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.