Fortgeschrittene Techniken der Kryptographie

Jürgen Fuß

Episode 2: Performance des RSA-Verfahrens

HAGENBERG | LINZ | STEYR | WELS

FH OBERÖSTERREICH UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Das RSA-Verfahren

Schlüsselerzeugung:

- $p, q \xleftarrow{R} \mathbb{P}$
- $n := pq,\ \varphi(n) = (p-1)(q-1)$
- $e \in \mathbb{Z}^*_{\varphi(n)}$ (beliebig)
- $d := e^{-1} \bmod \varphi(n)$

$\rightarrow$ Public Key (n, e)

$\rightarrow$ Private Key $(\cancel{p, q}, d)$

Verschlüsseln:

$$c := m^e \bmod n$$

$(m^e)^{1/e}$

Entschlüsseln:

$$m := c^d \bmod n$$

Square & Multiply

Um 53^{37} zu berechnen, werden zunächst Potenzen für Zweierpotenzen berechnet.

$$
\begin{aligned}
53^{2^0} &= 53^1 = 53, \\
53^{2^1} &= 53^2 = 2809, && \text{1 Mult.} \\
53^{2^2} &= 53^4 = (53^2)^2 = 2809^2 = 7890481, && \text{1 Mult.} \\
53^{2^3} &= 53^8 = (53^4)^2 = 7890481^2 = 62259690411361, && \text{1 Mult.} \\
53^{2^4} &= 53^{16} = (53^8)^2 = 62259690411361^2 = 3876269050118516845397872321, && \text{1 Mult.} \\
53^{2^5} &= 53^{32} = (53^{16})^2 = \\
&= 15025461748906708859452861070130993269553796873817927041
\end{aligned}
$$

Square & Multiply (II)

Mit diesen Ergebnissen lässt sich nun 53^{37} berechnen. Es ist $37 = 32 + 4 + 1 = 2^5 + 2^2 + 2^0$. Die Binärdarstellung von 37 ist $(100101)_2$. Somit ist

$$
\begin{aligned}
53^{37} =& 53^{32} \cdot 53^4 \cdot 53^1 = \\
=& 1502546174890670885945286107013099326955379687381792704 1 \\
& \cdot 7890481 \cdot 53 = 628358038363 \ldots 49026213.
\end{aligned}
$$

Für diese Berechnung waren fünf Multiplikationen in Schritt 1 (**Square**) und zwei Multiplikationen in Schritt 2 (**Multiply**) nötig, in Summe also sieben Multiplikationen.

Kleinere Zahlen (I)

Um 53^{37} mod 77 zu berechnen, werden zunächst Potenzen für Zweierpotenzen berechnet. Dabei wird sofort modulo 77 reduziert.

$$
\begin{aligned}
53^{2^0} &= 53^1 = 53, \\
53^{2^1} &= 53^2 = 37 \pmod{77}, \\
53^{2^2} &= 53^4 = (53^2)^2 = 37^2 = 60 \pmod{77}, \\
53^{2^3} &= 53^8 = (53^4)^2 = 60^2 = 58 \pmod{77}, \\
53^{2^4} &= 53^{16} = (53^8)^2 = 58^2 = 53 \pmod{77}, \\
53^{2^5} &= 53^{32} = (53^{16})^2 = 53^2 = 37 \pmod{77}
\end{aligned}
$$

Kleinere Zahlen (II)

Mit diesen Ergebnissen lässt sich nun $53^{37} \bmod 77$ berechnen. Es ist $37 = 32 + 4 + 1 = 2^5 + 2^2 + 2^0$. Die Binärdarstellung von 37 ist $(100101)_2$. Somit ist

$$53^{37} = 53^{32} \cdot 53^4 \cdot 53^1 = \underline{37} \cdot \underline{60} \cdot \underline{53} = 4 \pmod{77}.$$

Für diese Berechnung waren fünf Square- und zwei Multiply-Operationen nötig, in Summe also sieben Multiplikationen (modulo 77).

In Python lassen sich modulare Potenzen einfach mit der Funktion `pow()` berechnen.

```
> pow( 53, 37, 77 )
4
```

Beobachtungen (I)

Ist die Binärdarstellung des Exponenten n Bits lang und enthält w Einsen, so sind im ersten Schritt (Square) $n-1$ und in zweiten Schritt (Multiply) $w-1$ Multiplikationen erforderlich.

Merke! Verdoppelt sich die Bitlänge des Exponenten, so steigt der Aufwand für das Potenzieren auf das Doppelte.
Der Aufwand für das Potenzieren wächst **linear** mit der (proportional zur) Bitlänge des Exponenten.

Der Aufwand für eine einzelne Multiplikation hängt ebenfalls von der Größe der Zahlen ab. Für bis zu 64 Bit lange Zahlen kann eine Multiplikation als eine Instruktion aufgefasst werden. Für längere Zahlen kann die Multiplikation von 64-Bit-Zahlen als das „kleine Einmaleins" aufgefasst werden, dann muss jede (64-Bit-)Ziffer der ersten mit jeder der zweiten Zahl multipliziert werden.

Merke! Verdoppelt sich die Länge der zu multiplizierenden Zahlen, so steigt damit der Aufwand auf das Vierfache.
Der Aufwand für das Multiplizieren wächst **quadratisch** mit (proportional zum Quadrat) der Bitlänge der Faktoren.

$3 \cdot 7 \longrightarrow 35 \cdot 47$

140
245
1645

Effizienteres RSA – Verschlüsseln

Um das Verschlüsseln effizient zu gestalten, kann ein kleiner Exponent e gewählt werden, typischerweise ist dies $e = 65537$. Betrachtet man die Binärdarstellung

$$(10000000000000001)_2$$

dieser Zahl, so wird klar, warum – sowohl in Schritt 1 wie auch in Schritt 2 – Square and Multiply besonders effizient durchgeführt werden kann ($n = 17$, $w = 2$). Da es sich überdies bei 65537 um eine Primzahl handelt, ist mit großer Wahrscheinlichkeit auch $\mathrm{ggT}(\varphi(n), e) = 1$.

Merke! Verdoppelt sich die Schlüssellänge, steigt der Aufwand für das Verschlüsseln auf das Vierfache.
Der Aufwand für das Verschlüsseln wächst **quadratisch** mit (proportional zum Quadrat) der Schlüssellänge.

Für das Entschlüsseln können keine kleinen Exponenten d gewählt werden (mehr dazu in E.3). Der Exponent ist in der Regel so lang wie n.

$\approx 2^{16} \cdot d = 1$

$d = e^{-1} \mod \varphi(n)$

$e d = 1 \mod \underbrace{\varphi(n)}_{\approx n}$

Merke! Verdoppelt sich die Schlüssellänge, steigt der Aufwand für das Entschlüsseln auf das Achtfache.
Der Aufwand für das Entschlüsseln wächst **kubisch** mit (proportional zur dritten Potenz) der Schlüssellänge.

Schlüsselerzeugung

Der aufwendigste Teil der Schlüsselerzeugung ist, zwei große Primzahlen zu finden. Dazu werden zufällig gewählte Zahlen mit dem Miller-Rabin-Test auf Primalität getestet. Der Aufwand eines solchen Tests entspricht dem modularen Potenzieren, auch hier steigt der Aufwand – wie beim Entschlüsseln – bei einer Verdopplung der Schlüssellänge auf das Achtfache. Zusätzlich ergibt sich aus den Primzahlverteilungssätzen jedoch, dass doppelt so lange Primzahlen nur etwa halb so häufig sind, d. h. dass die Suche aus diesem Grund etwa doppelt so lange dauert.

Merke Der Aufwand für die Schlüsselerzeugung steigt um den Faktor 16, wenn die Schlüssellänge verdoppelt wird.
Der Aufwand für das Entschlüsseln wächst **proportional zur vierten Potenz** der Schlüssellänge.

RSA-CRS

RSA und der chinesische Restsatz

Der chinesische Restsatz erlaubt es, die Entschlüsselung um den Faktor 4 zu beschleunigen. Es geht darum,

erw Euklid (p, q)

$px + qy = 1$

$$m = c^d \bmod \underbrace{pq}_{n}$$

zu berechnen. Alternativ lassen sich

$pq = 15$

$$\begin{aligned} m_p &:= \boxed{c^d \bmod p} = (c^d \bmod pq) \bmod p = \boxed{m \bmod p} \text{ und} \\ m_q &:= \boxed{c^d \bmod q} = (c^d \bmod pq) \bmod q = \boxed{m \bmod q} \end{aligned}$$

CRS

$\Rightarrow m \bmod pq$

berechnen und da p und q relativ prim sind, ergibt sich m (modulo pq) mit dem chinesischen Restsatz aus m_p und m_q.

Sind p und q ungefähr gleich lang, so sind beide nur halb so lang wie pq, damit werden alle Multiplikationen ca. viermal so schnell. Da jetzt statt einmal zweimal potenziert werden muss, geht ein Teil des Geschwindigkeitsgewinns wieder verloren[1], es bleibt aber immer noch ein Faktor 2. Allerdings lassen sich nun auch die Potenzen deutlich schneller berechnen, denn statt

$$m_p = c^d \bmod p$$

darf man (Satz von Fermat) auch

$z^{p-1} = 1 \bmod p$

$$m_p = c^{\underline{\underline{d \bmod p-1}}} \bmod p$$

$c^{d \bmod p-1}$ q

(Analoges für m_q) rechnen. Das ursprüngliche d war so lang wie pq, $d_p = d \bmod p$ ist aber nur halb so lang. Damit lässt sich noch einmal ein Faktor 2 gewinnen.

[1] Die Berechnungen modulo p und modulo q lassen sich einfach parallel zueinander durchführen. So ist noch einmal ein Faktor 2 an Geschwindigkeit zu gewinnen.

Man erzeuge das RSA-Schlüsselpaar (n, e), (p, q, d) wie gehabt. Weiterhin berechne man

$$d_p := d \bmod p - 1 \text{ und}$$
$$d_q := d \bmod q - 1$$

$qy = 1 - px$

sowie mit dem erweiterten Euklidschen Algorithmus Zahlen x, y, so dass $\boxed{px + qy = 1}$ gilt. Bei der Entschlüsselung berechnet man aus m_p und m_q mit dem chinesischen Restsatz

$$m := m_p qy + m_q px \bmod pq.$$

$$\begin{cases} m_p = c^{d_p} \bmod p \\ m_q = c^{d_q} \bmod q \end{cases}$$

Auf den zweiten Blick erkennt man, dass es reicht, sich die Zahl y zu merken[2], denn es ist $px + qy = 1$ und daher kann man px durch $1 - qy$ ersetzen. Der Overhead durch den chinesischen Restsatz ist nur eine Multiplikation und zwei Additionen. Diese RSA-Variante ist insbesondere im Standard RSA-PKCS#1 beschrieben.[3]

[2] Die Zahl y wird in RFC 8017 als `qInv` bezeichnet. Es handelt sich dabei ja um den Kehrwert von q modulo p.

[3] RFC 8017 – https://datatracker.ietf.org/doc/html/rfc8017

RSA-CRS (á la RFC 8017)

$1\times$ **Modifizierte Schlüsselerzeugung:** Man erzeuge das RSA-Schlüsselpaar (n, e), (p, q, d) wie gehabt. Weiterhin berechne man $d_p := d \bmod p - 1$ und $d_q := d \bmod q - 1$ sowie mit dem erweiterten Euklidschen Algorithmus Zahlen x, y, so dass $px + qy = 1$ gilt. Der erweiterte Private Key ist (p, q, d_p, d_q, y).

$qy = 1 - px$

$1 - qy = px$

Modifizierte Entschlüsselung: Man berechne

$m \bmod pq$ $\xleftarrow{\text{crs}}$

$$\left\{\begin{aligned} m_p &:= c^{d_p} \bmod p, \\ m_q &:= c^{d_q} \bmod q, \end{aligned}\right.$$

$$h := (m_p - m_q)y \bmod p,$$

$$\Rightarrow\ m := m_q + qh \bmod pq.$$

$m \bmod p \xrightarrow{?} m_p$

$m_q + q \cdot h \bmod p$

$\Rightarrow m_q + q \cdot (m_p - m_q) \cdot y \bmod p$

$m_q + (m_p - m_q)(1 - \cancel{px}) \bmod p$

$\cancel{m_q} + m_p - \cancel{m_q} \bmod p$

$m \bmod p$ $\qquad$ $m_q + \cancel{q \cdot h} \bmod p$