1 Das RSA-Verfahren und seine algorithmischen Grundlagen

Das wichtigste – d. h., am weitesten verbreitete und am meisten analysierte – asymmetrische Verfahren 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 wichtigsten drei Algorithmen für die Implementierung des RSA-Verfahrens sind

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

letztere zwei schon im Kapitel I bei den linearen Chiffren behandelt, 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 RSA-Verfahrens

Parameter

Die drei Parameter

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

sind natürliche Zahlen 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, besser mit zufälligen Leitbits – 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 Die CARMICHAEL-Funktion

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

Die CARMICHAEL-Funktion ist definiert als Exponent der multiplikativen Gruppe $\mathbb{M}_n = (\mathbb{Z}/n\mathbb{Z})^\times$:

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

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 1 *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 $g^q = 1$, $h^q = 1$, $r|q$, $s|q$, $t|q$. $\diamond$

Aus dem chinesischen Restsatz folgt unmittelbar (da $\mathbb{M}_{mn} \cong \mathbb{M}_m \times \mathbb{M}_n$)

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 für Potenzen ungerader Primzahlen (Beweis als **Übungsaufgabe** – oder im Anhang A.3):

$$\lambda(p^e) = \varphi(p^e) = p^{e-1} \cdot (p-1) \quad \text{für } p \text{ prim} \geq 3.$$

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 2 *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 2 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 2 *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 $n | m$. $\diamond$

1.4 Geeignete RSA-Parameter

Satz 3 *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) **[RSA-Gleichung]** *Für jedes $d \in \mathbb{N}$ 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 mit $\mathrm{ggT}(d, \lambda(n)) = 1$ und e dazu passend mittels Kongruenzdivision $\mathrm{mod}\lambda(n)$, 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 aufwendig. 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.

Übungsaufgaben

1. Sei $n = p^2q$ mit zwei verschiedenen ungeraden Primzahlen p und q. Für welche $a \in \mathbb{Z}/n\mathbb{Z}$ gilt die RSA-Gleichung $a^{de} \equiv a \pmod{n}$? Verallgemeinerung auf beliebige n?

2. Zeige, dass $d \in \mathbb{N}$ genau dann zu $\lambda(n)$ teilerfremd ist, wenn es zu $\varphi(n)$ teilerfremd ist.