RSA Parameter

RSA Parameter
$$\begin{cases} \text{öffentlich:} & N = pq \text{ mit } p, q \text{ prim und } e \in \mathbb{Z}^*_{\phi(N)} \\ \text{geheim:} & d \in \mathbb{Z}^*_{\phi(N)} \text{ mit } ed = 1 \bmod \phi(N). \end{cases}$$

Satz RSA Parameter Generierung

RSA-Parameter (N, e, d) können in Zeit $\mathcal{O}(\log^4 N)$ generiert werden.

Algorithmus RSA Schlüsselgenerierung

EINGABE: 1^k, wobei k ein Sicherheitsparameter ist.

1. $p, q \leftarrow$ Wähle mittels Primzahltest zwei zufällige k-Bit Primzahlen.
2. $N \leftarrow pq$
3. $\phi(N) \leftarrow (p-1)(q-1)$
4. $e \leftarrow$ Wähle zufälliges $e' \in \mathbb{Z}_{\phi(N)}$ bis $\mathrm{ggT}(e', \phi(N)) = 1$.
5. $d \leftarrow e^{-1} \bmod \phi(N)$.

AUSGABE: (N, e, d)

Laufzeit RSA Parameter Generierung

Erwartete Laufzeit:

- Schritt 1: $\mathcal{O}(k^3 \cdot k)$, da eine k-Bit Zahl mit Ws $\approx \frac{1}{k}$ prim ist. (Folgt aus dem Gaußschen Primzahlsatz)
- D.h. wir müssen ca. k Zahlen wählen, bis eine Zahl prim ist.
- Schritt 2-5: $\mathcal{O}(k^2)$, d.h. gesamt $\mathcal{O}(k^4) = \mathcal{O}(\log^4 N)$.
- Man beachte: Laufzeit ist polynomiell im Sicherheitsparameter 1^k.

Definition RSA Ver- und Entschlüsselung

Seien (N, e, d) RSA Parameter.

1. Verschlüsselung: $E : \mathbb{Z}_N \to \mathbb{Z}_N$ mit $m \mapsto m^e \bmod N$
2. Entschlüsselung: $D : \mathbb{Z}_N \to \mathbb{Z}_N$ mit $c \mapsto c^d \bmod N$

Laufzeit:

- E und D können in Zeit $\mathcal{O}(\log^3 N)$ berechnet werden.

Korrektheit RSA Ver- und Entschlüsselung

Satz Korrektheit RSA

Seien (N, e, d) RSA-Parameter. Dann gilt $D(E(m)) = m$ für alle $m \in \mathbb{Z}_N$.

Beweis:

- Nach CRT-Isomorphismus gilt $D(E(m)) = m^{ed} = m \bmod N$ gdw
$$m^{ed} = m \bmod p \text{ und } m^{ed} = m \bmod q.$$
- Wir zeigen die Korrektheit modulo p. (Beweis analog modulo q)
- Wir schreiben $ed = 1 \bmod \phi(N)$ als $ed = 1 + k\phi(N)$ für ein $k \in \mathbb{N}$.
- D.h. $m^{ed} = m^{1+k\phi(N)} = m \cdot (m^{p-1})^{k(q-1)} = m \bmod p$ für $m \in \mathbb{Z}_p^*$.
- Für $m = 0 \notin \mathbb{Z}_p^*$ gilt ebenfalls $0^{ed} = 0 \bmod p$.

Faktorisieren und Berechnen von d

Definition Polynomialzeit-Äquivalenz

Wir nennen zwei Probleme *Polynomialzeit-äquivalent*, falls ein Polynomialzeit-Algorithmus für das eine Problem einen Polynomialzeit-Algorithmus für das andere Problem impliziert.

Satz Faktorisieren und RSA

Die folgenden drei Probleme sind Polynomialzeit-äquivalent.

1. Faktorisieren von N
2. Berechnen von $\phi(N)$
3. Berechnen von d aus (N, e)

Beweis:

- $1 \Rightarrow 2$: Berechne $\phi(N) = (p-1)(q-1)$.
- $2 \Rightarrow 3$: Berechne $d = e^{-1} \bmod \phi(N)$.
- $3 \Rightarrow 1$: Müssen zeigen, dass (N, e, d) das Tupel (p, q) liefert.

Ringschluss: Faktorisieren mit Hilfe von d

Beweisidee: $3 \Rightarrow 1$

- Sei $ed - 1 = k(p-1)(q-1) = 2^r t$ für $r \geq 2$ und t ungerade.
- Satz von Euler: Für beliebiges $a \in \mathbb{Z}_N^*$ gilt $a^{2^r t} = 1 \bmod N$.
- Wir ziehen nun sukzessive Quadratwurzel von $a^{2^r t} = 1 \bmod N$.
- **Fall 1:** Die Quadratwurzeln $a^{2^{r-1} t}, \ldots, a^t \bmod N$ sind alle 1.
  - In diesem Fall muss ein anderes $a \in \mathbb{Z}_N^*$ gewählt werden.
- **Fall 2:** $a^{2^\ell t} = (-1) \bmod N$ für ein $\ell \in \mathbb{Z}_r$.
  - In diesem Fall wird ebenfalls ein anderes $a \in \mathbb{Z}_N^*$ gewählt.
- **Fall 3:** $a^{2^\ell t} \neq (\pm 1) \bmod N$ und $a^{2^{\ell+1} t} = 1 \bmod N$ für ein $\ell \in \mathbb{Z}_r$.
  - D.h. es wurde eine nicht-triviale Quadratwurzel der Eins gefunden.
  - Damit liefert $\mathrm{ggT}(a^{2^\ell t} \pm 1, N)$ die Faktoren p, q.
- Man kann zeigen, dass Fall 3 für $a \in_R \mathbb{Z}_N^*$ mit Ws mind. $\frac{1}{2}$ eintritt.
- D.h. im Erwartungswert wählt man zwei $a \in_R \mathbb{Z}_N^*$.
- Die erwartete Laufzeit zum Faktorisieren ist damit $\mathcal{O}(\log^3 N)$.

Sicherheit von RSA

RSA Problem:

Gegeben: $N, e, c = m^e \bmod N$
Gesucht: $m \in \mathbb{Z}_N$

Anmerkungen:

- D.h. $m = c^{\frac{1}{e}}$ ist die e-te Wurzel von c in $\mathbb{Z}_N$.
- Das RSA-Problem ist nicht für alle m, e schwer.
- Z.B. kann für $m^e < N$, die Wurzel $c^{\frac{1}{e}}$ effizient berechnet werden.
- Das RSA-Problem ist nicht NP-schwer.
  (unter geeigneten Komplexitätsannahmen)
- RSA-Problem besitzt auf Quantenrechnern Laufzeitkomplexität
  $$\mathcal{O}(\log^2 N \cdot \log\log N \cdot \log\log\log N).$$
- **Offenes Problem:** Sind das RSA Problem und das Faktorisierungsproblem Polynomialzeit-äquivalent?

Polynome

Definition Polynom

Sei $R \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}\}$. Wir bezeichnen die Elemente der Menge

$$R[x] = \{\sum_{i=1}^{n} a_i x^i \mid n \geq (-1), a_i \in R \text{ und } a_n \neq 0\}$$

als *Polynome über R*. Sei $p(x) = \sum_{i=1}^{n} a_i x^i \in R[x]$.

1. $a_0, \ldots, a_n$ heißen Koeffizienten von $p(x)$.
2. a_n heißt *führender Koeffizient*. $p(x)$ heißt *monisch* falls $a_n = 1$.
3. $p(x)$ besitzt *Grad* $grad(p) = n$. Das Nullpolynom $p'(x) = \sum_{i=0}^{-1} a_i x_i$ besitzt Grad -1.
4. *Evaluation* von $p(x)$ ist die Abbildung $\text{eval} : R \to R$ mit $x_0 \mapsto p(x_0)$.

Abkürzende Notation: p statt $p(x)$.

Anmerkungen

- Wir nehmen vereinfachend an, dass die Operationen $(+, -, \cdot, :)$ auf Elementen aus R in Zeit $\mathcal{O}(1)$ ausgeführt werden können.

Komplexität von eval, Addition und Subtraktion

Evaluation:

- Naives Auswerten von $p(x)$ mit Grad n liefert $\sum_{i=0}^{n+1} = \mathcal{O}(n^2)$ Multiplikationen und n Additionen, d.h. Komplexität $\mathcal{O}(n^2)$.
- Das sogenannte **Horner-Schema** wertet $p(x)$ wie folgt aus

$$p(x) = (\ldots((a_n x + a_{n-1})x + a_{n-2})x + \ldots + a_1)x + a_0.$$

- Dies erfordert n Multiplikationen und n Additionen.
- eval besitzt damit lineare Komplexität $\mathcal{O}(n)$.

Addition/Subtraktion:

- Sei $a(x), b(x) \in R[x]$ mit $n = \mathrm{grad}(a) > \mathrm{grad}(b) = m$.
- Wir setzen $b_{m+1} = \ldots = b_n = 0$.
- Berechne $(a \pm b)(x) = \sum_{i=0}^{n} c_i x^i$ mit $c_i = a_i \pm b_i$ für $i \in [n+1]$.
- Damit besitzt die Addition/Subtraktion lineare Komplexität $\mathcal{O}(n)$.

Multiplikation mit Schulmethode

Multiplikation:

- Seien $a(x), b(x) \in R[x]$ mit Grad n bzw. m und $n \geq m$.
- Wir setzen $a_{n+1} = \ldots a_{n+m} = 0$ und $b_{m+1} = \ldots = b_{n+m} = 0$.
- Dann gilt

$$\begin{aligned}(ab)(x) &= (a_n b_m)x^{n+m} + (a_{n-1}b_m + a_n b_{m-1})x^{n+m-1} + \ldots \\ &\quad + (a_0 b_1 + a_1 b_0)x + a_0 b_0.\end{aligned}$$

- D.h. $(ab)(x) = \sum_{i=0}^{n+m} c_i x^i$ mit $c_i = \sum_{j=0}^{i} a_j b_{i-j}$.
- $(ab)(x)$ besitzt $n + m + 1 = \mathcal{O}(n)$ Koeffizienten.
- Pro Koeffizient benötigt man $\mathcal{O}(n)$ Additionen und Multiplikationen.
- Damit erhalten wir insgesamt eine Komplexität von $\mathcal{O}(n^2)$.