Asymmetrische Verschlüsselung

Klaus Pommerening
Fachbereich Mathematik
der Johannes-Gutenberg-Universität
Saarstraße 21
D-55099 Mainz

9. April 1997, letzte Revision 9. August 2003

Die neue Idee im Vergleich mit den bisher behandelten klassischen und Bitblock-Chiffren ist:

> Verschlüsselungs- und Entschlüsselungsfunktion unterscheiden sich *wesentlich*; das soll bedeuten, dass sich die Entschlüsselungsfunktion *nicht effizient* aus der Verschlüsselungsfunktion bestimmen lässt.

Ist das so, so kann jede Kommunikationsteilnehmerin ein solches Paar von Funktionen besitzen, dessen Verschlüsselungsteil sie, repräsentiert durch den „öffentlichen Schlüssel“, weltweit offen zur Verfügung stellt, dessen Entschlüsselungsteil, repräsentiert durch den „privaten Schlüssel“, sie dagegen als persönliches Geheimnis bei sich bewahrt und mit *niemandem* teilt.

Die Existenz eines solchen streng persönlichen Geheimnisses hat weitere interessante Anwendungen:

- sicherer Identitätsnachweis (≫starke Authentisierung≪),
- digitale Signatur (elektronische Unterschrift).

Letztere ist einfach die umgekehrte Anwendung von privatem und öffentlichen Schlüssel:

- die Verschlüsselung mit dem privatem Schlüssel kann *niemand außer der Besitzerin* ausführen,
- die Nachprüfung mit dem öffentlichen Schlüssel kann *jeder* ausführen und sich dabei zweifelsfrei überzeugen, dass *die Urheberin und der Inhalt* der Nachricht echt sind.

Für weitere allgemeine Betrachtungen und Anwendungen siehe den Abschnitt „Asymmetrische Verschlüsselung“ der Vorlesung „Datenschutz und Datensicherheit“.

Zur Geschichte:

- Erfindung von Diffie/Hellman 1976 (in der öffentlichen Wissenschaft).
- Bekanntestes Verfahren: RSA 1978.
- James Ellis (Britische Geheimdienstbehörde CESG) 1970, bekannt seit 1997.
- Möglicherweise ca. 1965 bei der NSA.
  - Mögliche Idee: Codebuch-Paradigma - *Umkehrung einer Funktion schwer.*
  - Wahrscheinliche Anwendung: Techniker können Auslösecodes für Kernwaffen verschlüsseln, aber nicht selbst entschlüsseln; dies können nur die zuständigen Befehlshaber.

1 Das RSA-Verfahren und seine algorithmischen Grundlagen

Das wichtigste – d. h., am weitesten verbreitete und am meisten analysierte – ist das RSA-Verfahren, benannt nach seinen Erfindern Ron Rivest, Adi Shamir und Len Adleman. Es beruht auf elementaren zahlentheoretischen Algorithmen und darauf, dass es für die Zerlegung großer natürlicher Zahlen in Primfaktoren keinen effizienten Algorithmus gibt.

Die drei Algorithmen

- binärer Potenzalgorithmus,
- Euklidischer Algorithmus,
- chinesischer Restalgorithmus,

sind *die* grundlegenden Algorithmen der algorithmischen Algebra und Zahlentheorie („Computer-Algebra“), aber von großer Bedeutung auch in der numerischen Mathematik – immer dann, wenn es nicht um näherungsweises Rechnen mit Gleitkomma-Zahlen, sondern um exaktes Rechnen mit ganzen oder rationalen Zahlen oder symbolischen Ausdrücken geht.

1.1 Beschreibung des Verfahrens

Parameter

$n =$ Modul,
$e =$ öffentlicher Exponent,
$d =$ privater Exponent.

mit der Eigenschaft

$$(\star) \quad m^{ed} \equiv m \pmod{n} \qquad \text{für alle } m \in [0 \dots n-1].$$

Naive Beschreibung

In „erster Näherung“ setzt man

$$M = C = \mathbb{Z}/n\mathbb{Z}, \quad K \subseteq [1 \dots n-1] \times [1 \dots n-1].$$

Für $k = (e, d)$ ist

$$E_k : M \longrightarrow C, \qquad m \mapsto c = m^e \bmod n,$$
$$D_k : C \longrightarrow M, \qquad c \mapsto m = c^d \bmod n.$$

Diese Beschreibung ist naiv, weil n variabel und zwar (sogar zwingend, wie sich später zeigen wird) Teil des öffentlichen Schlüssels ist. Insbesondere sind sogar die oben verwendeten Mengen M und C variabel.

Genauere Beschreibung

Um zu einer Beschreibung zu kommen, die auf die allgemeine Definition einer Chiffre passt, gibt man als Parameter vor:

$l =$ Länge des Moduls in Bit („Schlüssellänge“),
$l_1 < l$ Bitlänge der Klartextblöcke,
$l_2 \geq l$ Bitlänge der Geheimtextblöcke.

Es wird eine Block-Chiffre über dem Alphabet $\Sigma = \mathbb{F}_2$ mit

$$M = \mathbb{F}_2^{l_1} \subseteq \mathbb{Z}/n\mathbb{Z} \subseteq \mathbb{F}_2^{l_2} = C$$

konstruiert. Dabei wird ein Schlüssel $k = (n, e, d) \in \mathbb{N}^3$ gewählt mit

$$\ell(n) := \lfloor {}^2\log n \rfloor + 1 = l, \quad 1 \leq e \leq n-1, \quad 1 \leq d \leq n-1,$$

so dass die obige Eigenschaft $(\star)$ erfüllt ist. Dabei ist $\ell(n)$ die Zahl der Bits, das heißt, die Länge der binären Darstellung von n.

Ein Klartextblock m der Länge l_1 wird als Binärdarstellung einer natürlichen Zahl $< n$ gedeutet und kann so mit E_k verschlüsselt werden; das Ergebnis c, wieder eine natürliche Zahl $< n$, wird mit l_2 Bits – eventuell mit führenden Nullen – binär dargestellt.

Der Geheimtextblock c lässt sich zum Entschlüsseln wieder als Zahl $c < n$ deuten und in $m = c^d \bmod n$ transformieren.

Ganz genaue Beschreibung

Siehe PKCS = 'Public Key Cryptography Standard' bei RSA – `http://www.rsasecurity.com/rsalabs/pkcs/`.

Zu beantwortende Fragen

- Wie findet man geeignete Parameter n, d, e, so dass $(\star)$ erfüllt ist?
- Wie implementiert man das Verfahren hinreichend effizient?
- Wie weist man die Sicherheit nach?

Geschwindigkeit

Siehe Vorlesung „Datenschutz und Datensicherheit“,
`http://www.uni-mainz.de/~pommeren/DSVorlesung/KryptoBasis/RSA.html`

1.2 Der binäre Potenzalgorithmus

Das Verfahren zum möglichst effizienten Potenzieren wird sinnvollerweise im allgemeinen Rahmen einer Halbgruppe H mit Einselement 1 abgehandelt. Das Problem besteht dann darin, für $x \in H$ die Potenz x^n mit einer natürlichen Zahl n, also das Produkt aus n Faktoren x durch *möglichst wenige Multiplikationen* auszudrücken. Die ganz naive Methode,

$$x^n = x \cdots x,$$

benötigt $n-1$ Multiplikationen. Der Aufwand ist proportional zu n, wächst also *exponentiell* mit der Stellenzahl von n. Die bessere Idee ist das **binäre Potenzieren**, das im Falle einer additiv geschriebenen Verknüpfung (insbesondere für die Halbgruppe $H = \mathbb{N}$) auch als russische Bauernmultiplikation bekannt ist, und schon bei den Ägyptern um 1800 v. Chr. sowie bei den Indern vor 200 v. Chr. verwendet wurde.

Die Beschreibung des Verfahrens beginnt bei der binären Darstellung des Exponenten n (o. B. d. A. $n > 0$),

$$n = b_k 2^k + \cdots + b_0 2^0 \qquad \text{mit } b_i \in \{0,1\},\ b_k = 1,$$

also $k = \lfloor {}^2\log n \rfloor = \ell(n) - 1$. Dann ist

$$x^n = (x^{2^k})^{b_k} \cdots (x^2)^{b_1} \cdot x^{b_0}.$$

Man erzeugt also der Reihe nach $x, x^2, x^4, \ldots, x^{2^k}$, indem man k-mal quadriert, und multipliziert dann die x^{2^i} mit $b_i = 1$ miteinander; das sind $\nu(n)$ Stück, wobei $\nu(n)$ die Anzahl der Einsen in der binären Darstellung ist. Insbesondere ist $\nu(n) \le \ell(n)$. Insgesamt muss man also $\ell(n) + \nu(n) - 2$ Multiplikationen ausführen.

Damit ist gezeigt:

Satz 1 *Sei H eine Halbgruppe mit 1. Dann lässt sich x^n für alle $x \in H$ und $n \in \mathbb{N}$ mit höchstens $2 \cdot \lfloor {}^2\log n \rfloor$ Multiplikationen berechnen.*

Der Aufwand ist also nur *linear* in der Stellenzahl von n. Natürlich muss man zur Abschätzung des gesamten Rechenbedarfs auch den Aufwand für die Multiplikation in der Halbgruppe H berücksichtigen.

Eine Beschreibung in Pseudocode sieht so aus:

Prozedur BinPot
Eingabeparameter:
x = zu potenzierender Wert
[dient lokal zur Speicherung der iterativen Quadrate].
n = Exponent.
Ausgabeparameter:

y = Ergebnis x^n
[dient lokal zur Akkumulation des Teilprodukts].

Anweisungen:

$y := 1$.
Solange $n > 0$:
Falls n ungerade: $y := yx$.
$x := x^2$.
$n := \lfloor n/2 \rfloor$.

Anmerkungen

1. Der Algorithmus ist im wesentlichen, aber doch nicht ganz optimal. Mit der Theorie der „Additionsketten“ aus der Zahlentheorie kann man zeigen, dass die durchschnittlich benötigte Zahl der Multiplikationen sich asymptotisch wie ${}^2\log n$ verhält, also nur halb so groß ist.

2. Die unterschiedliche Zahl von benötigten Multiplikationen je nach Exponent ist Ansatzpunkt der *Zeitbedarfs-* und der *Stromverbrauchsanalyse* (timing attacks, power attacks nach Paul Kocher), wo ein Gerät, etwa eine Chipkarte, das mit einem geheimen Exponenten potenziert, analysiert wird, um ihm das Geheimnis zu entlocken.

1.3 Der EUKLIDische Algorithmus

Der Euklidische Algorithmus liefert den größten gemeinsamen Teiler (ggT) zweier ganzer Zahlen,

$$\mathrm{ggT}(a,b) = \max\{d \in \mathbb{Z} \mid d|a, d|b\}$$

Wenn man der Einfachheit halber noch $\mathrm{ggT}(0,0) = 0$ setzt, hat man die Funktion

$$\mathrm{ggT} : \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{N}$$

mit den folgenden Eigenschaften:

Hilfssatz 1 *Für beliebige $a, b, c, q \in \mathbb{Z}$ gilt:*
(i) $\mathrm{ggT}(a,b) = \mathrm{ggT}(b,a)$.
(ii) $\mathrm{ggT}(a,-b) = \mathrm{ggT}(a,b)$.
(iii) $\mathrm{ggT}(a,0) = |a|$.
(iv) $\mathrm{ggT}(a-qb,b) = \mathrm{ggT}(a,b)$.

Beweis. Trivial; für (iv) verwendet man die Äquivalenz $d|a,b \Longleftrightarrow d|a-qb,b$. ◇

Der Euklidische Algorithmus wird gewöhnlich als Folge von Divisionen mit Rest aufgeschrieben:

$$r_0 = |a|, r_1 = |b|, \ldots, r_{i-1} = q_i r_i + r_{i+1},$$

wobei q_i der ganzzahlige Quotient und r_{i+1} der eindeutig bestimmte Divisionsrest mit $0 \leq r_{i+1} < r_i$ ist. Ist dann $r_n \neq 0$ und $r_{n+1} = 0$, so ist $r_n = \mathrm{ggT}(a,b)$. Denn aus Hilfssatz 1 folgt

$$\mathrm{ggT}(a,b) = \mathrm{ggT}(r_0,r_1) = \mathrm{ggT}(r_1,r_2) = \ldots = \mathrm{ggT}(r_n,0) = r_n.$$

Da außerdem

$$r_1 > r_2 > \ldots > r_i \geq 0 \quad \text{für alle } i,$$

wird die Abbruchbedingung $r_{n+1} = 0$ nach spätestens $n \leq |b|$ Iterationsschritten (also Divisionen) erreicht.

Eine kleine Erweiterung liefert sogar noch mehr. Es ist nämlich jedes r_i ganzzahlige Linearkombination der beiden vorhergehenden Divisionsreste, also auch von $|a|$ und $|b|$; für r_0 und r_1 ist das trivial, und allgemein folgt es durch Induktion: Sei schon $r_j = |a|x_j + |b|y_j$ für $0 \leq j \leq i$. Dann folgt

$$\begin{aligned} r_{i+1} = r_{i-1} - q_i r_i &= |a|x_{i-1} + |b|y_{i-1} - q_i|a|x_i - q_i|b|y_i \\ &= |a|(x_{i-1} - q_i x_i) + |b|(y_{i-1} - q_i y_i). \end{aligned}$$

Diese Überlegung liefert gleich eine explizite Konstruktion für die Koeffizienten mit; sie erfüllen nämlich die Rekursionsformeln

$$x_{i+1} = x_{i-1} - q_i x_i \quad \text{mit} \quad x_0 = 1, x_1 = 0,$$

$$y_{i+1} = y_{i-1} - q_i y_i \quad \text{mit} \quad y_0 = 0, y_1 = 1,$$

die bis auf die Startwerte mit der Formel für die r_i übereinstimmen:

$$r_{i+1} = r_{i-1} - q_i r_i \quad \text{mit} \quad r_0 = |a|, r_1 = |b|.$$

Der **erweiterte Euklidische Algorithmus** (auch Algorithmus von LAGRANGE genannt) ist die Zusammenfassung dieser drei Rekursionsformeln. Damit ist gezeigt (wenn man die Vorzeichen von x_n und y_n passend justiert):

Satz 2 *Der erweiterte Euklidische Algorithmus liefert in endlich vielen Schritten zu zwei ganzen Zahlen a und b den größten gemeinsamen Teiler d und ganzzahlige Koeffizienten x und y mit $ax + by = d$.*

Bemerkungen

1. Das kleinste gemeinsame Vielfache berechnet man nach der Formel

   $$\mathrm{kgV}(a, b) = \frac{ab}{\mathrm{ggT}(a, b)}$$

   ebenfalls effizient.

2. Der größte gemeinsame Teiler mehrerer Zahlen kann man nach der Formel

   $$\mathrm{ggT}(\ldots(\mathrm{ggT}(\mathrm{ggT}(a_1, a_2), a_3)\ldots, a_r)$$

   berechnen; hier sind noch kleine Optimierungen möglich. Analoges gilt für das kleinste gemeinsame Vielfache.

1.4 Analyse des EUKLIDischen Algorithmus

Ein kleines Problem hat sich im vorigen Abschnitt eingeschlichen: Zwar sind die Quotienten und Divisionsreste sicher durch die Eingabeparameter beschränkt; aber die Koeffizienten x_i und y_i sind auf den ersten Blick nicht kontrollierbar. Wie kann man garantieren, dass es hier nicht zu einem Überlauf bei der üblichen Ganzzahl-Arithmetik mit beschränkter Stellenzahl kommt? Nun, das Wachstum wird durch die folgende Überlegung kontrolliert:

Hilfssatz 2 *Für die Koeffizienten x_i und y_i im erweiterten Euklidischen Algorithmus gilt:*

(i) $x_i > 0$, *wenn i gerade,* $x_i \leq 0$, *wenn i ungerade, und* $|x_{i+1}| \geq |x_i|$ *für* $i = 1, \ldots, n$.

(ii) $y_i \leq 0$, *wenn i gerade,* $y_i > 0$, *wenn i ungerade, und* $|y_{i+1}| \geq |y_i|$ *für* $i = 2, \ldots, n$.

(iii) $x_{i+1}y_i - x_i y_{i+1} = (-1)^{i+1}$ *für* $i = 0, \ldots, n$; *insbesondere sind x_i und y_i stets teilerfremd für* $i = 0, \ldots, n+1$.

(iv) $|x_i| \leq |b|, |y_i| \leq |a|$ *für* $i = 0, \ldots, n+1$, *falls* $b \neq 0$ *bzw.* $a \neq 0$.

Beweis. (Nur angedeutet.) (i), (ii) und (iii) zeigt man durch Induktion. Aus $0 = r_{n+1} = |a|x_{n+1} + |b|y_{n+1}$ folgt dann $x_{n+1}|b$ und $y_{n+1}|a$. $\diamond$

Von besonderem Interesse ist, dass der Euklidische Algorithmus sehr effizient ist – die Zahl der Iterationsschritte wächst nur linear mit der *Stellenzahl* der Eingabeparameter, die gesamte Rechenzeit quadratisch. Es ist eine ziemlich genaue Analyse möglich, die wie folgt aussieht.

Die Divisionskette habe die Länge n (o.B.d.A. $b \neq 0$). Wie groß muss b dann mindestens sein? Es ist $r_n \geq 1, r_{n-1} \geq 2$ und $r_{i-1} \geq r_i + r_{i+1}$. Die FIBONACCI-Zahlen F_n sind rekursiv definiert durch

$$F_0 = 0,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2} \quad \text{für } n \geq 2.$$

Durch Induktion erhält man also $r_i \geq F_{n+2-i}$, wobei der Induktionsanfang heißt: $r_n \geq 1 = F_2$, $r_{n-1} \geq 2 = F_3$; insbesondere folgt $b \geq F_{n+1}$. Anders formuliert:

Satz 3 (BINET 1841) *Für $a, b \in \mathbb{Z}$ mit $0 < b < F_{n+1}$ ergibt der Euklidische Algorithmus den größten gemeinsamen Teiler in höchstens $n-1$ Iterationsschritten.*

Zusatz. *Das gilt auch für $b = F_{n+1}$, außer wenn $a \equiv F_{n+2} \equiv F_n \pmod{b}$.*

Damit haben wir eine elegante mathematische Formulierung, aber noch keine Lösung. Jedoch ist das Wachstum der FIBONACCI-Zahlen sehr genau bekannt. Man kann es durch den goldenen Schnitt $\varphi = \frac{1+\sqrt{5}}{2}$ ausdrücken; es ist $\varphi^2 - \varphi - 1 = 0$.

Hilfssatz 3 *Für eine reelle Zahl $c \in \mathbb{R}$ und einen Index $k \in \mathbb{N}$ sei $F_k > c \cdot \varphi^k$ und $F_{k+1} > c \cdot \varphi^{k+1}$. Dann gilt $F_n > c \cdot \varphi^n$ für alle $n \geq k$.*

Beweis. (Durch Induktion.)

$$F_n = F_{n-1} + F_{n-2} > c\varphi^{n-1} + c\varphi^{n-2} = c\varphi^{n-2}(\varphi + 1) = c\varphi^n$$

für $n \geq k + 2$. $\diamond$

Korollar 1 *$F_{n+1} > 0.43769 \cdot \varphi^{n+1}$ für $n \geq 2$.*

Beweis.

$$\begin{aligned} \varphi^2 &= \varphi + 1 = \frac{3 + \sqrt{5}}{2}, \\ \varphi^3 &= \varphi^2 + \varphi = 2 + \sqrt{5}, \\ \varphi^4 &= \varphi^3 + \varphi^2 = \frac{7 + 3\sqrt{5}}{2}. \end{aligned}$$

Daraus folgt

$$\begin{aligned} \frac{F_3}{\varphi^3} &= \frac{2}{2 + \sqrt{5}} = \frac{2(\sqrt{5} - 2)}{1} = 2\sqrt{5} - 4 > 0.47, \\ \frac{F_4}{\varphi^4} &= \frac{3 \cdot 2}{7 + 3\sqrt{5}} = \frac{6(7 - 3\sqrt{5})}{49 - 45} = \frac{21 - 9\sqrt{5}}{2} > 0.43769 \end{aligned}$$

und daraus die Behauptung. $\diamond$

Korollar 2 *Seien $a, b \in \mathbb{Z}$ mit $b \geq 2$. Dann ist die Anzahl der Iterationsschritte im Euklidischen Algorithmus für* $\mathrm{ggT}(a, b)$ *kleiner als* $0.718 + 4.785 \cdot \log(b)$.

Beweis. Wenn die Divisionskette die Länge n hat, ist $b \geq F_{n+1}$,

$$b \geq F_{n+1} > 0.43769 \cdot \varphi^{n+1},$$

$$\log(b) > \log(0.43769) + (n + 1) \cdot \log(\varphi),$$

also $n < 0.718 + 4.785 \cdot \log(b)$. $\diamond$

Etwas gröber, aber einfacher zu merken, ist die folgende Version:

Korollar 3 *Seien $a, b \in \mathbb{Z}$ mit $b \geq 2$. Dann ist die Anzahl der Iterationsschritte im Euklidischen Algorithmus für* $\mathrm{ggT}(a, b)$ *kleiner als fünfmal die Zahl der Dezimalstellen von b außer für $b = 8, a \equiv 5 \pmod 8$, wo man 5 Iterationsschritte braucht.*

Berücksichtigt man noch die Stellenzahl der vorkommenden Zahlen und den Aufwand für die Multiplikation und Division langer Zahlen, kommt man auf eine Rechenzeit, die quadratisch mit der Stellenzahl wächst, wie im folgenden gezeigt wird.

Hat a (bezüglich der Basis B) die Stellenzahl m und b die Stellenzahl p, so ist der Aufwand für die erste Division alleine schon $\leq c \cdot (m-p) \cdot p$; dabei ist c eine Konstante, die höchstens zweimal so groß ist wie die, die den Aufwand für die „Rückmultiplikation Quotient $\times$ Divisor“ beschreibt. Für B wird man bei heutigen Rechnerarchitekturen in der Regel 2^{32} annehmen, und als primitive Operationen werden die Grundrechenschritte Addition, Subtraktion, Multiplikation, Division mit Rest und Vergleich von einstelligen Zahlen (zur Basis B) gezählt. Zum Glück nehmen im Verlauf der euklidischen Divisionskette die zu dividierenden Zahlen exponentiell ab. Der Divisionsschritt

$$r_{i-1} = q_i r_i + r_{i+1}$$

benötigt noch $\leq c \cdot {}^B\log(q_1) {}^B\log(r_i)$ primitive Operationen, die gesamte Divisionskette also

$$\begin{aligned} A(a,b) &\leq c \cdot \sum_{i=1}^{n} {}^B\log(q_i) {}^B\log(r_i) \leq c \cdot {}^B\log|b| \cdot \sum_{i=1}^{n} {}^B\log(q_i) \\ &= c \cdot {}^B\log|b| \cdot {}^B\log(q_1 \cdots q_n). \end{aligned}$$

Das Produkt der q_i lässt sich weiter abschätzen:

$$|a| = r_0 = q_1 r_1 + r_2 = q_1(q_2 r_2 + r_3) + r_2 = \ldots = q_1 \cdots q_n r_n + \cdots \geq q_1 \cdots q_n;$$

also haben wir die grobe Abschätzung

$$A(a,b) \leq c \cdot {}^B\log|b| \cdot {}^B\log|a|.$$

Satz 4 *Die Anzahl der primitiven Operationen im Euklidischen Algorithmus für ganze Zahlen a und b der Stellenzahlen $\leq m$ ist $\leq c \cdot m^2$.*

Der Aufwand für den Euklidischen Algorithmus mit Input a und b ist also nicht wesentlich größer als der für die Multiplikation von a und b. Eine feinere Abschätzung soll hier nicht durchgeführt werden; ebensowenig werden mögliche Verbesserungen diskutiert. Erwähnt soll aber werden, dass ein Verfahren von LEHMER erlaubt, einen großen Anteil der Langzahl-Divisionen im Euklidischen Algortithmus durch primitive Operationen zu ersetzen.

1.5 Kongruenzdivision

Der erweiterte Euklidische Algorithmus liefert nun eine Lösung des nicht ganz trivialen Problems, im Ring $\mathbb{Z}/n\mathbb{Z}$ der ganzen Zahlen $\bmod n$ effizient zu dividieren.

Satz 5 *Gegeben seien $n \in \mathbb{N}, n \geq 2$, und $a, b \in \mathbb{Z}$ mit $\mathrm{ggT}(b, n) = d$. Genau dann ist a in $\mathbb{Z}/n\mathbb{Z}$ durch b teilbar, wenn $d|a$. Ist dies der Fall, so gibt es genau d Lösungen z von $zb \equiv a \pmod{n}$ mit $0 \leq z < n$, und je zwei solche unterscheiden sich um ein Vielfaches von n/d. Ist $d = xn + yb$ und $a = td$, so ist $z = yt$ Lösung.*

Beweis. Ist a durch b teilbar, $a \equiv bz \pmod{n}$, so $a = bz + kn$, also $d|a$. Ungekehrt sei $a = td$. Nach Satz 2 findet man x, y mit $nx + by = d$; also ist $nxt + byt = a$ und $byt \equiv a \pmod{n}$. Ist auch $a \equiv bw \pmod{n}$, so $b(z-w) \equiv 0 \pmod{n}$, also $z - w$ Vielfaches von n/d. $\diamond$

Ein expliziter Algorithmus für die Division ist dem Beweis von Satz 5 direkt zu entnehmen. Wichtig – und wesentlich einfacher zu formulieren – ist der Spezialfall $d = 1$:

Korollar 1 *Ist b zu n teilerfremd, so ist jedes a in $\mathbb{Z}/n\mathbb{Z}$ eindeutig durch b teilbar.*

Die Berechnung des Inversen y zu b folgt dann, da $d = 1$, sofort aus der Formel $1 = nx + by$; es ist nämlich $by \equiv 1 \pmod{n}$.

Korollar 2 $(\mathbb{Z}/n\mathbb{Z})^\times = \{b \bmod n \mid \mathrm{ggT}(b, n) = 1\}$.

Die invertierbaren Elemente im Ring $\mathbb{Z}/n\mathbb{Z}$ sind also genau die Restklassen der zu n teilerfremden ganzen Zahlen. Der wichtigste Fall ist: $n = p$ Primzahl. Dann gilt

Korollar 3 $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$ *ist ein Körper.*

Beweis. Ist $b \in \mathbb{F}_p, b \neq 0$, so gibt es genau ein $c \in \mathbb{F}_p$ mit $bc = 1$. $\diamond$

Korollar 4 (Kleiner Satz von Fermat) $a^p \equiv a \pmod{p}$ *für alle* $a \in \mathbb{Z}$.

Beweis. Die Elemente $\neq 0$ von $\mathbb{F}_p$ bilden die multiplikative Gruppe $\mathbb{F}_p^\times$. Da die Ordnung eines Elements stets Teiler der Gruppenordnung ist, gilt $a^{p-1} \equiv 1 \pmod{p}$ wenn a zu p teilerfremd ist. Andernfalls gilt $p|a$, also $a \equiv 0 \equiv a^p \pmod{p}$. $\diamond$

1.6 Der chinesische Restalgorithmus

Das chinesische Restproblem ist die Frage nach der Lösung simultaner Kongruenzen. Der einfachste erwähnenswerte Fall geht so:

Satz 6 (Chinesischer Restsatz) *Seien m und n teilerfremde natürliche Zahlen ≥ 1 und a, b beliebige ganze Zahlen. Dann gibt es genau eine ganze Zahl x, $0 \leq x < mn$, mit*

$$x \equiv a \pmod{m}, \; x \equiv b \pmod{n}.$$

Beweis. Die Eindeutigkeit folgt so: Ist auch y eine solche Zahl, so $y = x + km = x + ln$ mit ganzen Zahlen k und l, und $km = ln$. Da m und n teilerfremd sind, folgt $n|k$, $k = cn$,

$$y = x + cmn \equiv x \pmod{mn}.$$

Für den Existenzbeweis setzt man $x = a + tm$ an; dann ist $x \equiv a \pmod{m}$ erfüllt, und

$$x \equiv b \pmod{n} \Longleftrightarrow b - a \equiv x - a \equiv tm \pmod{n}.$$

Ein solches t existiert aber nach Satz 5. Die so gefundene Lösung x wird noch $\bmod(mn)$ reduziert. $\diamond$

Der Beweis ist konstruktiv und leicht in einen Algorithmus umzusetzen. Im allgemeinen Fall, für mehrfache Kongruenzen, lautet das chinesische Restproblem so:

- Gegeben sind q paarweise teilerfremde ganze Zahlen $n_1, \ldots, n_q \geq 1$ und q ganze Zahlen $a_1, \ldots, a_q$.
- Gesucht ist eine ganze Zahl x mit $x \equiv a_i \pmod{n_i}$ für $i = 1, \ldots q$.

Man kann Satz 6 entsprechend verallgemeinern. Interessanter ist aber eine abstrakte Formulierung, die auch die Interpolationsaufgabe für Polynome mit einschließt; auch in dieser allgemeinen Formulierung erkannt man Satz 6 samt Beweis leicht wieder, wenn man daran denkt, dass für ganze Zahlen m und n mit größtem gemeinsamen Teiler d gilt:

$$m, n \text{ teilerfremd} \Longleftrightarrow d = 1 \Longleftrightarrow \mathbb{Z}m + \mathbb{Z}n = \mathbb{Z}.$$

Satz 7 (Allgemeiner chinesischer Restsatz) *Sei R ein kommutativer Ring mit Einselement, $q \geq 1$, $\mathfrak{a}_1, \ldots, \mathfrak{a}_q \trianglelefteq R$ Ideale mit $\mathfrak{a}_i + \mathfrak{a}_j = R$ für $i \neq j$. Seien Elemente $a_1, \ldots, a_q \in R$ gegeben. Dann gibt es ein $x \in R$ mit $x - a_i \in \mathfrak{a}_i$ für $i = 1, \ldots, q$, und die Restklasse $x \bmod \mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_q$ ist eindeutig bestimmt.*

Beweis. Die Eindeutigkeit ist auch hier einfach: Ist $x - a_i, y - a_i \in \mathfrak{a}_i$, so $x - y \in \mathfrak{a}_i$; gilt das für alle i, so $x - y \in \mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_q$.

Die Existenz wird durch Induktion über q bewiesen. Im Fall $q = 1$ nimmt man $x = a_1$. Sei nun $q \geq 2$ und y mit $y - a_i \in \mathfrak{a}_i$ für $i = 1, \ldots, q-1$ schon gefunden. Idee: Zu y kann man ein $s \in \mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_{q-1}$ addieren, ohne das bisher erreichte, nämlich die Lösung der ersten $q-1$ Kongruenzen, wieder aufzugeben. Benötigt wird dazu die Aussage: Zu jedem $r \in R$ gibt es ein $s \in \mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_{q-1}$ mit $r - s \in \mathfrak{a}_q$, oder, anders ausgedrückt,

$$(\mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_{q-1}) + \mathfrak{a}_q = R.$$

Zum Beweis dieser Zwischenbehauptung wählt man $c_i \in \mathfrak{a}_i$ für $i = 1, \ldots, q-1$ und $b_1, \ldots, b_{q-1} \in \mathfrak{a}_q$ mit $b_i + c_i = 1$. Dann ist

$$1 = (b_1 + c_1) \cdots (b_{q-1} + c_{q-1}) = c_1 \cdots c_{q-1} + b$$

mit $c_1 \cdots c_{q-1} \in \mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_{q-1}$ und $b \in \mathfrak{a}_q$.

Nun wird zu $a_q - y \in R$ ein $s \in \mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_{q-1}$ gewählt mit $a_q - y - s \in \mathfrak{a}_q$ und dann $x = y + s$ gesetzt. Dann ist $x \equiv y \equiv a_i \pmod{\mathfrak{a}_i}$ für $i = 1, \ldots, q-1$ und $x \equiv y + s \equiv a_q \pmod{\mathfrak{a}_q}$. $\diamond$

Bemerkungen und Beispiele

1. Ist $R = \mathbb{Z}$ oder sonst ein Hauptidealring und $\mathfrak{a}_i = Rn_i$, so ist $\mathfrak{a}_1 \cap \cdots \cap \mathfrak{a}_q = R(n_1 \cdots n_q)$. Daraus erhält man die übliche Formulierung des chinesischen Restsatzes.

2. Ist R ein Hauptidealring, so ist läuft die Konstruktion der Lösung wie folgt: Ist $\mathfrak{a}_i = Rn_i$, so wird s in der Zwischenbehauptung so gewählt, dass $s = tn_1 \cdots n_{q-1}$ mit

   $$r - tn_1 \cdots n_{q-1} \in Rn_q$$

   (Kongruenzdivision mod n_q). Ein expliziter Algorithmus für das chinesische Restproblem existiert also, wenn einer für die Kongruenzdivision existiert, auf jeden Fall also für $R = \mathbb{Z}$.

3. Im Fall $R = \mathbb{Z}$ berechnet man iterativ

   $$\begin{aligned} x_1 = a_1 \bmod n_1, &\qquad s_1 = n_1, \\ t_i \text{ mit } 0 \leq t_i \leq n_i - 1 &\quad \text{und } a_i - x_{i-1} - t_i s_{i-1} \in Rn_i, \\ x_i = x_{i-1} + t_i s_{i-1}, &\qquad s_i = s_{i-1} n_i. \end{aligned}$$

   Insbesondere ist $s_k = n_1 \cdots n_k$. Durch Induktion beweist man sofort $0 \leq x_i \leq s_i - 1$ für alle i. Am Ende erhält man die Lösung $x = x_q$. Die

eben durchgeführte Überlegung garantiert, dass kein Zwischenergebnis einen Überlauf erzeugt. Der Aufwand besteht im wesentlichen aus $q-1$ Kongruenzdivisionen und $2\cdot(q-1)$ gewöhnlichen Ganzzahlmultiplikationen. Der Gesamtaufwand ist also ungefähr $cq\times$ dem Aufwand für eine Langzahl-Multiplikation mit einer kleinen Konstanten c.

4. Die allgemeine Gestalt der Lösungsformel ist

$$x = x_1 + t_1 n_1 + \cdots + t_{q-1} n_1 \cdots n_{q-1}.$$

5. Als Beispiel wird die Aufgabe von Sun-Tsu aus dem 1. Jahrhundert behandelt, die in unserer Schreibweise so heißt: Finde x mit

$$x \equiv 2 \pmod 3,\ x \equiv 3 \pmod 5,\ x \equiv 2 \pmod 7.$$

   Der Algorithmus liefert der Reihe nach:

$$\begin{aligned} x_1 = 2, &\quad s_1 = 3, \\ 1 - 3t_2 \in 5\mathbb{Z}, &\quad t_2 = 2, \\ x_2 = 2 + 2\cdot 3 = 8, &\quad s_2 = 15, \\ -6 - 15t_3 \in 7\mathbb{Z}, &\quad t_3 = 1, \\ x = x_3 = 8 + 1\cdot 15 = 23. & \end{aligned}$$

6. Für den Polynomring $K[T]$ über einem Körper K erhält man die Lösung des Interpolationsproblems. Der Algorithmus ist dabei gerade das Interpolationsverfahren von Newton.

1.7 Die EULERsche φ-Funktion

Sinnvollerweise wird hier stets $n \geq 2$ vorausgesetzt. Die multiplikative Gruppe mod$n$, die so oft vorkommt, wird abgekürzt als

$$\mathbb{M}_n := (\mathbb{Z}/n\mathbb{Z})^\times.$$

Eine wichtige Anwendung des chinesischen Restsatzes ist die folgende:

Die ganzen Zahlen mod n bilden den Ring $\mathbb{Z}/n\mathbb{Z}$. Dessen invertierbare Elemente bilden die *multiplikative Gruppe* mod n, $\mathbb{M}_n$. Ihre Ordnung wird durch die EULERsche φ-Funktion beschrieben:

$$\varphi(n) = \#\mathbb{M}_n = \#\{a \in [0 \cdots n-1] \mid a \text{ teilerfremd zu } n\}.$$

Korollar 1 *Sind m und n teilerfremd, so ist $\varphi(mn) = \varphi(m)\varphi(n)$.*

Beweis. Die Aussage des chinesischen Restsatzes bedeutet gerade, daß der natürliche Ring-Homomorphismus

$$F\colon \mathbb{Z}/mn\mathbb{Z} \longrightarrow \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}, \quad x \mapsto (x \bmod m, x \bmod n),$$

bijektiv, also sogar ein Ring-Isomorphismus ist. Außerdem ist $F(\mathbb{M}_{mn}) = (\mathbb{M}_m \times \mathbb{M}_n)$. Also ist

$$\varphi(mn) = \#\mathbb{M}_{mn} = \#\mathbb{M}_n \cdot \mathbb{M}_n = \varphi(m)\varphi(n),$$

wie behauptet. $\diamond$

Ist p prim, so $\varphi(p) = p - 1$, allgemeiner $\varphi(p^e) = p^e - p^{e-1} = p^e(1 - \frac{1}{p})$, wenn $e \geq 1$, denn p^e hat genau die Teiler px mit $1 \leq x \leq p^{e-1}$. Aus Korollar 1 folgt also:

Korollar 2 *Ist $n = p_1^{e_1} \cdots p_r^{e_r}$ die Primfaktorzerlegung (alle $e_i \geq 1$), so*

$$\varphi(n) = n \cdot \prod_{i=1}^{r} (1 - \frac{1}{p_i}).$$

1.8 Die CARMICHAEL-Funktion

Auch hier wird stets $n \geq 2$ vorausgesetzt.

Die CARMICHAEL-Funktion ist definiert als Exponent der multiplikativen Gruppe:

$$\lambda(n) := \mathrm{Exp}(\mathbb{M}_n) = \min\{s \mid a^s \equiv 1 \pmod{n} \quad \text{für alle } a \in \mathbb{M}_n\};$$

d. h., $\lambda(n)$ ist das Maximum der Ordnungen der Elemente von $\mathbb{M}_n$.

Bemerkungen

1. Den Satz von EULER kann man ausdrücken durch $\lambda(n)|\varphi(n)$ („Exponent teilt Gruppenordnung“). Üblich ist die Formulierung

$$a^{\varphi(n)} \equiv 1 \pmod{n} \quad \text{für alle } a \in \mathbb{Z} \text{ mit } \mathrm{ggT}(a, n) = 1.$$

   Beide Formen folgen unmittelbar aus der Definition.

2. Ist p prim, so $\mathbb{M}_p$ zyklisch – siehe unten –, also

$$\lambda(p) = \varphi(p) = p - 1.$$

Hilfssatz 4 *Sei G eine Gruppe vom Exponenten r, H eine Gruppe vom Exponenten s. Dann hat $G \times H$ den Exponenten $t = \mathrm{kgV}(r, s)$.*

Beweis. Da $(g, h)^t = (g^t, h^t) = (1, 1)$ für $g \in G$, $h \in H$, ist der Exponent $\leq t$. Hat $g \in G$ die Ordnung r, $h \in H$ die Ordnung s und (g, h) die Ordnung q, so ist $(g^q, h^q) = (g, h)^q = (1, 1)$, also $g^q = 1$, $h^q = 1$, $r|q$, $s|q$, $t|q$. $\diamond$

Korollar 1 *Sind $m, n \in \mathbb{N}_2$ teilerfremd, so ist*

$$\lambda(mn) = \mathrm{kgV}(\lambda(m), \lambda(n)).$$

Korollar 2 *Ist $n = p_1^{e_1} \cdots p_r^{e_r}$ die Primzerlegung von $n \in \mathbb{N}_2$, so ist*

$$\lambda(n) = \mathrm{kgV}(\lambda(p_1^{e_1}), \ldots, \lambda(p_r^{e_r})).$$

Bemerkungen

3. Die CARMICHAEL-Funktion der Zweierpotenzen (Beweis als Übungsaufgabe – oder im Anhang A.1):

$$\lambda(2) = 1, \quad \lambda(4) = 2, \quad \lambda(2^e) = 2^{e-2} \quad \text{für } e \geq 3.$$

4. Die CARMICHAEL-Funktion ungerader Primpotenzen (Beweis als Übungsaufgabe – oder im Anhang A.3):

$$\lambda(p^e) = \varphi(p^e) = p^{e-1} \cdot (p-1) \quad \text{für } p \text{ prim} \geq 2.$$

Zum Beweis der Aussage in Bemerkung 2 ist noch zu zeigen, dass die multiplikative Gruppe $\bmod p$ tatsächlich zyklisch ist. Das folgt direkt aus einem Standard-Ergebnis der Algebra:

Satz 8 *Sei K ein Körper und $G \leq K^\times$ eine endliche Untergruppe mit $\#G = n$. Dann ist G zyklisch und besteht genau aus den n-ten Einheitswurzeln in K.*

Beweis. Für $a \in G$ ist $a^n = 1$, also ist G enthalten in der Menge der Nullstellen des Polynoms $T^n - 1 \in K[T]$. Also hat K genau n Stück n-te Einheitswurzeln, und G besteht gerade aus diesen. Sei nun m der Exponent von G, insbesondere $m \leq n$. Der folgende Hilfssatz 5 ergibt: Alle $a \in G$ sind schon m-te Einheitswurzeln. Also ist auch $n \leq m$, also $n = m$, und es gibt ein Element in G mit der Ordnung n. $\diamond$

Hilfssatz 5 *Sei G eine abelsche Gruppe.*

(i) *Seien $a, b \in G$, $\operatorname{Ord} a = m$, $\operatorname{Ord} b = n$, m, n endlich und teilerfremd. Dann ist $\operatorname{Ord} ab = mn$.*

(ii) *Seien $a, b \in G$, $\operatorname{Ord} a$, $\operatorname{Ord} b$ endlich, $q = \operatorname{kgV}(\operatorname{Ord} a, \operatorname{Ord} b)$. Dann gibt es ein $c \in G$ mit $\operatorname{Ord} c = q$.*

(iii) *Sei $m = \max\{\operatorname{Ord} a \mid a \in G\} = \operatorname{Exp}(G)$ endlich. Dann gilt $\operatorname{Ord} b \mid m$ für alle $b \in G$.*

Beweis. (i) Sei $k := \operatorname{Ord}(ab)$. Da $(ab)^{mn} = (a^m)^n \cdot (b^n)^m = 1$, ist $k | mn$. Da $a^{kn} = a^{kn} \cdot (b^n)^k = (ab)^{kn} = 1$, gilt $m | kn$, also $m | k$, ebenso $n | k$, also $mn | k$.

(ii) Sei p^e eine Primzahlpotenz mit $p^e | q$, etwa $p^e | m := \operatorname{Ord} a$. Dann hat a^{m/p^e} die Ordnung p^e. Ist nun $q = p_1^{e_1} \cdots p_r^{e_r}$ die Primzahl-Zerlegung mit verschiedenen Primzahlen p_i, so gibt es je ein $c_i \in G$ mit $\operatorname{Ord} c_i = p_i^{e_i}$. Nach (i) hat $c = c_1 \cdots c_r$ die Ordnung q.

(iii) Sei $\operatorname{Ord} b = n$. Dann gibt es ein $c \in G$ mit $\operatorname{Ord} c = \operatorname{kgV}(m, n)$. Also ist $\operatorname{kgV}(m, n) \leq m$, also $= m$, also $n | m$. $\diamond$

1.9 Geeignete RSA-Parameter

Satz 9 *Für eine natürliche Zahl $n \geq 3$ sind äquivalent:*

(i) *n ist quadratfrei.*

(ii) *Es gibt ein $r \geq 2$ mit $a^r \equiv a \pmod{n}$ für alle $a \in \mathbb{Z}$.*

(iii) *Für jedes $d \geq 2$ mit $\mathrm{ggT}(d, \lambda(n)) = 1$ und $e \in \mathbb{N}$ mit $de \equiv 1 \pmod{\lambda(n)}$ gilt $a^{de} \equiv a \pmod{n}$ für alle $a \in \mathbb{Z}$.*

(iv) *Für jedes $k \in \mathbb{N}$ gilt $a^{k \cdot \lambda(n)+1} \equiv a \pmod{n}$ für alle $a \in \mathbb{Z}$.*

Beweis. „(iv) $\Longrightarrow$ (iii)“: Da $de \equiv 1 \pmod{\lambda(n)}$, ist $de = k \cdot \lambda(n) + 1$ für ein geeignetes k. Also ist $a^{de} \equiv a \pmod{n}$ für alle $a \in \mathbb{Z}$.

„(iii) $\Longrightarrow$ (ii)“: Da $n \geq 3$, ist $\lambda(n) \geq 2$. Wählt man d beliebig und e dazu passend mittels Kongruenzdivision mod$\lambda(n)$, so ist (ii) erfüllt mit $r = de$.

„(ii) $\Longrightarrow$ (i)“: Gäbe es eine Primzahl p mit $p^2 | n$, so müsste nach (ii) $p^r \equiv p \pmod{p^2}$ gelten. Da $r \geq 2$, ist aber $p^r \equiv 0 \pmod{p^2}$, Widerspruch.

„(i) $\Longrightarrow$ (iv)“: Nach dem chinesischen Restsatz reicht es zu zeigen, dass $a^{k \cdot \lambda(n)+1} \equiv a \pmod{p}$ für alle Primfaktoren $p | n$.

1. Fall: $p | a$. Dann ist $a \equiv 0 \equiv a^{k \cdot \lambda(n)+1} \pmod{p}$.

2. Fall: $p \nmid a$. Da $p - 1 | \lambda(n)$ ist $a^{\lambda(n)} \equiv 1 \pmod{p}$, also $a^{k \cdot \lambda(n)+1} \equiv a \cdot (a^{\lambda(n)})^k \equiv a \pmod{p}$. $\diamond$

Korollar 1 *Das RSA-Verfahren ist mit einem Modul n genau dann durchführbar, wenn n quadratfrei ist.*

Um passende Exponenten d und e zu finden, muss man $\lambda(n)$ kennen, also am besten (und wie sich zeigen wird, sogar notwendigerweise) die Primzerlegung von n.

Damit wird folgendes Verfahren zur Schlüsselerzeugung nahegelegt:

1. Wahl von verschiedenen Primzahlen $p_1, \ldots, p_r$; Bildung des Moduls $n := p_1 \cdots p_r$.
2. Bestimmung von $\lambda(n) = \mathrm{kgV}(p_1 - 1, \ldots, p_r - 1)$ mit dem Euklidischen Algorithmus.
3. Wahl eines öffentlichen Exponenten $e \in \mathbb{N}_2$, teilerfremd zu $\lambda(n)$, insbesondere e ungerade.
4. Bestimmung des privaten Exponenten d mit $de \equiv 1 \pmod{\lambda(n)}$ durch Kongruenzdivision.

Als öffentlicher Schlüssel wird das Paar (n, e) genommen, als privater Schlüssel der Exponent d.

Korollar 2 *Wer die Primzerlegung von n kennt, kann aus dem öffentlichen Schlüssel (n, e) den privaten Schlüssel d bestimmen.*

Praktische Erwägungen

1. Man wählt so gut wie immer $r = 2$, hat also nur zwei, dafür aber sehr große Primfaktoren p und q. Solche Zahlen $n = pq$ sind besonders schwer zu faktorisieren. Die Primfaktoren sollen außerdem zufällig gewählt, also besonders schwer zu erraten sein. Mehr dazu später.

2. Für e kann man eine Primzahl wählen mit $e \nmid \lambda(n)$ oder eine „kleine“ Zahl ab $e = 3$ – mehr dazu später.

   Eine verbreitete Standard-Wahl ist die Primzahl $e = 2^{16} + 1$, sofern $\nmid \lambda(n)$. Da diese Zahl nur zwei Einsen in ihrer Binärdarstellung hat, ist das binäre Potenzieren für die Verschlüsselung besonders effizient. (Bei der digitalen Signatur ist dies das Verfahren der Signaturprüfung.) Für die Entschlüsselung (bzw. die Erzeugung einer digitalen Signatur) bringt eine solche Wahl von e allerdings keinen Effizienzvorteil.

3. p, q und $\lambda(n)$ werden nach der Schlüsselerzeugung nicht mehr benötigt, könnten also eigentlich vergessen werden.

   Aber: Da d eine „zufällige“ Zahl im Bereich $[1 \dots n]$ ist, ist das binäre Potenzieren mit d aufwändig. Zur Erleichterung kann die Besitzerin des privaten Schlüssels $c^d \bmod p$ und $\bmod q$ rechnen – also mit nur etwa halb so langen Zahlen – und das Ergebnis $\bmod n$ mit dem chinesischen Restsatz zusammensetzen. Dadurch ergibt sich ein kleiner Geschwindigkeitsvorteil bei der Entschlüsselung (bzw. der Erstellung einer digitalen Signatur).

4. Statt $\lambda(n)$ kann man für die Bestimmung der Exponenten auch das Vielfache $\varphi(n) = (p-1)(q-1)$ verwenden.

   Vorteil: Man spart sich (einmal) die kgV-Bestimmung.

   Nachteil: Der Exponent d wird im allgemeinen größer, und das wirkt sich bei jeder Entschlüsselung aus.

5. Trotz des obigen Korollars 1 kann man das RSA-Verfahren auch durchführen, wenn der Modul n nicht quadratfrei ist – der Entschlüsselungsschritt ist etwas komplizierter, da noch ein zusätzlicher „Hensel-Lift“ zwischengeschaltet werden muss. Außerdem geht die Entschlüsselung schief, wenn der Klartext a ein Vielfaches einer Primzahl p mit $p^2 | n$ ist. [D. h., es gibt keinen Widerspruch zum Korollar 1!] Die Gefahr, dass ein Klartext Vielfaches eines Primfaktors von n ist wird stets vernachlässigt; auch für einen quadratfreien Modul n würde ein solcher Klartext ja sofort zur Faktorisierung von n und somit zur Bestimmung des privaten Schlüssels führen.

Achtung

Die kryptoanalytischen Ansätze im folgenden Abschnitt ergeben eine Reihe von Nebenbedingungen, die für die Sicherheit des RSA-Verfahrens bei der Schlüsselerzeugung beachtet werden müssen.

2 Kryptoanalyse des RSA-Verfahrens

„Kryptoanalyse des RSA-Verfahrens“ bedeutet *nicht*, dass das Verfahren gebrochen wird, sondern nur, dass seine Rahmenbedingungen soweit abgesteckt werden, dass man nach menschlichem Ermessen sicher in der Anwendung ist. Insbesondere gilt es einige Fallen zu vermeiden. Zu beantwortende Fragen sind insbesondere:

- Gibt es genügend Schlüssel, um einem Exhaustionsangriff zu entgehen?
- Auf welche mathematische Probleme führt das Brechen eines RSA-Geheimtexts? Oder das Bestimmen des geheimen Schlüssels aus dem öffentlichen?
- Wie muss man die Parameter wählen, um Schwachstellen zu vermeiden?

Einen guten Überblick über das Thema gibt der Artikel

D. Boneh: *Twenty years of attacks on the RSA cryptosystem.* Notices of the American Mathematical Society 46 (1999), 203–213.

Online unter

`http://crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html`

2.1 Der Primzahlsatz

Sei $\pi(x)$ die Anzahl aller Primzahlen $p \leq x$. Etwas allgemeiner sei $\pi_{a,b}(x)$ die Anzahl der Primzahlen $p \leq x$ der Form $p = ak + b$. Der Primzahlsatz ist die asymptotische Relation

$$\pi_{a,b}(x) \sim \frac{1}{\varphi(a)} \cdot \frac{x}{\ln(x)}$$

unter der Voraussetzung, daß a und b teilerfremd sind. Speziell für $a = 1$ und $b = 0$ wird asymptotisch geschätzt:

$$\pi(x) \sim \frac{x}{\ln(x)}.$$

Über die Qualität dieser Approximation gibt es viele theoretische und empirische Ergebnisse, zum Beispiel eine Formel von Rosser und Schoenfeld

$$\frac{x}{\ln(x)} \cdot \left(1 + \frac{1}{2\ln(x)}\right) < \pi(x) < \frac{x}{\ln(x)} \cdot \left(1 + \frac{3}{2\ln(x)}\right) \text{ für } x \geq 59.$$

Mit Hilfe des Primzahlsatzes kann man, wenn auch nicht völlig exakt, folgende Fragen beantworten:

Wieviele Primzahlen $< 2^k$ gibt es?

Antwort: $\pi(2^k)$, also ungefähr

$$\frac{2^k}{k \cdot \ln(2)}$$

Stück, mindestens (für $k \geq 6$)

$$\frac{2^k}{k \cdot \ln(2)} \cdot \left(1 + \frac{1}{2k\ln(2)}\right).$$

Für $k = 128$ sind das ungefähr $3.8 \cdot 10^{36}$, für $k = 255$ ungefähr $3.3 \cdot 10^{74}$, für $k = 256$ ungefähr $6.5 \cdot 10^{74}$.

Wieviele k-Bit-Primzahlen gibt es?

Antwort: $\pi(2^k) - \pi(2^{k-1})$, also ungefähr

$$\frac{2^k}{k \cdot \ln(2)} - \frac{2^{k-1}}{(k-1) \cdot \ln(2)} = \frac{2^{k-1}}{\ln(2)} \cdot \frac{k-2}{k(k-1)} \approx \frac{1}{2} \cdot \pi(2^k)$$

Stück. Für $k = 128$ sind das ungefähr $1.9 \cdot 10^{36}$, für $k = 255$ ungefähr $1.6 \cdot 10^{74}$, für $k = 256$ ungefähr $3.2 \cdot 10^{74}$. Anders ausgedrückt ist eine zufällig gewählte k-Bit-Zahl mit der Wahrscheinlichkeit

$$\frac{\pi(2^k) - \pi(2^{k-1})}{2^{k-1}} \approx \frac{\pi(2^k)}{2^k} \approx \frac{1}{k \cdot \ln(2)} \approx \frac{1.44}{k}$$

Primzahl; für $k = 256$ ist das ungefähr 0.0056.

Eine zuverlässige untere Schranke erhält man aus der Abschätzung

$$\pi(2^k) - \pi(2^{k-1}) > 0.71867 \cdot \frac{2^k}{k} \qquad \text{für } k \geq 21.$$

Auf jeden Fall gibt es in den für das RSA-Verfahren relevanten Größenbereichen so viele Primzahlen, daß ein Exhaustionsangriff völlig wirkungslos bleiben müsste.

Erweiterungen

Sei p_n die n-te Primzahl, also $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, ... Ferner sei $\vartheta(x)$ die Summe der Logarithmen der Primzahlen $\leq x$,

$$\vartheta(x) = \sum_{p \leq x,\, p \text{ prim}} \ln(p).$$

Dann gelten die asymyptotischen Formeln

$$\begin{aligned} p_n &\sim n \cdot \ln(n), \\ \vartheta(x) &\sim x, \end{aligned}$$

sowie die Fehlerschranken von Rosser/Schoenfeld:

$$\begin{aligned} n \cdot \left(\ln(n) + \ln\ln(n) - \frac{3}{2}\right) &< p_n < n \cdot \left(\ln(n) + \ln\ln(n) - \frac{1}{2}\right) \\ &\qquad \text{für } n \geq 20, \\ x \cdot \left(1 - \frac{1}{\ln(x)}\right) &< \vartheta(x) < x \cdot \left(1 - \frac{1}{2\ln(x)}\right) \quad \text{für } n \geq 41. \end{aligned}$$

2.2 Schlüsselbestimmung und Faktorisierung

Frage: *Wie kann man beim RSA-Verfahren den geheimen Exponenten d aus dem öffentlichen Exponenten e und dem Modul n bestimmen?*

Antwort: Jede der folgenden Aufgaben lässt sich effizient auf die anderen zurückführen:

(A) Finden eines passenden geheimen Schlüssels d.

(B) Bestimmung von $\lambda(n)$ (CARMICHAEL-Funktion).

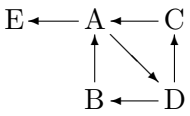
(C) Bestimmung von $\varphi(n)$ (EULER-Funktion).

(D) Faktorisierung von n.

Das Brechen von RSA ist die (möglicherweise echt) leichtere Aufgabe:

(E) Ziehen von e-ten Wurzeln in $\mathbb{Z}/n\mathbb{Z}$.

Der Beweis folgt dem Schema:



Sei dazu $n = p_1 \cdots p_r$ mit verschiedenen Primzahlen $p_1, \ldots, p_r$. Es wird stets angenommen, dass außer n auch der „öffentliche“ Exponent e bekannt ist.

Es ist klar, dass „A $\longrightarrow$ E“ gilt: Wenn d bekannt ist, zieht man die e-te Wurzel durch Potenzieren mit d. Die Umkehrung ist hier allerdings nicht bekannt: *Es könnte sein, dass das Brechen von RSA leichter als die Faktorisierung ist.*

„D $\longrightarrow$ C“: $\varphi(n) = (p_1 - 1) \cdots (p_r - 1)$.

„D $\longrightarrow$ B“: $\lambda(n) = \mathrm{kgV}(p_1 - 1, \ldots, p_r - 1)$.

„B $\longrightarrow$ A“: d wird durch Kongruenzdivision aus $de \equiv 1 \pmod{\lambda(n)}$ gewonnen.

„C $\longrightarrow$ A“: Da $\varphi(n)$ genau die gleichen Primfaktoren hat wie $\lambda(n)$, ist e auch zu $\varphi(n)$ teilerfremd. Durch Kongruenzdivision aus $de \equiv 1 \pmod{\varphi(n)}$ wird ein Exponent d gewonnen, der nicht der „echte“ ist, aber genauso als geheimer Schlüssel funktioniert, da auch $de \equiv 1 \pmod{\lambda(n)}$.

„A $\longrightarrow$ D“ ist wesentlich komplizierter zu zeigen; es wird auch nur ein probabilistischer Algorithmus konstruiert.

Vorbemerkungen

1. *Es reicht, n in zwei echte Faktoren zu zerlegen.*

   (a) Ist nämlich $n = n_1 n_2$ eine solche Zerlegung und o. B. d. A. $n_1 = p_1 \cdots p_s$ mit $1 < s < r$ so ist

   $$\lambda(n_1) = \mathrm{kgV}(p_1 - 1, \ldots, p_s - 1) \,|\, \mathrm{kgV}(p_1 - 1, \ldots, p_r - 1) = \lambda(n),$$

   also auch $de \equiv 1 \pmod{\lambda(n_1)}$. Also ist das Problem auf das analoge für n_1 und n_2 reduziert.

   (b) Da die Zahl der Primfaktoren von n höchstens ${}^2\log(n)$ ist, ist die dadurch gegebene rekursive Reduktion effizient.

2. *Wie kann eine zufällig gewählte Restklasse $w \in \mathbb{Z}/n\mathbb{Z}$ bei der Faktorisierung von n helfen?*

   (a) Findet man ein $w \in [1 \ldots n-1]$ mit $\mathrm{ggT}(w, n) > 1$, so ist n faktorisiert, da $\mathrm{ggT}(w, n)$ ein echter Teiler von n ist.

   (b) Findet man ein $w \in [2 \ldots n-2]$ mit $w^2 \equiv 1 \pmod{n}$ (also eine nichttriviale Quadratwurzel aus 1 in $\mathbb{Z}/n\mathbb{Z}$), so ist n ebenfalls faktorisiert:
   Da $n | w^2 - 1 = (w+1)(w-1)$ und $n \nmid w \pm 1$, ist $\mathrm{ggT}(n, w+1) > 1$, also n nach 1. faktorisiert.

Sei also jetzt ein Paar (d, e) von zusammengehörigen Exponenten bekannt. Dann ist auch $u := ed - 1 = k \cdot \lambda(n)$ bekannt (k und $\lambda(n)$ allerdings nicht).

Da $\lambda(n)$ gerade ist, ist

$$u = r \cdot 2^s \qquad \text{mit } s \geq 1 \text{ und } r \text{ ungerade.}$$

Wählt man irgendein $w \in [1 \ldots n-1]$, so gibt es zwei Möglichkeiten:

- $\mathrm{ggT}(w, n) > 1$ – dann ist n faktorisiert.
- $\mathrm{ggT}(w, n) = 1$ – dann ist $w^{r2^s} \equiv 1 \pmod{n}$.

Im zweiten Fall findet man effizient das minimale $t \geq 0$ mit

$$w^{r2^t} \equiv 1 \pmod{n}.$$

Es gibt wieder zwei Fälle:

- $t = 0$ – Pech gehabt.
- $t > 0$ – dann ist $w^{r2^{t-1}}$ eine Quadratwurzel $\neq 1$ aus 1 in $\mathbb{Z}/n\mathbb{Z}$.

Im zweiten Fall wird unterschieden:

- $w^{r2^{t-1}} \equiv -1 \pmod{n}$ – Pech gehabt.
- $w^{r2^{t-1}} \not\equiv -1 \pmod{n}$ – dann ist n nach Vorbemerkung 2 faktorisiert.

Insgesamt haben wir bei diesem Verfahren nach beliebiger Wahl von $w \in [1 \ldots n-1]$ vier Ausgänge, zwei, bei denen n faktorisiert wird, und zwei, bei denen dies nicht der Fall ist. Die letzteren werden mit

$$\begin{array}{lll} (\mathrm{E}_{n,u}(w)/\mathrm{I}) & w^r \equiv 1 \pmod{n} & \text{und} \\ (\mathrm{E}_{n,u}(w)/\mathrm{II}) & w^{r2^t} \equiv -1 \pmod{n} & \text{für ein } t \text{ mit } 0 \leq t < s \end{array}$$

bezeichnet. Insgesamt ergibt sich die Baumstruktur:

$w \in [1 \ldots n-1] \longrightarrow$
$\quad \mathrm{ggT}(w,n) > 1 \longrightarrow n$ faktorisiert.$\diamond$
$\quad w \in \mathbb{M}_n \longrightarrow$
$\quad\quad w^r \equiv 1 \pmod{n} \longrightarrow (\mathrm{E}_{n,u}(w)/\mathrm{I}.)\diamond$
$\quad\quad w^r \not\equiv 1 \pmod{n} \longrightarrow$
$\quad\quad\quad w^{r2^t} \equiv -1 \pmod{n} \longrightarrow (\mathrm{E}_{n,u}(w)/\mathrm{II}.)\diamond$
$\quad\quad\quad w^{r2^t} \not\equiv -1 \pmod{n} \longrightarrow n$ faktorisiert.$\diamond$

Wir können also n „mit hoher Wahrscheinlichkeit“ faktorisieren, wenn es nur „wenige“ solcher „schlechten“ Zahlen mit $(\mathrm{E}_{n,u}(w)/\mathrm{I,II})$ gibt. Wie viele es sind, wird im nächsten Abschnitt untersucht.

2.3 Die Wahrscheinlichkeit der Faktorisierung

Sei $n \in \mathbb{N}_3$. Ferner sei zunächst $u \in \mathbb{N}_2$ beliebig gerade, $u = r \cdot 2^s$ mit ungeradem r und $s \geq 1$. Dazu werden folgende Mengen eingeführt:

$$\begin{aligned}
A_u^{(0)} &= B_u^{(0)} := \{w \in \mathbb{M}_n \mid w^r = 1\} \quad [\text{Fall } (\mathrm{E}_{n,u}/\mathrm{I})],\\
A_u^{(t)} &:= \{w \in \mathbb{M}_n \mid w^{r\cdot 2^t} = 1, w^{r\cdot 2^{t-1}} \neq 1\} \quad \text{für } 1 \leq t \leq s,\\
B_u^{(t)} &:= \{w \in A_u^{(t)} \mid w^{r\cdot 2^{t-1}} = -1\} \quad [\text{Fall } (\mathrm{E}_{n,u}/\mathrm{II})],\\
A_u &:= \bigcup_{t=0}^{s} A_u^{(t)} = \{w \in \mathbb{M}_n \mid w^u = 1\},\\
B_u &:= \bigcup_{t=0}^{s} B_u^{(t)} \quad [\text{Fall } (\mathrm{E}_{n,u}) \text{ (I oder II)}].\\
C_0 &:= \{w \in \mathbb{M}_n \mid \operatorname{Ord} w \text{ ungerade}\},\\
C_1 &:= \{w \in \mathbb{M}_n \mid -1 \in \langle w \rangle\},\\
C &:= C_0 \cup C_1.
\end{aligned}$$

Bemerkungen

1. $A_u^0 \leq A_u \leq \mathbb{M}_n$ sind Untergruppen, ebenso $A_u^0 \leq C_0 \leq \mathbb{M}_n$.
2. $B_u^{(t)} = A_u^{(t)} \cap C$ für $t = 0, \ldots, s$, denn in einer zyklischen Gruppe $\langle w \rangle$ kann es außer 1 nur eine weitere Quadratwurzel aus 1 geben.
3. Also gilt auch $B_u = A_u \cap C$.
4. B_u ist im Fall von Abschnitt 2.2 genau die Ausnahmemenge mit $(\mathrm{E}_{n,u})$, die nicht zur Faktorisierung von n führt. Der folgende Satz sagt aus, dass die Wahrscheinlichkeit, zufällig ein Element dieser Ausnahmemenge zu erwischen, $< \frac{1}{2}$ ist; probiert man der Reihe nach k zufällige Elemente, ist die Wahrscheinlichkeit, n nicht faktorisiert zu haben, $< 1/2^k$, wird also sehr schnell *extrem* klein.

Satz 1 *Sei n ungerade und keine Primpotenz. Sei $u = r \cdot 2^s$ ein Vielfaches von $\lambda(n)$ mit ungeradem r. Dann ist*

$$\#B_u \leq \frac{1}{2} \cdot \varphi(n).$$

Beweis. Nach dem *folgenden Hilfssatz* ist C und damit erst recht B_u in einer echten Untergruppe von $\mathbb{M}_n$ enthalten. $\diamond$

Hilfssatz 1 (Dixon, AMM 1984) *Sei $n \in \mathbb{N}_3$. Ferner sei $\langle C \rangle = \mathbb{M}_n$. Dann ist n eine Primpotenz oder gerade.*

Beweis. Sei in diesem Beweis $\lambda(n) = r \cdot 2^s$ mit ungeradem r. (Da $n \geq 3$, ist $s \geq 1$. In der „alten“ Bedeutung werden r und s hier nicht benötigt.) Sei

$$h : \mathbb{M}_n \longrightarrow \mathbb{M}_n, \qquad w \mapsto w^{r \cdot 2^{s-1}}.$$

Dann ist h Gruppen-Homomorphismus mit $h(\mathbb{M}_n) \subseteq \{v \in \mathbb{M}_n \mid v^2 = 1\}$ (Gruppe der zweiten Einheitswurzeln $\bmod n$). Da die $w \in C_0$ ungerade Ordnung haben, ist $h(C_0) \subseteq \{1\}$. Für $w \in C_1$ ist $h(w) \in \langle w \rangle$ und $h(w)^2 = 1$, also $h(w)$ eine der beiden Einheitswurzeln $\pm 1 \in \langle w \rangle$.

Insgesamt ist $h(C) \subseteq \{\pm 1\}$.

Ist n keine Primpotenz, so $n = pq$ mit teilerfremden $p, q \in \mathbb{N}_2$. Da $2^s | \lambda(n) = \mathrm{kgV}(\lambda(p), \lambda(q))$, können wir o. B. d. A. $2^s | \lambda(p)$ annehmen. Nach dem chinesischen Restsatz gibt es ein $w \in \mathbb{M}_n$ mit $w \equiv 1 \pmod{q}$, so dass $w \bmod p$ die Ordnung 2^s hat. Dann ist $h(w) \not\equiv 1 \pmod{p}$, also erst recht $h(w) \neq 1$. Da $h(w) \equiv 1 \pmod{q}$, ist auch $h(w) \neq -1$ – außer wenn $q = 2$.

Also ist $h(\mathbb{M}_n) \not\subseteq \{\pm 1\}$, es sei denn, n erfüllt die Behauptung des Hilfssatzes. $\diamond$

2.4 Faktorisierungsalgorithmen

Wie schnell kann man große Zahlen faktorisieren?

- Es gibt „schnelle“ Faktorisierungverfahren für Zahlen der Gestalt $a^b \pm c$ mit „kleinen“ Werten a und c, z. B. für die MERSENNE- und FERMAT-Zahlen $2^b \pm 1$. Die Wahrscheinlichkeit, bei der Erzeugung von RSA-Schlüsseln aus zufällig gewählten Primzahlen einen solchen Modul zu konstruieren, ist verschwindend gering und wird gewöhnlich vernachlässigt.

- Die FERMAT-Faktorisierung von n: Man sucht eine Zahl $a \geq \sqrt{n}$, so dass $a^2 - n$ eine Quadratzahl $= b^2$ ist. Dann ist

$$n = a^2 - b^2 = (a + b)(a - b)$$

  faktorisiert. [Beispiel: $n = 97343$, $\sqrt{n} \approx 311.998$, $312^2 - n = 1$, $n = 313 \cdot 311$.] Diese Methode ist effizient, wenn a nahe bei $\sqrt{n}$ gefunden wird, also $a^2 \approx n$ ist, also im Fall $n = pq$, wenn die Differenz $|p - q|$ der Faktoren klein ist.

- Die schnellsten allgemein anwendbaren Faktorisierungverfahren –
  - Zahlkörpersieb (SILVERMAN 1987, POMERANCE 1988, A. K. LENSTRA/ H. W. LENSTRA/ MANASSE/ POLLARD 1990),
  - Elliptische-Kurven-Faktorisierung (H. W. LENSTRA 1987, ATKIN/ MORAIN 1993), –

  haben einen Zeitaufwand der Größenordnung

$$L_n := e^{\sqrt[3]{\ln n \cdot (\ln \ln n)^2}},$$

  sind also „subexponenziell“, aber noch „superpolynomiell“. *Insbesondere ist die Faktoriserung als Angriff auf das RSA-Verfahren* wesentlich *effizienter als die vollständige Suche.*

Daraus ergeben sich folgende Schätzungen:

Zahl	Bits	Dezimal-stellen	Aufwand (MIPS-Jahre)	Status
rsa120	399	120	100	auf PC $<$ 1 Woche
rsa140	466	140	2000	TE RIELE 1999
rsa154	512	154	100 000	TE RIELE 1999
	1024	308	10^{11}	nicht mehr sicher
	2048	616	10^{15}	kurzfristig sicher

bei denen zu beachten ist:

- Sie sind mit großer Unsicherheit behaftet,
- sie gelten nur, solange keine wesentlich schnelleren Faktorisierungsalgorithmen gefunden werden.

Insbesondere ist *bisher nicht bewiesen*, dass es nicht vielleicht doch einen polynomiellen Faktorisierungsalgorithmus gibt.

Neuere Entwicklungen (in der obigen Tabelle schon berücksichtigt) sind:

- Ein Artikel von A. K. Lenstra/ E. Verheul, *Selecting cryptographic key sizes*, der den Stand der Technik im Jahr 2000 zusammenfasst und extrapoliert. Die Abschätzungen wurden als zu pessimistisch bewertet, da der Speicherbedarf der Verfahren nicht berücksichtigt wurde.
- Ein Vorschlag von Bernstein, *Circuits for integer factorization*, der die Stellenzahl (!) verdreifacht, die bei festem Aufwand mit dem schnellsten Faktorisierungsverfahren erreichbar ist.
- Spezielle Hardware-Designs von Shamir und Mitarbeitern:
  - TWINKLE (The Weizmann Institute Key Locating Machine) – die Hardware-Umsetzung einer Idee von Lehmer aus den dreißiger Jahren, die das Faktorisieren um den Faktor 100 – 1000 beschleunigt (1999),
  - TWIRL (The Weizmann Institute Relation Locator) – der das Faktorisieren um einen weiteren Faktor 1000 – 10000 beschleunigt (2003) unter Berücksichtigung des Vorschlags von Bernstein,

  also insgesamt ein Faktor etwa 10^6 für das Zahlkörpersieb, ohne allerdings die Größenordnung L_n der Komplexität zu ändern.

Insbesondere sieht die Abschätzung von Lenstra/ Verheul jetzt eher zu optimistisch aus. 1024-*Bit-Schlüssel sollten schnellstens aus dem Verkehr gezogen werden.* 2048-Bit-Schlüssel sind gerade noch für ein paar Jahre als sicher zu betrachten.

Empfehlung: Die Primzahlen p und q, aus denen der RSA-Modul $n = pq$ konstruiert wird, sollen mit mindestens 1024 Bit Länge gewählt werden, und zwar so, dass auch ihre Differenz $|p-q|$ eine Länge in der Größenordnung 1024 Bit hat.

2.5 Iterationsangriff

Sei allgemein $E\colon M \longrightarrow M$ eine bijektive Abbildung der endlichen Menge M und $D = E^{-1}$ die Umkehrabbildung; wir stellen uns E als Verschlüsselungsfunktion vor.

Dann ist E in der Permutationsgruppe $\mathfrak{S}(M)$ enthalten, die die (riesige) Ordnung $\#\mathfrak{S}(M) = (\#M)!$ hat. Immerhin ist sie endlich, und somit gibt es ein $s \in \mathbb{N}_1$ mit $E^s = \mathbf{1}_M$, also

$$D = E^{s-1}.$$

Also ist D aus E bestimmbar durch hinreichend häufige Iteration – ein Angriff, der natürlich nur für asymmetrische Verfahren relevant ist. Um sich davor zu schützen, muss man *die Ordnung von E – das kleinste $s \geq 1$ mit $E^s = \mathbf{1}_M$ – möglichst groß wählen.*

Das Beispiel RSA

Hier ist $M = C = \mathbb{Z}/n\mathbb{Z}$, also $\#\mathfrak{S}(M) = n!$, wobei n schon eine sehr große Zahl ist, so dass noch nicht unmittelbar eine Gefahr zu sehen ist – der Angreifer könnte zwar $E^{n!-1}$ bilden, aber das wird er selbst mit dem effizientesten Potenzalgorithmus in diesem Universum nicht schaffen.

Allerdings sind die RSA-Verschlüsselungsfunktionen in einer wesentlich kleineren Untergruppe von $\mathfrak{S}(M)$ enthalten – deren Ordnung der Angreifer glücklicherweise nicht kennt:

Um das herzuleiten, betrachten wir die Abbildung

$$\Phi : \mathbb{N} \longrightarrow \mathrm{Abb}(M, M), \quad e \mapsto E_e \quad \text{mit } E_e(a) = a^e \bmod n.$$

Sie hat die folgenden Eigenschaften:

Bemerkungen

1. Für $e, f \in \mathbb{N}$ ist $E_{ef} = E_e \circ E_f$, weil $a^{ef} \equiv (a^f)^e \pmod n$ für alle $a \in M$. Also ist Φ Homomorphismus der multiplikativen Halbgruppe $\mathbb{N}$.

2. Ist $e \equiv f \pmod{\lambda(n)}$, so ist $E_e = E_f$: Wenn $f = e + k\lambda(n)$, folgt $a^f = a^{e+k\lambda(n)} \equiv a^e \pmod n$ für alle $a \in M$.

3. Ist e mod $\lambda(n)$ invertierbar, so ist E_e bijektiv: Ist $de \equiv 1 \pmod{\lambda(n)}$, so $E_d \circ E_e = E_1 = \mathbf{1}_M$. Also ist die von Φ induzierte Abbildung,

   $$\bar{\Phi} : \mathbb{M}_{\lambda(n)} \longrightarrow \mathfrak{S}(M),$$

   ein Gruppen-Homomorphismus.

4. $\bar{\Phi}$ ist injektiv: Ist nämlich $\Phi(e) = E_e = \mathbf{1}_M$, so ist $a^e \equiv a \pmod{n}$ für alle $a \in M$, also $a^{e-1} \equiv 1 \pmod{n}$ für alle $a \in \mathbb{M}_n$, also $\lambda(n)|e-1$, also $e \equiv 1 \pmod{\lambda(n)}$.

Damit ist bewiesen:

Satz 2 *Die RSA-Verschlüsselungsfunktionen E_e bilden eine zu $\mathbb{M}_{\lambda(n)}$ isomorphe Untergruppe $H \leq \mathfrak{S}(M)$ von der Ordnung $\varphi(\lambda(n))$ und vom Exponenten $\lambda(\lambda(n))$.*

Die Ordnung einer einzelnen Verschlüsselungsfunktion E_e kann natürlich noch viel kleiner sein; die zyklische Untergruppe $\langle e \rangle \leq \mathbb{M}_{\lambda(n)}$ hat die Ordnung $s := \mathrm{Ord}(e)|\lambda(\lambda(n))$.

Damit sind wir auf folgende Probleme gestossen:

1. Wie groß ist $\lambda(\lambda(n))$?

Antwort (ohne Beweis): „Im allgemeinen“ ist $\lambda(\lambda(n)) \approx \frac{n}{4}$.

Will man sich dessen sicher sein, wählt man p, q speziell, d. h., $p = 2p'+1$, $q = 2q'+1$ mit verschiedenen Primzahlen $p', q' \geq 3$. (Solche Primzahlen p', q' heißen Germain-**Primzahlen** nach Sophie Germain, die durch Betrachtung dieser Zahlen einen wesentlichen Fortschritt für das Fermat-Problem erzielt hatte.) Für $n = pq$ ist dann

$$\lambda(n) = \mathrm{kgV}(2p', 2q') = 2p'q' \approx \frac{n}{2}.$$

Sind weiterhin auch $p' = 2p'' + 1$ und $q' = 2q'' + 1$ speziell, so ist

$$\lambda(\lambda(n)) = 2p''q'' \approx \frac{n}{4}.$$

Der Primzahlsatz lässt erwarten, dass es auch solche Zahlen noch in astronomischen Mengen gibt.

2. Wann ist $\mathrm{Ord}(e) = \lambda(\lambda(n))$? Oder nicht wesentlich kleiner?

Antwort: meistens. (Auch hier existiert eine mathematische Analyse.)

Als Folgerung kann man festhalten: *Bis auf vernachlässigbar unwahrscheinliche Ausnahmen ist s von der gleichen Größenordnung wie n.*

Ergänzend zu Abschnitt 2.2 lässt sich jetzt die Aufgabe

(F) Finden der Ordnung s der Verschlüsselungsfunktion

hinzufügen. Es gilt die komplexitätstheoretische Implikation

$$\text{(F)} \longrightarrow \text{(A)}$$

(wenn die Ordnung s bekannt ist, ist $D = E^{s-1}$ und daher auch $d = e^{s-1}$ bekannt), aber vielleicht nicht die Umkehrung. *Die Ordnung der Verschlüsselungsfunktion zu finden, ist mindestens so schwierig das Faktorisieren des Moduls.*

2.6 Brechen *eines* Geheimtextes

Das Brechen von Geheimtexten könnte aber noch leichter sein: Für einen gegebenen Geheimtext c könnte nämlich $E_e^r(c) = c$ sein, obwohl $E_e^r \neq \mathbf{1}_M$ ist. Ist dann a der Klartext, also $c = E_e(a)$, so kann der Kryptoanalytiker berechnen:

$$E_e^{r-1}(c) = D_e(E_e^r(c)) = D_e(c) = a.$$

Die mathematische Beschreibung dieser Situation sieht so aus:

- $\mathbb{M}_{\lambda(n)}$ operiert auf der Menge $M = \mathbb{Z}/n\mathbb{Z}$, ebenso die zyklische Untergruppe $G := \langle e \rangle \leq \mathbb{M}_{\lambda(n)}$.
- Für $a \in M$ ist $G \cdot a = \{a^{e^k} \mid 0 \leq k < s\}$ die Bahn.
- Der Stabilisator $G_a = \{f \in G \mid a^f \equiv a \pmod{n}\}$ ist Untergruppe von G; zwischen den Mengen $G \cdot a$ und G/G_a gibt es eine natürliche Bijektion.
- Für die Bahnlänge $t = \#G \cdot a$ gilt

$$t = \frac{s}{\#G_a}, \qquad t|s|\lambda(\lambda(n))$$
$$E_e^r(c) = c \iff E_e^r(a) = a \iff t|r.$$

- $G \cdot c = G \cdot a$ und $\#G_c = \#G_a$. (Die beiden Stabilisatoren sind zueinander konjugiert.)

Damit sind wir auf ein weiteres Problem gestoßen:

3. Wann ist $t = s$, d. h., der Stabilisator G_a trivial? Oder zumindest sehr klein?

   Antwort auch hier: meistens.

Das Finden der Bahnlänge t von a und c ist also mindestens so schwierig wie das Brechen des Geheimtextes c.

Zwei neuere Artikel zeigen, wie gering das Risiko ist, versehentlich eine kleine Bahnlänge zu erwischen und somit den Iterationsangriff zu ermöglichen:

- J. J. Brennan/ Bruce Geist, Analysis of iterated modular exponentiation: The orbits of x^α mod N. **Designs, Codes and Cryptography** 13 (1998), 229–245.
- John B. Friedlander/ Carl Pomerance/ Igor E. Shparlinski, Period of the power generator and small values of Carmichael's function. **Mathematics of Computation** xx (2003), yyy–zzz.

2.7 Mehrfach-Verwendung eines Moduls

Frage: Was passiert, wenn beim RSA-Verfahren zwei verschiedene Teilnehmer denselben Modul n verwenden?

D. h., A und B haben die öffentlichen Schlüssel (n, e_A) und (n, e_B).

Zunächst sind offensichtlich A und B voreinander nicht sicher, da beide n faktorisieren, also auch den geheimen Schlüssel des jeweils anderen bestimmen können. Ein gemeinsamer Modul ist also höchstens in einer Kooperationssituation sinnvoll, wo A und B einander uneingeschränkt vertrauen.

Es kommt aber noch schlimmer: Falls jemand die gleiche Nachricht a an A und B schickt, kann *jeder* diese entschlüsseln. Die Geheimtexte sind:

$$c_\mathrm{A} = a^{e_\mathrm{A}} \bmod n, \quad c_\mathrm{B} = a^{e_\mathrm{B}} \bmod n.$$

Unter der nicht wesentlich einschränkenden Annahme, dass e_A und e_B teilerfremd sind, findet eine Angreiferin mit dem erweiterten EUKLIDischen Algorithmus Koeffizienten x und y mit

$$xe_\mathrm{A} + ye_\mathrm{B} = 1;$$

dabei haben x und y notwendigerweise verschiedenes Vorzeichen, o. B. d. A. $x < 0$. Ist $\mathrm{ggT}(c_\mathrm{A}, n) > 1$, so kann die Angreiferin n faktorisieren und ist fertig. Andernfalls bestimmt sie

$$g := c_\mathrm{A}^{-1} \bmod n$$

durch Kongruenzdivision (also wieder mit dem EUKLIDischen Algorithmus) und hat dann

$$g^{-x} \cdot c_\mathrm{B}^{y} \equiv (a^{e_\mathrm{A}})^x \cdot (a^{e_\mathrm{B}})^y \equiv a \pmod{n}.$$

Die geheimen Schlüssel d_A und d_B hat sie damit allerdings nicht bestimmt.

Der gemeinsame Modul n ist also nur sicher, wenn A und B sich voll vertrauen und diesen Modul außerdem geheim halten. Dann können sie nur miteinander kommunizieren – und da ist es viel effizienter, gleich eine symmetrische Chiffre zu verwenden.

2.8 Kleine Exponenten

Frage: Ist es beim RSA-Verfahren gefährlich, einen kleinen öffentlichen Exponenten e zu wählen?

Der Exponent $e = 1$ ist völlig unbrauchbar, da dann nicht verschlüsselt wird.

Der Exponent $e = 2$ ist für RSA nicht verwendbar, da er gerade und somit nicht teilerfremd zu $\lambda(n)$ ist. Es gibt allerdings eine verwandte Chiffre, das RABIN-Verfahren, das gerade mit $e = 2$ so arbeitet. Bei diesem muss der Empfänger Quadratwurzeln $\bmod n$ ziehen können, und das kann er, wenn er n faktorisieren kann (siehe später). (Er muss auch noch eine Möglichkeit haben, unter den verschiedenen Quadratwurzeln den richtigen Klartext zu erkennen.)

Der kleinste für das RSA-Verfahren prinzipiell geeignete Exponent ist $e = 3$. Hier entsteht eine Schwäche, sobald jemand die gleiche Nachricht a an *drei* verschiedene Teilnehmer A, B und C schickt. Deren öffentliche Schlüssel seien $(n_\mathrm{A}, 3)$, $(n_\mathrm{B}, 3)$ und $(n_\mathrm{C}, 3)$. O. B. d. A. sind die Moduln n_A, n_B und n_C paarweise teilerfremd, sonst kann die Angreiferin mindestens zwei davon faktorisieren, also erst recht a lesen. Ansonsten hat sie (mit etwas Glück) drei Geheimtexte

$$c_\mathrm{A} = a^3 \bmod n_\mathrm{A}, \quad c_\mathrm{B} = a^3 \bmod n_\mathrm{B}, \quad c_\mathrm{C} = a^3 \bmod n_\mathrm{C}$$

mit $0 \le a < n_\mathrm{A}, n_\mathrm{B}, n_\mathrm{C}$, also $a^3 < n_\mathrm{A} n_\mathrm{B} n_\mathrm{C}$. Mit dem chinesischen Restalgorithmus konstruiert sie daraus eine Zahl $\tilde{c} \in \mathbb{Z}$ mit

$$0 \le \tilde{c} < n_\mathrm{A} n_\mathrm{B} n_\mathrm{C}$$

und

$$\tilde{c} \equiv c_\mathrm{X} \quad \text{für X} = \mathrm{A}, \mathrm{B}, \mathrm{C}.$$

Wegen der Eindeutigkeit ist $\tilde{c} = a^3$ in $\mathbb{Z}$. Damit bestimmt sie $a = \sqrt[3]{\tilde{c}}$ durch Wurzelziehen in $\mathbb{Z}$. (Auch hier gelingt es ihr nicht, die geheimen Exponenten zu bestimmen.)

Die Verallgemeinerung dieses Angriffs auf einen beliebigen „kleinen“ gemeinsamen öffentlichen Exponenten ist offensichtlich: Wenn eine identische Nachricht an e verschiedene Teilnehmer geschickt wird, kann diese von jedem gelesen werden. So abwegig ist dieser Angriffspunkt nicht, wenn man sich etwa eine konstante „Protokollinformation“ am Beginn längerer Nachrichten vorstellt. In der Praxis wird der Exponent $e = 2^{16} + 1 = 65537$ meist als hinreichend sicher für „normale“ Anwendungen angesehen.

2.9 Die Signaturfalle

Ein Problem, das nicht die Sicherheit der Chiffre RSA, sondern die Rahmenbedingungen ihrer Anwendung betrifft, ist die Signaturfalle: Da RSA in umgekehrter Reihenfolge zur digitalen Signatur verwendet wird, muss man stets darauf achten, dass man nicht im Glauben, einen Text digital zu signieren, in Wirklichkeit einen vorgelegten Geheimtext entschlüsselt. Würden wirklich normlerweise Klartexte signiert, würde das sofort ins Auge springen. Es gibt aber (mindestens) drei Gründe, warum das nicht so ist:

1. Bei der digitalen Signatur wird aus Performanzgründen so gut wie immer ein (kryptographischer) Hash-Wert der Nachricht signiert. Dieser ist von einer Zufalls-Bitkette nicht zu unterscheiden.

2. Bei der starken Authentisierung wird statt einer Passworteingabe als Identitätsnachweis eine zufällige Bitkette digital signiert. Selbst wenn das Ergebnis ein entschlüsselter Klartext wäre – man sieht es in der Regel gar nicht, es wird direkt dem Kommunikationspartner (z. B. Zielrechner) übersandt.

3. Außerdem kann ein beliebiger Text durch „Camouflage“ vorverschlüsselt zum Signieren bzw. Entschlüsseln vorgelegt werden. Selbst wenn man sich das Ergebnis dieses Vorgangs ansieht, erkannt man nicht, dass man in Wirklichkeit entschlüsselt hat – siehe unten. Diese Eigenschaft des RSA-Verfahrens ist sogar sehr nützlich: Sie ist Grundlage der blinden Signatur und damit der Erzeugung von digitalen Pseudonymen und anonymen Berechtigungsnachweisen. Siehe
   `http://www.uni-mainz.de/~pommeren/DSVorlesung/KryptoProt/`

Es handelt sich hierbei übrigens um einen *Angriff mit gewähltem Geheimtext*. Dieses Problem wird in der Praxis dadurch umgangen, dass man für die drei – evtl. vier – Funktionen

- Chiffrierung,
- digitale Signatur,
- starke Authentisierung,
- evtl. blinde Signatur

jeweils separate Schlüsselpaare erzeugt und verwendet.

Nun also zu der „Camouflage“, die zur Verschleierung des Angriffs mit gewähltem Geheimtext dient. Der Ablauf ist:

1. Der Angreifer M hat einen Geheimtext $x = E_{\mathrm{A}}(a)$ aufgefangen. Er verschlüsselt ihn mit einer nur ihm bekannten Funktion C zu $y = C(x)$.

2. Er schiebt y dem Opfer A zur digitalen Signatur unter; dieses erzeugt $z = D_{\mathrm{A}}(y)$.

3. Der Angreifer entfernt die „Camouflage“ durch eine geeignete Rücktransformation C'. Hat er ein Paar (C, C') zur Verfügung, so dass $C' \circ D_{\mathrm{A}} \circ C = D_{\mathrm{A}}$, so ist $a = D_{\mathrm{A}}(x) = C'(z)$.

Eine Besonderheit des RSA-Verfahrens ist, dass ein solches Paar (C, C') von Transformationen existiert; ist $E_{\mathrm{A}}(a) = a^e \bmod n$, so ist C die Verschiebechiffre auf $\mathbb{M}_n = (\mathbb{Z}/n\mathbb{Z})^\times$ mit u^e und C' die Multiplikation mit $u^{-1} \bmod n$, wobei $u \in \mathbb{M}_n$ zufällig gewählt wird. Der Ablauf des Angriffs ist also:

1. M wählt u und bildet $y = C(x) = u^e x \bmod n$.

2. A erzeugt $z = y^d \bmod n$.

3. M berechnet $C'(z) = zu^{-1} = y^d u^{-1} = u^{ed} x^d u^{-1} = x^d = a$ in $\mathbb{Z}/n\mathbb{Z}$.

2.10 Weitere Angriffe

Einige weitere Angriffsansätze werden hier nur überblicksartig erwähnt; für eine ausführlichere Behandlung sei auf den in der Einleitung zitierten Artikel von D. Boneh verwiesen:

1. **Kleine private Exponenten.** M. Wiener hat eine Möglichkeit entdeckt, mit Hilfe eines Kettenbruch-Algorithmus den privaten Schlüssel d aus dem öffentlichen Schlüssel (n, e) effizient zu bestimmen, wenn $d < \frac{1}{3} \cdot \sqrt[4]{n}$.

2. **Abhängige Klartexte** nach Franklin/ Reiter. Gibt es einen affinen Zusammenhang $a_2 = sa_1 + t$ zwischen zwei verschiedenen Klartexten a_1 und a_2 mit bekannten Koeffizienten s und $t \neq 0$, so sind die Klartexte effizient aus dem öffentlichen Schlüssel (n, e), den Koeffizienten s und t sowie den zugehörigen Geheimtexten bestimmbar. Coppersmith hat gezeigt, wie man einen solchen affinen Zusammenhang antreffen kann, wenn a_1 und a_2 aus demselben Klartext durch unterschiedliches „Padding“ entstehen.

3. **Partielles Leck** nach Boneh/ Durfee/ Frankel/ Coppersmith. Falls das letzte Viertel der Bits von einer der Zahlen d (dem privaten Schlüssel) p oder q (den Primfaktoren des Moduls) bekannt sind, ist n effizient faktorisierbar.

4. **Timing- und Power-Attacken** nach Kocher. Hier wird der Prozessor beim Verschlüsseln beobachtet und aus seinem Zeitbedarf oder Stromverbrauch auf die Bits des geheimen Schlüssels geschlossen; dieser Angriff beruht darauf, dass diese Werte in den Ressourcenverbrauch des binären Potenzalgorithmus eingehen.

5. **Differenzielle Fehleranalyse** nach Shamir u. a. Hier wird der Prozessor (etwa auf einer Chipkarte) ein wenig über den spezifizierten Bereich seines einwandfreien Funktionierens gebracht – etwa durch Verbiegen, Erwärmen, Bestrahlen. Aus einzelnen Bitfehlern werden statistische Aussagen über die internen Parameter gewonnen.

Eine Reihe weiterer Angriffe sind bekannt, die sich nicht gegen das RSA-Verfahren direkt richten, sondern gegen Fehler bei der Implementation, Verwendung in kryptographischen Protokollen, Einbindung in eine Systemumgebung u. ä.

3 Primzahltests

Eine Frage ist zur Durchführbarkeit des RSA-Verfahrens noch zu klären: Gibt es überhaupt Möglichkeiten, die für die Schlüsselerzeugung nötigen Primzahlen zu finden? Die Antwort wird lauten: Ja, das ist effizient möglich. Man startet mit einer zufälligen Zahl der benötigten Bitlänge und prüft, ob sie eine Primzahl ist. Falls nein, prüft man die nächstgrößere Zahl usw., bis man eine Primzahl gefunden hat.

Nötig sind also Verfahren, die effizient entscheiden, ob eine natürliche Zahl prim ist – sogenannte Primzahltests.

Hierbei tritt ein Phänomen auf, das man auch von anderen mathematischen Problemen (z. B. lineare Optimierung, Bestimmung von Polynomnullstellen) kennt:

- Es gibt einen Algorithmus, der mit polynomialem Aufwand auskommt.
- Es gibt einen „Standard-Algorithmus“ (in den Beispielen: Simplex-Methode, Newton-Verfahren), der in den „meisten“ Fällen deutlich effizienter ist, im schlechtesten Fall („worst case“) allerdings nicht mit polynomialem Aufwand auskommt. Dieser wird in der Praxis bevorzugt.

Bei den Primzahltests ist der neu entdeckte AKS-Algorithmus polynomial, kann aber mit dem etablierten Rabin-Algorithmus nicht mithalten, der sehr effizient ist, aber im schlechtesten Fall nicht einmal das richtige Ergebnis liefert.

Da alle genannten Primzahltests einen beachtlichen Overhead haben, wird man in der praktischen Anwendung stets zuvor die Teilbarkeit durch „kleine“ Primzahlen prüfen, etwa durch Primzahlen $< 10^6$, je nach verfügbarem Speicherplatz: Man legt nämlich dazu eine Liste L dieser Primzahlen an.

Benötigt man nun eine zufällig gewählte Primzahl einer bestimmten Größe, so wählt man zufällig eine Zahl r in diesem Größenbereich – sollte r gerade sein, erhöht man um 1. Dann siebt man nach dem Eratosthenes-Verfahren das Intervall $[r, r+s]$ nach den Vielfachen der Primzahlen in L aus; damit die Chance, noch etwas übrig zu behalten, hinreichend groß ist, sollte man s als deutlich größer als die größte Zahl in der Liste L wählen. Die Zahlen die übrig bleiben, werden der Reihe nach dem gewünschten Primzahltest unterzogen, bis man eine gefunden hat, die den Test besteht.

3.1 Der Pseudoprimzahltest

Woran erkennt man, dass eine Zahl prim ist? Der „naive“ Ansatz, Probedivisionen durch alle Zahle $\leq \sqrt{n}$ durchzuführen – perfektioniert im Sieb des ERATOSTHENES –, ist nicht effizient, da $\sqrt{n} = \exp(\frac{1}{2}\log(n))$ immer noch exponentiell mit der Stellenzahl $\log n$ von n wächst.

Einen Ansatz, Primzahlen ohne Probedivision zu erkennen, bietet der Satz von FERMAT: Ist n prim, so $a^{n-1} \equiv 1 \pmod{n}$ für alle $a = 1, \ldots, n-1$. Umgekehrt sagt man, dass n den **Pseudoprimzahltest zur Basis** a besteht, wenn $a^{n-1} \equiv 1 \pmod{n}$. Eine Primzahl besteht diesen Test also zu jeder Basis $a = 1, \ldots, n-1$. Die Kongruenz $2^{14} \equiv 4 \pmod{15}$ beweist, dass 15 nicht prim ist. Allerdings ist $2^{340} \equiv 1 \pmod{341}$, obwohl $341 = 11 \cdot 31$; aber immerhin ist $3^{340} \equiv 56 \pmod{341}$, so dass 341 durch den Pseudoprimzahltest zur Basis 3 fällt.

Trotzdem reicht dieses Kriterium nicht, um umgekehrt die Primzahleigenschaft zu beweisen. Man nennt n CARMICHAEL**-Zahl**, wenn n den Pseudoprimzahltest zu jeder zu n teilerfremden Basis a besteht, aber nicht prim ist.

Den Pseudoprimzahltest kann man auch dadurch ausdrücken, dass die Ordnung von a in $\mathbb{M}_n$ ein Teiler von $n-1$ ist. Also ist n genau dann CARMICHAEL-Zahl oder prim, wenn $\lambda(n) \mid n-1$ für die CARMICHAEL-Funktion λ. Es gibt zu viele CARMICHAEL-Zahlen, als dass der Pseudoprimzahltest nützlich sein könnte. Insbesondere haben ALFORD, GRANVILLE und POMERANCE 1992 bewiesen, dass es unendlich viele CARMICHAEL-Zahlen gibt.

Die kleinste CARMICHAEL-Zahl ist $561 = 3 \cdot 11 \cdot 17$; das folgt leicht aus dem nächsten Satz.

Satz 1 *Eine natürliche Zahl n ist genau dann* CARMICHAEL*-Zahl, wenn sie zuammengesetzt und quadratfrei ist, und $p-1 \mid n-1$ für jeden Primteiler p von n. Eine ungerade* CARMICHAEL*-Zahl hat mindestens 3 Primfaktoren.*

Beweis. „$\Longrightarrow$“: Wäre $p^2 | n$, so enthielte $\mathbb{M}_n$ eine zu $\mathbb{M}_{p^e}$ mit geeignetem $e \geq 2$ isomorphe Untergruppe, also auch eine zyklische Gruppe der Ordnung p; also wäre $p \mid n-1$, Widerspruch. Da aber $\mathbb{M}_n$ eine zyklische Gruppe der Ordnung $p-1$ enthält, gibt es ein Element a der Ordnung $p-1$, und $a^{n-1} \equiv 1 \pmod{n}$, also $p-1 \mid n-1$.

„$\Longleftarrow$“: Da n quadratfrei ist, ist nach dem chinesischen Restsatz die multiplikative Gruppe $\mathbb{M}_n$ das direkte Produkt der zyklischen Gruppen $\mathbb{F}_p^\times$, wobei p die Primteiler von n durchläuft. Da stets $p-1 \mid n-1$, hat jedes Element von $\mathbb{M}_n$ eine Ordnung, die $n-1$ teilt.

Zusatz: Angenommen, $n = pq$ mit zwei Primzahlen p und q, etwa $p < q$. Dann ist $q-1 \mid n-1 = pq-1$, also $p-1 \equiv pq-1 \equiv 0 \pmod{q-1}$, Widerspruch. $\diamond$

3.2 Der strenge Pseudoprimzahltest

Man kann den Pseudoprimzahltest verschärfen, indem man weitere Kennzeichen für Primzahlen auswertet. Sei n zunächst ungerade und zusammengesetzt. Dann gibt es außer ± 1 auch nichttriviale Quadratwurzeln aus 1 in $\mathbb{Z}/n\mathbb{Z}$; findet man eine solche, so hat man nachgewiesen, dass n zusammengesetzt ist. Aber wie soll man nichttriviale Einheitsquadratwurzeln finden, wenn man die Primzerlegung von n nicht kennt? Dazu wird, der Idee aus 2.2 folgend, $n-1$ aufgespalten in $n-1 = 2^s \cdot r$ mit ungeradem r.

Sei $a \in \mathbb{M}_n$. Falls n den Pseudoprimzahltest zur Basis a nicht besteht, ist es als zusammengesetzt erkannt. Andernfalls hat a in der multiplikativen Gruppe $\mathbb{M}_n$ eine Ordnung $\mathrm{Ord}(a) \mid n-1$. In der Folge

$$a^r \bmod n, \quad a^{2r} \bmod n, \quad \ldots, \quad a^{2^s r} \bmod n = 1$$

könnte bereits $a^r \equiv 1 \pmod{n}$ sein. Dann wird a verworfen, ohne dass eine Entscheidung über n getroffen wird. Andernfalls tritt die 1 erstmals an späterer Stelle auf; das davor stehende Element muß dann Einheitswurzel $\neq 1$ sein. Es kann -1 sein; auch dann wird a ohne Entscheidung verworfen. Andernfalls ist eine nichttriviale Einheitswurzel gefunden und n als zusammengesetzt erkannt. Ist dagegen n prim, so gibt es in dieser Situation stets ein k mit $0 \le k < s$, so dass

$$a^{2^k r} \equiv -1 \pmod{n}.$$

Sei nun n eine beliebige positive ganze Zahl, und $n-1$ habe die Zweierordnung s und den ungeraden Teil r. Dann sagt man, n bestehe den **strengen Pseudoprimzahltest zur Basis** a [nach Selfridge ca. 1975], wenn

$$a^r \equiv 1 \pmod{n} \quad \text{oder} \quad a^{2^k r} \equiv -1 \pmod{n} \quad \text{für ein } k = 0, \ldots, s-1.$$

Insbesondere gilt dann $a^{n-1} \equiv 1 \pmod{n}$.

Wir haben also die gleiche Situation wie in Abschnitt 2.3 mit $u = n-1$. Die dortige Menge

$$B_u = \bigcup_{t=0}^{s} \{w \in \mathbb{M}_n \mid w^{r \cdot 2^t} = 1, w^{r \cdot 2^{t-1}} = -1\}$$

ist jetzt genau die Menge der Basen, für die n den strengen Pseudoprimzahltest besteht, also die Eigenschaft ($\mathrm{E}_{n,u}$) hat. Diese Basen werden auch **Primzeugen** für n genannt.

Jede Primzahl besteht den strengen Pseudoprimzahltest zu jeder Basis, die kein Vielfaches dieser Primzahl ist. Die Carmichael-Zahl $n = 561$ fällt schon bei $a = 2$ durch: Es ist $n - 1 = 560 = 16 \cdot 35$,

$$\begin{aligned} 2^{35} &\equiv 263 \pmod{561}, & 2^{70} &\equiv 166 \pmod{561}, \\ 2^{140} &\equiv 67 \pmod{561}, & 2^{280} &\equiv 1 \pmod{561}. \end{aligned}$$

Also ist 561 als zusammengesetzt erkannt. Die kleinste zusammengesetzte Zahl, die den strengen Pseudoprimzahltest für 2, 3 und 5 besteht, ist $25326001 = 2251 \cdot 11251$. Die einzige zusammengesetzte Zahl $< 10^{11}$, die ihn für 2, 3, 5 und 7 besteht, ist 3 215 031 751. Das erweckt die Hoffnung, dass dieser Test zum Erkennen von Primzahlen tatsächlich geeignet ist.

Satz 2 *Sei $n \geq 3$ ungerade. Dann sind äquivalent:*

(i) *n ist prim.*

(ii) *n besteht den strengen Pseudoprimzahltest zu jeder Basis a, die kein Vielfaches von n ist.*

Beweis. „(i) $\Longrightarrow$ (ii)“ wurde oben gezeigt.

„(ii) $\Longrightarrow$ (i)“: Wegen Satz 1 ist n prim oder eine CARMICHAEL-Zahl, insbesondere $\lambda(n) \mid n - 1 = u$. Also ist der Hilfssatz in 2.3 anwendbar; da nach Voraussetzung $B_u = \mathbb{M}_n$, folgt also, dass n Primpotenz, insgesamt also prim ist. $\diamond$

Korollar 3 *Ist n nicht prim, so gibt es höchstens $\frac{\varphi(n)}{2}$ Basen $< n$, für die n nicht den strengen Pseudoprimzahltest besteht.*

Beweis. Falls n CARMICHAEL-Zahl ist, folgt das aus Satz 1 in 2.3. Andernfalls ist $A_u = \{w \in \mathbb{M}_n \mid w^{n-1} = 1\} < \mathbb{M}_n$ echte Untergruppe, und $B_u \subseteq A_u$. $\diamond$

Bei genauerem Hinsehen kann man sogar die Schranke $\frac{\varphi(n)}{4}$ von RABIN/MONIER herleiten.

3.3 Der Primzahltest von Miller

Wie ist das im vorigen Abschnitt entwickelte Kriterium, der strenge Pseudoprimzahltest zu einer oder genügend vielen Basen, praktisch verwertbar? Dazu ist zunächst der Algorithmus für eine Basis a zu formulieren und sein Aufwand zu bestimmen.

Da a^{n-1} sowieso nach dem binären Potenzalgorithmus berechnet wird, ist es effizienter, gleich die ganze Folge der Potenzen ab a^r zu bestimmen; dann ist der Aufwand nicht wesentlich größer als für den „schwachen“ Pseudoprimzahltest. Der strenge Pseudoprimzahltest zur Basis a sieht dann so aus:

Prozedur sPPT(a)
[Strenger Pseudoprimzahltest zu einer Basis a.]
Eingabeparameter:
n = die zu prüfende Zahl (ungerade ≥ 3),
s = Zweierordnung von $n-1$ (vorberechnet),
r = ungerader Teil von $n-1$ (vorberechnet),
a = Basis (im Bereich $[2 \ldots n-1]$).
Ausgabeparameter:
zus = ein Boolescher Wert mit der Bedeutung
TRUE: n ist sicher zusammengesetzt,
FALSE: die Prüfung gab kein definitives Ergebnis
[d. h., n ist strenge Pseudoprimzahl zur Basis a].
Anweisungen:
Bestimme $b = a^r \bmod n$.
Setze $k = 0$.
[Schleife: Am Eingang ist $b = a^{2^k r} \bmod n$;
die Boolesche Variable ‘Ende’, vorbesetzt mit FALSE, entscheidet über das nochmalige Durchlaufen der Schleife.]
Solange nicht Ende:
Falls $b = 1$: Setze Ende = TRUE;
falls $k = 0$, setze zus = FALSE,
sonst setze zus = TRUE. [1 ohne vorherige -1]
Falls $b = n-1$ und $k < s$:
Setze zus = FALSE, Ende = TRUE.
Falls $k = s$ und $b \neq 1$:
Setze zus = TRUE, Ende = TRUE.
In allen anderen Fällen $[k < s, b \neq 1, b \neq n-1]$
ersetze b durch $b^2 \bmod n$,
ersetze k durch $k+1$.

Der Aufwand läßt sich in Einzelschritte aufbrechen, in denen jeweils zwei Zahlen $\bmod n$ multipliziert werden. Zur Berechnung von $a^r \bmod n$ sind

höchstens $2 \cdot {}^2\log(r)$ solcher Schritte nötig. Bei den höchstens s Schleifendurchläufen wird noch je einmal quadriert. Da ${}^2\log(n-1) = s + {}^2\log(r)$, sind also insgesamt höchstens $2 \cdot {}^2\log(n)$ Quadrate $\bmod n$ zu bilden. Jedes solche Quadrat erfordert höchstens N^2 „primitive“ Ganzzahl-Multiplikationen, wobei N die Stellenzahl von n in der verwendeten Basis des Zahlsystems ist. Die Bestimmung von r bedeutet s Divisionen durch 2 und kann hier vernachlässigt werden. Der Gesamtaufwand ist also, grob geschätzt, $\mathrm{O}(\log(n)^3)$.

Der Primzahltest von MILLER ist nun einfach die Aneinanderreihung der strengen Pseudoprimzahltests zu den Basen $2, 3, 4, 5, \ldots$. Das sieht zunächst nicht effizient aus, denn wenn man tatsächlich eine Primzahl testet, muss man scheinbar alle Basen $< n$ durchlaufen. MILLER hat aber gezeigt, dass man mit entscheidend weniger auskommt – *vorausgesetzt, die erweiterte* RIEMANN*sche Vermutung ist wahr.* Hierzu im nächsten Abschnitt einige Erläuterungen ohne vollständige Beweise.

3.4 Die erweiterte Riemannsche Vermutung (ERH)

Ein **Charakter** mod n ist eine Funktion

$$\chi : \mathbb{Z} \longrightarrow \mathbb{C}$$

mit den Eigenschaften:

(i) χ hat die Periode n.

(ii) $\chi(xy) = \chi(x)\chi(y)$ für alle $x, y \in \mathbb{Z}$.

(iii) $\chi(x) = 0$ genau dann, wenn $\mathrm{ggT}(x, n) > 1$.

Die Charaktere mod n entsprechen kanonisch bijektiv genau den Gruppen-Homomorphismen

$$\bar{\chi} : \mathbb{M}_n \longrightarrow \mathbb{C}^\times.$$

Beispiele sind der **triviale Charakter** $\chi(a) = 1$ für alle zu n teilerfremden a und der aus der Theorie der quadratischen Reziprozität bekannte Jacobi-**Charakter** $\chi(a) = (\frac{a}{n})$.

Zu einem Charakter gehört eine **L-Funktion**, die durch die Dirichlet-Reihe

$$L_\chi(z) = \sum_{a=1}^{\infty} \frac{\chi(a)}{a^z}$$

definiert ist. Die Reihe konvergiert absolut und lokal gleichmäßig in der Halbebene $\{z \in \mathbb{C} \mid \mathrm{Re}(z) > 1\}$, weil $a^{i\cdot \mathrm{Im}(z)} = e^{i\cdot \ln(a)\cdot \mathrm{Im}(z)}$ den Betrag 1 hat, also

$$\left|\frac{\chi(a)}{a^z}\right| = \left|\frac{\chi(a)}{a^{\mathrm{Re}(z)} \cdot a^{i\cdot \mathrm{Im}(z)}}\right| = \frac{1}{a^{\mathrm{Re}(z)}} \qquad \text{oder } 0.$$

Sie läßt sich analytisch auf die rechte Halbebene $\mathrm{Re}(z) > 0$ fortsetzen und ist dort holomorph, außer im Fall des trivialen Charakters, wo 1 ein einfacher Pol ist. Die Funktion L_χ hat die Riemann-**Eigenschaft**, wenn sie innerhalb des Streifens $0 < \mathrm{Re}(z) \leq 1$ Nullstellen nur auf der Geraden $\mathrm{Re}(z) = \frac{1}{2}$ hat. Die Riemannsche Vermutung behauptet dies gerade für die Riemannsche Zeta-Funktion, die **erweiterte** Riemannsche **Vermutung** für alle L-Funktionen zu Charakteren mod n. Die Zeta-Funktion ist für $\mathrm{Re}(z) > 1$ definiert durch

$$\zeta(z) := \sum_{a=1}^{\infty} \frac{1}{a^z} = \prod_{p \text{ prim}} \frac{1}{1 - \frac{1}{p^z}},$$

wobei die zweite Gleichung die Produktformel von Euler ist. Für den trivialen Charakter χ_1 mod n gilt also:

$$L_{\chi_1}(z) = \sum_{\mathrm{ggT}(a,n)=1} \frac{1}{a^z} = \zeta(z) \cdot \prod_{p|n \text{ prim}} \left(1 - \frac{1}{p^z}\right);$$

diese L-Funktion hat in $\mathrm{Re}(z) > 0$ die gleichen Nullstellen wie ζ.

Satz 3 (ANKENEY/MONTGOMERY/BACH) *Für $c = 2/\ln(3)^2 = 1.65707\ldots$ gilt: Ist χ ein nichttrivialer Charakter* mod n*, dessen L-Funktion L_χ die* RIEMANN*-Eigenschaft hat, so gibt es eine Primzahl $p < c \cdot \ln(n)^2$ mit $\chi(p) \neq 1$.*

Der Beweis soll hier nicht geführt werden.

Korollar 1 *Es gelte die verallgemeinerte* RIEMANN*sche Vermutung. Sei $G < \mathbb{M}_n$ eine echte Untergruppe. Dann gibt es eine Primzahl p mit $p < c \cdot \ln(n)^2$, deren Restklasse* mod n *im Komplement $\mathbb{M}_n - G$ liegt.*

Beweis. Es gibt einen nichttrivialen Homomorphismus $\mathbb{M}_n/G \longrightarrow \mathbb{C}^\times$, also einen Charakter mod n mit $G \subseteq \operatorname{Kern}\chi \subseteq \mathbb{M}_n$. $\diamond$

Satz 4 (MILLER) *Die ungerade Zahl $n \geq 3$ bestehe den strengen Pseudoprimzahltest für alle primen Basen $a < c \cdot \ln(n)^2$ mit c wie in Satz 3, und die L-Funktion jedes Charakters für jeden Teiler von n habe die Riemann-Eigenschaft. Dann ist n prim.*

Beweis. Zuerst wird gezeigt, dass n quadratfrei ist. Angenommen $p^2 \mid n$ für eine Primzahl p. Die multiplikative Gruppe $\mathbb{M}_{p^2}$ ist zyklisch von der Ordnung $p(p-1)$; insbesondere ist der Homomorphismus

$$\mathbb{M}_{p^2} \longrightarrow \mathbb{M}_{p^2}, a \mapsto a^{p-1} \bmod p^2,$$

nichttrivial. Sein Bild ist eine Untergruppe $G < \mathbb{M}_{p^2}$ der Ordnung p, die zyklisch, also isomorph zur Gruppe der p-ten Einheitswurzeln in $\mathbb{C}$ ist. Die Zusammensetzung ergibt einen Charakter mod p^2, und Satz 3 ergibt eine Primzahl $a < c \cdot \ln(p^2)^2$ mit $a^{p-1} \not\equiv 1 \bmod p^2$. Die Ordnung von a in $\mathbb{M}_{p^2}$ teilt $p(p-1)$. Wäre $a^{n-1} \equiv 1 \bmod n$, so müsste die Ordnung auch $n-1$ teilen. Da p zu $n-1$ teilerfremd ist, müsste sie Teiler von $p-1$ sein, was sie ja gerade nicht ist. Also ist $a^{n-1} \not\equiv 1 \bmod n$, und das widerspricht nun wieder dem bestandenen Pseudoprimzahltest. Also ist n quadratfrei.

Jetzt wird gezeigt, dass n auch nicht zwei verschiedene Primfaktoren haben kann. Nehmen wir an, p und q seien zwei solche, o. B. d. A. $\nu_2(p-1) \geq \nu_2(q-1)$. [$\nu_2(x)$ der Exponent des Primfaktors 2 in x.] Sei

$$r = \begin{cases} p, & \text{falls } \nu_2(p-1) > \nu_2(q-1), \\ pq, & \text{falls } \nu_2(p-1) = \nu_2(q-1). \end{cases}$$

Wieder nach dem Satz 3 gibt es ein $a < c \cdot \ln(r)^2$ mit $(\frac{a}{r}) = -1$. Ist u der ungerade Teil von $n-1$ und $b = a^u$, so ist auch $(\frac{b}{r}) = -1$, insbesondere $b \neq 1$. Wegen des strengen Pseudoprimzahltests gibt es also ein k mit $b^{2^k} \equiv -1 \bmod n$. Dann hat also b in $\mathbb{M}_p$ und in $\mathbb{M}_q$ die Ordnung 2^{k+1}. Insbesondere ist $2^{k+1} \mid q-1$.

Falls nun $\nu_2(p-1) > \nu_2(q-1)$ ist, muss sogar $2^{k+1} \mid \frac{p-1}{2}$ sein. Daraus folgt im Widerspruch zum EULER-Kriterium $b^{(p-1)/2} \equiv 1 \pmod{p}$, aber $(\frac{b}{p}) = -1$.

Ist aber $\nu_2(p-1) = \nu_2(q-1)$, so $(\frac{b}{p})(\frac{b}{q}) = (\frac{b}{r}) = -1$; also o. B. d. A. $(\frac{b}{p}) = -1$, $(\frac{b}{q}) = 1$. Nach dem EULER-Kriterium ist $b^{(q-1)/2} \equiv 1 \pmod{q}$, also $2^{k+1} \mid \frac{q-1}{2}$, $k+2 \leq \nu_2(q-1) = \nu_2(p-1)$, also auch $b^{(p-1)/2} \equiv 1 \pmod{p}$, im Widerspruch zu $(\frac{b}{p}) = -1$. $\diamond$

Für den Primzahltest von Miller reicht es also, den strengen Pseudoprimzahltest für alle Primzahlen $a < c \cdot \ln(n)^2$ durchzuführen. Für eine 512-Bit-Zahl, also $n < 2^{512}$, reicht es, die 18698 Primzahlen < 208704 durchzuprobieren. Das dauert natürlich bei aller Effizienz seine Zeit. In der Praxis hat sich daher eine Modifikation dieses Tests durchgesetzt, die (in einem noch zu spezifizierenden Sinne) nicht ganz exakt, aber wesentlich schneller ist. Sie wird im nächsten Abschnitt behandelt.

3.5 Der probabilistische Primzahltest von Rabin

Grundlage ist die Übertragung einer Idee von Solovay und Strassen auf den Millerschen Test, die Rabin vorgeschlagen hat. Wählt man a zufällig in $[2 \dots n-1]$, so fällt n „in der Regel“ durch den strengen Pseudoprimzahltest zur Basis a, wenn es zusammengesetzt ist. Was heißt aber „in der Regel“? Wie groß ist die Wahrscheinlichkeit? Diese Frage wurde schon im Korollar zu Satz 2 beantwortet, wobei die schärfere Schranke $\frac{1}{4}$ ohne Beweis angegeben wurde.

Zu bemerken ist, dass die Schranke $\frac{1}{4}$ scharf ist. Das sieht man an den Zahlen der Form

$$n = (1+2t)(1+4t)$$

mit ungeradem t (sofern die Faktoren $p = 1+2t$ und $q = 1+4t$ prim sind – Beispiel: $t = 24969$, $p = 49939$, $q = 99877$). Dann ist $n-1 = 2r$ mit $r = 3t+4t^2$,

$$B_u = \{a \mid a^r \equiv 1 \pmod{n}\} \cup \{a \mid a^r \equiv -1 \pmod{n}\}.$$

Da $\mathrm{ggT}(r, p-1) = \mathrm{ggT}(3t+4t^2, 2t) = t = \mathrm{ggT}(r, q-1)$, hat jede dieser beiden Kongruenzen genau t^2 Lösungen. Also ist $\#B_u = 2t^2$,

$$\frac{\#B_u}{n-1} = \frac{2t^2}{2\cdot(3t+4t^2)} = \frac{t}{3+4t} = \frac{1}{4+\frac{3}{t}}.$$

Die Grenze $\frac{1}{4}$ wird allerdings für die meisten zusammengesetzten Zahlen längst nicht erreicht.

Allgemein sei eine Familie $(B_{(n)})_{n\geq 1}$ von Mengen $B_{(n)} \subseteq [1 \dots n-1]$ und ein $\varepsilon \in]0,1[$ gegeben mit

1. $B_{(n)} = [1 \dots n-1]$, wenn n prim,
2. $\#B_{(n)} \leq \varepsilon \cdot (n-1)$ für alle genügend großen ungeraden zusammengesetzten Zahlen n.

Ferner soll die Entscheidung, ob $a \in B_{(n)}$, für alle $a \in [1 \dots n-1]$ effizient möglich sein, also mit einem Aufwand, der höchstens polynomial mit $\ln(n)$ wächst. Dann gibt es einen zugehörigen (abstrakten) Pseudoprimzahltest:

1. Wähle $a \in [1 \dots n-1]$ zufällig.
2. Prüfe, ob $a \in B_{(n)}$.
3. Ausgabe:
   (a) Falls **nein**: n ist sicher zusammengesetzt.
   (b) Falls **ja**: n ist pseudoprim für a.

Der zugehörige **probabilistische Primzahltest** besteht aus der Aneinanderreihung von k solchen Pseudoprimzahltests mit unabhängig voneinander zufällig gewählten Basen a (die sich also auch wiederholen dürfen). Ist stets $a \in B_{(n)}$, so ist n fast sicher prim – man kann diesem Ergebnis die „Irrtumswahrscheinlichkeit“ δ zuweisen, die aber nicht $= \varepsilon^k$ ist, sondern sich so berechnet:

In der Menge der ungeraden Zahlen $< 2^r$, also mit höchstens r Bits sei X die Teilmenge der *zusammengesetzten* Zahlen und Y_k die Menge der Zahlen, die k unabhängige (abstrakte) Pseudoprimzahltests bestehen. Die Wahrscheinlichkeit, dass das einer zusammengesetzten Zahl gelingt, ist dann gegeben durch die bedingte Wahrscheinlichkeit $P(Y_k|X) \leq \varepsilon^k$. Für die praktische Anwendung interessanter ist allerdings die „umgekehrte“ Wahrscheinlichkeit $\delta = P(X|Y_k)$ dafür, dass eine Zahl n, die bestanden hat, trotzdem zusammengesetzt ist. Diese kann man mit Hilfe der Bayesschen Formel abschätzen:

$$P(X|Y_k) = \frac{P(X) \cdot P(Y_k|X)}{P(Y_k)} \leq \frac{P(Y_k|X)}{P(Y_k)} \leq \frac{1}{q} \cdot \varepsilon^k \leq r \cdot \ln(2) \cdot \varepsilon^k,$$

wobei die Dichte der Primzahlen nach dem Primzahlsatz eingeht:

$$P(Y_k) \geq P(\text{prim}) =: q \geq \frac{1}{r \cdot \ln(2)}$$

(die letzte Ungleichung ist sogar großzügig, da wir nur ungeade Zahlen betrachten). Die gesuchte „Irrtumswahrscheinlichkeit“ $\delta = P(X|Y_k)$ könnte also durchaus größer als ε^k sein. Man kann (und) sollte sie dadurch verringern, dass man die Grundmenge, in der man nach Primzahlen sucht, einschränkt und damit $P(Y_k)$ vergrößert, z. B. indem man vor Anwendung des Pseudoprimzahltests durch alle Primzahlen etwa $< 100r$ probeweise dividiert.

Beim **Primzahltest von** Rabin ist $B_{(n)}$ die Menge der Basen a, für die n den strengen Pseudoprimzahltest zur Basis a besteht, und $\varepsilon = \frac{1}{4}$. Übersteht n die 25-fache Anwendung, so ist es mit einer sehr kleinen Irrtumswahrscheinlichkeit prim. Es ist eher wahrscheinlich, dass die Berechnung wegen eines Hard- oder Software-Fehlers falsch ist, als dass der Algorithmus die Primzahl-Eigenschaft falsch schätzt. Knuth bezweifelt auch, dass ein veröffentlichter Beweis der erweiterten Riemannschen Vermutung jemals eine so hohe Glaubwürdigkeit haben kann. Dennoch ist es natürlich mathematisch unbefriedigend, nicht mit absoluter Sicherheit sagen zu können, dass wirklich eine Primzahl vorliegt.

Als weiterführende Literatur zur Frage, wie „gut“ ein probabilistischer Primzahltest ist, sei

- S. H. Kim/ C. Pomerance: The probability that a random probable prime is composite. Math Comp. 53 (1989), 721–741,

empfohlen.

3.6 RSA und Pseudoprimzahlen

Das für die Anwendbarkeit des RSA-Verfahrens grundlegende Problem, wie man Primzahlen findet, wird durch den probabilistischen RABIN-Test zwar hocheffizient, aber nicht restlos befriedigend gelöst: Was passiert, wenn man eine „falsche“ Primzahl erwischt?

Sei dazu $n = pq$ ein vermeintlicher RSA-Modul, für den p und q nicht notwendig Primzahlen – aber zueinander teilerfremd – sind. Bei der Konstruktion der Schlüssel d, e mit

$$de \equiv 1 \pmod{\tilde{\lambda}(n)}$$

werden dann statt der wahren Werte $\varphi(n)$ und $\lambda(n)$ für EULER- und CARMICHAEL-Funktion die möglicherweise davon abweichenden Werte

$$\tilde{\varphi}(n) := (p-1)(q-1), \quad \tilde{\lambda}(n) := \mathrm{kgV}(p-1, q-1)$$

verwendet.

Funktioniert das RSA-Verfahren noch? Sei $a \in \mathbb{Z}/n\mathbb{Z}$ ein Klartext. Der Fall $\mathrm{ggT}(a, n) > 1$ führt – wie auch sonst – zur Faktorisierung des Moduls und wird hier – wie auch sonst – wegen seiner extrem geringen Wahrscheinlichkeit ignoriert. Andernfalls ist $\mathrm{ggT}(a, n) = 1$, und zu fragen ist, ob

$$a^{de-1} \stackrel{?}{\equiv} 1 \pmod{n}$$

gilt. Nun, das ist nach dem chinesischen Restsatz genau dann der Fall, wenn

$$a^{de-1} \equiv 1 \pmod{p} \quad \text{und} \quad \pmod{q}$$

ist. Hinreichend dafür ist

$$a^{p-1} \equiv 1 \pmod{p} \quad \text{und} \quad a^{q-1} \equiv 1 \pmod{q};$$

d. h., eine Nachricht a wird höchstens dann nicht korrekt entschlüsselt, wenn p oder q nicht pseudoprim zur Basis a ist. Also:

- Verwendet man statt einem Primfaktor eine CARMICHAEL-Zahl, funktioniert das RSA-Verfahren trotzdem korrekt; allerdings ist die (extrem geringe) Wahrscheinlichkeit, durch einen Klartext a, der nicht zu n teilerfremd ist, zufällig den Modul n zu faktorisieren, geringfügig vergrößert.
- Andernfalls gibt es eine geringe Chance, dass eine Nachricht nicht korrekt entschlüsselt werden kann.

Aus diesem Grund werden bei vielen Implementierungen des RSA-Verfahrens nach der Schlüsselerzeugung – wenn der probabilistische Primzahltest von RABIN eingesetzt wird – ein paar Probever- und -entschlüsselungen durchgeführt. Geht dabei etwas schief, wird der Modul verworfen. Es ist nicht bekannt, ob dieser Fall schon einmal eingetreten ist.

3.7 Der AKS-Primzahltest

Die Frage, ob es einen deterministischen Primzahltest gibt, der **mit polynomialem Aufwand** auskommt, war bisher nur – durch MILLER – auf die erweiterte RIEMANNsche Vermutung zurückgeführt worden. Alle anderen bekannten Primzahltests benötigten einen höheren Aufwand oder waren probabilistisch. Im August 2002 überraschten drei Inder, Manindra AGRAWAL, Neeraj KAYAL und Nitin SAXENA, die Fachwelt mit einem vollständigen Beweis, der auf einem überraschend einfachen Algorithmus beruht. Dieser erhielt sofort den Namen „AKS-Primzahltest“.

Satz 5 (Grundkriterium) *Seien $a, n \in \mathbb{Z}$ teilerfremd, $n \geq 2$. Dann sind äquivalent:*

(i) *n ist prim.*

(ii) *$(X+a)^n \equiv X^n + a \pmod{n}$ im Polynomring $\mathbb{Z}[X]$.*

Beweis. Aus dem binomischen Lehrsatz folgt

$$(X+a)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} X^i$$

in $\mathbb{Z}[X]$.

(i) Ist n prim, so $n | \binom{n}{i}$ für $i = 1, \ldots, n-1$, also $(X+a)^n \equiv X^n + a^n \pmod{n}$, und nach dem Satz von FERMAT ist $a^n \equiv a \pmod{n}$.

(ii) Ist n dagegen zusammengesetzt, so wählt man einen Primfaktor $q | n$ und k mit $q^k | n$ und $q^{k+1} \nmid n$. Dann ist $q \neq n$ und

$$q^k \nmid \binom{n}{q} = \frac{n \cdots (n-q+1)}{1 \cdots q}.$$

Also hat $(X+a)^n$ bei X^q einen Koeffizienten $\neq 0$ in $\mathbb{Z}/n\mathbb{Z}$. $\diamond$

Bemerkungen

1. Der Blick auf das absolute Glied in (ii) zeigt, dass das Grundkriterium eine Verallgemeinerung des Satzes von FERMAT ist.

2. Sei $\mathfrak{q} := (n, X^r - 1) \trianglelefteq \mathbb{Z}[X]$ (Ideal im Polynomring) für $r \in \mathbb{N}$. Ist n prim, so $(X+a)^n \equiv X^n + a \pmod{\mathfrak{q}}$. Also ist gezeigt:

Korollar 1 *Ist n prim, so gilt im Polynomring $\mathbb{Z}[X]$*

$$(X+a)^n \equiv X^n + a \pmod{\mathfrak{q}}$$

für alle $a \in \mathbb{Z}$ mit $\mathrm{ggT}(a, n) = 1$ und alle $r \in \mathbb{N}$.

Die naive Anwendung des Grundkriteriums als Primzahltest würde mit dem binären Potenzalgorithmus etwa ${}^2\log n$ Multiplikationen von Polynomen in $\mathbb{Z}/n\mathbb{Z}[X]$ erfordern, die aber immer aufwendiger werden: Im letzten Schritt sind zwei Polynome vom Grad etwa $\frac{n}{2}$ zu multiplizieren, was einen Aufwand der Größenordnung n erfordert. Das Korollar beschränkt den Grad durch $r-1$, ist aber nicht hinreichend.

Der Kernpunkt des AKS-Algorithmus ist, dass man das Korollar im wesentlichen umkehren kann, wenn man genügend viele, aber insgesamt nur „wenige“ a bei einem geeigneten festen r durchprobiert:

Satz 6 (AKS-Kriterium, Version von H. W. Lenstra**)** *Sei n eine natürliche Zahl ≥ 2. Gegeben sei eine zu n teilerfremde Zahl $r \in \mathbb{N}$. Sei $q := \mathrm{Ord}_r\, n$ die Ordnung von n in der multiplikativen Gruppe $\mathbb{M}_r = (\mathbb{Z}/r\mathbb{Z})^\times$. Ferner sei gegeben eine natürliche Zahl $s \geq 1$ mit $\mathrm{ggT}(n,a) = 1$ für alle $a = 1, \ldots, s$ und*

$$\binom{\varphi(r)+s-1}{s} \geq n^{2d\cdot\lfloor\sqrt{\frac{\varphi(r)}{d}}\rfloor}$$

für jeden Teiler $d|\frac{\varphi(r)}{q}$. Für das Ideal $\mathfrak{q} = (n, X^r-1) \trianglelefteq \mathbb{Z}[X]$ gelte

$$(X+a)^n \equiv X^n + a \pmod{\mathfrak{q}} \quad \text{für alle } a = 1, \ldots s.$$

Dann ist n eine Primzahlpotenz.

Der Beweis (nach D. Bernstein) wird in einige Hilfssätze zerlegt.

Hilfssatz 1 *Für alle $a = 1, \ldots s$ und alle $i \in \mathbb{N}$ gilt:*

$$(X+a)^{n^i} \equiv X^{n^i} + a \pmod{\mathfrak{q}}.$$

Beweis. Das folgt durch Induktion über i, wenn man in

$$(X+a)^n = X^n + a + n\cdot f(X) + (X^r-1)\cdot g(X)$$

in $\mathbb{Z}[X]$ die Substitution $X \mapsto X^{n^i}$ ausführt:

$$(X+a)^{n^{i+1}} \equiv (X^{n^i}+a)^n = X^{n^i\cdot n} + a + n\cdot f(X^{n^i}) + (X^{n^i\cdot r}-1)\cdot g(X^{n^i})$$

$$\equiv X^{n^{i+1}} + a \pmod{\mathfrak{q}},$$

da $X^{n^i r} - 1 = (X^r)^{n^i} - 1 = (X^r-1)(X^{r\cdot(n^i-1)} + \cdots + X^r + 1)$ Vielfaches von $X^r - 1$ ist. $\diamond$

Sei jetzt $p|n$ ein Primteiler. *Ziel* ist zu zeigen, dass n eine Potenz von p ist.

Das Ideal $\mathfrak{q} = (n, X^r-1) \trianglelefteq \mathbb{Z}[X]$ wird vergrößert zu $\hat{\mathfrak{q}} := (p, X^r-1) \trianglelefteq \mathbb{Z}[X]$. Die Identität aus Hilfssatz 1 gilt dann auch mod $\hat{\mathfrak{q}}$, und es gilt sogar, da jetzt ja mod p gerechnet wird:

Korollar 2 *Für alle $a = 1, \dots s$ und alle $i, j \in \mathbb{N}$ gilt*

$$(X + a)^{n^i p^j} \equiv X^{n^i p^j} + a \pmod{\hat{\mathfrak{q}}}.$$

Sei $H := \langle n, p \rangle \leq \mathbb{M}_r$ die von den Restklassen $n \bmod r$ und $p \bmod r$ erzeugte Untergruppe. Sei

$$d := \#(\mathbb{M}_r / H) = \frac{\varphi(r)}{\#H}.$$

Da $q = \mathrm{Ord}_r\, n \,|\, \#H$, ist $d \,|\, \frac{\varphi(r)}{q}$; also erfüllt d die Voraussetzung von Satz 6. Ein vollständiges Repräsentantensystem $\{m_1, \dots, m_d\} \subseteq \mathbb{M}_r$ von $\mathbb{M}_r / H$ sei für den Rest des Beweises fest gewählt. Korollar 2 wird dann erweitert zu

Korollar 3 *Für alle $a = 1, \dots s$, alle $k = 1, \dots, d$ und alle $i, j \in \mathbb{N}$ gilt*

$$(X^{m_k} + a)^{n^i p^j} \equiv X^{m_k n^i p^j} + a \pmod{\hat{\mathfrak{q}}}.$$

Beweis. Nach dem gleichen Trick wie in Hilfssatz 1 wird $X \mapsto X^{m_k}$ in $\mathbb{Z}[X]$ substituiert:

$$(X + a)^{n^i p^j} = X^{n^i p^j} + a + p \cdot f(X) + (X^r - 1) \cdot g(X) \text{ in } \mathbb{Z}[X],$$

$$(X^{m_k} + a)^{n^i p^j} = X^{m_k n^i p^j} + a + p \cdot f(X^{m_k}) + (X^{m_k \cdot r} - 1) \cdot g(X^{m_k}),$$

und daraus folgt die Behauptung. $\diamond$

Für die Produkte $n^i p^j \in \mathbb{N}$ mit $0 \leq i, j \leq \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor$ gilt

$$1 \leq n^i p^j \leq n^{2 \cdot \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor}.$$

Es gibt $(\lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor + 1)^2 > \frac{\varphi(r)}{d}$ solcher Paare $(i, j) \in \mathbb{N}^2$, und alle $n^i p^j \bmod r$ liegen in der Untergruppe H mit $\#H = \frac{\varphi(r)}{d}$; also gibt es verschiedene $(i, j) \neq (h, l)$ mit

$$n^i p^j \equiv n^h p^l \pmod{r},$$

und dafür muss sogar $i \neq h$ sein – sonst wäre $p^j \equiv p^l \pmod{r}$, also $p | r$. Damit ist auch schon der erste Teil des folgenden Hilfssatzes gezeigt:

Hilfssatz 2 *Es gibt i, j, h, l mit $0 \leq i, j, h, l \leq \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor$ und $i \neq h$, so dass für $t := n^i p^j$, $u := n^h p^l$ die Kongruenz $t \equiv u \pmod{r}$ erfüllt ist, und $|t - u| \leq n^{2 \cdot \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor} - 1$. Damit gilt*

$$(X^{m_k} + a)^t \equiv (X^{m_k} + a)^u \pmod{\hat{\mathfrak{q}}}$$

für alle $a = 1, \dots, s$ und alle $k = 1, \dots d$.

Beweis. Die letzte Kongruenz folgt aus $X^t = X^{u+cr} \equiv X^u \pmod{X^r - 1}$, also

$$(X^{m_k} + a)^t \equiv X^{m_k t} + a \equiv X^{m_k u} + a \equiv (X^{m_k} + a)^u \pmod{\hat{\mathfrak{q}}},$$

für alle a und k. $\diamond$

Da r zu n teilerfremd und p ein Primteiler von n ist, hat $X^r - 1$ im algebraischen Abschluss von $\mathbb{F}_p$ keine mehrfachen Nulstellen, also r verschiedene Nullstellen, die r-ten Einheitswurzeln $\bmod p$. Diese bilden (als endliche multiplikative Untergruppe eines Körpers) eine zyklische Gruppe. Sei ζ ein erzeugendes Element davon, also eine primitive r-te Einheitswurzel. Es gibt einen irreduziblen Teiler $h \in \mathbb{F}_p[X]$ von $X^r - 1$ mit $h(\zeta) = 0$. Sei

$$K = \mathbb{F}_p[\zeta] \cong \mathbb{F}_p[X]/h\mathbb{F}_p[X] \cong \mathbb{Z}[X]/\hat{\hat{\mathfrak{q}}}$$

mit dem Ideal $\hat{\hat{\mathfrak{q}}} = (p, h) \trianglelefteq \mathbb{Z}[X]$. Wir haben also die aufsteigende Kette von Idealen

$$\mathfrak{q} = (n, X^r - 1) \hookrightarrow \hat{\mathfrak{q}} = (p, X^r - 1) \hookrightarrow \hat{\hat{\mathfrak{q}}} = (p, h) \trianglelefteq \mathbb{Z}[X]$$

und umgekehrt die Kette von Surjektionen

$$\mathbb{Z}[X] \longrightarrow \mathbb{Z}[X]/\mathfrak{q} \longrightarrow \mathbb{F}_p[X]/(X^r - 1) \longrightarrow K = \mathbb{F}_p[\zeta] \cong \mathbb{F}_p[X]/h\mathbb{F}_p[X].$$

Hilfssatz 3 *In K gilt:*

(i) $(\zeta^{m_k} + a)^t = (\zeta^{m_k} + a)^u$ *für alle* $a = 1, \ldots, s$ *und alle* $k = 1, \ldots d$.

(ii) *Ist* $G \leq K^\times$ *die von den* $\zeta^{m_k} + a \neq 0$ *erzeugte Untergruppe, so gilt* $g^t = g^u$ *für alle* $g \in \bar{G} := G \cup \{0\}$.

Beweis. (i) folgt aus Hilfssatz 2 mit dem Homomorphismus $\mathbb{Z}[X] \longrightarrow K$, $X \mapsto \zeta$, der den Kern $\hat{\hat{\mathfrak{q}}} \supseteq \hat{\mathfrak{q}}$ hat.

(ii) folgt direkt aus (i). $\diamond$

Die $X + a \in \mathbb{F}_p[X]$ für $a = 1, \ldots s$ sind paarweise verschiedene irreduzible Polynome, da $p > s$ nach der Voraussetzung von Satz 6. Also sind auch alle Produkte

$$f_e := \prod_{a=1}^{s} (X + a)^{e_a} \quad \text{für } e = (e_1, \ldots, e_s) \in \mathbb{N}^s$$

in $\mathbb{F}_p[X]$ verschieden. Was passiert bei der Abbildung

$$\begin{array}{rcl} \Phi\colon \mathbb{F}_p[X] & \longrightarrow & K^d, \\ f & \mapsto & (f(\zeta^{m_1}), \ldots, f(\zeta^{m_d})), \end{array}$$

mit den Polynomen f_e ?

Hilfssatz 4 *Für die f_e mit* $\operatorname{Grad} f_e = \sum_{a=1}^{s} e_a \le \varphi(r) - 1$ *sind die Bilder* $\Phi(f_e) \in K^d$ *paarweise verschieden.*

Beweis. Angenommen, $\Phi(f_c) = \Phi(f_e)$. Nach Korollar 3 gilt für $k = 1, \ldots, d$

$$f_c(X^{m_k})^{n^i p^j} = \prod_{a=1}^{s} (X^{m_k} + a)^{n^i p^j c_a} \equiv \prod_{a=1}^{s} (X^{m_k n^i p^j} + a)^{c_a}$$
$$= f_c(X^{m_k n^i p^j}) \pmod{\hat{\mathfrak{q}}}$$

und ebenso

$$f_e(X^{m_k})^{n^i p^j} \equiv f_e(X^{m_k n^i p^j}) \pmod{\hat{\mathfrak{q}}}.$$

erst recht mod $\hat{\hat{\mathfrak{q}}}$. Anwendung von Φ auf die linken Seiten ergibt

$$f_c(X^{m_k n^i p^j}) \equiv f_e(X^{m_k n^i p^j}) \pmod{\hat{\hat{\mathfrak{q}}}}.$$

Für die Differenz $g := f_c - f_e \in \mathbb{F}_p[X]$ gilt also $g(X^{m_k n^i p^j}) \in h\mathbb{F}_p[X]$ für alle $k = 1, \ldots, d$. Sei $b \in [1 \ldots r-1]$ zu r teilerfremd – also Repräsentant eines Elements von $\mathbb{M}_r$. Dann ist b in einer der Nebenklassen $m_k H$ von $\mathbb{M}_r / H$ enthalten. Es gibt also k, i und j mit $b \equiv m_k n^i p^j \pmod{r}$. Also ist

$$g(X^b) - g(X^{m_k n^i p^j}) \in (X^r - 1)\mathbb{F}_p[X] \subseteq h\mathbb{F}_p[X],$$

also $g(X^b) \in h\mathbb{F}_p[X]$, also $g(\zeta^b) = 0$. Daher hat g in K die $\varphi(r)$ verschiedenen Nullstellen ζ^b. Der Grad von g ist aber $< \varphi(r)$. Also ist $g = 0$, also $f_c = f_e$. $\diamond$

Korollar 4

$$\#\bar{G} \ge \binom{\varphi(r) + s - 1}{s}^{1/d} \ge |t - u| + 1.$$

Beweis. Es gibt $\binom{\varphi(r)+s-1}{s}$ Möglichkeiten, die Exponenten $(e_1, \ldots, e_s)$ wie in Hilfssatz 4 zu wählen. Da alle $\Phi(f_e) \in \bar{G}^d$, folgt

$$\#\bar{G}^d \ge \binom{\varphi(r) + s - 1}{s} \ge n^{2d \cdot \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor},$$

nach der Voraussetzung von Satz 6, also

$$\#\bar{G} \ge n^{2 \cdot \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor} \ge |t - u| + 1$$

nach Hilfssatz 2. $\diamond$

Damit ist der Beweis von Satz 6 leicht fertigzustellen: Da $g^t = g^u$ für alle $g \in \bar{G} \subseteq K$, hat das Polynom $X^{|t-u|}$ in K mehr als $|t - u|$ Nullstellen. Das geht nur, wenn $t = u$. Nach der Definition von t und u in Hilfssatz 2 ist also n eine Potenz von p.

Damit ist Satz 6 bewiesen. $\diamond$

3.8 Der AKS-Algorithmus

Der Algorithmus wird hier in der Version nach Lenstra/Bernstein beschrieben. Diese ist nicht auf möglichst effiziente Ausführung getrimmt, sondern auf einen möglichst einfachen Nachweis der Polynomialität.

Eingabe

Eingegeben wird eine natürliche Zahl $n \geq 2$.

Die Länge der Eingabe wird durch die Zahl ℓ der Bits in der Darstellung von n zur Basis 2 gemessen, also durch

$$\ell = \begin{cases} \lceil {}^2\log n \rceil, & \text{falls } n \text{ keine Zweierpotenz,} \\ k+1, & \text{falls } n = 2^k. \end{cases}$$

Ausgabe

Ausgegeben wird ein Boolescher Wert, sinngemäß ausgedrückt durch „ZUSAMMENGESETZT“ oder „PRIM“.

Schritt 1

Zweierpotenzen werden vorab abgefangen –

- Falls $n = 2$: Ausgabe „PRIM“, **Ende**.
- (Sonst) Falls n Zweierpotenz: Ausgabe „ZUSAMMENGESETZT“, **Ende**.

  Diesen Fall erkennt man daran, dass ${}^2\log n$ ganzzahlig ist.

Von jetzt an kann man annehmen, dass n keine Zweierpotenz, also $\ell = \lceil {}^2\log n \rceil$, ist.

Schritt 2

Eine große Zahl $N \in \mathbb{N}$ wird vorherberechnet, und zwar

$$N = 2n \cdot (n-1)(n^2-1)(n^3-1)\cdots(n^{4\ell^2}-1) = 2n \cdot \prod_{i=1}^{4\ell^2}(n^i-1).$$

Diese Zahl ist zwar riesengroß, aber, und das ist hier entscheidend:

- Die Zahl $4\ell^2$ der Multiplikationen ist polynomial in ℓ.
- Da
  $$N \leq 2n \cdot n^{\sum_{i=1}^{4\ell^2} i} = 2n \cdot n^{\frac{4\ell^2(4\ell^2+1)}{2}} \leq 2n \cdot n^{16\ell^4},$$
  ist $k := \lceil {}^2\log N \rceil \leq 1 + (16\ell^4+1) \cdot \ell$ polynomial in ℓ.

Die Zahl k wird im folgenden weiterverwendet; es ist $N < 2^k$, und k ist die kleinste natürliche Zahl mit dieser Eigenschaft.

Anforderungen

Es sind jetzt natürliche Zahlen r und s zu finden, die folgende Anforderungen erfüllen:

1. r ist zu n teilerfremd.
2. Im Intervall $[1, \ldots, s]$ hat n keinen Primteiler.
3. Für jeden Teiler $d \mid \frac{\varphi(r)}{q}$ – wobei $q = \mathrm{Ord}_r\, n$ – gilt
$$\binom{\varphi(r) + s - 1}{s} \geq n^{2d \cdot \lfloor \frac{\varphi(r)}{d} \rfloor}.$$
4. Das eigentliche Primzahlkriterium:
$$(X + a)^n \equiv X^n + a \pmod{(n, X^r - 1)}$$
für alle $a = 1, \ldots, s$.

Schritt 3

r wird als die kleinste Primzahl genommen, die kein Teiler von N ist; insbesondere ist r dann auch kein Teiler von n; insbesondere ist die Bedingung 1 erfüllt. Warum wird r mit polynomialem Aufwand gefunden?

Nach einer Erweiterung des Primzahlsatzes gilt für die Summe $\vartheta(x)$ der Logarithmen der Primzahlen $\leq x$,
$$\vartheta(x) = \sum_{p \leq x,\, p \text{ prim}} \ln p\,,$$
die asymptotische Relation
$$\vartheta(x) \sim x$$
sowie die Fehlerabschätzung von Rosser/Schoenfeld
$$x \cdot \left(1 - \frac{1}{\ln x}\right) < \vartheta(x) < x \cdot \left(1 - \frac{1}{2 \ln x}\right) \quad \text{für } x \geq 41.$$
Daher ist
$$\prod_{p \leq 2k,\, p \text{ prim}} p = e^{\vartheta(2k)} > 2^k > N.$$
Es können also nicht alle Primzahlen $< 2k$ Teiler von N sein.

Mit einem Aufwand, der höchstens quadratisch in $2k$, also auch polynomial in in ℓ ist, erhält man daher – etwa mit dem Sieb des Eratosthenes – die Liste aller Primzahlen $\leq r$.

Schritt 4

$s := r$. Die Bedingung 2 ist nicht ohne weiteres erfüllt. Daher wird folgendes Verfahren durchgeführt:

- Die (aus Schritt 3 bekannte) Liste der Primzahlen $p < r$ wird durchlaufen.
  - Falls $p = n$: Ausgabe „PRIM“, **Ende**.
    [Das kann nur für „kleine“ n vorkommen, da n exponentiell mit ℓ wächst, r aber nur polynomial.]
  - (Sonst) Falls $p|n$: Ausgabe „ZUSAMMENGESETZT“, **Ende**.

Falls dieser Punkt im Algorithmus noch erreicht wird, ist die Bedingung 2 für s erfüllt.

Bedingung 3

Der Nachweis der Bedingung 3 beginnt mit der Beobachtung, dass $q := \mathrm{Ord}_r\, n > 4\ell^2$.

> Sonst wäre $n^i \equiv 1 \pmod r$ für ein i mit $1 \le i \le 4\ell^2$, also $r \mid n^i - 1 \mid N$, Widerspruch.

Ist nun d ein Teiler von $\frac{\varphi(r)}{q}$, so

$$\begin{aligned} d &\le \frac{\varphi(r)}{q} < \frac{\varphi(r)}{4\ell^2}\,, \\ 2d \cdot \lfloor\sqrt{\frac{\varphi(r)}{d}}\rfloor &\le 2d \cdot \sqrt{\frac{\varphi(r)}{d}} = \sqrt{4d\varphi(r)} < \frac{\varphi(r)}{\ell} < \frac{\varphi(r)}{{}^2\log n}\,, \\ n^{2d\cdot\lfloor\sqrt{\frac{\varphi(r)}{d}}\rfloor} &< n^{\frac{\varphi(r)}{{}^2\log n}} = 2^{\varphi(r)}. \end{aligned}$$

Andererseits ist, da $\varphi(r) \ge 2$,

$$\binom{\varphi(r)+s-1}{s} = \binom{\varphi(r)+r-1}{r} = \binom{2\varphi(r)}{\varphi(r)+1} \ge 2^{\varphi(r)}.$$

Also ist die Bedingung 3 erfüllt.

Schritt 5

Nun wird die Bedingung 4,

$$(X+a)^n \equiv X^n + a \pmod{(n, X^r - 1)}$$

in einer Schleife für $a = 1, \ldots, r$ überprüft. Die Anzahl der Schleifendurchläufe ist höchstens r, also $\le 2k$, also polynomial in ℓ. Bei jedem Schleifendurchlauf wird zweimal binär potenziert, also insgesamt höchstens 4ℓ Mal

multipliziert; multipliziert werden dabei Polynome von einem Grad $< r$ – der also polynomial in ℓ ist – mit Koeffizienten, deren Größe $< n$, deren Bitlänge also auch polynomial in ℓ ist.

- Falls für ein a die Bedingung 4 verletzt ist, wird „ZUSAMMENGESETZT“ ausgegeben, **Ende**.

Ansonsten ist für alle a die Bedingung 4 erfüllt, also n nach dem AKS-Kriterium eine Primzahlpotenz.

Schritt 6

Nun ist noch zu entscheiden, ob n eine echte Primzahlpotenz ist. Da die Primzahlen $\leq r$ keine Teiler von n sind, ist in einer Schleife über t mit $1 \leq t <^r \log n$ zu prüfen:

- Falls $\sqrt[t]{n}$ ganzzahlig: Ausgabe „ZUSAMMENGESETZT“, **Ende**.

Die Zahl der Schleifendurchläufe ist $\leq \ell$, und die Prüfung in jedem Durchlauf ist ebenfalls mit polynomialem Aufwand durchzuführen, wenn man $\lfloor \sqrt[t]{n} \rfloor$ mit einer binären Suche im Intervall $[1 \ldots n-1]$ sucht.

Falls der Algorithmus bis an diese Stelle kommt, wird „PRIM“ ausgegeben, **Ende**.

Damit ist gezeigt:

Hauptsatz 1 *Der AKS-Algorithmus bestimmt, ob n eine Primzahl ist, mit einem Aufwand, der polynomial von $\log n$ abhängt.*

4 Der diskrete Logarithmus mit Anwendungen

4.1 Der diskrete Logarithmus

Für eine ganze Zahl $a \in \mathbb{Z}$ mit $\mathrm{ggT}(a, n) = 1$ hat die **Exponentialfunktion** $\bmod n$ zur Basis a

$$\exp_a \colon \mathbb{Z} \longrightarrow \mathbb{M}_n, \quad x \mapsto a^x \bmod n,$$

die Periode $s := \operatorname{Ord} a | \lambda(n) | \varphi(n)$. Es gibt also eine Umkehrfunktion

$$\log_a \colon \langle a \rangle \longrightarrow \mathbb{Z}/s\mathbb{Z}$$

auf der zyklischen Untergruppe $\langle a \rangle \subseteq \mathbb{M}_n$, den **diskreten Logarithmus** $\bmod n$ zur Basis a. Es gilt

$$\exp_a(x + y) = \exp_a(x) \cdot \exp_a(y) \quad \text{für alle } x, y \in \mathbb{Z};$$

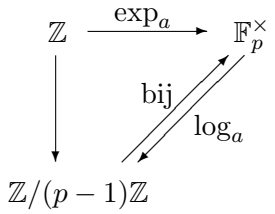
also ist $\exp_a$ ein Halbgruppen-Homomorphismus, $\log_a$ ein Gruppenisomorphismus.

Es ist kein effizienter Algorithmus bekannt, den diskreten Logarithmus $\log_a$ für große $s = \operatorname{Ord} a$ zu bestimmen, d. h., die Exponentialfunktion umzukehren – auch kein probabilistischer.

> **Informelle Definition:** Eine Funktion $f \colon M \longrightarrow N$ heißt **Einwegfunktion**, wenn für „fast alle“ Bilder $y \in N$ ein Urbild $x \in M$ mit $f(x) = y$ nicht effizient bestimmbar ist.
>
> Eine mathematisch präzise Formulierung dieser Definition lässt sich im Rahmen der Komplexitätstheorie geben, siehe später.

Diskreter-Logarithmus-Vermutung: Die Exponentialfunktion $\exp_a \bmod n$ ist für „fast alle“ Basen a eine Einwegfunktion.

Der wichtigste Spezialfall ist: Der Modul ist eine Primzahl $p \geq 3$ und $a \in [2, \ldots, p-2]$ ist ein primitives Element für p, d. h., $\operatorname{Ord} a = p - 1$.



Damit der diskrete Logarithmus praktisch nicht effizient zu berechnen ist, muss man den Primzahlmodul p etwa in der Größenordnung wie einen RSA-Modul wählen, d. h. nach dem heutigen Stand der Technik reichen 1024-Bit-Primzahlen als Modul nicht aus, 2048-Bit-Primzahlen gelten noch als für ein paar Jahre sicher.

4.2 Diffie-Hellman-Schlüsselaustausch

Übertragen werden soll ein Schlüssel für eine symmetrische Chiffrierung. Dazu haben Diffie und Hellman 1976 das folgende Verfahren vorgeschlagen, das auf der Exponentialfunktion in endlichen Körpern, also einer (mutmaßlichen) Einweg-Funktion beruht:

1. A (Alice) und B (Bob) einigen sich (öffentlich) über eine Primzahl p und eine zugehörige Primitivwurzel a.
2. A erzeugt eine Zufallszahl x, bildet $u = a^x \bmod p$ und sendet u an B.
3. B erzeugt eine Zufallszahl y, bildet $v = a^y \bmod p$ und sendet v an A.
4. A berechnet $k = v^x \bmod p$, und B berechnet $k = u^y \bmod p$.

Die Zahl k ist der gemeinsame geheime Schlüssel (oder dient zu dessen Bestimmung nach einem öffentlich bekannten Verfahren). Dass sowohl A als auch B den gleichen Schlüssel haben, liegt an der Gleichung

$$v^x \equiv a^{xy} \equiv u^y \pmod{p}.$$

Ein Lauscher kann nur die Zahlen p, a, u und v abfangen, die ihm nicht gestatten, k oder x oder y effizient zu berechnen.

Damit hat man also auch so etwas wie ein hybrides Verschlüsselungsverfahren; es unterscheidet sich von einem „echten“ asymmetrischen Verfahren aber insofern, als A nicht an B spontan eine Nachricht senden kann, sondern eine Synchronisation herstellen muss.

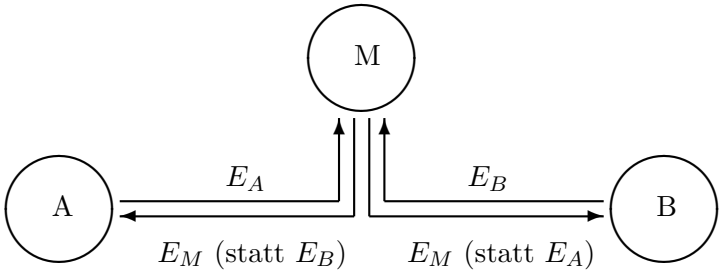
Wenn ein Angreifer effizient diskrete Logarithmen berechnen kann, kann er auch das Diffie-Hellman-Verfahren effizient brechen. Ob auch die umgekehrte Implikation gilt, ist unbekannt.

Beim britischen Geheimdienst CESC war das Verfahren schon 1974 entdeckt, aber natürlich geheim gehalten worden.

4.3 Der Mann in der Mitte

In diesem Abschnitt wird ein Kommunikationssystem mit asymmetrischer Chiffrierung betrachtet; für den DIFFIE-HELLMAN-Schlüsselaustausch funktioniert der Angriff genauso. Das Problem ist, dass ein Angreifer seinen Schlüssel in die Kommunikation einschleusen kann. Genauer:

A = Alice und B = Bob wollen miteinander kommunizieren. Dazu sendet A an B ihren öffentlichen Schlüssel E_A und B an A seinen öffentlichen Schlüssel E_B. Der Angreifer M = Mallory, der „Mann in der Mitte“, fängt diese Sendungen ab und ersetzt beide Male den abgefangenen öffentlichen Schlüssel durch seinen eigenen E_M. Dann kann M unbemerkt den ganzen Nachrichtenverkehr zwischen A und B abhören und sogar verfälschen; diese Situation ist in der folgenden Abbildung dargestellt.



Es gibt verschiedene Auswege aus dieser Bedrohung, die aber alle die asymmetrische Kryptographie verkomplizieren; der übliche Ausweg ist die Verwendung von Zertifikaten: Die öffentlichen Schlüssel aller Teilnehmer am Kommunikationsverfahren werden von einer bekannten und „vertrauenswürdigen“ Stelle – einem sogenannten Trustcenter – digital signiert. Zertifikat = öffentlicher Schlüssel mit digitaler Signatur des Trustcenters.

Übungsaufgabe. Welche Information könnte man beim DIFFIE-HELLMAN-Verfahren als Zertifikat verwenden?

4.4 Geheime Kommunikation ohne Schlüsselaustausch

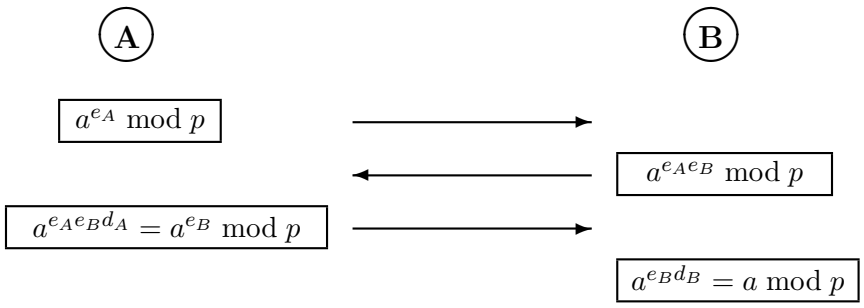
Es ist auch möglich, vertraulich zu kommunizieren, ohne vorher Schlüssel vereinbart zu haben – der Angriff durch den Mann in der Mitte ist hier aber ebenfalls möglich.

Die Idee lässt sich an einem Bild aus dem Alltagsleben verdeutlichen:

- Alice legt die Nachricht in eine Kiste und verschließt sie mit einem Vorhängeschloss, zu dem nur sie den Schlüssel hat. Sie schickt die Kiste an Bob.
- Bob kann die Kiste natürlich nicht öffnen. Statt dessen verschließt er sie mit einem weiteren Vorhängeschloss, zu dem nur er den Schlüssel hat. Er schickt die doppelt verschlossene Kiste an Alice zurück.
- Alice entfernt ihr Vorhängeschloss und schickt die nunmehr nur noch mit Bobs Schloss verschlossene Kiste wieder an Bob.
- Bob entfernt sein Schloss, öffnet die Kiste und liest die Nachricht.

Dieses kryptographische Protokoll heißt Massey-Omura-Schema oder Shamirs No-Key-Algorithmus. Es lässt sich mit Hilfe der diskreten Exponentialfunktion umsetzen; seine Sicherheit beruht auf der Komplexität des diskreten Logarithmus:

Vorgegeben ist eine öffentlich bekannte, allgemein gültige (große) Primzahl p. Alice und Bob wählen je ein Paar von Exponenten d und e mit $ed \equiv 1 \pmod{p-1}$, also $a^{de} \equiv a \pmod{p}$ für alle ganzen Zahlen $a \in \mathbb{Z}$. Jeder hält seine *beiden* Exponenten geheim. Das Protokoll, mit dem Alice nun eine Nachricht a an Bob schickt, sieht so aus:



Ein Angreifer, der diskrete Logarithmen berechnen könnte, könnte aus den erlauschten Geheimtexten $a^{e_A} \bmod p$ und $a^{e_A e_B} \bmod p$ den Exponenten e_B und daraus durch Kongruenzdivision auch d_B berechnen. Ebenso könnte er aus $a^{e_A e_B} \bmod p$ und $a^{e_A e_B d_A} = a^{e_B} \bmod p$ den Exponenten d_A und dann auch e_A berechnen.

Ein anderer Angriff ist nicht bekannt.

4.5 ElGamal-Chiffrierung – Idee

Die ElGamal-Chiffre ist ein asymmetrisches Verfahren – genauer gesagt ein hybrides –, das ebenfalls auf der Komplexität des diskreten Logarithmus beruht.

Eine Primzahl p und ein $g \in [2 \dots p-2]$ sind öffentlich festgelegt; g sollte von hoher Ordnung, am besten primitives Element sein.

> p und g können für alle Teilnehmer gemeinsam gelten, können aber auch individuell verschieden sein.

Jeder Teilnehmer wählt sich eine zufällige Zahl

$$d \in [2 \dots p-2]$$

als privaten Schlüssel und bildet daraus

$$e = g^d \bmod p$$

als zugehörigen öffentlichen Schlüssel. Die Bestimmung von d aus e bedeutet gerade die Berechnung eines diskreten Logarithmus.

Wie soll man nun eine Nachricht a so transformieren, dass sie nur mit Kenntnis von d wiederzugewinnen ist? Die naive Idee, $e^a = g^{da} \bmod p$ zu schicken, nützt nichts – auch der Empfänger kann trotz Kenntnis von d die Nachricht a nicht entschlüsseln. Auch $r = g^a \bmod p$ zu schicken, nützt nichts – der Empfänger kann nur $r^d = e^a \bmod p$ bestimmen, aber nicht a.

Eine bessere Idee ist, erst einen Schlüssel zu erzeugen, mit dem dann ein hybrides Verfahren durchgeführt wird:

- Alice wählt ein zufälliges $k \in [2 \dots p-2]$. Als Schlüssel soll $K = e^k \bmod p$ mit dem öffentlichen Schlüssel von Bob verwendet werden; dieses kann Alice berechnen.
- Damit auch Bob den Schlüssel K bestimmen kann, schickt Alice mit ihrer Nachricht die *Schlüssel-Information* $r = g^k \bmod p$ mit.
- Bob kann daraus $r^d = g^{kd} = e^k = K \bmod p$ mit Hilfe seines privaten Schlüssels d berechnen.

Als symmetrische Chiffe beim Hybridverfahren wird die Verschiebechiffre auf $\mathbb{F}_p^\times$ genommen, wobei K als Einmalschlüssel dient – d. h., für jeden Klartextblock ist ein neuer Schlüssel K zu erzeugen und die entsprechende Schlüssel-Information mitzuschicken. Dadurch ist die Länge des Geheimtexts gerade das Doppelte der Länge des Klartexts.

Die Verschlüsselung läuft dann nach Bildung des Schlüssels und der Schlüssel-Information so weiter ab:

- Alice berechnet den Geheimtext $c = Ka \bmod p$ und schickt diesen zusammen mit r an Bob.

Die Entschlüsselung ist dann, nachdem Bob den Schlüssel K wie oben beschrieben berechnet hat:

- $a = K^{-1}c \bmod p$ durch Kongruenzdivision.

4.6 Berechnung des diskreten Logarithmus

Der klassische Algorithmus zur Berechnung des diskreten Logarithmus ist der **Index-Calculus** von ADLEMAN – „Index“ war übrigens die Bezeichnung, die GAUSS für den diskreten Logarithmus verwendet hatte.

Sie $p \geq 3$ eine Primzahl und a eine Primitivwurzel für p.

Vorbereitung

Dieser Schritt muss zu gegebenen p und a nur einmal ausgeführt werden.

Seien $p_1 = 2$, $p_2 = 3$, …, p_k die ersten k Primzahlen – wie k gewählt wird, wird noch festgelegt.

Für einen zufällig gewählten Exponenten r könnte es sein, dass $a^r \bmod p$ – als ganze Zahl $\in \mathbb{Z}$ betrachtet – nur Primfaktoren in $\{p_1, \ldots, p_k\}$ hat. Nach h solchen Glücksfällen hat man in $\mathbb{Z}$, also erst recht in $\mathbb{F}_p$, ein System von h Gleichungen

$$\begin{aligned} a^{r_1} \bmod p &= p_1^{\alpha_{11}} \cdots p_k^{\alpha_{1k}}, \\ &\vdots \\ a^{r_h} \bmod p &= p_1^{\alpha_{h1}} \cdots p_k^{\alpha_{hk}}. \end{aligned}$$

Daraus entsteht ein lineares Gleichungssystem für die k unbekannten Größen $\log_a p_i$ über dem Ring $\mathbb{Z}/(p-1)\mathbb{Z}$:

$$\begin{aligned} r_1 &= \alpha_{11} \cdot \log_a p_1 + \cdots + \alpha_{1k} \cdot \log_a p_k, \\ &\vdots \\ r_h &= \alpha_{h1} \cdot \log_a p_1 + \cdots + \alpha_{hk} \cdot \log_a p_k. \end{aligned}$$

Zu dessen Auflösung gibt es, wie wir aus Kapitel I wissen, effiziente Algorithmen; ist h groß genug – mindestens $h \geq k$ –, lassen sich $\log_a p_1$, …, $\log_a p_k$ bestimmen.

Die Suche nach den „Glücksfällen“ ergibt also einen probabilistischen Algorithmus.

Berechnung

Sei $y \in \mathbb{F}_p^\times$ gegeben; gesucht ist $\log_a y$.

Für einen zufällig gewählten Exponenten s könnte in $\mathbb{Z}$

$$y \cdot a^s \bmod p = p_1^{\beta_1} \cdots p_k^{\beta_k}$$

sein. Dann ist

$$\log_a y = \beta_1 \cdot \log_a p_1 + \cdots + \beta_k \cdot \log_a p_k - s$$

ganz leicht berechnet.

Die Berechnung des diskreten Logarithmus für beliebige Elemente ist daher auf die Berechnung für die Elemente der **Faktorbasis** $(p_1, \ldots, p_k)$ *reduziert.* Auch diese Reduktion ist probabilistisch.

Varianten

Aus dem vorgestellten Ansatz werden verschiedene Algorithmen konstruiert, die unterschiedlich schnell sind. Sie unterscheiden sich in der Wahl der Faktorbasis – diese kann an y adaptiert sein und muss nicht aus der lückenlosen Folge der ersten Primzahlen bestehen – sowie in der Strategie für die Wahl der Exponenten r und s.

Die schnellste Variante verwendet ein Zahlkörpersieb, wie es auch bei der Faktorisierung großer Zahlen eingesetzt wird, und kommt auf einen Aufwand

$$\approx e^{c\cdot \sqrt[3]{\log p \cdot (\log\log p)^2}},$$

wie er auch bei der Faktorisierung einer gleichlangen zusammengesetzten Zahl benötigt wird. Nach dem aktuellen Stand sind also 1024-Bit-Primzahlmoduln nur ganz kurzfristig als sicher für kryptographische Anwendungen anzusehen, mittelfristig braucht man mindestens 2048 Bit.

Als Kuriosum sei angemerkt, dass das „Secure NFS“ von SUN noch in den 90er Jahren eine Primzahl der Länge 192 Bits (58 Dezimalstellen) verwendete.

Spezielle Primzahlen

Es gibt Argumente, p speziell, also als die größere Zahl eines Germain-Paares zu wählen – also $p = 2q + 1$ mit q prim:

1. Es gibt Algorithmen, die besonders schnell sind, wenn $p-1$ nur kleine Primfaktoren hat. Dieses Argument wird heute nicht mehr für stichhaltig angesehen, da die Chance, aus Versehen eine soche „schlechte“ Primzahl zu erwischen, extrem gering ist; außerdem ist das Zahlkörpersieb so schnell, dass die speziellen Algorithmen kaum noch einen Vorteil bringen.

2. Die Bestimmung eines primitiven Elements ist sehr viel leichter, siehe Abschnitt A.10 im Anhang.

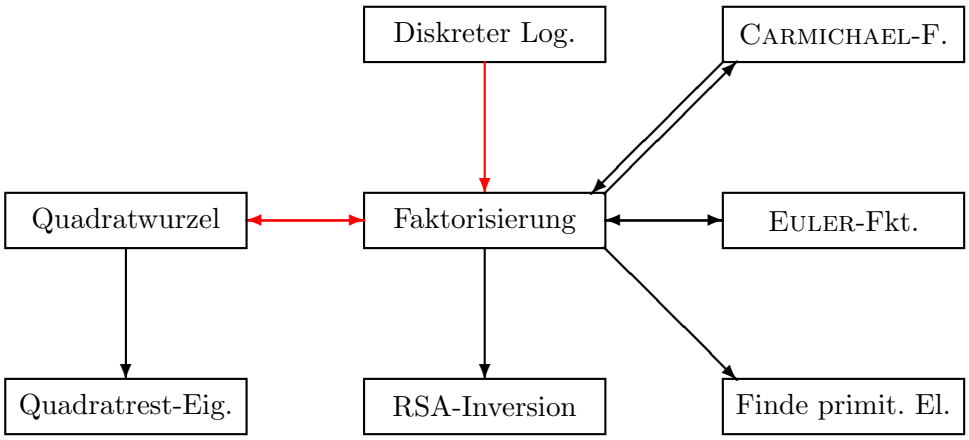
5 Harte zahlentheoretische Probleme

Die folgende Tabelle gibt einen Überblick über kryptologisch relevante zahlentheoretische Berechnungsprobleme. „Effizient“ bedeutet dabei „mit polynomialem Aufwand lösbar“.

Berechnungsproblem	**effizient?**	**behandelt in**
Primzahltest	ja	3.1–3.8
Für Primzahl p		
Finde primitives Element	ja (ERH oder prob.)	A.2, A.10
Finde quadratischen Nichtrest	ja (ERH oder prob.)	A.9
Quadratrest-Eigenschaft	ja	A.6
Quadratwurzel ziehen	ja (prob.)	folgt in 5.3
Diskreter Logarithmus	? (vermutlich nein)	4.1, 4.6
Für zusammengesetzte Zahl n		
Faktorisierung	? (vermutlich nein)	2.2, 2.4
RSA-Inversion (e-te Wurzel)	? (vermutlich nein)	2.2
Berechn. der Euler-Funktion	? (vermutlich nein)	2.2
Berechn. der Carmichael-F.	? (vermutlich nein)	2.2
Finde primitives Element	? (vermutlich nein)	A.4
Quadratrest-Eigenschaft	? (vermutlich nein)	A.8
Quadratwurzel ziehen	? (vermutlich nein)	folgt in 5.5
Diskreter Logarithmus	? (vermutlich nein)	folgt in 5.1

„ERH“ bedeutet „unter Annahme der erweiterten Riemannschen Vermutung“, „prob.“ bedeutet „mit einem probabilistischen Algorithmus“.

Der Zusammenhang zwischen den Berechnungsproblemen für eine zusammengesetzte Zahl n wird in der folgenden Grafik dargestellt. Ein Pfeil von A nach B bedeutet dabei, dass das Problem B mit einem effizienten probabilistischen Algorithmus auf das Problem A reduzierbar ist. Die Umkehrungen bei den einfachen Pfeilen sind jeweils unbekannt. Die Reduktionen, die durch rote Pfeile bezeichnet sind, werden im folgenden bewiesen. (Z. T. nur für den Fall, dass n Produkt zweier Primzahlen ist.)


Diskreter Log.
Carmichael-F.
Quadratwurzel
Faktorisierung
Euler-Fkt.
Quadratrest-Eig.
RSA-Inversion
Finde primit. El.

5.1 Diskreter Logarithmus und Faktorisierung

Für $a \in \mathbb{M}_n$ mit $\operatorname{Ord} a = s$ und zugehöriger Exponentialfunktion

$$\exp_a : \mathbb{Z} \longrightarrow \mathbb{M}_n$$

besteht das Problem des diskreten Logarithmus darin, einen Algorithmus zu finden, der für jedes $y \in \mathbb{M}_n$

- „nein“ ausgibt, wenn $y \notin \langle a \rangle$,
- sonst ein $r \in \mathbb{Z}$ ausgibt mit $0 \leq r < s$ und $y = a^r \bmod n$.

Satz 1 (E. Bach) *Sei $n = pq$ mit verschiedenen Primzahlen $p, q \geq 3$. Dann ist die Faktorisierung von n probabilistisch effizient auf das Problem des diskreten Logarithmus* mod n *reduzierbar.*

Beweis. Es ist $\varphi(n) = (p-1)(q-1)$. Für ein zufällig gewähltes $x \in \mathbb{M}_n$ ist stets $x^{\varphi(n)} \equiv 1 \pmod{n}$. Sei $y := x^n \bmod n$, also

$$y \equiv x^n \equiv x^{n-\varphi(n)} = x^{pq-(p-1)(q-1)} = x^{p+q-1} \pmod{n}.$$

Der diskrete Logarithmus liefert ein r mit $0 \leq r < \operatorname{Ord} x \leq \lambda(n)$ und $y = x^r \bmod n$. Also ist

$$x^{r-(p+q-1)} \equiv 1 \pmod{n}, \quad \operatorname{Ord} x \mid r - (p+q-1).$$

Da $|r - (p+q-1)| < \lambda(n)$, besteht eine große Wahrscheinlichkeit, dass $r = p+q-1$ – z. B. tritt das ein, wenn $\operatorname{Ord} x = \lambda(n)$. Andernfalls wird ein anderes x gewählt.

Aus den beiden Gleichungen

$$\begin{aligned} p+q &= r+1, \\ p \cdot q &= n \end{aligned}$$

sind die Faktoren p und q bestimmbar. $\diamond$

5.2 Quadratwurzeln und Faktorisierung

Satz 2 (M. RABIN) *Sei $n = pq$ mit verschiedenen Primzahlen $p, q \geq 3$. Dann ist die Faktorisierung von n probabilistisch effizient auf das Problem des Quadratwurzelziehens* mod n *reduzierbar.*

Beweis. Es gibt vier Einheitswurzeln in $\mathbb{Z}/n\mathbb{Z}$, also auch zu jedem Quadrat in $\mathbb{M}_n$ vier Quadratwurzeln.

Wählt man nun $x \in M_n$ zufällig, so liefert der Quadratwurzel-Algorithmus eine Wurzel $y \in \mathbb{M}_n$ von x^2, also

$$y^2 \equiv x^2 \pmod{n}.$$

Mit Wahrscheinlichkeit $\frac{1}{2}$ ist $y \not\equiv \pm x \pmod{n}$. Da

$$n \mid (x^2 - y^2) = (x+y)(x-y), \quad n \nmid (x \pm y),$$

ist $\mathrm{ggT}(n, x+y)$ echter Teiler von n. (Alternativ: $y/x \bmod n$ ist nichttriviale Einheitswurzel.) $\diamond$

D. h., wer Quadratwurzeln mod n ziehen kann, kann auch n faktorisieren. Die Umkehrung folgt in Abschnit 5.5.

5.3 Quadratwurzeln in endlichen Primkörpern

Hier soll gezeigt werden, wie man in einem endlichen Primkörper $\mathbb{F}_p$ effizient Quadratwurzeln zieht. Ist $p \equiv 3 \pmod 4$, ist das sogar besonders einfach: Sei $p = 4k + 3$. Ist $z \in \mathbb{M}_p^2$ ein Quadratrest, so ist $x = z^{k+1} \bmod p$ die Quadratwurzel (sogar die in $\mathbb{M}_p^2$), denn $x^2 \equiv z^{2k+2} = z^{1+\frac{p-1}{2}} \equiv z$. Die Gruppe $\mathbb{M}_p^2$ hat nämlich die Ordung $\frac{p-1}{2}$. Der Aufwand für dieses Quadratwurzelziehen besteht aus höchstens $2 \cdot \log_2(p)$ Kongruenzmultiplikationen.

Beispiele

1. Für $p = 7 = 4 \cdot 1 + 3$ ist $k + 1 = 2$. Nach A.9 ist 2 Quadratrest; eine Wurzel ist $2^2 = 4$. Probe: $4^2 = 16 \equiv 2$.

2. Für $p = 23 = 4 \cdot 5 + 3$ ist $k + 1 = 6$. Nach A.9 ist 2 Quadratrest; eine Wurzel ist $2^6 = 64 \equiv 18$. Probe: $18^2 \equiv (-5)^2 = 25 \equiv 2$.

Ist $p \equiv 1 \pmod 4$, ist allerdings kein so einfaches Verfahren möglich. Es ist nämlich z. B. -1 ein Quadrat, aber keine Potenz von -1 kann gleichzeitig Quadratwurzel sein.

Es gibt aber u. a. ein allgemein funktionierendes Verfahren, das nach Adleman, Manders und Miller auch AMM benannt wird, im wesentlichen aber schon 1903 von Cipolla angegeben wurde. Dazu wird $p - 1$ zerlegt in $p - 1 = 2^e \cdot u$ mit ungeradem u. Ferner wählt man (ein- für allemal) einen beliebigen Nichtquadratrest $b \in \mathbb{F}_p^\times - \mathbb{M}_p^2$; dies ist der einzige nichtdeterministische Schritt – dazu siehe Abschnitt A.9.

Nun soll aus dem Quadratrest $z \in \mathbb{M}_p^2$ die Wurzel gezogen werden. Da $z \in \mathbb{M}_p^2$, ist $\mathrm{Ord}(z) \mid \frac{p-1}{2}$, die Zweierordnung $r = \nu_2(\mathrm{Ord}(z))$ von $\mathrm{Ord}(z)$ also $\leq e - 1$, und r ist minimal mit $z^{u2^r} \equiv 1$.

Jetzt wird rekursiv eine Folge $z_1, z_2, \ldots$ gebildet:

$$z_1 = z \quad \text{mit } r_1 = \nu_2(\mathrm{Ord}(z_1)).$$

Ist bereits $z_i \in \mathbb{M}_p^2$ gewählt und r_i die Zweierordnung von $\mathrm{Ord}(z_i)$, so bricht die Folge ab, wenn $r_i = 0$; sonst wird

$$z_{i+1} = z_i \cdot b^{2^{e-r_i}}$$

gesetzt. Dann ist $z_{i+1} \in \mathbb{M}_p^2$. Ferner ist

$$z_{i+1}^{u \cdot 2^{r_i - 1}} \equiv z_i^{u \cdot 2^{r_i - 1}} \cdot b^{u \cdot 2^{e-1}} \equiv 1,$$

denn der erste Faktor ist $\equiv -1$, weil r_i minimal war, und der zweite $\equiv (\frac{b}{p}) = -1$, weil $u \cdot 2^{e-1} = \frac{p-1}{2}$. Also ist $r_{i+1} < r_i$. Der Abbruchpunkt $r_n = 0$ wird spätestens nach e Folgegliedern erreicht, $n \leq e \leq \log_2(p)$.

Dann wird rückwärts berechnet:

$$x_n = z_n^{\frac{u+1}{2}} \bmod p$$

mit $x_n^2 \equiv z_n^{u+1} \equiv z_n$ (denn $\mathrm{Ord}(z_n) \mid u$, da es ungerade ist). Und weiter:

$$x_i = x_{i+1}/b^{2^{e-r_i-1}} \bmod p,$$

das per Induktion

$$x_i^2 \equiv x_{i+1}^2/b^{2^{e-r_i}} \equiv z_{i+1}/b^{2^{e-r_i}} \equiv z_i$$

erfüllt. Also ist $x = x_1$ die gesuchte Wurzel von z.

Abgesehen vom Aufwand, um b zu finden, sind folgende Schritte nötig:

- Berechnung der Potenzen $b^2, \ldots, b^{2^{e-1}}$, und das bedeutet $(e-1)$-mal modular quadrieren.
- Berechnung der Potenzen $b^u, b^{2u}, \ldots, b^{2^{e-1}u}$, und das bedeutet höchstens $2 \cdot \log_2(u) + e - 1$ Kongruenzmultiplikationen.
- Berechnung von z^u mit höchstens $2 \cdot \log_2(u)$ Kongruenzmultiplikationen.
- Dann wird für jedes $i = 1, \ldots, n \leq e$ berechnet:
  - z_i mit einer Kongruenzmultiplikation,
  - z_i^u aus z_{i-1}^u mit einer Kongruenzmultiplikation,
  - $z_i^{u2^r}$ aus $z_{i-1}^{u2^r}$ mit einer Kongruenzmultiplikation,
  - und daraus r_i.

  Das sind höchstens $3 \cdot (e-1)$ Kongruenzmultiplikationen.
- x_n als Potenz mit höchstens $2 \cdot \log_2(u)$ Kongruenzmultiplikationen.
- x_i aus x_{i+1} jeweils durch eine Kongruenzdivision mit Aufwand $\mathrm{O}(\log(p)^2)$.

Das macht zusammen einen Aufwand der Größenordnung $\mathrm{O}(\log(p)^3)$ mit einer eher kleinen Proportionalitätskonstanten.

Beispiel. Sei $p = 29$ und $z = 5$. Dann ist $p-1 = 4 \cdot 7$, also $e = 2$ und $u = 7$. Nach den obigen Bemerkungen ist $b = 2$ geeigneter Nichtquadratrest. Zu berechnen sind die Potenzen

$$b^2 = 4, b^u \equiv 128 \equiv 12, b^{2u} \equiv 144 \equiv -1,$$

$$z^2 \equiv 25 \equiv -4, z^4 \equiv 16, z^6 \equiv -64 \equiv -6, z^7 \equiv -30 \equiv -1.$$

Nun ist

$$z_1 = 5, z_1^u \equiv -1, z_1^{2u} \equiv 1, r_1 = 1,$$
$$z_2 \equiv z_1 b^2 \equiv 5 \cdot 4 = 20, z_2^u \equiv z_1^u b^{2u} \equiv (-1)(-1) = 1, r_2 = 0.$$

Jetzt geht's rückwärts:

$$x_2 \equiv z_2^{\frac{u+1}{2}} = z_2^4 = (z_2^2)^2 \equiv 400^2 \equiv (-6)^2 = 36 \equiv 7,$$

$$x_1 = x_2/b \bmod p = 7/2 \bmod 29 = 18.$$

Also ist $x = 18$ die gesuchte Wurzel. Probe: $18^2 = 324 \equiv 34 \equiv 5$.

Übungsaufgaben. Finde je einen deterministischen Algorithmus (eine einfache Formel) zum Ziehen von Quadratwurzeln in den Körpern

- $\mathbb{F}_p$ mit $p \equiv 5 \pmod 8$,
- $\mathbb{F}_{2^m}$ mit $m \geq 2$.

Alternative Algorithmen. Fast alle bekannten effizienten Algorithmen, die den Fall $p \equiv 1 \pmod 4$ vollständig abdecken, sind probabilistisch; ihre deterministische Version ist unter der erweiterten RIEMANNschen Vermutung noch von polynomialem Aufwand. In dem Buch von FORSTER (*Algorithmische Zahlentheorie*) wird eine Variante des CIPOLLA/ AMM-Algorithmus beschrieben, die die quadratische Körpererweiterung $\mathbb{F}_{p^2} \supseteq \mathbb{F}_p$ benützt und konzeptionell besonders einfach ist. Im *Handbook of Applied Cryptography* (MENEZES/ VAN OORSCHOT/ VANSTONE) wird ein Algorithmus angegeben, der von TONELLI 1891 stammt und recht kurz zu formulieren ist, aber den Aufwand $O(\log(p)^4)$ benötigt. Eine weitere Methode ist ein Spezialfall des Verfahrens von CANTOR/ ZASSENHAUS zur Faktorisierung von Polynomen über endlichen Körpern, siehe VON ZUR GATHEN/ GERHARD: *Modern Computer Algebra.* Ein weiteres Verfahren beruht auf der LUCAS-Folge (a_n) nit $a_1 = b$, $a_2 = b^2 - 2z$, wobei $b^2 - 4z$ Nicht-Quadratrest ist; dieses Verfahren stammt von LEHMER [keine Referenz]. Der einzige bekannte deterministische Algorithmus mit polynomialem Aufwand stammt von SCHOOF, verwendet die Theorie der elliptischen Kurven und ist praktisch unterlegen – Aufwand $O(\log(p)^9)$.

Für einen Überblick siehe:

- E. BACH/ J. SHALLIT: *Algorithmic Number Theory.* MIT Press, Cambridge Mass. 1996.
- D. J. BERNSTEIN: Faster square roots in annoying finite fields. Preprint (siehe die Homepage des Autors `http://cr.yp.to/`).

5.4 Quadratwurzeln bei Primzahlpotenz-Moduln

Mit einem einfachen Verfahren (hinter dem das HENSELsche Lemma steckt) kann man von Primzahlmoduln zu Primpotenzmoduln übergehen. Sei p eine Primzahl $\neq 2$ und $e \geq 2$. Sei z ein Quadratrest $\bmod p^e$. Gesucht ist eine passende Quadratwurzel.

Natürlich ist z auch Quadratrest $\bmod p^{e-1}$. Angenommen, dafür haben wir schon eine Wurzel gefunden, also ein y mit $y^2 \equiv z \pmod{p^{e-1}}$. Sei

$$a = 1/2y \bmod p$$

und $y^2 - z = p^{e-1} \cdot u$. Dann wird

$$x = y - a \cdot (y^2 - z) \bmod p^e$$

gesetzt. Damit gilt

$$\begin{aligned} x^2 &\equiv y^2 - 2ay(y^2 - z) + a^2(y^2 - z)^2 \equiv y^2 - 2ayp^{e-1}u \\ &\equiv y^2 - p^{e-1}u = z \pmod{p^e}. \end{aligned}$$

Also ist x die gesuchte Wurzel.

Dieser Algorithmus soll hier nicht explizit aufgeschrieben, aber an zwei Beispielen verdeutlicht werden:

Beispiele

1. $n = 25$, $z = 19$. Es ist $p = 5$, $19 \bmod 5 = 4$. Also kann man $y = 2$ und $a = 1/4 \bmod 5 = 4$ nehmen. Dann ist $y^2 - z = -15$ und

   $$x = 2 + 15 \cdot 4 \bmod 25 = 62 \bmod 25 = 12.$$

   Probe: $12^2 = 144 = 125 + 19$.

2. $n = 27$, $z = 19$. Es ist $p = 3$, $19 \bmod 3 = 1$. Also kann man im ersten Schritt $y = 1$ und $a = 1/2 \bmod 3 = 2$ nehmen. Dann ist $y^2 - z = -18$ und

   $$x = 1 + 2 \cdot 18 \bmod 9 = 37 \bmod 9 = 1.$$

   Beim zweiten Schritt (von 9 nach 27) ist also wieder $y = 1$, $y^2 - z = -18$ und damit

   $$x = 37 \bmod 27 = 10.$$

   Probe: $10^2 = 100 = 81 + 19$.

Der Aufwand besteht aus zwei Teilen

1. $\bmod p$ wird eine Wurzel gezogen und einmal dividiert. (Der Quotient a muss insgesamt nur einmal bestimmt werden, da $x \equiv y \pmod p$.)

2. Bei jeder Erhöhung des Exponenten sind zwei Kongruenzmultiplikationen und zwei Subtraktionen fällig.

Der Gesamtaufwand bleibt also $O(\log(n)^3)$, wenn n der Modul ist.

Es bleibt noch der Fall zu untersuchen, dass $n = 2^e$ eine Zweierpotenz ist. Ist $e \leq 3$, so ist 1 der einzige Quadratrest, und seine Wurzel ist 1. Für größere Exponenten e wird wieder auf $e - 1$ rekurriert: Sei z eine ungerade Zahl (alle invertierbaren Elemente sind ungerade). Sei y bereits gefunden mit $y^2 \equiv z \pmod{2^{e-1}}$. Dann ist $y^2 - z = 2^{e-1} \cdot t$. Ist t gerade, so auch $y^2 \equiv z \pmod{2^e}$. Andernfalls setzt man $x = y + 2^{e-2}$. Dann ist

$$x^2 \equiv y^2 + 2^{e-1}y + 2^{2e-2} \equiv z + 2^{e-1} \cdot (t + y) \equiv z \pmod{2^e},$$

da $t + y$ gerade ist. Also ist $x = y$ oder $y + 2^{e-2}$ die gesuchte Wurzel. Der Gesamtaufwand ist hier sogar kleiner als $O(\log(n)^3)$.

Nebenbei haben wir gezeigt, dass z genau dann Quadratrest mod 2^e ist (für $e \geq 3$), wenn $z \equiv 1 \pmod{8}$.

5.5 Quadratwurzeln bei zusammengesetzten Moduln

Ist die Primzerlegung eines Moduls n bekannt, so lassen sich in $\mathbb{M}_n$ effizient Quadratwurzeln ziehen; die Probleme „Faktorisierung“ und „Ziehen von Quadratwurzeln“ sind also in ihrer Komplexität äquivalent.

Zur Durchführung wird n sukzessive in teilerfremde Faktoren zerlegt (bis hinunter zu den Primpotenzen).

Sei also $n = rs$ mit teilerfremden Faktoren r und s. Zuerst werden mit dem erweiterten Euklidischen Algorithmus Koeffizienten a und b mit $ar + bs = 1$ bestimmt. Aus z soll die Quadratwurzel gezogen werden. Sei u die Quadratwurzel $\bmod r$ und v die Quadratwurzel $\bmod s$. Dann erfüllt $x = arv + bsu \bmod n$:

$$\begin{aligned} x &\equiv bsu \equiv u \pmod{r}, & x &\equiv arv \equiv v \pmod{s}, \\ x^2 &\equiv u^2 \equiv z \pmod{r}, & x^2 &\equiv v^2 \equiv z \pmod{s}, \end{aligned}$$

insbesondere ist $x^2 \equiv z \pmod{n}$.

Der Aufwand für dieses Verfahren besteht aus zwei Quadratwurzeln modulo den Faktoren, einem Euklidischen Algorithmus und 4 Kongruenzmultiplikationen (+ 1 Kongruenzaddition). Er bleibt also in der Größenordnung $\mathrm{O}(\log(n)^3)$.

Für Blum-Zahlen gibt es sogar einen noch einfacheren Algorithmus, nämlich eine explizite Formel:

Korollar 1 *Sei $n = pq$ mit Primzahlen $p, q \equiv 3 \pmod 4$. Dann gilt*

(i) *$d = \frac{(p-1)(q-1)+4}{8}$ ist ganzzahlig.*

(ii) *Für jedes Quadrat $x \in \mathbb{M}_n^2$ ist x^d Quadratwurzel aus x.*

Beweis. (i) Ist $p = 4k + 3$, $q = 4l + 3$, so $(p-1)(q-1) = 16kl + 8k + 8l + 4$, also $d = 2kl + k + l + 1$.

(ii) Der Exponent der multiplikativen Gruppe $\mathbb{M}_n$,

$$\lambda(n) = \mathrm{kgV}(p-1, q-1) = 2 \cdot \mathrm{kgV}(2k+1, 2l+1)$$

ist Teiler von $2 \cdot (2k+1) \cdot (2l+1)$, der Exponent der Quadrat-Untergruppe $\mathbb{M}_n^2$ ist $\frac{\lambda(n)}{2}$, also Teiler von $(2k+1) \cdot (2l+1) = 4kl + 2k + 2l + 1 = 2d - 1$. Also gilt $x^{2d} \equiv x \pmod n$ für alle $x \in \mathbb{M}_n^2$, d. h., das Quadrat von x^d ergibt x. $\diamond$

6 Kryptographische Basisfunktionen und ihre Äquivalenz

Für Anwendungen der Kryptographie im praktischen Leben – also für die Konstruktion kryptographischer Protokolle – werden die folgenden Basisfunktionen benötigt:

1. Symmetrische Chiffren:
   (a) Bitblock-Chiffren,
   (b) Bitstrom-Chiffren.
2. Asymmetrische Chiffren.
3. Schlüsselfreie Chiffren:
   (a) Einweg-Funktionen,
   (b) Hash-Funktionen.
4. Zufall:
   (a) Physikalische Zufallsgeneratoren,
   (b) (Algorithmische) Pseudozufallsgeneratoren.
5. Steganographie.

Es wird sich zeigen, dass die Existenz der meisten davon – nämlich 1a, 1b, 3a, 3b, 4b in geeigneten Varianten – jeweils zum Grundproblem der theoretischen Informatik $\mathbf{P} \stackrel{?}{\neq} \mathbf{NP}$ äquivalent und damit bisher unbewiesen ist.

In diesem Abschnitt werden Komplexitätsaussagen noch naiv formuliert. Der Ansatz zur Formalisierung mittels TURING-Maschinen wird skizziert. Er ist allerdings für die Kryptologie nicht ausreichend. Die mathematisch strenge Version von Komplexitätsaussagen für kryptologische Verfahren folgt im nächsten Abschnitt 7.

6.1 Einweg-Funktionen

Es wird weiterhin die informelle Definition aus 4.1 verwendet. Eine exakte Definition wird in 7.5 nachgereicht.

Anwendung: Einweg-Funktionen können direkt zur Einweg-Verschlüsselung verwendet werden. Das bedeutet:

- *Jeder* kann verschlüsseln.
- *Niemand* kann entschlüsseln.

Wozu soll das gut sein, wenn niemand entschlüsseln kann? Dafür gibt es durchaus eine Reihe von Anwendungen (die z. T. den Spezialfall der Hash-Funktionen, siehe 6.2, einsetzen):

- Die Passwort-Verwaltung, z. B. unter Unix oder MS-Windows. Hier soll niemand das verschlüsselt abgespeicherte Passwort ermitteln können, wohl aber muss das Betriebssystem die Möglichkeit haben, das neu eingebene Passwort nach Verschlüsselung mit dem verschlüsselt abgelegten zu vergleichen.
- Ähnlich sieht die Anwendung bei Pseudonymisierung aus: Die Daten eines Falls sollen mit anderswo abgelegten Daten zusammengeführt werden, ohne dass jemand die zum Fall gehörenden Identitätsdaten sehen oder ermitteln kann.
- Eine weitere Anwendung betrifft die digitale Signatur, siehe 6.2.
- Schliesslich ist auch der entscheidende Aspekt der asymmetrischen Verschlüsselung, dass niemand den privaten Schlüssel aus dem öffentlichen ableiten kann. Hierzu sind allerdings Einweg-Funktionen nicht ohne weiteres direkt einsetzbar, wie schon das Beispiel der ElGamal-Chiffre in 4.5 gezeigt hat.

Beispiele für mutmaßliche Einweg-Funktionen:

1. Die diskrete Exponential-Funktion, siehe 4.1.
2. Eine Standard-Methode, aus einer Bitblock-Chiffre

   $$F\colon M \times K \longrightarrow C,$$

   die resistent gegen einen Angriff mit bekanntem Klartext ist, eine Einweg-Funktion $f\colon K \longrightarrow C$ abzuleiten, geht so:

   $$f(x) := F(m_0, x);$$

   es wird also ein fester Klartext m_0 – etwa der Bitblock, der aus lauter Nullen besteht – mit einem Schlüssel, der genau aus dem Einweg-umzuwandelnden Block x besteht, verschlüsselt. Die Umkehrung dieser Einweg-Funktion würde genau dem Angriff mit bekanntem Klartext m_0 auf die Chiffre F entsprechen.

3. Sei $n \in \mathbb{N}$ ein zusammengesetzter Modul. Wir wissen aus 5.2, dass zumindest im Fall, dass n aus zwei großen ungeraden Primzahlen zusammengesetzt ist, das Ziehen der Quadratwurzel $\bmod n$ nicht effizient möglich ist. Damit ist die Quadratabbildung $x \mapsto x^2 \bmod n$ im Restklassenring $\mathbb{Z}/n\mathbb{Z}$ eine Einweg-Funktion – immer unter der Annahme, das die Faktorisierung nicht effizient möglich ist. Allerdings ist die Umkehrung möglich, wenn eine Zusatzinformation vorliegt, nämlich die Primfaktoren von n. Eine solche Zusatzinformation heißt 'trapdoor' (Falltür), und man spricht dann auch von einer „Trapdoor-Einweg-Funktion".
4. Das gleiche gilt für die RSA-Funktion $x \mapsto x^e \bmod n$ mit einem zu $\lambda(n)$ (oder $\varphi(n)$) teilerfremden Exponenten e.

6.2 Hash-Funktionen

Ein besonders wichtiger Spezialfall der Einweg-Funktionen sind die Hash-Funktionen, auch „Message Digest“ oder „kryptographische Prüfsumme“ genannt.

Definition 1. Sei Σ ein Alphabet und $n \in \mathbb{N}$ eine feste natürliche Zahl ≥ 1. Dann heißt eine Einweg-Funktion

$$h\colon \Sigma^* \longrightarrow \Sigma^n$$

schwache Hash-Funktion über Σ.

Zeichenketten *beliebiger* Länge werden dabei also auf Zeichenketten vorgegebener *fester* Länge abgebildet. Da Σ^* unendlich ist, ist gemeint, dass die Einschränkung von h auf Σ^r für alle genügend großen r Einweg-Funktion ist.

Definition 2. Eine Einweg-Funktion $f : M \longrightarrow N$ heißt **kollisionsfrei**, wenn es nicht effizient möglich ist, $x_1, x_2 \in M$ zu finden mit $x_1 \neq x_2$, aber $f(x_1) = f(x_2)$.

Man könnte das auch als „praktisch injektiv“ bezeichnen; injektive Einweg-Funktionen sind natürlich kollisionsfrei. Ist $\#M > \#N$, so kann f nicht injektiv, könnte aber durchaus kollisionsfrei sein.

Definition 3. Eine **(starke) Hash-Funktion** ist eine kollisionsfreie schwache Hash-Funktion.

Für die praktische Anwendung (meistens mit $\Sigma = \mathbb{F}_2$) soll die Länge n der Hashwerte klein sein. Da die Umkehrbarkeit aber nicht effizient sein darf, will man kryptographische Sicherheit erreichen, muss n andererseits auch genügend groß sein. Geht man davon aus, dass die Hash-Funktion statistisch zufällig aussehende, gleichverteilte Werte liefert, so muss man bei einer schwachen Hash-Funktion also sicher vor Exhaustion (vollständiger Suche) sein. Das bedeutet, dass $n = 80$ als Untergrenze gerade nicht mehr ausreichend ist, man also besser 128-Bit-Hashwerte verwenden sollte. Das ist gerade die Länge bei den bekannten Verfahren MD2, MD4, MD5.

Bei praktisch allen Anwendungen ist aber auch die Kollisionsfreiheit wichtig. Hier ist das Geburtstagsphänomen, siehe I.2.6, zu berücksichtigen: Um Kollisionen mit hinlänglicher Sicherheit unauffindbar zu machen, ist etwa die doppelte Bitlänge nötig. Hashwerte von 160 Bit sind also gerade nicht mehr lang genug. Die noch gültigen Standard-Hashverfahren SHA-1 und RIPEMD verwenden genau diese Länge, sollten also schleunigst ausgemustert werden. Im Kontext der AES-Standardisierung wurde das Hash-Verfahren SHA-2 mit mindestens 256-Bit-Hashwerten eingeführt,

das dann auch passenderweise als SHA-256 usw. bezeichnet wird [siehe `http://csrc.nist.gov/publications/`]. Für die MDx-Verfahren wurden tatsächlich schon systematisch Kollisionen gefunden [Dobbertin 1996ff.].

Anwendungen: (Starke) Hash-Funktionen verwendet man

- bei der digitalen Signatur. Eine lange Nachricht direkt mit dem privaten Schlüssel zu verschlüsseln würde bei der Langsamkeit der asymmetrischen Verfahren zu lange dauern. Daher signiert man einen Hashwert der Nachricht. Damit das sicher ist, braucht man eine kollisionsfreie Hash-Funktion. Sonst nämlich könnte ein Angreifer sich auf folgende Weise ein beliebiges Dokument a von Alice signieren lassen: Er fertigt ein unverdächtiges Dokument b an, das Alice gerne unterschreibt. Von beiden Dokumenten stellt er $m = 2^k$ Varianten $a_1, \ldots, a_m$ und $b_1, \ldots, b_m$ her, indem er an k verschiedenen Stellen jeweils ein Leerzeichen einfügt oder nicht. Falls er eine Kollision findet, etwa $h(a_i) = h(b_j)$, lässt er b_j von Alice signieren und hat dann eine gültige Signatur für a_i.
- für die Umwandlung einer langen, für einen Menschen merkbaren Passphrase in einen (schwer merk- und eingebbaren) n-Bit-Schlüssel für eine symmetrische Chiffre.

6.3 Umwandlungstricks

Die Äquivalenz der folgenden Aussagen (A) bis (D) und ihre Implikation von (E) wird plausibel hergeleitet; für einen richtigen mathematischen Beweis fehlen ja noch die exakten Definitionen. Die Implikationen haben durchaus auch praktische Bedeutung für die Konstruktion von Basisfunktionen aus anderen. Diese kann man stark vereinfacht in Hinblick auf die immer wieder aufkommende politische Debatte über eine Kryptographie-Regulierung so zusammenfassen:

- Wer Kryptographie verbieten will, muss auch Hash-Funktionen und Pseudozufallsgeneratoren verbieten.
- Wer Kryptographie unmöglich machen will, muss $\mathbf{P} = \mathbf{NP}$ beweisen.

(A) Es gibt eine Einweg-Funktion $f\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$.

($\tilde{\mathbf{A}}$) Es gibt eine Einweg-Funktion $\tilde{f}\colon \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^n$.

(B) Es gibt eine schwache Hash-Funktion $h\colon \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^n$.

(C) Es gibt eine starke symmetrische Chiffre $F\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ (d. h. eine, die sicher vor einem Angriff mit bekanntem Klartext ist).

(D) Es gibt einen perfekten Zufallsgenerator $\sigma\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^{p(n)}$.

(E) $\mathbf{P} \neq \mathbf{NP}$.

Anmerkung 1: Bei der komplexitätstheoretischen Präzisierung steht in den Aussagen (A) – (D) stets eine mit n parametrisierte Familie von Funktionen.

Anmerkung 2: Die Perfektheit des Zufallsgenerators besagt, dass bei unbekanntem Urbild $x \in \mathbb{F}_2^n$ aus einigen bekannten Bits des Bildes $\sigma(x)$ keine weiteren Bits effizient bestimmt werden können; insbesondere auch nicht das Urbild x. In der Definition ist p ein ganzzahliges Polynom, das „Streckungspolynom“.

Die Implikation „(D) $\Longrightarrow$ (E)“ wird hier nicht bewiesen.

„(C) $\Longrightarrow$ (D)“: Man setzt $\sigma(x) = (s_1, \ldots, s_{p(n)/n})$ mit $s_0 := x$ und $s_i := F(s_{i-1}, z)$ für $i \geq 1$, wobei als Schlüssel z ein geheim gehaltener konstanter Parameter verwendet wird; man erkennt den OFB-Modus für Bitblock-Chiffren wieder. Dann kann aus jedem Block s_i der Folge der Vorgängerblock s_{i-1} nicht bestimmt werden – sonst wäre die Chiffre nicht sicher. – Dass das schon für die Perfektheit reicht, wird in Kapitel IV gezeigt.

„(D) $\Longrightarrow$ (C)“: Die Bitstrom-Chiffre mit $\sigma(x)$ als Bitstrom zum Schlüssel x ist sicher.

„(A) $\Longrightarrow$ (C)“: Am einfachsten ist der Ansatz von E. Backus; dabei wird $F(a,k) = f(a) + f(k)$ gesetzt. Bei einem Angriff mit bekanntem Klartext sind a und $c = F(a,k)$ bekannt; damit ist auch $f(k) = c + f(a)$ bekannt. Der Angriff ist also auf die Umkehrung von f reduziert. [Andere Ansätze sind MDC (= Message Digest Cryptography) von P. Gutmann und das Feistel-Prinzip.]

„(C) $\Longrightarrow$ (A)“: Das war als Beispiel in Abschnitt 6.1 vorgestellt worden.

„(A) $\Longrightarrow$ ($\tilde{\text{A}}$)“: Sei $\tilde{f}$ durch $\tilde{f}(x,y) := f(x+y)$ definiert. Ist dann zu c ein Urbild (x,y) unter $\tilde{f}$ bestimmbar, so hat man auch das Urbild $x+y$ unter f bestimmt.

„($\tilde{\text{A}}$) $\Longrightarrow$ (B)“: $x \in \mathbb{F}_2^*$ wird mit (höchstens $n-1$) Nullen zu $(x_1, \ldots, x_r) \in (\mathbb{F}_2^n)^r$ aufgefüllt. Dann setzt man

$$\begin{aligned} c_0 &:= 0, \\ c_i &:= \tilde{f}(c_{i-1}, x_i) \quad \text{für } 1 \le i \le r, \\ h(x) &:= c_r. \end{aligned}$$

Damit ist $h\colon \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^n$ definiert. Findet man nun zu gegebenem $y \in \mathbb{F}_2^n$ ein Urbild $x \in (\mathbb{F}_2^n)^r$ mit $h(x) = y$, so auch ein $z \in (\mathbb{F}_2^n)^2$ mit $\tilde{f}(z) = y$, nämlich $z = (c_{r-1}, x_r)$ (wobei $y = c_r$ in der Konstruktion von h).

„(B) $\Longrightarrow$ (A)“: Die Einschränkung von h auf $\mathbb{F}_2^n$ ist auch Einweg-Funktion.

6.4 Physikalische Komplexität

Der unmittelbar einleuchtende Ansatz zur Beurteilung der Komplexität von Algorithmen ist die Zählung von Basisoperationen, wie sie auf einem handelsüblichen Prozessor ausgeführt werden, oder genauer von Taktzyklen. Dies würde konkrete Komplexitätsaussagen der Art: „Zur Berechnung von ... sind mindestens (z. B.) 10^{80} der folgenden Rechenschritte nötig: ... “.

Derartige Aussagen sind natürlich für konkret gegebene Algorithmen möglich. Ergebnisse, die aussagen, wieviele Rechenschritte *jeder* Algorithmus zur Lösung eines bestimmten Problems mindestens enthalten muss, hat leider keine Art von Komplexitätstheorie zu bieten, außer für ganz einfache Probleme wie die Auswertung eines Polynoms an einer Stelle. Mit solchen Aussagen könnte man wirkliche Sicherheitsaussagen für Verschlüsselungsverfahren mathematisch beweisen, ohne auf unbewiesene Vermutungen und heuristische Argumente zurückgreifen zu müssen.

Dazu würde man auf physikalische Schranken zurückgreifen können, die sagen, welche Ressourcen Rechner höchstens zur Verfügung haben. Eine bekannte Abschätzung dieser Art sieht so aus (nach Louis K. Scheffer in `sci.crypt`):

- Es gibt höchstens 10^{90} Elementarteilchen im Universum – das ist eine Schranke für Zahl der möglichen CPUs –,
- Es braucht mindestens 10^{-35} Sekunden, um ein Elementarteilchen mit Lichtgeschwindigkeit zu durchqueren – das ist eine Zeitschranke für eine Operation –,
- Das Universum hat eine Lebensdauer von höchstens 10^{18} Sekunden ($\approx 30 \times 10^9$ Jahre) – das ist eine Schranke für die verfügbare Zeit.

Daraus folgt, dass höchstens $10^{143} \approx 2^{475}$ Operationen in diesem Universum möglich sind. Insbesondere sind 500-Bit-Schlüssel sicher vor vollständiger Suche ...

> ... until such time as computers are built from something other than matter, and occupy something other than space. (Paul Ciszek)

Diese Sicherheitsschranke gilt aber nur für den einen Algorithmus „vollständige Suche“; sie sagt nichts über die Sicherheit auch nur eines einzigen kryptographischen Verfahrens!

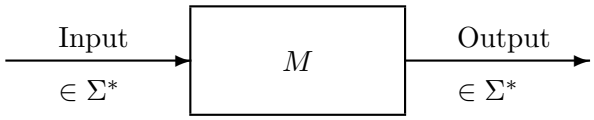
Natürlich ist eine realistische obere Schranke um viele Größenordnungen kleiner.

Zum Vergleich noch ein paar kryptologisch relevante Größen:

Sekunden/Jahr	3×10^7
CPU-Zyklen/Jahr auf 1-GHz-Rechner	3.2×10^{16}
Alter des Universums in Jahren	10^{10}
CPU-Zyklen seither (1 GHz)	3.2×10^{26}
Atome in der Erde	10^{51}
Elektronen im Universum	8.37×10^{77}
ASCII-Ketten der Länge 8 (95^8)	6.6×10^{15}
Binärketten der Länge 56 (2^{56})	7.2×10^{16}
Binärketten der Länge 80	1.2×10^{24}
Binärketten der Länge 128	3.4×10^{38}
Binärketten der Länge 256	1.2×10^{77}
75-stellige Primzahlen (ca 250 Bit)	5.2×10^{72}

6.5 TURING-Maschinen

Die mathematische Komplexitätstheorie führt fast ausschließlich zu asymptotischen Aufwandsabschätzungen, so gut wie immer Abschätzungen nach oben. Sie beruht in ihren verschiedenen Varianten auf verschiedenen Berechnungs- oder Maschinen-Modellen. Hier wird die übliche Formalisierung von Komplexitätsaussagen durch TURING-Maschinen kurz skizziert.



Dabei ist Σ wie üblich ein endliches Alphabet. Der Input ist eine endliche Zeichenkette auf einem (in beide Richtungen unendlich langen) Band. Die TURING-Maschine M besitzt sie eine endliche Zustandsmenge, die unter anderem den Zustand „halt“ enthält. In Abhängigkeit vom Zustand führt sie gewisse Operationen aus, z. B. ein Zeichen vom Band lesen oder auf das Band schreiben und den Lesekopf um eine Stelle nach links oder rechts verschieben. Kommt M in den Zustand „halt“, so ist die dann auf dem Band befindliche Zeichenkette der Output.

Sei $L \subseteq \Sigma^*$ eine Sprache. Falls M für alle $x \in L$ nach endlich vielen Schritten den Zustand „halt“ erreicht, sagt man, M **akzeptiert die Sprache** L. Ist $f : L \longrightarrow \Sigma^*$ eine Funktion und kommt M für jedes $x \in L$ nach endlich vielen Schritten zum Zustand „halt“ mit dem Output $f(x)$, so sagt man, M **berechnet** f.

Mit etwas Mühe und nicht besonders elegant lassen sich alle Algorithmen im naiven Sinne durch TURING-Maschinen beschreiben. Die Komplexität lässt sich durch Zählen der Schritte ausdrücken; für den Input x braucht M bis zum Zustand „halt“ τ_x Schritte.

Meistens betrachtet man die „Worst-Case“-Komplexität. Sei wie üblich $L_n := L \cap \Sigma^n$. Dann wird die Funktion

$$t_M : \mathbb{N} \longrightarrow \mathbb{N}, \quad t_M(n) := \max\{\tau_x \mid x \in L_n\},$$

als **(Zeit-) Komplexität** der TURING-Maschine M bezeichnet.

Die Teilmenge $\mathbf{P}$ („polynomiale Zeit“) der Menge aller Funktionen aus Σ^* nach Σ^* wird so definiert: Die Funktion $f : L \longrightarrow \Sigma^*$ liegt in $\mathbf{P}$, wenn es eine TURING-Maschine M und eine natürliche Zahl $k \in \mathbb{N}$ gibt mit

(i) M berechnet f,

(ii) $t_M(n) \leq n^k$ für fast alle $n \in \mathbb{N}$.

Bemerkung. Äquivalent zu (ii) ist: Es gibt ein Polynom $p \in \mathbb{N}[X]$ mit $t_M(n) \leq p(n)$ für alle $n \in \mathbb{N}$.

Gibt es nämlich solch ein Polynom $p = a_r X^r + \cdots + a_0$ mit $a_r \neq 0$, so ist

$$\begin{aligned} a_r n^r &\geq a_{r-1} n^{r-1} + \cdots + a_0 \quad \text{für } n \geq n_0, \\ p(n) &\leq 2a_r n^r \quad \text{für } n \geq n_0, \\ p(n) &\leq n^{r+1} \quad \text{für } n \geq n_1 = \max\{2a_r, n_0\}. \end{aligned}$$

Ist umgekehrt $t_M(n) \leq n^k$ für $n \geq n_0$, so kann man $c \in \mathbb{N}$ wählen mit $t_M(n) \leq c$ für die endlich vielen $n = 0, \ldots, n_0 - 1$. Dann ist $t_M(n) \leq p(n)$ für alle $n \in \mathbb{N}$ mit $p = X^k + c$.

Analog ist die Menge **EXPTIME** („exponentielle Zeit“) definiert: f liegt in **EXPTIME**, wenn es eine TURING-Maschine M, eine natürliche Zahl $k \in \mathbb{N}$ und reelle Zahlen $a, b \in \mathbb{R}$ gibt mit

(i) M berechnet f,

(ii) $t_M(n) \leq a \cdot 2^{bn^k}$ für fast alle $n \in \mathbb{N}$.

Klar ist $\mathbf{P} \subseteq \mathbf{EXPTIME}$.

Beispiele mit $\Sigma = \mathbb{F}_2$.

1. Sei
$$L := \{(p, z) \in \mathbb{N}^2 \mid p \text{ prim } \equiv 3 \pmod 4,\ z \in \mathbb{M}_n^2\},$$
durch eine geeignete Binärdarstellung als Teilmenge von Σ^* codiert. Sei $f(p, z) =$ Quadratwurzel von $z \mod p$ – ebenfalls als Element von Σ^* codiert. Dann ist $f \in \mathbf{P}$ nach 5.3.

2. Sei $L = \mathbb{N}_2$ die Menge aller natürlichen Zahlen ≥ 2 (binär codiert). Sei $f(x) =$ der kleinste Primfaktor von x. Dann ist $f \in \mathbf{EXPTIME}$ – man kann ja alle Zahlen $\leq \sqrt{x} \leq 2^{n/2}$ durchprobieren. *Vermutlich* ist aber $f \notin \mathbf{P}$.

3. **Das Rucksackproblem** (‘knapsack problem’). Hier ist
$$L = \{(m, a_1, \ldots, a_m, N) \mid m, a_1, \ldots, a_m, N \in \mathbb{N}\}$$
in geeigneter binärer Codierung,
$$f(m, a_1, \ldots, a_m, N) = \begin{cases} 1, & \text{wenn es } S \subseteq \{1, \ldots, m\} \text{ gibt} \\ & \text{mit } \sum_{i \in S} a_i = N, \\ 0 & \text{sonst.} \end{cases}$$
Dann ist $f \in \mathbf{EXPTIME}$ – man kann ja alle 2^m Teilmengen $S \subseteq \{1, \ldots, m\}$ durchprobieren. *Vermutlich* ist $f \notin \mathbf{P}$.

6.6 Die Klasse NP

Die TURING-Maschine M berechnet $f : L \longrightarrow \Sigma^*$ **nichtdeterministisch**, wenn es zu jedem $x \in L$ ein $y \in \Sigma^*$ gibt, so dass M mit der Verkettung xy von x und y als Input nach endlich vielen Schritten mit dem Output $f(x)$ anhält.

Beispiel. Sei $\Sigma = \mathbb{F}_2$ und $L = \{(n, a, x) \in \mathbb{N}^3 \mid n \geq 2, a, x \in \mathbb{M}_n\}$. Sei $f = \log_a \bmod n$ der diskrete Logarithmus.

Zu gegebenem x sei y der Logarithmus von x – woher wir ihn haben spielt in der Definition keine Rolle, er existiert jedenfalls. Dann muss die TURING-Maschine M nur noch prüfen, ob $a^y = x$.

Vorstellung. Ein Kandidat y für die Lösung wird vorgegeben, M macht nur noch die Probe.

Alternativ-Vorstellung. Unbeschränkt viele *parallele* TURING-Maschinen probieren je ein $y \in \Sigma^*$ auf Eignung aus.

Die Menge **NP** („nichtdeteministisch-polynomiale Zeit“) ist definiert als die Menge der Funktionen, für die es eine TURING-Maschine M und eine natürliche Zahl $k \in \mathbb{N}$ gibt mit

(i) M berechnet f nichtdeterministisch,

(ii) $t_M(n) \leq n^k$ für fast alle $n \in \mathbb{N}$.

Es gelten die Inklusionen

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME};$$

die erste davon ist trivial, die zweite ein Satz, der hier nicht bewiesen wird.

Das schon öfter angesprochene wichtigste ungelöste Problem der theoretischen Informatik ist die Vermutung

$$\mathbf{P} \neq \mathbf{NP}.$$

Ebenfalls unbewiesen ist die Vermutung

$$\mathbf{NP} \neq \mathbf{EXPTIME}.$$

Bewiesen ist allerdings

$$\mathbf{P} \neq \mathbf{EXPTIME},$$

wenn auch nur durch „künstliche“ Probleme; ein interessantes „natürliches“ Problem in der Differenzmenge ist nicht bekannt.

Die Kryptoanalyse schwieriger als **NP** zu machen, ist übrigens nicht möglich: Die Exhaustion – das Durchprobieren aller Schlüssel mit jeweiliger

Probeverschlüsselung bei bekanntem Klartext – ist nämlich immer möglich und die Verschlüsselungsfunktion muss effizient, also in **P** sein.

Beispiele

1. Ist f der diskrete Logarithmus wie oben, so $f \in \mathbf{NP}$.
2. Genauso ist die Faktorisierung natürlicher Zahlen in **NP**.
3. Auch das Rucksackproblem ist in **NP**.

Die Funktion f heißt **NP-vollständig**, wenn es für jede TURING-Maschine, die f berechnet (deterministisch!) und jede Funktion $g \in \mathbf{NP}$ eine TURING-Maschine, die g berechnet, und eine natürliche Zahl $k \in \mathbb{N}$ gibt, so dass

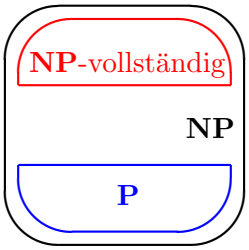
$$t_N(n) \leq t_M(n)^k \quad \text{für fast alle } n \in \mathbb{N}.$$

D. h., die Komplexität von N ist höchstens polynomial in der Komplexität von M.

Vorstellung: **NP**-vollständige Probleme sind die maximal komplexen unter denen in **NP**.

Es gibt **NP***-vollständige Probleme.* – Das ist ein Satz, der hier nicht bewiesen wird. Z. B. ist das Rucksack-Problem **NP**-vollständig, ebenso die Nullstellenbestimmung von (Polynom-) Funktionen $p\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$. Die Faktorisierung natürlicher Zahlen ist vermutlich nicht **NP**-vollständig.

Sollte $\mathbf{P} = \mathbf{NP}$ sein – was niemand glaubt –, so wären alle Funktionen in $\mathbf{P} = \mathbf{NP}$ auch **NP**-vollständig. Andernfalls gibt die folgende Skizze eine Vorstellung von der relativen Lage dieser Mengen:



7 Komplexitätstheorie in der Kryptologie

Es gibt (mindestens) drei Gründe, warum die „gewöhnliche“ Komplexitätstheorie (TURING-Maschinen) für die Kryptologie nicht ausreicht:

1. Die Komplexitätstheorie behandelt die Frage, ob der *schlechteste* Fall ('worst case') hart ist (d. h., nicht effizient berechenbar). Um eine effiziente Kryptoanalyse auszuschließen, muss der *Normalfall* hart sein.

   Insbesondere reicht die Aussage $\mathbf{P} \stackrel{?}{\neq} \mathbf{NP}$ *nicht* für die Existenz starker kryptographischer Basisfunktionen aus.

   Beispiele für ein solches Phänomen aus anderen Bereichen sind der NEWTON-Algorithmus zur Bestimmung von Polynom-Nullstellen und die Simplex-Methode zur linearen Optimierung, die im schlechtesten Fall hart, im Normalfall aber sehr effizient sind.

2. Man muss auch probabilistische Algorithmen zulassen, wie bei den zahlentheoretischen Problemen schon einigemale gesehen („Monte-Carlo“-Algorithmen, die sehr effizient sind, aber nicht immer das richtige Ergebnis liefern).

   Die exakte mathematische Behandlung verwendet stochastische Begriffe: Teile des Inputs enstammen einem Wahrscheinlichkeitsraum Ω; es werden Aussagen über die Verteilung des Outputs gemacht.

3. Der Kryptoanalytiker kann seine Methode an das konkrete Problem adaptieren; er benötigt nicht unbedingt einen universellen Algorithmus, der für *alle* Instanzen seines Problems effizient ist. Z. B. kann er je nach Länge des Schlüssels einen anderen Algorithmus zum Brechen der Chiffre wählen.

TURING-Maschinen reichen zur Formalisierung der in der Kryptologie nötigen Komplexitätstheorie also nicht. Man kann das beheben, indem man Familien von TURING-Maschinen zulässt – eine für jede Input-Länge – und auch probabilistischen Input einbezieht. Einfacher zu beschreiben und eleganter anzuwenden ist allerdings ein Maschinenmodell, das auf BOOLEschen Schaltnetzen (englisch: circuits) beruht: probabilistische polynomiale Schaltnetzfamilien (PPS). – Die Umsetzung der gängigen Algorithmen in Schaltnetze ist deutlich einfacher als die Programmierung einer TURING-Maschine.

7.1 Schaltnetze

Eine formale Beschreibung von „Algorithmen“ kann man wie gesehen mit Hilfe von TURING-Maschinen geben. Ein einfacher zu definierendes, verstehendes und handzuhabendes Konzept ist das BOOLEsche **Schaltnetz**, das die in einem Algorithmus auszuführenden Bitoperationen in Form eines Ablaufdiagramms darstellt. Es wird auf zwei Weisen verallgemeinert:

probabilistisches Schaltnetz – damit werden probabilistische Algorithmen formalisiert,

polynomiale Schaltnetzfamilie – mit deren Hilfe kann die Komplexität eines Algorithmus bei wachsender Größe der Eingabedaten ausgedrückt werden.

Man kommt so zum Begriff der Schaltnetzkomplexität, der 1949 von SHANNON eingeführt wurde und für die Anwendung in der Kryptologie besonders gut geeignet ist.

Ein **Schaltnetz** ist ein azyklischer gerichteter markierter Graph, dessen Knoten sämtlich den Innengrad 0 oder 2 haben.

> Das heisst, Knoten sind mit gerichteten Kanten (Pfeilen) verbunden, wobei kein geschlossener Weg entsteht, alle Knoten sind mit Markierungen (Attributen) versehen, in jedem Knoten enden 0 oder 2 Pfeile.

Ein **Eingang** ist ein Knoten mit Innengrad 0. Ein **Ausgang** ist ein Knoten mit Außengrad 0; von ihm gehen also keine weiteren Pfeile aus. Alle Knoten sind mit einem der Symbole $\oplus$ oder $\otimes$ markiert – das entspricht der Addition und Multiplikation zweier Bits im Körper $\mathbb{F}_2$. Einige Eingänge sind als konstant gekennzeichnet; an ihnen liegt stets ein festes Bit 0 oder 1 an. Eine **Eingabe** ist eine Belegung der r nichtkonstanten Eingänge mit einer Bitfolge $x = (x_1, \ldots, x_r) \in \mathbb{F}_2^r$. An den inneren Knoten (“Gattern“) werden die anliegenden Bits dann jeweils $\bmod 2$ addiert oder multipliziert, je nach Markierung des Knotens. Am Ausgang ensteht nach Durchlaufen des gesamten Schaltnetzes die **Ausgabe** $y \in \mathbb{F}_2^s$. Das Schaltnetz definiert also eine Funktion

$$C\colon \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$$

und gibt deren algorithmische Zerlegung in Bitoperationen wieder. Durch ein Schaltnetz ausdrücken lässt sich jede solche Funktion, die nur durch Addition und Multiplikation gebildet werden kann, also jedes Polynom, also nach II (Einschub über BOOLEsche Abbildngen) jede Funktion $\mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$. Im Fall $r = 2$, $s = 1$ gibt es

$$(\#\mathbb{F}_2)^{\#(\mathbb{F}_2 \times \mathbb{F}_2)} = 2^4 = 16$$

derartige Funktionen. Sie sind alle in Tabelle 1 aufgezählt, aus der man auch leicht die entsprechenden Schaltnetze entnehmen kann; sie werden als Polynome $a+bX+cY+dXY \in \mathbb{F}_2[X,Y]$ beschrieben (algebraische Normalform). Allgemein werden Schaltnetz und zugehörige Funktion, da Verwirrung nicht zu befürchten ist, mit dem gleichen Buchstaben bezeichnet.

a	b	c	d	Polynom (ANF)	logische Operation	
0	0	0	0	0	FALSE	Konstante
1	0	0	0	1	TRUE	Konstante
0	1	0	0	X	X	Projektion
1	1	0	0	$1+X$	$\neg X$	negierte Projektion
0	0	1	0	Y	Y	Projektion
1	0	1	0	$1+Y$	$\neg Y$	negierte Projektion
0	1	1	0	$X+Y$	$X \oplus Y$	XOR
1	1	1	0	$1+X+Y$	$X \Longleftrightarrow Y$	Äquivalenz
0	0	0	1	XY	$X \wedge Y$	AND
1	0	0	1	$1+XY$	$\neg(X \wedge Y)$	NAND
0	1	0	1	$X+XY$	$X \wedge (\neg Y)$	
1	1	0	1	$1+X+XY$	$X \Longrightarrow Y$	Implikation
0	0	1	1	$Y+XY$	$(\neg X) \wedge Y$	
1	0	1	1	$1+Y+XY$	$X \Longleftarrow Y$	Implikation
0	1	1	1	$X+Y+XY$	$X \vee Y$	OR
1	1	1	1	$1+X+Y+XY$	$\neg(X \vee Y)$	NOR

Tabelle 1: Die 16 zweistelligen Bitoperationen

7.2 Schaltnetze für grundlegende Operationen

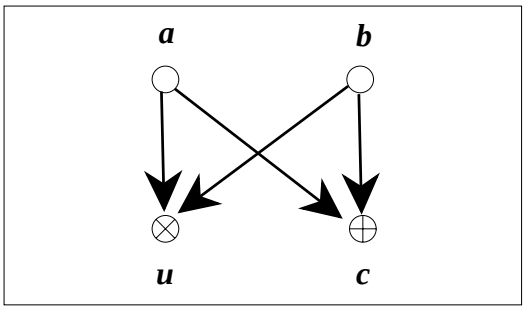


Abbildung 1: Schaltnetz zur Addition zweier Einbit-Zahlen

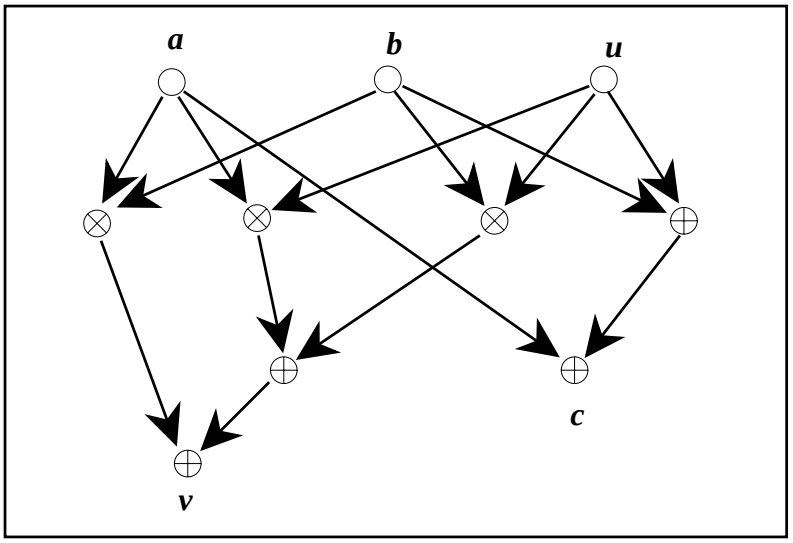


Abbildung 2: Schaltnetz zur Addition dreier Einbit-Zahlen

1. Die Addition zweier Bits a, b mit mod 2-Summe c und Übertrag u, also die „primitive“ Addition in $\mathbb{N}$ von Ziffern zur Basis 2, ist eine Funktion $C : \mathbb{F}_2^2 \longrightarrow \mathbb{F}_2^2$. Ein passendes Schaltnetz ist in Abbildung 1 wiedergegeben.

2. Die analoge Addition dreier Bits a, b, u mit mod 2-Summe c und Übertrag v ist der Grundbaustein, mit dem die Addition beliebiger Ganzzahlen zur Basis 2 aufgebaut wird. Sie wird durch eine Funktion $C : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^2$ mit dem Schaltnetz aus Abbildung 2 beschrieben. Eine

Formel für v ist nämlich:

$$v = (a \otimes b) \oplus (a \otimes u) \oplus (b \otimes u) = \begin{cases} 1, & \text{wenn mindestens 2 Eingaben 1 sind,} \\ 0 & \text{sonst.} \end{cases}$$

3. Die Addition einer Einbit-Zahl zu einer s-Bit-Zahl verläuft nach einem ähnlichen, umfangreicheren Schema und lässt sich durch ein Schaltnetz mit $3s + 1$ Knoten, davon $s + 1$ Ausgängen, erledigen.

4. Die Multiplikation zweier Bits ist trivial. Interessanter ist es, die Multiplikation zweier Zweibit-Zahlen $2a_1 + a_0$ und $2b_1 + b_0$ mit Vierbit-Ergebnis $8c_3 + 4c_2 + 2c_1 + c_0$ durch ein Schaltnetz zu beschreiben. Der klassische Algorithmus ergibt die Formeln

$$\begin{aligned} &c_0 = a_0 \otimes b_0, \\ &t_1 = a_0 \otimes b_1, \quad t_2 = a_1 \otimes b_0, \quad c_1 = t_2 \oplus t_1, \quad u_1 = t_2 \otimes t_1, \\ &t_3 = a_1 \otimes b_1, \quad c_2 = t_3 \oplus u_1, \quad c_3 = t_3 \otimes u_1 \end{aligned}$$

mit Hilfsbits t_1, t_2, t_3 und u_1. Sie lassen sich in das Schaltnetz aus Abbildung 3 übersetzen.

5. Auch logische Operationen und Verzweigungen kann man auf Additionen und Multiplikationen in $\mathbb{F}_2$ reduzieren. Für die logischen Operationen sieht man das an der Tabelle 1. Eine Verzweigung wird etwa so beschrieben: Gegeben seien drei Schaltnetze $C_0, C_1 \colon \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$ und $E : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2$. Gesucht ist ein Schaltnetz $C \colon \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$ mit

$$C(x) = \begin{cases} C_0(x), & \text{falls } E(x) = 0, \\ C_1(x), & \text{falls } E(x) = 1. \end{cases}$$

Dieses konstruiert man mit Hilfe der Formeln

$$\begin{aligned} y \otimes v &= \begin{cases} 0, & \text{falls } y = 0, \\ v, & \text{falls } y = 1. \end{cases} \\ (\neg y) \otimes u &= \begin{cases} u, & \text{falls } y = 0, \\ 0, & \text{falls } y = 1. \end{cases} \end{aligned}$$

$$[y \otimes v] \oplus [(1 \oplus y) \otimes u] = \begin{cases} u, & \text{falls } y = 0, \\ v, & \text{falls } y = 1. \end{cases}$$

Setzt man hier C_0, C_1 und E ein, so erhält man als Lösung

$$C(x) = [E(x) \otimes C_1(x)] \oplus [(1 \oplus E(x)) \otimes C_0(x)].$$

Dieses Schaltnetz ist in Abbildung 4 dargestellt. Es hat auch einen konstanten Eingang.

6. Der Vergleich $x \geq y$ zweier Bits ist eine der 16 zweistelligen Bitoperationen, nämlich $1 \oplus y \oplus x \otimes y$. Allgemeiner lässt sich ein Vergleich von $n+1$-Bit-Zahlen $x = (x_n \dots x_0)$ und $y = (y_n \dots y_0)$ so zusammenbasteln:

$$C(x,y) = \begin{cases} 1, & \text{wenn } x_n > y_n \\ & \text{oder } x_n = y_n \text{ und } x' \geq y', \\ 0, & \text{wenn } x_n = y_n \text{ und } x' < y' \\ & \text{oder } x_n < y_n \end{cases}$$

mit $x' = (x_{n-1} \dots x_0)$ und $y' = (y_{n-1} \dots y_0)$.

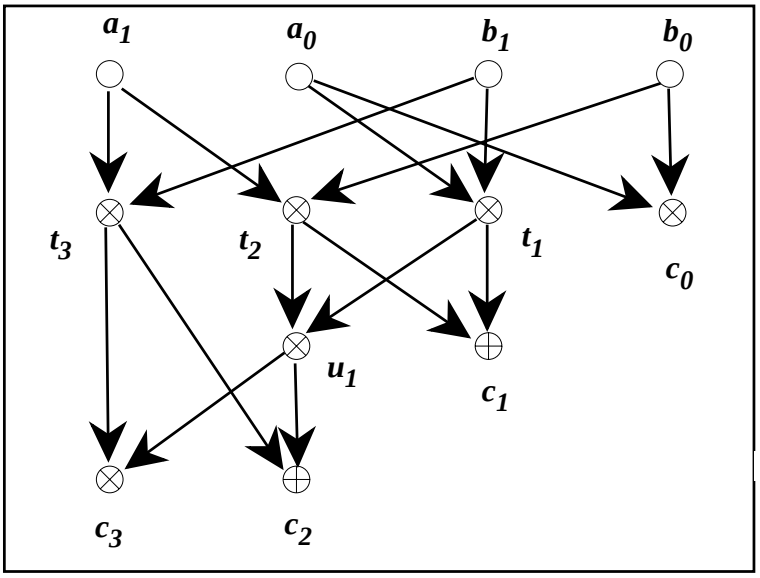


Abbildung 3: Schaltnetz zur Multiplikation zweier Zweibit-Zahlen

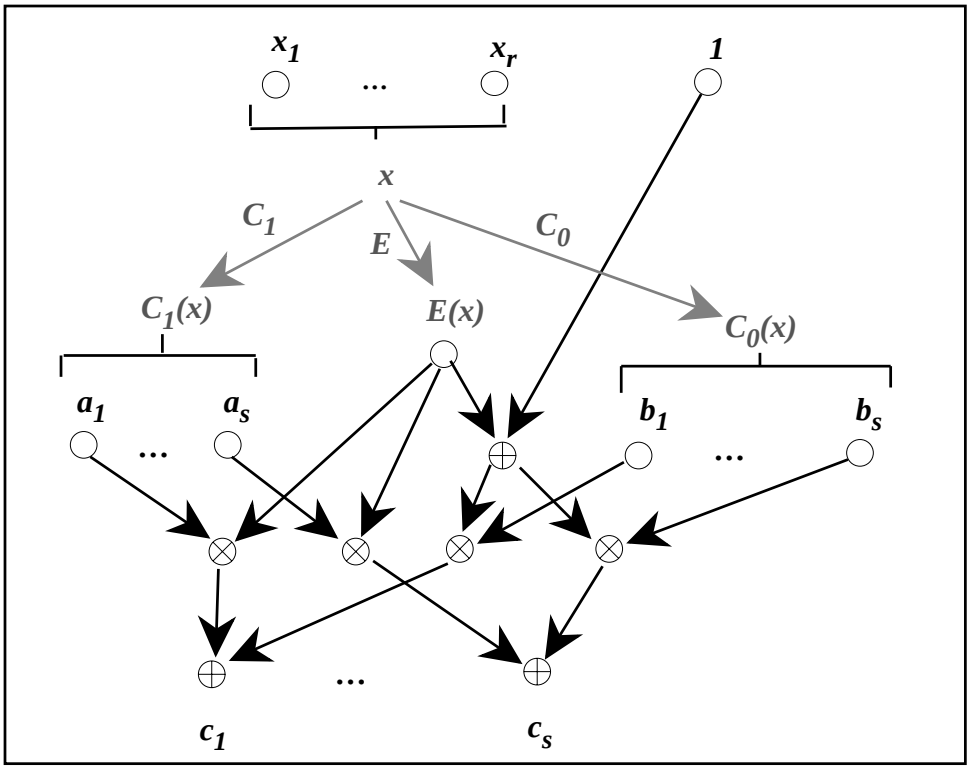


Abbildung 4: Schaltnetz für eine Verzweigung

7.3 Komplexitätsmaße für Schaltnetze

Es ist intuitiv klar und auch leicht zu zeigen, dass sich jeder Algorithmus durch ein Schaltnetz beschreiben lässt; genau nach diesem Prinzip der bitweisen Operationen arbeiten ja Computer auf der untersten Ebene – Das Schaltnetz ist also ein universelles Maschinenmodell. Klar ist auch, dass man mit einem solchen simplen Modell nicht direkt genaue Aufwandsberechnungen für reale Computer durchführen kann; wohl aber ist die Unterscheidung, ob ein Algorithmus effizient, also mit polynomialem Zeitaufwand, arbeitet oder nicht, damit sehr gut möglich. Dazu betrachtet man die beiden Werte $\#C$, die **Größe** des Schaltnetzes, also die Anzahl der Knoten, und $t(C)$, die **Tiefe**, also die größte Weglänge innerhalb des Schaltnetzes. Die Größe ist ein Komplexitätsmaß für die serielle Abarbeitung, die Tiefe eines für die unbeschränkt parallele Abarbeitung des Algorithmus. **Beispiele**

1. Das Schaltnetz für die Addition zweier Bits, Abbildung 1 hat die Größe 4 und die Tiefe 1.

2. Das Schaltnetz für die Addition dreier Bits, Abbildung 2 hat die Größe 10 und die Tiefe 3.

3. Aus dem obigen Beispiel 3 erhält man ein einfaches (längst nicht optimales) Schaltnetz zur Addition von s Einbit-Zahlen mit $A(s)$ Knoten, darunter $a(s)$ Ausgängen. Nach Beispiel 1 ist $A(2) = 4$, $a(2) = 2$, nach Beispiel 3 gilt die Rekursion $a(s) = a(s-1) + 1$ und $A(s) = A(s-1) + 2a(s-1) + 1$. Daraus folgt $a(s) = s$ und $A(s) = s^2$ durch Induktion.

4. Das Schaltnetz für die Multiplikation zweier Zweibitzahlen, Abbildung 3 hat die Größe 12 und die Tiefe 3.

5. Beim Schaltnetz für die Verzweigung, Abbildung 4 gelten die Formeln

$$\begin{aligned} \#C &= \#C_1 + \#C_2 + \#E - 2r + 3s + 2, \\ t(C) &= \max\{t(C_0) + 2, t(C_1) + 2, t(E) + 3\}. \end{aligned}$$

6. Für die Größe $V(n)$ des Schaltnetzes zum Vergleich in Beispiel 6 gilt die Rekursionsformel $V(n) = V(n-1)+11$ mit V(2) = 6, also $V(n) = 11n - 5$.

7. Die Aufwandsabschätzungen für die Ganzzahl-Operationen mit der Basis $B = 2$ des Zahlensystems besagen, dass Paare von n-Bitzahlen mit Schaltnetzen der Größe $O(n)$ addiert und subtrahiert und mit Schaltnetzen der Größe $O(n^2)$ multipliziert und dividiert werden können.

Anmerkungen

Bei der Definition von Schaltnetzen wird manchmal der Innengrad nicht durch 2 beschränkt. Diese Beschränkung ist aber durchaus sinnvoll, da reale Maschinen in jedem Schritt nur eine bestimmte Anzahl von Bits verarbeiten können. Ob man die Schranke auf 2 festsetzt oder höher, spielt für die folgenden Komplexitätsüberlegungen allerdings keine Rolle.

Zwischenergebnisse können in einem Schaltnetz wegen des unbeschränkten Außengrades beliebig oft verwendet werden; das entspricht einem unbeschränktem Speicher auf einem realen Computer. Ein engerer Begriff, bei dem dies vermieden wird, ist die BOOLEsche Formel: Eine solche ist ein Schaltnetz, dessen Graph ein Baum ist. Das bedeutet, dass der Außengrad jedes Knotens 1 ist (außer den Ausgängen mit Außengrad 0). Ein solches Schaltnetz entspricht genau dem Aufbau einer ausgeschriebenen Formel aus zweistelligen Operationen, wo ja auch jedes Zwischenergebnis nur einmal verwendet werden kann. Insbesondere muss jede Eingabe so oft wiederholt werden, wie sie gebraucht wird.

Bei einer weiteren allgemeineren Definition von Schaltnetzen werden auch beliebige zweistellige Bitoperationen statt nur $\oplus$ (XOR) und $\otimes$ (AND) zugelassen. Tabelle 1 zeigt auch, wie sich alle solchen durch $\oplus$ und $\otimes$ ausdrücken lassen. Diese allgemeinere Definition gestattet in der Regel etwas kürzere Schaltnetze, hat aber keine lohnenden Auswirkungen auf die Komplexitätsüberlegungen.

Schaltnetze (oder Algorithmen) dienen nicht nur zur Berechnung von Funktionen, sondern auch zum Finden von Lösungen, formalisiert durch das Erfüllen einer Relation. Seien $X \subseteq \mathbb{F}_2^r$ und $Y \subseteq \mathbb{F}_2^s$ zwei Mengen und E eine Relation auf $X \times Y$, beschrieben durch eine Funktion

$$E : X \times Y \longrightarrow \mathbb{F}_2.$$

Gefragt sind Algorithmen, die zu gegebenem $x \in X$ ein $y \in Y$ finden mit $E(x, y) = 1$. Den bisher behandelten Spezialfall einer auszuwertenden Funktion $f : X \longrightarrow Y$ findet man als rechtseindeutige Relation wieder – $E(x, y) = 1 \Longleftrightarrow y = f(x)$. Man erfasst aber auch das Lösen von Gleichungen, etwa von linearen Gleichungssystemen: $X = \mathbf{M}_{m,n}(\mathbb{F}_2)$ = die Menge der $m \times n$-Matrizen, $Y = \mathbb{F}_2^m \times \mathbb{F}_2^n$, $E(x, y) = 1 \Longleftrightarrow xy_1 = y_2$ (als Produkt Matrix $\times$ Vektor).

Ein Schaltnetz $C : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$ erfüllt E, wenn $C(X) \subseteq Y$ und $E(x, C(x)) = 1$ für alle $x \in X$.

7.4 Probabilistische Schaltnetze

Nun zur Formalisierung probabilistischer Algorithmen. Das sieht im Ansatz etwa so aus: Gegeben sei eine Funktion

$$f\colon A \longrightarrow \mathbb{F}_2^s$$

auf einer Menge A. Ein probabilistischer Algorithmus über dem Wahrscheinlichkeitsraum Ω soll eine Funktion

$$C\colon A \times \Omega \longrightarrow \mathbb{F}_2^s$$

definieren; er dient zur (probabilistischen) Berechnung von $f(x)$ bzw. f, wenn die Wahrscheinlickeit

$$P(\{\omega \mid C(x,\omega) = f(x)\}) \quad (\text{„lokal“}) \text{ bzw.}$$

$$P(\{(x,\omega) \mid C(x,\omega) = f(x)\}) \quad (\text{„global“})$$

genügend groß ist (signifikant $> \frac{1}{2^s}$). Im lokalen Fall wird bei festem x über Ω gemittelt, im globalen Fall auch über A. Dabei sollen im allgemeinen die Wahrscheinlichkeitsräume Ω und $A \times \Omega$ endlich und mit der Gleichverteilung versehen sein.

Um probabilistische Algorithmen beschreiben zu können, muss man Schaltnetze mit *drei* verschiedenen Typen von Eingängen betrachten: r **deterministische Eingänge**, die mit einer Eingabe $x \in \mathbb{F}_2^r$ belegt werden, einige konstante Eingänge – da der Innengrad unbeschränkt ist, reichen zwei, je einer für die Konstanten 0 und 1 – und k **probabilistische Eingänge**, die mit einem Element des LAPLACEschen Wahrscheinlichkeitsraums $\Omega = \mathbb{F}_2^k$ belegt werden (k „Münzenwürfe“), oder mit Elementen eines Teilraums $\Omega \subseteq \mathbb{F}_2^k$. Über die Ausgabe $y \in \mathbb{F}_2^s$ werden dann Wahrscheinlichkeitsaussagen der oben beschriebenen Art gemacht.

Beispiele

1. Die Suche nach einem Nicht-Quadratrest für einen Primzahlmodul p: Dabei sei p eine n-Bitzahl. Dazu wurde $b \in [1 \ldots p-1]$ zufällig gewählt und $(\frac{b}{p})$ (das LEGENDRE-Symbol, das für Quadratreste 1, für Nichtquadratreste -1 ist) berechnet. Die Wahrscheinlichkeit für einen Erfolg ist dabei $\frac{1}{2}$, der Aufwand $O(n^2)$ (siehe Anhang A.9). Betrachten wir allgemeiner die Frage, ob in einem unabhängig gewählten h-Tupel $(b_1, \ldots, b_h) \in \Omega = [1 \ldots p-1]^h$ ein Nicht-Quadratrest vorkommt. Es gibt (für das fest gewählte p) ein Schaltnetz ohne deterministische Eingänge (aber mit konstanten Eingängen zur Einspeisung von p),

$$C : \mathbb{F}_2^{hn} \longrightarrow \mathbb{F}_2^n,$$

$$C(\omega) = \begin{cases} b_i, & \text{das erste } b_i, \text{ das Nicht-Quadratrest ist,} \\ 0, & \text{falls kein Nicht-Quadratrest gefunden wurde,} \end{cases}$$

das die Größe $\mathrm{O}(hn^2)$ hat und mit der Wahrscheinlichkeit $1 - \frac{1}{2^h}$ erfolgreich ist.

2. Der strenge Pseudoprimzahltest: Hier entstammen die Eingaben der Menge A der ungeraden Zahlen in $[3 \dots 2^n - 1]$. Berechnet werden soll die Funktion

$$f : A \longrightarrow \mathbb{F}_2, \quad f(m) = \begin{cases} 1, & \text{falls } m \text{ zusammengesetzt,} \\ 0, & \text{falls } m \text{ prim.} \end{cases}$$

Die zufälligen Eingänge werden durch die Wahl eines Basiselements $a \in \Omega = [2 \dots 2^n - 1]$ besetzt. Der strenge Pseudoprimzahltest liefert ein Schaltnetz

$$C : \mathbb{F}_2^n \times \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$$

der Größe $\mathrm{O}(n^3)$ mit dem Ergebnis 1, wenn m durchfällt (dann ist m sicher zusammengesetzt), oder 0, wenn m besteht (dann ist m möglicherweise prim).

Die Eigenschaft eines (probabilistischen) Schaltnetzes C mit r deterministischen Eingängen, eine Entscheidung mit einer Wahrscheinlichkeit richtig zu treffen, die signifikant vom zufälligen Erraten des Wertes $f(x) \in \mathbb{F}_2^s$ abweicht, wird so formalisiert: Ein Schaltnetz

$$C : \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2^s$$

hat einen **ε-Vorteil** mit $\varepsilon \geq 0$ bei der Berechnung von $f(x)$ bzw. f, wenn

$$P(\{\omega \in \Omega \mid C(x, \omega) = f(x)\}) \geq \frac{1}{2^s} + \varepsilon \quad (\text{„lokal“}) \text{ bzw.}$$

$$P(\{(x, \omega) \in A \times \Omega \mid C(x, \omega) = f(x)\}) \geq \frac{1}{2^s} + \varepsilon \quad (\text{„global“}).$$

Die Wahrscheinlichkeit bezüglich ω für das richtige Ergebnis wird also im globalen Fall noch über $x \in A$ gemittelt. Der Vorteil 0, also die Wahrscheinlichkeit $\frac{1}{2^s}$, entspricht dem puren Raten des Ergebnisses.

C hat eine **Irrtumswahrscheinlichkeit** ε bei der Berechnung von $f(x)$ bzw. f, wenn

$$P(\{\omega \in \Omega \mid C(x, \omega) = f(x)\}) \geq 1 - \varepsilon \text{ bzw.}$$

$$P(\{(x, \omega) \in A \times \Omega \mid C(x, \omega) = f(x)\}) \geq 1 - \varepsilon.$$

Beispiele

1. Bei der Suche nach einem Nicht-Quadratrest $\bmod p$ ist

$$P(\{\omega \in \Omega \mid C(\omega) = 1\}) = 1 - \frac{1}{2^h}.$$

Das Schaltnetz hat also einen $(\frac{1}{2} - \frac{1}{2^h})$-Vorteil und eine Irrtumswahrscheinlichkeit von $\frac{1}{2^h}$.

2. Beim strengen Pseudoprimzahltest ist für festes m

$$P(\{\omega \in \Omega \mid C(m,\omega) = f(m)\}) \begin{cases} \geq \frac{3}{4}, & \text{wenn } m \text{ zusammengestzt,} \\ = 1, & \text{wenn } m \text{ prim.} \end{cases}$$

Das ergibt über alle m gemittelt

$$P(\{(m,\omega) \in A \times \Omega \mid C(m,\omega) = f(m\}) \geq \frac{3}{4},$$

also einen $\frac{1}{4}$-Vorteil und eine Irrtumswahrscheinlichkeit $\frac{1}{4}$. (Da es wesentlich mehr zusammengesetzte Zahlen als Primzahlen gibt, wird bei der Mittelung über m der Wert $\frac{1}{4}$ nicht wesentlich erhöht.)

7.5 Polynomiale Schaltnetzfamilien

Ein Schaltnetz hat eine festgelegte Zahl von Eingängen, kann also (im Gegensatz zu einer TURING-Maschine) nur Eingaben bestimmter Länge verarbeiten. Bei Effizienzuntersuchungen will man aber meist das Wachstum des Aufwandes bei immer weiterer Vergrößerung der Eingabelänge abschätzen. Dazu unterstellt man eine ganze Familie $(C_n)_{n\in\mathbb{N}}$ von Schaltnetzen mit wachsender Zahl deterministischer Eingänge, deren Größe $\#C_n$ kontrolliert wächst; damit kann man dann das Wachstum des Aufwandes als Funktion der Länge der Eingabe ausdrücken. Genauer wird definiert: Eine **polynomiale (probabilistische) Schaltnetzfamilie (PPS)** ist eine Familie $C = (C_n)_{n\in\mathbb{N}}$,

$$C_n \colon \mathbb{F}_2^{r(n)} \times \mathbb{F}_2^{k(n)} \longrightarrow \mathbb{F}_2^{s(n)}, \tag{1}$$

von (probabilistischen) Schaltnetzen mit $r(n)$ deterministischen und $k(n)$ probabilistischen Eingängen, so dass es ein Polynom $\alpha \in \mathbb{N}[X]$ (nichtnegative ganzzahlige Koeffizienten) gibt mit $\#C_n \leq \alpha(n)$ für alle $n \in \mathbb{N}$. Insbesondere ist die Zahl der Eingänge aller Arten und die Zahl $s(n)$ der Ausgänge polynomial beschränkt.

Bemerkung. Das bedeutet nicht notwendig, dass die Funktionen r, k, s selbst Polynome sind.

Durch dieses Berechnungsmodell (im deterministischen Fall) könnten mehr Probleme berechenbar sein als durch das gebräuchliche Modell der TURING-Maschinen (und sind es tatsächlich), da für jede Eingabelänge ein anderer Algorithmus gewählt werden kann. Man spricht daher auch von einem „nicht-gleichmäßigen Berechnungsmodell“. Das erscheint auf den ersten Blick vielleicht als Nachteil dieses Berechnungsmodells, ist aber für die Kryptoanalyse sogar besonders realistisch: Nach Wahl der Inputlänge n hat der Kryptoanalytiker die Möglichkeit, einen passenden Algorithmus zu wählen. Das heisst, Aussagen über Nichteffizienz, die unter diesem Berechnungsmodell bewiesen werden, erlauben, die Inputlänge als öffentlich bekannt anzunehmen.

Ist eine TURING-Berechnung in polynomialer Zeit möglich, so gibt es für das gleiche Problem auch eine polynomiale Schaltnetzfamilie. Die Umkehrung davon gilt nicht; es sind allerdings nur „künstliche“ Beispiele bekannt. Gäbe es für irgendein NP-vollständiges Problem eine polynomiale Schaltnetzfamilie, so für alle. Nichtgleichmäßige Komplexität kann man allerdings auch mit TURING-Maschinen modellieren, indem man für jede Eingabelänge eine andere TURING-Maschine zulässt. Analog kann man natürlich auch probabilistische TURING-Maschinen zulassen.

Ein Problem soll **hart** heissen, wenn es kein PPS gibt, das es mit signifikanter Wahrscheinlichkeit löst. Die früher behandelten „harten zahlentheoretischen Probleme“ sind vermutlich in diesem Sinne hart, z. B. die Primzerlegung; präzisiert wird das später.

Wir wissen bereits, dass die Grundoperationen für ganze Zahlen mit polynomialen Schaltnetzfamilien berechenbar sind (sogar deterministisch).

7.6 Effiziente Algorithmen

Um die Ergebnisse aus 7.4 übertragen zu können, müssen auch die Begriffe des Vorteils und der Irrtumswahrscheinlichkeit auf Familien verallgemeinert werden. Gegeben sei dazu eine Menge $L \subseteq \mathbb{F}_2^*$, also eine Sprache über dem binären Alphabet, wobei wie üblich $L_n := L \cap \mathbb{F}_2^n$ gesetzt wird, sowie eine Abbildung

$$f\colon L \longrightarrow \mathbb{F}_2^* \quad \text{mit } f(L_{r(n)}) \subseteq \mathbb{F}_2^{s(n)}, \tag{2}$$

wobei $r(n)$ die monoton wachsende Folge der Indizes i mit $L_i \neq \emptyset$ ist. Diese Abbildung soll mit einer PPS wie in (1) berechnet werden.

Beispiele

1. Die Funktion $f(x, y, z) := xy \bmod z$ für n-Bit-Zahlen x, y, z lässt sich mit einem (deterministischen) Schaltnetz

   $$C_n\colon \mathbb{F}_2^{3n} \longrightarrow \mathbb{F}_2^n$$

   der Größe $\#C_n = \mathrm{O}(n^3)$ (und Irrtumswahrscheinlichkeit 0) berechnen; hier ist $r(n) = 3n$ und $s(n) = n$.

2. Sei L die Menge der (binären Codierungen von) ungeraden Zahlen ≥ 3 und $f : L \longrightarrow \mathbb{F}_2$ der „Primzahlanzeiger“ wie in Abschnitt 7.4. Die dort vorgestellte PPS für den strengen Pseudoprimzahltest hat die Größe $\mathrm{O}(n^3)$ und konstanten Vorteil $\frac{1}{4}$ sowie konstante Irrtumswahrscheinlichkeit $\frac{1}{4}$. Mit t Basen kommt man auf die Größe $\mathrm{O}(tn^3)$ und die Irrtumswahrscheinlichkeit $\frac{1}{4^t}$.

Definition 1. Sei $f : L \longrightarrow \mathbb{F}_2^*$ wie in (2). Sei C eine PPS, die bei der Berechnung von f auf $\mathbb{F}_2^{r(n)}$ eine Irrtumswahrscheinlichkeit von ε_n hat, und für jedes nichtkonstante Polynom $\eta \in \mathbb{N}[X]$ sei

$$\varepsilon_n \leq \frac{1}{\eta(n)} \quad \text{für fast alle } n \in \mathbb{N}.$$

Dann heißt C **effizienter Algorithmus** für f.

f heißt effizient berechenbar, wenn es einen effizienten Algorithmus für f gibt.

Für den Primzahltest von Rabin, also die wiederholte Ausführung des strengen Pseudoprimzahltests, kann man diese Forderung erfüllen, wenn man die Zahl t der Basen mit n wachsen lässt; damit die Familie polynomial bleibt, nimmt man t als Polynom $\tau \in \mathbb{N}[X]$. Dann hat C_n n deterministische und $n\tau(n)$ probabilistische Eingänge, die Größe $\mathrm{O}(n^3\tau(n))$ und die Irrtumswahrscheinlichkeit $\frac{1}{4^{\tau(n)}}$. Damit ist gezeigt:

Satz 1 *Der Primzahltest von* Rabin *ist ein effizienter probabilistischer Algorithmus für die Bestimmung der Primzahleigenschaft.*

7.7 Harte Probleme

Etwas kniffliger ist die exakte Definition eines harten Problems. Es ist klar, dass die Forderung, das Problem solle für fast alle Eingaben nicht effizient lösbar sein, durch die schlichte Negierung der Eigenschaft „effizient“ nicht erfüllt wird. Näher kommt dem schon die Forderung, der Vorteil des Algorithmus solle mit wachsendem n gegen 0 gehen; aber auch das ist keine geeignete Definition, da der Vorteil nur eine untere Schranke ist. Der beste Ansatz ist, zu verlangen, dass es keinen Vorteil gibt, der „zu langsam“ gegen 0 geht. „Zu langsam“ soll bedeuten

$$\frac{1}{\eta(n)} \quad \text{mit einem beliebigen Polynom } \eta \in \mathbb{N}[X],$$

und es soll „fast keine“ Eingaben geben, die Ausnahmen sind – die Ausnahmemenge soll „dünn“ sein. Diese Vorstellung wird jetzt in eine exakte Definition umgesetzt.

Für $x \in L_{r(n)}$ betrachten wir die Wahrscheinlichkeit

$$p_x := P(\{\omega \in \Omega_{k(n)} \mid C_n(x,\omega) = f(x)\}),$$

ferner die Menge der Eingaben x, für die C_n einen ε-Vorteil hat:

$$L_{r(n)}(\varepsilon) := \{x \in L_{r(n)} \mid p_x \geq \frac{1}{2^{s(n)}} + \varepsilon\}.$$

Für ein Polynom $\eta \in \mathbb{N}[X]$ ist dann $L_{r(n)}(\frac{1}{\eta(n)})$ die Menge von Eingaben x, für die $f(x)$ von C mit Vorteil $\frac{1}{\eta(n)}$ berechnet wird. Die Ausnahmemenge für η ist damit

$$L^{[f,C,\eta]} := \bigcup_{n\in\mathbb{N}} L_{r(n)}(\frac{1}{\eta(n)}).$$

Wir bezeichnen sie als „**Vorteilsmenge für f, C und η**“. Ihre Bestandteile sollen mit wachsendem n immer unbedeutender werden, und das wird so definiert:

Definition 2. Eine Teilmenge $A \subseteq L$ heisst **dünn**, wenn für alle nichtkonstanten Polynome $\varphi \in \mathbb{N}[X]$ mit $A_n = A \cap L_n$ gilt

$$\#A_n \leq \frac{\#L_n}{\varphi(n)} \quad (\text{also } \varphi(n) \cdot \#A_n \leq \#L_n) \quad \text{für fast alle } n \in \mathbb{N}.$$

Bemerkungen und Beispiele

1. Ist $\#A_n = c$ konstant und $L_n = \mathbb{F}_2^n$, so ist A dünn in L, denn die verlangte Ungleichung ist $c \cdot \varphi(n) \leq 2^n$.

2. Wächst $\#A_n$ höchstens wie ein Polynom, aber $\#L_n$ schneller als jedes Polynom, so ist A dünn in L.

3. Ist $\#A_n = c \cdot \#L_n$ ein fester Anteil, so ist A nicht dünn in L.

4. Ist $L = \mathbb{N}$ und A die Menge der Primzahlen (beides binär codiert), so ist nach dem Primzahlsatz

$$\#A_n \approx \frac{2^{n-1}}{n \cdot \ln(2)} = \frac{\#L_n}{n \cdot \ln(2)}.$$

   Die Menge der Primzahlen ist also nicht dünn in $\mathbb{N}$.

5. Es ist kein effizienter Algorithmus bekannt, der mehr als eine dünne Teilmenge der Menge M aller Produkte von zwei Primzahlen, die sich in der Länge um höchstens ein Bit unterscheiden, faktorisieren kann.

Definition 3. Sei f wie in (2). Dann heißt f **hart**, wenn für jede PPS wie in (1) und für jedes Polynom $\eta \in \mathbb{N}[X]$ die Vorteilsmenge $L^{[f,C,\eta]}$ dünne Teilmenge von L ist.

Beispiele

1. Nach Bemerkung 5 ist die Primzerlegung vermutlich hart.

2. **Quadratrest-Vermutung:** Sei B die Menge von Produkten zweier Primzahlen $\equiv 3 \pmod 4$ (BLUM-Zahlen),

$$L = \{(m, a) \mid m \in B, 1 \leq a \leq m-1\},$$

$$f : L \longrightarrow \mathbb{F}_2$$

   die Funktion

$$f(m,a) = \begin{cases} 1, & \text{wenn } a \text{ Quadratrest mod } m, \\ 0 & \text{sonst.} \end{cases}$$

   Dann ist f hart.

7.8 Kryptographische Basisfunktionen

Damit haben wir auch die theoretische Grundlage, um Einwegfunktionen und starke symmetrische Chiffren exakt zu definieren:

Definition 4. Gegeben sei $f : L \longrightarrow \mathbb{F}_2^*$ wie in (2). Eine Rechtsinverse zu f ist eine Abbildung $g : f(L) \longrightarrow L \subseteq \mathbb{F}_2^*$ mit $f(g(y)) = y$ für alle $y \in f(L)$ – d. h., g findet Urbilder für f. f heißt **Einwegfunktion**, wenn jede Rechtinverse von f hart ist.

Nach dieser Definition ist die diskrete Exponentialfunktion in endlichen Primkörpern vermutlich Einwegfunktion.

Definition 5. Sei $f : L \longrightarrow \mathbb{F}_2^*$ eine Zusammensetzung von Abbildungen

$$f_n : \mathbb{F}_2^{r(n)} \times \mathbb{F}_2^{q(n)} \longrightarrow \mathbb{F}_2^{r(n)}$$

mit streng monoton wachsenden r und q, so dass $f_n(\bullet, k)$ für jedes $k \in \mathbb{F}_2^{q(n)}$ bijektiv ist und f sowie $(y, k) \mapsto f_n(\bullet, k)^{-1}(y)$ jeweils effizient berechenbar sind. Ein Angriff auf f mit bekanntem Klartext ist eine Abbildung g, zusammengesetzt aus Teilen

$$g_n : \mathbb{F}_2^{r(n)} \times \mathbb{F}_2^{r(n)} \longrightarrow \mathbb{F}_2^{q(n)}$$

mit

$$f_n(x, g_n(x, y)) = y \quad \text{für alle } x, y \in \mathbb{F}_2^{r(n)}.$$

f heißt **starke symmetrische Chiffre**, wenn jeder Angriff auf f mit bekanntem Klartext hart ist.

Die Definition einer Hash-Funktion ist etwas kniffliger und wird hier nicht ausgeführt.

A Primitive Elemente und Quadratreste

Eine ganze Zahl a heißt **primitives Element** mod n, wenn a mod n in der multiplikativen Gruppe $\mathbb{M}_n = (\mathbb{Z}/n\mathbb{Z})^\times$ des Restklassenrings $\mathbb{Z}/n\mathbb{Z}$ invertierbar ist und seine Ordnung maximal, also gleich $\lambda(n)$, dem Exponenten der Gruppe $\mathbb{M}_n$, also der CARMICHAEL-Funktion von n ist.

Wie findet man ein primitives Element? Hier sind zunächst die einfachsten Fälle zu behandeln, nämlich wenn der Modul n eine Primzahl oder eine Zweierpotenz ist, und zwar der leichtere Fall der Zweierpotenz zuerst.

Die Quadratreste mod n sind die $x \in \mathbb{Z}$ mit $\mathrm{ggT}(x, n) = 1$, für die $x \equiv z^2 \pmod{n}$ für ein $z \in \mathbb{Z}$. Die Quadratrest-Eigenschaft ist für Primzahl-Moduln effizient bestimmbar, sonst nicht.

Beide Themen sind eng mit der Struktur der multiplikativen Gruppe $\mathbb{M}_n$ verknüpft.

A.1 Primitive Elemente für Zweierpotenzen

Die Fälle $n = 2$ oder 4 sind trivial: $\mathbb{M}_2$ ist die einelementige Gruppe, $\mathbb{M}_4$ zyklisch von der Ordnung 2. Sei also bis auf weiteres $n = 2^e$ mit $e \geq 3$. Da $\mathbb{M}_n$ dann gerade aus den Restklassen der ungeraden Zahlen besteht, ist $\varphi(n) = 2^{e-1}$. Wir benötigen zwei Hilfssätze.

Hilfssatz 1 *Sei $n = 2^e$ mit $e \geq 2$.*

(i) *Ist a ungerade, so*

$$a^{2^s} \equiv 1 \pmod{2^{s+2}} \quad \textit{für alle } s \geq 1.$$

(ii) *Ist $a \equiv 3 \pmod 4$, so $n \mid 1 + a + \cdots + a^{n/2-1}$.*

Beweis. (i) Ist $a = 4q + 1$, so $a^2 = 16q^2 + 8q + 1$; ist $a = 4q + 3$, so $a^2 = 16q^2 + 24q + 9 \equiv 1 \pmod 8$. Damit ist die Behauptung für $s = 1$ bewiesen. Der allgemeine Fall folgt durch Induktion:

$$a^{2^{s-1}} = 1 + t2^{s+1} \Longrightarrow a^{2^s} = (a^{2^{s-1}})^2 = 1 + 2t2^{s+1} + t^2 2^{2s+2}.$$

(ii) Es ist $2n = 2^{e+1} \mid a^{n/2} - 1$ nach (i). Da in $a - 1$ nur die erste Potenz von 2 aufgeht, folgt

$$n = 2^e \mid \frac{a^{n/2} - 1}{a - 1},$$

wie behauptet. $\diamond$

Hilfssatz 2 *Sei p eine Primzahl und e eine natürliche Zahl mit $p^e \geq 3$. Sei p^e die größte p-Potenz, die in $x - 1$ aufgeht. Dann ist p^{e+1} die größte p-Potenz, die in $x^p - 1$ aufgeht.*

Beweis. Es ist $x = 1 + tp^e$ mit einer ganzen Zahl t, die kein Vielfaches von p ist. Nach der Binomialformel ist

$$x^p = 1 + \sum_{k=1}^{p} \binom{p}{k} t^k p^{ke}.$$

Da p alle Binomialkoeffizienten $\binom{p}{k} = \frac{p!}{k!(p-k)!}$ für $k = 1, \ldots, p-1$ teilt, kann man aus der Summe sogar p^{e+1} ausklammern:

$$x^p = 1 + tp^{e+1}s$$

mit einer ganzen Zahl s. Also geht p^{e+1} in $x^p - 1$ auf. Zu zeigen ist noch, dass s kein Vielfaches von p ist. Dazu sieht man sich s genauer an:

$$\begin{aligned} s &= \sum_{k=1}^{p} \frac{1}{p}\binom{p}{k} \cdot t^{k-1} p^{e(k-1)} \\ &= 1 + \frac{1}{p}\binom{p}{2} \cdot tp^e + \cdots + \frac{1}{p} \cdot t^{p-1} p^{e(p-1)}. \end{aligned}$$

Da $p^e \geq 3$, ist $e(p-1) \geq 2$, also $s \equiv 1 \pmod{p}$. $\diamond$

Aus Hilfssatz 1 folgt insbesondere

$$a^{2^{e-2}} \equiv 1 \pmod{n} \quad \text{für alle ungeraden } a.$$

Daher ist der Exponent $\lambda(n) \leq 2^{e-2}$ und $\mathbb{M}_n$ schon mal nicht zyklisch. Genauer gilt:

Satz 1 *Sei $n = 2^e$ mit $e \geq 3$. Dann gilt:*

(i) *In $G = \mathbb{M}_n$ hat -1 die Ordnung 2 und 5 die Ordnung 2^{e-2}, und G ist das direkte Produkt der von -1 und 5 erzeugten zyklischen Gruppen.*

(ii) *Wenn $e \geq 4$, sind die primitiven Elemente* mod n *genau die ganzen Zahlen $a \equiv 3, 5 \pmod{8}$. Insbesondere ist ihre Anzahl $n/4$.*

Beweis. (i) Da $\operatorname{Ord} 5 \mid 2^e$ und $\operatorname{Ord} 5 \leq 2^e - 2$, ist $\operatorname{Ord} 5$ eine Zweierpotenz $\leq 2^{e-2}$. Da 2^2 die größte Zweierpotenz in $5-1$ ist, ist 2^3 die größte Zweierpotenz in $5^2 - 1$ (nach Hilfssatz 2) und sukzessive 2^{e-1} die größte Zweierpotenz in $5^{2^{e-3}} - 1$. Also ist die 2^{e-2}-te Potenz von 5 die kleinste, die $\equiv 1 \pmod{2^e}$ ist. Das Produkt der beiden Untergruppen ist direkt, weil -1 keine Potenz von 5 ist – wäre $5^k \equiv -1 \pmod{n}$, so wegen $e \geq 2$ auch $5^k \equiv -1 \pmod{4}$, Widerspruch zu $5 \equiv 1 \pmod{4}$. Das direkte Produkt ist ganz G, weil es die Ordnung $2 \cdot 2^{e-2}$ hat.

(ii) Jedes Element $a \in G$ lässt sich nach (i) eindeutig in der Form $a = (-1)^r 5^s$ schreiben mit $r = 0$ oder 1 und $0 \leq s < 2^{e-2}$. Damit ist a^k genau dann 1 in $\mathbb{Z}/n\mathbb{Z}$, wenn kr gerade und ks ein Vielfaches von 2^{e-2} ist, also wenn ks ein Vielfaches von 2^{e-2} ist. Also ist a genau dann primitiv, wenn s ungerade ist, und das bedeutet $a \equiv \pm 5 \pmod{8}$. $\diamond$

Insbesondere ist $\lambda(2^e) = 2^{e-2}$ für $e \geq 4$ und $\lambda(8) = 2$.

A.2 Primitive Elemente für Primzahlmoduln

Schwieriger (und mathematisch interessanter) ist die Suche nach den primitiven Elementen für einen Primzahlmodul. Man kann sich dabei auf die Suche nach *einem* solchen Element beschränken; alle übrigen erhält man durch Potenzieren mit einem Exponenten, der zu $p-1$ teilerfremd ist, insbesondere gibt es genau $\varphi(p-1)$ Stück davon. Im üblichen Sprachgebrauch nennt man die primitiven Elemente zu einem Modul n im Falle, dass $\mathbb{M}_n$ zyklisch ist, auch **Primitivwurzeln** $\bmod n$. Das Problem, eine Primitivwurzel $\bmod p$ zu finden, hat schon Gauss beschäftigt.

Die einfachste, aber noch nicht bestmögliche Methode heißt: Probiere $x = 2, 3, 4, \ldots$; teste, ob $x^d \neq 1$ für jeden echten Teiler d von $p-1$. Die Anzahl der dazu nötigen Tests wird verringert durch:

Hilfssatz 3 *Sei p eine Primzahl ≥ 5. Eine ganze Zahl x ist genau dann Primitivwurzel* $\bmod p$*, wenn für jeden Primfaktor q von $p-1$ gilt $x^{(p-1)/q} \neq 1$ in $\mathbb{F}_p$.*

Beweis. Die Ordnung von x ist ein Teiler von $p-1$, und jeder echte Teiler von $p-1$ teilt einen solchen Quotienten $\frac{p-1}{q}$. $\diamond$

Um dieses Kriterium anwenden zu können, braucht man also die Primzerlegung von $p-1$.

Beispiel. Für $p = 41$ ist $p-1 = 40 = 2^3 \cdot 5$, also x genau dann Primitivwurzel, wenn $x^{20} \neq 1$ und $x^8 \neq 1$. In $\mathbb{F}_{41}$ laufen dann folgende Rechenschritte ab:

$$
\begin{array}{lllll}
x = 2: & x^2 = 4, & x^4 = 16, & \left\{ \begin{array}{l} x^8 = 10, \\ x^{20} = x^8 x^8 x^4 = 1. \end{array} \right. \\
x = 3: & x^2 = 9, & x^4 = 81, & x^4 = -1, \ x^8 = 1. \\
x = 4: & x = 2^2, & \text{also} & x^{20} = 1. \\
x = 5: & x^2 = 25, & x^4 = 10 & \left\{ \begin{array}{l} x^8 = 18, \\ x^{20} = x^8 x^8 x^4 = 1. \end{array} \right. \\
x = 6: & x^2 = 36, & x^4 = 25 & \left\{ \begin{array}{l} x^8 = 10, \\ x^{20} = x^8 x^8 x^4 = -1. \end{array} \right.
\end{array}
$$

Also ist 6 *Primitivwurzel für $p = 41$.*

Es stellt sich die Frage, wie lange man auf diese Weise nach einer Primitivwurzel suchen muss. Hier einige Aspekte. Sei

$$\alpha(p) := \min\{x \in \mathbb{N} \mid x \text{ ist Primitivwurzel für } p\}.$$

Dann kann man zeigen: Die Funktion α ist nicht beschränkt. Nach einem Ergebnis von Burgess 1962 ist (etwas vergröbert)

$$\alpha(p) = \mathrm{O}(\sqrt[6]{p}).$$

Falls die erweiterte RIEMANNsche Vermutung richtig sein sollte, kann man diese immer noch exponentielle Schranke zu einer polynomialen verbessern; das beste bekannte Ergebnis scheint von SHOUP 1990 zu stammen und besagt etwas vergröbert:

$$\alpha(p) = \mathrm{O}(\log(p)^6(1 + \log\log(p))^4).$$

Es gibt einige weitere offene Probleme:

- Ist 2 für unendlich viele Primzahlen Primitivwurzel?
- Ist 10 für unendlich viele Primzahlen Primitivwurzel? Das wurde von GAUSS vermutet.

Allgemeiner besagt eine *Vermutung von* ARTIN: Sei $a \in \mathbb{N}$, a kein Quadrat (also $a \neq 0, 1, 4, 9, \ldots$). Dann ist a für unendlich viele Primzahlen Primitivwurzel.

Relevante Literatur:

- D. R. HEATH-BROWN: Artin's conjecture for primitive roots. Quart. J. Math. Oxford 37 (1986), 27–38.
- M. RAM MURTY: Artin's conjecture for primitive roots. Math. Intelligencer 10 (1988), 59–67.
- V. SHOUP: Searching for primitive roots in finite fields. Proc. 22nd STOC 1990, 546–554.
- MURATA: On the magnitude of the least prime primitive root. J. Number Theory 37 (1991), 47–66.

A.3 Primitive Elemente für Primpotenzen

Für die Primzahlpotenzen braucht man einen weiteren Hilfssatz.

Hilfssatz 4 *Sei p eine Primzahl ≥ 3, k eine ganze Zahl und $d \geq 0$. Dann gilt*

$$(1 + kp)^{p^d} \equiv 1 + kp^{d+1} \pmod{p^{d+2}}.$$

Beweis. Für $d = 0$ ist die Aussage trivialerweise richtig. Weiter schließt man durch Induktion: Sei $d \geq 1$ und

$$(1 + kp)^{p^{d-1}} = 1 + kp^d + rp^{d+1}.$$

Dann folgt

$$(1+kp)^{p^d} = (1+(k+rp)p^d)^p \equiv 1+p\cdot(k+rp)\cdot p^d \equiv 1+kp^{d+1} \pmod{p^{d+2}},$$

da $d + 2 \leq 2d + 1$ und $p \geq 3$. $\diamond$

Satz 2 *Sei p eine Primzahl ≥ 3, e ein Exponent ≥ 2 und a eine Primitivwurzel* mod p. *Dann gilt:*

(i) *Genau dann erzeugt a die Gruppe $\mathbb{M}_{p^e}$, wenn $a^{p-1} \bmod p^2 \neq 1$.*

(ii) *a oder $a + p$ erzeugt $\mathbb{M}_{p^e}$.*

(iii) *$\mathbb{M}_{p^e}$ ist zyklisch und $\lambda(p^e) = \varphi(p^e) = p^{e-1}(p - 1)$.*

Beweis. (i) Sei t die multiplikative Ordnung von $a \bmod p^e$. Sie ist sicher ein Vielfaches der von $a \bmod p$, also von $p - 1$. Andererseits teilt sie $\varphi(p^e) = p^{e-1}(p-1)$. Daher ist $t = p^d(p-1)$ mit $0 \leq d \leq e-1$. Ist nun $a^{p-1} = 1+kp$, so nach Hilfssatz 4

$$(a^{p-1})^{p^{e-2}} \equiv 1 + kp^{e-1} \equiv 1 \pmod{p^e} \iff p|k \iff a^{p-1} \equiv 1 \pmod{p^2}.$$

Das ist genau dann nicht der Fall, wenn $d = e - 1$.

(ii) Erzeugt a nicht $\mathbb{M}_{p^e}$, so ist $a^{p-1} \equiv 1 \pmod{p^2}$, also

$$(a + p)^{p-1} \equiv a^{p-1} + (p - 1)a^{p-2}p \equiv 1 - a^{p-2} \pmod{p^2},$$

und dies ist sicher nicht $\equiv 1 \pmod{p^2}$.

(iii) folgt direkt aus (ii). $\diamond$

Daraus lässt sich unmittelbar eine analoge Aussage für das Zweifache einer Primzahlpotenz geweinnen:

Korollar 1 *Sei $q = p^e$ eine Potenz der Primzahl $p \geq 3$. Dann gilt:*

(i) *Die multiplikative Gruppe* $\mathbb{M}_{2q}$ *ist natürlich isomorph zu* $\mathbb{M}_q$*, also zyklisch.*

(ii) *Ist* a *Primitivwurzel* $\bmod q$*, so ist* a *Primitivwurzel* $\bmod 2q$*, falls* a *ungerade, und* $a+q$ *Primitivwurzel* $\bmod 2q$*, falls* a *gerade.*

(iii) $\lambda(2p^e) = p^{e-1}(p-1)$.

Beweis. (i) Da q und 2 teilerfremd sind und $\mathbb{M}_2$ die triviale Gruppe ist, ist nach dem chinesischen Restsatz $\mathbb{M}_{2q} \cong \mathbb{M}_2 \times \mathbb{M}_q \cong \mathbb{M}_q$. Die Abbildung ist explizit durch $a \bmod 2q \mapsto a \bmod q$ gegeben.

(ii) Die Umkehrabbildung des natürlichen Isomorphismus ist

$$a \mapsto \begin{cases} a, & \text{falls } a \text{ ungerade,} \\ a+q, & \text{falls } a \text{ gerade.} \end{cases}$$

(iii) klar. $\diamond$

A.4 Die Struktur der multiplikativen Gruppe

Damit können wir genau sagen, wann die multiplikative Gruppe $\mathbb{M}_n$ zyklisch ist (d. h., wann eine Primitivwurzel mod n existiert):

Korollar 2 (GAUSS 1799) *Die multiplikative Gruppe $\mathbb{M}_n$ ist für $n \geq 2$ genau dann zyklisch, wenn n eine der Zahlen $2, 4, p^e$ oder $2p^e$ mit einer ungeraden Primzahl p ist.*

Beweis. Das folgt aus Satz 2, Korollar 1 und dem folgenden Hilfssatz 5. ◇

Hilfssatz 5 *Sind $m, n \geq 3$ teilerfremd, so ist $\mathbb{M}_{mn}$ nicht zyklisch und $\lambda(mn) < \varphi(mn)$.*

Beweis. Ist $n \geq 3$, so $\varphi(n)$ gerade; für eine Primzahlpotenz folgt das aus der expliziten Form und allgemein aus der Multiplikativität der φ-Funktion. Damit folgt

$$\mathrm{kgV}(\varphi(m), \varphi(n)) < \varphi(m)\varphi(n) = \varphi(mn),$$

$$\lambda(mn) = \mathrm{kgV}(\lambda(m), \lambda(n)) \leq \mathrm{kgV}(\varphi(m), \varphi(n)) < \varphi(mn).$$

Also ist $\mathbb{M}_{mn}$ nicht zyklisch. ◇

Damit ist die Struktur der multiplikativen Gruppe auch im allgemeinen Fall bekannt; mit $\mathcal{Z}_d$ wird dabei die zyklische Gruppe der Ordnung d bezeichnet.

Satz 3 *Sei $n = 2^e p_1^{e_1} \cdots p_r^{e_r}$ die Primzerlegung der natürlichen Zahl $n \geq 2$ mit $e \geq 0$, $r \geq 0$, $e_1, \ldots e_r \geq 1$ und verschiedenen ungeraden Primzahlen $p_1, \ldots, p_r$. Sei $q_i = p_i^{e_i}$ und $q_i' = p_i^{e_i - 1}(p_i - 1)$ für $i = 1, \ldots, r$. Dann ist*

$$\mathbb{M}_n \cong \begin{cases} \mathcal{Z}_{q_1'} \times \cdots \times \mathcal{Z}_{q_r'}, & \text{falls } e = 0 \text{ oder } 1, \\ \mathcal{Z}_2 \times \mathcal{Z}_{2^{e-2}} \times \mathcal{Z}_{q_1'} \times \cdots \times \mathcal{Z}_{q_r'}, & \text{falls } e \geq 2. \end{cases}$$

Ein primitives Element $a \bmod n$ findet man, indem man primitive Elemente $a_0 \bmod 2^e$ (falls $e \geq 2$) und $a_i \bmod q_i$ wählt und die simultanen Kongruenzen $a \equiv a_i \pmod{q_i}$, gegebenenfalls $a \equiv a_0 \pmod{2^e}$, löst.

Beweis. All dies folgt jetzt aus dem chinesischen Restsatz. ◇

A.5 Das JACOBI-Symbol

Auf der multiplikativen Gruppe $\mathbb{M}_n = (\mathbb{Z}/n\mathbb{Z})^\times$ für einen Modul $n \geq 2$ ist die Quadrat-Abbildung

$$\mathbf{q} : \mathbb{M}_n \longrightarrow \mathbb{M}_n, \quad x \mapsto x^2 \bmod n,$$

ein Gruppen-Homomorphismus. Die Elemente im Bild von $\mathbf{q}$ heißen die **Quadratreste** mod n. Eine ganze Zahl x ist also Quadratrest mod n, wenn sie mod n invertierbar ist und es eine ganze Zahl u mit $u^2 \equiv x \pmod{n}$ gibt. Die Menge der Quadratreste – als Teilmenge des Restklassenrings $\mathbb{Z}/n\mathbb{Z}$ aufgefasst – ist also $\mathbb{M}_n^2$. (Das ist allerdings keine Standard-Bezeichnung, so wenig wie $\mathbb{M}_n$. Aber jedesmal $((\mathbb{Z}/n\mathbb{Z})^\times)^2$ zu schreiben ist nicht sehr angenehm.)

Bemerkungen und Beispiele

1. Im Fall $n = 2$ ist $\mathbb{M}_n^2 = \mathbb{M}_n = \{1\}$.

2. Für $n \geq 3$ ist $\mathbf{q}$ sicher nicht surjektiv, da $(-1)^2 = 1$; es gibt also Zahlen, die nicht Quadratrest sind.

3. Sei $n = p \geq 3$ eine Primzahl. Dann besteht der Kern von $\mathbf{q}$ genau aus den Nullstellen des Polynoms $X^2 - 1$ im Körper $\mathbb{F}_p$, also aus $\{\pm 1\}$. Daher gibt es genau $\frac{p-1}{2}$ Quadratreste.

4. Allgemeiner sei $n = q = p^e$ Potenz einer ungeraden Primzahl p. Dann ist $\mathbb{M}_n$ zyklisch von der Ordnung $\varphi(q) = q \cdot (1 - \frac{1}{p})$ nach Satz 2. Also hat 1 in $\mathbb{M}_q$ genau die Quadratwurzeln ± 1, und es gibt $\varphi(q)/2$ Quadratreste.

5. Sei n das Produkt zweier verschiedener ungerader Primzahlen p und q. Der chinesische Restsatz sagt dann, dass die natürliche Abbildung $\mathbb{M}_n \longrightarrow \mathbb{M}_p \times \mathbb{M}_q$ ein Isomorphismus ist. Also gibt es in $\mathbb{M}_n$ genau 4 Quadratwurzeln aus 1, und $\mathbb{M}_n^2$ hat den Index 4 in $\mathbb{M}_n$.

6. Ganz allgemein sei $n = 2^e p_1^{e_1} \cdots p_r^{e_r}$ die Primzerlegung mit $e \geq 0$, verschiedenen ungeraden Primzahlen $p_1, \ldots, p_r$ und $e_1, \ldots, e_r \geq 1$. Aus Satz 3 kann man ablesen, dass folgendes die Anzahl der Quadratwurzeln aus 1 in $\mathbb{M}_n$ ist:

$$\begin{array}{ll} 2^r, & \text{falls } e = 0 \text{ oder } 1, \\ 2^{r+1}, & \text{falls } e = 2, \\ 2^{r+2}, & \text{falls } e \geq 3. \end{array}$$

Der naive Algorithmus zur Bestimmung der Quadratrest-Eigenschaft von a mod n probiert der Reihe nach $1^2, 2^2, 3^2, \ldots$, bis a gefunden ist. Dazu sind

für einen Nicht-Quadratrest stets $n/2$, für einen Quadratrest im Durchschnitt $n/4$ Schritte nötig. Der Aufwand wächst also exponentiell mit der Stellenzahl $\log n$. Für den Fall, dass n eine *Primzahl* ist, werden bessere Algorithmen hergeleitet. Das Phänomen, dass es für eine *zusammengesetzte* Zahl n keinen effizienten Algorithmus gibt, wird Grundlage für die Konstruktion des einfachsten perfekten Zufallsgenerators sein.

Im Falle eines Primzahlmoduls p dient das LEGENDRE-**Symbol** als Anzeiger der Quadratrest-Eigenschaft:

$$(\frac{x}{p}) = \begin{cases} 1, & \text{wenn } x \text{ Quadratrest,} \\ 0, & \text{wenn } p|x, \\ -1 & \text{sonst.} \end{cases}$$

Das LEGENDRE-Symbol definiert also insbesondere einen Homomorphismus

$$(\frac{\bullet}{p}) : \mathbb{M}_p \longrightarrow \mathbb{M}_p/\mathbb{M}_p^2 \cong \{\pm 1\}.$$

Im Spezialfall $p = 2$ ist

$$(\frac{x}{2}) = \begin{cases} 1, & \text{wenn } x \text{ ungerade,} \\ 0, & \text{wenn } x \text{ gerade.} \end{cases}$$

Satz 4 (EULER-Kriterium)) *Sei p eine ungerade Primzahl. Dann ist*

$$x^{\frac{p-1}{2}} \equiv (\frac{x}{p}) \pmod{p} \quad \textit{für alle } x.$$

Beweis. Falls $p|x$, sind beide Seiten 0. Andernfalls ist $(x^{\frac{p-1}{2}})^2 = x^{p-1} \equiv 1$, also $x^{\frac{p-1}{2}} \equiv \pm 1$. Sei nun a ein primitives Element $\bmod p$. Dann ist die Behauptung für $x = a$ richtig – beide Seiten sind -1. Da ferner beide Seiten der Behauptung Homomorphismen $\mathbb{F}_p^\times \longrightarrow \{\pm 1\}$ definieren, folgt die Behauptung auch für alle x, die nicht Vielfache von p sind. $\diamond$

Das EULER-Kriterium ergibt schon einen effizienten Algorithmus zur Entscheidung der Quadratrest-Eigenschaft für einen Primzahlmodul: Man hat $\bmod p$ mit $\frac{p-1}{2}$ zu potenzieren. Bekanntlich kann man das mit höchstens $2\lfloor {}^2\log(\frac{p-1}{2})\rfloor$ Multiplikationen $\bmod p$. Berücksichtigt man den Aufwand für die modularen Multiplikationen, kommt man in die Größenordnung ${}^2\log(p)^3$.

Nach dem EULER-Kriterium ist -1 genau dann ein Quadratrest, wenn $\frac{p-1}{2}$ gerade, also $p \equiv 1 \pmod 4$ ist. Aber schon die Entscheidung, ob 2 oder 3 Quadratrest ist, fällt immer noch schwer. Der Algorithmus lässt sich aber noch verbessern, und das wird im folgenden Abschnitt A.6 beschrieben.

Das LEGENDRE-Symbol wird durch das genauso geschriebene JACOBI-Symbol verallgemeinert: Ist $n > 0$ und $n = p_1 \cdots p_r$ die Primzerlegung, so

$$(\frac{x}{n}) := (\frac{x}{p_1}) \cdots (\frac{x}{p_r}) \quad \text{für } x \in \mathbb{M}_n,$$

insbesondere $(\frac{x}{n}) = 0$, wenn x und n nicht teilerfremd sind. Ergänzt wird das durch $(\frac{x}{1}) = 1$, $(\frac{x}{n}) = (\frac{x}{-n})$, wenn $n < 0$, und $(\frac{x}{0}) = 0$. Dann ist das JACOBI-Symbol eine Funktion

$$(\frac{\bullet}{\bullet}) : \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z}$$

mit Werten in $\{0, \pm 1\}$, und zwar in Zähler und Nenner multiplikativ. Insbesondere definiert das JACOBI-Symbol ebenfalls noch einen Homomorphismus $(\frac{\bullet}{n})$ von $\mathbb{M}_n$ nach $\{\pm 1\}$, ist aber *kein* Anzeiger für die Quadratrest-Eigenschaft mehr. Ist also $\mathbb{M}_n^+ = Kern(\frac{\bullet}{n})$ und $\mathbb{M}_n^- = \mathbb{M}_n - \mathbb{M}_n^+$, so ist $\mathbb{M}_n^2$ im allgemeinen echte Untergruppe von $\mathbb{M}_n^+$. Ihren Index kann man aus dem obigen Beispiel 6 bestimmen: Gibt es 2^k Quadratwurzeln aus 1 und ist dabei $k \geq 1$, so hat $\mathbb{M}_n^2$ in $\mathbb{M}_n^+$ den Index 2^{k-1}. Stets hängt $(\frac{x}{n})$ nur von der Restklasse x mod n ab. Klar ist

$$(\frac{x}{2^k}) = \begin{cases} 1, & \text{wenn } x \text{ ungerade,} \\ 0, & \text{wenn } x \text{ gerade.} \end{cases}$$

A.6 Das quadratische Reziprozitätsgesetz

Die Grundlage zur Berechnung des LEGENDRE- (und JACOBI-) -Symbols sind die folgenden beiden Sätze; zunächst aber ein Hilfssatz, mit dem sich zusammengesetzte Moduln auf Primzahlen reduzieren lassen.

Hilfssatz 6 *Seien $s, t \in \mathbb{Z}$ ungerade. Dann gilt*

(i) $\frac{s-1}{2} + \frac{t-1}{2} \equiv \frac{st-1}{2} \pmod 2$,

(ii) $\frac{s^2-1}{8} + \frac{t^2-1}{8} \equiv \frac{s^2t^2-1}{8} \pmod 2$.

Beweis. Ist $s = 2k+1$ und $t = 2l+1$, so $st = 4kl + 2k + 2l + 1$,

$$\frac{st-1}{2} = 2kl + k + l.$$

Ferner $s^2 = 4{\cdot}(k^2{+}k){+}1$, $t^2 = 4{\cdot}(l^2{+}l){+}1$, $s^2t^2 = 16{\cdot}\ldots{+}4{\cdot}(k^2{+}k{+}l^2{+}l){+}1$,

$$\frac{s^2t^2-1}{8} = 2 \cdot \ldots + \frac{k^2+k+l^2+l}{2},$$

und daraus folgt direkt die Behauptung. $\diamond$

Satz 5 *Sei n ungerade. Dann gilt:*

(i) $(\frac{-1}{n}) = (-1)^{\frac{n-1}{2}}$,

(ii) $(\frac{2}{n}) = (-1)^{\frac{n^2-1}{8}}$

Beweis. Wendet man den Hilfssatz auf die Primzerlegung von n an, so darf man o. B. d. A. für beide Behauptungen $n = p$ prim annehmen.

(i) folgt direkt aus dem EULER-Kriterium, Satz 4.

(ii) Es ist

$$(-1)^k \cdot k \equiv \begin{cases} k, & \text{falls } k \text{ gerade,} \\ p-k, & \text{falls } k \text{ ungerade,} \end{cases}$$

$$\prod_{k=1}^{\frac{p-1}{2}} (-1)^k \cdot k \equiv 2 \cdot 4 \cdots p-1 = 2^{\frac{p-1}{2}} \cdot (\frac{p-1}{2})!.$$

Andererseits ist

$$\prod_{k=1}^{\frac{p-1}{2}} (-1)^k \cdot k = (\frac{p-1}{2})! \cdot (-1)^{\frac{p^2-1}{8}}, \quad \text{da} \quad \sum_{k=1}^{\frac{p-1}{2}} k = \frac{(p-1)(p+1)}{2 \cdot 2 \cdot 2}.$$

Da $(\frac{p-1}{2})!$ Produkt von Zahlen $< p$ ist, ist es kein Vielfaches von p, darf also wegdividiert werden. Durch Gleichsetzen und mit dem EULER-Kriterium folgt also

$$(-1)^{\frac{p^2-1}{8}} \equiv 2^{\frac{p-1}{2}} \equiv (\frac{2}{p}) \pmod{p}.$$

Da $p \geq 3$, folgt aus der Kongruenz die Gleichheit. $\diamond$

Insbesondere ist 2 genau dann Quadratrest modulo der Primzahl p, wenn $(p^2-1)/8$ gerade, also $p^2 \equiv 1 \pmod{16}$, also $p \equiv 1$ oder $7 \pmod 8$.

Hauptsatz 1 (Quadratisches Reziprozitätsgesetz) *Für je zwei verschiedene ungerade und zueinander teilerfremde natürliche Zahlen m und n gilt*

$$(\frac{m}{n})(\frac{n}{m}) = (-1)^{\frac{m-1}{2}\frac{n-1}{2}}.$$

Eine etwas leichter verständliche Form des quadratischen Reziprozitätsgesetzes ist:

$$(\frac{m}{n}) = \begin{cases} -(\frac{n}{m}) & \text{wenn } m \equiv n \equiv 3 \pmod 4, \\ (\frac{n}{m}) & \text{sonst.} \end{cases}$$

Der Beweis folgt. Zunächst wird die Berechnung an einem Beispiel gezeigt: Ist 7 Quadratrest mod 107? Dies wird durch folgende Rechnung verneint:

$$(\frac{7}{107}) = -(\frac{107}{7}) = -(\frac{2}{7}) = -1.$$

Genauso ist 7 kein Quadratrest mod 11:

$$(\frac{7}{11}) = -(\frac{11}{7}) = -(\frac{4}{7}) = -(\frac{2}{7})(\frac{2}{7}) = -1.$$

Also ist 7 auch nicht Quadratrest mod 1177, da $1177 = 11 \cdot 107$. Aber $(\frac{7}{1177}) = 1$.

Der aus dem quadratischen Reziprozitätsgesetz abgeleitete Algorithmus besteht dann aus der folgenden Prozedur:

Prozedur JacobiSymbol

Eingabeparameter
m, n = zwei ganze Zahlen.
Ausgabeparameter
jac $= (\frac{m}{n})$.
Anweisungen
Falls $n = 0$, gib jac $= 0$ aus. **Ende**
Falls $m = 0$, gib jac $= 0$ aus. **Ende**
Falls ggT$(m, n) > 1$, gib jac $= 0$ aus. **Ende**

[Jetzt sind $m, n \neq 0$ teilerfremd, also jac $= \pm 1$.]
jac $= 1$.
Falls $n < 0$, ersetze n durch $-n$.
Falls n gerade, dividiere n durch die größtmögliche Zweierpotenz 2^k.
Falls $m < 0$,
    ersetze m durch $-m$,
    falls $n \equiv 3 \pmod 4$, ersetze jac durch $-$jac.
[Ab jetzt sind m und n teilerfremd und n ist positiv und ungerade;]
[außer dass am Ende der Fall $m = 0$ und $n = 1$ eintreten kann.]
Falls $m > n$, ersetze m durch $m \bmod n$.
Solange $n > 1$:
    Falls m gerade:
        Dividiere m durch die größtmögliche Zweierpotenz 2^k.
        Falls (k ungerade und $n \equiv \pm 3 \pmod 8$) ersetze jac durch $-$jac.
    [Jetzt sind m und n ungerade und teilerfremd, $0 < m < n$.]
    [Das quadratische Reziprozitätsgesetz ist anwendbar.]
    Falls ($m \equiv 3 \pmod 4$ und $n \equiv 3 \pmod 4$)
        ersetze jac durch $-$jac.
    Setze $d = m$, $m = n \bmod m$, $n = d$.

Dieser Algorithmus lässt sich ähnlich wie der Euklidische Algorithmus analysieren: Man benötigt höchstens $5 \cdot \log(m)$ Schritte, wobei jeder Schritt im wesentlichen aus einer Ganzzahl-Division besteht. Da die Größe der Operanden dabei schnell abnimmt, kommt man insgesamt auf einen Aufwand von $O(\log(m)^2)$. Das ist deutlich schneller als die Anwendung des Euler-Kriteriums.

A.7 Beweis des quadratischen Reziprozitätsgesetzes

Nun zum Beweis des quadratischen Reziprozitätsgesetzes. Von den vielen bekannten Beweisen wird hier einer durchgeführt, der auf der Theorie der endlichen Körper beruht, auf Ideen von Zolotarev (Nouvelles Annales de Mathematiques 11 (1872), 354–362) und Swan (Pacific J. Math. 12 (1962), 1099–1106) beruht.

Hilfssatz 7 *Sei p eine ungerade Primzahl und a zu p teilerfremd. Dann sind äquivalent:*

(i) *a ist Quadratrest* mod p.
(ii) *Die Multiplikation mit a ist eine gerade Permutation von $\mathbb{F}_p$.*

Beweis. Die Multiplikation sei mit $\mu_a : \mathbb{F}_p \longrightarrow \mathbb{F}_p$, $x \mapsto ax \bmod p$, bezeichnet. Dann ist $a \mapsto \mu_a$ ein injektiver Gruppenhomomorphismus $\mu : \mathbb{F}_p^\times \longrightarrow \mathfrak{S}_p$ in die volle Permutationsgruppe. Ist a primitiv, so hat μ_a genau zwei Zykel: $\{0\}$ und $\mathbb{F}_p^\times$. Da p ungerade ist, hat μ_a das Vorzeichen $\sigma(\mu_a) = (-1)^{p-2} = -1$, ist also ungerade. Da a die Gruppe $\mathbb{F}_p^\times$ erzeugt, folgt die Gleichheit der Homomorphismen

$$(\frac{\bullet}{p}) = \sigma \circ \mu : \mathbb{F}_p^\times \longrightarrow \{\pm 1\},$$

und daraus die Behauptung. $\diamond$

Ein weiteres Hilfsmittel ist die **Diskriminante** eines Polynoms $f = a_n T^n + \cdots + a_0 \in K[T]$. Sie kann in jedem Erweiterungskörper $L \supseteq K$ gebildet werden, der alle Nullstellen $t_1, \ldots, t_n$ von f enthält, und ist dann

$$D(f) = a_n^{2n-2} \cdot \prod_{1 \le i < j \le n} (t_i - t_j)^2.$$

(Da sie unter allen Permutationen der Nullstellen invariant ist, liegt sie sogar in K, aber das folgt in dem hier interessanten Fall auch aus der expliziten Berechnung.) Die normale Methode zu ihrer Berechnung aus den Koeffizienten besteht im Vergleich mit der Resultanten von f und der Ableitung f'. Für das Kreisteilungspolynom $f = T^n - 1$ ist die Berechnung aber ganz einfach:

Hilfssatz 8 *Das Polynom $f = T^n - 1 \in K[T]$ (mit* $\operatorname{char} K \nmid n$*) hat die Diskriminante*

$$D(f) = (-1)^{\frac{n(n-1)}{2}} \cdot n^n.$$

Beweis. Sei ζ eine primitive n-te Einheitswurzel (in einem geeigneten Erweiterungskörper). Dann ist

$$\begin{aligned} f &= \prod_{i=0}^{n-1}(T-\zeta^i), \\ D(f) &= \prod_{0\le i<j\le n-1}(\zeta^i-\zeta^j)^2 = (-1)^{\frac{n(n-1)}{2}}\cdot\prod_{i\neq j}(\zeta^i-\zeta^j) \\ &= (-1)^{\frac{n(n-1)}{2}}\cdot\prod_{i=0}^{n-1}\left[\zeta^i\cdot\prod_{k=1}^{n-1}(1-\zeta^k)\right]. \end{aligned}$$

Für das Polynom

$$g = T^{n-1}+\cdots+1 = \prod_{k=1}^{n-1}(T-\zeta^k) \in K[T]$$

ist $g(1) = n$. Also folgt

$$D(f) = (-1)^{\frac{n(n-1)}{2}}\cdot\prod_{i=0}^{n-1}[\zeta^i\cdot n] = (-1)^{\frac{n(n-1)}{2}}\cdot n^n,$$

wie behauptet. $\diamond$

Hilfssatz 9 *Sei p eine ungerade Primzahl und n ungerade und zu p teilerfremd. Dann sind äquivalent:*

(i) *Die Diskriminante von $T^n-1 \in \mathbb{F}_p[T]$ ist Quadratrest* mod p.
(ii) $l = (-1)^{(n-1)/2}\cdot n$ *ist Quadratrest* mod p.

Beweis. Die Diskriminante ist $D(f) = l^n$ nach Hilfssatz 8. Ist $n = 2k+1$, so ist $D(f)$ Produkt des Quadratrests l^{2k} mit l. $\diamond$

Die Diskriminante eines Polynoms $f \in K[T]$ ist in dem Erweiterungskörper $L \supseteq K$, der die Nullstellen von f enthält, ein Quadrat:

$$D(f) = \Delta(f)^2 \quad \text{mit} \quad \Delta(f) = a_n^{n-1}\cdot\prod_{i<j}(t_i-t_j).$$

Aber $\Delta(f)$ ändert sich bei einer Permutation der Nullstellen mit dem Vorzeichen der Permutation und liegt daher im allgemeinen nicht in K.

Beweis des Hauptsatzes. Wegen Hilfssatz 6(i) reicht es, das quadratische Reziprozitätsgesetz für zwei verschiedene ungerade Primzahlen p und q zu beweisen. Sei $K = \mathbb{F}_p$, ζ eine primitive q-te Einheitswurzel, $L = K(\zeta)$ und

$f = T^q - 1$. Dann definiert $\zeta \mapsto \zeta^p$ eine Permutation der Einheitswurzeln und einen Automorphismus von L über K. Es folgt:

$$\sigma(\mu_p) \cdot \Delta(f) = \prod_{i<j} (\zeta^{pi} - \zeta^{pj}) = \Delta(f)^p.$$

Damit kann man eine Kette von Äquivalenzen aufstellen:

$$(-1)^{\frac{q-1}{2}} \cdot q \,\text{Quadratrest mod } p \iff D(f) \,\text{Quadratrest mod } p \iff \Delta(f) \in \mathbb{F}_p$$

$$\iff \Delta(f) = \Delta(f)^p \iff \sigma(\mu_p) = 1 \iff p \text{ Quadratrest mod } q.$$

Also ist mit Satz 5 (i)

$$(\frac{p}{q}) = (\frac{(-1)^{\frac{q-1}{2}} q}{p}) = (\frac{q}{p}) \cdot (\frac{-1}{p})^{\frac{q-1}{2}} = (\frac{q}{p}) \cdot (-1)^{\frac{p-1}{2}\frac{q-1}{2}},$$

wie behauptet. $\diamondsuit$

A.8 BLUM-Zahlen

Sei nun $n = pq$ mit verschiedenen Primzahlen $p, q \geq 3$. Dann ist

$$\mathbb{M}_n/\mathbb{M}_n^2 \cong \mathbb{M}_p/\mathbb{M}_p^2 \times \mathbb{M}_q/\mathbb{M}_q^2 \cong \mathcal{Z}_2 \times \mathcal{Z}_2,$$

insbesondere $\#(\mathbb{M}_n/\mathbb{M}_n^2) = 4$. Die Untergruppen $\mathbb{M}_n^2 \leq \mathbb{M}_n^+$ und $\mathbb{M}_n^+ \leq \mathbb{M}_n$ sind jeweils echt und daher vom Index 2. Im Ring $\mathbb{Z}/n\mathbb{Z}$ gibt es genau 4 Einheitswurzeln: $1, -1, \tau, -\tau$ mit

$$\tau \equiv -1 \pmod{p}, \quad \tau \equiv 1 \pmod{q},$$

also $(\frac{\tau}{n}) = -1$; anders ausgedrückt: Der Kern der Quadratabbildung $\mathbf{q} : \mathbb{M}_n \longrightarrow \mathbb{M}_n^2$ ist $K = \{\pm 1, \pm\tau\}$.

Eine Zahl der Form $n = pq$ mit verschiedenen Primzahlen $p, q \equiv 3 \pmod 4$ heißt BLUM-**Zahl** (z. B. die Zahl 1177 oben). Für eine solche ist also -1 kein Quadratrest in $\mathbb{M}_p$ und $\mathbb{M}_q$, also auch nicht in $\mathbb{M}_n$, aber

$$(\frac{-1}{n}) = (\frac{-1}{p})(\frac{-1}{q}) = (-1)^2 = 1,$$

also $-1 \in \mathbb{M}_n^+$. Es folgt

$$(\frac{-x}{n}) = (\frac{-1}{n})(\frac{x}{n}) = (\frac{x}{n})$$

für jedes x. Es ist $\mathbb{M}_n^2 \cap K = \{1\}$, also die Einschränkung von $\mathbf{q}$ auf $\mathbb{M}_n^2$ injektiv, also bijektiv, und $\mathbb{M}_n$ ist das direkte Produkt

$$\mathbb{M}_n = K \times \mathbb{M}_n^2, \quad \mathbb{M}_n^+ = \{\pm 1\} \times \mathbb{M}_n^2.$$

Jeder Quadratrest $a \in \mathbb{M}_n^2$ hat in jeder der vier Nebenklassen von $\mathbb{M}_n/\mathbb{M}_n^2$ genau eine Quadratwurzel; ist $x \in \mathbb{M}_n^2$ die eine, so sind $-x, \tau x, -\tau x$ die anderen. Damit ist gezeigt:

Satz 6 *Sei n eine* BLUM*-Zahl. Dann gilt:*

(i) *Ist $x^2 \equiv y^2 \pmod n$ für $x, y \in \mathbb{M}_n$ und sind $x, -x, y, -y \bmod n$ paarweise verschieden, so $(\frac{x}{n}) = -(\frac{y}{n})$.*

(ii) *Die Quadrat-Abbildung $\mathbf{q}$ ist eine Bijektion von $\mathbb{M}_n^2$ auf sich.*

(iii) *Jedes $a \in \mathbb{M}_n^2$ hat genau zwei Quadratwurzeln in $\mathbb{M}_n^+$; ist x die eine, so $-x \bmod n$ die andere, und genau eine von beiden ist selbst Quadratrest. Ferner hat a noch genau zwei weitere Quadratwurzeln, und diese liegen in $\mathbb{M}_n^-$.*

Von den vier Quadratwurzeln eines Quadratrests x ist also genau eine wieder ein Quadratrest. Diese wird als etwas besonderes betrachtet und mit $\sqrt{x} \bmod n$ bezeichnet. Das letzte Bit ("least significant bit") von x, das man auch als Parität von x oder als $x \bmod 2$ beschreiben kann, wird mit $\mathrm{lsb}(x)$ bezeichnet.

Korollar 1 *Sei $x \in \mathbb{M}_n^+$. Dann ist x genau dann Quadratrest, wenn*

$$\mathrm{lsb}(x) = \mathrm{lsb}(\sqrt{x^2} \bmod n).$$

Beweis. Ist x Quadratrest, so $x = \sqrt{x^2} \bmod n$. Ist x kein Quadratrest, so sei $y = \sqrt{x^2} \bmod n$. Nach (iii) muss $y = -x \bmod n = n - x$ sein. Da n ungerade ist, haben x und y verschiedene Parität. $\diamond$

Wie kann man die Quadratrest-Eigenschaft $\bmod n$ entscheiden? Wenn die Primzerlegung $n = pq$ bekannt ist, ist das effizient möglich:

$$x \in Q_n \Longleftrightarrow (\frac{x}{p}) = (\frac{x}{q}) = 1.$$

Es ist kein effizientes Verfahren bekannt, das ohne die Kenntnis der Primfaktoren auskommt; *möglicherweise* ist die Entscheidung der Quadratrest-Eigenschaft zur Primzerlegung komplexitätstheoretisch äquivalent. Allgemein für plausibel gehalten wird jedenfalls die **Quadratrest-Vermutung:** Die Entscheidung der Quadratrest-Eigenschaft für Blum-Zahlen ist hart.

A.9 Quadratische Nichtreste

Wie findet man einen quadratischen *Nichtrest* modulo einer Primzahl p? – Also eine Zahl a mit $p \not| a$, die kein Quadratrest $\bmod a$ ist. Bevorzugt als Lösung wird dabei (natürlich) eine möglichst kleine natürliche Zahl; trotzdem beginnen wir mit der -1:

Satz 7 *Sei $p \geq 3$ eine Primzahl.*

(i) -1 *ist quadratischer Nichtrest* $\bmod p \iff p \equiv 3 \pmod 4$.

(ii) 2 *ist quadratischer Nichtrest* $\bmod p \iff p \equiv 3$ *oder* $5 \pmod 8$.

(iii) (*Für $p \geq 5$*) 3 *ist quadratischer Nichtrest* $\bmod p \iff p \equiv 5$ *oder* $7 \pmod{12}$.

(iv) (*Für $p \geq 7$*) 5 *ist quadratischer Nichtrest* $\bmod p \iff p \equiv 2$ *oder* $3 \pmod 5$.

Beweis. (i) Das kann man aus Satz 5 folgern. Ein noch einfacherer Beweis geht so:

$$\begin{aligned} -1 \in \mathbb{M}_p^2 &\iff \bigvee_{i\in\mathbb{Z}} i^2 \equiv -1 \pmod p \iff \bigvee_{i\in\mathbb{Z}} \operatorname{Ord}_p i = 4 \\ &\iff 4 \mid \#\mathbb{F}_p^\times = p-1 \iff p \equiv 1 \pmod 4. \end{aligned}$$

(ii) Auch dies folgt aus Satz 5: Nach der dort anschließenden Bemerkung ist $2 \in \mathbb{M}_p^2 \iff p \equiv 1$ oder $7 \pmod 8$.

(iii) Hierzu wird das quadratische Reziprozitätsgesetz verwendet:

$$\begin{aligned} (\frac{3}{p}) = (-1)^{\frac{p-1}{2}}(\frac{p}{3}) &= \begin{cases} (-1)^{6k}(\frac{1}{3}) = 1, & \text{wenn } p = 12k+1, \\ (-1)^{6k+2}(\frac{2}{3}) = -1, & \text{wenn } p = 12k+5, \\ (-1)^{6k+3}(\frac{1}{3}) = -1, & \text{wenn } p = 12k+7, \\ (-1)^{6k+5}(\frac{2}{3}) = 1, & \text{wenn } p = 12k+11, \end{cases} \\ &= \begin{cases} 1, & \text{wenn } p \equiv 1 \text{ oder } 11 \pmod{12}, \\ -1, & \text{wenn } p \equiv 5 \text{ oder } 7 \pmod{12}. \end{cases} \end{aligned}$$

(iv) Hier ist nach dem quadratischen Reziprozitätsgesetz

$$(\frac{5}{p}) = (\frac{p}{5}) = \begin{cases} 1, & \text{wenn } p \equiv 1 \text{ oder } 4 \pmod 5, \\ -1, & \text{wenn } p \equiv 2 \text{ oder } 3 \pmod 5, \end{cases}$$

wie behauptet. $\diamond$

Korollar 1 241 *ist die einzige ungerade Primzahl* < 400*, für die weder* -1*,* 2*,* 3 *noch* 5 *quadratische Nichtreste sind.*

Korollar 2 *Nur für ungerade Primzahlen* $\equiv 1, 49 \pmod{120}$ *ist weder* -1*,* 2*,* 3 *noch* 5 *quadratischer Nichtrest.*

Die Aussage des Satzes lässt sich auf beliebige, nicht notwendig prime, Moduln übertragen. Dazu:

Hilfssatz 10 *Sei* $n \in \mathbb{N}$*,* $n \geq 2$*. Für* $a \in \mathbb{Z}$ *sei* $(\frac{a}{n}) = -1$*. Dann ist* a *kein Quadrat in* $\mathbb{Z}/n\mathbb{Z}$*.*

Beweis. Sei $n = p_1^{e_1} \cdots p_r^{e_r}$ die Primzerlegung. Dann ist

$$(\frac{a}{n}) = (\frac{a}{p_1})^{e_1} \cdots (\frac{a}{p_r})^{e_r}.$$

Also gibt es ein k, für das der Exponent e_k ungerade und $(\frac{a}{p_k}) = -1$ ist. Also ist a kein Quadrat $\bmod p_k$. Da $\mathbb{F}_{p_k}$ homomorphes Bild von $\mathbb{Z}/n\mathbb{Z}$ ist, ist a erst recht kein Quadrat $\bmod n$. $\diamond$

Korollar 3 *Sei* $n \in \mathbb{N}$*,* $n \geq 2$*, und* n *kein Quadrat. Dann gilt:*

(i) *Ist* $n \equiv 3 \pmod 4$*, so ist* -1 *kein Quadrat in* $\mathbb{Z}/n\mathbb{Z}$*.*

(ii) *Ist* $n \equiv 5 \pmod 8$*, so ist* 2 *kein Quadrat in* $\mathbb{Z}/n\mathbb{Z}$*.*

Usw. Man kommt auf diesem Weg aber nie zu einer vollständigen Erfassung aller Fälle, siehe die Anmerkung unten. Einen Algorithmus zur Bestimmung eines quadratischen Nichtrests braucht man allerdings höchstens, wenn $n \equiv 1 \pmod 8$. Hier gibt es wieder zwei Versionen:

- Einen deterministischen Algorithmus, der $a = 2, 3, 5, \ldots$ durchprobiert. Dieser ist unter der Annahme der erweiterten RIEMANNschen Vermutung – angewendet auf den Charakter $\chi = (\frac{\bullet}{n})$ – polynomial in der Stellenzahl $\log(n)$.

- Einen probabilistischen Algorithmus, der zufällig gewählte a probiert und jeweils mit Wahrscheinlichkeit $\frac{1}{2}$ reüssiert, d. h. $(\frac{a}{n}) = -1$ liefert. Zur Berechnung des JACOBI-Symbols sind $O(log(n)^2)$ Schritte nötig. Im Mittel sind zwei Versuche nötig, bis ein Nichtquadratrest gefunden ist.

Übungsaufgabe. Für welche Primzahlmoduln ist 7, 11 oder 13 quadratischer Nichtrest? Welches ist der kleinste Primzahlmodul, für den dann immer noch kein quadratischer Nichtrest gefunden ist?

Anmerkung. Es gibt keine untere Schranke, bis zu der man mit Sicherheit für alle Moduln n einen quadratischen Nichtrest findet. Ein Satz von Chowla/ Fridlender/ Salié besagt, dass es (mit einer Konstanten $c > 0$) unendlich viele Primzahlen gibt, so dass alle Zahlen a mit $1 \leq a \leq c \cdot \log(p)$ Quadratreste mod p sind. Ringrose/ Graham und – unter der erweiterten Riemannschen Vermutung – Montgomery bewiesen noch etwas schärfere Versionen.

Relevante Literatur:

- V. R. Fridlender: On the least n-th power non-residue. Dokl. Akad. Nauk. SSSR 66 (1949), 351–352.
- H. Salié: Über den kleinsten positiven quadratischen Nichtrest nach einer Primzahl. Math. Nachr. 3 (1949), 7–8.
- N. C. Ankeney: The least quadratic nonresidue. Ann. of Math. 55 (1952), 65–72.
- H. L. Montgomery: *Topics in Multiplicative Number Theory.* Springer LNM 227 (1971).
- J. Buchmann/V. Shoup: Constructing nonresidues in finite fields and the extended Riemann hypothesis. Preprint 1990.
- S. W. Graham/C. Ringrose: Lower bounds for least quadratic non-residues. In: B. C. Berndt et al. (Eds): *Analytic Number Theory*, Birkhäuser, Boston 1990, 270–309.
- D. J. Bernstein: Faster algorithms to find non-squares modulo worst-case integers. Preprint 2002.

A.10 Primitivwurzeln für spezielle Primzahlen

Ebenso wie quadratische Nichtreste sind auch Primitivwurzeln für viele Primzahlmoduln besonders leicht zu finden:

Satz 8 *Sei (q,p) ein* Germain-*Paar, d. h., q und $p = 2q+1$ seien Primzahlen. Dann gilt:*

(i) *$a \in [2 \dots p-2]$ ist $\bmod p$ genau dann Primitivwurzel, wenn es quadratischer Nichtrest ist.*

(ii) *$(-1)^{\frac{q-1}{2}} \cdot 2$ ist Primitivwurzel $\bmod p$.*

Beweis. Für $q = 2$, $p = 5$, ist das klar – 2 und 3 sind sowohl die Primitivwurzeln als auch die quadratischen Nichtreste. Also kann man $q \geq 3$, also $p \geq 7$, annehmen.

(i) Dann ist $p \equiv 3 \pmod 4$ und -1 quadratischer Nichtrest. Da die Gruppenordnung $\#\mathbb{F}_p^\times = p-1$ gerade ist, ist außerdem jede Primitivwurzel quadratischer Nichtrest. Da dieses $\varphi(p-1) = q-1$ Stück sind, haben wir damit bereits q quadratische Nichtreste gefunden. Da $q = \frac{p-1}{2}$, sind das alle.

(ii) Im Fall $q \equiv 1 \pmod 4$ ist $p \equiv 3 \pmod 8$, also $2 = (-1)^{\frac{q-1}{2}} \cdot 2$ quadratischer Nichtrest, also auch Primitivwurzel.

Im Fall $q \equiv 3 \pmod 4$ ist $p \equiv 7 \pmod 8$, also 2 quadratischer Rest und -1 quadratischer Nichtrest, also $-2 = (-1)^{\frac{q-1}{2}} \cdot 2$ quadratischer Nichtrest, also auch Primitivwurzel. $\diamond$