Eulersche φ-Funktion, RSA

Manfred Gruber

http://www.lrz-muenchen.de/~gruber

SS 2010, KW 25

Die Eulersche φ-Funktion

Sei $\varphi(m)$ die Anzahl der Elemente in $\mathbf{Z}_m^*$ ($m \in \mathbf{N}$) ("Eulersche φ-Funktion"). Sei $m = p_1^{k_1} \cdots p_r^{k_r}$ mit paarweise disjunkten Primzahlen p_i. Nach dem Chinesischen Restsatz ist ein $x \in \mathbf{Z}_m$ genau dann invertierbar, wenn alle $x \bmod p_i^{k_i}$ in $\mathbf{Z}_{p_i^{k_i}}$ invertierbar sind, also ist

$$\varphi(m) = \varphi(p_1^{k_1}) \cdots \varphi(p_r^{k_r})\,.$$

In jedem $\mathbf{Z}_{p_i^{k_i}}$ sind alle Elemente zu $p_i^{k_i}$ teilerfremd ausser Vielfache von p_i; von diesen gibt es $p_i^{k_i-1}$ Stück, nämlich $0 \cdot p_i, 1 \cdot p_i, 2 \cdot p_i, \ldots, (p_i^{k_i-1} - 1) \cdot p_i$. Also ist

$$\varphi(p_i^{k_i}) = p_i^{k_i} - p_i^{k_i-1} = p_i^{k_i}(1 - 1/p_i)$$

und insgesamt

$$\varphi(m) = \prod_i p_i^{k_i}(1 - 1/p_i) = m \prod_i (1 - 1/p_i)\,.$$

Beispiel 1. $9699690 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19.$

$$\begin{aligned}
\varphi(9699690) &= 9699690 \cdot (1 - 1/2) \cdots (1 - 1/19) \\
&= 9699690 \cdot 1/2 \cdots 18/19 \\
&= 1 \cdot 2 \cdot 4 \cdot 6 \cdot 10 \cdot 12 \cdot 16 \cdot 18 \\
&= 1658880
\end{aligned}$$

RSA: Schlüsselerzeugung

1. Man bestimmt große Primzahlen p und q ("groß" sind z.B. 1024-Bit-Zahlen) und berechnet $N = pq$.

   N ist der "Modulus".

2. Man berechnet $\varphi(N) = (p-1)(q-1)$ und wählt eine zu $\varphi(N)$ teilerfremde Zahl e zwischen 1 und $\varphi(N)$.

   e ist der "öffentliche Exponent".

3. Man bestimmt das $\varphi(N)$-modulare Inverse d zu e, d.h. diejenige Zahl d zwischen 1 und $\varphi(N)$, für die $d \cdot e \bmod \varphi(N) = 1$ gilt.

   d ist der "private Exponent".

4. (N, e) ist der "öffentliche Schlüssel".

Es ist keine einfache Methode bekannt, N zu faktorisieren, oder aus (N, e) den privaten Exponenten d zu berechnen.

RSA: Ver- und Entschlüsseln

Ein Sender will einem Empfänger eine geheime Botschaft schicken. Der öffentliche Schlüssel des Empfängers sei (N, e). Die Botschaft kann eine beliebige Zahl $x \in \mathbf{Z}_N = \{z \bmod N \mid z \in \mathbf{Z}\}$ sein.

1. Der Sender verschlüsselt x mit der "Verschlüsselungsfunktion"

$$E : \mathbf{Z}_N \to \mathbf{Z}_N, \quad x \mapsto x^e \bmod N$$

   und sendet $E(x) = x^e \bmod N$.

2. Der Empfänger entschlüsselt $E(x)$ mit der "Entschlüsselungsfunktion"

$$D : \mathbf{Z}_N \to \mathbf{Z}_N, \quad y \mapsto y^d \bmod N$$

   und kommt so wegen

$$D(E(x)) = E(x)^d \bmod N = (x^e)^d \bmod N = x$$

   zum Klartext der Botschaft.

Wesentlich ist hier: $(x^e)^d \bmod N = x$. [1]

[1] Erklärung nächste Seite.

RSA: $(x^e)^d \bmod N = x$, warum?

Betrachte die Botschaft x in $\mathbf{Z}_p \times \mathbf{Z}_q$:

$$x = (x \bmod p, x \bmod q).$$

Es ist $(x \bmod p)^{ed} = x \bmod p$ und $(x \bmod q)^{ed} = x \bmod q$, denn $ed \bmod \varphi(N) = 1$, d.h.

$$ed = k\varphi(N) + 1 = k(p-1)(q-1) + 1,$$

folglich (Kleiner Fermat!)

$$(x \bmod p)^{ed} = (\underbrace{(x \bmod p)^{p-1}}_{=1})^{k(q-1)}(x \bmod p)$$

$$(x \bmod q)^{ed} = (\underbrace{(x \bmod q)^{q-1}}_{=1})^{k(p-1)}(x \bmod q).$$

RSA: Beispiel

Beispiel 2. *(Unrealistisch, Zahlen viel zu klein.)*

1. *Wir wählen* $p = 97, q = 103 \Rightarrow N = pq = 9991$.
2. $\varphi(N) = (p-1)(q-1) = 96 \cdot 102 = 9792$. *Wir wählen als Verschlüsselungsexponent* $e = 193$ *(teilerfremd zu* 9792*).*
3. *Wir bestimmen pro forma mit dem erweiterten euklidischen Algorithmus den* $\mathrm{ggT}(193, 9792)$ *(der natürlich* 1 *ist) und bekommen die Darstellung* $(-53) \cdot 9792 + 2689 \cdot 193 = 1$*, der wir entnehmen, dass* 2689 *das* 9792*-modulare Inverse zu* 193 *ist. Der Entschlüsselungsexponent* d *ist also* 2689.
4. *Die Botschaft* x *sei* 1000*. Die verschlüsselte Botschaft ist dann* $y = 1000^{193} \bmod 9991 = 7014$.
5. *Zur Entschlüsselung von* y *berechnet man* $y^{2689} \bmod 9991 = 7014^{2689} \bmod 9991 = 1000$ *und bekommt so den Klartext der Botschaft wieder.*

Literatur

[CM] Ronald L. Graham, Donald E. Knuth and Oren Patashnik, *Concrete Mathematics: A Foundation for Computer Science.* Addison-Wesley, 1989; second edition, 1994.
`http://www-cs-faculty.stanford.edu/~knuth/gkp.html`

[AZ] Otto Forster, *Algorithmische Zahlentheorie*, Vieweg-Verlag, 1996.
`http://www.mathematik.uni-muenchen.de/~forster/books/azth/algzth.html`

The RSA encryption algorithm is an asymmetric encryption algorithm. RSA is widely used in public key encryption and electronic commerce. The RSA was proposed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. The RSA is composed of the letters of the three names of the three of them. The reliability of the RSA algorithm is determined by the difficulty of maximizing integer factorization. In other words, the more difficult it is to factorize a very large integer, the more reliable the RSA algorithm is. I've been working on an RSA encryption script in Lua, with the assistance of BigNumbers (http://oss.digirati.com.br/luabignum/bn/index.htm), and I pretty much have a working code.Â print(rsa_encrypt(part)) end end. function rsa_encrypt(msg) bnrsa_e = BigNum.new(rsa_e) bnrsa_n = BigNum.new(rsa_n) bnmsg = BigNum.new(msg) result = 0 quo = BigNum.new() rsa_c = BigNum.new() result = BigNum.pow(bnmsg, bnrsa_e) BigNum.div(result, bnrsa_n, quo, rsa_c).