RSA

Grundlegend:

RSA bezeichnet ein von den Personen Rivest (R), Shamir (S) und Adlemann (A) im Jahr 1977 erfundenes, asymmetrisches Verschlüsselungsverfahren.

Die Asymmetrische Verschlüsselung wird unter anderem auch als Public-Key-Kryptographie bezeichnet. Im Unterschied zur symmetrischen Verschlüsselung kann der Schlüssel zur Verschlüsselung von Klartexten öffentlich gemacht werden, ohne die Sicherheit der Kommunikation zu gefährden. Zur Entschlüsselung ist ein entsprechender aus dem öffentlichen Schlüssel berechenbarer privater Schlüssel notwendig. Da umgekehrt dies nur mit sehr hohem Aufwand möglich ist, ist der private Schlüssel nur dem Schlüsselerzeuger bekannt.

Dadurch entstanden neue Möglichkeiten in der Kommunikation über unsichere Übertragungsmedien (z.B. Internet), da somit die Sicherheit durch das Public-Key-Verfahren gewährleistet werden kann. Heute werden asymmetrische Verschlüsselungsverfahren häufig verwendet, um Schlüssel für symmetrische Verschlüsselungsverfahren auszutauschen, da diese in der Regel effizienter sind.

Verschlüsselung und Signatur mit RSA:

Das RSA-Verfahren kann in Schritten erläutert werden, die zwei Kommunikationspartner durchführen müssen, um eine Information mit Verschlüsselungen austauschen zu können. Eine Nachricht *m* soll von Standort A nach Standort B geschickt werden. Um dies zu veranschaulichen, werden die Kommunikationspartner Alice und Bob verwendet.

1. Bob wählt 2 Primzahlen (*p & q*) so, dass ihr Produkt *möglichst groß ist*
2. Bob wählt eine Primzahl *e* zwischen 1 und $\phi(n) = (p - 1) \cdot (q - 1)$
   Dazu bestimmt er eine passende Zahl $d \in N$ zwischen 1 und φ(n), die dem multiplikativen Inversen modulo n entspricht => $(e \cdot d) \bmod \phi(n) = 1$
3. Bob sendet seinen öffentlichen Schlüssel (*n*, *e*) an Alice.
4. Alice berechnet damit den Geheimtext $c = E(m) = m^e \bmod n$ und schickt ihn an Bob
5. Bob kann die Nachricht mit seinem privaten Schlüssel *(n, d*) berechnen: $D(c) = c^d \bmod n = m$

Mit dem RSA-Verfahren kann auch der Urheber eines Dokumentes verifiziert werden. Diesen Prozess nennt man digitale Signatur.
Wenn wir davon ausgehen, dass Alice ein Dokument *m* signieren und an Bob weiterschicken will, dann geschieht dies folgendermaßen:

1. Alice berechnet mit ihrem privaten Schlüssel $s = m^d \bmod n$ und schickt *s* an Bob
2. Wenn Bob $m = s^e \bmod n$ berechnen kann, dann weiß er, dass Alice diese Signatur erstellt hat.

Beispiel: Verschlüsselung Zeichenfolge „Public-Key“
p = 251, q = 209 -> n = p * q = 67519 sowie φ(n) = 67000.
Weiters wählen wir e = 50253 und d = 27817

Die Zeichenfolge soll in ASCII-Blöcke unterteilt werden. Wir erhalten eine maximale Blockgröße von 2 Zeichen (2,29), da der Dezimalwert eines ASCII-Blocks kleiner sein muss als der Modulo n
Verschlüsselung:

ASCII	Dezimalwert	Geheimtext C
Pu	$80 * 128^1 + 117 * 128^0 = 10357$	$10357^{50253} \bmod 67519 = 17922$
bl	$98 * 128^1 + 108 * 128^0 = 12652$	$12652^{50253} \bmod 67519 = 15793$
ic	$105 * 128^1 + 143 * 128^0 = 13538$	$5835^{50253} \bmod 67519 = 56395$
-K	$45 * 128^1 + 75 * 128^0 = 5835$	$5835^{50253} \bmod 67519 = 10922$
ey	$102 * 128^1 + 121 * 128^0 = 13177$	$13177^{50253} \bmod 67519 = 54812$

Entschlüsselung:

C	Dezimalwert	ASCII
17922	$17922^{27917} \bmod 67519 = 10357$	$10357 \text{div} 128^1 = 80$ => **P**
		$10357 \bmod 128^1 = 117$ => **u**
		12652div128 = 98 => **b**
		12652 mod 128 = 108 => **l**
56395	…	…
10922	…	…
54812	…	…

Sicherheit:

Die Sicherheit des RSA-Verfahren beruht auf dem Problem der Faktorisierung von ganzen Zahlen. Momentan ist kein mit technischen Mitteln realisierbarer Algorithmus bekannt, der effizient die Primfaktoren von sehr großen ganzen Zahlen bestimmen kann. Daher ist es bei entsprechend großen Zahlen *n* zumindest nicht mit vertretbarem Aufwand möglich *φ(n) = φ(pq)* zu berechnen und damit den privaten Schlüssel *(d, n)* aus dem öffentlichen Schlüssel *(e, n)* zu bestimmen.
Es wurde jedoch schon erfolgreich eine 1039-bit Zahl faktorisiert, daher sollte min. eine 2048-bit Zahl verwendet werden. Weiters sollten sich die Primzahlen *p* und *q* in ihrer Länge unterscheiden.

Implementierung:

Um RSA implementieren zu können, ist es notwendig eine BigInteger-Bibliothek einzusetzen. Bei der Schlüsselerzeugung sollten Primzahlen per Zufallsgenerator und Primtest ermittelt werden. Die meisten Algorithmen zur Prüfung auf Primzahlen basieren auf dem Satz von Fermat.
Der Verschlüsselungsexponent e kann auch per Zufallsgenerator und Primtest gewählt werden.

Vorteile:

- Hohe Sicherheit
- Schlüsselzahl wächst nur linear zu Teilnehmerzahl (symmetrische Verschlüsselung wächst viel schneller)

Nachteile:

- Hohe Rechenzeit (ca. 1000x langsamer als symmetrische)
- Erhöhter Aufwand bei mehreren Empfängern, da die Verschlüsselung mit dem individuellen Public Key eines jeden Empfänger erfolgt, muss die Nachricht für jeden Empfängern einzeln verschlüsselt werden.

Low-Exponent-Angriff:

Falls der Exponent e des öffentlichen zu klein ist (z.B. e = 3), ist eine sogenannter Low-Exponent-Angriff möglich.

Wenn nun der gleiche Klartext m an drei verschiedene Empfänger gesendet wird, lauten deren öffentlichen Schlüssel folgendermaßen: $(n_0, 3)$, $(n_1, 3)$ und $(n_2, 3)$, und werden ebenfalls die Geheimtexte c_0, c_1 und c_2 abgefangen, lässt sich mithilfe des chinesischen Restsatzes der Klartext m berechnen.

Aus den obigen Angaben gilt nämlich, dass:

$m^3 = c_0 (\mathrm{mod}\ n_0)$

$m^3 = c_1 (\mathrm{mod}\ n_1)$

$m^3 = c_2 (\mathrm{mod}\ n_2)$

Als Lösung vom chinesischen Restsatz wird dann eine eindeutige Lösung $\mathrm{mod}(n_0 * n_1 * n_2)$ für m^3 geliefert. Wenn nun die dritte Wurzel gezogen wird, erhält man den Klartext m.

Dieses Verfahren funktioniert auch für größere Exponenten, allerdings müssen dafür dementsprechend mehr Geheimtexte, die den gleichen Klartext darstellen, abgefangen werden.

Public-Key-Only-Angriff:

Dieser Angriff basiert auf der Tatsache, dass sich ein „Zuhörer“ mithilfe des Public-Keys beliebig viele Klartexte verschlüsseln lassen kann. Bei RSA erhält man nämlich bei der Verschlüsselung des gleichen Textes immer den gleichen Geheimtext, vorausgesetzt dass der gleiche Schlüssel verwendet wird. Dies liegt daran, da RSA deterministisch ist.

Wenn dem „Zuhörer“ nun ein entsprechender verschlüsselter Text vorliegt, kann er sich mithilfe des Verschlüsselns von beliebig vielen Klartexten eine Datenbank an Wörtern und Sätzen mit der entsprechenden Verschlüsselung anlegen, wodurch die Verschlüsselung gebrochen werden kann. Dieses Verfahren ist in der Realität aber nicht anwendbar, da die

Datenbank an verschlüsseltem Text, die dazu benötigt werden, viel zu groß ausfallen würde, damit sich dieser Angriff lohnt.

Um sich gegen einen solchen Angriff zu schützen, kann ein sogenanntes Padding verwendet werden. Dabei werden im Klartext zufällige Bits verstreut, die beim Verschlüsseln dafür sorgen, dass schließlich kein verschlüsselter Text dem anderen gleicht. Die Position der Zufallsbits lässt sich dabei mithilfe einem Pseudozufallsgenerator bestimmen, sodass sich das Padding bei der Entschlüsselung wieder rückgängig machen lässt.

Quellen

https://archive.model.in.tum.de/um/courses/seminar/krypto/SS09/litzel/zusammenfassung.pdf

https://www.inf.hs-flensburg.de/lang/krypto/protokolle/rsa-low-exponent-attack.htm

http://rmg.zum.de/wiki/Benutzer:Deininger_Matthias/Facharbeit/Angriffe_auf_RSA#Public-Key-Only-Attacke