Sicherheit des geheimen Schlüssels

Michal Starosta
5. April 2006

1. Motivation

Wir leben in einem Zeitalter in dem die Kryptographie nicht mehr nur in der Mathematik eine Anwendung findet. Im nahezu jeden Lebensbereich des Alltags kommen wir, meistens unbewusst mit ihrer praktischer Anwendung in Berührung. Im Zeitalter des Internets ist es sogar leichtsinnig ohne der geeigneten Absicherung der Daten wichtige Inhalte zu übermitteln. Darum kann es wichtig und vielleicht auch interessant sein die Grundlagen der Chiffrierung kennen zu lernen, um es selbst festzustellen wie wichtig es ist manche Daten geheim zu halten und vor allem wie Kleinigkeiten es ermöglichen können sogar den schwersten Code zu knacken.

2. Sicherheit des geheimen Schlüssels

Es wird behauptet, dass das RSA - Verfahren ein Public – Key – System ist. Es soll also unmöglich sein, aus dem öffentlichen Schlüssel (n, e) den geheimen Schlüssel d zu berechnen.
Es wird gezeigt, dass man das Problem der Sicherheit des geheimen Schlüssels, nur auf die mathematische Ebene reduzieren kann. Nämlich, es wird vorgewiesen, dass die Bestimmung des geheimen Schlüssels genau so schwer ist, wie die Zerlegung des RSA – Moduls n in seine Primfaktoren. Das ist das so genannte Faktorisierungsproblem. Mit diesem Problem beschäftigen sich Mathematiker schon seit Jahrhunderten. Heutzutage glaubt man, dass es sehr schwierig ist, es wurde jedoch nicht verneint. Sollte sich das Faktorisierungsproblem als einfach herausstellen, wird es schnell in der Öffentlichkeit bekannt gegeben, da sich viele Wissenschaftler zur Zeit mit dem Faktorisierungsproblem beschäftigen. Niemand kann also von der Unsicherheit des RSA auf Dauer Vorteile gewinnen können.
Warum glaubt man, dass es genau so schwer ist, den geheimen RSA-Schlüssel zu finden, wie das RSA-Modul zu faktorisieren? Den geheimen Schlüssel kann man nur aus der Gleichung $de \equiv 1 \mod (p-1)(q-1)$ ausrechnen; q und p kann aber nur durch die Faktorisierung ermittelt werden.
Leider gibt es immer noch offene Fragen deren Antworten einen großen Einfluss auf die Sicherheit des RSA-Verfahrens ausüben können. Man weiß nämlich nicht, ob man den Klartext aus einem RSA-Schlüsseltext ohne den geheimen Schlüssel ermitteln kann, und zweitens ob das Faktorisierungsproblem wirklich so schwer ist.
Es ist daher sehr gefährlich, die Public-Key-Kryptographie-Anwendungen nur mit RSA zu realisieren. Und außerdem müssen die Forscher nach Alternativen zu RSA suchen um im Notfall RSA ersetzen zu können.

2.1. Das Faktorisierungsproblem („welche Zahlen sind groß genug?“)

Es gibt keine eindeutige Antwort. Man faktoriesiert immer größere Zahlen. Bis heute sind die größten RSA-576 (diese Nummer bedeutet die Anzahl von Bits die für die Darstellung dieser Zahl nötig sind und das ist eine Zahl die 174 Ziffer hat – alte Notation RSA-174) und RSA-640 (193 Stellen). RSA-576 ist erst im Jahr 2003 und RSA-640 im Jahr 2005 faktorisiert worden. In der nächsten Reihenfolge warten RSA-704, RSA-786, RSA-896, RSA-1024 (309 Stellen), RSA-1536, RSA-2048 (617 Stellen). Für die richtige Lösung kann man zwischen 20.000 und 200.000 Dollar bekommen (www.rsasecurity.com).
Das Hauptproblem bei der Faktorisierung ist die Erfindung eines schnelleren Algorithmus. Ohne den sind auch die schnellsten Rechner unbrauchbar.

2.2. Einfache Algorithmen

Beispiel: Systematisches Probieren
Man möchte eine Zahl x faktorisieren, die „nur“ 80 Ziffer hat. Man muss also testen ob eine von den Zahlen die kleiner als 10^{40} (ungefähr) der Teiler von x ist. Bei optimistischer Annahme, dass nur 1 von 10.000 Zahlen prim ist, gibt es 10^{36} Divisionen die man durchführen muss. Bei der Annahme, dass ein Rechner 10^{24} Operationen pro Sekunde durchführen kann, vergehen 10^{12} Sekunden, bis ein ganzes Intervall bearbeitet wird. Das ergibt 31.000 Jahre. Auch beim großen Glück, wo der Treffer nach 1% von allen Primzahlen vorkommt, überschreitet der dafür gebrauchte Zeitraum das menschliche Leben.

Die Faktorisierungsmethode von Fermat
Es sei $n = p \cdot q$ das Produkt zweier Primzahlen p und q. Ist der Abstand zwischen p und q sehr gering, so kann man n mit der Faktorisierungsmethode von Fermat faktorisieren.
In der Praxis ist ein sehr kleiner Abstand zwischen zufällig ausgewählten 512 – Bit-Primzahlen p und g extrem unwahrscheinlich und daher kann man das RSA-Modul n im Allgemeinen nicht auf diese Art und Weise faktorisieren.
Trotzdem dient diese einfache Methode als die Grundlage für bessere Faktorisierungsalgorithmen und ist daher sehr lehrreich.

Idee:

Man nimmt $q > p$ an. Die 3. Binomische Formel mit $x = \frac{q+p}{2}$ und $y = \frac{q-p}{2}$ liefert dann:

$$x^2 - y^2 = (x - y)(x + y) = p \cdot q = n$$
$$\Rightarrow x^2 - n = y^2$$

Ist q-p sehr klein, so ist auch $y = \frac{q-p}{2}$ sehr klein. Damit kann folgendes Vorgehen erfolgreich sein:

1. Berechne $\sqrt{n}$.
2. Wähle sukzessiv $x \in Z$ mit $x > \sqrt{n}$ und teste, ob x^2-n eine Quadratzahl y^2 ist.
3. Ist y^2 gefunden, so ist $n = x^2 - y^2 = (x - y)(x + y)$ eine Faktorisierung von n.

2.3. Effiziente Algorithmen

Heutzutage benutzt man drei effiziente Algorithmen:

- *Elliptic Curve Method* – für die Faktorisierung der Zahlen, die bis 43 Ziffern enthalten
- *Quadratic Sieve* – für die Zahlen die bis 110 Ziffer haben (schnellere Varianten sind *Multiple Polynomial Quadratic Sieve* und *Double Large Prime Variation of the Multiple Polynomial Quadratic Sieve*).

  Laufzeit: $\exp(\frac{1}{2}(\ln n \cdot \ln\ln n)^{\frac{1}{2}})$
- *Number Field Sieve* – für die Zahlen die mehr als 110 Ziffer haben. Bis heute die schnellste Methode. Sie wurde benutzt für die Faktorisierung von RSA-140, RSA-155 und der neunten Fermatzahl (155 Stellen).
  Die neuste Variation *Greater Number Field Sieve* wurde benutzt für die Faktorisierung von RSA-160 und RSA-576.

  Laufzeit: $\exp((C+o(1))(n)^{\frac{1}{3}}(\log n)^{\frac{2}{3}})$ wobei:

  für die allgemeine Variation ist $C = (\frac{64}{9})^{\frac{1}{3}} = 1{,}922999\ldots$

  für die spezielle Variationen ist $C = (\frac{32}{9})^{\frac{1}{3}} = 1{,}526285\ldots$

Schon heute gibt es einen Algorithmus, der das Faktorisierungsproblem löst - den *Shor-Algorithmus* (Laufzeit: $O(\log n)$). Er ist für die Quantencomputer gedacht. Leider gibt es zur Zeit noch keinen geeigneten Rechner.

3. Die Auswahl von *p*, *q*, *e* und *d*

Die Auswahl von den Elementen des Schlüssels ist sehr wichtig, da die Sicherheit nicht gefährdet sein soll.

3.1. Die Auswahl von *p* und *q*

Es gibt konkrete Voraussetzungen, welche bei der Auswahl von *p* und *q* erfüllt werden müssen:

- beide Elemente sollen zufällig und möglichst gleichverteilt gewählt werden
- *p* und *q* sollen möglichst gleich groß sein

Heutzutage sind die beiden Faktoren 512-Bit-Zahlen. Für die längerfristige Verwendung und Sicherheit des RSA - Algorithmus sollen die Faktoren 1024 oder 2048-Bits lang sein. Das ist aber nur eine Vermutung, da die Entwicklung von Algorithmen und der Hardware schwer vorauszusehen sind.

3.2. Die Auswahl von *e*

Bei der Wahl von *e*, muss auch die Effizienz der Verschlüsselung im Auge behalten werden. Und das wichtigste dabei - die Sicherheit kann nicht gefährdet werden. Es werden normalerweise für *e* die Zahlen 3 , 7, 17 oder 65537 gewählt. Warum nicht 2? Weil $\varphi(n) = (p - 1)(q - 1)$ gerade ist und $ggT(e, (p - 1)(q - 1)) = 1$ gelten muss.

Aber wenn man nur über die Effizienz denkt, existiert die Gefahr, dass ein Angreifer die so genannte *Low – Exponent – Attacke* anwenden wird (siehe 4.3).

3.3. Auswahl von *d*

Um die Entschlüsselung zu beschleunigen kann man einen kleinen Faktor *d* wählen, *d* darf jedoch nicht zu klein sein – D. Boneh und G. Durfee haben bewiesen, dass das RSA – Verfahren gebrochen werden kann, wenn $d < n^{0.292}$ ist.

4. Angriffmöglichkeiten

4.1. Ermittlung des geheimen Schlüssels

Wenn der Angreifer nur den öffentlichen Schlüssel kennt, gibt es bis heute keine Methode um den geheimen Schlüssel in einer kurzen Zeit zu berechnen. Aber was passiert wenn der Angreifer noch andere Elemente vom geheimen Schlüssel *(p, q, t, d)* kennen sollte?

- *(n, e, p)* oder *(n, e, q)* – Annahme: *p* ist bekannt (analog für *q*). *q* kann man von *q=n/p* ausrechnen. Wenn man schon *p* und *q* hat ist der Rest einfach, es bleibt nur *p-1, q-1* zu berechnen und dann *d* aus $ed \equiv 1 \bmod (p-1)(q-1)$
- *(n, e, t)* – hier neben $t=(p-1)(q-1)/ggT(p-1,q-1)$ ist auch wichtig, dass $(p-1)(q-1)$ nah von *n* liegt. Man kann also $ggT(p-1, q-1)$ finden - eine ganze Zahl die nah von *n/t* liegt. Der Wert von $ggT(p-1,q-1)$ ist normalerweise nicht zu groß, da die Wahrscheinlichkeit, dass zwei zufällige Zahlen einen großen gemeinsamen Teiler haben werden, sehr gering ist. So kann man $(p-1)(q-1)$ ausrechnen. Dan muss man nur die Formel $ed \equiv 1 \bmod (p-1)(q-1)$ anwenden.
  Oft außer *t* wird $\varphi(n)$, so genannte eulersche Funktion, benutzt. $\varphi(n)$ ist Vielfache von *t*. Wenn der Angreifer $\varphi(n)$ kennt, kann er *d* aus $ed \equiv 1 \bmod \varphi(n)$ ausrechnen.
- *(n, e, d)* – der Angreifer kann ohne Probleme gesendete Nachricht entschlüsseln.

4.2. Multiplikativität

Sei *(n, e)* ein öffentlicher RSA-Schlüssel. Wenn man damit zwei Nachrichten m_1 und m_2 verschlüsselt, erhält man

$$c_1 = m_1^e \bmod n \text{ und } c_2 = m_2^e \bmod n$$

Es gilt dann

$$c = c_1 c_2 \bmod n = (m_1 m_2)^e \bmod n$$

Im Fall, dass jemand c_1 und c_2 kennt, eröffnet sich die Möglichkeit die Verschlüsselung $m=m_1m_2$ zu berechnen, ohne den Klartext $m=m_1m_2$ zu kennen. Ein Angreifer kann schon so was ausnutzen.
Wie kann man sich von solchen Betrugmöglichkeiten schützen? Man kann den Klartextraum auf Klartexte mit bestimmter Struktur so einschränken, dass das Produkt von zwei Klartexten kein gültiger Klartext ist. Eine Möglichkeit besteht darin einen Klartext immer so vorzubereiten, dass der erste und der letzte Byte in einem gültigen Klartext übereinstimmen. Für so konstruierte m_1 und m_2 wird das Produkt m_1m_2 diese Eigenschaft nicht besitzen.
Wenn man also solch eine Nachricht bekommt und feststellt, dass sie eine falsche Struktur hat, ist es gewiss, dass es sich hier um eine falsche Nachricht handelt.

4.3. Low – Exponent – Attacke

Bevor *LEA* erklärt wird, kommt noch kurz der chinesische Restsatz.

Satz - Chinesischen Restsatz:
Seien $m_1,\ldots,m_n$ paarweise teilerfremde ganze Zahlen, dann existiert für jedes Tupel ganzer Zahlen $a_1,\ldots,a_n$ eine ganze Zahl x, die die folgende simultane Kongruenz erfüllt:

$$x \equiv a_i \bmod m_i \text{ für } i = 1,\ldots,n$$

Alle Lösungen dieser Kongruenz sind kongruent modulo

$$M := m_1 m_2 \ldots m_n .$$

Das Produkt M stimmt hier wegen Teilerfremdheit mit dem *kgV* überein.

Eine Lösung kann man wie folgt ermitteln. Für jedes i sind die Zahlen $M_i := M/m_i$ teilerfremd, also kann man z.B. mit dem erweiterten euklidischen Algorithmus zwei Zahlen r_i und s_i finden, so dass

$$r_i \cdot m_i + s_i \cdot M_i + 1 .$$

Setzen wir $e_i := s_i \cdot M_i$, dann gilt

$$e_i \equiv 1 \bmod m_i$$
$$e_i \equiv 0 \bmod m_j , j \neq i$$

Die Zahl $x := \sum_{i=1}^{n} a_i e_i$ ist dann eine Lösung der simultanen Kongruenz.

Beispiel
Gesucht sei eine Zahl x mit der Eigenschaft

$$x \equiv 2 (\bmod 3)$$
$$x \equiv 3 (\bmod 4)$$
$$x \equiv 2 (\bmod 5)$$

Hier ist $M = 3 \cdot 4 \cdot 5 = 60$, $M_1 = M/3 = 20$, $M_2 = M/4 = 15$, $M_3 = M/5 = 12$
Mit Hilfe des erweiterten Euklidischen Algorithmus berechnet man

$$(-13) \cdot 3 + 2 \cdot 20 = 1, \text{ also } e_1 = 40$$
$$(-11) \cdot 4 + 3 \cdot 15 = 1, \text{ also } e_2 = 45$$
$$5 \cdot 5 + (-2) \cdot 12 = 1, \text{ also } e_3 = -24$$

Eine Lösung ist dann $x = 2 \cdot 40 + 3 \cdot 45 + 2 \cdot (-24) = 167$. Wegen $167 \equiv 47 \bmod 60$ sind alle anderen Lösungen kongruent zu 47 modulo 60.

Low – Exponnent – Attacke basiert auf dem folgenden Theorem.

Theorem 1:
Seien $e \in N$, $n_1, n_2, \ldots, n_e \in N$ paarweise teilerfremd und $m \in N$ mit $0 \leq m \leq n_i$, $1 \leq i \leq e$. Sei $c \in N$ mit $c \equiv m^e \bmod n_i$, $1 \leq i \leq e$ und $0 \leq c < \prod_{i=1}^{e} n_i$. Dann folgt $c = m^e$.

Beweis:
Die Zahl $c' = m^e$ erfüllt die simultane Kongruenz $c' \equiv m^e \ mod \ n_i$, $1 \le i \le e$ und es gilt $0 \le c < \prod_{i=1}^{e} n_i$, weil $0 \le m \le n_i$, $1 \le i \le e$, vorausgesetzt ist. Da eine solche Lösung der simultanen Kongruenz aber nach dem chinesischen Restsatz eindeutig bestimmt ist, folgt $c = c'$.

Wie funktioniert das *LEA*?
LEA wird angewandt wenn dieselbe Nachricht an mehrere Personen verschickt und dafür derselben e benutzt wird. Trotz verschiedenen öffentlichen Schlüsseln n_i kann man die Nachricht m dekodieren.
Man kennt die Schlüsseltexte $c_i = m^e \ mod \ n_i$, $i \le 1 \le e$. Mit dem chinesischen Restsatz wird eine ganze Zahl c mit $c \equiv c_i \ mod \ n_i$, $i \le 1 \le e$ und $0 \le c < \prod_{i=1}^{e} n_i$ berechnet. Nach dem Theorem gilt $c = m^e$. Man muss also e–te Wurzel ausrechnen, was in der Polynomzeit möglich ist.

Beispiel:
Sei e = 3, n_1 = 143, n_2 = 391, n_3 = 899l, m = 135. Dann ist c_1 = 60, c_2 = 203, c_3 = 711. Um den chinesischen Restsatz zu verwenden berechne x_1, x_2, x_3 mit $x_1 n_2 n_3 \equiv 1$ mod n_1, $n_1 x_2 n_3 \equiv 1$ mod n_2 und $n_1 n_2 x_3 \equiv 1$ mod n_3. Es ergibt sich x_1 = -19, x_2 = -62, x_3 = 262. Dann ist $c = (c_1 x_1 n_2 n_3 + c_2 n_1 x_2 n_3 + c_3 n_1 n_2 x_3)$ mod $n_1 n_2 n_3$ = 2460375 und $m = 2460375^{1/3} = 135$.

Wie kann man *LEA* verhindern?
Es gibt zwei Möglichkeiten:
- die Klartextblöcke können kürzer gewählt werden, als es nötig ist, und die letzten Bits sollen zufällig gewählt werden
- Verschlüsselungsexponent e kann größer gewählt werden. Mann soll aber nicht die Effizienz vergessen. Üblich ist $e = 2^{16} + 1$.

5. Erweiterung des 2. Kapitels

(Kapitel 9.3.4 von Buchman J. - „Einführung in die Kryptographie")

In diesem Kapitel wird folgende Äquivalenz bewiesen.
Wenn der Angreifer die Primfaktoren p und q des RSA-Moduls n kennt, kann er aus n und dem Verschlüsselungsexponent e den geheimen Schlüssel d durch Lösen der Kongruenz $de \equiv 1 \ mod \ (p-1)(q-1)$ berechnen.
Es wird gezeigt, dass die Umkehrung auch gilt, wie man nämlich aus n, e, d die Faktoren p und q berechnet.
Dazu setzt man

$$s = max\{t \in N : 2^t \text{ teilt } ed - 1\} \text{ und } k = (ed - 1)/2^s.$$

Lemma 1:
Für alle zu n teilerfremde ganzen Zahlen a gilt $order(a^k + nZ) \in \{2^i : 0 \le i \le s\}$.

Beweis:
Sei a eine zu n teilerfremde ganze Zahl. Nach Theorem 2.1 gilt $a^{ed-1} \equiv 1 \ mod \ n$. Daraus folgt $(a^k)^r \equiv 1 \ mod \ n$ (wobei $r = 2^s$). Also impliziert Theorem 2.2 , dass die Ordnung von $a^k + nZ$ ein Teiler von 2^s ist.

Der Algorithmus, der n faktorisiert, beruht auf folgenden Theorem.

Theorem 2:
Sei a eine zu n teilerfremde ganze Zahl. Wenn die Ordnung von $a^k \bmod p$ und $\bmod q$ verschieden ist, so ist $1<ggT(a^{rk} - 1,n)<n$ (wobei $r = 2^t$ ist) für ein $t\in \{0,1,2,\ldots,s-1\}$.

Beweis:
Nach Lemma 1. Liegt die Ordnung von $a^k \bmod p$ und $\bmod q$ in der Menge $\{2^i : 0\leq i \leq s\}$. Sei die Ordnung von $a^k \bmod p$ größer als die von $a^k \bmod q$. Die Ordnung von $a^k \bmod g$ sei 2^t. Dann gilt $t<s$, $a^{rk}\equiv 1 \bmod q$, aber $a^{rk}\equiv 1 \bmod q$ gilt nicht (wobei $r = 2^t$ ist) und daher $ggT(a^{rk} - 1,n) = q$.

Um n zu faktorisieren, wählt man zufällig und gleichverteilt eine Zahl a in der Menge $\{1,\ldots,n-1\}$. Dann berechnet man $g = ggT(a,n)$. Ist $g>1$, so ist g echter Teiler von n. Der wurde ja gesucht. Also ist der Algorithmus fertig. Ist $g=1$, so berechnet man $g=ggT(a^{rk} - 1,n)$ (wobei $r = 2^t$ ist), $t=s-1, s-2,\ldots,0$.

Findet man dabei einen Teiler von, dann ist der Algorithmus fertig. Anderfalls wird ein neues a gewählt und dieselben Operationen werden für dieses a ausgeführt. Es wird jetzt gezeigt, dass in jeder Iteration dieses Verfahrens die Wahrscheinlichkeit dafür, dass ein Primteiler von n gefunden wird, wenigstens ½ ist. Die Wahrscheinlichkeit dafür, dass das Verfahren nach m Iterationen einen Faktor gefunden hat, ist dann größer als $1- 1/2^m$.

Theorem 3:
Die Anzahl der zu n primen Zahlen a in der Menge $\{1,\ldots,n-1\}$, für die $a^k \bmod p$ und $\bmod q$ eine verschiedene Ordnung hat, ist wenigstens $(p-1)(p-q)/2$.

Beweis:
Sei g eine Primitivwurzel $\bmod p$ und $\bmod q$. Eine solche existiert nach dem chinesischen Restsatz.
Zuerst Annahme – die Ordnung von $g^k \bmod p$ sei größer als die Ordnung von $g^k \bmod q$. Diese beide Ordnungen sind nach Lemma1. Potenzen von 2. Sei x eine ungerade Zahl in $\{1,\ldots,p-1\}$ und sei $y\in \{1,\ldots,q-2\}$. Sei a eine Lösung der simultanen Kongruenz

$$a\equiv g^x \bmod p \text{ und } a\equiv g^y \bmod q \qquad (1°).$$

Dann ist die Ordnung von $a^k \bmod p$ dieselbe wie die Ordnung von $g^k \bmod p$. Die Ordnung von $a^k \bmod p$ ist aber höchstens so groß wie die Ordnung von $g^k \bmod q$, also kleiner als die Ordnung von $a^k \bmod p$. Schließlich sind diese Lösungen mod n paarweise verschieden, weil g eine Primitivwurzel $\bmod p$ und $\bmod q$ ist. Damit sind $(p-1)(q-1)/2$ zu n teilerfremde Zahlen a gefunden, für die die ordnung von $a^k \bmod p$ und q verschieden ist.
Ist die Ordnung von $g^k \bmod q$ größer als die $\bmod p$, so geht man genauso vor.
Sei schließlich angenommen, dass die Ordnung von $g^k \bmod p$ und $\bmod q$ gleich ist. Da $p-1$ und $q-1$ beide gerade sind, ist diese Ordnung wenigstens 2. Man bestimmt die gesuchten Zahlen a wieder als Lösung der simultanen Kongruenz (1°). Die Exponentenpaare (x,y) müssen diesmal aber aus einer geraden und einer ungeraden Zahl bestehen. So gibt $(p-1)(q-1)/2 \bmod n$ verschiedene Lösungen und hat gewünschte Eigenschften.

Aus Theorem 3 folgt unmittelbar, dass die Erfolgswahrscheinlichkeit des Faktorisierungsverfahren in jeder Iteration wenigstens 1/2 ist.

Hilfsdefinitionen:

Theorem 2.1
Sei *(n,e)* ein öffentlichen und *d* der entsprechende private Schlüssel im RSA-Verfahren. Dann gilt $(m^e)^d \bmod n = m$ für jede natürliche Zahl *m* mit $0 \leq m < n$.

Theorem 2.2
Sei $g \in G$ und $e \in Z$. Dann gilt $g^e = 1$ genau dann, wenn *e* durch die Ordnung von *g* in G teilbar ist.

6. Literatur

1. Buchmann, Jochanes: Einführung in der Kryptographie; Kapitel 7; Springer; 1999
2. Karbowski, Marcin: Grundlagen der Kryptographie; Kapitel 1 und 2; Gliwice; 2006
3. Ferguson, Niels; Schneier, Bruce; Practical Cryptography; 2003
4. www.rsasecurity.com