1 Kryptographie: Das RSA-Verfahren

1.1 Einführung

Das RSA-Verfahren ist ein klassischer Vertreter der *asymmetrischen* Verschlüsselungsverfahren. Bei diesen Verfahren werden zum Ver- und Entschlüsseln unterschiedliche Schlüssel verwendet. Der grundsätzliche Ablauf einer sicheren Kommunikation verläuft dabei nach folgendem Schema:

Der Sender einer Nachricht besorgt sich von einer zentralen Stelle den *public key* des Empfängers und verschlüsselt mit diesem seine Nachricht. Der Empfänger besitzt einen *private key*, den er geheim hält und der deshalb (hoffentlich) nur ihm bekannt ist. Mit Hilfe dieses private key können Nachrichten entschlüsselt werden, die mit dem zugehörigen public key verschlüsselt wurden. Der Vorteil solcher Verfahren besteht darin, daß vor der Kommunikation keine Schlüssel auf sicheren Kanälen ausgetauscht werden müssen, da auch ein potentieller Angreifer den public key kennen darf. Sie werden deshalb häufig dazu verwendet, geheime Schlüssel für symmetrische Verfahren auszutauschen, die dann für die nachfolgende Kommunikation verwendet werden.

Oftmals können asymmetrische Verfahren auch in der Gegenrichtung verwendet werden. Das bedeutet, daß der Sender einer Nachricht diese mit seinem private key verschlüsselt und der Empfänger sie mit dem public key entschlüsselt. Selbstverständlich dient dieses Vorgehen nicht der Geheimhaltung der Nachricht (da der public key allgemein zugänglich ist), sondern zur Realisierung einer *digitalen Signatur*. Wenn durch das Entschlüsseln mit dem public key ein sinnvoller Klartext zum Vorschein kommt, so kann der Empfänger mit hoher Wahrscheinlichkeit davon ausgehen, daß die Nachricht auch wirklich vom Sender mit dem zugehörigen private key stammt. Es ist äußert schwierig, ohne Kenntnis des private key eine Nachricht zu erzeugen, die nach Entschlüsselung mit dem public key irgendeinen Sinn ergibt.

1.2 Etwas Zahlentheorie

Die Gruppe $\mathbb{Z}_n^* := \{x \in \mathbb{Z}_n \setminus \{0\} \mid \mathrm{ggT}(x,n) = 1\}$ spielt beim RSA-Verfahren eine wichtige Rolle. Die Anzahl der Elemente in $\mathbb{Z}_n^*$ wird angegeben durch folgende Funktion:

Definition 1 *Die Eulersche Φ-Funktion ist definiert durch $\Phi(n) := |\mathbb{Z}_n^*|$.*

Wir benötigen die Werte von Φ nur für einen einfachen Spezialfall:

Lemma 2 *Für $n = pq$ mit p, q prim gilt $\Phi(n) = (p-1)(q-1)$.*

Ferner verwendet das RSA-Verfahren einen Algorithmus zur Berechnung der multiplikativen Inversen in $\mathbb{Z}_n^*$.

Satz 3 *Für $n > 0$ kann das multiplikative Inverse zu $z \in \mathbb{Z}_n^*$ in polynomieller Zeit berechnet werden.*

Beweis: Wir berechnen mit Hilfe des erweiterten Euklidischen Algorithmus x, y so, daß

$$1 = \mathrm{ggT}(n, z) = xn + yz$$

Offensichtlich folgt daraus $1 = yz \bmod n$ und y ist somit das multiplikative Inverse zu z. □

Das folgende Theorem ist eine Verallgemeinerung des Satzes von Fermat (da $\Phi(p) = p - 1$ falls p prim).

Satz 4 (Satz von Euler) *Für $n > 1$ und $x \in \mathbb{Z}_n^*$ gilt*

$$x^{\Phi(n)} = 1 \bmod n$$

1.3 Funktionsweise von RSA

1.3.1 Initialisierungsphase

Wenn Alice mit RSA verschlüsselte Nachrichten empfangen will, so muß sie zunächst zwei Primzahlen p und q wählen. Diese Primzahlen sollten beide eine recht große Bitlänge besitzen, da davon die Sicherheit der Kommunikation abhängt. Allerdings steigt mit der Größe der Zahlen auch der Aufwand zum Ver- und Entschlüsseln.

Alice berechnet nun $n = pq$, sowie $\Phi(n) = (p-1)(q-1)$. Ferner wählt sie zufällig ein Element $e \in \mathbb{Z}_{\Phi(n)}^*$ mit $e > 1$. Mit Hilfe des Euklidischen Algorithmus kann sie effizient das multiplikative Inverse $d := e^{-1} \bmod \Phi(n)$ zu e berechnen. Da $ed = 1 \bmod \Phi(n)$ können wir ansetzen, daß $ed = j\Phi(n) + 1$ für ein geeignetes j.

Damit ist die Vorbereitung abgeschlossen. Das Paar (n, e) wird von Alice veröffentlicht und bildet den public key, während d den private key darstellt. Die Primzahlen p und q werden für die eigentliche Kommunikation nicht mehr benötigt, müssen aber geheimgehalten werden, denn mittels p und q kann $\Phi(n)$ und damit d effizient berechnet werden. Wenn ein Angreifer nur n kennt, so nimmt man an, daß die Faktorisierung $n = pq$ nicht effizient berechnet werden kann. Es konnte leider bislang noch nicht bewiesen werden, daß es keinen effizienten Faktorisierungsalgorithmus gibt, aber es wird angenommen, daß dieses Problem nicht in polynomieller Zeit gelöst werden kann.

1.3.2 Kommunikationsphase

Bob schickt nun an Alice eine Nachricht m. Wir nehmen an, daß die Nachricht als Zahl $m < n$ gegeben ist. Dazu unterteilt man beispielsweise einen Text in hinreichend kleine Blöcke (z. B. zu k Bits, wenn 2^k die größte Zweierpotenz kleiner als n ist) und interpretiert die Binärdarstellung der Nachricht als Zahl).

Die Kommunikation verläuft wie folgt:

1. Bob berechnet die verschlüsselte Nachricht c durch $c := m^e \bmod n$ und sendet c an Alice.

2. Alice erhält den Klartext, indem sie $c^d \bmod n$ berechnet, denn

$$c^d = (m^e)^d = m^{ed} = m^{j\Phi(n)+1} = (m^{\Phi(n)})^j \cdot m = 1^j \cdot m = m \bmod n$$

wegen Satz 4.

1.3.3 Digitale Signatur

Das digitale Signieren einer Nachricht funktioniert analog, wobei d und e die Rollen tauschen:

1. Alice berechnet die signierte Nachricht s durch $s := m^d \bmod n$ und sendet s an Bob.

2. Bob erhält den Klartext, indem er $s^e \bmod n$ berechnet, denn

$$s^e = (m^d)^e = m \bmod n$$

1.4 Generierung zufälliger Primzahlen

Wie bereits in Abschnitt 1.3.1 beschrieben, werden für den RSA-Algorithmus zufällig generierte Primzahlen benötigt. Der Algorithmus hierzu ist sehr einfach zu realisieren, wenn man über einen Primzahltest verfügt. Man generiert hierzu eine Zufallszahl der gewünschten Größe und testet, ob sie prim ist. Wenn nein, so wiederholt man das Verfahren, bis man Erfolg hat.

Man kann zeigen, daß es zwischen 1 und n ungefähr $n/\log(n)$ Primzahlen gibt. Also hat man bei jedem Versuch eine Erfolgswahrscheinlichkeit von etwa $1/\log(n)$, wenn man jede Zahl zwischen 1 und n gleichwahrscheinlich wählt, und ist somit im Mittel nach $\log(n)$ fertig. In der Praxis wird man allerdings bei der Wahl von p und q die Länge der Zahlen in Bits festlegen und keine kleineren Zahlen generieren, da, wie oben erwähnt, die Sicherheit von RSA unmittelbar davon abhängt. Außerdem macht es natürlich keinen Sinn, gerade Zahlen zu generieren.