RSA

Überblick

RSA ist ein Trick, der auf der Multiplikation von Ganzzahlen bezüglich eines Modulus und der Schwierigkeit der Primfaktorzerlegung beruht. Was sind die Primfaktoren von 76085064334921111 51?

- Plaintext m ist eine Ganzzahl $\in [1, n-1] \subset \mathbb{Z}$
- Cyphertext c ist eine Ganzzahl $\in [1, n-1] \subset \mathbb{Z}$
- n bezeichnet den Modulus. $n = p \cdot q$, $p, q \in \mathbb{P}$ (Menge der Primzahlen)
- $\phi(n) = (p-1) \cdot (q-1)$
- e – Public Exponent, d – Private Exponent. $e, d \in [1, \phi(n)]$
- Euler'sche ϕ-Funktion: $\phi(n) := |\{a \in \mathbb{N} | 1 \leq a \leq n \wedge ggT(a, n) = 1\}|$. Zählt
  - die Anzahl der der teilerfremden Zahlen:
    - $\phi(7) = |\{1, 2, 3, 4, 5, 6\}| = 6 (= 7 - 1) \rightarrow 7$ ist Primzahl
    - $\phi(6) = \phi(2 \cdot 3) = \phi(2) \cdot \phi(3) = (2-1) \cdot (3-1) = 2$ und tatsächlich $\phi(6) = |\{1, 5\}| = 2$
    - $p, q \in \mathbb{P} \rightarrow \phi(p \cdot q) = (p-1) \cdot (q-1)$. Gleichtes gilt für alle Primfaktoren jeder Ganzzahl
  - Und daher genauso die Anzahl der Elemente einer Gruppe $\mathbb{Z}_n^*, n \in \mathbb{N}$ (der eigentliche Grund für die Verwendung in RSA)

RSA Trapdoor Permutation

Der Grund, warum es funktioniert:

- e, d sind invers: $e \cdot d \mod \phi(n) = 1$
- Daher: $c^d \mod n = (m^e)^d \mod n = m^{e \cdot d} \mod n = m$

RSA Key Generation

1. Wähle zwei zufällige Primzahlen p, q:
   - Sie müssen groß sein (jeweils mindestens irgendwo im Bereich 2^{1024}) damit $p \cdot q \approx 2^{2048}$ und nicht zu nah bei einander liegen.
2. Generiere *public exponent* e:
   - $e =$ randprime($\phi(n)$) (zufällige Primzahl kleiner $\phi(n)$)
3. Generiere *private exponent* d:
   - Inverse von e bezüglich $mod \phi(n)$ mithilfe des erweiterten euklidischen Algorithmus:
     - mathematisch: $e \cdot d + \phi(n) \cdot t = ggT(e, \phi(n))$
     - Pseudocode: (ggT, d, t) $\leftarrow$ `xgcd(e,` ϕ(n)`)`

$\phi(n)$ muss offensichtlich geheim bleiben, da man sonst d aus e und $\phi(n)$ mit `xgcd()` berechnen kann. Ebenso müssen p, q geheim bleiben, da so schnell $\phi(n)$ berechnet werden kann. Modulus n muss hingegen öffentlich sein, damit verschlüsselt werden kann.

RSA Encryption

$c = m^e \mod n$

RSA Decryption

$m = c^d \mod n$

Angriffe gegen RSA *Verschlüsselung*

RSA ist deterministisch

Gleiche Daten werden immer gleich verschlüsselt. Es sollte Padding auf eine Standardlänge genutzt werden. Das Padding sollte ungefähr 64 Bits an Randomness haben.

RSA ist "weich"

Man kann Cyphertext-Nachrichten mit einander multiplizieren und erhält wieder eine valide Cyphertext-Nachricht:

$$c_1 \cdot c_2 \mod n = m_1^e \cdot m_2^e \mod n = (m_1 \cdot m_2)^e \mod n$$

Aus vorhandenen Cyphertexten können neue Cyphertexte generiert werden. Das hilft bei *known cyphertext* und *known plaintext* Attacken. Es kann auch nach einem inversen Element für den Cyphertext gesucht werden, der bei der Multiplikation mit dem Cyphertext *1* ergibt, wofür Plaintext und Cyphertext gleich sind.

Signieren mit RSA

→ Verschlüsseln einer Nachricht m mithilfe des *privaten* Schlüssels.

Es ist möglich, eine Signatur mithilfe seines privaten Schlüssels zu erzeugen, doch davon wird abgeraten: Es ist sehr langsam und kann nur Daten (Ganzzahlen) signieren, die kleiner sind, als der Modulus n.

Daher ist es eher üblich die Hashsumme der Daten zu bilden und diese Hashsumme zu signieren. So kann die Hashsumme auch bis zur Größe des Modulus (sha512sum und auffärts) herankommen, was sehr sicher ist.

Probleme mit RSA *Signaturen*

- Known Plaintext Attack wird einem quasi geschenkt, wenn die Signatur unter die Nachricht kommt.
- $0^d \mod n = 0, 1^d \mod n = 1, (n-1)^d \mod n = n-1$. Diese Signaturen kann jeder erzeugen.
- *Blinding Attack*: Sei M eine Nachricht, die nicht signiert werden würde, aber R^eM schon (MR^e geht auch). Signiert der Besitzer des *privaten* Schlüssels R^eM, kann auf folgende Art eine Signatur für M erzeugt werden:

$$(R^e M)^d \mod n = R^{e \cdot d} M^d \mod n = R M^d \mod n$$

Jetzt muss die Signatur S nur noch durch R geteilt werden:

$$S/R \mod n = \frac{R M^d}{R} \mod n = M^d \mod n$$