Chapter 8

RSA

1. Most popular public-key cryptosystem.
2. Invented by Rivest/Shamir/Adleman in 1977 at MIT.
3. Patented until 2000.

8.1 Cryptosystem

Set-up Stage

1. Choose two large primes p and q.
2. Compute $n = p \cdot q$.
3. Compute $\Phi(n) = (p-1)(q-1)$.
4. Choose random b; $0 < b < \Phi(n)$, with $\gcd(b, \Phi(n)) = 1$.

   Note that b has inverse in $Z_{\Phi(n)}$.
5. Compute inverse $a = b^{-1} \bmod \Phi(n)$:

$$b \cdot a \equiv 1 \bmod \Phi(n).$$

6. Public key: $k_{pub} = (n, b)$.

   Private key: $k_{pr} = (p, q, a)$.

Encryption: done using public key, k_{pub}.

$y = e_{k_{pub}}(x) = x^b \bmod n$.

$x \in Z_n = \{0, 1, \ldots, n-1\}$.

Decryption: done using private key, k_{pr}.

$x = d_{k_{pr}}(y) = y^a \bmod n$.

Example:

Alice sends encrypted message ($x = 4$) to Bob after Bob sends her the public key.

Alice		Bob
		(1) choose $p = 3$; $q = 11$
		(2) $n = p \cdot q = 33$
		(3) $\Phi(n) = (3-1)(11-1) = 2 \cdot 10 = 20$
		(4) choose $b = 3$; $\gcd(20, 3) = 1$
$x = 4$	$\xleftarrow{k_{pub}(3,33)}$	(5) $a = b^{-1} = 7 \bmod 20$
$y = x^b \bmod n = 4^3 = 64 \equiv 31 \bmod 33$	$\xrightarrow{y=31}$	$x = y^a = 31^7 \equiv 4 \bmod 33$

Why does RSA work?

We have to show that: $d_{k_{pr}}(y) = d_{k_{pr}}(e_{k_{pub}}(x)) = x$.

$d_{k_{pr}} = y^a = x^{ba} = x^{ab} \bmod n$.

$a \cdot b \equiv 1 \bmod \Phi(n) \Longleftrightarrow a \cdot b \equiv 1 + t \cdot \Phi(n)$; t is an integer.

$d_{k_{pr}} = x^{ab} = x^{t \cdot \Phi(n)} \cdot x^1 = (x^{\Phi(n)})^t \cdot x \bmod n$.

if $x^{\Phi(n)} \equiv 1 \bmod n$ then $d_{k_{pr}} = (x^{\Phi(n)})^t \cdot x = 1^t \cdot x = 1 \cdot x = x \bmod n$.

1. Case: $\gcd(x, n) = \gcd(x, p \cdot q) = 1$

Euler's Theorem: $x^{\Phi(n)} \equiv 1 \bmod n$, *q.e.d.*

2. Case: $\gcd(x, n) = \gcd(x, p \cdot q) \neq 1$

either $x = r \cdot p$ or $x = s \cdot q$; r, s are integers such that; $r < q$, $s < p$.

assume $x = r \cdot p \Rightarrow \gcd(x, q) = 1$

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

$x^{\Phi(n)} = 1 + c \cdot q$; where c is an integer

$x \cdot x^{\Phi(n)} = x + x \cdot c \cdot q = x + r \cdot p \cdot c \cdot q = x + r \cdot c \cdot p \cdot q = x + r \cdot c \cdot n$

$x \cdot x^{\Phi(n)} \equiv x \bmod n$

$x^{\Phi(n)} \equiv 1 \bmod\ n, \quad q.e.d.$

8.2 Computational Aspects

8.2.1 Choosing p and q

Problem: Finding two large primes p, q (each > 250 bits).

Principle:

> Pick a large integer and apply primality test. In practice, a "Monte Carlo" test developed by Miller-Rabbin (pg. 136 in [Sti95]) is used. Note that a primality test does NOT require factorization.

Miller-Rabin Algorithm:

Input: p or q and arbitrary number $r < p, q$.
Output 1: Statement "p, q is composite" $\rightarrow$ always true.
Output 2: Statement "p, q is prime" $\rightarrow$ true with probability > 0.75.

In practice, the above algorithm is run 3 times (for a 1000 bit prime) and upto 12 times (for a 150 bit prime) [AM97, Table 4.4 page 148] with different parameters r. If the answer is always "p is prime", then p is with very high probability a prime.

$\mathcal{P}(p \text{ is composite }) \leq 0.25^t$ where $t =$ number of tries.

Question: What is the likelihood that a randomly picked integer p or q is prime?

Answer: $\mathcal{P}(p \text{ is prime }) \approx \frac{1}{ln(p)}$.

Example: $p \approx 2^{250} \rightarrow$ (250 bits).

$\mathcal{P}(p \text{ is prime }) = \frac{1}{\ln(2^{250})} \approx \frac{1}{173}$.

8.2.2 Choosing a and b

$k_{pub} = b$; condition: $\gcd(b, \Phi(n)) = 1$; where $\Phi(n) = (p-1)\cdot(q-1)$.

$k_{pr} = a$; where $a = b^{-1} \bmod \Phi(n)$.

Pick arbitrary b (large!) and compute:

1. Euclidean Algorithm: $s \cdot \Phi(n) + t \cdot b = \gcd(b, \Phi(n))$
2. Test if $\gcd(b, \Phi(n)) = 1$
3. Calculate a:

   Question: What is $t \cdot b \bmod \Phi(n)$?

$$\begin{aligned} t \cdot b &= (-s)\Phi(n) + 1 \\ \Rightarrow t \cdot b &\equiv 1 \bmod \Phi(n) \\ \Rightarrow t &= b^{-1} = a \bmod \Phi(n) \end{aligned}$$

Remark:

It is not necessary to find s for the computation of a.

8.2.3 Encryption/Decryption

encryption: $e_{k_{pub}}(x) = x^b \bmod n = y$.

decryption: $d_{k_{pr}}(y) = y^a \bmod n = x$.

Question: How many multiplications are required for computing x^8?

Answer: $\underbrace{x \cdot x = x^2}_{1}$; $\underbrace{x^2 \cdot x^2 = x^4}_{2}$; $\underbrace{x^4 \cdot x^4 = x^8}_{3}$.

if $0 < b < \Phi(n)$ then $\mathcal{O}(\Phi(n)) \approx \mathcal{O}(n)$.

Question: How many multiplications are required for computing x^{13}?

Answer: $\underbrace{x \cdot x = x^2}_{\text{SQ}}$; $\underbrace{x^2 \cdot x = x^3}_{\text{MUL}}$; $\underbrace{x^3 \cdot x^3 = x^6}_{\text{SQ}}$; $\underbrace{x^6 \cdot x^6 = x^{12}}_{\text{SQ}}$; $\underbrace{x^{12} \cdot x = x^{13}}_{\text{MUL}}$.

Square-and-multiply algorithm

First: binary representation of the exponent $\rightarrow x^B$; $B \leq 15$

$B = b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0$

$B = (b_3 \cdot 2 + b_2)2^2 + b_1 \cdot 2 + b_0 = ((b_3 \cdot 2 + b_2)2 + b_1)2 + b_0$

$x^B = x^{((b_3 \cdot 2 + b_2)2 + b_1)2 + b_0}$

Step	x^B
#1	$x^{b_3 \cdot 2}$
#2	$(x^{b_3 \cdot 2} \cdot x^{b_2})$
#3	$(x^{b_3 \cdot 2} \cdot x^{b_2})^2$
#4	$(x^{b_3 \cdot 2} \cdot x^{b_2})^2 \cdot x^{b_1}$
#5	$((x^{b_3 \cdot 2} \cdot x^{b_2})^2 \cdot x^{b_1})^2$
#6	$((x^{b_3 \cdot 2} \cdot x^{b_2})^2 \cdot x^{b_1})^2 \cdot x^{b_0}$

Example: $x^{13} = x^{1101_2} = x^{(b_3, b_2, b_1, b_0)_2}$

#1	$x^{b_3 \cdot 2} = x^2$	SQ
#2	$x^2 \cdot x^{b_3} = x^2 \cdot x = x^3$	MUL
#3	$(x^3)^2 = x^6$	SQ
#4	$x^6 \cdot x^0\ x^6 \cdot 1 = x^6$	
#5	$(x^6)^2 = x^{12}$	SQ
#6	$x^{12} \cdot x^{b_0} = x^{12} \cdot x = x^{13}$	MUL

Complexity: $[\log_2 n] \cdot \text{SQ} + [\frac{1}{2} \log_2 n] \cdot \text{MUL}$.

Comparison: $B = 2^{1000}$

Straight forward exponentiation: $2^{1000} \approx 10^{300}$ multiplications

$\rightarrow$ computationally impossible.

Square-and-multiply: $1.5 \cdot \log_2(2^{1000}) = 1500$ multiplications and squarings

$\rightarrow$ relatively easy.

Remark: Remember to apply modulo reduction after every multiplication and squaring operation.

Algorithm [Sti95]: computes x^B, where $B = \sum_{i=0}^{l-1} b_i 2^i$

1. $z = x$
2. for $i = l - 1$ downto 0 do:
    (a) $z = z^2 \bmod n$
    (b) if $(b_i = 1)$ then $z = z \cdot x \bmod n$

8.3 Attacks

8.3.1 Brute Force

Given $y = x^b \bmod n$, try all possible keys a; $0 \leq a < \Phi(n)$ to obtain $x = y^a \bmod n$. In practice $|\mathcal{K}| = \Phi(n) \approx n > 2^{500} \Rightarrow$ impossible.

8.3.2 Finding $\Phi(n)$

Given $n, b, y = x^b \bmod n$, find $\Phi(n)$ and compute $a = b^{-1} \bmod \Phi(n)$.
$\Rightarrow$ computing $\Phi(n)$ is believed to be as difficult as factoring n.

8.3.3 Finding a directly

Given $n, b, y = x^b \bmod n$, find a directly and compute $x = y^a \bmod n$.
$\Rightarrow$ computing a directly is believed to be as difficult as factoring n.

8.3.4 Factorization of n

Given $n, b, y = x^b \bmod n$, find $p \cdot q = n$ and compute:

$\Phi(n) = (p-1)(q-1)$

$b = a^{-1} \bmod \Phi(n)$

$x = y^a \bmod n$

$\rightarrow$ This approach is the only attack believed to be practical.

Factoring Algorithms:

1. Quadratic Sieve (QS): speed depends on the size of n; record: in 1994 factoring of $n =$RSA129, $\log_{10} n = 129$ digits, $\log_2 n = 426$ bits.
2. Elliptic Curve: similar to QS; speed depends on the size of the smallest prime factor of n, i.e., on p and q.
3. Number Field Sieve: asymptotically better than QS; record: in 1996 factoring of $n =$RSA140; $\log_{10} n = 140$ digits; $\log_2 n = 466$ bits.

Algorithm	*Complexity*
Quadratic Sieve	$\mathcal{O}(e^{(1+o(1))\sqrt{\ln(n)\ln(\ln(n))}})$
Elliptic Curve	$\mathcal{O}(e^{(1+o(1))\sqrt{2\ln(p)\ln(\ln(p))}})$
Number Field Sieve	$\mathcal{O}(e^{(1.92+o(1))(\ln(n))^{1/3}(\ln(\ln(n)))^{2/3}})$

number	**month**	**MIPS-years**	**algorithm**
RSA-100	April 1991	7	quadratic sieve
RSA-110	April 1992	75	quadratic sieve
RSA-120	June 1993	830	quadratic sieve
RSA-129	April 1994	5000	quadratic sieve
RSA-130	April 1996	500	generalized number field sieve
RSA-140	February 1999	1500	generalized number field sieve
RSA-155	August 1999	8000	generalized number field sieve

8.4 Implementation

- Hardware: 1024 bit decryption in less that 5 ms.
- Software: 1024 bit decryption in 43 ms; 1024 bit encryption in 0.65 ms
- hybrid systems, consisting of public-key and private-key algorithms: most commonly used in practice
  1. key exchange and authentication with (slow) public-key algorithm
  2. bulk data encryption with (fast) block ciphers