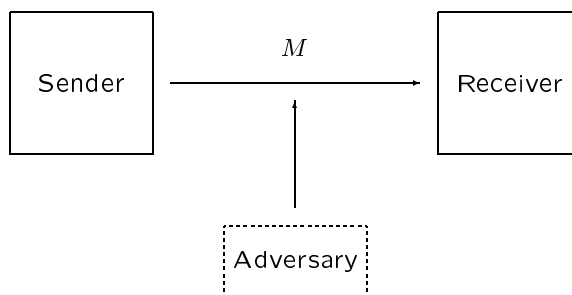


Public-Key Cryptography

Çetin Kaya Koç
Oregon State University

1

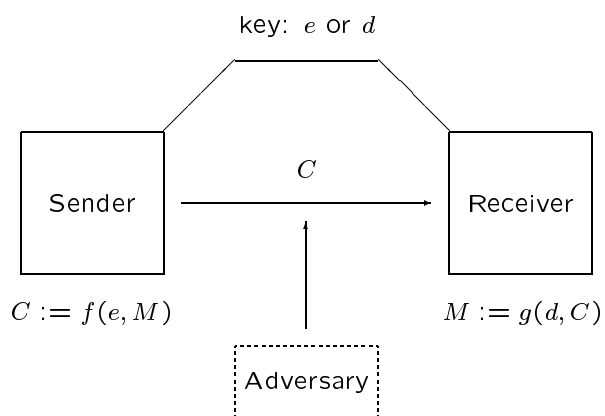


Objective: Secure communication over an insecure channel

2

Solution: Secret-key cryptography

Exchange the key over a **secure** channel



3

Secret-Key Cryptography

- Functions $f(e, -)$ and $g(d, -)$ are inverses of one another
- Encryption and decryptions processes are **symmetric**:

Either $f = g$ and $e \neq d$

$C := f(e, M)$ and $M := f(d, C)$

d is **easily** deduced from e

e is **easily** deduced from d

Or $f \neq g$ and $e = d$

$C := f(e, M)$ and $M := g(e, C)$

g is **easily** deduced from f

f is **easily** deduced from g

4

Example: **Hill Algebra**

Encode 26 letters A,B,C,...,Z as 0,1,2,...,25

Select $d \times d$ matrix Z of integer elements and find its inverse Z^{-1} modulo 26, e.g.,

$$Z = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \text{ and } Z^{-1} = \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix}$$

Verify:

$$\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} = \begin{bmatrix} 105 & 78 \\ 130 & 79 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Encryption: $C := ZM \text{ mod } 26$

Encryption Key: $e = Z$

Decryption: $M := Z^{-1}C \text{ mod } 26$

Decryption Key: $d = Z^{-1}$

Functions: $f = g$: matrix-vector product

Plaintext : HELP

$$M_1 = \begin{bmatrix} H \\ E \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \end{bmatrix} ; M_2 = \begin{bmatrix} L \\ P \end{bmatrix} = \begin{bmatrix} 11 \\ 15 \end{bmatrix}$$

Encryption: $C_1 := ZM_1$

$$\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 7 \\ 4 \end{bmatrix} = \begin{bmatrix} 33 \\ 34 \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} H \\ I \end{bmatrix}$$

Encryption: $C_2 := ZM_2$

$$\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 11 \\ 15 \end{bmatrix} = \begin{bmatrix} 78 \\ 97 \end{bmatrix} = \begin{bmatrix} 0 \\ 19 \end{bmatrix} = \begin{bmatrix} A \\ T \end{bmatrix}$$

Cryptotext: HIAT

Cryptotext: HIAT

$$C_1 = \begin{bmatrix} H \\ I \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \end{bmatrix} ; C_2 = \begin{bmatrix} A \\ T \end{bmatrix} = \begin{bmatrix} 0 \\ 19 \end{bmatrix}$$

Decryption: $M_1 := Z^{-1}C_1$

$$\begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 241 \\ 212 \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \end{bmatrix} = \begin{bmatrix} H \\ E \end{bmatrix}$$

Decryption: $M_2 := Z^{-1}C_2$

$$\begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} \begin{bmatrix} 0 \\ 19 \end{bmatrix} = \begin{bmatrix} 323 \\ 171 \end{bmatrix} = \begin{bmatrix} 11 \\ 15 \end{bmatrix} = \begin{bmatrix} L \\ P \end{bmatrix}$$

Plaintext: HELP

Problems with secret-key cryptography:

- requires establishment of a **secure** channel for key exchange
- two parties **cannot** start communication if they never met

Alternative: Public-Key Cryptography

- requires establishment of a **public-key directory** in which everyone publishes their **encryption** keys
- two parties **can** start communication even if they never met

- ♣ provides ability to **sign** digital data

Key Exchange

- Parties S and R agree on a large prime number p
(This can be accomplished in public)
- S selects $a \in GF(p)$ and computes a^{-1} such that $aa^{-1} \equiv \text{mod } p - 1$
S keeps these integers secret
- R selects $b \in GF(P)$ and computes b^{-1} such that $bb^{-1} \equiv \text{mod } p - 1$
R keeps these integers secret
- Suppose S wants to pass the key x to R

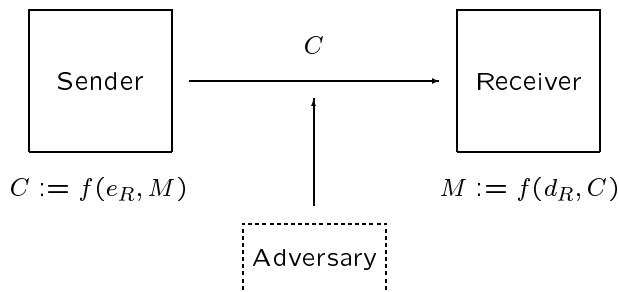
9

- S computes $x^a \in GF(p)$ and sends it to R
- R computes $(x^a)^b = x^{ab} \in GF(p)$ and sends it to S
- S computes $(x^{ab})^{a^{-1}} = x^b \in GF(p)$ and sends it to R
- R Computes $(x^b)^{b^{-1}} = x \in GF(p)$
- Adversary knows the field $GF(p)$ and sees x^a, x^{ab}, x^b
- Computing α from x^α in the field $GF(p)$ is discrete analogue of taking logarithms
- **Discrete logarithm problem** is known to be very hard

10

public-key directory

User	Key
S	e_S
R	e_R



11

Public-Key Cryptography

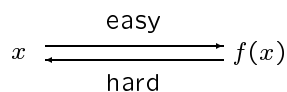
- Functions $f(e_R, -)$ and $f(d_R, -)$ are inverse of one another
- Encryption and decryptions processes are **asymmetric**:
 $e_R \neq d_R$
 $C := f(e_R, M)$ and $M := f(d_R, C)$
 e_R is **public**; known to everyone
 d_R is **private**; known only to User R
 e_R is **easily** deduced from d_R
 d_R is **NOT easily** deduced from e_R

12

A public-key cryptography system is based on a function $f(x)$ such that

Given x , computing $y = f(x)$ is EASY

Given $y = f(x)$, computing x is HARD



We call $f(x)$ a **one-way function**

In order to decide what is hard:

Sometimes we use **the theory of complexity**

Often **the test of time** determines

13

Example: **Discrete Logarithm**

Given x , a , and p , computing $y \equiv x^a \pmod{p}$ is EASY

However, given y , x , and p , computing a is HARD

Example: **Factoring**

Given x and y , computing $n = xy$ is EASY

However, given n , computing the factors x and y is HARD

Example: **Discrete Square-root**

Given x and n , computing $a \equiv x^2 \pmod{n}$ is EASY

However, given a and n , computing x is HARD

14

Discrete Logarithm Example

For $x = 6$, $a = 9$, $p = 11$, we compute

$$y \equiv x^a \equiv x((x^2)^2)^2 \pmod{p}$$

with 4 multiplications:

$$\begin{aligned} y &= 6((6^2)^2)^2 = 6((36)^2)^2 \\ &= 6((3)^2)^2 = 6(9)^2 \\ &= 6(81) = 6(4) \\ &= 24 = 2 \end{aligned}$$

However, finding an a such that

$$6^a \equiv 2 \pmod{11}$$

is hard

We need to try all possibilities (from 1 to $p-1$) to obtain such a

15

One additional structure about the function $y = f(x)$ is needed to design a public-key cryptosystem

Given y and some **special information** about $f(x)$, computing x is EASY

Given y without this special information, computing x is HARD

We call $f(x)$ a **one-way trapdoor function**

Special information is **trapdoor** information

16

Example: **Trapdoor Knapsack**

Knapsack Problem: Let $I = \{0, 1, \dots, n-1\}$. Given the integer vector $A = \{a_0, a_1, \dots, a_{n-1}\}$ and another integer X , is there a $J \subseteq I$ such that

$$\sum_{i \in J} a_i = X$$

Easy Knapsack Problem: If the numbers a_i have the **superincreasing property**, i.e.,

$$\sum_{i=0}^{j-1} a_i < a_j$$

then the knapsack problem is easy

Hard Knapsack Problem: Without the superincreasing property the knapsack problem (in general) is hard

17

Easy Example:

$A = \{1, 2, 4, 8, 16\}$; Superincreasing

$1 < 2$; $1+2 < 4$; $1+2+4 < 8$; $1+2+4+8 < 16$

Let $X = 23$. Solution is found by computing the binary expansion of $X = 23 = (10111)_2$, thus $1 + 2 + 4 + 16 = 23$

Hard Example:

$A = \{3, 4, 5, 12, 13\}$; Nonsuperincreasing

Let $X = 19$. We need to try all subsets of A to find out which one of these sums to 19

$$\begin{array}{lll} 3 + 4 = 7 & 3 + 5 = 8 & 3 + 12 = 15 \\ 3 + 13 = 16 & 4 + 5 = 9 & 4 + 12 = 16 \\ 4 + 13 = 17 & 5 + 12 = 17 & 5 + 13 = 18 \\ 12 + 13 = 25 \end{array}$$

$$\begin{array}{ll} 3 + 4 + 5 = 12 & 3 + 4 + 12 = 19 \\ 3 + 4 + 13 = 20 & 4 + 5 + 12 = 21 \\ 4 + 5 + 13 = 22 & 5 + 12 + 13 = 30 \end{array}$$

18

How to design a trapdoor knapsack

Take an easy knapsack and disguise it

Example: $A = \{1, 2, 4, 8, 16\}$ Select a prime p larger than the sum 31, for example $p = 37$ Select t and compute $t^{-1} \bmod p$, for example, $t = 17$ and $t^{-1} = 24$

Produce a new knapsack vector B from A such that

$$b_i \equiv a_i t \pmod{p}$$

This gives $B = \{17, 34, 31, 25, 13\}$

This knapsack problem is nonsuperincreasing

However, with the special trapdoor information $t = 17$, $t^{-1} = 24$, and $p = 37$, we can convert this problem to the easy version

19

Example: Given B and $X = 72$, is there a subset of B summing to X ?

Solve the easy knapsack with X'

$$\begin{aligned} X' &\equiv X t^{-1} \pmod{37} \\ &= 26 \end{aligned}$$

$A = \{1, 2, 4, 8, 16\}$ and $X' = 2 + 8 + 16 = 26$

This gives the solution for the hard knapsack:

$B = \{17, 34, 31, 25, 13\}$ and $X = 34 + 25 + 13 = 72$

20

Knapsack Public-Key Cryptosystem

User R:

Selects an easy vector A with $|A| = n > 100$

Selects a prime p larger than the sum $\sum_{i=0}^{n-1} a_i$

Selects t and t^{-1} such that $tt^{-1} \equiv 1 \pmod{p}$

Obtains the hard knapsack B from A using
 $b_i \equiv a_i t \pmod{p}$

Publishes B

Keeps A , t , t^{-1} , and p secret

21

User S: wants to send a message to R

User S:

Takes the message M and breaks into n bits

Let m_i be the i th bit of M ($m_i = 0$ or $m_i = 1$)

Computes C as

$$C := \sum_{i=0}^{n-1} m_i b_i$$

and sends C to R

User R:

Receives C and computes $C' \equiv Ct^{-1} \pmod{p}$ and solves the easy knapsack

Uses this solution to obtain M

22

Example

User R: Publishes $B = \{17, 34, 31, 25, 13\}$

Keeps $A = \{1, 2, 4, 8, 16\}$, $t = 17$, $t^{-1} = 24$, and $p = 37$ secret

User S: wants to send the message $M = 12$ to User R

User S: Takes $M = 12 = (01100)_2$

Computes

$C := 0 \cdot 17 + 1 \cdot 34 + 1 \cdot 31 + 0 \cdot 25 + 0 \cdot 13$
which gives $C = 65$

Send $C = 65$ to User R

23

User R:

Receives $C = 65$

Computes $C' = 65t^{-1} = 65 \cdot 24 \equiv 6 \pmod{37}$

Solves the easy knapsack problem:

$$6 = \underline{0} \cdot 1 + \underline{1} \cdot 2 + \underline{1} \cdot 4 + \underline{0} \cdot 8 + \underline{0} \cdot 16$$

This gives the message as $(01100)_2 = 12$

24

Other important public-key cryptosystems:

RSA (1978): factoring problem

McEliece (1978): decoding problem for general linear codes

ElGamal (1985): discrete log problem

DSS (1992): discrete log problem

25

The RSA Algorithm

The RSA algorithm was invented by **Rivest**, **Shamir**, and **Adleman** in 1977

p and q be two distinct large random primes

The modulus n is the product of these two primes: $n = pq$

Euler's totient function of n is given by

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

Now, select a number $1 < e < \phi(n)$ such that

$$\gcd(e, \phi(n)) = 1$$

and compute d with

$$d = e^{-1} \pmod{\phi(n)}$$

using extended Euclid's algorithm

26

Here, e is the **public exponent** and d is the **private exponent**

Usually one selects a small public exponent, e.g., $e = 2^{16} + 1$

The modulus n and the public exponent e are **published**

The value of d and the prime numbers p and q are kept **secret**

Encryption is performed by computing

$$C = M^e \pmod{n}$$

where M is the plaintext such that $0 \leq M < n$

Decryption is performed by computing

$$M = C^d \pmod{n}$$

27

The correctness of the RSA algorithm follows from Euler's theorem:

Let n and a be positive, relatively prime integers. Then

$$a^{\phi(n)} = 1 \pmod{n}$$

Since we have $ed = 1 \pmod{\phi(n)}$, i.e., $ed = 1 + K\phi(n)$ for some integer K , we can write

$$\begin{aligned} C^d &= (M^e)^d \pmod{n} \\ &= M^{ed} \pmod{n} \\ &= M^{1+K\phi(n)} \pmod{n} \\ &= M \cdot (M^{\phi(n)})^K \pmod{n} \\ &= M \cdot 1 \pmod{n} \end{aligned}$$

provided that $\gcd(M, n) = 1$

28

The exception $\gcd(M, n) > 1$ can be dealt as follows: According to Carmichael's theorem

$$M^{\lambda(n)} = 1 \pmod{n}$$

where $\lambda(n)$ is Carmichael's function

$$\lambda(n) = \lambda(pq) = \frac{(p-1)(q-1)}{\gcd(p-1, q-1)}$$

$\lambda(n)$ is always a proper divisor of $\phi(n)$ when n is the product of distinct odd primes; in this case $\lambda(n)$ is smaller than $\phi(n)$

The relationship between e and d is given by

$$M^{ed} = M \pmod{n} \text{ if } ed = 1 \pmod{\lambda(n)}$$

If n is a product of distinct primes, the above holds for all M , thus covering the exception $\gcd(M, n) > 1$ in Euler's theorem

As an example, we construct a simple RSA cryptosystem as follows:

Pick $p = 11$ and $q = 13$, and compute

$$n = p \cdot q = 11 \cdot 13 = 143$$

$$\phi(n) = (p-1) \cdot (q-1) = 10 \cdot 12 = 120$$

We can also compute Carmichael's function of n as

$$\begin{aligned} \lambda(pq) &= \frac{(p-1)(q-1)}{\gcd(p-1, q-1)} \\ &= \frac{10 \cdot 12}{\gcd(10, 12)} = \frac{120}{2} = 60 \end{aligned}$$

Thus,

$$M^{ed} = M \pmod{143} \text{ if } ed = 1 \pmod{60}$$

for all $0 \leq M \leq 142$

The public exponent e is selected such that $1 < e < \phi(n)$ and

$$\gcd(e, \phi(n)) = \gcd(e, 120) = 1$$

For example, $e = 17$ would satisfy this constraint

The private key exponent d is computed by

$$\begin{aligned} d &= e^{-1} \pmod{\phi(n)} \\ &= 17^{-1} \pmod{120} \\ &= 113 \end{aligned}$$

which is computed using the extended Euclid algorithm, or any other algorithm for computing the modular inverse

The user publishes the public exponent and the modulus pair: $(e, n) = (17, 143)$,

and keeps the following as private: $d = 113$, $p = 11$, $q = 13$

A typical encryption/decryption process is executed as follows:

$$\begin{aligned} \text{Plaintext: } & M = 50 \\ \text{Encryption: } & C := M^e \pmod{n} \\ & C := 50^{17} \pmod{143} \\ & C = 85 \end{aligned}$$

$$\begin{aligned} \text{Ciphertext: } & C = 85 \\ \text{Decryption: } & M := M^d \pmod{n} \\ & M := 85^{113} \pmod{143} \\ & M = 50 \end{aligned}$$

Exchange of Private Messages

The public-key directory contains pairs (e, n) for each user. The users wishing to send private and/or signed messages to one another refer to the directory to get these parameters. For example, the directory might be arranged as follows:

User	Public Keys
Alice	(e_a, n_a)
Bob	(e_b, n_b)
Charlie	(e_c, n_c)
...	...

where the pair n_a and e_a respectively are the modulus and the public exponent for Alice, and so on. As an example, we show how Alice sends her private message M to Bob.

In our simple protocol example Alice executes the following steps:

1. Alice locates Bob's name in the directory and obtains his public exponent and the modulus: (e_b, n_b)
2. Alice computes $C := M^{e_b} \pmod{n_b}$
3. Alice sends C to Bob over the network
4. Bob receives C
5. Bob computes $M = C^{d_b} \pmod{n_b}$ in order to obtain M

Exchange of Signed Private Messages

The procedure for sending a signed private message is a little more involved. For example, suppose Alice wants to send her signed private message to Bob.

1. Alice locates Bob's name in the directory and obtains his public-key exponent and the modulus: (e_b, n_b)
2. Alice compares her modulus n_a to Bob's modulus. One of these will be true:
 - $n_a < n_b$, or
 - $n_b < n_a$

The order of signing and encrypting are reversed for each of these cases

Alice first forms her message M such that $0 \leq M < \min(n_a, n_b)$

Assuming $n_a < n_b$:

3. Alice signs: $C_1 = M^{d_a} \pmod{n_a}$
4. Alice encrypts: $C_2 = C_1^{e_b} \pmod{n_b}$
5. Alice sends C_2 to Bob
6. Bob receives C_2
7. Bob decrypts: $C_1 = C_2^{d_b} \pmod{n_b}$
8. Bob verifies: $M = C_1^{e_a} \pmod{n_a}$

Assuming $n_b < n_a$:

3. Alice encrypts: $C_1 = M^{e_b} \pmod{n_b}$

4. Alice signs: $C_2 = C_1^{d_a} \pmod{n_a}$

5. Alice sends C_2 to Bob

6. Bob receives C_2

7. Bob verifies: $C_1 = C_2^{e_a} \pmod{n_a}$

8. Bob decrypts: $M = C_1^{d_b} \pmod{n_b}$

Example

User	Private Keys
Alice	$d_a = 113, p_a = 11, q_a = 13$
Bob	$d_b = 263, p_b = 17, q_b = 19$

User	Public Keys
Alice	$e_a = 17, n_a = 143$
Bob	$e_b = 23, n_b = 323$

Alice executes the following steps in order to send a signed and private message:

1. Alice locates Bob's name in the directory and obtains his public-key exponent and the modulus: $(e_b, n_b) = (23, 323)$
2. Alice compares her modulus $n_a = 143$ to Bob's modulus $n_b = 323$. Since $n_a < n_b$, she executes the following steps:

3. Alice forms her message $0 \leq M < 143$, let $M = 32$

4. Alice signs: $C_1 = 32^{113} \pmod{143}$, and obtains $C_1 = 54$

5. Alice encrypts: $C_2 = 54^{23} \pmod{323}$, and obtains $C_2 = 232$

6. Alice sends $C_2 = 232$ to Bob

7. Bob decrypts: $C_1 = 232^{263} \pmod{323}$, obtaining $C_1 = 54$

8. Bob verifies: $M = 54^{17} \pmod{143}$, obtaining $M = 32$

Note that **changing the order** of signing and encrypting would be **incorrect**. If Alice encrypts before signing, then she would compute:

$$\begin{aligned} M &= 32 \\ C_1 &= 32^{23} \pmod{323} \\ &= 280 \\ C_2 &= 280^{113} \pmod{143} \\ &= 37 \end{aligned}$$

and send $C_2 = 37$ to Bob. After having received 37, Bob would compute

$$\begin{aligned} C_1 &= 37^{17} \pmod{143} \\ &= 137 \\ M &= 137^{263} \pmod{323} \\ &= 35 \end{aligned}$$

which is not equal to the message: $35 \neq 32$

Practice

In practice the protocols are more complicated; for example, the order of signing and encryption are not interchanged and signing apply to messages of arbitrary length.

The signature is often computed by first computing a hash value of the long message and then signing this hash value.

Security of the RSA Algorithm

If one can factor quickly, then one can break the RSA algorithm:

Alice's public keys are published: (e_a, n_a)

Factor n_a to get p_a and q_a

Compute $\phi(n_a) = \phi(p_a q_a) = (p_a - 1)(q_a - 1)$

Compute $d_a = e_a^{-1} \pmod{\phi(n_a)}$

Thus, we can now intercept and decrypt all messages sent to Alice

Thus, factoring \implies breaking RSA

However, breaking RSA $\stackrel{?}{\implies}$ factoring

No proof exists that breaking RSA is equivalent factoring

Difficulty of Factoring

The best algorithm (Number Field Sieve) requires roughly

$$e^{1.92(\ln n)^{1/3}(\ln \ln n)^{2/3}}$$

operations to factor the composite number n

Digits	MIPS-years
100	203
150	$3 \cdot 10^5$
200	$1 \cdot 10^8$
250	$2 \cdot 10^{10}$
300	$2 \cdot 10^{12}$
350	$1 \cdot 10^{14}$
400	$5 \cdot 10^{15}$
450	$2 \cdot 10^{17}$
500	$4 \cdot 10^{18}$