a) Bitfolgen als Vektoren

Mit einem einzigen Bit läßt sich nicht viel Information darstellen und verarbeiten; interessant wird es erst mit Bitfolgen. Natürlich können wir Folgen von N Bits als Elemente des Vektorraums $\mathbb{F}_2^N$ betrachten. Da im Körper $\mathbb{F}_2$ die Summen $0+0$ und $1+1$ beide gleich 0 sind, hat dieser Vektorraum die Eigenschaft

$$\vec{v}+\vec{v}=\vec{0} \quad \text{für alle } \vec{v}\in\mathbb{F}_2^N ,$$

jeder Vektor ist also zu sich selbst invers, und genau wie auch in $\mathbb{F}_2$ gibt es keinen Unterschied zwischen plus und minus.

Der Vektorraum $\mathbb{F}_2^N$ hat eine sehr einfache Struktur: Die Vektoraddition ist in jeder Komponente einfach die logische Antivalenz, und bitweise logische Antivalenz für ganze Wörter gehört zu den Grundbefehlen der meisten Prozessoren und auch Programmiersprachen. Bei einer Maschine mit 32 Bit-Prozessor läßt sich also eine Vektoraddition in $\mathbb{F}_2^{32}$ mit einem einzigen Befehl ausführen; in C oder C++ wäre der entsprechende Ausdruck gleich `a∧b`.

Noch einfacher ist die Multiplikation mit einem Skalar, denn es gibt nur zwei Skalare: Multiplikation mit Eins ändert nichts, Multiplikation mit Null hat immer die Bitfolge aus lauter Nullen als Ergebnis.

Das Rechnen in $\mathbb{F}_2^N$ ist also sehr einfach und effizient, und es kann schon in dieser ganz trivialen Form auch nützlich sein:

Eine Anwendung ist etwa die Fehlererkennung in der Informationsübertragung: Dazu werden Daten beispielsweise oft zusammen mit einem „Paritätsbit“ übertragen, d.h. jede Folge von sieben Bits wird um ein achtes „Prüfbit“ erweitert, so daß im entstehenden Byte immer eine gerade Anzahl von Einsen vorkommt; es hat also gerade Parität. Vor der Übertragung wird also auf jede Folge von sieben Bit die lineare Abbildung

$$\varphi\colon \begin{cases} \mathbb{F}_2^7 \to \mathbb{F}_2^8 \\ (x_1,\dots,x_7)\mapsto(x_1,\dots,x_7,\ x_1+\cdots+x_7) \end{cases}$$

angewendet. Auch die Überprüfung, ob ein gegebenes Byte tatsächlich gerade Parität hat, läßt sich mit einer linearen Abbildung realisieren:

Die Bytes mit gerader Parität sind offenbar gerade die aus dem Kern der linearen Abbildung

$$\psi\colon \begin{cases} \mathbb{F}_2^8 \to \mathbb{F}_2 \\ (x_1,\dots,x_8)\mapsto(x_1+\cdots+x_8) \end{cases}$$

Mit etwas mehr Aufwand kann man Fehler nicht nur erkennen, sondern auch korrigieren: Als Beispiel dafür konstruieren wir eine Abbildung

$$\varphi\colon \mathbb{F}_2^{nm} \to \mathbb{F}_2^{(n+1)(m+1)}$$

wie folgt: Wir schreiben die Elemente von $\mathbb{F}_2^{nm}$ in der Form

$$X=\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n}\\ x_{21} & x_{22} & \dots & x_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ x_{m1} & x_{m2} & \dots & x_{mn}\end{pmatrix}$$

und bilden ein solches Element ab auf

$$\varphi(X)=\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} & x_{1,n+1}\\ x_{21} & x_{22} & \dots & x_{2n} & x_{2,n+1}\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ x_{m1} & x_{m2} & \dots & x_{mn} & x_{m,n+1}\\ x_{m+1,1} & x_{m+1,2} & \dots & x_{m+1,n} & x_{m+1,n+1}\end{pmatrix},$$

wobei

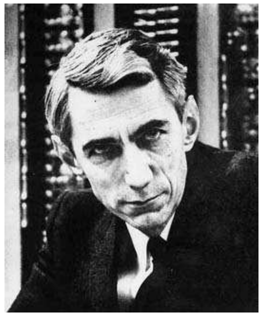
$$x_{i,n+1}=\sum_{j=1}^n x_{ij} \quad\text{und}\quad x_{m+1,j}=\sum_{i=1}^m x_{ij}$$

sein soll. Es braucht uns dabei nicht stören, daß $x_{n+1,m+1}$ hier auf zwei verschiedene Weisen definiert ist: Wie man sich leicht überlegt, führen beide Definitionen ausgeschrieben zu

$$x_{n+1,m+1}=\sum_{i=1}^n\sum_{j=1}^m x_{ij}\,.$$

Hier gibt es also $n+m+1$ Prüfbits; in $\varphi(X)$ sind alle Zeilensummen und alle Spaltensummen Null. Falls nun durch einen Übertragungsfehler das Bit x_{ij} (und sonst keines) verfälscht wurde, ist genau in der i-ten Zeile und der j-ten Spalte die entsprechende Summe gleich eins, es ist also klar, daß x_{ij} korrigiert werden muß.

Mit entsprechend größerem Aufwand lassen sich auch mehr Fehler korrigieren; tatsächlich können nach zwei Sätzen von SHANNON, wenn man nur genügend lange Codewörter zuläßt, mit beliebig geringem (relativem) Aufwand beliebig hohe (vorgegebene) Fehlerraten korrigiert werden – vorausgesetzt natürlich, diese Raten sind echt kleiner als $1/2$. Bei einer Fehlerrate von $1/2$ kommen nur Zufallsbits ohne jeglichen Informationsgehalt an.



CLAUDE ELWOOD SHANNON (1916–2001) wurde in Petoskey im US-Bundesstaat Michigan geboren; 1936 verließ er die University of Michigan mit sowohl einem Bachelor der Mathematik als auch einem Bachelor der Elektrotechnik, um am M.I.T. weiterzustudieren. Seine 1938 geschriebene Diplomarbeit *A symbolic analysis of relay and switching circuits* bildet die Grundlage der digitalen Informationsverarbeitung auf der Grundlage der hier entwickelten Schaltlogik; seine Dissertation 1940 befaßte sich mit Anwendungen der Algebra auf die MENDELschen Gesetze. Danach arbeitete er bis 1956 bei den Bell Labs, wo er während des zweiten Weltkriegs insbesondere über die Sicherheit kryptographischer Systeme forschte. Seine *Mathematical theory of cryptography* wurde aus Geheimhaltungsgründen erst 1949 zur Veröffentlichung freigegeben. Seine wohl bekannteste Arbeit ist die 1948 erschienene *Mathematical theory of communication,* in der er die fehlerfreie Übertragung von Nachrichten über einen gestörten Kanal untersuchte. Von 1956 bis zu seiner Emeritierung 1978 lehrte er am M.I.T., das er dadurch zur führenden Universität auf dem Gebiet der Informationstheorie und Kommunikationstechnik machte. Zu seinen zahlreichen Arbeiten zählt auch eine über die mathematische Theorie der Jongliermuster, anhand derer Jongleure eine Reihe neuer Muster gefunden haben; auch konstruierte er mehrere Jonglierroboter.

Beim nächsten Beispiel geht es um die Sicherung von Information gegen *absichtliche* Manipulation und unberechtigtes Mithören:

Während des kalten Kriegs hielten viele (wohl zu Recht) die Gefahr eines Atomkriegs aus Versehen für erheblich größer als die eines absichtlichen Atomkriegs. Um ersteren weniger wahrscheinlich zu machen, einigten sich die beiden Großmächte im Juni 1963 in Genf darauf, das sogenannte *Rote Telephon* einzurichten; es funktioniert seit dem 30. August 1963.

Natürlich handelt es sich dabei nicht wirklich um ein Telephon, denn zu keinem Zeitpunkt des kalten Krieges reichten die Sprachkenntnisse eines amerikanischen Präsidenten oder eines Generalsekretärs der KPdSU auch nur für ein direktes Gespräch über das Wetter.

Tatsächlich war das *Rote Telephon* eine Fernschreibverbindung mit je vier Fernschreibern (geliefert von Siemens Mannheim) an beiden Enden: jeweils zwei mit lateinischem und zwei mit kyrillischem Alphabet. Bislang verbrachten sie ihre meiste Zeit damit, stündliche Testnachrichten zu drucken wie amerikanische Baseball-Ergebnisse oder TURGENJEWS *Aufzeichnungen einer Jägers.*

Aus Sicherheitsgründen wurden zwei Leitungen eingerichtet, eine entlang der Route Washington-London-Kopenhagen-Stockholm-Helsinki-Moskau, die andere via Tanger. Natürlich war es unmöglich, diese Leitungen auf ihrer ganzen Länge zu überwachen, so daß niemand ausschließen konnte, daß irgendwo zwischen Moskau und Washington eine vertrauliche Kommunikation abgehört oder – schlimmer noch – eine gefälschte Nachricht eingespielt wurde.

Zum Schutz davor wurde die gesamte Kommunikation verschlüsselt. Wegen der hohen Sicherheitsanforderungen konnte dazu allerdings keines der üblicherweise in heutiger Office-Software eingebauten Verfahren verwendet werden: Wer noch irgendwelche Illusionen über die Sicherheit gängiger kommerzieller Programme hat, sollte unter

`http://pwcrack.com`

nachlesen, für welche vergleichsweise bescheidenen Beträge spezialisierte Unternehmen dazu bereit sind, „vergessene“ Paßwörter zu rekonstruieren.

Das *Rote Telephon* benutzte stattdessen eine Variante eines alten, absolut sicheren, Verschlüsselungsverfahrens, des sogenannten *one time pads:* Von Zeit zu Zeit tauschten die beiden Seiten per Kurier Magnetbänder mit zufallserzeugten Bitfolgen aus. Jedesmal, wenn eine Nachricht übermittelt werden sollte, übersetzte der Fernschreiber diese in eine Bitfolge, d.h. in einen Vektor $\vec{v}$ aus einem Vektorraum $\mathbb{F}_2^N$. Aus den ersten N bislang noch nicht benutzten Bits auf dem Magnetband wurde dazu ein weiterer Vektor $\vec{w} \in \mathbb{F}_2^N$ gebildet, und tatsächlich übertragen wurde die Summe $\vec{s} = \vec{v} + \vec{w}$.

Am anderen Ende der Leitung, wo eine Kopie des Magnetbands vorlag, war $\vec{w}$ bekannt, so daß die Nachricht

$$\vec{v} = \vec{v} + \vec{0} = \vec{v} + (\vec{w} + \vec{w}) = (\vec{v} + \vec{w}) + \vec{w} = \vec{s} + \vec{w}$$

rekonstruiert werden konnte.

Ein Lauscher ohne Magnetband konnte nur die Länge N der Nachricht ermitteln, was bei den seitenlangen in Diplomatensprache formulierten Texten, die über diese Leitung liefen, so gut wie keine konkrete Information lieferte.

Betrachten wir als Beispiel einen Text, der zwar wohl nie über das rote Telephon geschickt wurde, der aber in Büchern über Kryptographie oft als Beispiel verwendet wird: *Angriff im Morgengrauen!*

Um einen Vektor über $\mathbb{F}_2$ zu bekommen, betrachten wir die Buchstaben als ASCII-Zeichen und bekommen den Vektor

01000001 01101110 01100111 01110010 01101001 01100110 01100110 00100000
01101001 01101101 00100000 01001101 01101111 01110010 01100111 01100101
01101110 01100111 01110010 01100001 01110101 01100101 01101110 00100001

aus $\mathbb{F}_2^{192}$. Als Schlüssel verwenden wir eine (möglichst wirklich) zufällige Folge aus ebenfalls 192 Bit, z.B.

10110010 11111001 01110001 11001011 01010011 10100011 11101111 11110011
11010010 11011010 01100010 10111111 01011011 01100001 10000110 10110000
01010100 10000101 11100010 00111000 10111011 11111111 11000100 10010111 ,

die hinreichend lange vorher auch dem Empfänger bekannt gemacht wurde.

Die Summe der beiden Vektoren ist

11110011 10010111 00010110 10111001 00111010 11000101 10001001 11010011
10111011 10110111 01000010 11110010 00110100 00010011 11100001 11010101
00111010 11100010 10010000 01011001 11001110 10011010 10101010 10110110 ,

und diese Bitfolge wird übertragen.

Der Empfänger addiert dazu den ihm bekannten Schlüssel, und kommt wieder auf die ursprüngliche Bitfolge, die nach ASCII-Standard *Angriff im Morgengrauen!* bedeutet.

Wer den Schlüssel nicht kennt, aber errät, kommt natürlich auf dieselbe Entschlüsselung. Allerdings weiß er nicht, ob er richtig geraden hat, und versuchsweise Entschlüsselung mit einem anderen Schlüssel kann zu genauso wahrscheinlichen anderen Nachrichten führen: Mit

10110001 11100101 01111111 11010111 01011101 10100000 10101001 10010001
11010111 11000010 00101111 10010111 01011010 00110011 10000111 00101001
01001000 11000010 11011101 00101100 10111010 11101110 11000011 10010111

etwa erhält man die Entschlüsselung

01000010 01110010 01101001 01101110 01100111 01100101 00100000 01000010
01101100 01110101 01101101 01100101 01101110 00100000 01100110 11111100
01110010 00100000 01001101 01110101 01110100 01110100 01101001 00100001 ,

entsprechend dem Klartext *Bringe Blumen für Mutti!*

Entsprechend gibt es auch zu jedem anderen Text der Länge 24, egal ob sinnvoll oder nicht, einen Schlüssel, der auf genau diesen Text führt; der Lauscher erhält also definitiv keine Information außer der Länge der Nachricht.

Auch jemand, der einen Vektor $\vec{s}$ in die Leitung einspielt, hat so gut wie keine Chance, daß nach Addition von $\vec{w}$ daraus verständlicher Text wird; die Manipulation wird daher mit an Sicherheit grenzender Wahrscheinlichkeit entdeckt.

Kommunikation unter dem Schutz des *one time pad* ist also sehr sicher, aber leider auch sehr aufwendig: Wer einfach ein Buch im Internet bestellen will, hat üblicherweise keine Möglichkeit, über Kurier ein Magnetband oder eine CD-ROM mit dem Versandhaus auszutauschen, bevor er seine Kontendaten dorthin schickt. Für Alltagsanwendungen braucht man daher Verfahren, die einfacher anwendbar sind. Leider sind die wirklich guten darunter mathematisch deutlich anspruchsvoller als der *one time pad;* zwei davon werden wir im Laufe dieses Paragraphen noch kennenlernen.

b) Körper von Primzahlordnung

Der Körper mit zwei Elementen ist nur einen von vielen endlichen Körpern; beispielsweise gibt es zu jeder Primzahl p einen solchen Körper; wir bezeichnen ihn in Analogie zu $\mathbb{F}_2$ mit $\mathbb{F}_p$.

Als Menge ist $\mathbb{F}_p \underset{\text{def}}{=} \{0, 1, \dots, p-1\}$; Addition und Multiplikation werden definiert durch die Vorschriften

$$a \oplus b \underset{\text{def}}{=} (a+b) \bmod p \quad \text{und} \quad a \odot b \underset{\text{def}}{=} ab \bmod p\,,$$

d.h. für führen zunächst die entsprechenden Operationen für ganze Zahlen aus und betrachten dann den Divisionsrest des Ergebnisses bei Division durch die Primzahl p. Als Divisionsrest einer ganzen Zahl x modulo einer natürlichen Zahl y bezeichnen wir dabei jeweils die natürliche Zahl r zwischen 0 und $p-1$, für die es eine qanze Zahl q gibt, so daß $x = qy + r$ ist, genau wie man es (für natürliche Zahlen x) in der Grundschule gelernt hat.

Für $p = 2$ gibt es nur die beiden Divisionsreste 0 und 1, und die obigen Definitionen führen auf die inzwischen wohlbekannten Rechenoperationen von $\mathbb{F}_2$.

Da das Kommutativ- und das Assoziativgesetz für Addition und Multiplikation ganzer Zahlen gelten und zwei gleiche Zahlen insbesondere den gleichen Divisionsrest modulo p haben, gelten diese Gesetze auch für $\oplus$ und $\odot$; aus demselben Grund gilt auch das Distributivgesetz, und 0 und 1 sind Neutralelemente bezüglich $\oplus$ und $\odot$.

Das zu a inverse Element bezüglich der Addition ist für $a = 0$ natürlich a selbst, ansonsten $p - a$, denn

$$a \oplus (p-a) = a + (p-a) \bmod p = p \bmod p = 0\,.$$

Multiplikative Inverse lassen sich nicht so leicht finden, aber immerhin läßt sich relativ einfach sehen, daß die existieren: Für $a \in \mathbb{F}_p \setminus \{0\}$ betrachten wir die sämtlichen Elemente $a \cdot x$ mit $x \in \mathbb{F}_p$. Ist $a \cdot x = a \cdot y$, so ist $a \cdot (x-y) = 0$, d.h. $a(x-y)$ ist durch p teilbar. Da p eine Primzahl ist (das verwenden wir hier zum ersten Mal!), muß dann auch mindestens einer der beiden Faktoren durch p teilbar sein. a aber ist eine Zahl zwischen 1 und $p-1$, also sicherlich nicht durch p teilbar. Somit teilt p die Differenz $x - y$. Da x, y als Elemente von $\mathbb{F}_p$ höchstens gleich $p-1$ sind, hat ihre Differenz höchstens Betrag $p-1$, also kann sie nur dann durch p teilbar sein, wenn sie verschwindet. Somit gilt: Für $x, y \in \mathbb{F}_p$ ist $a \cdot x = a \cdot y$ genau dann, wenn $x = y$. Die p Elemente $a \cdot x$ sind somit

allesamt verschieden; da es in $\mathbb{F}_p$ nur p Elemente gibt, läßt sich also jedes von diesen in der Form $a \cdot x$ mit einem geeigneten $x \in \mathbb{F}_p$ schreiben. Dies gilt insbesondere für die Eins, und damit ist auch die Existenz multiplikativer Inverser als letztes des Körpereaxiome nachgewiesen.

$\mathbb{F}_p$ ist also in der Tat ein Körper.

Der Aufwand für das Rechnen in diesem Körper ist am geringsten für die Addition: Da $a + b$ für $a, b \in \mathbb{F}_p$ zwischen Null und $2p - 2$ liegt, ist

$$a \oplus b = \begin{cases} a+b & \text{falls } a+b < p \\ a+b-p & \text{sonst} \end{cases}\,,$$

hier braucht man also keine Division, und dasselbe gilt natürlich auch für die Subtraktion:

$$a \ominus b = \begin{cases} a-b & \text{falls } a-b \geq 0 \\ a-b+p & \text{sonst} \end{cases}\,.$$

Zur Berechnung des Produkts zweier Elemente von $\mathbb{F}_p$ brauchen wir eine Multiplikation ganzer Zahlen und eine Division mit Rest; hier ist also der Aufwand deutlich höher.

Am aufwendigsten ist die Division in $\mathbb{F}_p$: Bislang können wir nur durch a dividieren, indem wir alle Produkte von a mit Elementen aus $\mathbb{F}_p$ systematisch durchprobieren. Da p bei einigen kryptographischen Anwendungen durchaus mehrere hundert Dezimalstellen haben kann, brauchen wir dringend eine effizientere Alternative; diese wird uns der EUKLIDische Algorithmus liefern.

Im folgenden werde ich, wenn keine Verwechslungsgefahr mit ganzen Zahlen besteht, die Rechenoperationen in $\mathbb{F}_p$ meist einfach mit $+$ und $\cdot$ anstelle von $\oplus$ und $\odot$ bezeichnen.

c) Der Euklidische Algorithmus

Beginnen wir mit dem einfachsten Fall, für den der Algorithmus schon als Proposition zwei im siebten Buch der Elemente EUKLIDs zu finden ist: Wir suchen den größten gemeinsamen Teiler zweier nichtnegativer ganzer Zahlen a und b, d.h. die größte ganze Zahl d, die sowohl a als auch b teilt. Für $a = b = 0$ gibt es kein *größtes* solches d; hier setzen wir $d = 0$. Wir schreiben kurz $d = \mathrm{ggT}(a, b)$.

Grundidee des EUKLIDischen Algorithmus ist die Anwendung der Division mit Rest: Für je zwei natürliche Zahlen x und y gibt es nichtnegative ganze Zahlen q und r, so daß

$$x = qy + r \quad \text{und} \quad 0 \le r < y$$

ist. Alsdann ist

$$\mathrm{ggT}(x, y) = \mathrm{ggT}(y, r) ,$$

denn wegen der beiden Gleichungen

$$x = qy + r \quad \text{und} \quad r = x - qy$$

teilt jeder gemeinsame Teiler von x und y auch r, und jeder gemeinsame Teiler von y und r teilt auch x. Da außerdem offensichtlich

$$\mathrm{ggT}(x, 0) = x \quad \text{für alle } x \in \mathbb{N}_0$$

ist, können wir den ggT leicht rekursiv berechnens, indem wir die Regel $\mathrm{ggT}(x, y) = \mathrm{ggT}(y, r)$ so lange anwenden, bis $r = 0$ und damit der ggT gleich y ist. Wer Scheme oder einen anderen LISP-Dialekt kennt, kann den Algorithmus damit kurz und knapp als Einzeiler formulieren:

```
(define (ggT x y) (if (= y 0) x (ggT y (remainder x y))))
```

In mathematischer Sprechweise bedeutet das:

Schritt 0: Setze $r_0 = x$ und $r_1 = y$

Schritt i, $i \ge 1$**:** Falls $r_i = 0$ ist, endet der Algorithmus mit dem Ergebnis

$$\mathrm{ggT}(x, y) = r_{i-1} ;$$

andernfalls dividiere man r_{i-1} mit Rest durch r_i und bezeichne den Divisionsrest mit r_{i+1}.

Der Algorithmus bricht ab, da r_i (*bzw.* das zweite Argument y in der Scheme-Formulierung) in jedem Rekursionsschritt kleiner wird, aber stets eine nichtnegative ganze Zahl ist; nach endlich vielen Schritten

muß es also null sein, und der Algorithmus bricht ab. Die Korrektheit des Ergebnisses ist auch klar, denn aus der Gleichung

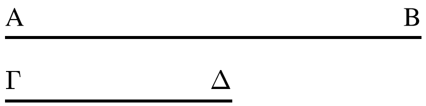
$$\mathrm{ggT}(x, y) = \mathrm{ggT}(y, x \bmod y)$$

folgt, daß stets Schritt $\mathrm{ggT}(r_{i-1}, r_i) = \mathrm{ggT}(x, y)$ ist.

Zum Vergleich sei hier noch EUKLIDs Beschreibung seines (wahrscheinlich schon mindestens 150 Jahre früher bereits den Pythagoräern bekannten) Algorithmus angegeben. In Proposition 2 des siebten Buchs seiner Elemente steht:

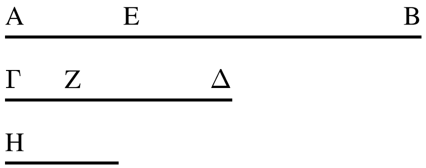
Zu zwei gegebenen Zahlen, die nicht prim gegeneinander sind, ihr größtes gemeinsames Maß zu finden.

Die zwei gegebenen Zahlen, die nicht prim, gegeneinander sind, seien AB, ΓΔ. Man soll das größte gemeinsame Maß von AB, ΓΔ finden.



Wenn ΓΔ hier AB mißt – sich selbst mißt es auch – dann ist ΓΔ gemeinsames Maß von ΓΔ, AB. Und es ist klar, daß es auch das größte ist, denn keine Zahl größer ΓΔ kann ΓΔ messen.

Wenn ΓΔ aber AB nicht mißt, und man nimmt bei AB, ΓΔ abwechselnd immer das kleinere vom größeren weg, dann muß (schließlich) eine Zahl übrig bleiben, die die vorangehende mißt. Die Einheit kann nämlich nicht übrig bleiben; sonst müßten AB, ΓΔ gegeneinander prim sein, gegen die Voraussetzung. Also muß eine Zahl übrigbleiben, die die vorangehende mißt. ΓΔ lasse, indem es BE mißt, EA, kleiner als sich selbst übrig; und EA lasse, indem es ΔZ mißt, ZΓ, kleiner als sich selbst übrig; und ΓZ messe AE.



Da ΓZ AE mißt und AE ΔZ, muß ΓZ auch ΔZ messen; es mißt aber auch sich selbst, muß also auch das Ganze ΓΔ messen. ΓΔ mißt aber BE; also mißt ΓZ auch BE; es mißt aber auch EA, muß also auch das Ganze BA messen. Und es mißt auch ΓΔ; ΓZ mißt also AB und ΓΔ; also ist ΓZ gemeinsames Maß von AB, ΓΔ. Ich behaupte, daß es auch das größte ist. Wäre nämlich ΓZ nicht das größte gemeinsame Maß von AB, ΓΔ, so müßte irgendeine Zahl größer ΓZ die Zahlen AB und ΓΔ messen. Dies geschehe; die Zahl sei H. Da H dann ΓΔ mäße und ΓΔ BE mißt, mäße H auch BE; es soll aber auch das Ganze BA messen, müßte also auch den Rest AE messen. AE mißt aber ΔZ; also müßte H auch ΔZ messen; es soll aber auch das Ganze ΔΓ messen, müßte also auch den Rest ΓZ messen, als größere Zahl die kleinere; dies ist unmöglich. Also kann keine Zahl größer ΓZ die Zahlen AB und ΓΔ messen; ΓZ ist also das größte gemeinsame Maß von AB, ΓΔ; dies hatte man beweisen sollen.

Es ist nicht ganz sicher, ob EUKLID wirklich gelebt hat; das nebenstehende Bild aus dem 18. Jahrhundert ist mit Sicherheit reine Phantasie. EUKLID ist vor allem bekannt als Autor der *Elemente,* in denen er die Geometrie seiner Zeit systematisch darstellte und (in gewisser Weise) auf wenige Definitionen sowie die berühmten fünf Postulate zurückführte. Diese Elemente entstanden um 300 v. Chr. und waren zwar nicht der erste, aber doch der erfolgreichste Versuch einer solchen Zusammenfassung. EUKLID arbeitete wohl am Museion in Alexandrien; außer den Elementen schrieb er auch ein Buch über Optik und weitere, teilweise verschollene Bücher.

Bislang ist noch nicht zu sehen, wie uns der EUKLIDische Algorithmus bei der Division in einem Körper $\mathbb{F}_p$ helfen kann. Dies leistet erst eine darauf beruhende und meist nach dem französischen Mathematiker ETIENNE BÉZOUT (1730–1783) benannte Identität, die dieser in 1766 in einem Lehrbuch beschrieb (und auf Polynome verallgemeinerte). Für Zahlen ist diese Erweiterung jedoch bereits 1624 zu finden in der zweiten Auflage des Buchs *Problèmes plaisants et délectables qui se fonts par les nombres* von BACHET DE MÉZIRIAC. Heute redet man meistens einfach vom erweiterten EUKLIDischen Algorithmus, obwohl es keinerlei Anhaltspunkte gibt, daß sich EUKLID je damit beschäftigte.

CLAUDE GASPAR BACHET SIEUR DE MÉZIRIAC (1581-1638) verbrachte den größten Teil seines Lebens in seinem Geburtsort Bourg-en-Bresse. Er studierte zwar bei den Jesuiten in Lyon und Milano und trat 1601 in den Orden ein, trat aber bereits 1602 wegen Krankheit wieder aus und kehrte nach Bourg zurück. Sein Buch erschien erstmalig 1612, zuletzt 1959. Am bekanntesten ist BACHET für seine lateinische Übersetzung der *Arithmetika* von DIOPHANTOS. In einem Exemplar davon schrieb FERMAT seine Vermutung an den Rand. Auch Gedichte von BACHET sind erhalten. 1635 wurde er Mitglied der französischen Akademie der Wissenschaften.

ETIENNE BÉZOUT (1730-1783) wurde in Nemours in der Ile-de-France geboren, wo seine Vorfahren Magistrate waren. Er ging stattdessen an die Akademie der Wissenschaften; seine Hauptbeschäftigung war die Zusammenstellung von Lehrbüchern für die Militärausbildung. Im 1766 erschienenen dritten Band (von vier) seines *Cours de Mathématiques à l'usage des Gardes du Pavillon et de la Marine* ist die Identität von BÉZOUT dargestellt. Seine Bücher waren so erfolgreich, daß sie ins Englische übersetzt und z.B. in Harvard als Lehrbücher benutzt wurden. Heute ist er vor allem bekannt durch seinen Beweis, daß sich zwei Kurven der Grade n und m in höchstens nm Punkten schneiden können.

Die Gleichung $u = qv + r$ zur Division mit Rest läßt sich auch umschreiben als $r = u - qv$; der Divisionsrest ist also eine ganzzahlige Linearkombination des Dividenden u und des Divisors v. Falls sich diese wiederum als Linearkombination der beiden Ausgangszahlen x und y darstellen lassen, erhalten wir eine entsprechende Darstellung für r:

$$u = ax + by \quad \text{und} \quad v = cx + dy \quad \Longrightarrow r = (a - qc)x + (b - qd)z \,.$$

Wir können also ausgehen von den Darstellungen

$$x = 1 \cdot x + 0 \cdot y \quad \text{und} \quad y = 0 \cdot x + 1 \cdot y \,,$$

bei jeder Division im EUKLIDischen Algorithmus den Divisionsrest als ganzzahlige Linearkombination von x und y darstellen und damit auch den ggT als den letzten nichtverschwindenen solchen Rest.

Dies führt zu folgenden Algorithmus:

Schritt 0: Setze $r_0 = a$, $r_1 = b$, $\alpha_0 = \beta_1 = 1$ und $\alpha_1 = \beta_0 = 0$. Mit $i = 1$ ist dann

$$r_{i-1} = \alpha_{i-1}a + \beta_{i-1}b \quad \text{und} \quad r_i = \alpha_i a + \beta_i b\,.$$

Diese Relationen bleiben in jedem der folgenden Schritte erhalten:

Schritt i, $i \geq 1$**:** Falls $r_i = 0$ ist, endet der Algorithmus mit

$$\mathrm{ggT}(a,b) = r_{i-1} = \alpha_{i-1}a + \beta_{i-1}b\,.$$

Andernfalls dividiere man r_{i-1} mit Rest durch r_i mit dem Ergebnis

$$r_{i-1} = q_i r_i + r_{i+1}\,.$$

Dann ist

$$\begin{aligned} r_{i+1} = -q_i r_i + r_{i-1} &= -q_i(\alpha_i a + \beta_i b) + (\alpha_{i-1}a + \beta_{i-1}b) \\ &= (\alpha_{i-1} - q_i\alpha_i)a + (\beta_{i-1} - q_i\beta_i)b\,; \end{aligned}$$

man setze also

$$\alpha_{i+1} = \alpha_{i-1} - q_i\alpha_i \quad \text{und} \quad \beta_{i+1} = \beta_{i-1} - q_i\beta_i\,.$$

Genau wie oben folgt, daß der Algorithmus für alle natürlichen Zahlen a und b endet und daß am Ende der richtige ggT berechnet wird; außerdem sind die α_i und β_i so definiert, daß in jedem Schritt $r_i = \alpha_i a + \beta_i b$ ist, insbesondere ist also im letzten Schritt der ggT als Linearkombination der Ausgangszahlen dargestellt.

Als Beispiel wollen wir den ggT von 200 und 148 als Linearkombination darstellen. Im nullten Schritt haben wir 200 und 148 als die trivialen Linearkombinationen

$$200 = 1\cdot 200 + 0\cdot 148 \quad \text{und} \quad 148 = 0\cdot 200 + 1\cdot 148\,.$$

Im ersten Schritt dividieren wir, da 148 nicht verschwindet, 200 mit Rest durch 148:

$$200 = 1\cdot 148 + 52 \Longrightarrow 52 = 1\cdot 200 - 1\cdot 148$$

Da auch $52 \neq 0$, dividieren wir im zweiten Schritt 148 durch 52 mit Ergebnis $148 = 2\cdot 52 + 44$, d.h.

$$44 = 148 - 2\cdot(1\cdot 200 - 1\cdot 148) = 3\cdot 148 - 2\cdot 200$$

Auch $44 \neq 0$, wir dividieren also weiter: $52 = 1\cdot 44 + 8$ und

$$\begin{aligned} 8 = 52 - 44 &= (1\cdot 200 - 1\cdot 148) - (3\cdot 148 - 2\cdot 200) \\ &= 3\cdot 200 - 4\cdot 148\,. \end{aligned}$$

Im nächsten Schritt erhalten wir $44 = 5\cdot 8 + 4$ und

$$\begin{aligned} 4 = 44 - 5\cdot 8 &= (3\cdot 148 - 2\cdot 200) - 5\cdot(3\cdot 200 - 4\cdot 148) \\ &= 23\cdot 148 - 17\cdot 200\,. \end{aligned}$$

Bei der Division von acht durch vier schließlich erhalten wir Divisionsrest Null; damit ist vier der ggT von 148 und 200 und kann in der angegebenen Weise linear kombiniert werden.

Der erweiterte EUKLIDische Algorithmus kann auch zur Lösung linearer diophantischer Gleichungen verwendet werden: Angenommen wir suchen ganzzahlige Lösungen (x, y) der linearen Gleichung

$$ax + by = c \quad \text{mit} \quad a, b, c \in \mathbb{Z}\,.$$

Da die linke Seite für alle x, y ein Vielfaches des ggT von a und b ist, kann es offensichtlich nur dann Lösungen geben, wenn $\mathrm{ggT}(a,b)$ ein Teiler von c ist. Falls dies gilt, können wir aus der linearen Darstellung

$$\mathrm{ggT}(a,b) = \alpha a + \beta b$$

durch Multiplikation mit $c/\,\mathrm{ggT}(a,b)$ eine lineare Darstellung

$$c = xa + yb$$

konstruieren, also eine Lösung der Gleichung.

Dies ist allerdings nicht die einzige Lösung: Wegen $ba - ab = 0$ ist offensichtlich auch $(x + b, y - a)$ eine, und ähnlich lassen sich leicht noch weitere Lösungen angeben.

Angenommen, (x', y') sei *irgendeine* andere Lösung. Dann ist

$$ax + by = ax' + by' = c\,, \quad \text{also} \quad a(x' - x) + b(y' - y) = c - c = 0\,.$$

Die ganzen Zahlen $u = x' - x$ und $v = y' - y$ erfüllen also die zugehörige homogene Gleichung

$$au + bv = 0\,.$$

Wenn wir den (trivialen) Fall $a = 0$ ausschließen, ist für jede ganzzahlige Lösung $(u, v) \neq (0, 0)$ dieser Gleichung

$$au = -bv \Longrightarrow \frac{u}{v} = -\frac{b}{a} = -\frac{\frac{b}{\mathrm{ggT}(a,b)}}{\frac{a}{\mathrm{ggT}(a,b)}}\,,$$

bis aufs Vorzeichen sind $\frac{u}{v}$ und $\frac{b}{a}$ also derselbe Bruch. Rechts steht dessen gekürzte Version, aus der $\frac{u}{v}$ durch Erweiterung mit einer ganzen Zahl k hervorgeht. Somit gibt es ein $k \in \mathbb{Z}$, für das

$$u = k \cdot \frac{b}{\mathrm{ggT}(a,b)} \quad \text{und} \quad v = -k \cdot \frac{a}{\mathrm{ggT}(a,b)}$$

ist. Somit kann jede Lösung der diophantischen Gleichung $ax + by = c$ in der Form

$$\left(x + \frac{kb}{\mathrm{ggT}(a,b)},\ y - \frac{ka}{\mathrm{ggT}(a,b)}\right) \quad \text{mit} \quad k \in \mathbb{Z}$$

geschrieben werden.

Auch das Problem der Division im Körper $\mathbb{F}_p$ wird durch den erweiterten EUKLIDischen Algorithmus effizient gelöst, denn suchen wir für $a \neq 0$ aus $\mathbb{F}_p$ ein Element $x \in \mathbb{F}_p$ mit $ax = c$, so ist das äquivalent zur diophantischen Gleichung $ax + yp = c$; wir müssen also einfach den erweiterten EUKLIDischen Algorithmus auf a und p anwenden, den Koeffizienten von a mit c multiplizieren und das Produkt modulo p reduzieren.

Als Beispiel wollen wir das Element 20^{-1} in $\mathbb{F}_{1009}$ berechnen. Dazu wenden wir den erweiterten EUKLIDischen Algorithmus an auf 1009 und 20:

$$\begin{aligned} 1009 : 20 = 50 \ \text{Rest}\ 9 \quad &\text{und} \quad 9 = 1 \cdot 1009 - 50 \cdot 20 \\ 20 : 9 = 2 \ \text{Rest}\ 2 \quad &\text{und} \quad 2 = 20 - 2 \cdot 2 = -2 \cdot 1009 + 101 \cdot 20 \\ 9 : 2 = 4 \ \text{Rest}\ 1 \quad &\text{und} \quad 1 = 9 - 4 \cdot 2 = 9 \cdot 1009 - 454 \cdot 20 \end{aligned}$$

Also ist $(-454) \cdot 20 \equiv 1 \bmod 1003$; das Inverse von 20 in $\mathbb{F}_{1009}$ ist somit -454 oder, besser ausgedrückt, $1009 - 454 = 555$. In der Tat ist

$$555 \cdot 20 = 11100 = 11 \cdot 1009 + 1 \equiv 1 \bmod 1009\,.$$

d) Das RSA-Verfahren

Als praktische Anwendungen der Körper $\mathbb{F}_p$ und des erweiterten EUKLIDischen Algorithmus möchte ich zwei Beispiele aus der Kryptographie betrachten.

Wir kennen bereits den absolut sicheren *one time pad;* leider können wir ihn aber nur anwenden, wenn vorher riesige Mengen an Schlüsselbits ausgetauscht wurden. Dies ist kein Problem für die diplomatischen Dienste großer Staaten, ist aber völlig unrealistisch für die meisten Fälle von Kommunikation zwischen Privatleuten. Hier ist gerade im Internet oft nicht einmal der sichere Austausch eines kurzen, für längere Zeit gültigen Schlüssels wirklich praktikabel.

Vor dem Hintergrund dieses Problems veröffentlichten 1976 MARTIN HELLMAN, damals Assistenzprofessor in Stanford, und sein Forschungsassistent WHITFIELD DIFFIE eine Arbeit mit dem Titel *New directions in cryptography* (IEEE Trans. Inform. Theory **22**, 644–654), in der sie vorschlugen, den Vorgang der *Ver*schlüsselung und den der *Ent*schlüsselung völlig voneinander zu trennen: Es sei schließlich nicht notwendig, daß der Sender einer verschlüsselten Nachricht auch in der Lage sei, diese zu *ent*schlüsseln.

Der Vorteil eines solchen Verfahrens wäre, daß jeder potentielle Empfänger nur einen einzigen Schlüssel bräuchte und dennoch sicher sein könnte, daß nur er selbst seine Post entschlüsseln kann. Der Schlüssel müßte nicht einmal geheimgehalten werden, da es ja nicht schadet, wenn jedermann Nachrichten *ver*schlüsseln kann. In einem Netzwerk mit n Teilnehmern bräuchte man also nur n Schlüssel, um es jedem Teilnehmer zu erlauben, mit jeden anderen so zu kommunizieren, und diese Schlüssel könnten sogar in einem öffentlichen Verzeichnis stehen. Bei einem symmetrischen Kryptosystem wäre der gleiche Zweck nur erreichbar mit $\frac{1}{2}n(n-1)$ Schlüsseln, die zudem noch durch ein sicheres Verfahren wie etwa ein persönliches Treffen oder durch vertrauenswürdige Boten ausgetauscht werden müßten.

DIFFIE und HELLMAN machten nur sehr vage Andeutungen, wie so ein System mit öffentlichen Schritten aussehen könnte; klar war nur, daß es mit einer sogenannten *Einwegfunktion* arbeiten mußte, d.h. mit einer

Funktion, die jedermann leicht berechnen kann, deren Umkehrfunktion aber nicht berechenbar ist.

BAILEY WHITFIELD DIFFIE wurde 1944 geboren. Erst im Alter von zehn Jahren lernte er lesen; im gleichen Jahr hielt eine Lehrerin an seiner New Yorker Grundschule einen Vortrag über Chiffren. Er ließ sich von seinem Vater alle verfügbare Literatur darüber besorgen, entschied sich dann 1961 aber doch für ein Mathematikstudium am MIT. Um einer Einberufung zu entgehen, arbeitete er nach seinem Bachelor bei Mitre; später, nachdem sein Interesse an der Kryptographie wieder erwacht war, kam er zu Martin Hellman nach Stanford, der ihn als Forschungsassistent einstellte. Seit 1991 arbeitet er als *chief security officer* bei Sun Microsystems. Seine dortige home page hat den URL `http://research.sun.com/people/diffie/` .

MARTIN HELLMAN wurde 1945 in New York geboren. Er studierte Elektrotechnik zunächst bis zum Bachelor an der dortigen Universität; für Master und Promotion studierte er in Stanford. Nach kurzen Zwischenaufenthalten am Watson Research Center der IBM und am MIT wurde er 1971 Professor an der Stanford University. Seit 1996 ist er emeritiert, gibt aber immer noch Kurse, mit denen er Schüler für mathematische Probleme interessieren will. Seine home page findet man unter `http://www-ee.stanford.edu/~hellman/` .

Mathematisch betrachtet kann es eine solche Funktion nicht geben: Da Verschlüsselungsfunktionen immer Funktionen zwischen endlichen Mengen sind und aus offensichtlichen Gründen injektiv sein müssen, kann *im Prinzip* jeder durch Probieren das Umkehrproblem lösen. Wenn man aber wüßte, daß jeder Versuch, die inverse Funktion zur Verschlüsselung zu finden, weit jenseits der Leistungsgrenze heutiger Computer liegt, wäre ein solches Verfahren trotzdem *praktisch* sicher.

Tatsächlich wäre es sogar so sicher, daß nicht einmal der legitime Empfänger seine Post lesen könnte; anwendbar wird es erst, wenn die Entschlüsselung *für genau eine Person* möglich ist auf Grund geheimer Information, über die sonst niemand verfügt. Solche Einwegfunktionen

bezeichnet man als *Einwegfunktionen mit Falltür.* DIFFIE und HELLMAN kannten kein Beispiel einer solche Funktion, und es gab unter vielen Experten große Skepsis bezüglich der Möglichkeit, je eine zu finden.

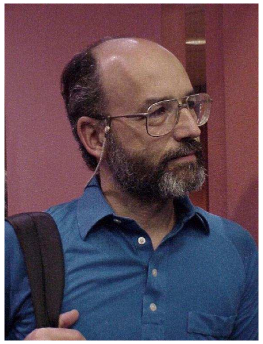
Wie inzwischen bekannt ist, gab es damals bereits Systeme, die auf solchen Funktionen beruhten; sie waren allerdings nicht in der offenen Literatur dokumentiert: Die britische *Communications-Electronics Security Group* (CESG) hatte sich bereits Ende der sechziger Jahre mit diesem Problem befaßt, um die Probleme des Militärs mit dem Schlüsselmanagement zu lösen, aufbauend auf (impraktikablen) Ansätzen von AT&T zur Sprachverschlüsselung, die während des zweiten Weltkriegs untersucht wurden. Die Briten sprachen nicht von Kryptographie mit öffentlichen Schlüsseln, sondern von *nichtgeheimer Verschlüsselung,* aber das Prinzip war das gleiche.

Erste Ideen dazu sind in einer auf Januar 1970 datierten Arbeit von JAMES H. ELLIS zu finden, ein praktikables System in einer auf den 20. November 1973 datierten Arbeit von CLIFF C. COCKS. Wie im Milieu üblich, gelangte nichts über diese Arbeiten an die Öffentlichkeit; erst 1997 veröffentlichten die *Goverment Communications Headquarters* (GCHQ), zu denen CESG gehört, einige Arbeiten aus der damaligen Zeit auf ihrer Website. Die Links sind inzwischen verschwunden, zeitweise findet man aber einige der Arbeiten trotzdem noch, wenn man auf `http://www.cesg.gov.uk/` unter *CESG Publications* direkt nach ELLIS oder COCKS sucht.

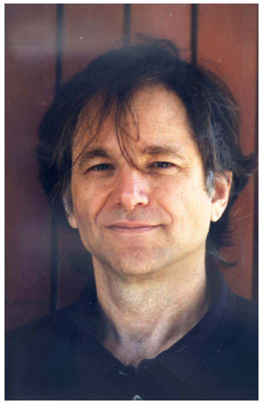
Im akademischen Bereich gab es ein Jahr nach Erscheinen der Arbeit von DIFFIE und HELLMAN das erste Kryptosystem mit öffentlichen Schlüsseln: Drei Professoren am Massachussetts Institute of Technology fanden nach rund vierzig erfolglosen Ansätzen 1977 schließlich jenes System, das heute nach ihren Anfangsbuchstaben mit RSA bezeichnet wird: RON RIVEST, ADI SHAMIR und LEN ADLEMAN.

Das System wurde 1983 von der eigens dafür gegründeten Firma RSA Computer Security Inc. patentiert und mit großem kommerziellem Erfolg vermarktet. Das Patent lief zwar im September 2000 aus, die Firma ist aber weiterhin erfolgreich im Kryptobereich tätig. Auch das RSA-Verfahren wird immer noch weithin verwendet; 2002 bekamen die drei

RONALD LINN RIVEST wurde 1947 in Schenectady im US-Bundesstaat New York geboren. Er studierte zunächst Mathematik an der Yale University, wo er 1969 seinen Bachelor bekam; danach studierte er in Stanford Informatik. Nach seiner Promotion 1974 wurde er Assistenzprofessor am Massachussetts Institute of Technology, wo er heute einen Lehrstuhl hat. Er arbeitet immer noch auf dem Gebiet der Kryptographie und entwickelte eine ganze Reihe weiterer Verfahren, auch symmetrische Verschlüsselungsalgorithmen und Hashverfahren. Er ist Koautor eines Lehrbuchs über Algorithmen. Seine home page ist `//http://theory.lcs.mit.edu/~rivest/` .



ADI SHAMIR wurde 1952 in Tel Aviv geboren. Er studierte zunächst Mathematik an der dortigen Universität; nach seinem Bachelor wechselte er ans Weizmann Institut, wo er 1975 seinen Master und 1977 die Promotion in Informatik erhielt. Nach einem Jahr als Postdoc an der Universität Warwick und drei Jahren am MIT kehrte er ans Weizmann Institut zurück, wo er bis heute Professor ist. Außer für RSA ist er bekannt sowohl für die Entwicklung weiterer Kryptoverfahren als auch für erfolgreiche Angriffe gegen Kryptoverfahren. Er schlug auch einen optischen Spezialrechner zur Faktorisierung großer Zahlen vor. Seine home page ist erreichbar unter `http://www.wisdom.weizmann.ac.il/math/profile/scientists/shamir-profile.html`



LEONARD ADLEMAN wurde 1945 in San Francisco geboren. Er studierte in Berkeley, wo er 1968 einen BS in Mathematik und 1976 einen PhD in Informatik erhielt. Thema seiner Dissertation waren zahlentheoretische Algorithmen und ihre Komplexität. Von 1976 bis 1980 war er an der mathematischen Fakultät des MIT; seit 1980 ist arbeitet er an der University of Southern California in Los Angelos. Seine Arbeiten beschäftigen sich mit Zahlentheorie, Kryptographie und Molekularbiologie. Er führte nicht nur 1994 die erste Berechnung mit einem „DNS-Computer“ durch, sondern arbeitete auch auf dem Gebiet der Aidsforschung. Heute hat er einen Lehrstuhl für Informatik und Molekularbiologie. `http://www.usc.edu/dept/molecular-science/fm-adleman.htm`

Autoren dafür den TURING-Preis der ACM, die höchste Auszeichnung der Informatik.

RSA ist übrigens identisch mit dem von COCKS vorgeschlagenen System, so daß Skeptiker auch Zweifel an den Behauptungen der GCHQ haben können. Die Beschreibung durch RIVEST, SHAMIR und ADLEMAN erschien 1978 unter dem Titel *A method for obtaining digital signatures and public-key cryptosystems* in Comm. ACM **21**, 120–126.

Grundidee ist die Verwendung einer Abbildung der Form

$$\{0, 1, \ldots, N-1\} \to \{0, 1, \ldots, N-1\}; \quad x \mapsto x^e \bmod N\,.$$

Sie ist einfach auszurechnen, für geeignetes e und N auch injektiv, und das Problem, ihre Umkehrabbildung effizient zu berechnen ist zumindest für hinreichend allgemeine große Werte von N ungelöst. Wie Abbildung 11 zeigt, sieht eine Potenzfunktion modulo N bei weitem nicht so harmlos aus wie eine Potenzfunktion in $\mathbb{R}$ – es ist zunächst nicht einmal wirklich klar, wann sie injektiv ist.

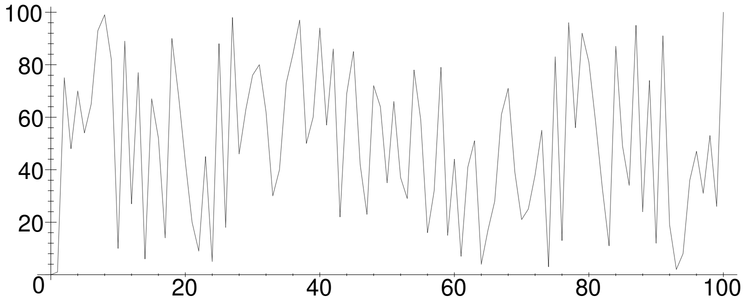


Abb. 11: Die Funktion $x \mapsto x^{17}$ in $\mathbb{F}_{101}$

Für den Fall, daß $N = p$ eine Primzahl ist und wir somit in einem der endlichen Körper $\mathbb{F}_p$ arbeiten, lassen sich die injektiven Potenzfunktionen charakterisieren und auch umkehren. Ausgangspunkt dazu ist der folgende Satz:

Kleiner Satz von Fermat: Für jedes $a \in \mathbb{Z}$ ist $a^p \equiv a \bmod p$. Ist a nicht durch p teilbar, ist auch $a^{p-1} \equiv 1 \bmod p$.

Beweis: Für zwei Zahlen $x, y \in \mathbb{Z}$ ist

$$(x+y)^p = \binom{p}{0}x^p + \binom{p}{1}x^{p-1}y + \cdots + \binom{p}{p-1}xy^{p-1} + \binom{p}{p}y^p$$

mit

$$\binom{p}{j} = \frac{p!}{j!(p-j)!}\,.$$

Für $j = 0$ und $j = p$ ist $\binom{p}{0} = \binom{p}{p} = 1$, ansonsten sind j und $p - j$ beide kleiner als p. In diesem Fall steht also im Nenner kein Faktor p, der das p aus dem Zähler wegkürzen könnte, so daß alle $\binom{p}{j}$ mit $1 \le j \le p - 1$ durch p teilbar sind. Modulo p verschwinden diese Koeffizienten, und übrig bleibt

$$(x+y)^p \equiv x^p + y^p \bmod p\,.$$

Induktiv folgt, daß eine entsprechende Gleichung auch für Summen mit mehr als zwei Summanden gilt, insbesondere auch für beliebig viele Summanden eins, d.h.

$$(\underbrace{1+\cdots+1}_{a\text{ mal}})^p \equiv \underbrace{1^p+\cdots+1^p}_{a\text{ mal}} = \underbrace{1+\cdots+1}_{a\text{ mal}} \bmod p\,.$$

Damit ist $a^p \equiv a \bmod p$ für jede natürliches Zahl a. Für $a = 0$ gilt die Beziehung auch, und da für jedes positive a

$$0 = \big(a + (-a)\big)^p = a^p + (-a)^p = a + (-a)^p \bmod p$$

ist, gilt sie auch für negative a.

Falls a nicht durch p teilbar ist, gibt es, da $\mathbb{F}_p$ ein Körper ist, ein b mit $ba \equiv 1 \bmod p$; mit diesem b ist

$$a^{p-1} \equiv (ba)a^{p-1} = ba^p \equiv ba \equiv 1 \bmod p\,.$$

■

Korollar: Falls die natürliche Zahl e keinen Teiler mit $p - 1$ gemeinsam hat, gibt es eine natürliche Zahl d, so daß

$$\left(a^e\right)^d \equiv a \bmod p$$

für alle $a \in \mathbb{Z}$. Insbesondere ist die Abbildung $x \mapsto x^e$ von $\mathbb{F}_p$ nach $\mathbb{F}_p$ dann injektiv.

Beweis: Nach dem erweiterten EUKLIDischen Algorithmus läßt sich der größte gemeinsame Teiler eins von e und $p - 1$ als ganzzahlige Linearkombination dieser beiden Zahlen schreiben, es gibt also ganze Zahlen d und r, so daß

$$d \cdot e + r \cdot (p-1) = 1$$

ist. Hier könnte d eventuell noch negativ sein, aber indem wir nötigenfalls noch ein Vielfaches der Gleichung

$$(p-1) \cdot e - e \cdot (p-1) = 0$$

dazuaddieren, können wir annehmen, daß d positiv ist (und r dann natürlich negativ). Für nicht durch p teilbares a ist dann

$$\left(a^e\right)^d = a^{de} = a^{1-r(p-1)} = a \cdot \left(a^{p-1}\right)^{-r} \equiv a \cdot 1^{-r} = a \bmod p\,,$$

wie behauptet.

Für durch p teilbares a sind beide Seiten der zu beweisenden Kongruenz durch p teilbar, so daß sie auch in diesem Fall richtig ist.

■

Der französische Mathematiker PIERRE DE FERMAT (1601–1665) wurde in Beaumont-de-Lomagne im Departement Tarn et Garonne geboren. Bekannt ist er heutzutage vor allem für seine 1994 von ANDREW WILES bewiesene Vermutung, wonach die Gleichung $x^n + y^n = z^n$ für $n \ge 3$ keine ganzzahlige Lösung mit $xyz \ne 0$ hat. Dieser „große“ Satzes von FERMAT, von dem FERMAT lediglich in einer Randnotitz behauptete, daß er ihn beweisen könne, erklärt den Namen der obigen Aussage. Obwohl FERMAT sich sein Leben lang sehr mit Mathematik beschäftigte und wesentliche Beiträge zur Zahlentheorie, Wahrscheinlichkeitstheorie und Analysis lieferte, war er hauptberuflich Jurist.

In diesem Fall gibt uns also der erweiterte EUKLIDische Algorithmus eine natürliche Zahl d, so daß die Umkehrfunktion zum Potenzieren mit e einfach das Potenzieren mit d ist. Den Exponenten d kann jeder berechnen, der p und e (und den erweiterten EUKLIDischen Algorithmus) kennt; da man p und e zum Verschlüsseln braucht, liefert das also kein Kryptosystem mit öffentlichen Schlüsseln, sondern höchstens bei ein klassisches System. (Unter dem Namen POHLIG/HELLMAN-Verfahren wird es für einige wenige Spezialfälle auch tatsächlich benutzt; beispielsweise kann man damit, so man unbedingt will, Internetvarianten von Poker oder auch anderen Kartenspielen entwickeln, bei denen keiner der Teilnehmer beim Mischen oder auch beim Ausspielen der Karten unbemerkt schummeln kann.

Das RSA-Verfahren arbeitet nicht modulo einer Primzahl p, sondern modulo dem Produkt $N = pq$ zweier Primzahlen p und q. Hier führt der kleine Satz von FERMAT zu folgenden Aussagen:

Satz: Ist $N = pq$ Produkt zweier verschiedener Primzahlen p und q, so gilt
a) Für jede ganze Zahl a ist $a^{1+(p-1)(q-1)} \equiv a \bmod N$.
b) Falls die natürliche Zahl e keinen Teiler mit $(p-1)(q-1)$ gemeinsam hat, gibt es eine natürliche Zahl d, so daß

$$\left(a^e\right)^d \equiv a \bmod p$$

für alle $a \in \mathbb{Z}$.
c) Die Berechnung von $(p-1)(q-1)$ aus N ist äquivalent zur Faktorisierung von N.

Beweis: a) Nach dem kleinen Satz von FERMAT ist für jedes nicht durch p teilbare a und jedes r

$$a^{1+r(p-1)} = a \cdot (a^{p-1})^r \equiv a \bmod p\,.$$

Für Vielfache von p sind beide Seiten durch p teilbar, so daß die Gleichung tatsächlich für alle ganzen Zahlen a gilt.

Entsprechendes gilt natürlich auch für die Primzahl q, und damit ist für jede ganze Zahl s

$$a^{1+s(p-1)(q-1)} \equiv a \bmod p \quad \text{und} \quad a^{1+(p-1)(q-1)} \equiv a \bmod q\,.$$

Diese beiden Kongruenzen besagen, daß die Differenz zwischen linker und rechter Seite sowohl durch p als auch durch q teilbar ist, also auch durch deren Produkt N, d.h.

$$a^{1+s(p-1)(q-1)} \equiv a \bmod N\,.$$

b) Geht wegen *a)* genau wie beim Primzahlfall im obigen Korollar.

c) Falls die Faktorisierung $N = pq$ bekannt ist, läßt sich natürlich leicht $(p-1)(q-1)$ berechnen. Ist umgekehrt

$$(p-1)(q-1) = pq - p - q + 1 = N + 1 - (p+q)$$

bekannt, so kennt man außer dem Produkt N auch die Summe M der beiden Primfaktoren, und diese sind die leicht berechenbaren Lösungen der quadratischen Gleichung $x^2 - Mx + N = 0$. ∎

Zur praktischen Durchführung des RSA-Verfahrens wählt man sich zwei verschiedene Primzahlen p, q, die unbedingt geheim gehalten werden müssen, und eine natürliche Zahl e, die keinen gemeinsamen Teiler mit $(p-1)(q-1)$ hat. Dann veröffentlicht man die beiden Zahlen $N = pq$ und e als öffentlichen Schlüssel.

Sodann berechnet man zu $(p-1)(q-1)$ gemäß obigem Satz nach dem EUKLIDischen Algorithmus eine Zahl d, so daß

$$\left(a^e\right)^d \equiv a \bmod N$$

für alle a; diese Zahl ist der geheime Schlüssel.

In der Praxis wird oft $e = 3$ gesetzt, um den Verschlüsselungsaufwand möglichst gering zu halten; in diesem Fall gibt es eine Zahl $k \in \{1, 2\}$ und ein $d \in \mathbb{N}$, so daß $de - k(p-1)(q-1) = 1$ ist. Zur Berechnung von d genügt es daher, in der Formel

$$d = \frac{1 + k(p-1)(q-1)}{e}$$

die Werte $k = 1$ und $k = 2$ einzusetzen und zu schauen, für welchen der beiden man ein ganzzahliges Ergebnis erhält.

Der Wert $e = 3$ ist allerdings dann problematisch, wenn man dieselbe Nachricht an mehrere Empfänger schickt, jeweils verschlüsselt mit deren öffentlichen Schlüsseln. Im Falle von drei Schlüsseln N_1, N_2, N_3 etwa würden also die drei Zahlen

$$a^3 \bmod N_1, \quad a^3 \bmod N_2 \quad \text{und} \quad a^3 \bmod N_3$$

verschickt, wobei N_1, N_2, N_3 hoffentlich teilerfremde Zahlen sind: Hätten nämlich etwa N_1 und N_2 einen echten gemeinsamen Teiler p, so könnte den jeder über den EUKLIDischen Algorithmus berechnet und somit die N_i faktorisieren.

Wenn a^3 aber modulo dreier teilerfremder Zahlen bekannt ist, ist es auch modulo deren Produkt $N_1N_2N_3$ bekannt, und da a kleiner als jedes N_i sein muß, ist $a^3 \bmod N_1N_2N_3 = a^3$. Somit ist a^3 bekannt, und eine gewöhnlich Wurzelberechnung in $\mathbb{N}$ führt auf a. Deshalb ist es vielleicht besser, eine größere Zahl e zu wählen; ebenfalls sehr beliebt und rechnerisch einfach ist $e = 2^{16} + 1$.

Um wirklich mit RSA umgehen zu können, müssen wir uns noch überlegen, wie man die Potenzen $x^e \bmod N$ und $x^d \bmod N$ effizient berechnen lassen.

Als erstes stellt sich die Frage, wie man überhaupt mit ganzen Zahlen rechnen kann, die deutlich länger sind als 32 oder auch 64 Bit. Das ist zum Glück recht einfach: Man stellt die Zahlen dar durch Ziffern bezüglich einer geeigneten Basis, meist 2^{32}, und führt damit Addition, Subtraktion und Multiplikation entsprechend der üblichen Schulmethoden aus. Der Aufwand für Addition und Subtraktion ist somit proportional zur Ziffernzahl, der für die Multiplikation zu deren Quadrat. Auch die Division benötigt diesen Aufwand, ist allerdings etwas komplizierter, da hier die Schulmethode keinen wirklichen Algorithmus liefert: Die jeweils nächste Ziffer des Quotienten wird dort schließlich nur erraten. In

> DONALD E. KNUTH: The Art of Computer Programming, vol. 2: Seminumerical Algorithms, *Addison Wesley* ^{3}1997

findet man aber auch dazu einen effizienten Algorithmus.

Tatsächlich gibt es Multiplikationsalgorithmen, deren Aufwand nicht mit dem Quadrat der Ziffernzahl steigt, sondern mit einer Potenz, die beliebig nahe bei der Eins liegen kann; auch das findet man im gerade zitierten Buch von KNUTH. Asymptotisch sind solche Algorithmen damit erheblich besser, und in der Tat werden sie beim Rechnen mit Zahlen, deren Stellenzahl im Bereich mehrerer Millionen liegt, auch mit Erfolg eingesetzt. Für „nur" ein paar Tausend Stellen sind aber die klassischen Algorithmen schneller, so daß in der Kryptographie, die mit ein paar hundert Stellen auskommt, nur diese verwendet werden.

Darüber hinaus gibt es inzwischen viele C und C++-Bibliotheken zur Langzahlarithmetik; Computeralgebrasysteme und LISP rechnen meist standardmäßig mit Zahlen beliebiger Länge, und in `java.math` gibt es eine Klasse BigInteger, die ebenfalls Zahlen beliebiger Länge bereitstellt. Sie ist aber leider gnadenlos objektorientiert, d.h. anstelle von

$$a + b, \quad a - b, \quad a \cdot b, \quad a/b \quad \text{oder} \quad a \bmod b$$

muß man

```
a.add(b),  a.subtract(b),  a.multiply(b),
     a.divide(b)  oder  a.remainder(b)
```

schreiben, was längere Formeln schnell unübersichtlich werden läßt. Entsprechen braucht man zum Vergleich eine Methode `equals`, und so weiter.

Erzeugt werden Langzahlen durch eine Vielzahl von Methoden; am wichtigsten sind die Methode `valueOf(x)` mit einer Zahl x vom Typ `long` und `BigInteger(`*string*`)`, wobei *string* aus den (beliebig vielen) Ziffern der Zahl besteht; umgekehrt gibt `toString()` die Zahl als Ziffernfolge aus. Auch verschiedene Algorithmen sind eingebaut; beispielsweise liefert `a.gcd(b)` den ggT von a und b.

Nächstes Problem ist die Berechnung von Potenzen.

Aus gutem Grund offerieren die meisten Programmiersprachen hier keine eingebauten Operatoren, denn je nach Problem können ganz verschiedene Strategien angebracht sein: Im Reellen beispielsweise wird die Formel $x^n = e^{n \ln x}$ meist zu einem brauchbaren Ergebnis führen; für ganzzahligei x oder Zahlen modulo einer natürlichen Zahl N aber wird man die Potenzierung eher aus Multiplikationen aufbauen wollen.

Für kleine Exponenten kann man hier ganz naiv vorgehen und x^n durch $n - 1$ Multiplikationen berechnen. Im bei RSA häufigen Fall $e = 3$ beispielsweise gibt es kaum eine effizientere Methode als die beiden offensichtlichen Multiplikationen.

Das d zu $e = 3$ wird allerdings in der Größenordnung von N liegen und damit in realistischen Anwendungen mehrere hundert Dezimalstellen haben. Für Potenzen mit solchen Exponenten ist eine entsprechende

Vorgehensweise nicht mehr realistisch und würde in der Tat schon für etwa dreißigstellige Exponenten zu einem Programm führen, das selbst die besten heutigen Supercomputer nicht ausführen könnten.

Zum Glück gibt es aber eine erheblich effizientere Alternative, die schon lange zum Standardwerkzeug der Mathematik und Informatik gehört: Um beispielsweise x^{32} zu berechnen brauchen wir keine 31 Multiplikationen, sondern wir können es über die Formel

$$x^{32} = \left(\left(\left(\left(x^2\right)^2\right)^2\right)^2\right)^2$$

mit nur fünf Multiplikationen (genauer: Quadrierungen) berechnen.

Entsprechend können wir für jede gerade Zahl $n = 2m$ die Potenz x^n als Quadrat von x^m berechnen. Für einen ungeraden Exponenten $n = 2m + 1$ erhalten wir x^n als Produkt von x mit dem Quadrat von x^m. Wenn wir dies rekursiv fortsetzen, kommen wir nach spätestens $\log_2 n$ Schritten auf $m = 1$, d.h. mit höchstens $\log_2 n$ Quadrierungen und eher weniger Multiplikationen mit x erhalten wir x^n. Beim Rechnen modulo einer natürlichen Zahl N muß dabei natürlich jede einzelne Operation modulo N ausgeführt werden; berechnet man zuerst x^m und reduziert erst dann modulo N, erhält man bei einer RSA-Entschlüsselung ein Zwischenresultat, das auch alle Computer der Welt zusammen nicht speichern könnten. Die BigInteger-Klasse von Java enthält eine Methode `modPow(e, N)` zur so optimierten Berechnung der e-ten Potenz des Objekts modulo N.

Als nächstes wollen wir uns anschauen, wie und wofür RSA angewendet wird.

1) Verschlüsselung: Jeder, der den öffentlichen Schlüssel (N, e) kennt, kann Nachrichten verschlüsseln: Er bricht die Nachricht auf in Blöcke, die durch ganze Zahlen zwischen null und $N - 1$ dargestellt werden können, berechnet für jeden so dargestellten Block den Chiffretext $b \equiv a^e \bmod N$, der als Zahl zwischen null und $N - 1$ interpretiert und an den Inhaber des geheimen Schlüssels geschickt wird. (Wir schreiben in solchen Fällen in Zukunft kurz $b = a^e \bmod N$, wobei „mod" in diesem Zusammenhang als die Berechnung des Divisionsrests bei Division durch N zu interpretieren ist.)

Der Empfänger berechnet $b^d \bmod N$; da $b^d \equiv a^{ed} \equiv a \bmod N$ ist, entschlüsselt dies die Nachricht.

2) Identitätsnachweis: Im Gegensatz zu symmetrischen Kryptoverfahren endet die Nützlichkeit des RSA-Verfahrens nicht mit der bloßen Möglichkeit einer Verschlüsselung; das Verfahren kann beispielsweise auch benutzt werden, um in Zugangskontrollsystemen, vor Geldautomaten oder bei einer Bestellung im Internet die Identität einer Person Zu beweisen: Nur der Inhaber des geheimen Schlüssels d kann zu einem gegebenen a eine Zahl b berechnen, für die $b^e \equiv a \bmod N$ ist.

Falls also der jeweilige Gegenüber eine Zufallszahl a erzeugt und als Antwort das zugehörige b verlangt, kann er anhand eines öffentlichen Schlüsselverzeichnisses die Richtigkeit von b überprüfen und sich so von der Identität seines Partners überzeugen. Im Gegensatz zu Kreditkarteninformation oder Paßwörtern ist dieses Verfahren auch immun gegen Abhören: Falls jedesmal ein neues zufälliges a erzeugt wird, nützt ein einmal abgehörtes b nichts.

Trotzdem ist das Verfahren in dieser Form nicht als Ersatz zur Übertragung von Kreditkarteninformation oder ähnlichem geeignet, da der Gegenüber anhand des öffentlichen Schlüssels jederzeit zu einer willkürlich gewählten Zahl b die Zahl $a = b^e \bmod N$ erzeugen kann um dann zu behaupten, er habe b als Antwort darauf empfangen. Man müßte also beispielsweise noch zusätzlich verlangen, daß die Zahl a eine spezielle Form hat, etwa daß die vordere Hälte der Ziffernfolge identisch mit der hinteren Hälfte ist.

3) Elektronische Unterschriften: Praktische Bedeutung hat vor allem eine weitere Variante: die elektronische Unterschrift. Her geht es darum, daß der Empfänger erstens davon überzeugt wird, daß eine Nachricht tatsächlich vom behaupteten Absender stammt, und daß er dies zweitens auch einem Dritten gegenüber beweisen kann. (In Deutschland sind solche elektronischen Unterschriften, sofern gewisse formale Voraussetzungen erfüllt sind, rechtsverbindlich.)

Um einen Nachrichtenblock a mit $0 \le a < N$ zu unterschreiben, berechnet der Inhaber des öffentlichen Schlüsseln (N, e) mit seinem geheimen Schlüssel d die Zahl

$$b = a^d \bmod N$$

und sendet das Paar (a, b) an den Empfänger. Dieser überprüft, ob

$$b^e \equiv a \bmod N\,;$$

falls ja, akzeptiert er dies als unterschriebene Nachricht a. Da er ohne Kenntnis des geheimen Schlüssels d nicht in der Lage ist, den Block (a, b) zu erzeugen, kann er auch gegenüber einem Dritten beweisen, daß der Absender die Nachricht a unterschrieben hat.

Für kurze Nachrichten ist dieses Verfahren in der vorgestellten Form praktikabel; in vielen Fällen kann man sogar auf die Übermittlung von a verzichten, da $b^e \bmod N$ für ein falsch berechnetes b mit an Sicherheit grenzender Wahrscheinlichkeit keine sinnvolle Nachricht ergibt.

Falls die übermittelte Nachricht geheimgehalten werden soll, müssen a und b natürlich noch vor der Übertragung mit dem öffentlichen Schlüssel des Empfängers oder nach irgendeinem anderen Kryptoverfahren verschlüsselt werden.

Bei langen Nachrichten ist die Verdoppelung der Nachrichtenlänge nicht mehr akzeptabel, und selbst, wenn man auf die Übertragung von a verzichten kann, ist das Unterschreiben jedes einzelnen Blocks sehr aufwendig. Deshalb unterschreibt man meist nicht die Nachricht selbst, sondern einen daraus extrahierten Hashwert. Dieser Wert muß natürlich erstens von der gesamten Nachricht abhängen, und zweitens muß es für den Empfänger (praktisch) unmöglich sein, zwei Nachrichten zu erzeugen, die zum gleichen Hashwert führen. Diese sogenannten kollisionsfreien Hashfunktionen sind daher deutlich verschieden von jenen Hashfunktionen, die etwa für Suchalgorithmen eingesetzt werden.

Eine wichtige praktische Anwendung haben elektronische Unterschriften bei Chipkarten, wie sie beispielsweise in Frankreich zum Bezahlen

auch in Supermärkten benutzt werden. Wie bei der deutschen Maestro-Karte muß beim Bezahlen eine Geheimzahl eingegeben werden, jedoch wird dann nicht wie hier eine Verbindung zur Bank aufgebaut, die diese Geheimzahl überprüft, sondern der Chip auf der Karte überprüft die eingegebenen Ziffern und meldet an das Terminal des Verkäufers, ob er sie akzeptiert oder nicht.

Solange Chips teuer und schwer erhältlich waren, gab es damit keine Probleme; heute kann sich aber jeder leicht selbst eine Chipkarte basteln, die auf *jede* Eingabe eine positive Antwort gibt. Um solche „Yes-Chips“ zu entlarven, ist daher auf einer echten Chipkarte noch eine RSA-unterschriebene Nachricht gespeichert mit dem Inhalt: Dies ist eine echte Chipkarte für . . . . Die Terminals kennen den öffentlichen Schlüssel des Bankenkonsortiums und können dies somit überprüfen.

4) RSA bei SSL/TLS: SSL steht für *secure socket layer,* TLS für *transport layer security;* Zweck ist jeweils der aufbau einer sicheren Internetverbindung. Wie im Internet üblich, können dazu die verschiedensten Verfahren benutzt werden; die auf Grundlage von RSA zählen derzeit zu den populärsten.

Natürlich ist RSA zu aufwendig, um damit eine längere Kommunikation wie beispielsweise eine *secure shell* Sitzung zu verschlüsseln; tatsächlich dient RSA daher nur zur Übetragung eines Schlüssels für ein konventionelles Kryptoverfahren wie AES, IDEA oder Triple-DES, auf das sich die Beteiligten unter SSL/TLS ebenfalls einigen müssen.

Am einfachsten wäre es, wenn der Client einen Schlüssel für ein solches Verfahren wählt und dann diesen mit dem RSA-Schlüssel des Servers verschlüsselt an diesen schickt – vorausgesetzt, er kennt diesen RSA-Schlüssel. Letzteres ist im allgemeinen nicht der Fall; daher muß zunächst der Server dem Client seinen Schlüssel mitteilen. Da der Client nicht sicher sein kann, mit dem richtigen Server verbunden zu sein, schickt er diesen Schlüssel meist zusammen mit einem Zertifikat, das sowohl seine Identität als auch seinen RSA-Schlüssel enthält und von einer Zertifizierungsstelle unterschrieben ist. Die öffentlichen Schlüssel der gängigen Zertifizierungsstellen sind in die Browserrprogramme eingebaut; bei weniger bekannten Zertifizierungsstellen wie etwa dem Rechenzentrum der Universität Mannheim fragt der Browser den Benutzer, ob er das Zertifikat anerkennen will oder nicht. Bei *secure shell* schließlich, wo die Gegenseite typischerweise keinerlei Zertifikat vorweisen kann, frägt das Programm beim ersten Verbindungsaufbau zu einem server, ob dessen Schlüssel anerkannt werden soll und speichert dann einen sogenannten *fingerprint* davon; dieser wird bei späteren Verbindungen zur Identitätsfeststellung benutzt.

5) Blinde Unterschriften und elektronisches Bargeld: Einer der erfolgversprechendsten Ansätze zum Aushebeln eines Kryptosystems besteht darin, sich auf die Dummheit seiner Mitmenschen zu verlassen.

So sollte es durch gutes Zureden nicht schwer sein, jemanden zu Demonstrationszwecken zum Unterschreiben einer sinnlosen Nachricht zu bewegen: Eine Folge von Nullen und Einsen ohne sinnvolle Interpretation hat schließlich keine rechtliche Wirkung.

Nun muß eine sinnlose Nachricht aber nicht unbedingt eine Zufallszahl sein: Sie kann sorgfältig präpariert sein. Sei dazu etwa m eine Nachricht, die ein Zahlungsversprechen enthält, (N, e) der öffentliche Schlüssel des Opfers und r eine Zufallszahl zwischen 2 und $N - 2$. Dann wird

$$x = m \cdot r^e \bmod N$$

wie eine Zufallsfolge aussehen, für die man eine Unterschrift

$$u = x^d \bmod N = (mr^e)^d \bmod N = m^d r \bmod N$$

bekommt. Multiplikation mit r^{-1} macht daraus eine Unterschrift unter die Zahlungsverpflichtung m.

Das angegebene Verfahren kann nicht nur von Trickbetrügern benutzt werden; blinde Unterschriften sind auch die Grundlage von *digitalem Bargeld.*

Zahlungen im Internet erfolgen meist über Kreditkarten; die Kreditkartengesellschaften haben also einen recht guten Überblick über die Ausgaben ihrer Kunden und machen teilweise auch recht gute Geschäfte mit Kundenprofilen.

Digitales Bargeld will die Anonymität von Geldscheinen mit elektronischer Übertragbarkeit kombinieren und so ein anonymes Zahlungssystem z.B. für das Internet bieten.

Es wir ausgegeben von einer Bank, die für jede angebotene Stückelung einen öffentlichen Schlüssel (N, e) bekanntgibt. Eine Banknote ist eine mit dem zugehörigen geheimen Schlüssel unterschriebene Seriennummer.

Die Seriennummer kann natürlich nicht einfach *jede* Zahl sein; sonst wäre jede Zahl kleiner N eine Banknote. Andererseit dürfen die Seriennummern aber auch nicht von der Bank vergeben werden, denn sonst wüßte diese, welcher Kunde Scheine mit welchem Seriennummern hat. Als Ausweg wählt man Seriennummern einer sehr speziellen Form: Ist $N > 10^{150}$, kann man etwa als Seriennummer eine 150-stellige Zahl wählen, deren Ziffern spiegelsymmetrisch zur Mitte sind, d.h. ab der 76. Ziffer werden die vorherigen Ziffern rückwärts wiederholt. Die Wahrscheinlichkeit, daß eine zufällige Zahl x nach Anwendung des öffentlichen Exponenten auf so eine Zahl führt, ist 10^{-75} und damit vernachlässigbar.

Seriennummern werden von den Kunden zufällig erzeugt. Für jede solche Seriennummer m erzeugt der Kunde eine Zufallszahl r, schickt $mr^e \bmod N$ an die Bank und erhält (nach Belastung seines Kontos) eine Unterschrift u für diese Nachricht zurück. Wie oben berechnet er daraus durch Multiplikation mit r^{-1} die Unterschrift $v = m^d \bmod N$ für die Seriennummer N, und mit diesem Block kann er bezahlen.

Der Zahlungsempfänger berechnet $v^e \bmod N$; falls dies die Form einer gültigen Seriennummer hat, kann er sicher sein, einen von der Bank unterschriebenen Geldschein vor sich zu haben. Er kann allerdings noch nicht sicher sein, daß dieser Geldschein nicht schon einmal ausgegeben wurde.

Deshalb muß er die Seriennummer an die Bank melden, die mit ihrer Datenbank bereits ausbezahlter Seriennummern vergleicht. Falls sie darin noch nicht vorkommt, wird sie eingetragen und der Händler bekommt sein Geld; andernfalls verweigert sie die Zahlung.

Schon bei nur 10^{75} möglichen Nummern liegt die Wahrscheinlichkeit dafür, daß zwei Kunden, die eine (wirklich) zufällige Zahl wählen, dieselbe Nummer erzeugen, bei etwa $10^{-37,5}$. Die Wahrscheinlichkeit, mit jeweils einem Spielschein fünf Wochen lang hintereinander sechs Richtige im Lotto zu haben, liegt dagegen bei $\binom{49}{6}^{-5} \approx 5 \cdot 10^{-35}$, also etwa um den Faktor sechzig höher. Zwei gleiche Seriennummern sind also praktisch auszuschließen, wenn auch theoretisch möglich.

Das System kann nur funktionieren, wenn in diesem Fall der zweite Geldschein mit derselben Seriennummer nicht anerkannt wird, so daß der zweite Kunde sein Geld verliert. Dies muß als eine zusätzliche Gebühr gesehen werden, die mit an Sicherheit grenzender Wahrscheinlichkeit nie fällig wird.

Da digitales Bargeld überdies nur in kleinen Stückelungen sinnvoll ist (Geldscheinen im Millionenwert wären auf Grund ihrer Seltenheit nicht wirklich anonym und würden, wegen der damit verbundenen Möglichkeiten zur Geldwäsche auch in keinem seriösen Wirtschaftssystem angeboten), wäre der theoretisch mögliche Verlust auch nicht sehr groß.

5) Größenordnung der Primzahlen: Selbstverständlich ist $N = 85$ kein geeigneter RSA-Modul: Hier findet jeder die beiden Primfaktoren. Auch

$$N = 2135987035920910082395022704999628797051095341826417406442524165008583957746445088405009430865999$$

ist nicht geeignet: Wie der elsässische Ingenieur SERGE HUMPICH 1997 nach sechs Wochen Rechenzeit mit einem japanschen *freeware* Programm auf seinem privatem PC herausfand, ist

$$\begin{aligned} &2135987035920910082395022704999628797051095341826417406442524165008583957746445088405009430865999 \\ &= 1113954325148827987925490175477024844070922844843 \\ &\times 1917481702524504439375786268230862180696934189293\,, \end{aligned}$$

worüber das französische Bankenkonsortium, das die dortigen Chipkarten herausgibt, überhaupt nicht erfreut war: Schließlich waren die Chips genau mit dieser Zahl geschützt. Das Konsortium setzte durch, daß

HUMPICH wegen des Eindringens in ein DV-System zu zehn Monaten Haft auf Bewährung sowie einem Franc Schadenersatz plus Zinsen verurteilt wurde; dazu kamen 12 000 F Geldstrafe. (Seit 1999 werden neu herausgegebene Chipkarten durch ein größeres N mit 768 statt 320 Bit geschützt; Zahlen dieser Größenordnung können wahrscheinlich erst in ein paar Jahren faktorisiert werden.)

Es ist also klar, daß man sich ernsthaft Gedanken über die Größe der zu verwendenden Primzahlen machen muß.

Ein treu sorgender Staat bleibt seinen Bürgern auf eine so wichtige Frage natürlich keine Antwort schuldig: Zwar gibt es noch keine oberste Bundesbehörde für Primzahlen, aber das Bundesamt für Sicherheit in der Informationstechnik (BSI) und die Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen erarbeiten jedes Jahr ein gemeinsames Dokument mit dem schönen Titel *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Geeignete Algorithmen);* es ist zu finden unter `www.bundesnetzagentur.de`.

Das Signaturgesetz und die Signaturverordnung legen fest, daß elektronische Unterschriften in Deutschland grundsätzlich zulässig und rechtsgültig sind, sofern gewisse Bedingungen erfüllt sind. Zu diesen Bedingungen gehört unter anderem, daß das Verfahren und die Schlüssellänge gemeinsam einen „geeigneten Kryptoalgorihmus“ im Sinne der jeweils gültigen Veröffentlichung der Bundesnetzagentur ist.

Da Rechner immer schneller und leistungsfähiger werden und auch auf der theoretisch-algorithmischen Seite fast jedes Jahr kleinere oder größere Fortschritte zu verzeichnen sind, gelten die jeweiligen Empfehlungen nur für etwa sechs Jahre. Offiziell geht es dabei jeweils nur um die Empfehlung geeigneter Algorithmen für elektronische Unterschriften und deren Schlüssellängen, aber wie die Entwicklung der letzten Jahre zeigte, drehen sich die Diskussionen, die zu den jeweiligen Empfehlungen führen, tatsächlich fast ausschließlich um die jeweils notwendige Schlüssellänge für RSA.

Natürlich hat in einer Demokratie bei so einer wichtigen Frage auch die Bevölkerung ein Mitspracherecht; deshalb beginnt das BSI jeweils zunächst einen Entwurf, zu dem es um Kommentare bittet; erst einige Monate später wird die endgültige Empfehlung verkündet und im Bundesanzeiger veröffentlicht.

Die interessierte Öffentlichkeit, von der die Kommentare zu den Entwürfen kommen, besteht naturgemäß in erster Linie aus Anbietern von Kryptographie-Software, und als erfahrene Experten für Datensicherheit wissen diese, daß ein Verfahren nur dann wirklich geeignet sein kann, wenn es die eigene Firma im Angebot hat. (Am geeignetsten sind natürlich die Verfahren, die keines der Konkurrenzunternehmen anbietet.)

Derzeit erhältliche Hardware-Implementierungen von RSA unterstützen typischerweise Schlüssellängen von bis zu 1024 Bit; größere Schlüssel sind vor allem in *public domain* Software wie PGP zu finden. Dies erklärt, warum es in den letzten Jahren recht lebhafte Diskussionen gab:

Bis Ende 2000 galten 768 Bit als ausreichende Größe für das Produkt N der beiden Primzahlen, aber die Richtlinien für 2000 legten fest, daß danach bis Mitte 2005 eine Mindestgröße von 1024 Bit erforderlich sei, danach bis Ende 2005 sogar 2048 Bit.

Anbieterproteste führten dazu, daß nach den Richtlinien von 2001 eine Schlüssellänge von 1024 dann doch noch bis Ende 2006 sicher war; die Schlüssellänge 2048 war nur noch „empfohlen“, also nicht mehr verbindlich.

Im April 2002 erschien der erste Entwurf für die 2002-Richtlinien; danach war für 2006 und 2007 nur eine Mindestlänge von 2048 Bit wirklich sicher. Einsprüche führten im September 2002 zu einem revidierten Entwurf, wonach 2006 doch noch 1024 Bit reichen, 2007 aber mindestens 1536 notwendig werden. Die Mindestlänge von 2048 Bit wurde wieder zur „Empfehlung“ zurückgestuft.

Am 2. Januar 2003 erschienen endlich die offiziellen Richtlinien des Jahres 2002; veröffentlich wurden sie am 11. März 2003 im Bundesanzeiger Nr. 48, S. 4202–4203. Danach reichen 1024 Bit auch noch bis Ende 2007, erst 2008 werden 1280 Bit erforderlich. 2048 Bit bleiben dringend empfohlen.

Die neuesten Richtlinien stammen vom 2. Januar 2006 (Bundesanzeiger Nr. 58 vom 23. März 2006, S. 1913–1915). Sie empfehlen grundsätzlich schon heute 2048 Bit, aber wirklich verbindlich sind

bis Ende	2007	2008	2009	2010	2011	
Minimallänge	1024	1280	1536	1728	1976	*Bit.*

(1976 unterscheidet sich nicht wesentlich von 2048; der minimal kleinere Wert wurde gewählt, weil die heute erhältlichen Chipkarten mit dem Betriebssystem SECCOS nicht mit den vollen 2048 Bit fertig werden.)

Die beiden Primfaktoren p, q sollen zufällig und unabhängig voneinander erzeugt werden und aus einem Bereich stammen, in dem

$$\varepsilon_1 < |\log_2 p - \log_2 q| < \varepsilon_2$$

gilt. Als *Anhaltspunkte* werden dabei die Werte $\varepsilon_1 = 0{,}1$ und $\varepsilon_2 = 30$ vorgeschlagen; ist p die kleinere der beiden Primzahlen, soll also

$$2^{-10}p < q < 2^{30}p \approx 10^9 p$$

sein, d.h. die beiden Primzahlen sollten zwar ungefähr dieselbe Größenordnung haben, aber nicht zu nahe beieinander liegen. Der Grund dafür ist ein von FERMAT entdecktes Faktorisierungsverfahren auf Grundlage der dritten binomischen Formel: Falls für eine Zahl N und eine natürliche Zahl y die Zahl $N + y^2$ eine Quadratzahl x^2 ist, ist $N = x^2 - y^2 = (x + y)(x - y)$, womit zwei Faktoren gefunden sind. Probiert man alle kleinen natürlichen Zahlen y systematisch durch, führt dieses Verfahren offensichtlich umso schneller zum Erfolg, je näher die beiden Faktoren von N beieinander liegen.

(Die Empfehlungen für 2007 sind noch nicht publiziert, jedoch sieht der Entwurf bei der RSA-Schlüssellänge keine Änderung vor: 1976 Bit sind also auch noch bis Ende 2012 sicher.)

e) Das Verfahren von Diffie und Hellman

DIFFIE und HELLMAN hatten in ihrer Arbeit zwar noch kein *public key* Verfahren angegaben, sie beschrieben aber einen Algorithmus zur

Schlüsselvereinbarung, das im Gegensatz zu RSA sogar ganz ohne vorher bekannte Schlüssel auskommt: Zwei Personen, die sich noch nie gesehen haben, vereinbaren über eine unsichere Leitung einen Schlüssel, den anschließend nur sie kennen.

Ausgangspunkt ist wieder das Potenzieren im Körper $\mathbb{F}_p$; hier betrachten wir aber die Exponentialfunktion $x \mapsto a^x$ zu einer geeigneten Basis a. Ihre Umkehrfunktion bezeichnet man als *Index* oder *diskreten Logarithmus* zur Basis a:

In $\mathbb{R}$ ist der Logarithmus zur Basis a die Umkehrfunktion der Funktion $x \mapsto a^x$, und genauso definieren wir ihn auch für endliche Körper:

$$y = a^x \Longrightarrow x = \log_a y \,.$$

Trotz dieser formalen Übereinstimmung gibt es es allerdings große Unterschiede zwischen reellen Logarithmen und ihren Analoga in endlichen Körpern: Während reelle Logarithmen sanft ansteigende stetige Funktionen sind, die man leicht mit beliebig guter Genauigkeit annähern kann, sieht der diskrete Logarithmus typischerweise so aus, wie es in der Abbildung zu sehen ist. Auch ist im Reellen der Logarithmus zur Basis $a > 1$ für jede positive Zahl definiert; in endlichen Körpern ist es viel schwerer zu entscheiden, ob ein bestimmter Logarithmus existiert: Modulo sieben etwa sind $2, 4$ und 1 die einzigen Zweierpotenzen, so daß $3, 5$ und 6 keine Zweierlogarithmen haben. Ein Satz aus der Algebra besagt allerdings, daß es stets Elemente a gibt, für die a^x jeden Wert außer der Null annimmt, die sogenannten primitiven Wurzeln. In $\mathbb{F}_7$ wären dies etwa drei und fünf.

Die Berechnung der Potenzfunktion durch sukzessives Quadrieren und Multiplizieren ist auch in endlichen Körpern einfach, für ihre Umkehrfunktion, den diskreten Logarithmus gibt es aber derzeit nur deutlich schlechtere Verfahren. Die derzeit besten Verfahren zur Berechnung von diskreten Logarithmen in Körpern mit N Elementen erfordern etwa denselben Aufwand wie die Faktorisierung eines RSA-Moduls der Größenordnung N. Diese Diskrepanz zwischen Potenzfunktion und Logarithmen kann kryptologisch ausgenutzt werden.

Als Körper verwendet man entweder Körper von Zweipotenzordnung, die wir weiter unten betrachten werden, oder Körper von Primzahl-

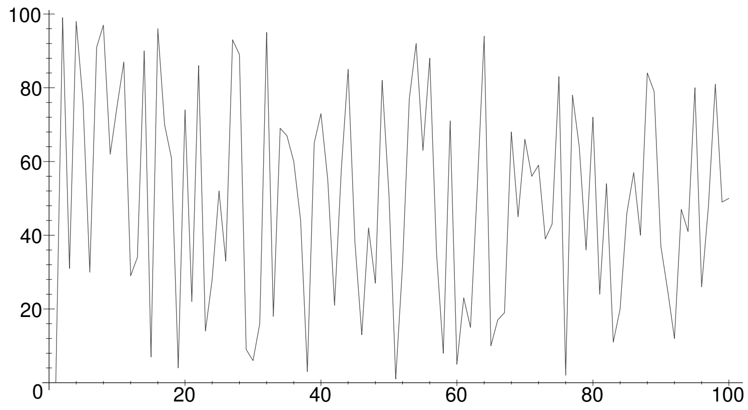


Abb. 12: Die Funktion $\log_{51} x$ in $\mathbb{F}_{101}$

ordnung. Da es für viele interessante Körper von Zweipotenzordnung bereits Chips gibt, die dort diskrete Logarithmen berechnen, dürften Körper von Primzahlordnung bei ungefähr gleicher Elementanzahl wohl etwas sicherer sein: Es gibt einfach viel mehr Primzahlen als Zweierpotenzen, und jeder Fall erfordert einen neue Hardwareentwurf. Falls man die Primzahlen hinreichend häufig wechselt, dürfte sich dieser Aufwand für kaum einen Gegner lohnen.

Da Körper von Primzahlordnung auch einfacher sind als solche von Primzahlpotenzordnung, wollen wir uns zunächst auf diese beschränken; die spätere Übertragung des Algorithmus auf Körper von Zweipotenzordnung sollte dem Leser keine Schwierigkeiten machen.

Beim DIFFIE-HELLMAN-Verfahren, dem ältesten auf der Grundlage diskreter Logarithmen, geht es wie gesagt darum, daß zwei Teilnehmer, die weder über gemeinsame Schlüsselinformation noch über eine sichere Leitung verfügen, einen Schlüssel vereinbaren wollen.

Dazu einigen sie sich zunächst (über die unsichere Leitung) auf eine Primzahl p und eine natürliche Zahl a derart, daß die Potenzfunktion $x \mapsto a^x$ möglichst viele Werte annimmt.

Als nächstes wählt Teilnehmer A eine Zufallszahl $x < p$ und B entsprechend $y < p$; A schickt $u = a^x \bmod p$ an B und erhält dafür $v = a^y \bmod p$.

Sodann berechnet A die Zahl

$$v^x \bmod p = \left(a^y\right)^x \bmod p = a^{xy} \bmod p$$

und B entsprechend

$$u^y \bmod p = \left(a^x\right)^y \bmod p = a^{xy} \bmod p\,;$$

beide haben also auf verschiedene Weise dieselbe Zahl berechnet, die sie nun als Schlüssel in einem klassischen Kryptosystem verwenden können: Beispielsweise könnten die letzten 128, 196 oder 256 Bit der Zahl als AES-Schlüssel dienen. (Den *Advanced Encryption Standard* werden wir weiter unten kennenlernen.)

Ein Gegner, der den Datenaustausch abgehört hat, kennt die Zahlen p, a, u und v; um $a^{xy} \bmod p$ zu finden, muß er den diskreten Logarithmus von u oder v berechnen.

Mit den besten heute bekannten Algorithmen ist die möglich, wenn p eine Primzahl von bis zu etwa 512 Bit ist, also ungefähr 155 Dezimalstellen hat; auch in diesem Fall dauert die Berechnung allerdings selbst bei massiver Parallelisierung über das Internet mehrere Monate, gefolgt von einer Schlußrechnung auf einem Supercomputer.

Da diskrete Logarithmen auch für die in Deutschland rechtlich bindenden digitalen Unterschriften verwendet wird, befaßt sich die Regulierungsbehörde für Telekommunikation und Post in ihrem Bericht über sichere Kryptoverfahren auch damit; bislang hat sie für die Länge der Primzahl noch immer dieselbe Mindestlänge vorgeschrieben wie für einen RSA-Modul.

Natürlich gibt es keine Garantie, daß kein Gegner mit einem besseren als den bislang bekannten Verfahren diskrete Logarithmen oder Faktorisierungen auch in weitaus größeren Körpern berechnen kann. Dazu bräuchte er allerdings einen Durchbruch entweder auf der mathematischen oder auf der technischen Seite, für den weit und breit keine Grundlage zu sehen ist.

Falls sich allerdings die sogenannten *Quantencomputer* realisieren lassen, werden alle heute bekannten Verfahren der Kryptographie mit öffentlichen Schlüsseln, egal ob mit diskreten Logarithmen, RSA oder elliptischen Kurven, unsicher sein. Bislang können Quantencomputer kaum mit drei Bit rechnen, und nicht alle Experten sind davon überzeugt, daß es je welche geben wird, die mit mehreren Tausend Bit rechnen können.

f) Körper von Zweipotenzordnung

Heutige Computer sind nicht für das Rechnen mit sechshundertstelligen Zahlen optimiert, sondern für den Umgang mit Bits und Bytes. Es liegt daher nahe, auch nach Körpern zu suchen, die dies ausnutzen können. Wie wir bald sehen werden, lassen sich in der Tat alle Vektorräume $\mathbb{F}_2^n$ zu Körpern machen können; für $n = 8$ spielt das beispielsweise eine große Rolle für die Fehlerkorrektur von CDs sowie für den neuen Kryptographiestandard AES.

Beginnen wir mit dem einfachsten Fall $\mathbb{F}_2^2$! Wir wissen schon, wie $\mathbb{R}^2$ zum Körper gemacht werden kann: Wir wählen eine Basis $\{1, i\}$ und müssen dann nur noch festlegen, was i^2 sein soll.

Entsprechend können wir auch für $\mathbb{F}_2^2$ eine Basis $\{1, \alpha\}$ wählen; dann läßt sich jedes Element von $\mathbb{F}_2^2$ schreiben als $a + b\alpha$. Da es nur viel Elemente gibt, können wir diese leicht explizit angeben: Es sind

$$0, \quad 1, \quad \alpha \quad \text{und} \quad 1 + \alpha\,.$$

Die Addition dieser Elemente ist klar: Schließlich haben wir bereits einen Vektorraum.

Zur Definition der Multiplikation hatten wir bei der Konstruktion von $\mathbb{C}$ festgelegt, daß $i^2 = -1$ sein sollte, d.h. also gleich einem Element, das in $\mathbb{R}$ kein Quadrat ist. Ein solches Element gibt es in $\mathbb{F}_2$ nicht: Jedes Element ist sein eigenes Quadrat. Daher muß entweder $\alpha^2 = \alpha$ oder $\alpha^2 = 1 + \alpha$ sein.

Wäre $\alpha^2 = \alpha$, so wäre $\alpha(\alpha - 1) = 0$, d.h. $\alpha = 0$ oder $\alpha = 1$, was wir natürlich nicht wollen. Also müssen wir

$$\alpha^2 = \alpha + 1$$

setzen. Damit ist dann alles klar, und wir erhalten die folgende Additions- und Multiplikationstafel, die uns insbesondere auch zeigen, daß wir tatsächlich einen Körper konstruiert haben:

$+$	0	1	α	$1+\alpha$
0	0	1	α	$1+\alpha$
1	1	0	$1+\alpha$	α
α	α	$1+\alpha$	0	1
$1+\alpha$	$1+\alpha$	α	1	0

und

$\cdot$	0	1	α	$1+\alpha$
0	0	0	0	0
1	0	1	α	$1+\alpha$
α	0	α	$1+\alpha$	1
$1+\alpha$	0	$1+\alpha$	1	α

Dieser Körper wird üblicherweise mit $\mathbb{F}_4$ bezeichnet: Das $\mathbb{F}$ steht für *finite,* und vier ist ist die Anzahl der Elemente.

Allgemein bezeichnet man einen endlichen Körper mit q Elementen, so es einen gibt, als $\mathbb{F}_q$; in einigen Büchern auch als GF(q), wobei GF für GALOIS *field* steht nach dem französischen Mathematiker EVARISTE GALOIS (1811–1832) und dem englischen Wort *field* für *Körper.*

Man kann zeigen, daß es genau dann einen solchen Körper gibt, wenn q eine Primzahlpotenz ist, und daß dieser Körper dann bis auf Isomorphie eindeutig bestimmt ist.

Uns interessiert vor allem der Fall, daß $q = 2^n$ eine Zweierpotenz ist. Die Addition von $\mathbb{F}_q$ ist dann die Vektoraddition in $\mathbb{F}_2^n$, und genau wie oben geht es darum, eine Multiplikation zu definieren.

Der einfachste Weg dorthin führt über Polynome: Wir identifizieren den ersten Vektor der Standardbasis mit der Eins von $\mathbb{F}_{2^n} = \mathbb{F}_2^n$, bezeichnen

den zweiten als α und definieren die α-Potenzen bis zur $(n-1)$-ten als die weiteren Basisvektoren:

$$1 = \begin{pmatrix}1\\0\\0\\\vdots\\0\end{pmatrix}, \quad \alpha = \begin{pmatrix}0\\1\\0\\\vdots\\0\end{pmatrix}, \quad \alpha^2 = \begin{pmatrix}0\\0\\1\\\vdots\\0\end{pmatrix}, \quad \dots, \quad \alpha^{n-1} = \begin{pmatrix}0\\0\\0\\\vdots\\1\end{pmatrix}.$$

Damit läßt sich jedes Element von $\mathbb{F}_{2^n}$ als Polynom

$$c_0 + c_1\alpha + c_2\alpha^2 + \cdots + c_{n-1}\alpha^{n-1}$$

schreiben mit $c_i \in \mathbb{F}_2$, und wir können Produkte via Poly-nommultiplikation definieren, sobald wir wissen, was die höheren Potenzen von α sind.

Tatsächlich reicht es bereits, wenn wir nur die Potenz α^n kennen: Diese muß in der Form

$$\alpha^n = p_0 + p_1\alpha + p_2\alpha^2 + \cdots + p_{n-1}\alpha^{n-1}$$

mit $p_i \in \mathbb{F}_2$ darstellbar sein, und sobald wir die Koeffizienten p_i kennen, können wir rekursiv auch alle weiteren α-Potenzen ausrechnen: Beispielsweise ist

$$\begin{aligned}\alpha^{n+1} = \alpha\cdot\alpha^n &= p_0\alpha + p_1\alpha^2 + p_2\alpha^3 + \cdots + p_{n-2}\alpha^{n-2} + p_{n-1}\alpha^n\\ &= p_0\alpha + p_1\alpha^2 + p_2\alpha^3 + \cdots + p_{n-2}\alpha^{n-1}\\ &\qquad + p_{n-1}\left(p_0 + p_1\alpha + p_2\alpha^2 + \cdots + p_{n-1}\alpha^{n-1}\right)\\ &= p_{n-1}p_0 + (p_{n-1}p_1 + p_2)\alpha + (p_{n-1}p_2 + p_3)\alpha^2 + \cdots\\ &\qquad + (p_{n-1}p_{n-2} + p_{n-1})\alpha^{n-2} + (p_{n-1}^2 + p_n)\alpha^{n-1},\end{aligned}$$

und entsprechend geht es weiter für die höheren Potenzen.

Wie wir schon beim Körper mit vier Elementen gesehen haben, können wir die Koeffizienten p_i nicht beliebig aus $\mathbb{F}_2$ wählen; nur in einem Fall ergab sich dort wirklich ein Körper.

Um zu sehen, welche Bedingungen wir an die p_i stellen müssen, nehmen wir an, wir hätten bereits Koeffizienten gefunden, für die sich ein

Körper $\mathbb{F}_{2^n}$ ergibt, und untersuchen, was wir dann über die p_i aussagen können.

Wir können die Gleichung

$$\alpha^n = p_0 + p_1\alpha + p_2\alpha^2 + \cdots + p_{n-1}\alpha^{n-1}$$

auch so auffassen, daß α eine Nullstelle des Polynoms

$$\begin{aligned} P(x) &= x^n - p_0 - p_1x - p_2x^2 - \cdots - p_{n-1}x^{n-1} \\ &= x^n + p_0 + p_1x + p_2x^2 + \cdots + p_{n-1}x^{n-1} \end{aligned}$$

im Körper $\mathbb{F}_{2^n}$ sein soll. (Das zweite Gleichheitszeichen kommt daher, daß es beim Rechnen im Körper $\mathbb{F}_2$ und in den Vektorräumen $\mathbb{F}_2^n$ keinen Unterschied gibt zwischen *plus* und *minus:* Für jeden Vektor $\vec{v} \in \mathbb{F}_2^n$ ist $\vec{v} + \vec{v} = \vec{0}$.)

Wenn wir nun ein Element von $\mathbb{F}_{2^n}$ als Polynom in α schreiben, ist diese Darstellung offensichtlich nicht eindeutig, denn beispielsweise ist

$$f(\alpha) = f(\alpha) + P(\alpha) = (f + P)(\alpha)$$

und allgemeiner gilt sogar für *jedes* Polynom g mit Koeffizienten in $\mathbb{F}_2$, daß

$$(f + g \cdot P)(\alpha) = f(\alpha) + g(\alpha) \cdot P(\alpha) = 0$$

ist. Offensichtlich ist $f(\alpha) = h(\alpha)$, wann immer das Polynom $f - h$ durch P teilbar ist.

Dies liefert einen neuen und schnelleren Zugang zur Multiplikation in $\mathbb{F}_{2^n}$: Um das Produkt zweier Elemente $f(\alpha)$ und $g(\alpha)$ auszurechnen, berechnen wir das Produktpolynom $f \cdot g$ und dividieren es mit Rest durch P, d.h.

$$(f \cdot g) : P = q \quad \text{Rest } h \qquad \text{oder} \qquad f \cdot g = q \cdot P + r\,.$$

Dann ist

$$(f \cdot g)(\alpha) = r(\alpha)\,,$$

und da r ein Polynom vom Grad höchstens $n - 1$ ist, kann $r(\alpha)$ direkt mit einem Vektor aus $\mathbb{F}_2^n$ identifiziert werden.

Rechnen wir etwa im Fall $n = 3$ mit dem Polynom $P = x^3 + x + 1$, so wird das Produkt der Vektoren

$$\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \quad \text{und} \quad \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

folgendermaßen bestimmt: Die beiden Vektoren lassen sich als Linearkombination der Potenzen von α schreiben als

$$1 + 0 \cdot \alpha + 1 \cdot \alpha^2 = 1 + \alpha^2 \quad \text{und} \quad 0 + 1 \cdot \alpha + 1 \cdot \alpha^2 = \alpha + \alpha^2\,;$$

das Produkt der beiden zugehörigen Polynome

$$f = 1 + x^2 \quad \text{und} \quad g = x + x^2 \quad \text{ist} \quad x + x^2 + x^3 + x^4\,.$$

Division durch P ergibt

$$(x^4 + x^3 + x^2 + x) : (x^3 + x + 1) = x + 1 \quad \text{Rest } x + 1\,,$$

d.h. $(1 + \alpha^2)(\alpha + \alpha^2) = 1 + \alpha$ oder

$$\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}\,.$$

Falls man das Polynom P als Produkt zweier Polynome f und g schreiben kann, die beide positiven Grad haben, haben beide insbesondere auch höchstens Grad $n - 1$, definieren also nichtverschwindende Elemente $f(\alpha)$ und $g(\alpha)$ aus $\mathbb{F}_{2^n}$. Deren Produkt ist aber $P(\alpha) = 0$, was in einem Körper natürlich nicht vorkommen darf. Damit haben wir eine erste Bedingung an P gefunden: P muß *irreduzibel* sein im Sinne der folgenden Definition:

Definition: Ein nichtkonstantes Polynom $P \in k[x]$ mit Koeffizienten aus einem Körper k heißt *reduzibel,* wenn es zwei nichtkonstante Polynome $f, g \in k[x]$ gibt, so daß $P = f \cdot g$ ist. Andernfalls heißt P *irreduzibel* (über k).

(Der Zusatz *über* k ist notwendig: Beispielsweise ist $x^2 + 1$ irreduzibel über $\mathbb{R}$, aber reduzibel über $\mathbb{C}$, denn dort ist $(x^2 + 1) = (x + i)(x - i)$. Da meist klar ist, über welchem Körper man arbeitet, wird der Zusatz aber oft weggelassen: Bei uns etwa geht es im Augenblick ausschließlich um Polynome über $\mathbb{F}_2$, so daß dieser Körper nicht ständig erwähnt werden muß.)

Wenn wir uns noch einmal die Konstruktion des Körpers mit vier Elementen anschauen, sehen wir, daß Irreduzibilität zumindest dort auch reicht: Von den vier Polynomen zweiten Grades über $\mathbb{F}_2$ ist genau eines irreduzibel, nämlich das, mit dem wir den Körper $\mathbb{F}_4$ definiert haben:

Ansatz für α^2	Polynom	Problem
$\alpha^2 = 0$	$f = x^2 = x \cdot x$	$\alpha \cdot \alpha = 0$
$\alpha^2 = 1$	$f = x^2 + 1 = (x+1) \cdot (x+1)$	$(\alpha+1)(\alpha+1) = 0$
$\alpha^2 = \alpha$	$f = x^2 + x = x(x+1)$	$\alpha \cdot (\alpha+1) = 0$
$\alpha^2 = \alpha + 1$	$f = x^2 + x + 1$	*keine Probleme*

Tatsächlich reicht die Irreduzibilität von P *immer* zur Definition eines Körpers; damit wir das zeigen können, müssen wir aber zunächst den EUKLIDischen Algorithmus auf Polynome verallgemeinern.

g) Der Euklidische Algorithmus für Polynome

Dazu sei k ein Körper, z.B. der Körper $\mathbb{F}_2$ mit zwei Elementen; außerdem seien

$$A = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$$

und

$$B = x^m + b_{m-1}x^{m-1} + \cdots + b_1x + b_0$$

Polynome mit Koeffizienten a_i, b_i aus k; wir bezeichnen

$$n = \deg A \quad \text{und} \quad m = \deg B$$

als die Grade von A und B.

Dann läßt sich das Polynom A mit Rest durch B dividieren, d.h. man kann Polynome Q, R bestimmen, für die

$$A = QB + R \quad \text{ist mit} \quad \deg R < \deg B\,.$$

Mit dieser Division lassen sich sowohl der gewöhnliche als auch der erweiterte EUKLIDische Algorithmus sofort verallgemeinern auf Polynome; da der Grad von R kleiner ist als der von B und Grade als nichtnegative ganze Zahlen nicht unbegrenzt kleiner werden können, folgt daß der Algorithmus auch für Polynome stets nach endlich vielen Schritten endet.

Das Ergebnis kann allerdings in manchen Fällen unerwartet ausfallen: Betrachten wir etwa über dem Körper $\mathbb{Q}$ der rationalen Zahlen die beiden Polynome

$$P = X^8 + X^6 - 3X^4 - 3X^3 + 8X^2 + 2X - 5$$

und

$$Q = 3X^6 + 5X^4 - 4X^2 - 9X + 21\,.$$

Division von P durch Q führt auf den Quotienten $X^2/3 - 2/9$ und Divisionsrest

$$R_2 = -\frac{5}{9}X^4 + \frac{1}{9}X^2 - \frac{1}{3}\,.$$

Division von Q durch R_2 ergibt

$$R_3 = -\frac{117}{25}X^2 - 9X + \frac{441}{25}\,,$$

bei der Division von R_2 durch R_3 bleibt Rest

$$R_4 = \frac{233150}{6591}X - \frac{102500}{2197}\,,$$

und bei der letzten Division verbleibt als Rest der ggT

$$R_5 = \frac{1288744821}{543589225}\,.$$

Da beide Ausgangspolynome ganzzahlige Koeffizienten haben, erscheint ein ggT mit einem so großen Nenner seltsam. In der Tat ist jedes Polynom durch jede von Null verschiedene Konstante teilbar; ist also ein Polynom P Teiler eines Polynoms Q, so ist auch jedes von Null verschiedene skalare Vielfache von P Teiler von Q. Somit können wir hier nicht sinnvoll von *dem* größten gemeinsamen Teiler zweier Polynome reden: Jede von Null verschiedene Konstante ist ein ggT. Meist sagt man in einem solchem Fall, „der“ ggT sei Eins. Die Computeralgebra kennt Modifikationen des EUKLIDischen Algorithmus, mit denen sich ohne den Umweg über Brüche stets ein möglichst einfacher ggT berechnen läßt.

Wir haben bislang noch nicht definiert, wann ein Polynom *größer* sein soll als ein anderes: Bei zwei natürlichen Zahlen ist klar, welche größer ist, aber schon bei reellen Polynomen ist alles andere als klar, ob etwa $x + 2$ größer sein soll als $2x + 1$ oder umgekehrt. Wir werden dieses Problem ignorieren und einfach sagen, P sei *ein* größter gemeinsamer Teiler von A und B, wenn P ein gemeinsamer Teiler ist und jeder andere gemeinsamen Teiler ein Teiler von P ist.

Der größte gemeinsame Teiler, den uns der EUKLIDISCHE Algorithmus für Polynome liefert, hat diese Eigenschaft, denn da dieser ggT als Linearkombination von A und B geschrieben werden kann, muß jedes Polynom, das sowohl A als auch B teilt, auch den ggT teilen.

Problematischer ist, daß es viele solche größten gemeinsamen Teiler geben kann: Zumindest jedes von Null verschiedene skalare Vielfache eines ggT ist selbst einer. Zum Glück ist das aber auch schon alles, was passieren kann: Sind nämlich P und Q zwei größte gemeinsame Teiler von A und B, so muß nach Definition P ein Teiler von Q sein und umgekehrt. Da der Grad eines Teilers stets kleiner oder gleich dem des Polynoms ist, haben die beiden also insbesondere denselben Grad, und ihr Quotient, egal in welcher Reihenfolge, hat Grad Null und ist somit eine Konstante.

Der größte gemeinsame Teiler zweier Polynome über einem Körper ist also eindeutig bis auf Multiplikation mit einer nichtverschwindenden Konstanten; diese Konstante kann nach Belieben gewählt werden und wird meist so gewählt, daß das Ergebnis in irgendeinem Sinne einfach wird.

Auf das obige Beispiel angewendet heißt das, daß mit

$$R_5 = \frac{1288744821}{543589225}$$

auch eins ein ggT von A und B ist und man daher im allgemeinen sagen würde, „der“ ggT von A und B sei eins. Es ist ein wohlbekanntes (und umgehbares) Problem der Computeralgebra, daß der EUKLIDische Algorithmus diese einfache Lösung in einer so komplizierten Form liefert; da uns vor allem Polynome über endlichen Körpern interessieren, braucht uns das nicht weiter zu kümmern.

Kehren wir zurück zum Ausgangsproblem: Wir wollen den Vektorraum $\mathbb{F}_2^n$ zu einem Körper machen. Da es in $\mathbb{F}_2$ genau ein von Null verschiedenes Element gibt, spielt die obige Diskussion hier keine Rolle: Für Polynome über $\mathbb{F}_2$ existiert *der* ggT. Trotzdem war diese Diskussion nicht umsonst, denn erstens werden wir im nächsten Kapitel im Zusammenhang mit der Integration rationaler Funktionen den EUKLIDischen Algorithmus auch auf reelle Polynome anwenden, und zweitens sei zumindest kurz erwähnt, daß die folgende Konstruktion auch für eine

beliebige Primzahl p Körper mit p^n Elementen liefert. Sie werden allerdings in der Informationstechnik nur selten benutzt: Dort interessieren praktisch nur die Körper $\mathbb{F}_{2^n}$ und die Körper $\mathbb{F}_p$, denn das Rechnen in $\mathbb{F}_{p^n}$ ist umständlicher als das Rechnen in einem Körper $\mathbb{F}_q$ mit einer Primzahl q der Größenordnung p^n und bietet für $p \neq 2$ keinerlei Vorteile. Lediglich für $p = 2$, wo die Vektorraumstruktur von $\mathbb{F}_2^N$ so gut an die heutige Computer-Hardware angepaßt wird, bieten Körper von Zweierpotenzordnung oft (wenn auch keinesfalls immer!) Vorteile über Körper von Primzahlordnung.

In Abschnitt *e)* hatten wir die ganzen Zahlen modulo p zu einem Körper gemacht; der einzige nichttriviale Schritt dabei war die Existenz des multiplikativen Inversen, die wir aus der linearen Kombinierbarkeit des ggT folgerten und daraus, daß der ggT einer Zahl mit einer Primzahl gleich eins ist, falls die Zahl kein Vielfaches der Primzahl ist.

Genauso wollen wir jetzt Körper definieren, indem wir Polynome über einem festen Körper k modulo einem vorgegebenen Polynom P betrachten: Für ein beliebiges Polynom A über k ist $A \bmod P$ gleich dem Rest bei der Division von A durch P.

Falls A kleineren Grad als P hat, ist natürlich einfach $A \bmod P = A$; zum konkreten Rechnen können wir daher ausgehen vom Vektorraum V aller Polynome vom Grad höchstens d, wobei $d + 1$ der Grad von P ist. Die Addition ist die gewöhnliche Addition von Polynomen, das Nullpolynom ist Neutralelement, und $-A$ ist invers zu A.

Das Produkt AB zweier Polynome $A, B \in V$ kann größeren Grad als d haben; wir setzen daher

$$A \odot B = AB \bmod P\,;$$

dies ist ein Polynom vom Grad höchstens d, und es ist klar, daß die so definierte Multiplikation kommutativ und assoziativ ist und das Distributivgesetz erfüllt. Das konstante Polynom 1 ist Neutralelement auch bezüglich dieser Multiplikation.

Ein inverses Polynom zu A ist ein Polynom B, für das $A \odot B = 1$ ist, d.h.

$$AB = 1 + CP \quad \text{oder} \quad AB + CP = 1$$

für ein geeignetes Polynom C. Zu vorgegebenen Polynomen A und P gibt es solche Polynome B und C genau dann, wenn der ggT von A und P gleich eins ist; alsdann lassen sich B und C nach dem EUKLIDischen Algorithmus berechnen.

Wenn wir möchten, daß jedes Polynom A, dessen Grad kleiner als deg P ist, ein Inverses hat, müssen wir sicherstellen, daß A und P immer teilerfremd sind; dies ist offensichtlich genau dann der Fall, wenn P keinen nichttrivialen Teiler hat, also irreduzibel ist.

Falls es ein irreduzibles Polynom P vom Grad n mit Koeffizienten aus k gibt, läßt sich der Vektorraum k^n also zu einem Körper machen, indem wir ein n-tupel $(a_0, \ldots, a_{n-1})$ mit dem Polynom

$$a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \cdots + a_1X + a_0$$

identifizieren und die Multiplikation als Multiplikation von Polynomen modulo P erklären.

Betrachten wir noch einmal das altbekannte Beispiel der komplexen Zahlen: Für $n = 2$ gibt es irreduzible Polynome vom Grad n über $\mathbb{R}$, beispielsweise das Polynom $P = X^2 + 1$. Da

$$\begin{aligned}(a_1X + a_0)(b_1X + b_0) &= a_1b_1X^2 + (a_0b_1 + a_1b_0)X + a_0b_0 \\ &\equiv (a_0b_1 + a_1b_0)X + (a_0b_0 - a_1b_1) \bmod X^2 + 1\end{aligned}$$

ist, folgt $(a_0, a_1) \odot (b_0, b_1) = (a_0b_0 - a_1b_1, a_0b_1 + a_1b_0)$, wir erhalten also den Körper der komplexen Zahlen. Weitere Beispiele über $\mathbb{R}$ gibt es nicht, denn ein irreduzibles reelles Polynom muß entweder Grad eins oder Grad zwei haben, und da jedes irreduzible quadratische Polynom zwei konjugiert komplexe Nullstellen hat, entstehen dabei immer die komplexen Zahlen – lediglich die Basis über $\mathbb{R}$ ändert sich.

Über endlichen Körpern ist die Situation etwas komplizierter: Hier wissen wir nicht einmal, für welche n es überhaupt ein irreduzibles Polynom vom Grad n gibt. Tatsächlich gibt es sogar ziemlich viele solche Polynome; die Tabelle zeigt deren Anzahl über $\mathbb{F}_2$ für $n \leq 16$.

Mit etwas mehr Algebra zeigt man leicht, daß es über jedem endlichen Körper irreduzible Polynome jedes beliebigen (positiven) Grads gibt

N	Polynome vom Grad N	*davon irreduzibel*	*in Prozent*
2	4	1	25.0 %
3	8	2	25.0 %
4	16	3	18.8 %
5	32	6	18.8 %
6	64	9	14.1 %
7	128	18	14.1 %
8	256	30	11.7 %
9	512	56	10.9 %
10	1024	99	9.7 %
11	2048	186	9.1 %
12	4096	335	8.2 %
13	8192	630	7.7 %
14	16384	1161	7.1 %
15	32768	2182	6.7 %
16	65536	4080	6.2 %

und daß zwei solche Polynome *im wesentlichen* (d.h. bis auf Isomorphie) zum selben Körper führen. Wir wollen uns darauf beschränken, für den uns hauptsächlich interessierenden Fall des Körpers $\mathbb{F}_{256}$ konkrete Polynome zu betrachten: Konkretes Rechnen mit Bitfolgen setzt schließlich ohnehin immer ein konkretes Polynom voraus.

f) Der Körper mit 256 Elementen und CD-Fehlerkorrektur

Zunächst müssen wir diesen Körper definieren, d.h. ein irreduzibles Polynom vom Grad acht über $\mathbb{F}_2$ finden. Wie die obige Tabelle zeigt, haben wir dazu dreißig Möglichkeiten. Diese führen zwar, abstrakt betrachtet, alle auf denselben Körper, aber das praktische Rechnen in diesem Körper hängt natürlich stark von der Wahl des Polynoms ab. Insbesondere wird die Geschwindigkeit umso höher, je weniger Terme das Polynom hat.

Dreizehn der dreißig Polynome bestehen aus sieben nichtverschwindenden Termen, die restlichen siebzehn aus fünf; Wir wählen natürlich eines der letzteren. Alle diese Polynome haben, wie jedes Polynom vom

Grad acht über $\mathbb{F}_2$, den führenden Term X^8; danach folgen vier weitere Terme. Zur Reduktion modulo einem solchen Polynom $P = X^8 + Rest$ benutzt man, daß dann $X^8 \equiv Rest$, $X^9 \equiv X \cdot Rest$ ist *usw.;* dies wird umso häufiger mehrfach angewandt werden müssen, je höheren Grad die Terme in *Rest* haben. Am effizientesten kann man daher rechnen, wenn das Polynom *Rest* den kleinstmöglichen Grad hat, und wenn zudem auch noch die hinteren Terme von *Rest* möglichst kleinen Grad haben. Inspektion der siebzehn Polynome mit fünf Termen zeigt, daß das Polynom $x^8 + x^4 + x^3 + x + 1$ in dieser Hinsicht optimal ist; es wird beim *Advanced Encryption Standard* AES verwendet, den wir im nächsten Abschnitt kurz betrachten werden. Für die Fehlerkorrektur auf CDs verwendet man das leicht verschiedene Polynom $x^8+x^4+x^3+x^2+1$.

Überlegen wir uns zunächst, worum es bei dieser Fehlerkorrektur geht:

Bei der Fertigung einer CD läßt sich die Fehlerwahrscheinlichkeit pro Bit auf etwa eins zu einer Million herunterdrücken; da aber bei einer Audio-CD rund vier Millionen Bit pro Sekunde verarbeitet werden, treten trotzdem jede Menge Fehler auf, die teilweise verheerende Folgen haben können: Falls beispielsweise das Wort 0000 0000 0001 0101 versehentlich als 1000 0000 0100 0101 interpretiert wird, wird aus einem Pianissimo ein ohrenbetäubender Knacklaut von ca. 90 dB.

Nun sollten allerdings nicht nur Fertigungsfehler korrigiert werden, sondern zumindest in gewissem Umfang auch Fehler durch Fingerabdrücke, Staubkörner *usw.*

Da die Spur bei einer CD etwa einen halben Mikrometer breit ist und die einzelnen Pits zwischen 0,833μ und 3,56μ lang sind, wohingegen ein Fingerabdruck bereits Linien mit einer Breite von 15μ erzeugt und eine beim Staubwischen übriggebliebene Baumwollfaser gar eine Breite von 150μ hat, ist klar, daß sich solche Fehler nicht im Bitbereich bewegen.

Dies wird auf einer CD zum einen dadurch berücksichtigt, daß man die Information nicht linear anordnet (die geraden Bytes werden gegenüber den ungeraden verzögert), zum anderen dadurch, daß man anstelle des Bits das Byte als grundlegende Einheit betrachtet, d.h. man arbeitet mit dem Körper $\mathbb{F}_{256}$.

Die Fehlerkorrektur arbeitet mit Prüfbytes, die (wie Paritätsbits) durch lineare Abbildungen definiert sind. Zu einem Vektor aus 24 Bytes werden in zwei Stufen insgesamt acht Prüfbytes berechnet, und zwar werden zunächst vier Prüfbytes angehängt derart, daß der entstehende Vektor aus $\mathbb{F}_{256}^{28}$ im Kern der linearen Abbildung

$$\varphi\colon \mathbb{F}_{256}^{28} \to \mathbb{F}_{256}^{4}; \quad \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{27} \\ x_{28} \end{pmatrix} \mapsto \begin{pmatrix} \sum_{i=1}^{28} x_i \\ \sum_{i=1}^{28} \alpha^{28-i} x_i \\ \sum_{i=1}^{28} \alpha^{2(28-i)} x_i \\ \sum_{i=1}^{28} \alpha^{3(28-i)} x_i \end{pmatrix}$$

liegt, danach werden vier weitere Bytes angehängt derart, daß der entstehende Vektor aus $\mathbb{F}_{256}^{32}$ im Kern der linearen Abbildung

$$\psi\colon \mathbb{F}_{256}^{32} \to \mathbb{F}_{256}^{4}; \quad \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{31} \\ x_{32} \end{pmatrix} \mapsto \begin{pmatrix} \sum_{i=1}^{32} x_i \\ \sum_{i=1}^{32} \alpha^{32-i} x_i \\ \sum_{i=1}^{32} \alpha^{2(32-i)} x_i \\ \sum_{i=1}^{32} \alpha^{3(32-i)} x_i \end{pmatrix}$$

liegt. $\alpha \in \mathbb{F}_{256}$ bezeichnet dabei wie üblich jenes Element, für das die Eins zusammen mit α bis α^7 eine $\mathbb{F}_2$-Basis von $\mathbb{F}_{256}$ ist, und das Nullstelle des definierenden Polynoms ist. Lineare Abbildungen wie φ und ψ bezeichnet man als REED-SOLOMON-Codes oder kurz CIRC-Codes *(cross interleaved Reed-Solomon-Codes).*

Durch Kombination dieser Prüfbytes mit einer geschickten (nichtlinearen) Anordnung der Bytes auf der Spirale lassen sich selbst Fehler einer Länge von etwa 4 000 Bit beheben – teils durch echte Korrektur, teils durch bloße Fehlererkennung und Interpolation aus unverfälschten Daten: Versuche von Physikern der University of Maryland haben ergeben, daß eine CD eingebohrte Löcher mit einem Durchmesser von 0,8 mm problemlos verkraftet, und selbst ein Lochdurchmesser von 1,5 mm führt kaum zu Knackgeräuschen. Einzelheiten findet man unter

`www.physics.umd.edu/lecdem/services/demos/demosh4/h4-67.htm` .

Auch die Fehlerkorrektur für DVDs funktioniert im wesentlichen nach dem gleichen Schema. Da eine DVD allerdings rund siebenmal so viele

Daten faßt, die damit auch erheblich dichter gepackt werden müssen, ist hier der Einfluß von Kratzern, Fingerabdrücken *usw.* noch einmal deutlich gravierender als bei der CD. Deshalb muß dort auf allen Ebenen der Kodierung mit mehr Prüfbits gearbeitet werden.

Bei den REED-SOLOMON-Codes startet man mit Blöcken aus 192 Zeilen à 172 Bytes. Jede dieser Zeilen wird durch zehn Prüfbytes gemäß einer ähnlichen linearen Abbildung wie oben auf 182 Bytes erweitert, so daß nun eine Matrix von 192 Zeilen und 182 Spalten entsteht. In dieser werden an jede Spalte noch in entsprechender Weise 16 Prüfbytes angehängt, so daß nun insgesamt 208 Zeilen à 182 Bytes entstanden sind. Diese werden auf die DVD geschrieben.

g) Der Körper mit 256 Elementen in der Kryptographie

Zwar lehnt es die Internationale Standardisierungsorganisation ISO ab, ein Kryptoverfahren zu standardisieren (Ein Grund dafür ist die dann befürchtete Bündelung krimineller Energie auf dieses Verfahren), aber das amerikanische Handelsministerium hat am 2. Januar 1997 die Suche nach einem Nachfolgealgorithmus AES *(Advanced Encryption Standard)* für den nach heutigen Maßstäben nicht mehr sicheren DES *(Data Encryption Standard)* international ausgeschrieben. Federführend für die Auswahl war das *National Institute of Standards and Technology* (NIST) in Gaithersburg, Maryland, das am 2. Oktober 2000 den Algorithmus Rijndael der beiden flämischen Kryptologen JOAN DAEMEN und VINCENT RIJMEN auswählte. (Als Aussprachehilfe für Personen, die kein Niederländisch, Flämisch, Surinamer oder Afrikaans sprechen, geben diese folgende englische Approximationen des Wortes „Rijndael“: „Reign Dahl“, „Rain Doll“ und „Rhine Dahl“.) Es steht zu erwarten, daß Rijndael mittelfristig auch außerhalb der USA zu *dem* Standardverfahren in der Kryptographie wird.

Grundidee sind, wie bei allen Kryptoverfahren, die beiden SHANNONschen Forderungen der *Diffusion* und *Konfusion:* Ersteres bedeutet, daß sich schon die Änderung eines einzigen Klartextbits an vielen, möglichst weit entfernten Stellen bemerkbar machen muß, das zweite bedeutet in erster Linie eine hohe Nichtlinearität der Verschlüsselungsabbildung, so daß diese ohne Kenntnis des Schlüssels nicht invertiert werden kann.

Nichtlinearität erreicht Rijndael durch die Abbildung $\mathbb{F}_{256} \to \mathbb{F}_{256}$, die die Null auf sich selbst abbildet und jedes andere Element auf sein multiplikatives Inverses. Über mehrere Runden hinweg wird diese Abbildung auf Byte-Ebene immer wieder mit linearen Abbildungen und Vektoradditionen auf $\mathbb{F}_{256}^4$ und Shift-Operationen auf $\mathbb{F}_{256}^{16}$ oder noch größeren Vektorräumen kombiniert. Alle linearen Abbildungen sind $\mathbb{F}_{256}$-linear, was auf Bitebene noch einmal eines Konfusionseffekt hat. Die einzelnen Operationen hängen ab von einem Schlüsselvektor, der Element von $\mathbb{F}_{256}^{16}, \mathbb{F}_{256}^{24}$ oder $\mathbb{F}_{256}^{32}$ sein kann und somit 128, 192 oder 256 Bit lang ist. Einzelheiten findet man unter `http://www.rijndael.com` .

§ 3: Matrizen und lineare Gleichungssysteme

Die abstrakte Art und Weise, wie wir Vektoren und lineare Abbildungen bisher betrachtet haben, hat zwar den Vorteil, daß wir damit viele Probleme behandeln können, die nichts mit den gewohnten anschaulichen Vektoren zu tun haben; sie hat aber auch den Nachteil, daß wir bislang noch sehr wenige nichttriviale Beispiele konkret durchrechnen können. Dieser Paragraph soll die wichtigsten Hilfsmittel zum Rechnen in endlichdimensionalen Vektorräumen bereitstellen.

a) Abbildungsmatrizen

Basen sind nicht nur nützlich, um Vektoren darzustellen, sie können auch den Umgang mit linearen Abbildungen vereinfachen. Der Grund liegt im folgenden Lemma:

Lemma: V und W seien k-Vektorräume, und $\mathcal{B}$ sei eine Basis von V. Dann ist jede lineare Abbildung $\varphi: V \to W$ eindeutig bestimmt durch die Bilder $\varphi(\vec{b})$ der Basisvektoren $\vec{b} \in \mathcal{B}$. Umgekehrt läßt sich jede Abbildung $\varphi: \mathcal{B} \to W$ eindeutig fortsetzen zu einer linearen Abbildung $\varphi: V \to W$