \cfrac{\ddots}{a_{n-1} + \cfrac{1}{\alpha_n}}}}$$

bzw. mit der einfacheren Notation

$$\alpha = [\alpha_0] = [a_0, \alpha_1] = [a_0, a_1 + \frac{1}{\alpha_2}] = [a_0, a_1, \alpha_2] = \dots = [a_0, a_1, a_2, \dots, a_{n-1}, \alpha_n].$$

$[a_0, a_1, a_2, \dots]$ heißt die Kettenbruchentwicklung von α, a_n der n-te Teilquotient von α.
Außerdem sieht man auch, daß für $m \leq n$ gilt

$$\alpha_m = [a_m, a_{m+1}, \dots, a_{n-1}, \alpha_n].$$

Die Kettenbruchentwicklung bricht ab, wenn es ein n gibt mit $\alpha_n \in \mathbf{Z}$. Dann ist $\alpha = [a_0, a_1, \dots, a_n]$.

Beispiele: Mit dem Kettenbruchalgorithmus erhalten wir folgende Kettenbruchentwicklungen:

(1)
$$\frac{7}{5} = 1 + \frac{2}{5} = 1 + \frac{1}{\frac{5}{2}} = 1 + \frac{1}{2 + \frac{1}{2}} = [1, 2, 2].$$

(2)
$$\frac{21}{13} = [\frac{21}{13}] = [1, \frac{13}{8}] = [1, 1, \frac{8}{5}] = [1, 1, 1, \frac{5}{3}] = [1, 1, 1, 1, \frac{3}{2}] = [1, 1, 1, 1, 1, 2].$$

(3) Zunächst ist
$$\sqrt{2} = 1 + \frac{1}{1 + \sqrt{2}} = [1, 1 + \sqrt{2}] \quad \text{und} \quad 1 + \sqrt{2} = 2 + \frac{1}{1 + \sqrt{2}} = [2, 1 + \sqrt{2}],$$
was durch Iteration
$$\sqrt{2} = [1, 1 + \sqrt{2}] = [1, 2, 1 + \sqrt{2}] = [1, 2, 2, 1 + \sqrt{2}] = \cdots = [1, 2, 2, \ldots, 2, 2, 1 + \sqrt{2}]$$
ergibt.

(4) Für $\alpha = \frac{1+\sqrt{5}}{2}$ gilt $\lfloor \alpha \rfloor = 1$ und $\alpha = 1 + \frac{1}{\alpha}$, was sofort zu
$$\alpha = [1, \alpha] = [1, 1, \alpha] = [1, 1, 1, \alpha] = \cdots = [1, 1, 1, \ldots, 1, \alpha]$$
führt.

(5) Für π und e findet man die Kettenbruchentwicklungen
$$\pi = [3, 7, 15, 1, 292, 1, 1, 1, 2, 1, \ldots] \quad \text{und} \quad e = [2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, \ldots].$$

Bemerkung: Bricht die Kettenbruchentwicklung einer reellen Zahl α ab, so ist $\alpha = [a_0, a_1, a_2, \ldots, a_n] \in \mathbf{Q}$ wegen $a_i \in \mathbf{Z}$. Die Umkehrung wird noch bewiesen.

LEMMA. *Sei $[a_0, a_1, a_2, \ldots]$ die Kettenbruchentwicklung der reellen Zahl α. Existiert a_n für ein $n \geq 1$, so gilt $\alpha_n > 1$ und $a_n \geq 1$.*

Beweis: Da a_n existiert, ist nach Konstruktion $\alpha_{n-1} \notin \mathbf{Z}$ und damit $a_{n-1} < \alpha_{n-1} < a_{n-1} + 1$. Es folgt $0 < \alpha_{n-1} - a_{n-1} < 1$, also $\alpha_n = \frac{1}{\alpha_{n-1} - a_{n-1}} > 1$ und damit auch $a_n = \lfloor \alpha_n \rfloor \geq 1$. ■

2. Wie berechnet man die Kettenbruchentwicklung?

Wir wollen jetzt die Kettenbruchentwicklung rationaler Zahlen näher anschauen.

SATZ. *Sei $\alpha = \frac{b_0}{b_1} \in \mathbf{Q}$ mit $b_0, b_1 \in \mathbf{Z}$ und $b_1 \geq 1$. Wir wenden den euklidischen Algorithmus auf b_0 und b_1 an und erhalten rekursiv $b_2, b_3, \ldots, b_{n+1}$, indem wir b_i durch b_{i+1} teilen (mit Quotient a_i und Rest b_{i+2}):*

$$\begin{aligned}
b_0 &= a_0 b_1 + b_2 \quad \textit{mit} \quad 0 < b_2 < b_1 \\
b_1 &= a_1 b_2 + b_3 \quad \textit{mit} \quad 0 < b_3 < b_2 \\
&\vdots \qquad\qquad \vdots \\
b_{n-1} &= a_{n-1} b_n + b_{n+1} \quad \textit{mit} \quad 0 < b_{n+1} < b_n \\
b_n &= a_n b_{n+1} + 0 \quad \textit{und} \quad b_{n+2} = 0
\end{aligned}$$

Dann ist
$$\alpha = [a_0, a_1, \ldots, a_n]$$
die Kettenbruchentwicklung von α. Insbesondere bricht die Kettenbruchentwicklung einer rationalen Zahl nach endlich vielen Schritten ab.

Beweis: Wir zeigen durch Induktion: Die in der Kettenbruchentwicklung von α auftretenden Teilquotienten sind genau die a_i's und $\alpha_i = \frac{b_i}{b_{i+1}}$.
Zum Induktionsanfang: Es gilt

$$\alpha = \alpha_0 = \frac{b_0}{b_1} = \frac{a_0 b_1 + b_2}{b_1} = a_0 + \frac{b_2}{b_1}.$$

Wegen $0 \le b_2 < b_1$ ist $0 \le \frac{b_2}{b_1} < 1$ und daher $a_0 = \lfloor \alpha_0 \rfloor$. Damit wird $\alpha_1 = \frac{b_1}{b_2}$.
Es gelte nun $\alpha_i = \frac{b_i}{b_{i+1}}$. Dann ist

$$\alpha_i = \frac{b_i}{b_{i+1}} = \frac{a_i b_{i+1} + b_{i+2}}{b_{i+1}} = a_i + \frac{b_{i+2}}{b_{i+1}}.$$

Wegen $0 \le b_{i+2} < b_{i+1}$ folgt $a_i = \lfloor \alpha_i \rfloor$, d.h. a_i ist der i-te Teilquotient. Damit wird

$$\alpha_{i+1} = \frac{1}{\alpha_i - a_i} = \frac{b_{i+1}}{b_{i+2}},$$

was die Induktionsbehauptung zeigt.
Nach Konstruktion ist

$$\alpha_n = \frac{b_n}{b_{n+1}} = a_n \in \mathbf{Z},$$

so daß der Kettenbruchalgorithmus hier abbricht. Dies beweist die Behauptung. ■

Die folgende Maple-Funktion bestimmt die Kettenbruchentwicklung einer rationalen Zahl:

```
# Kettenbruchentwicklung einer rationalen Zahl
kbe:=proc()
  b0:=numer(args[1]); b1:=denom(args[1]); k:=[];
  while b1>0 do
    b2:=b0 mod b1; a:=(b0-b2)/b1;
    k:=[op(k),a];
    b0:=b1; b1:=b2;
  od;
  k;
end;
```

Bemerkung: Ist $[a_0, a_1, \ldots, a_n]$ die Kettenbruchentwicklung einer rationalen Zahl und $n \ge 1$, so gilt $a_n \ge 2$, denn wir hatten früher bereits gesehen, daß $\alpha_n > 1$ gilt, und hier ist $a_n = \alpha_n$.

Die Länge der Kettenbruchentwicklung einer rationalen Zahl läßt sich mit folgendem Satz abschätzen.

SATZ. *Ist* $[a_0, a_1, \ldots, a_n]$ *die Kettenbruchentwicklung der rationalen Zahl* $\frac{b_0}{b_1}$, *so gelten die Abschätzungen*

$$n \le 4.785 \log_{10} b_1 \quad \textit{und} \quad n \le 1.45 \log_2 b_1.$$

Beweis:

(1) Die Kettenbruchentwicklung $\frac{b_0}{b_1} = [a_0, a_1, \ldots, a_n]$ erhalten wir mit dem euklidischen Algorithmus:

$$\begin{aligned} b_0 &= a_0 b_1 + b_2 \\ b_1 &= a_1 b_2 + b_3 \\ &\vdots \qquad\quad \vdots \\ b_{n-1} &= a_{n-1} b_n + b_{n+1} \\ b_n &= a_n b_{n+1} \end{aligned}$$

(2) Die Fibonacci-Folge f_i wird definiert durch $f_0 = 0$, $f_1 = 1$ und $f_i = f_{i-1} + f_{i-2}$ für $i \ge 2$, also

$$f_0 = 0, \quad f_1 = 1, \quad f_2 = 1, \quad f_3 = 2, \quad f_4 = 3, \quad f_5 = 5, \quad f_6 = 8, \quad \ldots$$

Wir beweisen durch Induktion, daß gilt

$$b_{n+3-i} \ge f_i \quad \text{für} \quad 2 \le i \le n+2.$$

(a) Für $i = 2$ ist $b_{n+1} \geq 1 = f_2$.
(b) Wir haben bereits bemerkt, daß $a_n \geq 2$ gilt, was für $i = 3$ die Aussage

$$b_n = a_n b_{n+1} \geq a_n \geq 2 = f_3$$

liefert.
(c) Sei nun $i \geq 4$. Dann folgt durch Induktion

$$b_{n+3-i} = a_{n+3-i} b_{n+3-(i-1)} + b_{n+3-(i-2)} \geq 1 \cdot f_{i-1} + f_{i-2} = f_i.$$

Setzt man $i = n + 2$ in die Formel ein, erhält man

$$b_1 \geq f_{n+2}.$$

(3) Für die Fibonacci-Folge hat man die Darstellung

$$f_n = \frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n] \approx 0.45 \cdot [1.62^n - (-0.62)^n],$$

woraus man erhält

$$\begin{aligned} f_{n+2} &= (\frac{1+\sqrt{5}}{2})^n \left[(\frac{1}{2} + \frac{3}{10}\sqrt{5}) + (-1)^{n+1}(-\frac{1}{2} + \frac{3}{10}\sqrt{5}) \cdot (\frac{3}{2} - \frac{1}{2}\sqrt{5})^n\right] \\ &\approx 1.62^n \cdot [1.17 + (-1)^{n+1} \cdot 0.17 \cdot 0.38^n]. \end{aligned}$$

Daraus ergibt sich sofort die Abschätzung

$$f_{n+2} \geq (\frac{1+\sqrt{5}}{2})^n \quad \text{und somit} \quad n \leq \frac{\log(f_{n+2})}{\log(\frac{1+\sqrt{5}}{2})},$$

was die Abschätzungen

$$n \leq 4.785 \log_{10} f_{n+2} \quad \text{und} \quad n \leq 1.45 \log_2 f_{n+2}$$

liefert.
(4) Mit $f_{n+2} \leq b_1$ folgt nun

$$n \leq 4.785 \log_{10} b_1 \quad \text{und} \quad n \leq 1.45 \log_2 b_1,$$

wie behauptet. ■

Der folgende Satz zeigt, daß die Abschätzung des letzten Satzes nicht (wesentlich) verbessert werden kann.

SATZ. *Sei f_n die durch $f_0 = 1$, $f_1 = 1$, $f_n = f_{n-1} + f_{n-2}$ (für $n \geq 2$) definierte Fibonacci-Folge, also*

$$f_0 = 0, \quad f_1 = 1, \quad f_2 = 1, \quad f_3 = 2, \quad f_4 = 3, \quad f_5 = 5, \quad f_6 = 8, \quad f_7 = 13, \quad \ldots$$

Dann ist die Kettenbruchentwicklung von $\frac{f_{n+3}}{f_{n+2}}$

$$\frac{f_{n+3}}{f_{n+2}} = [1, 1, \ldots, 1, 1, 2] \quad \textit{mit } n \textit{ Einsen},$$

also

$$\frac{f_3}{f_2} = 2 = [2], \quad \frac{f_4}{f_3} = \frac{3}{2} = [1,2], \quad \frac{f_5}{f_4} = \frac{5}{3} = [1,1,2], \quad \frac{f_6}{f_5} = \frac{8}{5} = [1,1,1,2], \quad \frac{f_7}{f_6} = \frac{13}{8} = [1,1,1,1,2], \quad \ldots$$

Es gilt

$$\frac{1}{\log(\frac{1+\sqrt{5}}{2})} \log f_{n+2} = n + \frac{\frac{1}{2} + \frac{3}{10}\sqrt{5}}{\frac{1}{2} + \frac{1}{2}\sqrt{5}} + o(1),$$

also

$$4.785 \log_{10} f_{n+2} \approx n + 0.328.$$

Beweis: Wir beweisen dies durch Induktion nach n, wobei wir die Fälle $n = 0, 1, 2, 3, 4$ bereits explizit angegeben haben. Aus $f_{n+3} = f_{n+2} + f_{n+1}$ und $f_{n+1} < f_{n+2}$ (für $n \geq 1$) folgt

$$\frac{f_{n+3}}{f_{n+2}} = 1 + \frac{f_{n+1}}{f_{n+2}}, \quad \text{also} \quad \lfloor \frac{f_{n+3}}{f_{n+2}} \rfloor = 1$$

und daher durch Induktion

$$\frac{f_{n+3}}{f_{n+2}} = 1 + \frac{f_{n+1}}{f_{n+2}} = 1 + \frac{1}{\frac{f_{n+2}}{f_{n+1}}} = 1 + \frac{1}{[1, 1, \ldots, 1, 2]} = [1, 1, 1, \ldots, 1, 2],$$

was die Behauptung beweist. ■

Maple berechnet Kettenbrüche mit der Funktion numtheory[cfrac]. Allerdings ist von vorne herein nicht klar, wieviele Kettenbruchstellen dabei gültig sind.
Will man die Kettenbruchentwicklung einer reellen Zahl bestimmen, kann man sie durch rationale Zahlen approximieren. Der nächste Satz macht eine Aussage über die Anzahl der Kettenbruchstellen, die man dabei erhält:

SATZ. *Seien $\beta < \gamma$ rationale Zahlen mit den Kettenbruchentwicklungen*

$$\beta = [b_0, b_1, b_2, \ldots] \quad \textit{und} \quad \gamma = [c_0, c_1, c_2, \ldots].$$

Gilt dann

$$b_0 = c_0, \quad b_1 = c_1, \quad \ldots \quad b_n = c_n, \quad b_n \neq c_n,$$

dann beginnt die Kettenbruchentwicklung aller reellen Zahlen α im Intervall $\beta \leq \alpha \leq \gamma$ mit

$$\alpha = [a_0, \ldots, a_n, a_{n+1}, \ldots],$$

wobei

$$a_0 = b_0 = c_0, \ldots, a_n = b_n = c_n \quad \textit{und} \quad \min(b_{n+1}, c_{n+1}) \leq a_{n+1} \leq \max(b_{n+1}, c_{n+1})$$

gilt. (Hat die Kettenbruchentwicklung von β bzw. γ nur Länge $n+1$, so sei $b_{n+1} = \infty$ bzw. $c_{n+1} = \infty$.)

Beweis: Wir betrachten für reelle $x > 0$ die reellwertige Funktion

$$f(x) = [a_0, a_1, \ldots, a_n, x].$$

Es ist

$$f([b_{n+1}, b_{n+2}, \ldots]) = \beta,$$
$$f([c_{n+1}, c_{n+2}, \ldots]) = \gamma.$$

Da $f(x)$ offensichtlich stetig ist, gibt es nach dem Zwischenwertsatz (für stetige Funktionen) eine reelle Zahl α' mit

$$f(\alpha') = \alpha$$

und

$$[b_{n+1}, b_{n+2}, \ldots] \leq \alpha' \leq [c_{n+1}, c_{n+2}, \ldots] \quad \text{oder} \quad [c_{n+1}, c_{n+2}, \ldots] \leq \alpha' \leq [b_{n+1}, b_{n+2}, \ldots].$$

Wir bilden die Kettenbruchentwicklung von α':

$$\alpha' = [a_{n+1}, a_{n+2}, \ldots].$$

Also ist

$$\alpha = [a_0, a_1, \ldots, a_n, a_{n+1}, a_{n+2}, \ldots].$$

Außerdem liegt a_{n+1} zwischen b_{n+1} und c_{n+1}. ■

Das folgende Maple-Programm bestimmt den gemeinsamen Aneil der Kettenbruchentwicklungen der Zahlen eines Intervalls:

```
kbe_gemeinsam:=proc()
  a:=kbe(args[1]); b:=kbe(args[2]);
  k:=[]; i:=1;
  while i<=nops(a) and i<=nops(b) and a[i]=b[i] do
    k:=[op(k),a[i]]; i:=i+1;
  od;
  if i>nops(a) then al:=b[i]; ar:=infinity;
  elif i>nops(b) then al:=a[i]; ar:=infinity;
  else al:=min(a[i],b[i]); ar:=max(a[i],b[i]);
  fi;
  printf("Gemeinsame Teilquotienten: %a\n",k);
  printf("Der naechste Teilquotient liegt zwischen %a und %a\n",
         al,ar);
end;
```

Beispiel: Maple liefert π mit 10 Nachkommastellen als 3.1415926536. Daher ist $\beta < \pi < \gamma$ mit

$$\beta = 3.14159265355 \quad \text{und} \quad \gamma = 3.14159265365.$$

$$\begin{aligned} \beta &= [3,7,15,1,292,1,1,2,3,1,19,7,1,1,1,1,4,1,1,3] \\ \gamma &= [3,7,15,1,292,1,2,2,1,10,1,7,1,2,21,1,8] \end{aligned}$$

Also ist

$$\pi = [3,7,15,1,292,1,a_6,\dots] \quad \text{mit} \quad 1 \le a_6 \le 2.$$

3. Eindeutigkeit

Das folgende Lemma zeigt, daß man eine rationale Zahl auf verschiedene Weise als Kettenbruch darstellen kann:

LEMMA. *Seien $a_0, a_1, \dots, a_n \in \mathbf{Z}$ und $a_1, \dots, a_n \ge 1$. Dann ist*

$$[a_0, a_1, \dots, a_n] = [a_0, a_1, \dots, a_n - 1, 1].$$

Beweis: Dies folgt sofort aus der Identität $a_n = (a_n - 1) + \frac{1}{1}$. ■

Andererseits gilt folgende Aussage:

LEMMA. *Ist*

$$\alpha = [a_0, a_1, \dots, a_{n-1}, \alpha_n] = [b_0, b_1, \dots, b_{n-1}, \beta_n]$$

mit $a_i, b_i \in \mathbf{Z}$, $a_i, b_i \ge 1$ für $i \ge 1$, $\alpha_n, \beta_n > 1$, so gilt $a_i = b_i$ für alle i und $\alpha_n = \beta_n$.

Beweis: Wir zeigen die Aussage durch Induktion nach n. Für $n = 1$ haben wir $\alpha = a_0 + \frac{1}{\alpha_1} = b_0 + \frac{1}{\beta_1}$. Wegen $\alpha_1, \beta_1 > 1$ folgt $a_0 = b_0 = \lfloor \alpha \rfloor$ und damit auch $\alpha_1 = \beta_1$. Sei jetzt $n > 1$. Setzt man $\alpha_{n-1} = a_{n-1} + \frac{1}{a_n}$ und $\beta_{n-1} = b_{n-1} + \frac{1}{b_n}$, so gilt

$$[a_0, \dots, a_{n-2}, \alpha_{n-1}] = [a_0, \dots, a_{n-2}, a_{n-1}, \alpha_n] = [b_0, \dots, b_{n-2}, b_{n-1}, \beta_n] = [b_0, \dots, b_{n-2}, \beta_{n-1}].$$

Wegen $\alpha_{n-1}, \beta_{n-1} > 1$ und der Induktionsvoraussetzung folgt $a_i = b_i$ für $0 \le i \le n-2$. Schließlich folgt aus $\alpha_{n-1} = \beta_{n-1}$ mit dem Fall $n = 1$ die Aussage $a_{n-1} = b_{n-1}$ und $\alpha_n = \beta_n$. ■

Die Kettenbruchdarstellung einer reellen Zahl ist also eindeutig bestimmt, wenn man bei rationalen Zahlen noch fordert, daß der letzte Teilquotient ≥ 2 ist.

4. Nährungsbrüche

DEFINITION. Ist $[a_0, a_1, a_2, \dots]$ die Kettenbruchentwicklung einer reellen Zahl α, so heißt

$$[a_0, a_1, \dots, a_n]$$

der n-te Näherungsbruch an α. Die Näherungsbrüche sind also

$$[a_0], \quad [a_0, a_1], \quad [a_0, a_1, a_2], \quad [a_0, a_1, a_2, a_3], \quad \dots$$

Beispiel: Die Zahl $\alpha = \frac{2963}{1281}$ hat die Kettenbruchentwicklung $\alpha = [2,3,5,7,11]$. Die Näherungsbrüche sind:

$$[2] = 2 = 2.000000000$$
$$[2,3] = \frac{7}{3} = 2.333333333$$
$$[2,3,5] = \frac{37}{16} = 2.312500000$$
$$[2,3,5,7] = \frac{266}{115} = 2.313043478$$
$$[2,3,5,7,11] = \frac{2963}{1281} = 2.313036690$$

SATZ. *Seien $a_0, a_1, a_2, \ldots$ gegeben. Dann gilt mit den Rekursionsformeln*

$$p_{-2} = 0, \quad p_{-1} = 1, \quad p_n = a_n p_{n-1} + p_{n-2} \quad \text{für } n \geq 0,$$
$$q_{-2} = 1, \quad q_{-1} = 0, \quad q_n = a_n q_{n-1} + q_{n-2} \quad \text{für } n \geq 0$$

die Gleichung

$$[a_0, a_1, \ldots, a_n] = \frac{p_n}{q_n}.$$

Beweis: Wir beweisen dies durch Induktion. Für $n = 0$ ist $p_0 = a_0$, $q_0 = 1$ und damit $\frac{p_0}{q_0} = \frac{a_0}{1} = a_0 = [a_0]$, für $n = 1$ ist $p_1 = a_1 a_0 + 1$, $q_1 = a_1$ und damit $\frac{p_1}{q_1} = \frac{a_1 a_0 + 1}{a_1} = a_0 + \frac{1}{a_1} = [a_0, a_1]$. Sei also jetzt $n \geq 2$. Wir wenden die Induktionsvoraussetzung auf $[a_0, a_1, \ldots, a_{n-2}, a_{n-1} + \frac{1}{a_n}]$ mit den Näherungsbrüchen $\frac{p_0}{q_0}, \ldots, \frac{p_{n-2}}{q_{n-2}}, \frac{\widetilde{p}_{n-1}}{\widetilde{q}_{n-1}}$ an und erhalten

$$[a_0, \ldots, a_n] = [a_0, \ldots, a_{n-2}, a_{n-1} + \frac{1}{a_n}] = \frac{\widetilde{p}_{n-1}}{\widetilde{q}_{n-1}} = \frac{(a_{n-1} + \frac{1}{a_n})p_{n-2} + p_{n-3}}{(a_{n-1} + \frac{1}{a_n})q_{n-2} + q_{n-3}} =$$
$$= \frac{a_n(a_{n-1}p_{n-2} + p_{n-3}) + p_{n-2}}{a_n(a_{n-1}q_{n-2} + q_{n-3}) + q_{n-2}} = \frac{a_n p_{n-1} + p_{n-2}}{a_n q_{n-1} + q_{n-2}} = \frac{p_n}{q_n},$$

was gezeigt werden sollte. ■

Die angegebenen Rekursionsformeln lassen eine leichte Berechnung der Näherungsbrüche zu, wie folgendes Maple-Programm zeigt:

```
# Kettenbruchentwicklung einer rationalen Zahl mit Naeherungsbruechen
kbe_nah:=proc()
  b0:=numer(args[1]); b1:=denom(args[1]); aa:=[]; pp:=[]; qq:=[];
  p_2:=0; p_1:=1;
  q_2:=1; q_1:=0;
  while b1>0 do
    b2:=b0 mod b1; a:=(b0-b2)/b1; b0:=b1; b1:=b2; aa:=[op(aa),a];
    p:=a*p_1+p_2; p_2:=p_1; p_1:=p; pp:=[op(pp),p];
    q:=a*q_1+q_2; q_2:=q_1; q_1:=q; qq:=[op(qq),q];
  od;
  [aa,pp,qq]
end;
```

SATZ. *Gegeben sei ein Kettenbruch $[a_0, a_1, a_2, \ldots]$ und dazu die durch obige Rekursionsformeln definierten Größen p_n, q_n.*

(1) *Für $n \geq 1$ gilt*

$$p_n q_{n-1} - p_{n-1} q_n = (-1)^{n-1} \quad \textit{d.h.} \quad \frac{p_n}{q_n} - \frac{p_{n-1}}{q_{n-1}} = \frac{(-1)^{n-1}}{q_{n-1} q_n}.$$

(2) *Für $n \geq 2$ gilt*

$$p_n q_{n-2} - p_{n-2} q_n = (-1)^n a_n \quad \textit{d.h.} \quad \frac{p_n}{q_n} - \frac{p_{n-2}}{q_{n-2}} = \frac{(-1)^n a_n}{q_{n-2} q_n}.$$

(3) *Ist* $\alpha = [a_0, a_1, \ldots, a_n, \alpha_{n+1}]$, *so gilt für* $n \geq 0$

$$\alpha - \frac{p_n}{q_n} = \frac{(-1)^n}{q_n(\alpha_{n+1}q_n + q_{n-1})}.$$

Beweis: 1. Für $n = 1$ gilt wegen $p_0 = a_0$, $q_0 = 1$, $p_1 = a_0a_1 + 1$, $q_1 = a_1$ die Beziehung $p_1q_0 - p_0q_1 = 1$. Für $n \geq 2$ folgt dann die Behauptung induktiv aus

$$p_nq_{n-1} - p_{n-1}q_n = (a_np_{n-1} + p_{n-2})q_{n-1} - p_{n-1}(a_nq_{n-1} + q_{n-2}) = -(p_{n-1}q_{n-2} - p_{n-2}q_{n-1}).$$

2. Mit der Gleichung aus 1. erhält man

$$p_nq_{n-2} - p_{n-2}q_n = (a_np_{n-1} + p_{n-2})q_{n-2} - p_{n-2}(a_nq_{n-1} + q_{n-2}) = a_n(p_{n-1}q_{n-2} - p_{n-2}q_{n-1}) = (-1)^n a_n,$$

wie behauptet.

3. Sind $\frac{\widetilde{p}_i}{\widetilde{q}_i}$ die Näherungsbrüche des Kettenbruchs $[a_0, a_1, \ldots, a_n, \alpha_{n+1}]$, so gilt

$$\frac{p_0}{q_0} = \frac{\widetilde{p}_0}{\widetilde{q}_0}, \quad \frac{p_1}{q_1} = \frac{\widetilde{p}_1}{\widetilde{q}_1}, \quad \ldots, \quad \frac{p_n}{q_n} = \frac{\widetilde{p}_n}{\widetilde{q}_n}, \quad \alpha = \frac{\widetilde{p}_{n+1}}{\widetilde{q}_{n+1}}.$$

Aus der Rekursionformel $\widetilde{q}_{n+1} = \alpha_{n+1}q_n + q_{n-1}$ folgt mit 1.

$$\alpha - \frac{p_n}{q_n} = \frac{\widetilde{p}_{n+1}}{\widetilde{q}_{n+1}} - \frac{\widetilde{p}_n}{\widetilde{q}_n} = \frac{(-1)^n}{\widetilde{q}_{n+1}q_n} = \frac{(-1)^n}{q_n(\alpha_{n+1}q_n + q_{n-1})},$$

wie behauptet. ■

Der folgende Satz stellt einige der wichtigsten Approximationseigenschaften von Kettenbrüchen zusammen.

SATZ. *Sei* $[a_0, a_1, a_2, \ldots]$ *die Kettenbruchentwicklung der reellen Zahl* α *mit den Näherungsbrüchen* $\frac{p_n}{q_n}$. *Dann gilt:*

(1)

$$1 = q_0 \leq q_1 < q_2 < q_3 < \ldots \quad \text{und} \quad q_n \geq (\frac{1+\sqrt{5}}{2})^{n-1} > 1.618^{n-1} \text{ für alle } n \geq 0.$$

(2) *Die Näherungsbrüche* $\frac{p_n}{q_n}$ *sind gekürzt, d.h.* $\mathrm{ggT}(p_n, q_n) = 1$.

(3)

$$\frac{p_0}{q_0} < \frac{p_2}{q_2} < \cdots \leq \alpha \leq \cdots < \frac{p_3}{q_3} < \frac{p_1}{q_1}$$

(4) *Ist* $\alpha \neq \frac{p_n}{q_n}$, *so gilt*

$$\frac{1}{2q_nq_{n+1}} \leq \frac{1}{q_n(q_{n+1} + q_n)} < \left|\alpha - \frac{p_n}{q_n}\right| \leq \frac{1}{q_nq_{n+1}}.$$

(5) *Für* $n \geq 0$ *gilt*

$$\left|\alpha - \frac{p_n}{q_n}\right| < \frac{1}{q_n^2} < 0.4^{n-1}.$$

(6) *Ist* $\alpha \notin \mathbf{Q}$, *so gilt*

$$\alpha = \lim_{n\to\infty} \frac{p_n}{q_n},$$

was man oft abkürzend auch als

$$\alpha = [a_0, a_1, a_2, \ldots]$$

schreibt.

(Wird eine Größe wie a_n, p_n *oder* q_n *benutzt, soll dies implizit voraussetzen, daß die Größen auch definiert sind.)*

Beweis:

(1) Wir haben bereits bemerkt, daß $a_i \geq 1$ für $i \geq 1$ gilt. Für $\lambda = \frac{1+\sqrt{5}}{2} \geq 1.618$ gilt $\lambda^2 = \lambda + 1$. Damit erhalten wir $q_0 = 1 > \lambda^{-1}$, $q_1 = a_1 \geq 1 = \lambda^0$ und durch Induktion für $n \geq 2$

$$q_n = a_nq_{n-1} + q_{n-2} \geq q_{n-1} + q_{n-2} \geq \lambda^{n-2} + \lambda^{n-3} = \lambda^{n-3}(\lambda + 1) = \lambda^{n-3} \cdot \lambda^2 = \lambda^{n-1},$$

wie behauptet.

(2) Aus $p_nq_{n-1} - p_{n-1}q_n = (-1)^{n-1}$ folgt $\mathrm{ggT}(p_n, q_n) = 1$.

(3) Die Formel

$$\frac{p_n}{q_n} - \frac{p_{n-2}}{q_{n-2}} = \frac{(-1)^n a_n}{q_{n-2}q_n}$$

liefert (mit $a_n \geq 1$ für $n \geq 1$) durch Einsetzen von $n = 2m$ bzw. $n = 2m+1$

$$\frac{p_{2m}}{q_{2m}} = \frac{p_{2m-2}}{q_{2m-2}} + \frac{a_{2m}}{q_{2m-2}q_{2m}} > \frac{p_{2m-2}}{q_{2m-2}} \quad \text{und} \quad \frac{p_{2m+1}}{q_{2m+1}} = \frac{p_{2m-1}}{q_{2m-1}} - \frac{a_{2m+1}}{q_{2m-1}q_{2m+1}} < \frac{p_{2m-1}}{q_{2m-1}}.$$

Die Formel

$$\frac{p_n}{q_n} - \frac{p_{n-1}}{q_{n-1}} = \frac{(-1)^{n-1}}{q_nq_{n-1}}$$

liefert durch Einsetzen von $n = 2m$ bzw. $n = 2m+1$

$$\begin{aligned} \frac{p_{2m}}{q_{2m}} &= \frac{p_{2m-1}}{q_{2m-1}} - \frac{1}{q_{2m}q_{2m-1}} < \frac{p_{2m-1}}{q_{2m-1}} \\ \frac{p_{2m+1}}{q_{2m+1}} &= \frac{p_{2m}}{q_{2m}} + \frac{1}{q_{2m+1}q_{2m}} > \frac{p_{2m}}{q_{2m}} \end{aligned}$$

Aus den beiden letzten Überlegungen folgt sofort die Ungleichungskette

$$\frac{p_0}{q_0} < \frac{p_2}{q_2} < \frac{p_4}{q_4} < \cdots < \frac{p_5}{q_5} < \frac{p_3}{q_3} < \frac{p_1}{q_1}.$$

Ist für ein $n \in \mathbf{N}_0$ (das bei der Kettenbruchentwicklung auftretende) $\alpha_n \in \mathbf{Z}$, so ist $\alpha = \frac{p_n}{q_n}$ und die Behauptung folgt aus der aufgestellten Ungleichungskette. Andernfalls ergibt sich aus

$$\alpha - \frac{p_n}{q_n} = \frac{(-1)^n}{q_n(\alpha_{n+1}q_n + q_{n-1})}$$

durch Einsetzen von $n = 2m$ bzw. $n = 2m+1$

$$\begin{aligned} \alpha &= \frac{p_{2m}}{q_{2m}} + \frac{1}{q_{2m}(\alpha_{2m+1}q_{2m} + q_{2m-1})} > \frac{p_{2m}}{q_{2m}} \quad \text{und} \\ \alpha &= \frac{p_{2m+1}}{q_{2m+1}} - \frac{1}{q_{2m+1}(\alpha_{2m+2}q_{2m+1} + q_{2m})} < \frac{p_{2m+1}}{q_{2m+1}}, \end{aligned}$$

was die behauptete Ungleichungskette liefert.

(4) Ist $\alpha \neq \frac{p_n}{q_n}$, so ist $\alpha_n \notin \mathbf{Z}$, also existiert $\alpha_{n+1} = \frac{1}{\alpha_n - a_n}$ mit $a_n = \lfloor \alpha_n \rfloor$. Der letzte Satz liefert

$$|\alpha - \frac{p_n}{q_n}| = \frac{1}{q_n(\alpha_{n+1}q_n + q_{n-1})}.$$

Zu zeigen ist also

$$\frac{1}{2q_nq_{n+1}} \leq \frac{1}{q_n(q_{n+1} + q_n)} < \frac{1}{q_n(\alpha_{n+1} + q_{n-1})} \leq \frac{1}{q_nq_{n+1}},$$

was äquivalent zu

$$q_{n+1} \leq \alpha_{n+1}q_n + q_{n-1} < q_{n+1} + q_n \leq 2q_{n+1}$$

ist. Nun haben wir $a_{n+1} \leq \alpha_{n+1} < a_{n+1} + 1$ und somit können wir abschätzen:

$$\begin{aligned} &q_{n+1} = a_{n+1}q_n + q_{n-1} \leq \alpha_{n+1}q_n + q_{n-1} < \\ &< (a_{n+1}+1)q_n + q_{n-1} = (a_{n+1}q_n + q_{n-1}) + q_n = q_{n+1} + q_n \leq 2q_{n+1}, \end{aligned}$$

was die Behauptung beweist.

(5) Die erste Ungleichung ist trivial für $n = 0$ wegen $q_0 = 1$ und folgt für $n \geq 1$ aus 4. mit $q_n < q_{n+1}$. Nach 1. gilt außerdem $q_n < 1.618^{n-1}$, was zusammen mit

$$\frac{1}{q_n^2} < \frac{1}{1.618^{2(n-1)}} < 0.4^{n-1}$$

die Behauptung zeigt.

(6) Dies folgt sofort aus 4. ■

Mit dem Kettenbruchalgorithmus erhält man also gute Approximationen an Irrationalzahlen.

Beispiel: $\pi = [3, 7, 15, 1, 292, \ldots]$ hat die Näherungsbrüche

$$\begin{aligned}
\frac{p_0}{q_0} &= 3 && = 3.0000000000\ldots \\
\frac{p_1}{q_1} &= \frac{22}{7} && = 3.1428571428\ldots \\
\frac{p_2}{q_2} &= \frac{333}{106} && = 3.1415094339\ldots \\
\frac{p_3}{q_3} &= \frac{355}{113} && = 3.1415929203\ldots \\
\frac{p_4}{q_4} &= \frac{103993}{33102} && = 3.1415926530\ldots \\
\pi &= && = 3.1415926535\ldots
\end{aligned}$$

FOLGERUNG. *Ist α irrational reell, so gibt es unendlich viele Brüche $\frac{p}{q}$ mit*

$$|\alpha - \frac{p}{q}| < \frac{1}{q^2}.$$

Bemerkungen:

(1) Ist α reell und q eine natürliche Zahl, so gibt es eine ganze Zahl m mit

$$\frac{m}{q} \leq \alpha < \frac{m+1}{q}.$$

Setzt man dann $p = m$ bzw. $p = m + 1$, so erhält man

$$|\alpha - \frac{p}{q}| < \frac{1}{2q}.$$

Ist q eine 10-er Potenz, so hat man die Approximation durch Dezimalbrüche. Die Kettenbruchapproximation ist aber wesentlich besser.

(2) Ist $\alpha = \frac{a}{b}$ rational, so folgt aus

$$0 < |\alpha - \frac{p}{q}| < \frac{1}{q^2} \quad \text{wegen} \quad \alpha - \frac{p}{q} = \frac{aq - bp}{bq}$$

und $|aq - bp| \in \mathbf{N}$ sofort sofort $q < b$. Also können nur endlich viele $\frac{p}{q}$ die Ungleichung erfüllen.

SATZ. *Von zwei aufeinanderfolgenden Näherungsbrüchen $\frac{p}{q} = \frac{p_{n-1}}{q_{n-1}}$ oder $\frac{p}{q} = \frac{p_n}{q_n}$ an die reelle Zahl α genügt zumindest eine der Ungleichung*

$$|\alpha - \frac{p}{q}| < \frac{1}{2q^2}.$$

Beweis: Wir nehmen an,

$$|\alpha - \frac{p_{n-1}}{q_{n-1}}| \geq \frac{1}{2q_{n-1}^2} \quad \text{und} \quad |\alpha - \frac{p_n}{q_n}| \geq \frac{1}{2q_n^2}.$$

Da α zwischen $\frac{p_{n-1}}{q_{n-1}}$ und $\frac{p_n}{q_n}$ liegt, ist

$$|\frac{p_{n-1}}{q_{n-1}} - \frac{p_n}{q_n}| = |\alpha - \frac{p_{n-1}}{q_{n-1}}| + |\alpha - \frac{p_n}{q_n}|$$

und damit

$$\frac{1}{q_{n-1}q_n} = \frac{|p_{n-1}q_n - p_nq_{n-1}|}{q_{n-1}q_n} = |\frac{p_{n-1}}{q_{n-1}} - \frac{p_n}{q_n}| = |\alpha - \frac{p_{n-1}}{q_{n-1}}| + |\alpha - \frac{p_n}{q_n}| \geq \frac{1}{2q_{n-1}^2} + \frac{1}{2q_n^2} = \frac{q_{n-1}^2 + q_n^2}{2q_{n-1}^2q_n^2},$$

was sofort $2q_{n-1}q_n \geq q_{n-1}^2 + q_n^2$ und damit $0 \geq (q_{n-1} - q_n)^2$, also $q_n = q_{n-1}$ liefert. Dies kann nur im Fall $n = 1$ auftreten, außerdem werden dann aus den Ungleichungen Gleichungen, d.h.

$$\alpha - \frac{p_0}{q_0} = \alpha - a_0 = \frac{1}{2} \quad \text{und} \quad \frac{p_1}{q_1} - \alpha = a_0 + 1 - \alpha = \frac{1}{2}.$$

Nun hat aber die Zahl $\alpha = a_0 + \frac{1}{2}$ die Kettenbruchentwicklung $\alpha = [a_0, 2]$ mit den Näherungsbrüchen $\frac{p_0}{q_0} = a_0$ und $\frac{p_1}{q_1} = \frac{2a_0+1}{2}$, so daß auch dieser Fall nicht möglich ist. ■

Beispiel: Die Kettenbruchentwicklung von

$$\alpha = \frac{509473}{788342},$$

das zufällig gewählt wurde, illustriert den letzten Satz.

i	a_i	p_i	q_i	$\frac{p_i}{q_i}$	$q_i^2(\alpha - \frac{p_i}{q_i})$
0	0	0	1	0.0000000000	+0.646259
1	1	1	1	1.0000000000	-0.353741
2	1	1	2	0.5000000000	+0.585035
3	1	2	3	0.6666666667	-0.183670
4	4	9	14	0.6428571429	+0.666736
5	1	11	17	0.6470588235	-0.231190
6	3	42	65	0.6461538462	+0.443672
7	1	53	82	0.6463414634	-0.555444
8	1	95	147	0.6462585034	+0.007645
9	130	12403	19192	0.6462588579	-0.243448
10	4	49707	76915	0.6462588572	+0.097566
11	10	509473	788342	0.6462588572	+0.000000

Der folgende Satz gibt ein wichtiges (hinreichendes) Kriterium an, wann ein Bruch Näherungsbruch in der Kettenbruchentwicklung einer Zahl ist.

SATZ. *Ist* $\alpha \in \mathbf{R}$ *und* $\frac{p}{q} \in \mathbf{Q}$ *mit*

$$|\alpha - \frac{p}{q}| < \frac{1}{2q^2},$$

so ist $\frac{p}{q}$ *Näherungsbruch in der Kettenbruchentwicklung von* α.

Beweis:

- Wir können schreiben
$$\alpha - \frac{p}{q} = \frac{\varepsilon\theta}{2q^2} \text{ mit } \varepsilon = \pm 1 \text{ und } 0 < \theta < 1.$$
- Da wir zeigen wollen, daß $\frac{p}{q}$ Näherungsbruch für α ist, bilden wir die Kettenbruchentwicklung von $\frac{p}{q}$:
$$\frac{p}{q} = [a_0, a_1, \ldots, a_n].$$
Indem wir eventuell a_n durch $a_n - 1, 1$ ersetzen und dadurch die Länge der Kettenbruchdarstellung von $\frac{p}{q}$ um 1 erhöhen, können wir erreichen, daß gilt
$$\varepsilon = (-1)^n.$$
Seien $\frac{p_i}{q_i}$ die zugehörigen Näherungsbrüche.
- Da der Fall $\alpha = \frac{p}{q} = \frac{p_n}{q_n}$ trivial ist, können wir ihn ausschließen und somit definieren:
$$x = -\frac{\alpha q_{n-1} - p_{n-1}}{\alpha q_n - p_n}.$$
Dann gilt
$$\alpha = \frac{xp_n + p_{n-1}}{xq_n + q_{n-1}}$$
und mit den üblichen Formeln für Kettenbrüche
$$\alpha = [a_0, a_1, \ldots, a_n, x].$$

- Nun haben wir mit entsprechenden Kettenbruchformeln

$$\frac{\varepsilon\theta}{2q^2} = \frac{(-1)^n\theta}{2q_n^2} = \alpha - \frac{p}{q} = \alpha - \frac{p_n}{q_n} = \frac{(-1)^n}{q_n(xq_n + q_{n-1})}.$$

Es folgt

$$\frac{2\theta}{q_n^2} = \frac{1}{q_n(xq_n + q_{n-1})}$$

und damit $xq_n - q_{n-1} = \frac{2}{\theta}q_n$, also

$$x = \frac{2}{\theta} - \frac{q_{n-1}}{q_n}.$$

Mit der Voraussetzung folgt $x > 1$, so daß wegen der Eindeutigkeit der Kettenbruchentwicklung $\alpha = [a_0, a_1, \ldots, a_n, x]$ tatsächlich der Anfang der Kettenbruchentwicklung von α ist. Also ist auch $\frac{p}{q}$ Näherungsbruch für α. ■

Das folgende Beispiel zeigt, daß man das Kriterium $|\alpha - \frac{p}{q}| < \frac{1}{2q^2}$ nicht durch eine Bedingung $|\alpha - \frac{p}{q}| < \frac{1}{(2-\varepsilon)q^2}$ ersetzen kann mit einem festen $\varepsilon > 0$.

Beispiel: Wir betrachten für $m \geq 1$ die rationale Zahl

$$\alpha = \frac{2m^2 + m + 2}{2m^3}.$$

Man rechnet leicht nach, daß gilt

$$[0, m-1, 2] < \alpha < [0, m-1, 3].$$

Für $x > 0$ ist die Funktion $x \mapsto [0, m-1, x]$ streng monoton steigend, also hat die Kettenbruchentwicklung von α die Gestalt

$$\alpha = [0, m-1, 2, \ldots].$$

Nun gilt

$$|\alpha - \frac{1}{m}| = \alpha - \frac{1}{m} = \frac{1}{2m^2} + \frac{1}{m^3} = \frac{1}{(2 - \frac{4}{m+2})m^2} < \frac{1}{m^2} \quad \text{für} \quad m \geq 3,$$

aber $\frac{1}{m} = [0, m] = [0, m-1, 1]$ ist kein Näherungsbruch von α.

5. Die Kettenbruchentwicklung von $\sqrt{d}$

Eine irrationale reelle Zahl α heißt reellquadratisch, wenn sie einer Gleichung $\alpha^2 + q_1\alpha + q_2 = 0$ mit $q_1, q_2 \in \mathbf{Q}$ genügt. Dann ist also

$$\alpha = \frac{1}{2}(-q_1 \pm \sqrt{q_1^2 - 4q_2}).$$

Wir wollen reellquadratische Zahlen wie in folgendem Lemma darstellen:

LEMMA. *Zu jeder reellquadratischen Zahl α gibt es $b, c \in \mathbf{Z}$, $d \in \mathbf{N}$, so daß d keine Quadratzahl ist, mit*

$$\alpha = \frac{b + \sqrt{d}}{c} \quad \textit{und} \quad c | d - b^2.$$

Beweis: Ist $\alpha^2 + q_1\alpha + q_2 = 0$ mit $q_1, q_2 \in \mathbf{Q}$ und schreibt man $q_1 = \frac{B}{A}$, $q_2 = \frac{C}{A}$ mit $A, B, C \in \mathbf{Z}$, so ist

$$\alpha = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \quad \text{oder} \quad \alpha = \frac{B + \sqrt{B^2 - 4AC}}{-2A}.$$

Setzt man also $d = B^2 - 4AC$ und $(b, c) = (-B, 2A)$ bzw. $(b, c) = (B, -2A)$, so gilt $c | d - b^2$ wegen $(\pm 2A) | (B^2 - 4AC) - (\pm B)^2$ und α erhält die gewünschte Form. ■

Beispiel: Für $n \in \mathbf{N}$ ist $-\frac{1}{n}\sqrt{5} = \frac{0 + \sqrt{5n^2}}{-n^2}$, wobei die letzte Darstellung der des Lemmas entspricht.

LEMMA. *Sei $\alpha = \frac{b+\sqrt{d}}{c}$ eine reellquadratische Zahl mit $c | b^2 - d$.*

(1) *Die Kettenbruchentwicklung von α ergibt sich mit $\alpha_0 = \alpha$, $b_0 = b$, $c_0 = c$ rekursiv aus*

$$a_n = \lfloor \alpha_n \rfloor, \quad b_{n+1} = a_n c_n - b_n, \quad c_{n+1} = \frac{d - b_{n+1}^2}{c_n}, \quad \alpha_{n+1} = \frac{b_{n+1} + \sqrt{d}}{c_{n+1}},$$

wobei wieder $c_{n+1} | d - b_{n+1}^2$ gilt.

(2) *Dabei kann man benutzen:*

$$a_n = \lfloor \frac{b_n + \sqrt{d}}{c_n} \rfloor = \begin{cases} \lfloor \frac{b_n + \lfloor \sqrt{d} \rfloor}{c_n} \rfloor & \text{im Fall } c_n > 0 \\ \lfloor \frac{b_n + \lceil \sqrt{d} \rceil}{c_n} \rfloor & \text{im Fall } c_n < 0 \end{cases}$$

Beweis: Wir nehmen an, wir haben bereits

$$\alpha_n = \frac{b_n + \sqrt{d}}{c_n} \quad \text{mit} \quad c_n | d - b_n^2 \quad \text{und} \quad \alpha = [a_0, a_1, \ldots, a_{n-1}, \alpha_n].$$

Natürlich muß man setzen $a_n = \lfloor \alpha_n \rfloor$. Wir definieren $b_{n+1} = a_n c_n - b_n$ und erhalten

$$\frac{d - b_{n+1}^2}{c_n} = \frac{d - a_n^2 c_n^2 + 2 a_n b_n c_n - b_n^2}{c_n} = \frac{d - b_n^2}{c_n} - a_n^2 c_n + 2 a_n b_n.$$

Setzen wir $c_{n+1} = \frac{d - b_{n+1}^2}{c_n}$, so ist wegen $c_n | d - b_n^2$ auch c_{n+1} eine ganze Zahl und erfüllt $c_{n+1} | d - b_{n+1}^2$. Mit $\alpha_n = a_n + \frac{1}{\alpha_{n+1}}$ erhält man nun

$$\alpha_{n+1} = \frac{1}{\alpha_n - a_n} = \frac{1}{\frac{b_n + \sqrt{d}}{c_n} - a_n} = \frac{c_n}{(b_n - a_n c_n) + \sqrt{d}} = \frac{c_n}{\sqrt{d} - b_{n+1}} =$$

$$= \frac{c_n (b_{n+1} + \sqrt{d})}{d - b_{n+1}^2} = \frac{b_{n+1} + \sqrt{d}}{\frac{d - b_{n+1}^2}{c_n}} = \frac{b_{n+1} + \sqrt{d}}{c_{n+1}}.$$

Dies beweist den ersten Teil des Lemmas. Für den zweiten Teil sei $u \in \mathbf{Z}$. Im Fall $c_n > 0$ gilt

$$u \le \frac{b_n + \sqrt{d}}{c_n} \iff c_n u \le b_n + \sqrt{d} \iff c_n u \le b_n + \lfloor \sqrt{d} \rfloor \iff u \le \frac{b_n + \lfloor \sqrt{d} \rfloor}{c_n},$$

im Fall $c_n < 0$ gilt

$$u \le \frac{b_n + \sqrt{d}}{c_n} \iff c_n u \ge b_n + \sqrt{d} \iff c_n u \ge b_n + \lceil \sqrt{d} \rceil \iff u \le \frac{b_n + \lceil \sqrt{d} \rceil}{c_n},$$

was auch die letzte Behauptung zeigt. ■

Beispiel: Wir berechnen mit dem Verfahren die Kettenbruchentwicklung von $\sqrt{19}$ mit $d = 19$:

n	b_n	c_n	α_n	a_n
0	0	1	$\sqrt{19}$	4
1	4	3	$\frac{4+\sqrt{19}}{3}$	2
2	2	5	$\frac{2+\sqrt{19}}{5}$	1
3	3	2	$\frac{3+\sqrt{19}}{2}$	3
4	3	5	$\frac{3+\sqrt{19}}{5}$	1
5	2	3	$\frac{2+\sqrt{19}}{3}$	2
6	4	1	$4 + \sqrt{19}$	8
7	4	3	$\frac{4+\sqrt{19}}{3}$	2
8	2	5	$\frac{2+\sqrt{19}}{5}$	1
9	3	2	$\frac{3+\sqrt{19}}{2}$	3
10	3	5	$\frac{3+\sqrt{19}}{5}$	1

Daraus ersieht man sofort

$$\sqrt{19} = [4, 2, 1, 3, 1, 2, 8, 2, 1, 3, 1, 2, 8, \ldots] = [4, \overline{2, 1, 3, 1, 2, 8}],$$

die Kettenbruchentwicklung wird also schließlich periodisch.

Bevor wir die Kettenbruchentwicklung von $\sqrt{d}$ allgemein untersuchen, stellen wir noch ein Lemma bereit:

LEMMA. *Die Bezeichnungen seien wie im letzten Lemma.*

(1) *Gelten für* $\alpha_n = \frac{b_n+\sqrt{d}}{c_n}$ *und* $\alpha_n' = \frac{b_n-\sqrt{d}}{c_n}$ *die Ungleichungen*

$$\alpha_n > 1 \quad \text{und} \quad -1 < \alpha_n' < 0,$$

so auch

$$\alpha_{n+1} > 1 \quad \text{und} \quad -1 < \alpha_{n+1}' < 0.$$

(2) *Die Bedingungen* $\alpha_n > 1$ *und* $-1 < \alpha_n' < 0$ *sind äquivalent zu*

$$1 \le b_n < \sqrt{d} \quad \text{und} \quad \sqrt{d} - b_n < c_n < \sqrt{d} + b_n.$$

(Insbesondere ist $1 \le c_n < 2\sqrt{d}$*. Man nennt* α_n *dann reduziert.)*

Beweis:

(1) Es ist $\alpha_n = a_n + \frac{1}{\alpha_{n+1}}$ mit $\alpha_{n+1} > 1$, also

$$\alpha_{n+1} = \frac{1}{\alpha_n - a_n}.$$

Man rechnet nun leicht nach, daß auch

$$\alpha_{n+1}' = \frac{1}{\alpha_n' - a_n}$$

gilt, woraus mit $a_n \ge 1$ und $-1 < \alpha_n' < 0$ sofort $-1 < \alpha_{n+1}' < 0$ folgt.

(2) Wir haben die Äquivalenzen

$$\begin{aligned}
-1 < \alpha_n' < 0 < 1 < \alpha_n \quad &\iff \quad -1 < \frac{b_n - \sqrt{d}}{c_n} < 0 < 1 < \frac{b_n + \sqrt{d}}{c_n} \\
&\iff \quad -1 < \frac{b_n - \sqrt{d}}{c_n} < 0 < 1 < \frac{b_n + \sqrt{d}}{c_n} \text{ und } c_n > 0 \\
&\iff \quad -c_n < b_n - \sqrt{d} < 0 < c_n < b_n + \sqrt{d} \\
&\iff \quad c_n > -b_n + \sqrt{d} > 0 \text{ und } 0 < c_n < b_n + \sqrt{d} \\
&\iff \quad b_n < \sqrt{d} \text{ und } \sqrt{d} - b_n < c_n < \sqrt{d} + b_n \\
&\iff \quad 1 \le b_n < \sqrt{d} \text{ und } \sqrt{d} - b_n < c_n < \sqrt{d} + b_n,
\end{aligned}$$

aus denen die Behauptung folgt. ■

Bemerkung: Aus dem letzten Lemma sieht man, daß es zu festem $d \in \mathbf{N}$ nur endlich viele $\alpha = \frac{b+\sqrt{d}}{c}$ mit $c | d - b^2$ gibt, die den Ungleichungen $\alpha > 1$ und $-1 < \alpha' < 0$ genügen.

SATZ. *Sei* $d \in \mathbf{N}$ *keine Quadratzahl.*

(1) *Mit* $b_0 = 0$, $c_0 = 1$ *erhält man die Kettenbruchentwicklung* $[a_0, a_1, a_2, \dots]$ *rekursiv mit den Formeln*

$$a_n = \lfloor \frac{b_n + \lfloor \sqrt{d} \rfloor}{c_n} \rfloor, \quad b_{n+1} = a_n c_n - b_n, \quad c_{n+1} = \frac{d - b_{n+1}^2}{c_n},$$

wobei wieder $c_{n+1} | d - b_{n+1}^2$ *gilt.*

(2) *Für* $n \ge 1$ *gilt*

$$1 \le b_n < \sqrt{d} \quad \text{und} \quad \sqrt{d} - b_n < c_n < \sqrt{d} + b_n,$$

insbesondere ist $1 \le c_n < 2\sqrt{d}$.

(3) *Für die Näherungsbrüche* $\frac{p_n}{q_n}$ *von* $\sqrt{d}$*, die man rekursiv aus* $p_{-2} = 0$, $q_{-2} = 1$, $p_{-1} = 1$, $q_{-1} = 0$ *und*

$$p_{n+1} = a_{n+1} p_n + p_{n-1}, \quad q_{n+1} = a_{n+1} q_n + q_{n-1}$$

berechnen kann, gilt

$$p_n^2 - d q_n^2 = (-1)^{n+1} c_{n+1}.$$

(4) *Ist $k \geq 1$ minimal mit $c_k = 1$, so ist die Kettenbruchentwicklung periodisch mit Periodenlänge k:*

$$\sqrt{d} = [a_0, \overline{a_1, \ldots, a_k}] = [a_0, a_1, \ldots, a_k, a_1, \ldots, a_k, a_1, \ldots, a_k, \ldots].$$

(5) *Außerdem gilt*

$$a_k = 2a_0 \quad \textit{und} \quad a_i = a_{k-i} \textit{ für } i = 1, \ldots, k-1.$$

Beweis:

2. Es ist $a_0 = \lfloor \sqrt{d} \rfloor$,

$$\alpha_1 = \frac{1}{\sqrt{d} - \lfloor \sqrt{d} \rfloor} \quad \text{und} \quad \alpha_1' = \frac{1}{-\sqrt{d} - \lfloor \sqrt{d} \rfloor},$$

also $-1 < \alpha_1' < 0$ und $\alpha_1 > 1$, woraus man mit dem letzten Lemma zunächst die Ungleichungen für b_1, c_1 und dann induktiv für b_n, c_n erhält.

1. Aus 2. folgt $c_n \geq 1$, so daß die Behauptung sich direkt aus den allgemeinen Formeln ergibt.

3. Die allgemeinen Formeln liefern

$$\sqrt{d} = [a_0, a_1, \ldots, a_n, \alpha_{n+1}] = \frac{\alpha_{n+1} p_n + p_{n-1}}{\alpha_{n+1} q_n + q_{n-1}}.$$

Setzt man hier $\alpha_{n+1} = \frac{b_{n+1} + \sqrt{d}}{c_{n+1}}$ ein und macht dann Koeffizientenvergleich bzgl. $\sqrt{d}$, so erhält man:

$$p_n = b_{n+1} q_n + c_{n+1} q_{n-1} \quad \text{und} \quad d q_n = b_{n+1} p_n + c_{n+1} p_{n-1},$$

also

$$p_n^2 - d q_n^2 = c_{n+1}(p_n q_{n-1} - p_{n-1} q_n) = c_{n+1}(-1)^{n+1},$$

wie behauptet.

4. Aus $1 \leq b_k < \sqrt{d}$ und $\sqrt{d} - b_k < c_k < \sqrt{d} + b_k$ folgt mit $c_k = 1$ die Ungleichung $b_k < \sqrt{d} < b_k + 1$, also $b_k = \lfloor \sqrt{d} \rfloor$ und damit

$$\alpha_k = \frac{b_k + \sqrt{d}}{c_k} = \lfloor \sqrt{d} \rfloor + \sqrt{d} \quad \text{und} \quad a_k = 2\lfloor \sqrt{d} \rfloor = 2a_0.$$

Es folgt

$$\alpha_{k+1} = \frac{1}{\alpha_k - a_k} = \frac{1}{\sqrt{d} - \lfloor \sqrt{d} \rfloor} = \alpha_1,$$

woraus sofort $\alpha_i = \alpha_{k+i}$ für alle $i \geq 1$ und damit die behauptete Periodizität folgt.

5. Man zeigt zunächst, daß sich für reduzierte α_n die Kettenbruchentwicklung von $-\frac{1}{\alpha_{n+1}'}$ aus

$$-\frac{1}{\alpha_{n+1}'} = a_n + \frac{1}{-\frac{1}{\alpha_n'}}$$

ergibt. Da $\alpha_1, \alpha_2, \ldots$ reduziert sind, folgt rekursiv

$$-\frac{1}{\alpha_k'} = [a_{k-1}, -\frac{1}{\alpha_{k-1}'}] = [a_{k-1}, a_{k-2}, -\frac{1}{\alpha_{k-2}'}] = \cdots = [a_{k-1}, a_{k-2}, \ldots, a_1, -\frac{1}{\alpha_1'}].$$

Nun ist

$$-\frac{1}{\alpha_k'} = -\frac{1}{\lfloor \sqrt{d} \rfloor - \sqrt{d}} = \frac{1}{\sqrt{d} - \lfloor \sqrt{d} \rfloor} = \alpha_1.$$

Aus den Kettenbruchentwicklungen

$$\begin{aligned} -\frac{1}{\alpha_k'} &= [a_{k-1}, a_{k-2}, \ldots, a_1, -\frac{1}{\alpha_1'}] \\ \alpha_1 &= [a_1, a_2, \ldots, a_{k-1}, \alpha_k] \end{aligned}$$

folgt durch Vergleich wegen der Eindeutigkeit der Kettenbruchentwicklung

$$a_1 = a_{k-1}, \quad a_2 = a_{k-2}, \quad \ldots, a_1 = a_{k-1}.$$

Die Gleichung $a_k = 2a_0$ wurde bereits in 4. bewiesen. ■

Bemerkung: Man kann zeigen, daß genau die reellquadratischen Zahlen periodische Kettenbruchentwicklungen haben. Außerdem ist die Kettenbruchentwicklung reinperiodisch, wenn α reduziert ist.

Die folgende Maple-Funktion bestimmt die Kettenbruchentwicklung von $\sqrt{d}$:

```
# Kettenbruchentwicklung von sqrt(d)
kbe_wd:=proc()
  d:=args[1];
  wd:=isqrt(d); if wd^2>d then wd:=wd-1; fi;
  aa:=[wd]; b:=wd; c:=d-wd^2; # Beginn mit b_1 und c_1
  while c>1 do
    a:=iquo(b+wd,c); aa:=[op(aa),a];
    b:=a*c-b;
    c:=(d-b^2)/c;
  od;
  a:=iquo(b+wd,c); aa:=[op(aa),a];
  aa;
end;
```

Beispiele:

$\sqrt{2} = [1, \overline{2}]$ $\sqrt{3} = [1, \overline{1, 2}]$ $\sqrt{5} = [2, \overline{4}]$

$\sqrt{6} = [2, \overline{2, 4}]$ $\sqrt{7} = [2, \overline{1, 1, 1, 4}]$ $\sqrt{8} = [2, \overline{1, 4}]$

$\sqrt{10} = [3, \overline{6}]$ $\sqrt{11} = [3, \overline{3, 6}]$ $\sqrt{12} = [3, \overline{2, 6}]$

$\sqrt{13} = [3, \overline{1, 1, 1, 1, 6}]$ $\sqrt{14} = [3, \overline{1, 2, 1, 6}]$ $\sqrt{15} = [3, \overline{1, 6}]$

$\sqrt{17} = [4, \overline{8}]$ $\sqrt{18} = [4, \overline{4, 8}]$ $\sqrt{19} = [4, \overline{2, 1, 3, 1, 2, 8}]$

$\sqrt{20} = [4, \overline{2, 8}]$ $\sqrt{21} = [4, \overline{1, 1, 2, 1, 1, 8}]$ $\sqrt{22} = [4, \overline{1, 2, 4, 2, 1, 8}]$

$\sqrt{23} = [4, \overline{1, 3, 1, 8}]$ $\sqrt{24} = [4, \overline{1, 8}]$ $\sqrt{26} = [5, \overline{10}]$

$\sqrt{27} = [5, \overline{5, 10}]$ $\sqrt{28} = [5, \overline{3, 2, 3, 10}]$ $\sqrt{29} = [5, \overline{2, 1, 1, 2, 10}]$

$\sqrt{30} = [5, \overline{2, 10}]$ $\sqrt{31} = [5, \overline{1, 1, 3, 5, 3, 1, 1, 10}]$ $\sqrt{32} = [5, \overline{1, 1, 1, 10}]$

$\sqrt{33} = [5, \overline{1, 2, 1, 10}]$ $\sqrt{34} = [5, \overline{1, 4, 1, 10}]$ $\sqrt{35} = [5, \overline{1, 10}]$

$\sqrt{37} = [6, \overline{12}]$ $\sqrt{38} = [6, \overline{6, 12}]$ $\sqrt{39} = [6, \overline{4, 12}]$

$\sqrt{40} = [6, \overline{3, 12}]$ $\sqrt{41} = [6, \overline{2, 2, 12}]$ $\sqrt{42} = [6, \overline{2, 12}]$

$\sqrt{43} = [6, \overline{1, 1, 3, 1, 5, 1, 3, 1, 1, 12}]$ $\sqrt{44} = [6, \overline{1, 1, 1, 2, 1, 1, 1, 12}]$ $\sqrt{45} = [6, \overline{1, 2, 2, 2, 1, 12}]$

$\sqrt{46} = [6, \overline{1, 3, 1, 1, 2, 6, 2, 1, 1, 3, 1, 12}]$ $\sqrt{47} = [6, \overline{1, 5, 1, 12}]$ $\sqrt{48} = [6, \overline{1, 12}]$

Beispiel: Für $m \in \mathbf{N}$ ist die Kettenbruchentwicklung von $\sqrt{m^2+1}$

$$\sqrt{m^2+1} = [m, \overline{2m}].$$

Beispiele: Im folgenden geben wir die Zahlen $d \leq 500$ an, deren Kettenbruchperiode ≥ 30 ist:

$$\begin{aligned}
\sqrt{331} &= [18, \overline{5,5,1,6,2,3,1,1,2,1,2,1,11,2,1,1,17,1,1,2,11,1,2,1,2,1,1,3,2,6,1,5,5,36}] \\
\sqrt{379} &= [19, \overline{2,7,3,2,2,6,12,1,4,1,1,1,3,4,19,4,3,1,1,1,4,1,12,6,2,2,3,7,2,38}] \\
\sqrt{421} &= [20, \overline{1,1,13,5,1,3,1,2,1,1,1,2,9,1,7,3,3,2,2,3,3,7,1,9,2,1,1,1,2,1,3,1,5,13,1,1,40}] \\
\sqrt{436} &= [20, \overline{1,7,2,1,1,1,13,3,2,2,5,1,1,4,10,4,1,1,5,2,2,3,13,1,1,1,2,7,1,40}] \\
\sqrt{454} &= [21, \overline{3,3,1,13,2,3,2,1,1,4,6,1,7,1,1,1,20,1,1,1,7,1,6,4,1,1,2,3,2,13,1,3,3,42}] \\
\sqrt{463} &= [21, \overline{1,1,13,1,5,4,1,1,1,1,2,2,6,1,3,21,3,1,6,2,2,1,1,1,1,4,5,1,13,1,1,42}] \\
\sqrt{478} &= [21, \overline{1,6,3,4,1,1,5,1,2,3,1,1,1,1,1,13,1,20,1,13,1,1,1,1,1,3,2,1,5,1,1,4,3,6,1,42}]
\end{aligned}$$

Die Kettenbruchentwicklung von $\sqrt{d}$ hilft beim Lösen der sogenannten Pellschen Gleichung $x^2 - dy^2 = 1$ in ganzen Zahlen x und y, wie folgender Satz zeigt:

SATZ. *Ist $d \in \mathbf{N}$ kein Quadrat, hat die Kettenbruchentwicklung von $\sqrt{d}$ Periodenlänge k und die Näherungsbrüche $\frac{p_n}{q_n}$, so gilt:*

$$\{(x,y) \in \mathbf{Z} \times \mathbf{Z} : x^2 - dy^2 = 1\} = \{(\pm 1, 0)\} \cup \{(\pm p_{\widetilde{k}l-1}, \pm q_{\widetilde{k}l-1}) : l \geq 1\} \text{ mit } \begin{cases} \widetilde{k} = k & \text{für } k \equiv 0 \bmod 2 \\ \widetilde{k} = 2k & \text{für } k \equiv 1 \bmod 2 \end{cases}$$

Beweis: Wir verwenden die bei der Kettenbruchentwicklung von $\sqrt{d}$ aufgetretenen Bezeichnungen des letzten Satzes. Sind $x, y \in \mathbf{Z}$ mit $x^2 - dy^2 = 1$, so gibt es die triviale Möglichkeit $(x,y) = (\pm 1, 0)$. Da außerdem mit (x,y) auch $(\pm x, \pm y)$ eine Lösung der Gleichung ist, können o.E. $y > 0$ und $x > 0$ annehmen. Es gilt dann $(x - \sqrt{d}y)(x + \sqrt{d}y) = 1$, insbesondere $x > \sqrt{d}y$ und damit

$$|\sqrt{d} - \frac{x}{y}| = \frac{1}{y(x + \sqrt{d}y)} < \frac{1}{y \cdot 2\sqrt{d}y} < \frac{1}{2y^2},$$

also ist $\frac{x}{y}$ ein Näherungsbruch der Kettenbruchentwicklung von $\sqrt{d}$, d.h. es gibt ein n mit $\frac{x}{y} = \frac{p_n}{q_n}$ und damit $x = p_n$, $y = q_n$. Nun wissen wir aber, daß

$$p_n^2 - dq_n^2 = (-1)^{n+1} c_{n+1}$$

gilt. Wir erhalten die Bedingung $n + 1 \equiv 0 \bmod 2$ und $c_{n+1} = 1$. Genau dann ist $c_{n+1} = 1$, wenn $n+1$ ein Vielfaches der Periodenlänge k ist, d.h. $n + 1 = kl$ mit $l \in \mathbf{N}$. Nun ist

$$p_{kl-1}^2 - dq_{kl-1}^2 = (-1)^{kl},$$

woraus sich sofort die Behauptung ergibt. ■

Beispiel: Wir wollen eine nichttriviale Lösung der Gleichung $x^2 - 103y^2 = 1$ bestimmen. Dazu brauchen wir die Kettenbruchentwicklung von $\sqrt{103}$. Wir benutzen die früheren Bezeichnungen:

n	b_n	c_n	a_n	p_n	q_n	$p_n^2 - 103q_n^2$
0	0	1	10	10	1	-3
1	10	3	6	61	6	13
2	8	13	1	71	7	-6
3	5	6	2	203	20	9
4	7	9	1	274	27	-11
5	2	11	1	477	47	2
6	9	2	9	4567	450	-11
7	9	11	1	5044	497	9
8	2	9	1	9611	947	-6
9	7	6	2	24266	2391	13
10	5	13	1	33877	3338	-3
11	8	3	6	227528	22419	1
12	10	1	20	4584437	451718	-3
13	10	3	6	27734150	2732727	13
14	8	13	1	32318587	3184445	-6
15	5	6	2	92371324	9101617	9

Also ist $(x, y) = (227528, 22419)$ eine Lösung. Die anderen Lösungen sind $(x, y) = (p_{12l-1}, q_{12l-1})$, $l = 2, 3, 4, \ldots$.

Bemerkung: Man kann zeigen

$$p_{kl-1} + q_{kl-1}\sqrt{d} = (p_{k-1} + q_{k-1}\sqrt{d})^l,$$

woraus man durch Koeffizientenvergleich (p_{kl-1}, q_{kl-1}) aus (p_{k-1}, q_{k-1}) gewinnen kann.

6. Eine Idee zur Primfaktorzerlegung

Erinnerung: Will man eine natürliche Zahl N faktorisieren, geht man meist nach folgendem Schema vor:

(1) Teile aus N alle kleinen Primteiler heraus, z.B. alle Primteiler $\leq 10^6$.
(2) Teste mit einem Primzahltest, ob N zusammengesetzt oder wahrscheinlich prim ist, z.B. mit einem Fermat-Test: Ist $2^{N-1} \not\equiv 1 \bmod N$, so ist N zusammengesetzt, andernfalls wahrscheinlich prim.
(3) Ist N zusammengesetzt, bestimme einen nichttrivialen Teiler von N, d.h. eine nichttriviale Faktorisierung $N = N_1 N_2$ und beginne mit N_1 und N_2 von vorne mit 2.
(4) Ist N wahrscheinlich prim, beweise, daß N prim ist, oder gib dich zufrieden mit der Aussage, daß N 'wahrscheinlich prim' ist.

Bemerkungen:

(1) Wir interessieren uns hier für den 3. Punkt, der Bestimmung eines nichttrivialen Teilers einer (zusammengesetzten) natürlichen Zahl (ohne kleine Teiler). Dies ist ein schwieriges Problem.
(2) Ist x irgendeine ganze Zahl, so gilt (bei entsprechender Normierung des ggT)

$$1 \leq \mathrm{ggT}(x, N) \leq N \quad \text{und} \quad \mathrm{ggT}(x, N) | N.$$

Da man den ggT mit dem euklidischen Algorithmus sehr schnell berechnen kann, kann man in den folgenden Anwendungen oft o.E. $\mathrm{ggT}(x, N) = 1$ annehmen, da man ansonsten sofort einen nichttrivialen Teiler von N hätte.

LEMMA. *Sei N eine ungerade natürliche Zahl.*

(1) *Sind x und y ganze Zahlen mit*

$$\mathrm{ggT}(x, N) = \mathrm{ggT}(y, N) = 1 \quad \textit{und} \quad x^2 \equiv y^2 \bmod N,$$

so gilt

$$N = \mathrm{ggT}(x+y, N) \cdot \mathrm{ggT}(x-y, N).$$

Außerdem ist

$$\{\mathrm{ggT}(x+y, N), \mathrm{ggT}(x-y, N)\} = \{1, N\} \quad \Longleftrightarrow \quad x \equiv \pm y \bmod N.$$

(2) *Hat man eine Faktorisierung $N = N_1 N_2$ mit natürlichen Zahlen $1 \leq N_1, N_2 \leq N$ und $\mathrm{ggT}(N_1, N_2) = 1$, so gilt*

$$(N_1 + N_2)^2 \equiv (N_1 - N_2)^2 \bmod N.$$

Beweis: Wir können ansetzen Primfaktorzerlegungen wie folgt ansetzen:

$$N = p_1^{a_1} \dots p_n^{a_n}, \quad \mathrm{ggT}(x+y, N) = p_1^{b_1} \dots p_n^{b_n}, \quad \mathrm{ggT}(x-y, N) = p_1^{c_1} \dots p_n^{c_n}$$

mit ungeraden Primzahlen p_i und $b_i, c_i \leq a_i$, da $ggT(x \pm y, N)$ Teiler von N sind. Wäre $b_i > 1$ und $c_i > 1$ für ein i, so würde $p_i | x+y$, $p_i | x-y$, also $p_i | 2x = (x+y) + (x-y)$ und damit $p_i | x$ folgen, ein Widerspruch zu $\mathrm{ggT}(x, N) = 1$. Also hat man

$$b_i > 0 \Longrightarrow c_i = 0 \quad \text{und} \quad c_i > 0 \Longrightarrow b_i = 0.$$

Aus $x^2 \equiv y^2 \bmod N$ folgt $N | (x+y)(x-y)$ und damit $p_i^{a_i} | p_i^{b_i + c_i}$, also $a_i \leq b_i + c_i$. Mit dem Vorangegangenen ergeben sich die zwei Möglichkeiten

$$b_i = a_i, c_i = 0 \quad \text{oder} \quad b_i = 0, c_i = a_i,$$

was die behauptete Faktorisierung beweist. Der Rest ist offensichtlich. ■

Lösungen der Kongruenz $x^2 \equiv y^2 \bmod N$ entsprechen also in gewissen Weise Faktorisierungen von N. Das macht folgenden Ansatz plausibel:

- Suche nach 'zufälligen' Lösungen der Kongruenz
$$x^2 \equiv y^2 \bmod N.$$
- Teste, ob
$$\mathrm{ggT}(x+y, N) \quad \text{oder} \quad \mathrm{ggT}(x-y, N)$$
nichttriviale Teiler von N sind.
- Findet man 'zufällig' Lösungen der Gleichung $x^2 \equiv y^2 \bmod N$, so sollte in mindestens 50% der Fälle $\mathrm{ggT}(x+y, N)$ ein nichttrivialer Teiler sein.

Die Frage ist nun: Wie findet man x und y mit $x^2 \equiv y^2 \bmod N$?

7. Faktorisierung mit Kettenbrüchen I — SQUFOF

Wir wollen die (zusammengesetzte) natürliche Zahl N (ohne kleine Teiler) faktorisieren und zu diesem Zweck x und y suchen mit $x^2 \equiv y^2 \bmod N$.
Wir bilden die Kettenbruchentwicklung von $\sqrt{N}$ mit den Rekursionsformeln $b_0 = 0$, $c_0 = 1$ und

$$a_n = \lfloor \frac{b_n + \lfloor \sqrt{d} \rfloor}{c_n} \rfloor, \quad b_{n+1} = a_n c_n - b_n, \quad c_{n+1} = \frac{d - b_{n+1}^2}{c_n},$$

wobei wieder $c_{n+1} | d - b_{n+1}^2$ gilt. Für die Näherungsbrüche $\frac{p_n}{q_n}$ gilt

$$p_n^2 - N q_n^2 = (-1)^{n+1} c_{n+1}.$$

Damit ist (für $m \geq 1$)

$$p_{2m-1}^2 \equiv c_{2m} \bmod N.$$

Ist jetzt c_{2m} ein Quadrat (und $\mathrm{ggT}(p_{2m-1}, N) = \mathrm{ggT}(c_{2m}, N) = 1$), so hat nach unseren Vorüberlegungen

$$N = \mathrm{ggT}(p_{2m-1} + \sqrt{c_{2m}}, N) \cdot \mathrm{ggT}(p_{2m-1} - \sqrt{c_{2m}}, N)$$

eine (mindestens 50%-ige) Chance, eine nichttriviale Faktorisierung von N zu sein.

Beispiel: $N = 1863623$. Die Kettenbruchentwicklung von $\sqrt{N}$ beginnt folgendermaßen:

$a_0 = 1365$	$p_0 = 1365$	$q_0 = 1$	$b_1 = 1365$	$c_1 = 398$
$a_1 = 6$	$p_1 = 8191$	$q_1 = 6$	$b_2 = 1023$	$c_2 = 2053$
$a_2 = 1$	$p_2 = 9556$	$q_2 = 7$	$b_3 = 1030$	$c_3 = 391$
$a_3 = 6$	$p_3 = 65527$	$q_3 = 48$	$b_4 = 1316$	$c_4 = 337$
$a_4 = 7$	$p_4 = 468245$	$q_4 = 343$	$b_5 = 1043$	$c_5 = 2302$
$a_5 = 1$	$p_5 = 533772$	$q_5 = 391$	$b_6 = 1259$	$c_6 = 121$

Nun ist $c_6 = 11^2$ ein Quadrat, und

$$\mathrm{ggT}(p_5 + 11, N) = 1847 \quad \text{und} \quad \mathrm{ggT}(p_5 - 11, N) = 1009,$$

was tatsächlich die Primfaktorzerlegung $N = 1009 \cdot 1847$ ergibt.

Wegen

$$\mathrm{ggT}(p_{2m-1} - \sqrt{c_{2m}}, N) = \mathrm{ggT}((p_{2m-1} \bmod N) - \sqrt{c_{2m}}, N)$$

müssen wir die Nenner p_n der Näherungsbrüche nur modulo N berechnen. Dies ergibt folgendes Verfahren:

Faktorisierungsversuch mit Kettenbrüchen: Sei N eine (zusammengesetzte) Zahl (ohne kleine Teiler).

(1) Setze

$$p_{-2} = 0, \quad p_{-1} = 1, \quad b_0 = 0, \quad c_0 = 1, \quad n = 0.$$

(2) Berechne

$$a_n = \lfloor \frac{b_n + \lfloor\sqrt{N}\rfloor}{c_n} \rfloor, \quad p_n = (a_n p_{n-1} + p_{n-2}) \bmod N, \quad b_{n+1} = a_n c_n - b_n, \quad c_{n+1} = \frac{N - b_{n+1}^2}{c_n}$$

(3) Ist $n \equiv 1 \bmod 2$, teste, ob c_{n+1} ein Quadrat ist, und überprüfe gegebenenfalls, ob

$$\mathrm{ggT}(p_n + \sqrt{c_{n+1}}, N) \quad \text{oder} \quad \mathrm{ggT}(p_n - \sqrt{c_{n+1}}, N)$$

ein nichttrivialer Teiler von N ist. Wenn ja, beende das Verfahren.

(4) Setze $n := n + 1$ und gehe zurück zu 2.

Wir haben dazu ein Programm kbfac_gmp.c geschrieben und damit die folgenden Beispiele gerechnet.

Beispiel: $N = 8014003$. Wir schreiben nur die Fälle an, bei denen c_{2m} ein Quadrat ist:

$p_{83} = 8013952 \bmod N \quad \sqrt{c_{84}} = 51 \quad \mathrm{ggT}(p_{83} + \sqrt{c_{84}}, N) = 8014003 \quad \mathrm{ggT}(p_{83} - \sqrt{c_{84}}, N) = 1$
$p_{107} = 4797192 \bmod N \quad \sqrt{c_{108}} = 7 \quad \mathrm{ggT}(p_{107} + \sqrt{c_{108}}, N) = 4001 \quad \mathrm{ggT}(p_{107} - \sqrt{c_{108}}, N) = 2003$

Dies ergibt die Primfaktorzerlegung $N = 2003 \cdot 4001$.

Beispiel: $N = 708990851153346323$

$$p_{11483} = 7693 \bmod N \qquad \sqrt{c_{11484}} = 7693 = p_{11483} \bmod N$$
$$p_{48341} = 708990851153343912 \bmod N \qquad \sqrt{c_{48342}} = 2411 = -p_{48341} \bmod N$$
$$p_{57899} = 161248758104205473 \bmod N \qquad \sqrt{c_{57900}} = 23203$$

$$\mathrm{ggT}(p_{57899} + \sqrt{c_{57900}}, N) = 1046353201 \quad \text{und} \quad \mathrm{ggT}(p_{57899} - \sqrt{c_{57900}}, N) = 677582723$$

Dies ergibt die Primfaktorzerlegung

$$N = 1046353201 \cdot 677582723$$

(Rechenzeit: 5 sec)

Beispiel: $N = 3721714774471783927219$

$$p_{40619} = 372171477447178389153 8 \bmod N \qquad \sqrt{c_{40620}} = 35681 = -p_{40619} \bmod N$$
$$p_{436961} = 2897495365218465298447 \bmod N \qquad \sqrt{c_{436962}} = 20671$$

$$\mathrm{ggT}(p_{436961} + \sqrt{c_{436962}}, N) = 56001153847 \quad \text{und} \quad \mathrm{ggT}(p_{436961} - \sqrt{c_{436962}}, N) = 66457823077$$

(Rechenzeit: 379 sec) Primfaktorzerlegung:

$$N = 56001153847 \cdot 66457823077$$

Beispiel: $N = 75807583696444466476144 3$.

$$p_{1458311} = 747339920083912957304697 \bmod N \quad \sqrt{c_{1458312}} = 297189$$

$$\mathrm{ggT}(p_{1458311} + \sqrt{c_{1458312}}, N) = 784123641211 \quad \text{und} \quad \mathrm{ggT}(p_{1458311} - \sqrt{c_{1458312}}, N) = 966780998713$$

(Rechenzeit: 6182 sec) Primfaktorzerlegung:

$$N = 784123641211 \cdot 966780998713$$

Bemerkung: Allerdings gibt es auch Fälle, bei denen dieses Verfahren nicht funktioniert, bei den zusammengesetzten ungeraden Zahlen < 100 sind dies 27, 51 und 65.

Shanks hat ein Faktorisierungsverfahren mit quadratischen Formen entwickelt, das sich auch gut mit Kettenbrüchen darstellen läßt. Wir geben das Verfahren an, beweisen aber nicht die Gültigkeit.

SQUFOF (square form factorization): Sei N eine ungerade zusammengesetzte natürliche Zahl.

(1) Setze
$$b_0 = 0, \quad c_0 = 1, \quad n = 0.$$

(2) Berechne
$$a_n = \lfloor \frac{b_n + \lfloor \sqrt{N} \rfloor}{c_n} \rfloor, \quad b_{n+1} = a_n c_n - b_n, \quad c_{n+1} = \frac{N - b_{n+1}^2}{c_n}$$

(3) Ist $n \equiv 1 \bmod 2$, teste, ob c_{n+1} ein Quadrat ist.
   (a) Ist c_{n+1} kein Quadrat, gehe zu 4.
   (b) Ist c_{n+1} ein Quadrat, überprüfe, ob es ein m gibt mit $0 \le m \le n$ und
$$\sqrt{c_{n+1}} = \begin{cases} c_m & \text{für } c_m \equiv 1 \bmod 2 \\ \frac{1}{2} c_m & \text{für } c_m \equiv 0 \bmod 2 \end{cases}.$$
   Wenn ja, gehe zu 4. Andernfalls berechne die Kettenbruchentwicklung von
$$\frac{-b_{n+1} + \sqrt{N}}{\sqrt{c_{n+1}}},$$
   d.h. setze
$$b'_0 = -b_{n+1}, \quad c'_0 = \sqrt{c_{n+1}}, \quad m = 0$$
   und gehe zu 5.

(4) Setze $n := n + 1$ und gehe zurück zu 2.

(5) Berechne
$$a'_m = \lfloor \frac{b'_m + \lfloor \sqrt{N} \rfloor}{c'_m} \rfloor, \quad b'_{m+1} = a'_m c'_m - b'_m, \quad c'_{m+1} = \frac{N - {b'}_{m+1}^2}{c'_m}$$

(6) Gilt $b'_m = b'_{m+1}$, so ist c'_m oder $\frac{1}{2} c'_m$ ein nichttrivialer Teiler von N, und das Verfahren ist beendet.

(7) Setze $m := m + 1$ und gehe zu 5.

Bemerkung: Wegen $c_{n+1} < 2\sqrt{N}$ kann man sich im Punkt 3.b auf den Vergleich mit solchen m's beschränken, für die $c_m < \sqrt{2\sqrt{N}}$ bzw. $\frac{1}{2} c_m < \sqrt{2\sqrt{N}}$ gilt. Es ist sinnvoll, sich diese Elemente in einer Liste zu merken.

Wir haben zu dem Verfahren ein Programm squfof_gmp.c geschrieben und damit folgende Beispiele gerechnet.

Beispiel: $N = 32033$ ist zusammengesetzt.

```
b_1=178   c_1=349
b_2=171   c_2=8    c_2/2=4
b_3=173   c_3=263
b_4=90    c_4=91
b_5=92    c_5=259
b_6=167   c_6=16   c_6/2=8
c_6 ist Quadrat, aber sqrt(c_6)=4=c_2/2
b_7=169   c_7=217
b_8=48    c_8=137
b_9=89    c_9=176
b_10=87   c_10=139
b_11=52   c_11=211
b_12=159  c_12=32  c_12/2=16
b_13=161  c_13=191
b_14=30   c_14=163
b_15=133  c_15=88
b_16=131  c_16=169
c_16 ist Quadrat, sqrt(c_16)=13 kommt nicht als c_m bzw. c_m/2 vor.
Kettenbruchentwicklung von (-131+sqrt(N))/13
b_1=170   c_1=241
b_2=71    c_2=112
b_3=153   c_3=77
b_4=155   c_4=104
b_5=157   c_5=71
b_6=127   c_6=224
b_7=97    c_7=101
b_8=105   c_8=208
b_9=103   c_9=103
b_10=b_9=103
ggT(N,c_9)=103
```

Dies ergibt die Primfaktorzerlegung $N = 103 \cdot 311$.

Beispiel:

```
N=49592509498708571407 ist zusammengesetzt
SQUFOF-Verfahren mit Kettenbruchentwicklung von sqrt(49592509498708571407)
Aufgenommen in Liste: c_5283=12527
sqrt(c_10584)=12527
Liste: 12527
Aufgenommen in Liste: c_30646=97001
sqrt(c_106756)=46833
Liste: 12527 97001
Da sqrt(c_106756) nicht in der Liste enthalten ist, wird die Kettenbruchentwicklung von
(-b_106756+sqrt(49592509498708571407)/sqrt(c_106756)
=(-6751084331+sqrt(49592509498708571407))/46833
betrachtet.
b'_53635=b'_53636=4943906659
ggT(N,c'_53635)=4943906659
Die restlichen Faktoren von N:
4943906659 (prim)
10031036773 (prim)
Zeit: 0 sec
```

Beispiel:

```
N=868719445232927625514336417981 ist zusammengesetzt
```

```
SQUFOF-Verfahren mit Kettenbruchentwicklung von sqrt(868719445232927625514336417981)
sqrt(c_4296562)=28192653
Liste:
Da sqrt(c_4296562) nicht in der Liste enthalten ist, wird die Kettenbruchentwicklung von
(-b_4296562+sqrt(868719445232927625514336417981)/sqrt(c_4296562)
=(-822063903952342+sqrt(868719445232927625514336417981))/28192653
betrachtet.
b'_2149558=b'_2149559=661177905109907
ggT(N,c'_2149558)=661177905109907
Die restlichen Faktoren von N:
661177905109907 (prim)
1313896666115183 (prim)
Zeit: 30 sec
```

Bemerkung: Beim Aufruf von 'ifactor(N,squfof)' versucht Maple, die Zahl N mit der SQUFOF-Methode zu faktorisieren.

8. Faktorisierung mit Kettenbrüchen II — CFRAC

Sei N eine zusammengesetzte natürliche Zahl ohne kleine Teiler. Wir wollen wieder Kongruenzen $x^2 \equiv y^2 \bmod N$ erhalten und dann testen, ob $\mathrm{ggT}(x+y, N)$ oder $\mathrm{ggT}(x-y, N)$ ein nichttrivialer Teiler von N ist.

Eine allgemeine Idee: Wir wählen endlich viele (kleine) Primzahlen $p_1, p_2, \ldots, p_{n-1}$ (und $p_0 = -1$), die sogenannte Faktorbasis. Wir suchen ganze Zahlen P_i und C_i mit

$$P_i^2 \equiv C_i \bmod N,$$

sodaß die Primfaktorzerlegung von C_i nur mit $p_0, \ldots, p_{n-1}$ auskommt, d.h.

$$C_i = (-1)^{a_0} \cdot p_1^{a_1} \cdot \cdots \cdot p_{n-1}^{a_{n-1}}$$

mit Zahlen $a_0, a_1, \ldots, a_{n-1} \in \mathbf{N}_0$. Neben P_i und C_i merkt man sich nur

$$v_i = (a_0 \bmod 2, a_1 \bmod 2, \ldots, a_{n-1} \bmod 2) \in \mathbf{F}_2^n.$$

Dann ist also (mit $v_i = (v_{i,0}, v_{i,1}, \ldots, v_{i,n-1})$)

$$C_i = (-1)^{v_{i,0}} p_1^{v_{i,1}} \cdot \cdots \cdot p_{n-1}^{v_{i,n-1}} \cdot \text{Quadrat}.$$

Findet man jetzt Indizes $i_1, \ldots, i_l$, so daß (in $\mathbf{F}_2^n$)

$$v_{i_1} + v_{i_2} + \cdots + v_{i_l} = 0$$

gilt, so ist

$$C_{i_1} C_{i_2} \ldots C_{i_l}$$

ein Quadrat in $\mathbf{Z}$, da der Exponent bei allen p_i's gerade ist. Also erhält man durch Aufmultiplizieren

$$(P_{i_1} \ldots P_{i_l})^2 \equiv (\sqrt{C_{i_1} \ldots C_{i_l}})^2 \bmod N,$$

und wir probieren, ob

$$\mathrm{ggT}(P_{i_1} P_{i_2} \ldots P_{i_l} + \sqrt{C_{i_1} \ldots C_{i_l}}, N) \quad \text{oder} \quad \mathrm{ggT}(P_{i_1} P_{i_2} \ldots P_{i_l} - \sqrt{C_{i_1} \ldots C_{i_l}}, N)$$

ein nichttrivialer Teiler von N ist.

Es stellen sich jetzt zwei Fragen:

(1) Wie findet man gesuchte Relationen $P_i^2 \equiv C_i \bmod N$?
(2) Hat man eine Anzahl von Relationen $P_i^2 \equiv C_i \bmod N$, $i = 0, \ldots, m$ gefunden, wie findet man Indizes $i_1, \ldots, i_l$, so daß $C_{i_1} \ldots C_{i_l}$ ein Quadrat ist bzw. $v_{i_1} + \cdots + v_{i_l} = 0$ gilt?

Bemerkung: Die beschriebene allgemeine Idee wird von verschiedenen Faktorisierungsmethoden benutzt, die sich im wesentlichen dadurch unterscheiden, wie man an geeignete Relationen $P_i^2 \equiv C_i \bmod N$ kommt, so daß sich C_i mit den Primzahlen der gewählten Faktorbasis faktorisieren läßt:

- CFRAC (nach Morrison-Brillhart), das Kettenbrüche benutzt und hier beschrieben werden soll,
- QS (quadratic sieve) - quadratisches Sieb,
- MPQS (multiple polynomial quadratic sieve),
- NFS (number field sieve) - Zahlkörpersieb.

Wir skizzieren jetzt das von Morrison und Brillhart entwickelte CFRAC-Verfahren: Sind $\frac{\widetilde{p}_i}{\widetilde{q}_i}$ die Näherungsbrüche in der Kettenbruchentwicklung von $\sqrt{N}$, so gilt

$$\widetilde{p}_i^2 - N\widetilde{q}_i^2 = (-1)^{i+1}c_{i+1},$$

also

$$\widetilde{p}_i^2 \equiv (-1)^{i+1}c_{i+1} \bmod N.$$

Kommen in der Primfaktorzerlegung von c_{i+1} nur die Primzahlen $p_1, \ldots, p_{n-1}$ vor, so hat man eine gewünschte Relation gefunden.

CFRAC - Morrison-Brillhart-Faktorisierungsverfahren: Sei N eine ungerade zusammengesetzte natürliche Zahl.

(1) Sei
$$\widetilde{p}_{-2} = 0, \quad \widetilde{p}_{-1} = 1, \quad b_0 = 0, \quad c_0 = 1, \quad k = 0, \quad m = -1, \quad l = -1.$$

(2) Berechne
$$a_k = \lfloor \frac{b_k + \lfloor\sqrt{N}\rfloor}{c_k} \rfloor, \quad \widetilde{p}_k = a_k\widetilde{p}_{k-1} + \widetilde{p}_{k-2} \bmod N, \quad b_{k+1} = a_kc_k - b_k, \quad c_{k+1} = \frac{N - b_{k+1}^2}{c_k}.$$

(3) Teste, ob sich c_{k+1} mit $p_1, p_2, \ldots, p_{n-1}$ vollständig faktorisieren läßt. Wenn nein, gehe zu 4. Andernfalls führt man folgende Schritte aus:

(a) Setze $l := l + 1$. Dann hat man mit
$$P_l = \widetilde{p}_k \quad \text{und} \quad C_l = (-1)^{k+1}c_{k+1}$$
eine Relation
$$P_l^2 \equiv C_l \bmod N.$$
Gilt
$$C_l = (-1)^{a_0}p_1^{a_1} \ldots p_{n-1}^{a_{n-1}},$$
so setzt man
$$v_l = (a_0 \bmod 2, a_1 \bmod 2, \ldots, a_{n-1} \bmod 2) \in \mathbf{F}_2^n.$$

(b) Die (Zeilen-)Vektoren $v_0, v_1, \ldots, v_l$ liefern eine Matrix
$$A = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_l \end{pmatrix} = \begin{pmatrix} v_{00} & v_{01} & \ldots & v_{0,n-1} \\ v_{10} & v_{11} & \ldots & v_{1,n-1} \\ \vdots & \vdots & & \vdots \\ v_{l0} & v_{l1} & \ldots & v_{l,n-1} \end{pmatrix}.$$
Nun testet man (mittels linearer Algebra über $\mathbf{F}_2$), ob es einen Vektor
$$u = (u_0, u_1, \ldots, u_l) \in \mathbf{F}_2^{l+1} \setminus \{0\}$$
gibt mit
$$uA = 0.$$
(Dies wird später noch genauer ausgeführt.) Wenn nein, geht man zu 4. Andernfalls: Die Gleichung $uA = 0$ ist äquivalent mit
$$\sum_{\substack{0 \le i \le l \\ u_i = 1}} v_i = 0,$$
so daß also
$$\prod_{\substack{0 \le i \le l \\ u_i = 1}} C_i$$

ein Quadrat ist, und damit

$$(\prod_{\substack{0\le i\le l\\ u_i=1}} P_i)^2 \equiv \prod_{\substack{0\le i\le l\\ u_i=1}} C_i \equiv \sqrt{\prod_{\substack{0\le i\le l\\ u_i=1}} C_i}^2 \bmod N$$

gilt. Also testet man, ob

$$\mathrm{ggT}(\prod_{\substack{0\le i\le l\\ u_i=1}} P_i + \sqrt{\prod_{\substack{0\le i\le l\\ u_i=1}} C_i}, N) \quad \text{oder} \quad \mathrm{ggT}(\prod_{\substack{0\le i\le l\\ u_i=1}} P_i - \sqrt{\prod_{\substack{0\le i\le l\\ u_i=1}} C_i}, N)$$

ein nichttrivialer Teiler von N ist. Wenn ja, ist man fertig, andernfalls geht man zu 4.

(4) Setze $k := k + 1$ und gehe zu 2.

Bemerkung: Wir haben zu dem Verfahren ein Programm cfrac_gmp.c geschrieben und damit die nachfolgenden Beispiele gerechnet.

Beispiel:

```
N=3025991 ist zusammengesetzt
Faktorbasis: -1 2 5 11 17 mit 5 Elementen

a_0=1739  p_0=1739  b_1=1739  c_1=1870=2*5*11*17*1
p_0^2-d*q_0^2=(-1)^1*c_1 liefert 1. Relation und P_0=p_0, C_0=c_1
Matrix A:
1 1 1 1 1

a_1=1  p_1=1740  b_2=131  c_2=1609=1609
a_2=1  p_2=3479  b_3=1478  c_3=523=523
a_3=6  p_3=22614  b_4=1660  c_4=517=11*47
a_4=6  p_4=139163  b_5=1442  c_5=1831=1831
a_5=1  p_5=161777  b_6=389  c_6=1570=2*5*157
a_6=1  p_6=300940  b_7=1181  c_7=1039=1039
a_7=2  p_7=763657  b_8=897  c_8=2138=2*1069
a_8=1  p_8=1064597  b_9=1241  c_9=695=5*139
a_9=4  p_9=1996054  b_10=1539  c_10=946=2*11*43
a_10=3  p_10=1000777  b_11=1299  c_11=1415=5*283
a_11=2  p_11=971617  b_12=1531  c_12=482=2*241
a_12=6  p_12=778497  b_13=1361  c_13=2435=5*487
a_13=1  p_13=1750114  b_14=1074  c_14=769=769
a_14=3  p_14=3002848  b_15=1233  c_15=1958=2*11*89
a_15=1  p_15=1726971  b_16=725  c_16=1277=1277
a_16=1  p_16=1703828  b_17=552  c_17=2131=2131
a_17=1  p_17=404808  b_18=1579  c_18=250=2*5^3*1
p_17^2-d*q_17^2=(-1)^18*c_18 liefert 2. Relation und P_1=p_17, C_1=c_18
Matrix A:
1 1 1 1 1
0 1 1 0 0

a_18=13  p_18=914350  b_19=1671  c_19=935=5*11*17*1
p_18^2-d*q_18^2=(-1)^19*c_19 liefert 3. Relation und P_2=p_18, C_2=c_19
Matrix A:
1 1 1 1 1
0 1 1 0 0
1 0 1 1 1

a_19=3  p_19=121867  b_20=1134  c_20=1861=1861
```

```
a_20=1  p_20=1036217  b_21=727  c_21=1342=2*11*61
a_21=1  p_21=1158084  b_22=615  c_22=1973=1973
a_22=1  p_22=2194301  b_23=1358  c_23=599=599
a_23=5  p_23=25625  b_24=1637  c_24=578=2*17^2*1
p_23^2-d*q_23^2=(-1)^24*c_24 liefert 4. Relation und P_3=p_23, C_3=c_24
Matrix A:
1 1 1 1 1
0 1 1 0 0
1 0 1 1 1
0 1 0 0 0
Mit v=(1,0,1,1) gilt vA=0 mod 2. Daher ist
C_0C_2C_3=y^2 mit y=31790.
Mit x=P_0P_2P_3=40745150406250
gilt x^2=y^2 mod N
ggT(x+y,N)=1, ggT(x-y,N)=3025991: triviale Faktorisierung.

a_24=5  p_24=2322426  b_25=1253  c_25=2519=11*229
a_25=1  p_25=2348051  b_26=1266  c_26=565=5*113
a_26=5  p_26=1958717  b_27=1559  c_27=1054=2*17*31
a_27=3  p_27=2172220  b_28=1603  c_28=433=433
a_28=7  p_28=2034302  b_29=1428  c_29=2279=2279
a_29=1  p_29=1180531  b_30=851  c_30=1010=2*5*101
a_30=2  p_30=1369373  b_31=1169  c_31=1643=1643
a_31=1  p_31=2549904  b_32=474  c_32=1705=5*11*31
a_32=1  p_32=893286  b_33=1231  c_33=886=2*443
a_33=3  p_33=2203771  b_34=1427  c_34=1117=1117
a_34=2  p_34=2274837  b_35=807  c_35=2126=2*1063
a_35=1  p_35=1452617  b_36=1319  c_36=605=5*11^2*1
p_35^2-d*q_35^2=(-1)^36*c_36 liefert 5. Relation und P_4=p_35, C_4=c_36
Matrix A:
1 1 1 1 1
0 1 1 0 0
1 0 1 1 1
0 1 0 0 0
0 0 1 0 0
Mit v=(1,1,1,0,1) gilt vA=0 mod 2. Daher ist
C_0C_1C_2C_4=y^2 mit y=514250.
Mit x=P_0P_1P_2P_4=935001398125435592400
gilt x^2=y^2 mod N
ggT(x+y,N)=3025991, ggT(x-y,N)=1: triviale Faktorisierung.

a_36=5  p_36=459949  b_37=1706  c_37=191=191
a_37=18  p_37=653726  b_38=1732  c_38=137=137
a_38=25  p_38=1673144  b_39=1693  c_39=1166=2*11*53
a_39=2  p_39=974023  b_40=639  c_40=2245=5*449
a_40=1  p_40=2647167  b_41=1606  c_41=199=199
a_41=16  p_41=964821  b_42=1578  c_42=2693=2693
a_42=1  p_42=585997  b_43=1115  c_43=662=2*331
a_43=4  p_43=282818  b_44=1533  c_44=1021=1021
a_44=3  p_44=1434451  b_45=1530  c_45=671=11*61
a_45=4  p_45=2994631  b_46=1154  c_46=2525=5^2*101
a_46=1  p_46=1403091  b_47=1371  c_47=454=2*227
a_47=6  p_47=2335204  b_48=1353  c_48=2633=2633
a_48=1  p_48=712304  b_49=1280  c_49=527=17*31
```

```
a_49=5  p_49=2870733  b_50=1355  c_50=2258=2*1129
a_50=1  p_50=557046  b_51=903  c_51=979=11*89
a_51=2  p_51=958834  b_52=1055  c_52=1954=2*977
a_52=1  p_52=1515880  b_53=899  c_53=1135=5*227
a_53=2  p_53=964603  b_54=1371  c_54=1010=2*5*101
a_54=3  p_54=1383698  b_55=1659  c_55=271=271
a_55=12  p_55=2439024  b_56=1593  c_56=1802=2*17*53
a_56=1  p_56=796731  b_57=209  c_57=1655=5*331
a_57=1  p_57=209764  b_58=1446  c_58=565=5*113
a_58=5  p_58=1845551  b_59=1379  c_59=1990=2*5*199
a_59=1  p_59=2055315  b_60=611  c_60=1333=1333
a_60=1  p_60=874875  b_61=722  c_61=1879=1879
a_61=1  p_61=2930190  b_62=1157  c_62=898=2*449
a_62=3  p_62=587472  b_63=1537  c_63=739=739
a_63=4  p_63=2254087  b_64=1419  c_64=1370=2*5*137
a_64=2  p_64=2069655  b_65=1321  c_65=935=5*11*17*1
p_64^2-d*q_64^2=(-1)^65*c_65 liefert 6. Relation und P_5=p_64, C_5=c_65
Matrix A:
1 1 1 1 1
0 1 1 0 0
1 0 1 1 1
0 1 0 0 0
0 0 1 0 0
1 0 1 1 1
Mit v=(0,0,1,0,0,1) gilt vA=0 mod 2. Daher ist
C_2C_5=y^2 mit y=935.
Mit x=P_2P_5=1892389049250
gilt x^2=y^2 mod N
ggT(x+y,N)=1009, ggT(x-y,N)=2999: nichttriviale Faktorisierung. Fertig!
```

Zum Suchen nach einer nichttrivialen Relation zwischen den Gleichungen $P_i^2 \equiv C_i \bmod N$, haben wir die Exponentenvektoren $v_i \in \mathbf{F}_2^n$ in eine Matrix A geschrieben und dann die Gleichung $uA = 0$ untersucht. Wir wollen dies noch etwas genauer ausführen:

(1) A sei eine $* \times n$-Matrix, $\widetilde{A}$ sei eine hinreichend große 'begleitende' Matrix mit folgender Charakterisierung: Sei $u = (u_{i0}, u_{i1}, \dots)$ die i-te Zeile von $\widetilde{A}$ und $(a_{i0}, a_{i1}, \dots, a_{i,n-1})$ die i-te Zeile von A. Dann ist

$$\prod_{\substack{0 \le j \le l \\ u_{ij}=1}} C_j = (-1)^{a_{i0}} p_1^{a_{i1}} \dots p_{n-1}^{a_{i,n-1}} \cdot \text{Quadrat} \quad (*).$$

(2) Findet man eine neue Relation $P_l^2 \equiv C_l \bmod N$, so hängt man den Exponentenvektor v_l an A und den l-ten Einheitsvektor an $\widetilde{A}$ an. Dann bleibt die Eigenschaft $(*)$ der Matrizen A und $\widetilde{A}$ erhalten.

(3) Die Zeilen r und s der Matrizen A und $\widetilde{A}$ liefern nach $(*)$ folgende Relationen:

$$\prod_{\substack{0 \le j \le l \\ u_{rj}=1}} C_j = (-1)^{a_{r0}} p_1^{a_{r1}} \dots p_{n-1}^{a_{r,n-1}} \cdot \text{Quadrat}$$

$$\prod_{\substack{0 \le j \le l \\ u_{sj}=1}} C_j = (-1)^{a_{s0}} p_1^{a_{s1}} \dots p_{n-1}^{a_{s,n-1}} \cdot \text{Quadrat}$$

Offensichtlich bleibt Eigenschaft $(*)$ erhalten, wenn die Zeilen r und s in A und $\widetilde{A}$ vertauscht werden. Durch Multiplikation der Gleichungen erhält man, wenn man die Quadrate auf der

linken Seite wegkürzt:

$$\prod_{\substack{0\le j\le l\\ u_{rj}+u_{sj}=1(\bmod 2)}} C_j = (-1)^{a_{r0}+a_{s0} \bmod 2} p_1^{a_{r1}+a_{s1} \bmod 2} \cdots p_{n-1}^{a_{r,n-1}+a_{s,n-1} \bmod 2} \cdot \text{Quadrat}$$

Dies entspricht aber genau der Addition der Zeilen r und s der Matrizen A und $\widetilde{A}$.

(4) Wir sehen also, daß Eigenschaft $(*)$ erhalten bleibt, wenn gleichzeitig in A und $\widetilde{A}$ Zeilen vertauscht oder Zeilen zueinander addiert werden. Wir bringen daher die Matrix A durch Zeilenvertauschungen und Zeilenadditionen auf Zeilenstufenform, wobei die Operationen gleichzeitig auch mit der Matrix $\widetilde{A}$ durchgeführt werden.

(5) Ist die unterste Zeile m in A identisch 0, so heißt dies

$$\prod_{\substack{0\le j\le l\\ u_{mj}=1}} C_j = \text{Quadrat},$$

also erhalten wir eine gesuchte Relation

$$\Big(\prod_{\substack{0\le j\le l\\ u_{mj}=1}} P_j\Big)^2 \equiv \Big(\sqrt{\prod_{\substack{0\le j\le l\\ u_{mj}=1}} C_j}\Big)^2 \bmod N.$$

Liefert dies die Faktorisierung von N, sind wir fertig. Im andern Fall ist die Relation überflüssig, also entfernen wir die Zeile m in A und $\widetilde{A}$.

Beispiel: $N = 1622623$

```
N=1622623 ist zusammengesetzt
Faktorbasis: -1 2 3 7 19 mit 5 Elementen

a_0=1273  p_0=1273  b_1=1273  c_1=2094=2*3*349
a_1=1  p_1=1274  b_2=821  c_2=453=3*151
a_2=4  p_2=6369  b_3=991  c_3=1414=2*7*101
a_3=1  p_3=7643  b_4=423  c_4=1021=1021
a_4=1  p_4=14012  b_5=598  c_5=1239=3*7*59
a_5=1  p_5=21655  b_6=641  c_6=978=2*3*163
a_6=1  p_6=35667  b_7=337  c_7=1543=1543
a_7=1  p_7=57322  b_8=1206  c_8=109=109
a_8=22  p_8=1296751  b_9=1192  c_9=1851=3*617
a_9=1  p_9=1354073  b_10=659  c_10=642=2*3*107
a_10=3  p_10=491101  b_11=1267  c_11=27=3^3*1
p_10^2-d*q_10^2=(-1)^11*c_11 liefert 1. Relation und P_0=p_10, C_0=c_11
Matrix A und Matrix AA:
1 0 1 0 0  1

a_11=94  p_11=461500  b_12=1271  c_12=266=2*7*19*1
p_11^2-d*q_11^2=(-1)^12*c_12 liefert 2. Relation und P_1=p_11, C_1=c_12
Matrix A und Matrix AA:
1 0 1 0 0  1 0
0 1 0 1 1  0 1
(A hat bereits Zeilenstufenform)

a_12=9  p_12=1399355  b_13=1123  c_13=1359=3^2*151
a_13=1  p_13=238232  b_14=236  c_14=1153=1153
a_14=1  p_14=14964  b_15=917  c_15=678=2*3*113
a_15=3  p_15=283124  b_16=1117  c_16=553=7*79
a_16=4  p_16=1147460  b_17=1095  c_17=766=2*383
a_17=3  p_17=480258  b_18=1203  c_18=229=229
```

```
a_18=10  p_18=1082171  b_19=1087  c_19=1926=2*3^2*107
a_19=1  p_19=1562429  b_20=839  c_20=477=3^2*53
a_20=4  p_20=841395  b_21=1069  c_21=1006=2*503
a_21=2  p_21=1622596  b_22=943  c_22=729=3^6*1
p_21^2-d*q_21^2=(-1)^22*c_22 liefert 3. Relation und P_2=p_21, C_2=c_22
Matrix A und Matrix AA:
1 0 1 0 0  1 0 0
0 1 0 1 1  0 1 0
0 0 0 0 0  0 0 1
Die letzte Zeile von A ist 0. Daher ist
C_2=y^2 mit y=27.
Mit x=P_2=1622596
gilt x^2=y^2 mod N
ggT(x+y,N)=1622623, ggT(x-y,N)=1: triviale Faktorisierung.
Daher wird die letzte Zeile von A entfernt. Es bleibt:
1 0 1 0 0  1 0 0
0 1 0 1 1  0 1 0

a_22=3  p_22=841314  b_23=1244  c_23=103=103
a_23=24  p_23=720033  b_24=1228  c_24=1113=3*7*53
a_24=2  p_24=658757  b_25=998  c_25=563=563
a_25=4  p_25=109815  b_26=1254  c_26=89=89
a_26=28  p_26=488331  b_27=1238  c_27=1011=3*337
a_27=2  p_27=1086477  b_28=784  c_28=997=997
a_28=2  p_28=1038662  b_29=1210  c_29=159=3*53
a_29=15  p_29=440177  b_30=1175  c_30=1522=2*761
a_30=1  p_30=1478839  b_31=347  c_31=987=3*7*47
a_31=1  p_31=296393  b_32=640  c_32=1229=1229
a_32=1  p_32=152609  b_33=589  c_33=1038=2*3*173
a_33=1  p_33=449002  b_34=449  c_34=1369=1369
a_34=1  p_34=601611  b_35=920  c_35=567=3^4*7*1
p_34^2-d*q_34^2=(-1)^35*c_35 liefert 4. Relation und P_3=p_34, C_3=c_35
Matrix A und Matrix AA:
1 0 1 0 0  1 0 0 0
0 1 0 1 1  0 1 0 0
1 0 0 1 0  0 0 0 1
Addiere Zeile 0 zu Zeile 2:
1 0 1 0 0  1 0 0 0
0 1 0 1 1  0 1 0 0
0 0 1 1 0  1 0 0 1

a_35=3  p_35=631212  b_36=781  c_36=1786=2*19*47
a_36=1  p_36=1232823  b_37=1005  c_37=343=7^3*1
p_36^2-d*q_36^2=(-1)^37*c_37 liefert 5. Relation und P_4=p_36, C_4=c_37
Matrix A und Matrix AA:
1 0 1 0 0  1 0 0 0 0
0 1 0 1 1  0 1 0 0 0
0 0 1 1 0  1 0 0 1 0
1 0 0 1 0  0 0 0 0 1
Addiere Zeile 0 zu Zeile 3:
1 0 1 0 0  1 0 0 0 0
0 1 0 1 1  0 1 0 0 0
0 0 1 1 0  1 0 0 1 0
0 0 1 1 0  1 0 0 0 1
```

```
Addiere Zeile 2 zu Zeile 3:
1 0 1 0 0  1 0 0 0 0
0 1 0 1 1  0 1 0 0 0
0 0 1 1 0  1 0 0 1 0
0 0 0 0 0  0 0 0 1 1
Die letzte Zeile von A ist 0. Daher ist
C_3C_4=y^2 mit y=441.
Mit x=P_3P_4=741679877853
gilt x^2=y^2 mod N
ggT(x+y,N)=907, ggT(x-y,N)=1789: Faktorisierung erreicht.
```

Wir erhalten die Primfaktorzerlegung $N = 907 \cdot 1789$.

Frage: Wie sollte die Faktorbasis $-1, p_1, p_2, \ldots, p_{n-1}$ gewählt werden?

(1) Gesuchte Relationen $P_i^2 \equiv C_i \bmod N$ erhalten wir aus Gleichungen

$$\widetilde{p}_j^2 - N\widetilde{q}_j^2 = (-1)^{j+1} c_{j+1},$$

wo $\frac{\widetilde{p}_j}{\widetilde{q}_j}$ Näherungsbrüche in der Kettenbruchentwicklung von $\sqrt{N}$ sind.
Ist p eine (kleine) Primzahl mit $p|c_{j+1}$, so folgt

$$\widetilde{p}_j^2 \equiv N\widetilde{q}_j^2 \bmod p.$$

Wegen $\mathrm{ggT}(\widetilde{p}_j, \widetilde{q}_j) = 1$ ist $\widetilde{q}_j \not\equiv 0 \bmod p$ und damit

$$N \equiv (\frac{\widetilde{p}_j}{\widetilde{q}_j})^2 \bmod p,$$

also ist N ein Quadrat modulo p (oder $p = 2$), d.h. mit dem Legendre-Symbol gilt

$$p = 2 \quad \text{oder} \quad \left(\frac{N}{p}\right) = 1.$$

Da sich c_{j+1} vollständig mit den Primzahlen der Faktorbasis faktorisieren lassen sollte, sind also nur Primzahlen mit $\left(\frac{N}{p}\right) = 1$ oder $p = 2$ interessant, d.h. nur solche sollte man sinnvollerweise in die Faktorbasis aufnehmen.

(2) Wählt man eine große Faktorbasis, so erhält man schneller Relationen. Dafür braucht man mehr Relationen, außerdem werden die Matrizen A und $\widetilde{A}$ größer. Wählt man die Faktorbasis zu klein, findet man vielleicht keine Relationen.

Beispiel: Wir wollen die zusammengesetzte 30-stellige Zahl

$$N = 86871944523292762551433641798 1$$

faktorisieren. Im folgenden bezeichnet '.', daß eine neue Relation gefunden wurde, '*', daß in der umgeformten Matrix A eine Zeile identisch 0 ist, also ein Faktorisierungsversuch gemacht werden kann. Wir testen Faktorbasen mit 50, 100, 500 und 1000 Elementen.

(1)
```
Faktorisierung von N=86871944523292762551433641798 1
Faktorbasis:
-1 2 3 5 7 19 31 37 71 79 89 97 107 109 113 131 137 139 151 167 173 179
181 199 211 223 229 233 241 251 257 263 269 283 293 317 347 349 353 359
367 373 383 421 439 463 487 503 521 523
Die Faktorbasis hat 50 Elemente.
.........................................*......*.*.*
Nichttriviale Faktorisierung N=N2*N3 mit
N2=661177905109907 (prim)
N3=1313896666115183 (prim)
Zeit: 32 sec
```

(2)
```
Faktorisierung von N=868719445232927625514336417981
Die Faktorbasis hat 100 Elemente.
.................................*.*.............................
.......................*
Nichttriviale Faktorisierung N=N2*N3 mit
N2=1313896666115183 (prim)
N3=661177905109907 (prim)
Zeit: 13 sec
```
(3)
```
Faktorisierung von N=868719445232927625514336417981
Die Faktorbasis hat 500 Elemente.
.....................................................................
................*....................................................
.....................................................................
.....................................................................
.........................................*...........................
.....................................................*
Nichttriviale Faktorisierung N=N2*N3 mit
N2=661177905109907 (prim)
N3=1313896666115183 (prim)
Zeit: 13 sec
```
(4)
```
Faktorisierung von N=868719445232927625514336417981
Die Faktorbasis hat 1000 Elemente.
.....................................................................
.....................................................................
.....................................................*...........
................*....................................................
.....................................................................
..........................*..........................................
.....................................................................
......*.....................*.....................................
.....................................................................
................*.................................*
Nichttriviale Faktorisierung N=N2*N3 mit
N2=1313896666115183 (prim)
N3=661177905109907 (prim)
Zeit: 28 sec
```

Beispiel: Wir wollen die (zusammengesetzte) 40-stellige Zahl

$$N = 708079746578928884362571476578114177362 3$$

faktorisieren und wählen jeweils eine Faktorbasis mit 100, 500 bzw. 1000 Elementen.

(1)
```
Faktorisierung von N=708079746578928884362571476578114177362 3
Die Faktorbasis hat 100 Elemente.
.....................................................................
.......................*
Nichttriviale Faktorisierung N=N2*N3 mit
N2=120832703460127480513 (prim)
N3=58600008631982804471 (prim)
Zeit: 3863 sec
```
(2)
```
Faktorisierung von N=708079746578928884362571476578114177362 3
Die Faktorbasis hat 500 Elemente.
.....................................................................
.....................................................................
.....................................................................
```

```
.....................................................................
.....................................................................
.....................................................................
.....................................................*
Nichttriviale Faktorisierung N=N2*N3 mit
N2=58600008631982804471 (prim)
N3=120832703460127480513 (prim)
Zeit: 784 sec
```

(3)
```
Faktorisierung von N=7080797465789288843625714765781141773623
Die Faktorbasis hat 1000 Elemente.
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................*
Nichttriviale Faktorisierung N=N2*N3 mit
N2=120832703460127480513 (prim)
N3=58600008631982804471 (prim)
Zeit: 823 sec
```

Wir haben für diese Zahl noch weitere Faktorisierungsversuche mit anderen Faktorbasen gemacht und dabei folgende Rechenzeiten in Abhängigkeit der Größe der Faktorbasis erhalten:

#Faktorbasis	100	400	500	600	700	800	900	1000
Zeit in sec	3863	826	784	752	752	752	784	823

(Maple faktorisiert die Zahl in 856.5 sec.)

9. Kettenbrüche und RSA-Schwächen

Erinnerung:

(1) Sind p und q verschiedene ungerade Primzahlen und $N = pq$, wählt man natürliche Zahlen e und d mit

$$ed \equiv 1 \bmod \varphi(N) \quad (\text{und } \varphi(N) = \varphi(pq) = (p-1)(q-1)),$$

so ist die Abbildung

$$E_{(N,e)} : \mathbf{Z}/(N) \to \mathbf{Z}/(N), \quad x \mapsto x^e \bmod N$$

bijektiv mit der Umkehrabbildung

$$D_{(N,d)} : \mathbf{Z}/(N) \to \mathbf{Z}/(N), \quad y \mapsto y^d \bmod N.$$

(2) Kennt jemand (N, e), kann aber N nicht faktorisieren, so kann er heutzutage im allgemeinen auch nicht (N, d) bestimmen, d.h. er kann $E_{(N,e)}$ anwenden, nicht jedoch $D_{(N,d)}$.

(3) Dies benutzt man zur Konstruktion des RSA-Public-Key-Kryptosystems mit öffentlichen Schlüsseln (N, e), privaten Schlüsseln (N, d), Verschlüsselungsabbildungen $E_{(N,e)}$, Entschlüsselungsabbildungen $D_{(N,d)}$.

Hier sind mögliche Maple-Funktionen für RSA-Verschlüsselung und RSA-Entschlüsselung:

```
# ‘bf:=convert(zk,bytes);’ wandelt eine Zeichenkette ‘zk’ in eine
# Bytefolge ‘bf’ um.
# ‘bf:=readbytes("datei",infinity);’ liefert die Bytefolge ‘bf’ der
# Datei ‘datei’.
# ‘zk:=convert(bf,bytes);’ wandelt eine Bytefolge ‘bf’ in eine
# Zeichenkette ‘zk’ um.
# writebytes("datei",bf);’ schreibt die Bytefolge ‘bf’ in eine Datei
# ‘datei’.

rsa_encrypt:=proc(bytefolge,N,e)
  local bl, bf, zuviel, i, zf, a, j, b;
  # bl ist die Blocklaenge
  bl:=0; while 256^(bl+1)<N do bl:=bl+1; od; if bl>255 then bl:=255; fi;

  bf:=bytefolge;
  # Die Bytefolge wird ergaenzt, damit Anzahl=0 mod Blocklaenge ist.
  # ‘zuviel’ Bytes muessen spaeter wieder weggestrichen werden.
  zuviel:=bl-(nops(bf) mod bl);
  for i from 1 to zuviel-1 do bf:=[op(bf),10]; od;
  bf:=[op(bf),zuviel];

  zf:=[]; # zf wird die auszugebende Zahlenfolge
  for i from 1 to nops(bf)/bl do
    # jeweils bl Bytes werden in eine Zahl a umgewandelt
    a:=0;for j from 1 to bl do a:=256*a+bf[(i-1)*bl+j]; od;
    b:=Power(a,e) mod N; # Verschluesselung b=a^e mod N
    zf:=[op(zf),b];
  od;
  zf;
end;

rsa_decrypt:=proc(zahlenfolge,N,d)
  local bl, bf, i, b, a, bf1, j, zuviel;
  bl:=0; while 256^(bl+1)<N do bl:=bl+1; od; if bl>255 then bl:=255; fi;

  bf:=[]; # bf wird die auszugebende Bytefolge
  for i from 1 to nops(zahlenfolge) do
    b:=zahlenfolge[i];
    a:=Power(b,d) mod N;
    bf1:=[]; # a wird in die bl-elementige Bytefolge bf1 umgewandelt
    for j from 1 to bl do
      bf1:=[a mod 256,op(bf1)];
      a:=iquo(a,256);
    od;
    bf:=[op(bf),op(bf1)];
  od;
  zuviel:=bf[nops(bf)]; # ‘zuviel’ Bytes werden weggestrichen
  for i from 1 to zuviel do bf:=subsop(nops(bf)=NULL,bf); od;
  bf;
end;
```

Ist (N, e) ein öffentlicher RSA-Schlüssel, (N, d) der zugehörige private, so sollte man aus der Kenntnis von (N, e) nicht auf (N, d) schließen können.

(1) N sollte nicht faktorisiert werden können, da man wegen
$$d \equiv \frac{1}{d} \bmod (p-1)(q-1)$$
sonst sofort (N, d) hätte.
(2) d sollte nicht zu klein sein, so daß man durch Probieren von $d = 3, 5, 7, \ldots$ auf d schließen kann. (Man testet z.B., ob $2^{ed} \equiv 2 \bmod N$ gilt.)
(3) Wir werden jetzt allgemeiner sehen, daß überhaupt $d \leq N^{\frac{1}{4}}$ gefährlich ist.

Sei (N, e) ein öffentlicher RSA-Schlüssel, (N, d) der zugehörige private Schlüssel. Wegen $ed \equiv 1 \bmod \varphi(N)$ ist $k = \frac{ed-1}{\varphi(N)}$ eine natürliche Zahl und damit
$$ed - k\varphi(N) = 1,$$
also
$$\frac{e}{\varphi(N)} - \frac{k}{d} = \frac{1}{d\varphi(N)}.$$
Ist N groß, so wird also $\frac{e}{\varphi(N)} \approx \frac{k}{d}$ gelten. Wir wollen jetzt zunächst eine Abschätzung für $\varphi(N)$ geben.

LEMMA. *Ist $N = pq$ Produkt zweier ungerader Primzahlen $p < q$, $s = p + q$, so ist $\varphi(N) = N + 1 - s$ und es gelten die Abschätzungen:*

(1)
$$\lfloor\sqrt{4N}\rfloor + 1 \leq s \leq \frac{1}{3}N + 3 \quad \textit{und} \quad \frac{2}{3}N - 2 \leq \varphi(N) \leq N - \lfloor\sqrt{4N}\rfloor.$$
(2) *Ist $p < q < 2p$, so gilt*
$$2\sqrt{N} < s < \frac{3}{2}\sqrt{2} \cdot \sqrt{N} \quad \textit{und} \quad N + 1 - 2\sqrt{N} < \varphi(N) < N + 1 - \frac{3}{2}\sqrt{2}\sqrt{N}.$$
(3) *Ist $p < q < 3.9p$, so gilt*
$$2\sqrt{N} < s < 2.48121125\sqrt{N} \quad \textit{und} \quad N + 1 - 2\sqrt{N} < \varphi(N) < N + 1 - 2.48121145\sqrt{N}.$$

Beweis:

(1) Es gilt
$$\varphi(N) = (p-1)(q-1) = pq - p - q + 1 = N + 1 - (p+q) = N + 1 - s.$$
(2) Wir schreiben $q = x\sqrt{N}$, $p = \frac{1}{x}\sqrt{N}$ mit $x > 1$. Dann ist
$$s = (x + \frac{1}{x})\sqrt{N}.$$
Bei festem N ist die Funktion s in Abhängigkeit von x streng monoton wachsend wegen $\frac{\partial s}{\partial x} = (1 - \frac{1}{x^2})\sqrt{N}$. Also erhalten wir
$$s > 2\sqrt{N} \quad \text{und} \quad s \geq \lfloor\sqrt{4N}\rfloor + 1.$$
Die obere Schranke kommt von einer möglichen Zerlegung $N = 3q$, also
$$s \leq 3 + \frac{1}{3}N.$$
Somit
$$2\sqrt{N} < \lfloor\sqrt{4N}\rfloor + 1 \leq s \leq 3 + \frac{1}{3}N.$$
Für die Eulersche φ-Funktion impliziert dies
$$N + 1 - 2\sqrt{N} > N - \lfloor\sqrt{4N}\rfloor \geq \varphi(N) \geq \frac{2}{3}N - 2.$$

(3) Die Bedingung $p < q < \mu p$ ist dann äquivalent mit $\frac{1}{x} < x < \frac{\mu}{x}$, also $1 < x < \sqrt{\mu}$. Da die Funktion

$$s = p + q = (x + \frac{1}{x})\sqrt{N}$$

(bei festem N) für $x > 0$ streng monoton steigend ist, folgt die Abschätzung

$$2\sqrt{N} < s < (\sqrt{\mu} + \frac{1}{\sqrt{\mu}})\sqrt{N}.$$

Wählt man $\mu = 2$ ergibt sich $s < \frac{3}{2}\sqrt{2}\sqrt{N}$, bei der Wahl von $\mu = 3.9$ ergibt sich $s < 2.48121145\sqrt{N}$. ■

Liegen die Primzahlen p und q eines RSA-Schlüssels nicht zu weit auseinander, erhalten wir also

$$\frac{k}{d} \approx \frac{e}{\varphi(N)} \approx \frac{e}{N} \approx \frac{e}{N - \lfloor\sqrt{4N}\rfloor}.$$

Wir betrachten ein Beispiel:

Beispiel: Wir nehmen die 20-stellige Zahl

$$N = 32758659611582346361 = 4072951217 \cdot 8042978633$$

mit

$$\varphi(N) = (p-1)(q-1) = 32758659599466416512.$$

1. Beispiel:

$d = 6231628522119231213 \approx N^{0.963}$, $e = 28554566343820204901 \approx N^{0.997}$, $k = 5431890444864056951$.

$$\begin{aligned}
\frac{k}{d} &= [0,1,6,1,3,1,4,3,1,12,7,2,\\
&\quad 2,17,1,2,12,7,88,1,1,3,1,6,104,1,1,1,1,1,3,1,1,2,8,1,3]\\
\frac{e}{\varphi(N)} &= [0,1,6,1,3,1,4,3,1,12,7,2,\\
&\quad 2,17,1,2,12,7,88,1,1,3,1,6,104,1,1,1,1,1,3,1,1,2,8,1,3,5]\\
\frac{e}{N} &= [0,1,6,1,3,1,4,3,1,12,5,1,\\
&\quad 15,4,2,1,12,2,1,4,2,9,6,3,1,37,1,9,1,1,3,1,1,1,1,1,2,1,2,3,1,22]\\
\frac{e}{N - \lfloor\sqrt{4N}\rfloor} &= [0,1,6,1,3,1,4,3,1,12,7,3,\\
&\quad 3,2,1,23,2,1,2,2,5,9,21,1,1,5,1,1,1,1,1,24,1,10,1,1,1,14,1,5,1,2]
\end{aligned}$$

Die betrachteten Kettenbruchentwicklungen fangen gleich an. $\frac{k}{d}$ ist Näherungsbruch von $\frac{e}{\varphi(N)}$.

2. Beispiel:

$$d = 5723518015 \approx N^{0.500}, \quad e = 25078774849122696639 \approx N^{0.994}, \quad k = 4381706132$$

$$\begin{aligned}
\frac{k}{d} &= [0,1,3,3,1,3,3,1,1,2,3,1,10,1,2,1,\\
&\quad 3,1,1,2,2,2,7,1,6,3]\\
\frac{e}{\varphi(N)} &= [0,1,3,3,1,3,3,1,1,2,3,1,10,1,2,1,\\
&\quad 3,1,1,2,2,2,7,1,6,2,1,5723518212]\\
\frac{e}{N} &= [0,1,3,3,1,3,3,1,1,2,3,1,9,1,2,1,\\
&\quad 1,1,21,1,5,16,7,10,1,143,1,1,4,1,2,2,1,1,1,7,3,2,2,5,1,3,2]\\
\frac{e}{N - \lfloor\sqrt{4N}\rfloor} &= [0,1,3,3,1,3,3,1,1,2,3,1,10,1,2,13,\\
&\quad 4,1,3,3,8,2,7,2,2,16,6,1,1,1,2,1,1,2,10,2,2,1,4,157]
\end{aligned}$$

$\frac{k}{d}$ ist Näherungsbruch von $\frac{e}{\varphi(N)}$. Für $\frac{k}{d}$ ist $\frac{e}{N-\lfloor\sqrt{4N}\rfloor}$ eine bessere Approximation als $\frac{e}{N}$.

3. Beispiel:

$$d = 118575 \approx N^{0.260}, \quad e = 30770071517687295567 \approx N^{0.999}, \quad k = 111377$$

$$\begin{aligned}
\frac{k}{d} &= [0,1,15,2,8,1,6,1,5,8] \\
\frac{e}{\varphi(N)} &= [0,1,15,2,8,1,6,1,5,7,1,276269530672286] \\
\frac{e}{N} &= [0,1,15,2,8,1,6,1,5,20, \\
&\quad 6,1,6,1,1,1,15,5,2,1,1,1,1,4,22,3,2,1,1,22,2,6,1,3,1,2,2,3,1,1,1,2] \\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0,1,15,2,8,1,6,1,5,8, \\
&\quad 3,1,1,2,2,2,4,3,1,7,6,1,4,1,1,3,1,1,1,2,4,1,1,1,2,2,1,1,4,1,1,1,2,1,8,2,28]
\end{aligned}$$

Wir sehen jetzt, daß $\frac{k}{d}$ Näherungsbruch in der Kettenbruchentwicklung von $\frac{e}{N-\lfloor\sqrt{4N}\rfloor}$ ist. Kennt man den öffentlichen Schlüssel (N,e), kann man also $\frac{k}{d}$ und damit den privaten Schlüssel (N,d) bestimmen.

Wir verallgemeinern das letzte Beispiel in einem Satz, dessen erste Version auf M. Wiener zurückgeht.

SATZ. *Sei $N = pq > 3000$ mit $p < q < 3.9p$, seien e, d natürliche Zahlen mit $1 < e, d < \varphi(N)$ und $ed \equiv 1 \bmod N$, sowie $k = \frac{ed-1}{\varphi(N)}$. Dann gilt die Implikation*

$$d \leq N^{\frac{1}{4}} \quad \Longrightarrow \quad \left|\frac{e}{N-\lfloor\sqrt{4N}\rfloor} - \frac{k}{d}\right| < \frac{1}{2d^2},$$

ist also $d \leq N^{0.25}$, so kommt $\frac{k}{d}$ als Näherungsbruch in der Kettenbruchentwicklung von $\frac{e}{N-\lfloor\sqrt{4N}\rfloor}$ vor.

Beweis:

(1) Das vorangegangene Lemma liefert mit $s = p + q$ die Abschätzung

$$\lfloor\sqrt{4N}\rfloor + 1 \leq s < 2.48121145\sqrt{N}.$$

(2) Im Fall $s = \lfloor\sqrt{4N}\rfloor + 1$ ist $\varphi(N) = N + 1 - s = N - \lfloor\sqrt{4N}\rfloor$ und damit

$$\left|\frac{e}{N-\lfloor\sqrt{4N}\rfloor} - \frac{k}{d}\right| = \left|\frac{e}{\varphi(N)} - \frac{k}{d}\right| = \left|\frac{e}{\varphi(N)} - \frac{e-\frac{1}{d}}{\varphi(N)}\right| = \frac{1}{d\varphi(N)}$$

Für $N \geq 10$ ist $2N^{\frac{1}{4}} + 2\sqrt{N} < N$, also

$$2d \leq 2N^{\frac{1}{4}} < N - 2\sqrt{N} \leq N - \lfloor\sqrt{4N}\rfloor = \varphi(N)$$

und damit

$$\left|\frac{e}{N-\lfloor\sqrt{4N}\rfloor} - \frac{k}{d}\right| = \frac{1}{d\varphi(N)} < \frac{1}{2d^2},$$

was gezeigt werden sollte.

(3) Im Fall $s > \lfloor\sqrt{4N}\rfloor + 1$ gilt

$$\begin{aligned}
k(N-\lfloor\sqrt{4N}\rfloor) - ed &= k(N-\lfloor\sqrt{4N}\rfloor) - (1 + k\varphi(N)) \geq \\
&\geq k(N-\lfloor\sqrt{4N}\rfloor) - k - k\varphi(N) = \\
&= k(N - \lfloor\sqrt{4N}\rfloor - 1 - (N + 1 - s)) = k(s - \lfloor\sqrt{4N}\rfloor - 2) \geq 0
\end{aligned}$$

und damit

$$\frac{k}{d} \geq \frac{e}{N-\lfloor\sqrt{4N}\rfloor}.$$

Dann ist

$$\begin{aligned}
|\frac{e}{N-\lfloor\sqrt{4N}\rfloor}-\frac{k}{d}| &= \frac{k}{d}-\frac{e}{N-\lfloor\sqrt{4N}\rfloor} < \frac{k}{d}-\frac{e}{N+1-2\sqrt{N}} = \\
&= \frac{e-\frac{1}{d}}{\varphi(N)}-\frac{e}{N+1-2\sqrt{N}} < \frac{e}{\varphi(N)}-\frac{e}{N+1-2\sqrt{N}} = \\
&= \frac{e((N+1-2\sqrt{N})-(N+1-s))}{\varphi(N)(N+1-2\sqrt{N})} < \frac{s-2\sqrt{N}}{N+1-2\sqrt{N}} < \\
&< \frac{(2.48121145-2)\sqrt{N}}{N+1-2\sqrt{N}} = \frac{0.48121145\sqrt{N}}{N+1-2\sqrt{N}}
\end{aligned}$$

Nun gilt

$$\begin{aligned}
\frac{0.48121145\sqrt{N}}{N+1-2\sqrt{N}} \le \frac{1}{2\sqrt{N}} \quad &\iff \quad 0.9624229N \le N+1-2\sqrt{N} \\
&\iff \quad 2\sqrt{N}-1 \le 0.0375771N \quad \iff \quad N \ge 2780
\end{aligned}$$

Damit folgt für $N \ge 3000$ und $d \le N^{\frac{1}{4}}$ zunächst $d^2 \le \sqrt{N}$ und damit

$$|\frac{e}{N-\lfloor\sqrt{4N}\rfloor}-\frac{k}{d}| < \frac{1}{2d^2},$$

was gezeigt werden sollte. ■

Wir geben jetzt noch konkret einen Algorithmus an, der versucht aus der Kettenbruchentwicklung von $\frac{e}{N-\lfloor\sqrt{4N}\rfloor}$ den privaten Schlüssel (N,d) zu berechnen. Dazu brauchen wir ein Lemma:

LEMMA. *Sei $N=pq$ mit ungeraden Primzahlen $p<q$, seien e,d natürliche Zahlen mit $1<e,d<\varphi(N)$ und $ed\equiv 1 \bmod N$.*

(1) *Setzt man nacheinander*

$$k=\frac{ed-1}{\varphi(N)}, \quad s=N+1-\frac{ed-1}{k}, \quad D=s^2-4N,$$

so gilt

$$ed\equiv 1 \bmod k, \quad D \text{ ist ein Quadrat} \quad \text{und} \quad p=\frac{s-\sqrt{D}}{2}, \quad q=\frac{s+\sqrt{D}}{2}.$$

(2) *Sind umgekehrt k',d' natürliche Zahlen, setzt man*

$$s'=N+1-\frac{ed'-1}{k'}, \quad D'=s'^2-4N,$$

gilt $ed'\equiv 1 \bmod k'$, und ist D' ein Quadrat, so ist

$$p=\frac{s'-\sqrt{D'}}{2} \quad \text{und} \quad q=\frac{s'+\sqrt{D'}}{2}.$$

Beweis: 1. Wir haben $ed-1=k\varphi(N)$, was sofort $k|ed-1$ zeigt. Dann ist

$$s=N+1-\frac{ed-1}{k}=N+1-\varphi(N)=pq+1-(p-1)(q-1)=p+q,$$

also

$$D=s^2-4N=(p+q)^2-4pq=(q-p)^2$$

ein Quadrat und

$$\frac{s-\sqrt{D}}{2}=\frac{(p+q)-(q-p)}{2}=p \quad \text{und} \quad \frac{s+\sqrt{D}}{2}=\frac{(p+q)+(q-p)}{2}=q,$$

was die erste Behauptung beweist.

2. Wegen $ed'\equiv 1 \bmod k'$ sind s' und D' ganze Zahlen und damit wegen $D'\equiv s' \bmod 2$ auch

$$p'=\frac{s'-\sqrt{D'}}{2} \quad \text{und} \quad q'=\frac{s'+\sqrt{D'}}{2}.$$

Wegen $D' = s'^2 - 4N$ ist $D' > s'^2$, also $1 \le p' \le q'$. Aus

$$p'q' = \frac{s' - \sqrt{D'}}{2} \cdot \frac{s' + \sqrt{D'}}{2} = \frac{s'^2 - D'}{4} = N$$

sieht man, daß es nur die beiden Möglichkeiten

$$p' = 1, q' = N \quad \text{oder} \quad p' = p, q' = q$$

gibt. Wäre $p' = 1, q' = N$, so wäre $s' = p' + q' = N + 1$ und damit $ed' = 1$, was der Voraussetzung $e > 1$ widerspricht. Also bleibt nur die zweite Möglichkeit, womit die Behauptung bewiesen ist. ■

Algorithmus: Hat man einen öffentlichen RSA-Schlüssel (N, e) gegeben, kann man nach folgendem Verfahren versuchen, die Faktorisierung $N = pq$ und den privaten Schlüssel d zu finden:

(1) Bestimme die Näherungsbrüche

$$\frac{k_0}{d_0}, \quad \frac{k_1}{d_1}, \quad \frac{k_2}{d_2}, \quad \ldots, \quad \frac{k_n}{d_n}$$

der Kettenbruchentwicklung von

$$\frac{e}{N - \lfloor \sqrt{4N} \rfloor}$$

und setze $i = 0$

(2) Setze $i := i + 1$. Ist $i > n$ beendige ohne Erfolg.

(3) Ist $ed_i \not\equiv 1 \bmod k_i$, gehe zu 2.

(4) Setze

$$s_i = N + 1 - \frac{ed_i - 1}{k_i} \quad \text{und} \quad D_i = s_i^2 - 4N.$$

(5) Ist D_i kein Quadrat, gehe zu 2.

(6) Ist D_i ein Quadrat, gib

$$p = \frac{s_i - \sqrt{D_i}}{2}, \quad q = \frac{s_i + \sqrt{D_i}}{2} \quad \text{und} \quad d \equiv \frac{1}{e} \bmod (p-1)(q-1)$$

als gesuchte Lösung aus.

Die folgende Maple-Funktion 'kb_rsa(N,e)' realisiert diesen Algorithmus:

```
# kb_rsa versucht aus dem oeffentlichen RSA-Schluessel (N,e) mittels
# Kettenbruechen die Faktorisierung N=pq zu bestimmen. Gelingt dies,
# wird [p,q,d] mit dem privaten Schluessel d ausgegeben, sonst [].
kb_rsa:=proc()
  local N, e;
  N:=args[1]; e:=args[2];
  w4N:=isqrt(4*N); if w4N^2>4*N then w4N:=w4N-1; fi;
  b0:=e; b1:=N-w4N;
  ki:=1; ki1:=0;
  di:=0; di1:=1;
  i:=-1;
  while b1>0 do
    i:=i+1;
    b2:=b0 mod b1; ai:=(b0-b2)/b1; b0:=b1; b1:=b2;
    k:=ai*ki+ki1; ki1:=ki; ki:=k;
    d:=ai*di+di1; di1:=di; di:=d;
    if i>0 and e*di mod ki=1 then
      si:=N+1-(e*di-1)/ki;
      Di:=si^2-4*N;
      if issqr(Di) then
        p:=(si-isqrt(Di))/2; q:=(si+isqrt(Di))/2;
        printf("ln(d)/ln(N)=%f\n",ln(di)/ln(N));
        return([p,q,di]);
```

```
      fi;
    fi;
  od;
  [];
end;
```

Beispiele: In den folgenden Beispielen ist $\frac{\ln(d)}{\ln(N)} \approx 0.25$. Manchmal ist der Algorithmus erfolgreich, manchmal nicht.

(1) Beispiel:

$$
\begin{aligned}
N &= 82257510389380447225013058328054960189274459591361757147434743211245 63\\
p &= 63642133071342787856743220237688643\\
q &= 1292500838983033992037693593175314 41\\
\frac{q}{p} &= 2.030889\\
e &= 65798253881547077561033458280194449501897084997263138289422141262433 3\\
d &= 1111235133951504677\\
\frac{\ln(d)}{\ln(N)} &= .258110\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0,12,1,1,169,4,1,7,3,2,1,1,1,1,1,1,1,1,1,7,1,1,6,24,1,8,1,2,1,8,3,1,2,1,1,2,\\
&\quad 211,1,6,4,15,5,6,14,2,1,32,1,5,1,1,7,7,17,1,1,2,1,1,2,21,1,2,1,32,2,4,3,37,4,\\
&\quad 6,9,2,2,1,1,3,5,1,7,1,4,8,2,2,3,1,40,2,1,7,10,1,1,54,1,24,1,1,5,1,1,4,3,3,1,10,\\
&\quad 1,5,1,61,1,5,3,2,1,10,1,1,4,3,2,2,2,8,1,1,1,7,1,3,2,4]\\
\frac{k}{d} &= [0,12,1,1,169,4,1,7,3,2,1,1,1,1,1,1,1,1,1,7,1,1,6,24,1,8,1,2,1,8,3,1,2,1,1,2,\\
&\quad 212]
\end{aligned}
$$

Der Algorithmus hat Erfolg.

(2) Beispiel:

$$
\begin{aligned}
N &= 16267828224447286944047076614006829226087184613526395634552214561790 97\\
p &= 24257763286132280790960960610552741\\
q &= 67062358687238516099099348034732517\\
\frac{q}{p} &= 2.764573\\
e &= 93287748061254136097346812144579989544445924143659894076836187483092 3\\
d &= 489467349331406467\\
\frac{\ln(d)}{\ln(N)} &= .255590\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0,1,1,2,1,9,2,1,1,2,12,63,1,6,3,1,2,2,1,2,5,5,1,1,1,104,34,1,1,2,1,18,1,1,1,7,\\
&\quad 24,1,6,1,13,2,9,63,4,15,2,2,1,22,3,5,3,9,3,1,2,1,2,4,1,1,54,1,1,1,4,70,1,6,2,1,\\
&\quad 29,1,2,1,2,42,4,4,3,9,5,2,1,2,17,1,20,1,2,2,3,8,1,2,1,4,4,3,2,2,6,5,2,1,5,1,2,\\
&\quad 3,8,29,1,2,1,22,2,24,1,1,4,3,1,4,6]\\
\frac{k}{d} &= [0,1,1,2,1,9,2,1,1,2,12,63,1,6,3,1,2,2,1,2,5,5,1,1,1,104,34,1,1,2,1,18,1,1,1,6]
\end{aligned}
$$

Der Algorithmus hat keinen Erfolg.

(3) Beispiel:

$$
\begin{aligned}
N &= 584848245832818966184145525652796528482850386898810478316658617713489 9\\
p &= 46936341332614722240707748528878477\\
q &= 12460456636112466525058138834595628 7\\
\frac{q}{p} &= 2.654757\\
e &= 43142442550884520582247716941939330380060285181477518135142620108715 1\\
d &= 699303130395894943\\
\frac{\ln(d)}{\ln(N)} &= .255775\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0,13,1,1,3,1,17,1,13,1,1,34,4,21,2,3,3,1,3,1,1,1,2,1,1,2,1,11,1,2,62,1,17,2,3,1,\\
&\quad 7,1,16,1,4,1,8,1,1,1,4,1,2,1,1,3,3,4,6,2,1,2,2,1,1,15,1,1,1,14,1,9,1,3,2,56,\\
&\quad 2,49,1,6,1,3,1,2,4,2,4,4,1,938,1,1,22,4,2,1,11,10,2,1,1,6,2,1,1,15,249,1,1,\\
&\quad 22,1,1,4,1,1,1,1,2,2,3,1,5,2,12,22,4,4,2,1,5,1,1,12,2,3,1,2,1,16]\\
\frac{k}{d} &= [0,13,1,1,3,1,17,1,13,1,1,34,4,21,2,3,3,1,3,1,1,1,2,1,1,2,1,11,1,2,62,1,17,2,4]
\end{aligned}
$$

Der Algorithmus hat Erfolg.

(4) Beispiel:

$$
\begin{aligned}
N &= 628132752154033093821256475001725389032356982878193434482485231932719 1\\
p &= 54625846065374370175746762876463151\\
q &= 11498819650359373865157379979463804 1\\
\frac{q}{p} &= 2.105014\\
e &= 26722347180210576408089314460079188881804027372274115107185929368163 41\\
d &= 885468195135947261\\
\frac{\ln(d)}{\ln(N)} &= .257130\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0,2,2,1,5,1,3,2,1,1,3,12,1,1,1,2,3,1,4,1,1,5,19,65,168,1,3,1,1,1,5,1,4,3,1,1,\\
&\quad 2,1,2,2,27,3,8,1,2,1,1,1,1,3,4,3,1,1,2,1,5,2,1,2,7,3,1,3,1,1,8,1,1,2,1,2,1,2,\\
&\quad 2,2,4,2,3,1,2,1,37,1,3,1,2,1,5,20,15,1,3,1,117,1,1,1,1,1,2,6,1,2,19,1,3,2,1,2,\\
&\quad 1,2,20,1,3,2,9548,3,8,1,27,1,1,1,19,12,2,1,18,2,9,1,1,1,1,1,7,2,2,2,2,1,1,1,2,4]\\
\frac{k}{d} &= [0,2,2,1,5,1,3,2,1,1,3,12,1,1,1,2,3,1,4,1,1,5,19,65,168,1,3,1,1,1,5,1,4,3,1,1,\\
&\quad 2,1,3]
\end{aligned}
$$

Der Algorithmus hat keinen Erfolg.

(5) Beispiel:

$$
\begin{aligned}
N &= 730775154928381039546551876976323682062989535857435467027483489450270 1\\
p &= 57783440945553532111201955741804617\\
q &= 126467919350277909294697866451674053\\
\frac{q}{p} &= 2.188653\\
e &= 29647158453458851092601160760591809938592627715881348674803234993660 67\\
d &= 650382652301446555\\
\frac{\ln(d)}{\ln(N)} &= .254970\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0,2,2,6,1,1,1,1,1,1,16,1,404,3,2,4,4,3,1,3,2,2,3,3,1,2,1,1,2,2,16,1,4,1,2,12,6,1,\\
&\quad 1,2,1,2,60,2,1,10,3,20,15,1,1,42,15,2,1,1,10,1,3,1,38,1,2,4,5,2,5,2,3,2,1,3,1,1,\\
&\quad 9,1,3,38,1,2,1,2,1,2,1,2,1,522,2,1,5,3,54,1,7,1,17,1,1,4,1,5,1,57,1,5,5,1,4,1,7,\\
&\quad 9,6,2,6,5,46,2,2,2,1,3,4,3,1,2,2,1,1,3,1,11,2]\\
\frac{k}{d} &= [0,2,2,6,1,1,1,1,1,1,16,1,404,3,2,4,4,3,1,3,2,2,3,3,1,2,1,1,2,2,16,1,4,1,2,12,6,2]
\end{aligned}
$$

Der Algorithmus hat Erfolg.

(6) Beispiel:

$$
\begin{aligned}
N &= 80914793681663322891827526865102315145471536253480728827613385380 5021\\
p &= 46079316959731559529079752265153921\\
q &= 17559894334452543830040433387857910 1\\
\frac{q}{p} &= 3.810797\\
e &= 62258362135515130034187056184179890461931111138647611747056531014358 01\\
d &= 410183415564524201\\
\frac{\ln(d)}{\ln(N)} &= .251945\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0,1,3,2,1,28,1,5,5,3,6,1,1,5,4,1,1,3,7,2,1,6,4,3,2,3,68,4,1,4,3,4,1,2,2,2,3,\\
&\quad 1,6,5,1,3,1,1,2,14,1,20,7,1,1,3,3,3,3,4,1,1,1,3,3,4,2,1,3,3,2,5,5,1,1,18,2,2,\\
&\quad 10,1,3,1,1,1,3,1,13,1,24,7,21,6,4,1,70,1,3,1,1,6,2,2,2,27,4,5,3,4,8,2,1,4,6,\\
&\quad 22,1,12,5,1,6,1,3,3,4,1,21,2,3,1,1,5,2,2,8,11,2,2,2,2]\\
\frac{k}{d} &= [0,1,3,2,1,28,1,5,5,3,6,1,1,5,4,1,1,3,7,2,1,6,4,3,2,3,68,4,1,4,3,4,1,2,3]
\end{aligned}
$$

Der Algorithmus hat keinen Erfolg.

(7) Beispiel:

$$
\begin{aligned}
N &= 643376869332068374724301342198714852170226180637043801621069981810597 9\\
p &= 56423664070528678643238376707235699\\
q &= 11402607043169999858241001120693372 1\\
\frac{q}{p} &= 2.020891\\
e &= 29285325747747486539283563856413504900582767071790531644165678559200 13\\
d &= 1003480606926816757\\
\frac{\ln(d)}{\ln(N)} &= .257870\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0,2,5,12,1,4,3,3,3,1,13,2,79,1,9,1,2,1,1,2,2,1,8,3,1,122,2,1,2,6,1,2,1,1,3,3,\\
&\quad 1,10,1,1,4,1,8,3,3,1,1,4,4,2,2,1,3,1,2,1,11,6,1,1,32,1,2,4,1,7,1,1,8,1,1,1,41,\\
&\quad 2,6,20,1,1,1,3,5,17,4,13,6,1,3,4,4,1,1,1,2,1,2,3,1,1,1,3,1,1,1,3,28,1,1,1,3,1,\\
&\quad 1,3,7,1,3,1,2,1,22,1,1,2,3,3,3,2,1,2,1,1,1,13,2,3,2,4,1,1,3,4,1,13,1,2,6,1,16,\\
&\quad 12,2]\\
\frac{k}{d} &= [0,2,5,12,1,4,3,3,3,1,13,2,79,1,9,1,2,1,1,2,2,1,8,3,1,122,2,1,2,6,1,2,1,1,3,3]
\end{aligned}
$$

Der Algorithmus hat Erfolg.

(8) Beispiel:

$$
\begin{aligned}
N &= 228256038604938033598820074871439418897116802789978151696884878572 1467\\
p &= 34145317906136610291201845614193909\\
q &= 66848415127485389842170447885842863\\
\frac{q}{p} &= 1.957762\\
e &= 19334744199692132573195893377658964176019232847954031322044411505247 7\\
d &= 400381488092107349\\
\frac{\ln(d)}{\ln(N)} &= .253790\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0,11,1,4,7,10,1,5,2,7,15,2,2,2,4,1,4,1,2,4,1,2,5,14,1,20,1,75,5,5,30,1,2,3,\\
&\quad 23,4,1,1,140,1,4,2,1,9,12,1,1,7,4,1,3,1,4,2,1,3,1,1,2,8,6,1,17,3,13,3,5,2,1,\\
&\quad 3,3,2,1,2,145,1,1,2,3,5,2,1,1,11,1,1,6,1,5,48,1,1,7,1,1,1,1,1,1,2,2,4,2,21,\\
&\quad 2,2,3,15,3,1,1,62,1,1,5,3,1,1,1,3,2,2,2,13,1,4,5,3,3,1,2]\\
\frac{k}{d} &= [0,11,1,4,7,10,1,5,2,7,15,2,2,2,4,1,4,1,2,4,1,2,5,14,1,20,1,75,5,5]
\end{aligned}
$$

Der Algorithmus hat Erfolg.

(9) Beispiel:

$$\begin{aligned}
N &= 44153262380073646954383015297853068889355406107977111244330514010805 63\\
p &= 46937605553258338686840832705710967\\
q &= 94067990600787247070603426159263589\\
\frac{q}{p} &= 2.004107\\
e &= 93987850860134983992362099290873380769008696834223250341768016254 55\\
d &= 7189884310724546759\\
\frac{\ln(d)}{\ln(N)} &= .270755\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0, 469, 1, 3, 2, 7, 1, 3, 3, 1, 3, 1, 608, 58, 1, 60, 6, 36, 1, 1, 11, 2, 1, 3, 1, 2, 2, 2, 4, 1, 1, 6,\\
&\quad 1, 1, 1, 1, 1, 1, 95, 1, 7, 3, 2, 6, 2, 4, 2, 1, 8, 1, 1, 90, 1, 1, 1, 5, 1, 24, 4, 2, 13, 1, 2, 1, 5, 6, 6,\\
&\quad 1, 3, 21, 27, 1, 9, 1, 9, 216, 1, 1, 8, 6, 5, 1, 6, 1, 1, 3, 1, 2, 6, 2, 1, 1, 11, 4, 2, 1, 23, 1, 1, 1, 1,\\
&\quad 6, 1, 124, 7, 3, 6, 57, 4, 13, 2, 1, 13, 1, 74]\\
\frac{k}{d} &= [0, 469, 1, 3, 2, 7, 1, 3, 3, 1, 3, 1, 608, 58, 1, 60, 6, 36, 1, 1, 11, 2, 1, 3, 1, 2, 2, 2]
\end{aligned}$$

Der Algorithmus hat Erfolg. (Dies ist ein interessantes Beispiel.)

(10) Beispiel:

$$\begin{aligned}
N &= 59814262581111495012520362949654103505409384361061312972576470421 89269\\
p &= 52855629701729781274727755693095853\\
q &= 11316535801134156545705016789726 4073\\
\frac{q}{p} &= 2.141028\\
e &= 86786843645279280511129693026396951865865848025826625695467692956 7791\\
d &= 775134741803520975\\
\frac{\ln(d)}{\ln(N)} &= .256380\\
\frac{e}{N-\lfloor\sqrt{4N}\rfloor} &= [0, 6, 1, 8, 3, 1, 2, 1, 20, 1, 26, 1, 2, 1, 37, 1, 1, 4, 2, 1, 4, 1, 1, 1, 3, 5, 1, 1, 134, 1, 3, 4, 6, 1, 25,\\
&\quad 1, 5, 11, 1, 1, 13, 41, 1, 3, 1, 3, 2, 3, 1, 9, 1, 1, 10, 1, 2, 1, 4, 3, 2, 5, 3, 1, 1, 2, 10, 1, 1, 1, 1, 1, 1,\\
&\quad 35, 2, 1, 3, 1, 1, 6, 1, 2, 1, 3, 1, 151, 5, 1, 1, 2, 1, 2, 13, 1, 1, 96, 2, 1, 1, 7, 1, 1, 3, 3, 7, 9, 2, 4, 1,\\
&\quad 1, 5, 2, 1, 1, 1, 9, 9, 14, 5, 5, 3, 1, 3, 3, 4, 517, 1, 1, 1, 2, 22, 3, 3, 1, 2, 1, 3, 1, 4, 4]\\
\frac{k}{d} &= [0, 6, 1, 8, 3, 1, 2, 1, 20, 1, 26, 1, 2, 1, 37, 1, 1, 4, 2, 1, 4, 1, 1, 1, 3, 5, 1, 1, 134, 1, 3, 4, 6, 1, 26]
\end{aligned}$$

Der Algorithmus hat Erfolg.

Eine Verallgemeinerung des obigen Satzes gibt folgender Satz:

SATZ. *Die Bezeichungen seien wie im letzten Satz. Sei* $\delta = |p-q| = N^\beta$, $d = N^\delta$. *Wir setzen voraus* $p, q \geq 23$ *und*

$$\beta + \delta \leq \frac{3}{4}.$$

Dann gilt:

$$\left|\frac{e}{N+1-2\sqrt{N}} - \frac{k}{d}\right| < \frac{1}{2d^2}.$$

Also kommt $\frac{k}{d}$ *als Näherungsbruch in der Kettenbruchentwicklung von* $\frac{e}{N+1-2\sqrt{N}}$ *vor.*

Betrachtet man aber Beispiele mit $q \gtrapprox 2p$, so ist $\beta \gtrapprox 0.5$, damit die Voraussetzung des letzten Satzes erfüllt ist, muß also $\delta \lessapprox 0.25$ gelten, der letzte Satz liefert dann in diesem Fall nichts Neues.

10. Anhang

Im folgenden werden die verwendeten Programme kbfac_gmp.c, squfof_gmp.c und cfrac_gmp.c wiedergegeben.

Das Programm kbfac_gmp.c:

```
/* kbfac_gmp.c - Faktorisierung mit Kettenbruechen
   Version vom 26.6.2001, 27.6.2001, 28.6.2001, 29.6.2001, 14.7.2001
*/

#include <stdio.h>
#include <gmp.h>
#include <time.h>

#define T 100000 /* Kleine Teiler bis T werden zuerst herausdividiert */

/* Test, ob n prim ist */
int prim( int n)
{
  int t;
  if (n<2 || (n%2==0 && n!=2)) return 0;
  for (t=3;t*t<=n;t=t+2)
    if (n%t==0) return 0;
  return 1;
}

main()
{
  int az, t, i, n;
  mpz_t N, hi1, hi2, d, wd, p1, hi, p0, b, c, a, wc;
  time_t zeit1, zeit2;

  mpz_init(N); mpz_init(hi1); mpz_init(hi2); mpz_init(d);
  mpz_init(wd); mpz_init(p1); mpz_init(hi); mpz_init(p0); mpz_init(b);
  mpz_init(c); mpz_init(a); mpz_init(wc);

  printf("Faktorisierung von ");
  mpz_inp_str(N,stdin,10);
  printf("N="); mpz_out_str(stdout,10,N); printf("\n");

  time(&zeit1);

  /*****************************************************************
  Kleine Teiler von N werden herausdividiert.
  *****************************************************************/
  printf("Primteiler < %d (kleine Teiler):\n",T);
  az=0;
  for (t=2;t<T;t++)
  {
    while ((t%2==0 || t%3==0 || t%5==0) && t!=2 && t!=3 && t!=5) t++;
    if (mpz_fdiv_qr_ui(hi1,hi2,N,t)==0)
    {
      az++;
      mpz_set(N,hi1); i=1; /* Exponent */
      while (mpz_fdiv_qr_ui(hi1,hi2,N,t)==0)
```

```
    {
      mpz_set(N,hi1); i++;
    }
    if (az>1) printf(" * ");
    if (i==1) printf("%d",t); else printf("%d^%d ",t,i);
  }
}
printf("\n");
printf("Rest: "); mpz_out_str(stdout,10,N); printf(" ");
if (mpz_cmp_ui(N,1)==0) { printf("\n"); return; }
if (mpz_probab_prime_p(N,10)>0)
{ printf("(wahrscheinlich) prim\n"); return; }
printf("zusammengesetzt\n");

/*****************************************************************
Faktorisierung mit Kettenbruechen
*****************************************************************/

mpz_set(d,N);
printf("Kettenbruchentwicklung von "); mpz_out_str(stdout,10,d);
printf("^1/2");

/* Initialisierung der Kettenbruchentwicklung */
mpz_sqrt(wd,d); /* wd=floor(d^1/2) */
mpz_set_ui(p1,1); /* p_(-1)=1 */
mpz_set_ui(p0,0); /* p_(-2)=0 */
mpz_set_ui(b,0); mpz_set_ui(c,1); /* b_0=0, c_0=1 */

for (n=0;;n++)
{
  /* a_n=floor((b_n+sqrt(d))/c_n) */
  mpz_add(hi,b,wd); mpz_fdiv_q(a,hi,c);

  /* p_n=a_n*p_(n-1)+p_(n-2) */
  mpz_mul(hi,a,p1); mpz_add(hi,hi,p0); mpz_set(p0,p1); mpz_set(p1,hi);

  /* b_(n+1)=a_n*c_n-b_n */
  mpz_mul(hi,a,c); mpz_sub(b,hi,b);

  /* c_(n+1)=(d-b_(n+1)^2)/c_n */
  mpz_mul(hi,b,b); mpz_sub(hi,d,hi); mpz_divexact(c,hi,c);

  mpz_mod(p1,p1,N);

  /* Es gilt p_n^2-d*q_n^2=(-1)^(n+1)*c_(n+1).
     Nun wird getestet, ob c_(n+1) ein Quadrat ist.
  */
  if (n%2==1)
  {
    mpz_sqrt(wc,c); mpz_mul(hi,wc,wc);
    if (mpz_cmp(c,hi)==0) /* c=wc^2? */
    {
      printf("\np_%d=",n); mpz_out_str(stdout,10,p1);
      printf(" mod N\n");
```

```
        printf("sqrt(c_%d)=",n+1); mpz_out_str(stdout,10,wc);

        mpz_add(hi1,p1,wc);
        mpz_sub(hi2,p1,wc);

        if (mpz_cmp(p1,wc)==0)
        {
          printf("=p_%d mod N\n",n);
        }
        else if (mpz_cmp(hi1,N)==0)
        {
          printf("=-p_%d mod N\n",n);
        }
        else
        {
          printf("\n");
          mpz_gcd(hi1,hi1,N); mpz_gcd(hi2,hi2,N);

          printf("ggT(p_%d+sqrt(c_%d),N)=",n,n+1);
           mpz_out_str(stdout,10,hi1);
          printf("\nggT(p_%d-sqrt(c_%d),N)=",n,n+1);
           mpz_out_str(stdout,10,hi2); printf("\n");

          time(&zeit2);

          printf("Zeit: %.0f sec\n",difftime(zeit2,zeit1));

          mpz_mul(hi,hi1,hi2);
          if (mpz_cmp(hi,N)!=0)
          {
            printf("Dies ist keine Faktorisierung von N.\n");
            return;
          }
          printf("Es gilt:\nN="); mpz_out_str(stdout,10,hi1);
          printf(" * "); mpz_out_str(stdout,10,hi2); printf("\n");
          (mpz_probab_prime_p(hi1,10)>0) ?
            printf("Der erste Faktor ist prim.\n") :
            printf("Der erste Faktor ist zusammengesetzt.\n");
          // if (mpz_probab_prime_p(hi1,10)>0)
          // {
            // printf("Der erste Faktor ist prim.\n");
          // }
          if (mpz_probab_prime_p(hi2,10)>0)
          {
            printf("Der zweite Faktor ist prim.\n");
          }

          return;
        }
      }
    }
  }
}
```

Das Programm squfof_gmp.c:

```
/* squfof.c - square form factorization nach Shanks bzw. Riesel
   Version vom 26.6.2001, 27.6.2001, 28.6.2001, 29.6.2001, 6.7.2001,
   14.7.2001
   Verfahren funktioniert nicht fuer N=3064333 (Fehler?)
*/

#include <stdio.h>
#include <gmp.h>
#include <time.h>

#define T 100000 /* Kleine Teiler bis T werden zuerst herausdividiert */
#define M 1000 /* Maximale Anzahl moeglicher Listenelemente */
#define INFO 0 /* Infolevel */

main( int argc, char *argv[])
{
  int az, t, i, n, v;
  mpz_t N, hi1, hi2, d, ww4d, wd, b, c, hi, a, C[M], wc, bb;
  time_t zeit1, zeit2;

  mpz_init(N); mpz_init(hi1); mpz_init(hi2); mpz_init(d);
  mpz_init(ww4d); mpz_init(wd); mpz_init(b); mpz_init(c); mpz_init(hi);
  mpz_init(a); mpz_init(wc); mpz_init(bb);

  if (argc==1)
  {
    printf("Faktorisierung von ");
    mpz_inp_str(N,stdin,10);
  }
  else
    mpz_set_str(N,argv[1],10);
  printf("N="); mpz_out_str(stdout,10,N); printf("\n");

  time(&zeit1);

  /*****************************************************************
  Kleine Teiler von N werden herausdividiert.
  *****************************************************************/
  printf("Kleine Primteiler: ");
  az=0;
  for (t=2;t<T;t++)
  {
    while ((t%2==0 || t%3==0 || t%5==0) && t!=2 && t!=3 && t!=5) t++;
    if (mpz_fdiv_qr_ui(hi1,hi2,N,t)==0)
    {
      az++;
      mpz_set(N,hi1); i=1; /* Exponent */
      while (mpz_fdiv_qr_ui(hi1,hi2,N,t)==0)
      {
        mpz_set(N,hi1); i++;
      }
      if (az>1) printf(" * ");
      if (i==1) printf("%d",t); else printf("%d^%d",t,i);
```

```
    }
  }
  printf("\n");
  printf("Rest: d="); mpz_out_str(stdout,10,N); printf(" ");
  if (mpz_cmp_ui(N,1)==0) { printf("\n"); return; }
  if (mpz_probab_prime_p(N,10)>0)
  { printf("(wahrscheinlich) prim\n"); return; }
  printf("zusammengesetzt\n");

  /*****************************************************************
  Faktorisierung mit Kettenbruechen
  *****************************************************************/

  printf("SQUFOF-Verfahren mit Kettenbruchentwicklung von sqrt(");
  mpz_set(d,N);
  mpz_out_str(stdout,10,d); printf(")\n");

  mpz_mul_ui(ww4d,d,4); mpz_sqrt(ww4d,ww4d); mpz_sqrt(ww4d,ww4d);

  /* Initialisierung der Kettenbruchentwicklung */
  mpz_sqrt(wd,d); /* wd=floor(d^1/2) */
  mpz_set_ui(b,0); mpz_set_ui(c,1); /* b_0=0, c_0=1 */

  for (n=0,az=0;;n++)
  {
    /* a_n=floor((b_n+sqrt(d))/c_n) */
    mpz_add(hi,b,wd); mpz_fdiv_q(a,hi,c);

    /* b_(n+1)=a_n*c_n-b_n */
    mpz_mul(hi,a,c); mpz_sub(b,hi,b);

    /* c_(n+1)=(d-b_(n+1)^2)/c_n */
    mpz_mul(hi,b,b); mpz_sub(hi,d,hi); mpz_divexact(c,hi,c);

    /* Falls c_(n+1)<=w4d bzw. c_(n+1)/2<=w4d ist,
       wird es in die Liste aufgenommen */
    if (mpz_fdiv_q_ui(hi2,c,2)!=0) mpz_set(hi2,c);
    if (mpz_cmp(hi2,ww4d)<=0 && mpz_cmp_ui(hi2,1)>0)
    {
      mpz_init_set(C[az++],hi2);
      printf("Aufgenommen in Liste: c_%d bzw. c_%d/2: ",n+1,n+1);
      mpz_out_str(stdout,10,hi2); printf("\n");
    }

    if (INFO==1)
    {
      printf("b_%d=",n+1); mpz_out_str(stdout,10,b);
      printf(" c_%d=",n+1); mpz_out_str(stdout,10,c);
      printf(" L={");
      for (i=0;i<az;i++)
      {
        mpz_out_str(stdout,10,C[i]); printf(" ");
      }
      printf("}\n");
```

```
}

/* Nun wird getestet, ob c_(n+1) ein Quadrat ist.  */
if (n%2==1)
{
  mpz_sqrt(wc,c); mpz_mul(hi,wc,wc);
  if (mpz_cmp(c,hi)==0) /* c=wc^2? */
  {
    printf("sqrt(c_%d)=",n+1); mpz_out_str(stdout,10,wc);
    printf("\n");

    if (mpz_cmp_ui(wc,1)==0)
    {
      printf("Das Verfahren funktioniert hier nicht!\n"); return;
    }

    printf("Liste: ");
    for (i=0;i<az;i++) { mpz_out_str(stdout,10,C[i]); printf(" "); }
    printf("\n");

    /* Ist wc in der Liste C enthalten? */
    v=1;
    for (i=0;i<az;i++)
    {
      if (mpz_cmp(C[i],wc)==0) { v=0; break; }
    }

    if (v==1) /* wc ist nicht in der Liste enthalten! */
    {
      /* Kettenbruchentwicklung von (-b_(n+1)+sqrt(d))/wc */

      mpz_neg(b,b); mpz_set(c,wc);

      printf("Da sqrt(c_%d) nicht in der Liste enthalten ",n+1);
      printf("ist, wird die \nKettenbruchentwicklung von \n");
      printf("(-b_%d+sqrt(",n+1); mpz_out_str(stdout,10,d);
      printf(")/sqrt(c_%d)=\n=(",n+1);
      mpz_out_str(stdout,10,b); printf("+sqrt(");
      mpz_out_str(stdout,10,d); printf("))/");
      mpz_out_str(stdout,10,c); printf(" \nbetrachtet.\n");

      for (n=0;;n++)
      {
        /* a_n=floor((b_n+sqrt(d))/c_n) */
        mpz_add(hi,b,wd); mpz_fdiv_q(a,hi,c);

        /* b_(n+1)=a_n*c_n-b_n */
        mpz_mul(hi,a,c); mpz_sub(bb,hi,b);

        /* Ist b_(n+1)=b_n? */
        if (mpz_cmp(bb,b)==0)
        {
          printf("b'_%d=b'_%d=",n,n+1); mpz_out_str(stdout,10,b);
          printf("\n");
```

```
              mpz_gcd(hi1,N,c);
              printf("ggT(N,c'_%d)=",n); mpz_out_str(stdout,10,hi1);
              printf("\n");
              mpz_divexact(hi2,N,hi1);
              printf("Die restlichen Faktoren von N:\n");
              mpz_out_str(stdout,10,hi1);
              if (mpz_probab_prime_p(hi1,10)>0) printf(" (prim)\n");
              else printf(" (zusammengesetzt)\n");
              mpz_out_str(stdout,10,hi2);
              if (mpz_probab_prime_p(hi2,10)>0) printf(" (prim)\n");
              else printf(" (zusammengesetzt)\n");
              time(&zeit2);
              printf("Zeit: %.0f sec\n",difftime(zeit2,zeit1));
              return;
            }
            mpz_set(b,bb);

            /* c_(n+1)=(d-b_(n+1)^2)/c_n */
            mpz_mul(hi,b,b); mpz_sub(hi,d,hi); mpz_divexact(c,hi,c);

            if (INFO==1)
            {
              printf("b_%d=",n+1); mpz_out_str(stdout,10,bb);
              printf(" c_%d=",n+1); mpz_out_str(stdout,10,c);
              printf(" a_%d=",n); mpz_out_str(stdout,10,a);
              printf("\n");
            }
          }
        }
      }
    }
  }
}
```

Das Programm cfrac_gmp.c:

```
/* cfrac_gmp.c
   Version vom 26.6.2001, 27.6.2001, 28.6.2001
   M=1101 funktioniert auf PC unter Linux nicht mehr
   INFO 0, 1, 2, 3, 4
*/

#include <stdio.h>
#include <gmp.h>
#include <time.h>

#define T 10 /* Kleine Teiler bis T werden zuerst herausdividiert */
#define M 1001 /* maximale Anzahl der Primzahlen der Faktorbasis etc. */
#define INFO 1 /* Infolevel */

/* Test, ob n prim ist */
int prim( int n)
{
  int t;
  if (n<2 || (n%2==0 && n!=2)) return 0;
```

```
  for (t=3;t*t<=n;t=t+2)
    if (n%t==0) return 0;
  return 1;
}

/* kbewdp: Kettenbruchentwicklung von Wurzel(d):
 * Eingabe:   d, b_n, c_n, floor(d^1/2),
 *            *, p_(n-1) mod d, p_(n-2) mod d
 * Rueckgabe: d, b_(n+1), c_(n+1), floor(d^1/2),
 *            a_n, p_n mod d, p_(n-1) mod d
 */
void kbewdp( mpz_t d, mpz_t b, mpz_t c, mpz_t wd,
             mpz_t a, mpz_t p1, mpz_t p0)
{
  mpz_t hi; mpz_init(hi);

  /* a_n=floor((b_n+sqrt(d))/c_n) */
  mpz_add(hi,b,wd); mpz_fdiv_q(a,hi,c);

  /* p_n=a_n*p_(n-1)+p_(n-2), q_n=a_n*q_(n-1)+q_(n-2) */
  mpz_mul(hi,a,p1); mpz_add(hi,hi,p0);
  mpz_set(p0,p1); mpz_mod(p1,hi,d);

  /* b_(n+1)=a_n*c_n-b_n */
  mpz_mul(hi,a,c); mpz_sub(b,hi,b);

  /* c_(n+1)=(d-b_(n+1)^2)/c_n */
  mpz_mul(hi,b,b); mpz_sub(hi,d,hi); mpz_divexact(c,hi,c);

  mpz_clear(hi);
  return;
}

/* p[1],...p[n-1] werden aus c herausdividiert. Der Rueckgabewert ist
   1, falls der Rest 1 ist. */
int kTtest( mpz_t c, int *p, int *e, int n)
{
  int i, w;
  mpz_t hi1, hi2, hi3;
  mpz_init_set(hi1,c); mpz_init(hi2); mpz_init(hi3);
  for (i=1;i<n;i++)
  {
    e[i]=0;
    while (mpz_fdiv_qr_ui(hi2,hi3,hi1,p[i])==0)
    {
      mpz_set(hi1,hi2);
      e[i]^=1;
    }
  }
  w=mpz_cmp_ui(hi1,1);
  mpz_clear(hi1); mpz_clear(hi2); mpz_clear(hi3);
  if (w==0) return 1; else return 0;
}
```

```
void kTc( mpz_t c, int *p, int *e, int n)
{
  int i, w=0;
  mpz_t hi1, hi2, hi3;
  mpz_init_set(hi1,c); mpz_init(hi2); mpz_init(hi3);
  for (i=1;i<n;i++)
  {
    e[i]=0;
    while (mpz_fdiv_qr_ui(hi2,hi3,hi1,p[i])==0)
    {
      mpz_set(hi1,hi2);
      e[i]++;
    }
    if (e[i]>0)
    {
      w++;
      if (w>1) printf("*");
      printf("%d",p[i]);
      if (e[i]>1) printf("^%d",e[i]);
    }
  }
  if (mpz_cmp_ui(hi1,1)>0)
  {
    if (w>0) printf("*"); mpz_out_str(stdout,10,hi1);
  }
  printf("\n");
  mpz_clear(hi1); mpz_clear(hi2); mpz_clear(hi3);
  return;
}

/* Ausgabe der Matrizen A und AA */
void pmat( int A[][M], int AA[][M], int m, int n, int l)
{
  int i, j;
  for (i=0;i<=m;i++)
  {
    for (j=0;j<n;j++) printf("%d ",A[i][j]);
    printf("\t");
    for (j=0;j<=l;j++) printf("%d ",AA[i][j]);
    printf("\n");
  }
  return;
}

int nofo( int A[][M], int AA[][M], int m, int n, int l)
{
  int i=0, ij=0, j, v;
  if (INFO>1) { printf("Matrix A und Matrix AA:\n"); pmat(A,AA,m,n,l); }
  while (i<m && ij<n)
  {
    if (A[i][ij]==1 && A[m][ij]==1) /* Addiere i-te zur m-ten Zeile */
    {
      if (INFO>2) printf("Addiere Zeile %d zu Zeile %d:\n",i,m);
      for (j=ij;j<n;j++) A[m][j]^=A[i][j];
```

```
      for (j=0;j<=l;j++) AA[m][j]^=AA[i][j];
      i++; ij++;
      if (INFO>2) pmat(A,AA,m,n,l);
    }
    else if (A[i][ij]==1 && A[m][ij]==0) /* Indizes i und ij erhoehen */
    {
      i++; ij++;
    }
    else if (A[i][ij]==0 && A[m][ij]==0) /* Index ij erhoehen */
    {
      ij++;
    }
    else /* Fall A[i][ij]=0, A[m][ij]==1: Vertausche Zeilen i und m */
    {
      if (INFO>2) printf("Vertausche Zeile %d und Zeile %d\n",i,m);
      for (j=ij;j<n;j++) { v=A[m][j]; A[m][j]=A[i][j]; A[i][j]=v; }
      for (j=0;j<=l;j++) { v=AA[m][j]; AA[m][j]=AA[i][j]; AA[i][j]=v; }
      i++; ij++;
      if (INFO>2) pmat(A,AA,m,n,l);
    }
  }
  if (INFO>1) printf("Zeilenstufentransformation beendet.\n");
  if (INFO==2) { printf("Matrix A und Matrix AA:\n"); pmat(A,AA,m,n,l);}
  for (j=0;j<n;j++)
    if (A[m][j]==1) return 0;
  return 1;
}

main()
{
  int az, t, i, n, *p, *e, j, m, A[M][M], AA[M][M], sgn, l;
  mpz_t N, hi1, hi2, hi, d, wd, a, b, c, p1, p0, P[M], C[M], x, y;
  time_t zeit0, zeit1;

  mpz_init(N); mpz_init(hi1); mpz_init(hi2); mpz_init(hi);
  mpz_init(d); mpz_init(wd); mpz_init(a); mpz_init(b); mpz_init(c);
  mpz_init(p1); mpz_init(p0); mpz_init(x); mpz_init(y);

  printf("Faktorisierung von ");
  mpz_inp_str(N,stdin,10);
  printf("N="); mpz_out_str(stdout,10,N); printf("\n");

  /*****************************************************************
  Kleine Teiler von N werden herausdividiert.
  *****************************************************************/
  printf("Primteiler < %d (kleine Teiler): ",T);
  az=0;
  for (t=2;t<T;t++)
  {
    while ((t%2==0 || t%3==0 || t%5==0) && t!=2 && t!=3 && t!=5) t++;
    if (mpz_fdiv_qr_ui(hi1,hi2,N,t)==0)
    {
      az++;
      mpz_set(N,hi1); i=1; /* Exponent */
```

```
      while (mpz_fdiv_qr_ui(hi1,hi2,N,t)==0)
      {
        mpz_set(N,hi1); i++;
      }
      if (az>1) printf(" * ");
      if (i==1) printf("%d",t); else printf("%d^%d ",t,i);
    }
  }
  printf("\n");
  printf("Rest: N1="); mpz_out_str(stdout,10,N); printf(" ");
  if (mpz_cmp_ui(N,1)==0) { printf("\n"); return; }
  if (mpz_probab_prime_p(N,10)>0)
  { printf("(wahrscheinlich) prim\n"); return; }
  printf("zusammengesetzt\n");

  /*****************************************************************
  Faktorisierung mit Kettenbruechen
  *****************************************************************/

  /* Faktorbasis */
  printf("Wieviele Elemente soll die Faktorbasis enthalten? ");
  scanf("%d",&n); printf("\n");
  time(&zeit0);
  if (n>=M) { printf("%d zu gross!\n",n); return; }
  p=(int *) malloc(n*sizeof(int));
  p[0]=-1; p[1]=2;
  for (t=3,i=2;i<n;t=t+2)
  {
    if (prim(t)==1)
    {
      mpz_set_ui(hi,t);
      if (mpz_legendre(N,hi)==1) p[i++]=t;
    }
  }
  /* Faktorbasis ist p[0],...,p[n-1] */
  if (INFO>0) printf("Die Faktorbasis hat %d Elemente.\n",n);
  if (INFO>1)
  {
    printf("Faktorbasis: ");
    for (i=0;i<n;i++) printf("%d ",p[i]); printf("\n");
  }
  e=(int *) malloc(n*sizeof(int));

  /* Begleitmatrix */
  for (i=0;i<2*n;i++) for (j=0;j<2*n;j++) AA[i][j]=0;

  mpz_set(d,N);
  printf("Kettenbruchentwicklung von "); mpz_out_str(stdout,10,d);
  printf("^1/2");
  if (INFO>0) printf("\n");

  /* Initialisierung der Kettenbruchentwicklung */
  mpz_sqrt(wd,d); /* wd=floor(d^1/2) */
  mpz_set_ui(p1,1); /* p_(-1)=1 */
```

```
mpz_set_ui(p0,0); /* p_(-2)=0 */
mpz_set_ui(b,0); mpz_set_ui(c,1); /* b_0=0, c_0=1 */

for (l=0,m=l,sgn=0,az=0;;)
{
  sgn^=1;
  kbewdp(d,b,c,wd,a,p1,p0);
  if (INFO>3)
  {
    printf("\n");
    printf("a_%d=",az); mpz_out_str(stdout,10,a); printf("  ");
    printf("p_%d=",az); mpz_out_str(stdout,10,p1); printf("  ");
    printf("b_%d=",az+1); mpz_out_str(stdout,10,b); printf("  ");
    printf("c_%d=",az+1); mpz_out_str(stdout,10,c); printf("=");
    kTc(c,p,e,n);
  }

  /* Es gilt p_n^2-d*q_n^2=(-1)^(n+1)*c_(n+1).
     Nun wird getestet, ob c_(n+1) sich als Produkt der
     Basisprimzahlen schreiben laesst.
  */
  if (kTtest(c,p,e,n)==1)
  {
    /* P[l]^2 = (-1)^sgn*C[l] mod N */
    mpz_init_set(P[l],p1); mpz_init_set(C[l],c);
    if (INFO==1) { printf("."); fflush(stdout); }
    if (INFO>1)
    {
      printf("\np_%d^2-d*q_%d^2=(-1)^%d*c_%d ",az,az,az+1,az+1);
      printf("liefert %d. Relation P_%d^2=",l+1,l);
      if (az%2==0) printf("-");
      printf("C_%d mod N mit \n",l);
      printf("P_%d=p_%d=",l,az); mpz_out_str(stdout,10,P[l]);
      if (az%2==0) printf("\n-C_%d=-c%d=-",l,az+1);
      else printf("\nC_%d=c_%d=",l,az+1);
      mpz_out_str(stdout,10,C[l]);
      printf("=(-1)^%d*",(az+1)%2);
      for (i=1;i<n;i++)
      {
        printf("%d^%d*",p[i],e[i]);
      }
      printf("Quadrat\n");
    }
    A[m][0]=sgn; /* Vorzeichen */
    for (j=1;j<n;j++) A[m][j]=e[j];
    AA[m][l]=1;

    if (nofo(A,AA,m,n,l)==1) /* Letzte Zeile von A gleich 0? */
    {
      mpz_set_ui(x,1); mpz_set_ui(y,1);
      for (j=0;j<=l;j++)
      if (AA[m][j]==1)
      {
        mpz_mul(x,x,P[j]); mpz_mul(y,y,C[j]);
```

```
      }
      mpz_sqrt(y,y);

      if (INFO==1) { printf("*"); fflush(stdout); }
      if (INFO>1)
      {
        printf("Die letzte Zeile von A ist 0. Daher ist\n");
        for (j=0;j<=l;j++) if (AA[m][j]==1) printf("C_%d",j);
        printf("=y^2 mit y="); mpz_out_str(stdout,10,y);
        printf(".\nMit x=");
        for (j=0;j<=l;j++) if (AA[m][j]==1) printf("P_%d",j);
        printf("="); mpz_out_str(stdout,10,x);
        printf("\ngilt x^2=y^2 mod N\n");
      }

      mpz_add(hi1,x,y); mpz_gcd(hi1,hi1,N); /* ggT(x+y,N) */
      mpz_sub(hi2,x,y); mpz_gcd(hi2,hi2,N); /* ggT(x-y,N) */
      if (INFO>1)
      {
        printf("ggT(x+y,N)="); mpz_out_str(stdout,10,hi1);
        printf("\n");
        printf("ggT(x-y,N)="); mpz_out_str(stdout,10,hi2);
        printf("\n");
      }
      if ((mpz_cmp(N,hi1)>0 && mpz_cmp_ui(hi1,1)>0) ||
          (mpz_cmp(N,hi2)>0 && mpz_cmp_ui(hi2,1)>0))
      {
        mpz_mul(hi,hi1,hi2);
        if (mpz_cmp(hi,N)==0)
          printf("\nNichttriviale Faktorisierung N1=N2*N3 mit \n");
        else
          printf("\nNichttriviale Faktoren von N1\n");
        printf("N2="); mpz_out_str(stdout,10,hi1);
        if (mpz_probab_prime_p(hi1,10)>0) printf(" (prim)");
        else printf(" (zusammengesetzt)");
        printf("\nN3="); mpz_out_str(stdout,10,hi2);
        if (mpz_probab_prime_p(hi2,10)>0) printf(" (prim)");
        else printf(" (zusammengesetzt)");
        printf("\n");
        time(&zeit1);
        printf("Zeit: %.0f sec\n",difftime(zeit1,zeit0));
        return;
      }
      for (j=0;j<=l;j++) AA[m][j]=0;
      m--;
      if (INFO>1)
      {
        printf("Triviale Faktorisierung. ");
        printf("Daher wird die letzte Zeile von A entfernt:\n");
        pmat(A,AA,m,n,l);
      }
    }
    l++; m++;
  }
```

```
    az++;
  }
}
```

Literatur

- G. H. Hardy, E. M. Wright, An Introduction to the Theory of Numbers, Fifth Edition, Clarendon Press, Oxford 1979.
- L. K. Hua, Introduction to Number Theory, Springer-Verlag 1982.
- O. Forster, Algorithmische Zahlentheorie, Vieweg 1996.
- H. Riesel, Prime Numbers and Computer Methods for Factorization, Birkhäuser 1987.
- M. J. Wiener, Cryptanalysis of short RSA secret exponents, IEEE Transactions on Information Theory **36** (1990), 553-558.

KAPITEL 4

Enigma

1. Einführung I

Mit dem Aufkommen elektrischer Schreibmaschinen lag es nahe, auch elektrische Chiffriermaschinen zu bauen. Um 1920 gab es mehrere Ansätze für sogenannte Rotormaschinen, die zur Verschlüsselung Rotoren/Walzen verwenden. In Deutschland baute Scherbius für kommerzielle Zwecke die Chiffriermaschine *Enigma* in verschiedenen Varianten. Die Enigma wurde auch in Heer, Marine und Luftwaffe eingeführt und spielte vor und während des 2. Weltkriegs bei den Deutschen eine große Rolle.

Es gibt viele Informationen über die Enigma, auch eine Reihe von Enigma-Simulatoren im Internet. Allerdings sind die Aussagen nicht alle konsistent.
Hier soll im folgenden eine Variante der Enigma mathematisch beschrieben werden, wie sie (in der beschriebenen Form hoffentlich) um 1938 von der Deutschen Wehrmacht benutzt wurde. Die Darstellung soll es ermöglichen, einen Enigma-Simulator zu programmieren.
Eine Darstellung der erfolgreichen polnischen und englischen Angriffe auf die Enigma-Verschlüsselung fehlt (noch).

2. Einführung II

Die Enigma verschlüsselt die 26 Buchstaben A, B, . . . , Z. Die Eingabe erfolgt durch Drücken von Tasten, die Ausgabe durch Aufleuchten elektrischer Lämpchen. Als Alphabet legen wir also

$$\mathfrak{A} = \{\mathrm{A},\mathrm{B},\mathrm{C},\ldots,\mathrm{Z}\}$$

Version vom 24.7.2001 (Anhang 25.7.2001)

zugrunde. Wir identifizieren A, B, C, ..., Z mit 0, 1, 2, ..., 25, haben somit eine Bijektion $\mathfrak{A} \simeq \{0,1,2,\ldots,25\} \simeq \mathbf{Z}/(26)$. (Entsprechende Rechnungen verstehen sich alle modulo 26.)
Für bijektive Abbildungen $R : \mathfrak{A} \to \mathfrak{A}$ verwenden wir die übliche Permutationsdarstellung. So bezeichnet z.B.

$$R = (\mathrm{AELTPHQXRU})(\mathrm{BKNW})(\mathrm{CMOY})(\mathrm{DFG})(\mathrm{IV})(\mathrm{JZ})(\mathrm{S})$$

die Abbildung mit

$$R(\mathrm{A}) = \mathrm{E}, \quad R(\mathrm{E}) = \mathrm{L}, \quad \ldots, R(\mathrm{U}) = \mathrm{A}, \quad R(\mathrm{B}) = \mathrm{K}, \quad \ldots, R(\mathrm{S}) = \mathrm{S}.$$

Auf (S) hätte man in der Darstellung auch verzichten können.

Der Chiffrierschlüssel der Enigma besteht aus bestimmten mechanischen Einstellungen:

(1) Aus 5 Rotoren/Walzen werden 3 ausgewählt und in die Maschine eingesetzt. Die entsprechende Anordnung wird *Walzenlage* genannt und hier mit (w_L, w_M, w_R) bezeichnet. Es ist also
$$(w_L, w_M, w_R) \in \{(1,2,3),(1,2,4),\ldots,(5,4,3)\}.$$
(2) Die sogenannte *Ringstellung* (r_L, r_M, r_R) ist eine bestimmte mechanische Einstellung an den ausgewählten Rotoren, wobei jede Wahl mit $0 \le r_L, r_M, r_R \le 25$ möglich ist.
(3) Die sogenannte *Steckerverbindung* S ist eine Involution
$$S : \mathfrak{A} \to \mathfrak{A},$$
d.h. $S \circ S = \mathrm{id}_{\mathfrak{A}}$ oder $S^{-1} = S$.
(4) Die *Grundstellung* (s_L, s_M, s_R) mit $0 \le s_L, s_M, s_R \le 25$ bestimmt eine Anfangseinstellung der ausgewählten Rotoren.

Ein Enigma-Schlüssel hat dann die Gestalt

$$K = (w_L, w_M, w_R, r_L, r_M, r_R, S, s_L, s_M, s_R).$$

Den zugehörigen Schlüsselraum bezeichnen wir mit $\mathfrak{S}$.

Bei Tastendruck wird der aktuelle Schlüssel in einen anderen Schlüssel abgewandelt. Die zugehörige Abbildung bezeichnen wir als Schlüsselnachfolgeabbildung

$$\Sigma : \mathfrak{S} \to \mathfrak{S}.$$

Die Enigma-Chiffrierung benutzt eine mechanisch realisierte Chiffrierfunktion

$$E : \mathfrak{S} \times \mathfrak{A} \to \mathfrak{A}.$$

Die Chiffrierung verläuft dann nach folgendem Schema:

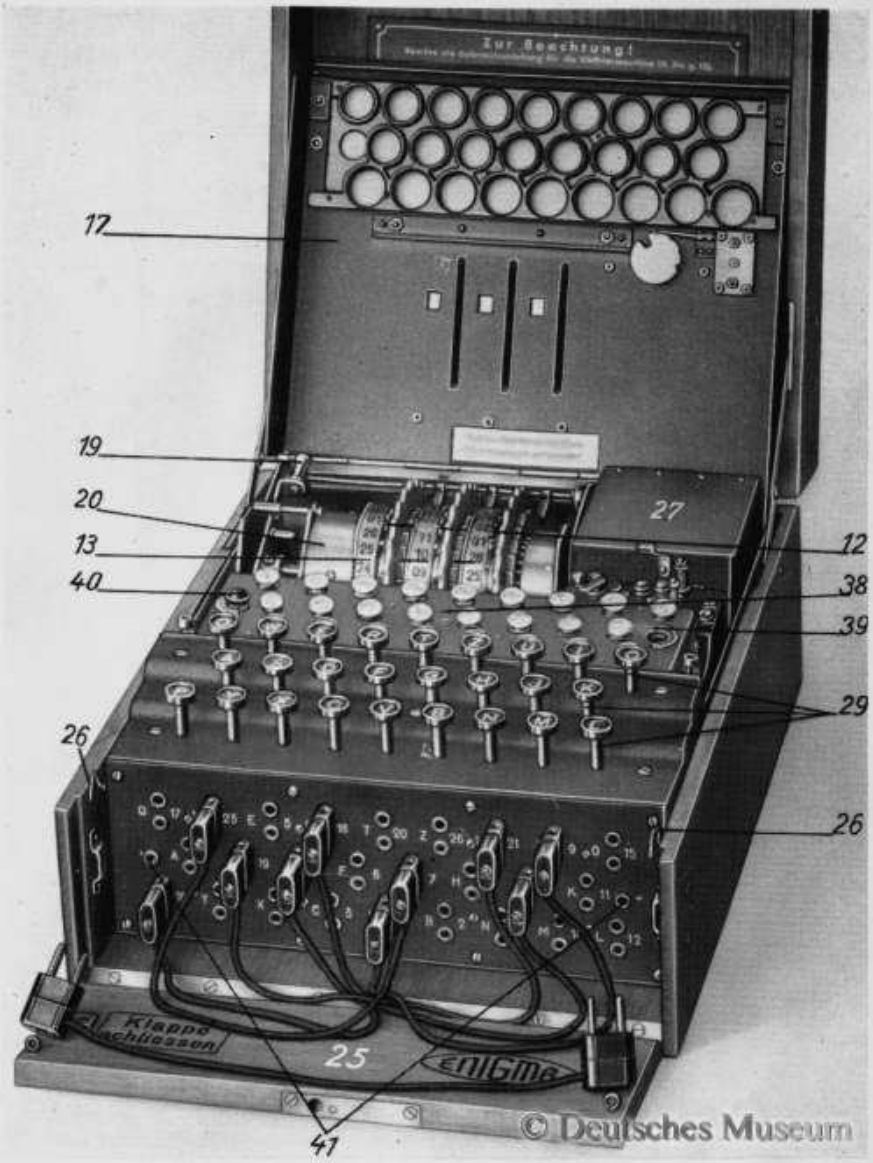
- Eingegeben wird ein Schlüssel $K = (w_L, w_M, w_R, r_L, r_M, r_R, S, s_L, s_M, s_R)$ und eine Buchstabenfolge $x_1, x_2, x_3, \ldots, x_n$ mit $x_i \in \mathfrak{A}$.
- Aus dem Eingangsschlüssel $K = K_0$ wird eine Schlüsselfolge $K_1, K_2, \ldots, K_n$ mit $K_i \in \mathfrak{S}$ wie folgt konstruiert:
$$K_1 = \Sigma(K), \quad K_2 = \Sigma(K_1), \quad K_3 = \Sigma(K_2), \quad \ldots, K_n = \Sigma(K_{n-1}).$$
- Der chiffrierte Text ist dann $y_1, y_2, y_3, \ldots, y_n$ mit
$$y_i = E(K_i, x_i).$$

Die Enigma-Chiffrierung ist also eine Art Stromchiffrierung.

Im folgenden werden die (mathematischen) Funktionen der wichtigsten Bauteile und Operationen beschrieben:

- die drei Rotoren bzw. Walzen,
- die Umkehrwalze,
- das Steckerbrett,
- das Weiterdrehen der Walzen.

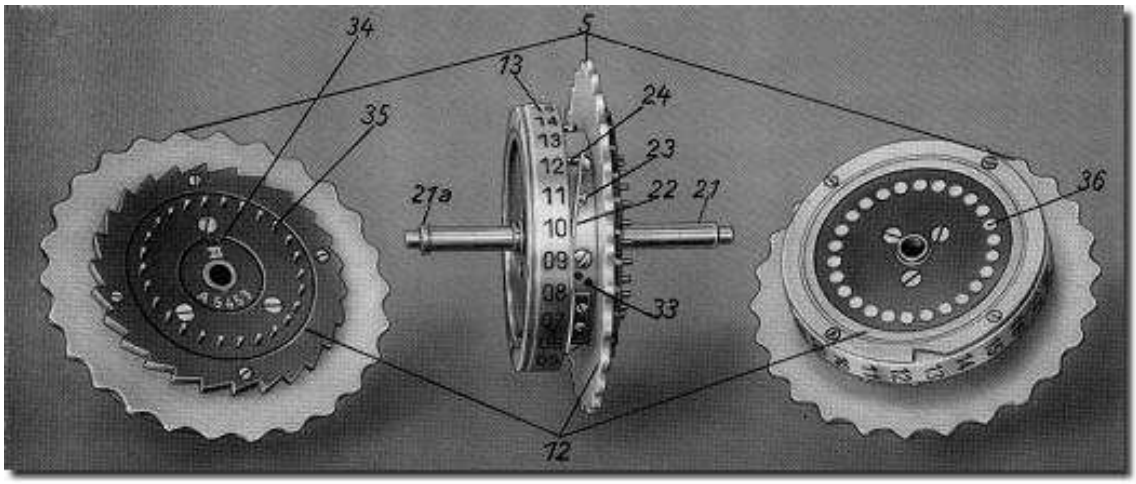
Bild III



12 Chiffrierwalzen
13 Zahlenringe
17 Metalldeckel
19 Haltehebel
20 Umkehrwalze
25 Stirnwand
26 Haken
27 Batteriekasten
29 Tastenbolzen
38 Glühlampenfeld
39 Lampenprüfung
40 Kabelprüfung
41 Unverw. Buchsen zur Kabelprüfung

3. Die Rotoren/Walzen

Ein Rotor ist eine Scheibe/Walze mit 26 elektrischen Kontakten auf der linken und rechten Seite.



Die Kontakte der linken und rechten Seite innen sind verdrahtet, so daß man eine Permutation

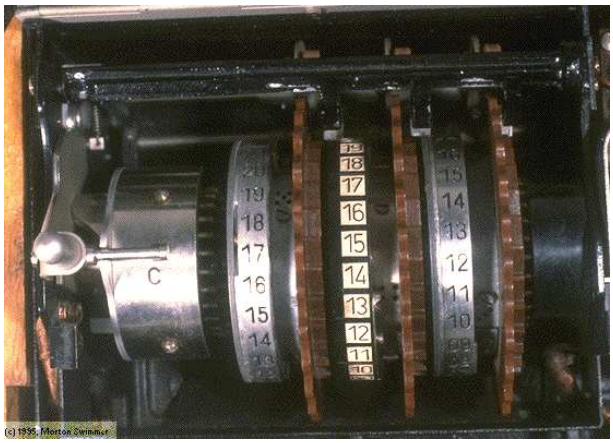
$$R : \mathfrak{A} \to \mathfrak{A}$$

erhält, wenn man die Kontakte der Reihe nach mit den Buchstaben A, B, ..., Z bezeichnet.

Für die hier betrachtete Enigma gibt es 5 verschiedene Walzen, die mit I=1, II=2, III=3, IV=4, V=5 bezeichnet werden. Die zugehörigen Permutationen sind:

$$\begin{aligned}
R_1 &= (\text{AELTPHQXRU})(\text{BKNW})(\text{CMOY})(\text{DFG})(\text{IV})(\text{JZ})(\text{S}) \\
R_2 &= (\text{A})(\text{BJ})(\text{CDKLHUP})(\text{ESZ})(\text{FIXVYOMW})(\text{GR})(\text{NT})(\text{Q}) \\
R_3 &= (\text{ABDHPEJT})(\text{CFLVMZOYQIRWUKXSG})(\text{N}) \\
R_4 &= (\text{AEPLIYWCOXMRFZBSTGJQNH})(\text{DV})(\text{KU}) \\
R_5 &= (\text{AVOLDRWFIUQ})(\text{BZKSMNHYC})(\text{EGTJPX})
\end{aligned}$$

Bei der Benutzung der Enigma werden von den 5 Rotoren 3 ausgewählt und in die Maschine eingesetzt. Die Walzenlage (w_L, w_M, w_R) gibt also an, daß der linke Rotor R_{w_L}, der mittlere R_{w_M}, der rechte R_{w_R} ist.



Die entscheidende Eigenschaft eines Rotors ist, daß man ihn jeweils um $\frac{1}{26}$ weiterdrehen kann und dadurch neue Permutationen erhält. Beim Drehen um $\frac{s}{26}$ erhält man die Permutation

$$x \mapsto R(x+s) - s.$$

Die entsprechende *Rotorstellung* s konnte man ablesen. Bei den drei Rotoren hat man dann die Rotorstellungen (s_L, s_M, s_R).

Beispiel: Beim Drehen der Walze I ergeben sich folgende Permutationen:

s	$x \mapsto R_1(x+s) - s$
0	(AELTPHQXRU)(BKNW)(CMOY)(DFG)(IV)(JZ)(S)
1	(AJMV)(BLNX)(CEF)(DKSOGPWQTZ)(HU)(IY)(R)
2	(AKMW)(BDE)(CJRNFOVPSY)(GT)(HX)(ILUZ)(Q)
3	(ACD)(BIQMENUORX)(FS)(GW)(HKTY)(JLVZ)(P)
4	(AHPLDMTNQW)(BCZ)(ER)(FV)(GJSX)(IKUY)(O)
5	(ABY)(CLSMPVZGOK)(DQ)(EU)(FIRW)(HJTX)(N)
6	(AXZ)(BKRLOUYFNJ)(CP)(DT)(EHQV)(GISW)(M)
7	(AJQKNTXEMI)(BO)(CS)(DGPU)(FHRV)(L)(WYZ)
8	(AN)(BR)(CFOT)(DLHZIPJMSW)(EGQU)(K)(VXY)
9	(AQ)(BENS)(CKGYHOILRV)(DFPT)(J)(MZ)(UWX)
10	(ADMR)(BJFXGNHKQU)(CEOS)(I)(LY)(PZ)(TVW)
11	(AIEWFMGJPT)(BDNR)(CLQZ)(H)(KX)(OY)(SUV)
12	(ACMQ)(BKPY)(DVELFIOSZH)(G)(JW)(NX)(RTU)
13	(AJOX)(BLPZ)(CUDKEHNRYG)(F)(IV)(MW)(QST)
14	(AKOY)(BTCJDGMQXF)(E)(HU)(INWZ)(LV)(PRS)
15	(ASBICFLPWE)(D)(GT)(HMVY)(JNXZ)(KU)(OQR)
16	(AHBEKOVDZR)(C)(FS)(GLUX)(IMWY)(JT)(NPQ)
17	(ADJNUCYQZG)(B)(ER)(FKTW)(HLVX)(IS)(MOP)
18	(A)(BXPYFZCIMT)(DQ)(EJSV)(GKUW)(HR)(LNO)
19	(AWOXEYBHLS)(CP)(DIRU)(FJTV)(GQ)(KMN)(Z)
20	(AGKRZVNWDX)(BO)(CHQT)(EISU)(FP)(JLM)(Y)
21	(AN)(BGPS)(CWZFJQYUMV)(DHRT)(EO)(IKL)(X)
22	(AFOR)(BVYEIPXTLU)(CGQS)(DN)(HJK)(MZ)(W)
23	(AUXDHOWSKT)(BFPR)(CM)(ENQZ)(GIJ)(LY)(V)
24	(AEOQ)(BL)(CGNVRJSZTW)(DMPY)(FHI)(KX)(U)
25	(AK)(BFMUQIRYSV)(CLOX)(DNPZ)(EGH)(JW)(T)

Bei den verwendeten Rotoren konnte man auch den inneren Teil jeweils in $\frac{1}{26}$-Schritten weiterdrehen. Die entsprechende Stellung wurde als *Ringstellung* r bezeichnet, wobei die Werte $0 \leq r \leq 25$ möglich sind. Bei Walze w, Ringstellung r und Rotorstellung s ergibt sich die Permutation $R[w,r,s]$ mit

$$R[w,r,s](x) = R_w(x+s-r) + r - s.$$

Bemerkung: Es ist

$$R[w,r,s]^{-1}(x) = R_w^{-1}(x+s-r) + r - s.$$

4. Die Umkehrwalze

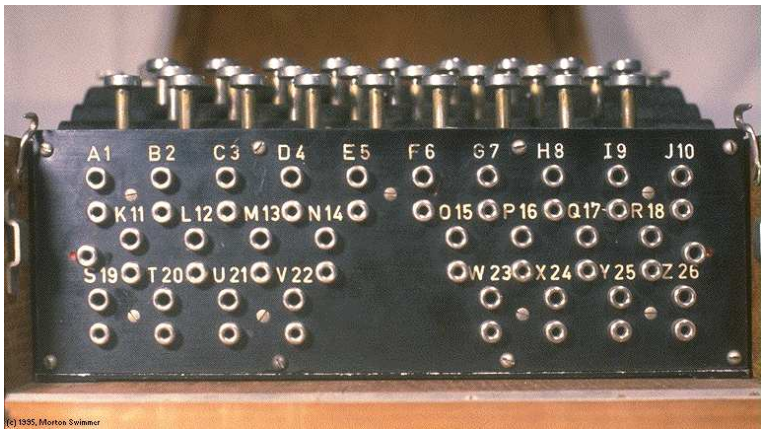
Eine links neben den anderen Rotoren/Walzen festeingebaute *Umkehrwalze* oder *Reflector* liefert eine Permutation

$$U = (\mathrm{AY})(\mathrm{BR})(\mathrm{CU})(\mathrm{DH})(\mathrm{EQ})(\mathrm{FS})(\mathrm{GL})(\mathrm{IP})(\mathrm{JX})(\mathrm{KN})(\mathrm{MO})(\mathrm{TZ})(\mathrm{VW}),$$

wobei also $U^{-1} = U$ gilt. Man beachte, daß U keine Fixpunkte besitzt, d.h. es gibt kein $x \in \mathfrak{A}$ mit $U(x) = x$.

5. Die Steckerverbindung

Am Steckerbrett konnte man durch Kabel jeweils zwei Buchstaben miteinander verbinden und erhielt dadurch eine Involution $S : \mathfrak{A} \to \mathfrak{A}$, die als *Steckerverbindung* bezeichnet wird.



Beispiel: Verbindet man die Buchstaben C und L, sowie X und M, erhält man die Steckerverbindung

$$S = (\mathrm{CL})(\mathrm{MX}).$$

6. Weiterdrehen — die Schlüsselnachfolgeabbildungen σ und Σ

Beim Eingeben eines Zeichens bzw. beim Drücken einer Taste werden die drei Walzen weitergedreht. Dies geschieht ähnlich wie beim Kilometerzähler eines Autos, aber doch nicht ganz so. Jede Walze w hat eine Nut/Einkerbung, die das Weiterdrehen regelt.

Die Nut befindet sich an einer Stelle $\nu(w)$, wobei bei unseren Walzen folgende Werte gelten:

$$\nu(1) = \mathrm{Q}, \quad \nu(2) = \mathrm{E}, \quad \nu(3) = \mathrm{V}, \quad \nu(4) = \mathrm{J}, \quad \nu(5) = \mathrm{Z}.$$

Beim Weiterdrehen verändern sich die Rotorstellungen (s_L, s_M, s_R). Mathematisch läßt sich das so beschreiben:

Die Nachfolgeabbildung σ: Gegeben sei eine Walzenlage (w_L, w_M, w_R) und eine Rotorstellung (s_L, s_M, s_R). Dann erhält man beim Weiterdrehen die Rotorstellung

$$\sigma(w_M, w_R, s_L, s_M, s_R) = (s'_L, s'_M, s'_R)$$

mit

$$(s'_L, s'_M, s'_R) = \begin{cases} (s_L + 1, s_M + 1, s_R + 1), & \text{falls } s_M = \nu(w_M), \\ (s_L, s_M + 1, s_R + 1), & \text{falls } s_R = \nu(w_R) \text{ und } s_M \neq \nu(w_M), \\ (s_L, s_M, s_R + 1), & \text{sonst.} \end{cases}$$

Mit Worten:

- Ist weder in der Mitte noch rechts die Nutstellung erreicht, d.h. $s_M \neq \nu(w_M)$, $s_R \neq \nu(w_R)$, so wird nur die rechte Walze weitergedreht.
- Ist die rechte Walze w_R in Nutstellung, die mittlere aber nicht, d.h. $s_R = \nu(w_R)$, $s_M \neq \nu(w_M)$, so wird die mittlere und rechte Walze weitergedreht.
- Ist die mittlere Walze w_M in Nutstellung, d.h. $s_M = \nu(w_M)$, so werden alle drei Walzen weitergedreht.

Beispiel: Wir wählen die Walzenlage $(1, 2, 3)$. Dann ist $\nu(2) = E$, $\nu(3) = V$. Wir beginnen mit der Rotorstellung AAA und erhalten dann durch Weiterdrehen die nachfolgende Rotorstellungsfolge:

```
AAA AAB AAC AAD AAE AAF AAG AAH AAI AAJ AAK AAL AAM AAN AAO AAP AAQ AAR AAS AAT
AAU AAV ABW ABX ABY ABZ ABA ABB ABC ABD ABE ABF ABG ABH ABI ABJ ABK ABL ABM ABN
...
ADC ADD ADE ADF ADG ADH ADI ADJ ADK ADL ADM ADN ADO ADP ADQ ADR ADS ADT ADU ADV
AEW BFX BFY BFZ BFA BFB BFC BFD BFE BFF BFG BFH BFI BFJ BFK BFL BFM BFN BFO BFP
...
BZQ BZR BZS BZT BZU BZV BAW BAX BAY BAZ BAA BAB BAC BAD BAE BAF BAG BAH BAI BAJ
...
BDM BDN BDO BDP BDQ BDR BDS BDT BDU BDV BEW CFX CFY CFZ CFA CFB CFC CFD CFE CFF
...
ZDM ZDN ZDO ZDP ZDQ ZDR ZDS ZDT ZDU ZDV ZEW AFX AFY AFZ AFA AFB AFC AFD AFE AFF
...
AZG AZH AZI AZJ AZK AZL AZM AZN AZO AZP AZQ AZR AZS AZT AZU AZV AAW AAX AAY AAZ
AAA AAB AAC ...
```

Bemerkung: Die Folge der Rotorstellungen (s_L, s_M, s_R) ist periodisch mit Periodenlänge $16900 = 26 \cdot 26 \cdot 25$ mit ein paar Ausnahmefällen:

- Ist $s_M = \nu(w_M) + 1$ und $s_R = \nu(w_R) + 1$, oder $s_M = \nu(w_M)$ und $s_R \neq \nu(w_M), \nu(w_M) + 1$, so gibt es eine Vorperiode der Länge 1.
- Ist $s_M = \nu(w_M)$ und $s_R = \nu(w_R)$, so gibt es eine Vorperiode der Länge 2.

Die Schlüsselnachfolgeabbildung Σ führen wir nur der einfacheren Bezeichnung halber ein: $\Sigma : \mathfrak{S} \to \mathfrak{S}$ wird definiert durch

$$\Sigma(w_L, w_M, w_R, r_L, r_M, r_R, S, s_L, s_M, s_R) = (w_L, w_M, w_R, r_L, r_M, r_R, S, s'_L, s'_M, s'_R),$$

wobei

$$(s'_L, s'_M, s'_R) = \sigma(w_M, w_R, s_L, s_M, s_R)$$

gilt.

Bemerkung: Die Schlüsselfolge K_i zu einem Anfangsschlüssel $K = K_0$ mit $K_i = \Sigma(K_{i-1})$ ist periodisch mit Periode 16900, wobei eine Vorperiode der Länge 1 oder 2 in Ausnahmefällen möglich ist.

Bemerkung: Die Ringstellung (r_L, r_M, r_R) wird am Innenteil der Walzen eingestellt und hat keinen Einfluß auf das Weiterdrehen.

7. Die Verschlüsselungsabbildung E

Ist $K = (w_L, w_M, w_R, r_L, r_M, r_R, S, s_L, s_M, s_R) \in \mathfrak{S}$ und $x \in \mathfrak{A}$, so ist die Verschlüsselungsabbildung

$$E : \mathfrak{S} \times \mathfrak{A} \to \mathfrak{A}$$

gegeben durch

$$\begin{aligned} E(K, x) \quad = \quad & (S \circ R[w_R, r_R, s_R]^{-1} \circ R[w_M, r_M, s_M]^{-1} \circ R[w_L, r_L, s_L]^{-1} \circ \\ & \circ U \circ R[w_L, r_L, s_L] \circ R[w_M, r_M, s_M] \circ R[w_M, r_M, s_M] \circ R[w_R, r_R, s_R] \circ S)(x). \end{aligned}$$

Wir schreiben auch

$$E_K : \mathfrak{A} \to \mathfrak{A} \quad \text{mit} \quad E_K(x) = E(K, x).$$

Bei Tastendruck wird also zuerst das Steckerbrett berücksichtigt, dann werden die drei Walzen von rechts nach links durchlaufen, dann kommt die Umkehrwalze, dann wieder die drei Walzen rückwärts von links nach rechts, schließlich wieder das Steckerbrett.

Beispiele:

Schlüssel K	Permutation E_K
(1,2,3,A,A,A,(),A,A,A)	(AU)(BE)(CJ)(DO)(FT)(GP)(HZ)(IW)(KN)(LS)(MR)(QV)(XY)
(1,2,3,A,A,A,(AB),A,A,A)	(AE)(BU)(CJ)(DO)(FT)(GP)(HZ)(IW)(KN)(LS)(MR)(QV)(XY)
(1,2,3,A,A,A,(AU),A,A,A)	(AU)(BE)(CJ)(DO)(FT)(GP)(HZ)(IW)(KN)(LS)(MR)(QV)(XY)
(5,2,3,P,W,F,(AP)(DK)(FJ)(OW), N,X,W)	(AH)(BG)(CR)(DN)(EZ)(FK)(IX)(JM)(LO)(PU)(QY)(ST)(VW)
(2,4,3,Z,T,R,(BV)(MN),G,F,D)	(AS)(BY)(CW)(DN)(EG)(FO)(HP)(IV)(JQ)(KR)(LM)(TZ)(UX)

Bemerkungen:

(1) Die Permutationen E_K sind Involutionen, d.h. $E_K^{-1} = E_K$.

(2) E_K besitzt keine Fixpunkte, denn angenommen es wäre $E_K(x) = x$, so hätte man

$$U\,(R[w_L, r_L, s_L](R[w_M, r_M, s_M](R[w_R, r_R, s_R](S(x))))) =$$
$$= R[w_L, r_L, s_L](R[w_M, r_M, s_M](R[w_R, r_R, s_R](S(x)))),$$

also hätte auch die Umkehrwalzenpermutation U einen Fixpunkt, was nach Konstruktion nicht der Fall war.

(3) Ist $K = (w_L, w_M, w_R, r_L, r_M, r_R, S, s_L, s_M, s_R)$ und $\widetilde{K} = (w_L, w_M, w_R, r_L, r_M, r_R, (), s_L, s_M, s_R)$, so gilt $E_K = S \circ E_{\widetilde{K}} \circ S$. Anders ausgedrückt

$$E(K, x) = S(E(\widetilde{K}, Sx)).$$

8. Der Enigma-Verschlüsselungsalgorithmus

Eingabe: Ein Schlüssel $K = (w_L, w_M, w_R, r_L, r_M, r_R, S, s_L, s_M, s_R)$, eine Buchstabenfolge $x_1, x_2, x_3, \ldots, x_n$.
Ausgabe: Die chiffrierte Buchstabenfolge $y_1, y_2, y_3, \ldots, y_n$.

(1) Setze $i = 0$ und $K_0 = K$. (Initialisierung)
(2) Setze $i := i + 1$. Ist $i > n$ beende das Verfahren.
(3) Berechne $K_i = \Sigma(K_{i-1})$.
(4) Berechne $y_i = E(K_i, x_i)$ und gib y_i aus.
(5) Gehe zurück zu 2.

Bemerkung: Da die Abbildungen E_{K_i} Involutionen sind, ist $x_i = E(K_i, y_i)$, zur Entschlüsselung muß man also nur den Algorithmus auf $y_1, \ldots, y_n$ mit Eingangsschlüssel K anwenden.

Beispiel: K = (1,2,3,A,A,A,(),A,A,A).

i	s_L	s_M	s_R	E_{K_i}
1	A	A	B	(AB)(CQ)(DM)(EF)(GX)(HI)(JS)(KW)(LP)(NY)(OT)(RV)(UZ)
2	A	A	C	(AD)(BJ)(CR)(EL)(FK)(GW)(HP)(IO)(MY)(NQ)(SX)(TZ)(UV)
3	A	A	D	(AZ)(BL)(CE)(DJ)(FU)(GI)(HT)(KM)(NO)(PS)(QV)(RX)(WY)
4	A	A	E	(AG)(BC)(DL)(EW)(FT)(HK)(IQ)(JZ)(MO)(NS)(PX)(RU)(VY)
5	A	A	F	(AO)(BS)(CN)(DI)(ER)(FG)(HM)(JP)(KV)(LU)(QW)(TX)(YZ)
6	A	A	G	(AW)(BY)(CM)(DV)(EN)(FQ)(GZ)(HJ)(IL)(KO)(PU)(RT)(SX)
7	A	A	H	(AC)(BJ)(DF)(EK)(GZ)(HT)(IP)(LU)(MO)(NR)(QW)(SV)(XY)
8	A	A	I	(AX)(BI)(CJ)(DU)(EY)(FS)(GL)(HV)(KZ)(MN)(OQ)(PW)(RT)
9	A	A	J	(AL)(BF)(CZ)(DR)(EW)(GY)(HT)(IJ)(KS)(MO)(NV)(PU)(QX)
10	A	A	K	(AT)(BK)(CE)(DZ)(FQ)(GY)(HN)(IU)(JW)(LP)(MV)(OS)(RX)

SAMSTAG wird also zu JDKNXWZ verschlüsselt.

Beispiel: K = (2,1,3,Z,W,D,(BL)(EZ)(IU)(JO)(MV)(PX)(RW),A,G,I).

i	s_L	s_M	s_R	E_{K_i}
1	A	G	J	(AQ)(BU)(CK)(DO)(EP)(FM)(GL)(HX)(IN)(JV)(RZ)(SY)(TW)
2	A	G	K	(AC)(BH)(DF)(ET)(GS)(IZ)(JO)(KU)(LN)(MX)(PQ)(RY)(VW)
3	A	G	L	(AS)(BG)(CI)(DK)(EW)(FR)(HO)(JQ)(LV)(MZ)(NU)(PX)(TY)
4	A	G	M	(AV)(BQ)(CE)(DW)(FO)(GS)(HK)(IU)(JP)(LT)(MZ)(NR)(XY)
5	A	G	N	(AG)(BL)(CT)(DV)(EW)(FS)(HN)(IQ)(JY)(KX)(MP)(OR)(UZ)
6	A	G	O	(AJ)(BZ)(CG)(DT)(EY)(FU)(HW)(IR)(KN)(LS)(MO)(PV)(QX)
7	A	G	P	(AU)(BP)(CK)(DI)(EQ)(FM)(GW)(HZ)(JS)(LR)(NY)(OV)(TX)
8	A	G	Q	(AM)(BP)(CU)(DQ)(EY)(FZ)(GS)(HT)(IL)(JX)(KV)(NW)(OR)
9	A	G	R	(AI)(BO)(CQ)(DF)(EJ)(GS)(HV)(KY)(LZ)(MT)(NW)(PU)(RX)
10	A	G	S	(AG)(BM)(CR)(DK)(EL)(FX)(HO)(IS)(JN)(PU)(QV)(TZ)(WY)

Beispiel: $K = (5, 4, 1, \mathrm{Y}, \mathrm{X}, \mathrm{C}, (\mathrm{ER})(\mathrm{IU})(\mathrm{OP})(\mathrm{QW})(\mathrm{TZ}), \mathrm{L}, \mathrm{I}, \mathrm{M})$.

i	s_L	s_M	s_R	E_{K_i}
1	L	I	N	(AL)(BO)(CP)(DK)(EX)(FG)(HV)(IY)(JM)(NQ)(RS)(TU)(WZ)
2	L	I	O	(AV)(BY)(CQ)(DU)(EH)(FM)(GK)(IL)(JZ)(NX)(OR)(PT)(SW)
3	L	I	P	(AQ)(BE)(CM)(DT)(FN)(GK)(HP)(IS)(JU)(LY)(OX)(RW)(VZ)
4	L	I	Q	(AF)(BJ)(CG)(DS)(EI)(HN)(KL)(MT)(OX)(PV)(QY)(RZ)(UW)
5	L	J	R	(AP)(BI)(CY)(DQ)(EL)(FX)(GU)(HV)(JM)(KW)(NO)(RT)(SZ)
6	M	K	S	(AM)(BY)(CR)(DI)(EL)(FZ)(GJ)(HT)(KP)(NV)(OQ)(SW)(UX)
7	M	K	T	(AP)(BR)(CY)(DU)(ES)(FI)(GN)(HK)(JQ)(LM)(OT)(VW)(XZ)
8	M	K	U	(AK)(BF)(CV)(DO)(EI)(GT)(HZ)(JR)(LY)(MP)(NS)(QU)(WX)
9	M	K	V	(AB)(CI)(DX)(ES)(FL)(GR)(HJ)(KW)(MV)(NO)(PT)(QZ)(UY)
10	M	K	W	(AL)(BT)(CN)(DE)(FQ)(GZ)(HY)(IR)(JW)(KP)(MU)(OV)(SX)

9. Ein C-Programm zur Enigma-Simulation

```
/* enigma_simulator.c
   Version vom 20.7.2001
   (Wahrscheinlich) Simulation einer Wehrmachts-Enigma um 1938.
   5 moegliche Walzen I=1, II=2, III=3, IV=4, V=5, feste Umkehrwalze 'B'
   Schluessel:
   1. Walzenlage (w_L,w_M,w_R)
   2. Ringstellung (r_L,r_M,r_R)
   3. Steckerverbindung S
   4. Grundstellung (s_L,s_M,s_R)
*/

char Rotor[5][26]={ "EKMFLGDQVZNTOWYHXUSPAIBRCJ",   // Rotor I
                    "AJDKSIRUXBLHWTMCQGZNPYFVOE",   // Rotor II
                    "BDFHJLCPRTXVZNYEIWGAKMUSQO",   // Rotor III
                    "ESOVPZJAYQUIRHXLNFTGKDCMWB",   // Rotor IV
                    "VZBRGITYUPSDNHLXAWMJQOFECK"    // Rotor V
                  };
char Nut[5][1]={ "Q", "E", "V", "J" , "Z"};
char ReflectorB[26]="YRUHQSLDPXNGOKMIEBFZCWVJAT";   // Reflector B

main()
{
  int i, n, x, c[80];
  int R_L[26], R_Linvers[26], nut_L, w_L, r_L, s_L;
  int R_M[26], R_Minvers[26], nut_M, w_M, r_M, s_M;
```

```
  int R_R[26], R_Rinvers[26], nut_R, w_R, r_R, s_R;
  int S[26], U[26];
  char Eingabe[80];

  /* Umkehrwalze ***************************************************/

  for (i=0;i<26;i++)
    U[i]=ReflectorB[i]-'A';

  /* Walzenlage ****************************************************/

  printf("Walzenlage: ");
  for (n=0;n<3;)
  {
    c[n]=getchar();
    if ('1'<=c[n] && c[n]<='8') n++;
  }
  w_L=c[0]-'0'; w_M=c[1]-'0'; w_R=c[2]-'0';

  for (i=0;i<26;i++)
  {
    R_L[i]=Rotor[w_L-1][i]-'A';
    R_M[i]=Rotor[w_M-1][i]-'A';
    R_R[i]=Rotor[w_R-1][i]-'A';
    R_Linvers[R_L[i]]=i;
    R_Minvers[R_M[i]]=i;
    R_Rinvers[R_R[i]]=i;
  }
  nut_L=Nut[w_L-1][0]-'A';
  nut_M=Nut[w_M-1][0]-'A';
  nut_R=Nut[w_R-1][0]-'A';

  /* Ringstellung **************************************************/

  printf("Ringstellung: ");
  for (n=0;n<3;)
  {
    c[n]=getchar();
    if ('A'<=c[n] && c[n]<='Z') n++;
  }
  r_L=c[0]-'A'; r_M=c[1]-'A'; r_R=c[2]-'A';
  c[n]=getchar(); // liest \n ein

  /* Steckerverbindung *********************************************/

  printf("Steckerverbindung: ");
  n=0;
  while ((c[n]=getchar())!=10)
  {
    if ('A'<=c[n] && c[n]<='Z') n++;
  }

  for (i=0;i<26;i++) S[i]=i;
  for (i=0;i<n/2;i++)
```

```
{
  S[c[2*i]-'A']=c[2*i+1]-'A';
  S[c[2*i+1]-'A']=c[2*i]-'A';
}

/* Grundstellung ************************************************/

printf("Grundstellung: ");
for (n=0;n<3;)
{
  c[n]=getchar();
  if ('A'<=c[n] && c[n]<='Z') n++;
}
s_L=c[0]-'A'; s_M=c[1]-'A'; s_R=c[2]-'A';


/* Schluessel-Ausgabe *******************************************/

printf("Schluessel:   Walzenlage   Ringstellung   ");
printf("Steckerverbindung   Grundstellung\n");
printf("%d%d%d   %c%c%c   ",w_L,w_M,w_R,r_L+'A',r_M+'A',r_R+'A');
for (i=0;i<26;i++)
  if (i<S[i]) printf("(%c%c)",'A'+i,'A'+S[i]);
printf("   %c%c%c\n",s_L+'A',s_M+'A',s_R+'A');

/* Verschluesselung *********************************************/

printf("Eingabe von Text in Grossbuchstaben oder '.' zur ");
printf("Beendigung.\n");

for (;;)
{
  printf("Eingabe: ");
  scanf("%s",Eingabe);

  if (Eingabe[0]=='.') return;

  printf("Ausgabe: ");
  for (i=0;i<strlen(Eingabe);i++)
  {
    /* Zunaechst werden die Rotoren weitergedreht: */
    if (s_M==nut_M) { s_L++; s_M++; s_R++; }
    else if (s_R==nut_R) { s_M++; s_R++; }
    else { s_R++; }
    s_L%=26; s_M%=26; s_R%=26;

    x=Eingabe[i]-'A';
    x=S[x]; /* Steckerverbindung */
    x=(R_R[(x+26-r_R+s_R)%26]+(26+r_R-s_R))%26; /* Rotor R_R */
    x=(R_M[(x+26-r_M+s_M)%26]+(26+r_M-s_M))%26; /* Rotor R_M */
    x=(R_L[(x+26-r_L+s_L)%26]+(26+r_L-s_L))%26; /* Rotor R_L */
    x=U[x]; /* Umkehrwalze */
    x=(R_Linvers[(x+26-r_L+s_L)%26]+(26+r_L-s_L))%26; /* Rotor R_L invers */
    x=(R_Minvers[(x+26-r_M+s_M)%26]+(26+r_M-s_M))%26; /* Rotor R_M invers */
```

```
      x=(R_Rinvers[(x+26-r_R+s_R)%26]+(26+r_R-s_R))%26; /* Rotor R_R invers */
      x=S[x]; /* Steckerverbindung */
      printf("%c",x+'A');
    }
    printf("\n");
  }
}
```

10. Der Schlüsselraum $\mathfrak{S}$

Wir wollen die Größe des Schlüsselraums $\mathfrak{S}$ bestimmen. Sei

$$K = (w_L, w_M, w_R, r_L, r_M, r_R, S, s_L, s_M, s_R) \in \mathfrak{S}.$$

Für die Walzenlage (w_L, w_M, w_R) muß man drei Zahlen aus $\{1, 2, 3, 4, 5\}$ wählen und anordnen, also gibt es

$$\binom{5}{3} \cdot 3! = 60$$

Möglichkeiten.

Für die Ringstellung (r_L, r_M, r_R) und die Rotorstellung (s_L, s_M, s_R) sind die Zahlen mit $0 \leq r_L, r_M, r_R, s_L, s_M, s_R \leq 25$ frei wählbar, also hat man 26^6 Möglichkeiten.

Wir wollen jetzt die Steckerverbindung S betrachten. Jede Involution $S : \mathfrak{A} \to \mathfrak{A}$ ist (prinzipiell) möglich. Für eine Involution $S = (a_1 a_2)(a_3 a_4) \ldots (a_{2m-1} a_{2m})$ mit m disjunkten Transpositionen gibt es

$$\frac{1}{m!}\left(\frac{26 \cdot 25}{2} \cdot \frac{24 \cdot 23}{2} \cdot \ldots \frac{(28-2m)(27-2m)}{2}\right)$$

Möglichkeiten, also erhält man folgende Tabelle:

m	Anzahl der Involutionen aus m (disjunkten) Transpositionen
0	1
1	325
2	44850
3	3453450
4	164038875
5	5019589575
6	100391791500
7	1305093289500
8	10767019638375
9	53835098191875
10	150738274937250
11	205552193096250
12	102776096548125
13	7905853580625

Insgesamt gibt es 532985208200576 Involutionen.

Damit erhalten wir für die Größe des Schlüsselraums

$$\begin{aligned} \#\mathfrak{S} &= 60 \cdot 26^6 \cdot 532985208200576 = \\ &= 60 \cdot 308915776 \cdot 532985208200576 = 98788523512681499212185 60 \approx 10^{25}. \end{aligned}$$

Man war daher davon überzeugt, daß die Enigma-Chiffrierung sicher war.

11. Historisches Beispiel

Zu verschiedenen Zeiten wurde die Enigma unterschiedlich gehandhabt.

Geheim!

Nicht ins Flugzeug mitnehmen!

Sonder-Maschinenschlüssel BGT

Datum	Walzenlage	Ringstellung	Steckerverbindungen	Kenngruppe
31.	I V III	06 20 24	UA PF RQ SO NI EY BG HL TX ZJ	jeu nyq aqm
30.	V II III	01 07 12	OF KV JM IB UW LX TD QS NA ZH	azs zds kck
29.	IV I V	11 17 26	CI OK PV ZL HX NB AW DJ FE ST	kap gwh lyx

Die Skizze zeigt die Vorgabe von Walzenlage (w_L, w_M, w_R), Ringstellung (r_L, r_M, r_R) und Steckerverbindungen S.

Wir erläutern eine Anwendungsmöglichkeit anhand des nachfolgenden Beispiels vom 21. September 1938. Der geheime Teil des Schlüssels ist

$$K = (2, 1, 3, \mathrm{Z}, \mathrm{W}, \mathrm{D}, (\mathrm{EZ})(\mathrm{BL})(\mathrm{XP})(\mathrm{WR})(\mathrm{IU})(\mathrm{VM})(\mathrm{JO}), \_, \_, \_).$$

Eine erste Grundstellung (s_L^0, s_M^0, s_R^0) wurde vom Absender frei gewählt und öffentlich übertragen, im Beispiel ist dies FRX, das zur Sicherheit zweimal notiert ist. Dann wurde die eigentliche Grundstellung (s_L, s_M, s_R) frei gewählt und mit dem Schlüssel

$$K^0 = (w_L, w_M, w_R, r_L, r_M, r_R, S, s_L^0, s_M^0, s_R^0)$$

zweimal verschlüsselt, d.h. $s_L s_M s_R s_L s_M s_R$ zu $y_1 y_2 y_3 y_4 y_5 y_6$. Die Buchstabenfolge $y_1 y_2 y_3 y_4 y_5 y_6$ wurde übertragen. Der eigentliche Text wurde mit dem Schlüssel

$$K = (w_L, w_M, w_R, r_L, r_M, r_R, S, s_L, s_M, s_R)$$

verschlüsselt und übertragen. Allerdings sind die Zeichen 11-15 zur Erkennung eingefügt und haben nichts mit der Verschlüsselung zu tun. Hier nun das verschlüsselte Beispiel:

```
AN HEERESGRUPPENKOMMANDO 2
2109 - 1750 - 3 TLE - FRX FRX

1TL - 172
HCALN UQKRQ AXPWT WUQTZ KFXZO MJFOY RHYZW VBXYS IWMMV WBLEB
DMWUW BTVHM RFLKS DCCEX IYPAH RMPZI OVBBR VLNHZ UPOSY EIPWJ
TUGYO SLAOX RHKVC HQOSV DTRBP DJEUK SBBXH TYGVH GFICA CVGUV
OQFAQ WBKXZ JSQJF ZPEVJ RO

2TL - 166
ZZWTV SYBDO YDTEC DMVWQ KWJPZ OCZJW XOFWP XWGAR KLRLX TOFCD
SZHEV INQWI NRMBS QPTCK LKCQR MTYVG UQODM EIEUT VSQFI MWORP
RPLHG XKMCM PASOM YRORP CVICA HUEAF BZNVR VZWXX MTWOE GIEBS
ZZQIU JAPGN FJXDK I

3TL - 176
DHHAO FWQQM EIHBF BMHTT YFBHK YYXJK IXKDF RTSHB HLUEJ MFLAC
ZRJDL CJZVK HFBYL GFSEW NRSGS KHLFW JKLLZ TFMWD QDQQV JUTJS
VPRDE MUVPM BPBXX USOPG IVHFC ISGPY IYKST VQUIO CAVCW AKEQQ
EFRVM XSLQC FPFTF SPIIU ENLUW O
```

Wir wollen dies entschlüsseln und machen einige Bemerkungen:
2109 - 1750 steht für den 21. September, 17:50 Uhr, 3 TLE steht für 3 Teile, FRX ist die erste Grundstellung. Der erste Schlüssel ist also

$$K^0 = (2, 1, 3, \mathrm{Z}, \mathrm{W}, \mathrm{D}, (\mathrm{EZ})(\mathrm{BL})(\mathrm{XP})(\mathrm{WR})(\mathrm{IU})(\mathrm{VM})(\mathrm{JO}), \mathrm{F}, \mathrm{R}, \mathrm{X}).$$

Damit entschlüsselt man jeweils die ersten 6 Zeichen der 3 Teile und erhält

$$\mathrm{HCALNU} \mapsto \mathrm{AGIAGI}, \quad \mathrm{ZZWTVS} \mapsto \mathrm{YBEYBE}, \quad \mathrm{DHHAOF} \mapsto \mathrm{LUNLUN}.$$

Die eigentlichen Schlüssel für die Teile 1 bis 3 sind also

$$\begin{aligned} K_{\text{Teil 1}} &= (2,1,3,\text{Z},\text{W},\text{D},(\text{EZ})(\text{BL})(\text{XP})(\text{WR})(\text{IU})(\text{VM})(\text{JO}),\text{A},\text{G},\text{I}),\\ K_{\text{Teil 2}} &= (2,1,3,\text{Z},\text{W},\text{D},(\text{EZ})(\text{BL})(\text{XP})(\text{WR})(\text{IU})(\text{VM})(\text{JO}),\text{Y},\text{B},\text{E}),\\ K_{\text{Teil 3}} &= (2,1,3,\text{Z},\text{W},\text{D},(\text{EZ})(\text{BL})(\text{XP})(\text{WR})(\text{IU})(\text{VM})(\text{JO}),\text{L},\text{U},\text{N}). \end{aligned}$$

Die Buchstaben 11-15 dienen jeweils der Erkennung und müssen bei der Entschlüsselung weggelassen werden. Genauso sind die ersten 6 Buchstaben zu streichen. Also haben wir die nachfolgenden Buchstabenfolgen mit den Schlüsseln $K_{\text{Teil 1}}$ bzw. $K_{\text{Teil 2}}$ bzw. $K_{\text{Teil 3}}$ zu entschlüsseln:

```
Teil 1:
       QKRQ       WUQTZ KFXZO MJFOY RHYZW VBXYS IWMMV WBLEB
DMWUW BTVHM RFLKS DCCEX IYPAH RMPZI OVBBR VLNHZ UPOSY EIPWJ
TUGYO SLAOX RHKVC HQOSV DTRBP DJEUK SBBXH TYGVH GFICA CVGUV
OQFAQ WBKXZ JSQJF ZPEVJ RO

Teil 2:
       YBDO       DMVWQ KWJPZ OCZJW XOFWP XWGAR KLRLX TOFCD
SZHEV INQWI NRMBS QPTCK LKCQR MTYVG UQODM EIEUT VSQFI MWORP
RPLHG XKMCM PASOM YRORP CVICA HUEAF BZNVR VZWXX MTWOE GIEBS
ZZQIU JAPGN FJXDK I

Teil 3:
       WQQM       BMHTT YFBHK YYXJK IXKDF RTSHB HLUEJ MFLAC
ZRJDL CJZVK HFBYL GFSEW NRSGS KHLFW JKLLZ TFMWD QDQQV JUTJS
VPRDE MUVPM BPBXX USOPG IVHFC ISGPY IYKST VQUIO CAVCW AKEQQ
EFRVM XSLQC FPFTF SPIIU ENLUW O
```

Entschlüsselt liefert Teil 1

```
AUFBEFEHLDESOBERSTENBEFEHLSHABERSSINDIMFALLEXZXZTXUNWAHRSCHEINLICHENXFRANZOESISQ
ENANGRIFFSDIEWESTBEFESTIGUNGENJEDERZAHLENMAESSIGENUEBERLEGENHEITZUMTROTZZUHALTENX
```

Etwas gegliedert:

```
AUF BEFEHL DES OBERSTEN BEFEHLSHABERS SIND IM FALLE XZXZTX
UNWAHRSCHEINLICHEN X FRANZOESISQEN ANGRIFFS DIE WESTBEFESTIGUNGEN
JEDER ZAHLENMAESSIGEN UEBERLEGENHEIT ZUM TROTZ ZUHALTENX
```

2. Teil:

```
FUEHRUNGUNDTRUPPEMUESSENVONDIESEREHRENPFLIQTDURQDRUNGENSEINXABSXDEMGEMAESSBEHALT
EIQMIRDIEERMAEQTIGUNGZURPUFGABEDERBEFESTIGUNGENODERAUQVONTEILENAUSDRUECKLIQ
```

Gegliedert:

```
FUEHRUNG UND TRUPPE MUESSEN VON DIESER EHRENPFLIQT DURQDRUNGEN SEIN X
ABS X DEMGEMAESS BEHALTE IQ MIR DIE ERMAEQTIGUNG ZUR PUFGABE DER
BEFESTIGUNGEN ODER AUQ VON TEILEN AUSDRUECKLIQ
```

3. Teil:

```
PERSOENLIQVORXABSXAENDERUNGDERANWEISUNGXOKHXGENXSTXDXHXERSTEABTXNRXDREIDREIZWOEI
NSXDREIAQTGXKDOSXVOMJULIEINSNEUNDREIAQTBLEIBTVORBEHALTENXDEROBERBEFEHLSHABERDESH
EERES
```

Gegliedert:

```
PERSOENLIQ VOR X ABS X AENDERUNG DER ANWEISUNG X OKH X GEN X ST X D X
H X ERSTE ABT X NR X DREI DREI ZWO EINS X DREI AQT G X K DOS X VOM
JULI EINS NEUN DREI AQT BLEIBT VORBEHALTEN X
DER OBERBEFEHLSHABER DES HEERES
```

12. Anhang

Literatur:

- F. L. Bauer, Entzifferte Geheimnisse, Springer-Verlag 1997.
- A. Carlson, Simulating the Enigma Cipher Machine, http://homepages.tesco.net/~andycarlson/enigma/simulating_enigma.html.
- F. Weierud, Authentic German Army Enigma Decrypt, http://home.cern.ch/~frode/crypto/tbombe.html.