Chapter 7

Introduction to Public-Key Cryptography

7.1 Principle

Quick review of private-key cryptography

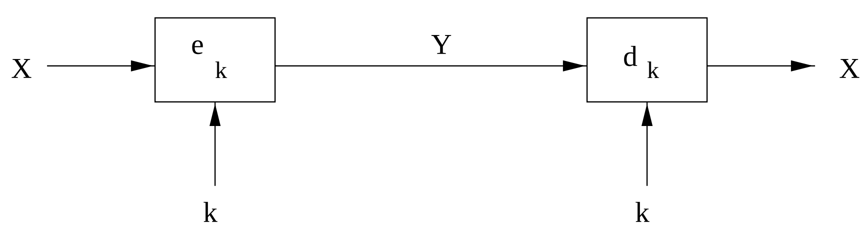

Figure 7.1: Private-key model

Two properties of private-key schemes:

1. The algorithm requires same secret key for encryption and decryption.

2. Encryption and decryption are essentially identical (symmetric algorithms).

Analogy for private key algorithms

Private key schemes are analogous to a safe box with a strong lock. Everyone with the key can deposit messages in it and retrieve messages.

Main problems with private key schemes are:

1. Requires secure transmission of secret key.

2. In a network environment, each pair of users has to have a different key resulting in too many keys ($n \cdot \ (n-1) \div 2$ key pairs).

New Idea:

Make a slot in the safe box so that everyone can deposit a message, but only the receiver can open the safe and look at the content of it. This idea was proposed in [WD76] in 1976 by Diffie/Hellman.

Idea: Split key.

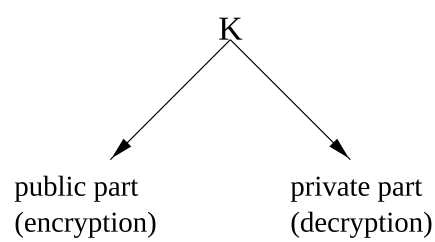

Figure 7.2: Split key idea

Protocol:

1. Alice and Bob agree on a public-key cryptosystem.

2. Bob sends Alice his public key.

3. Alice encrypts her message with Bob's public key and sends the ciphertext.

4. Bob decrypts ciphertext using his private key.

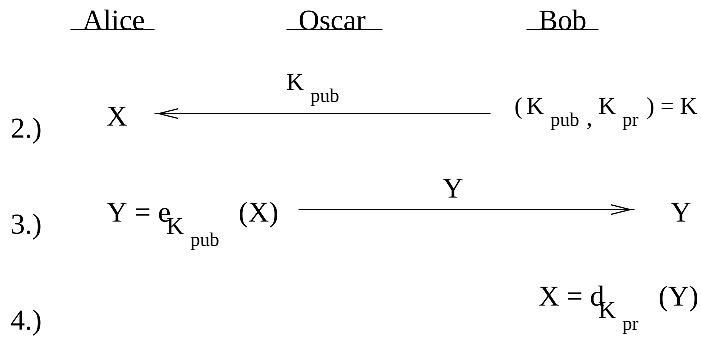

Figure 7.3: Public-key encryption protocol

7.2 One-Way Functions

All public-key algorithms are based on one-way functions.

Definition 7.2.1 *A function* f *is a "one-way function" if:*

(a) $y = f(x) \rightarrow$ *is easy to compute,*

(b) $x = f^{-1}(y) \rightarrow$ *is very hard to compute.*

Example: Discrete Logarithm (DL) one-way Function

$2^x \bmod 127 \equiv 31$

$x = ?$

Definition 7.2.2 *A* trapdoor one function *is a one-way function whose inverse is easy to compute given a side information such as the private key.*

7.3 Overview of Public-Key Algorithms

There are three families of Public-Key (PK) algorithms of practical relevance:

1. Integer factorization algorithms (RSA, ...)

2. Discrete logarithms (D–H, DSA, ...)

3. Elliptic curves (EC)

⇒ Generally speaking, public-key algorithms are much slower than private-key algorithms.

⇒ Public-Key algorithms are mainly used for key establishment and digital signatures and **not** for bulk data encryption.

Algorithm Family	Bit length of the operands
Integer Factorization (RSA)	1024
Discrete Logarithm (D–H, DSA)	1024
Elliptic curves	160
Block cipher	80

Table 7.1: Bit lengths for security level of approximately 2^{80} computations for successful attack.

7.4 Important Public-Key Standards

a) IEEE P1363. Comprehensive standard of public-key algorithms. Collection of IF, DL, and EC algorithm families, including in particular:

   – Key establishment algorithms

   – Key transport algorithms

   – Signature algorithms

   Note: IEEE P1363 does not recommend any bit lengths or security levels.

b) ANSI Banking Security standards.

ANSI#	Subject
X9.30–1	digital signature algorithm (DSA)
X9.30–2	hashing algorithm for RSA
X9.31–1	RSA signature algorithm
X9.32–2	hashing algorithms for RSA
X9.42	key management using Diffe-Hellman
X9.62 (draft)	elliptic curve digital signature algorithm (ECDSA)
X9.63 (draft)	elliptic curve key agreement and transport protocols

c) U.S. Government standards (FIPS)

FIPS#	Subject
FIPS 180-1	secure hash standard (SHA-1)
FIPS 186	digital signature standard (DSA)
FIPS JJJ (draft)	entity authentication (asymetric)

7.5 More Number Theory

7.5.1 Euclid's Algorithm

Basic Form

Given r_0 and r_1 with one larger than the other, compute the $\gcd(r_0, r_1)$.

Example 1:

$r_0 = 22, r_1 = 6.$

$\gcd(r_0, r_1) =?$

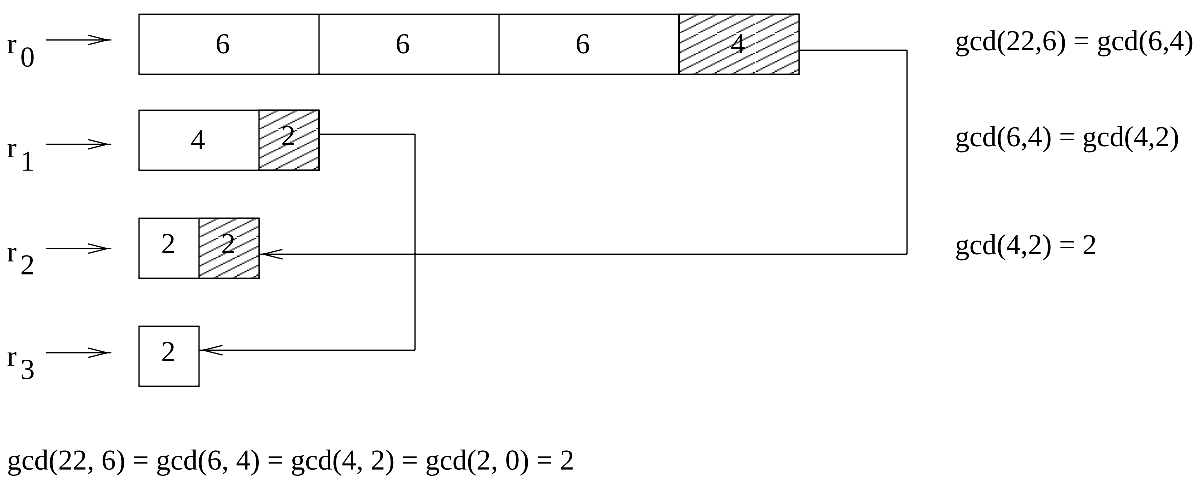

Figure 7.4: Euclid's algorithm example

Example 2:

$r_0 = 973;\ r_1 = 301.$

$973 = 3 \cdot 301 + 70.$

$301 = 4 \cdot 70 + 21.$

$70 = 3 \cdot 21 + 7.$

$21 = 3 \cdot 7 + 0.$

$\gcd(973, 301) = \gcd(301, 70) = \gcd(70, 21) = \gcd(21, 7) = 7.$

Algorithm:

$$
\begin{array}{ll}
\text{input: } r_0,\, r_1 & \\
r_0 = q_1 \cdot r_1 + r_2 & \gcd(r_0, r_1) = \gcd(r_1, r_2) \\
r_1 = q_2 \cdot r_2 + r_3 & \gcd(r_1, r_2) = \gcd(r_2, r_3) \\
\vdots & \vdots \\
r_{m-2} = q_{m-1} \cdot r_{m-1} + r_m & \gcd(r_{m-2}, r_{m-1}) = \gcd(r_{m-1}, r_m) \\
r_{m-1} = q_m \cdot r_m + 0 \leftarrow \dagger & \gcd(r_0, r_1) = \gcd(r_{m-1}, r_m) = r_m
\end{array}
$$

† - termination criteria

Extended Euclidean Algorithm

> **Theorem 7.5.1** *Given two integers r_0 and r_1, there exist two other integers s and t such that $s \cdot r_0 + t \cdot r_1 = \gcd(r_0, r_1)$.*

Question: How to find s and t?

Use Euclid's algorithm and express the current remainder r_i in every iteration in the form $r_i = s_i r_0 + t_i r_1$. Note that in the last iteration $r_m = \gcd(r_0, r_1) \stackrel{!}{=} s_m r_0 + t_m r_1 = s r_0 + t r_1$.

index	Euclid's Algorithm	$r_j = s_j \cdot r_0 + t_j \cdot r_1$
2	$r_0 = q_1 \cdot r_1 + r_2$	$r_2 = r_0 - q_1 \cdot r_1 = s_2 \cdot r_0 + t_2 \cdot r_1$
3	$r_1 = q_2 \cdot r_2 + r_3$	$r_3 = r_1 - q_2 \cdot r_2 = r_1 - q_2(r_0 - q_1 \cdot r_1)$
		$= [-q_2]r_0 + [1 + q_1 \cdot q_2]r_1 = s_3 \cdot r_0 + t_3 \cdot r_1$
$\vdots$	$\vdots$	$\vdots$
i	$r_{i-2} = q_{i-1} \cdot r_{i-1} + r_i$	$r_i = s_i \cdot r_0 + t_i \cdot r_1$
$i+1$	$r_{i-1} = q_i \cdot r_i + r_{i+1}$	$r_{i+1} = s_{i+1} \cdot r_0 + t_{i+1} \cdot r_1$
$i+2$	$r_i = q_{i+1} \cdot r_{i+1} + r_{i+2}$	$r_{i+2} = r_i - q_{i+1} \cdot r_{i+1}$
		$= (s_i \cdot r_0 + t_1 \cdot r_1) - q_{i+1}(s_{i+1} \cdot r_0 + t_{i+1} \cdot r_1)$
		$= [s_i - q_{i+1}] \cdot s_{i+1}]r_0 + [t_1 - q_{i+1} \cdot t_{i+1}]r_1$
		$= s_{i+2} \cdot r_0 + t_{i+2} \cdot r_1$
$\vdots$	$\vdots$	$\vdots$
m	$r_{m-2} = q_{m-1} \cdot r_{m-1} + r_m$	$r_m = \gcd(r_0, r_1) = s_m \cdot r_0 + t_m \cdot r_1$

Now: $s = s_m$, $t = t_m$

Recursive formulae:

$$
\boxed{
\begin{array}{ll}
s_0 = 1, & t_0 = 0 \\
s_1 = 0, & t_1 = 1 \\
s_i = s_{i-2} - q_{i-1} \cdot s_{i-1}, & t_i = t_{i-2} - q_{i-1} \cdot t_{i-1};\ i = 2, 3, 4 \ldots
\end{array}
}
$$

Remark:

a) Extended Euclidean algorithm is commonly used to compute the inverse element in Z_m. If $\gcd(r_0, r_1) = 1$, then $t = r_1^{-1} \mod r_0$.

b) For fast software implementation, the "binary extended Euclidean algorithm" is more efficient [AM97] because it avoids the division required in each iteration of the extended Euclidean algorithm shown above.

7.5.2 Euler's Phi Function

Definition 7.5.1 *The number of integers in Z_m relatively prime to m is denoted by $\Phi(m)$.*

Example 1:

$m = 6$; $Z_6 = \{0, 1, 2, 3, 4, 5\}$

$\gcd(0, 6) = 6$

$\gcd(1, 6) = 1 \ \leftarrow$

$\gcd(2, 6) = 2$

$\gcd(3, 6) = 3$

$\gcd(4, 6) = 2$

$\gcd(5, 6) = 1 \ \leftarrow$

$\Phi(6) = 2$

Example 2:

$m = 5$; $Z_5 = \{0, 1, 2, 3, 4\}$

$\gcd(0, 5) = 5$

$\gcd(1, 5) = 1 \;\leftarrow$

$\gcd(2, 5) = 1 \;\leftarrow$

$\gcd(3, 5) = 1 \;\leftarrow$

$\gcd(4, 5) = 1 \;\leftarrow$

$\Phi(5) = 4$

Theorem 7.5.2 *If* $m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$*, where* p_i *are prime numbers and* e_i *are integers, then:*

$$\Phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_{i-1}})$$

.

Example:

$m = 40 = 8 \cdot 5 = 2^3 \cdot 5 = p_1^{e_1} \cdot p_2^{e_2}$

$\Phi(m) = (2^3 - 2^2)(5^1 - 5^0) = (8 - 4)(5 - 1) = 4 \cdot 4 = 16$

Theorem 7.5.3 Euler's Theorem

If $\gcd(a, m) = 1$*, then:*

$$a^{\Phi(m)} \equiv 1 \bmod m$$

.

Example:

$m = 6$; $a = 5$

$\Phi(6) = \Phi(3 \cdot 2) = (3 - 1)(2 - 1) = 2$

$5^{\Phi(6)} = 5^2 = 25 \equiv 1 \bmod 6$