Public Key Cryptography

All **secret key** algorithms & **hash** algorithms do the same thing but **public key** algorithms look **very different** from each other.

What is common among all of them is that each participant has **two keys**, public and private, and most of them are based on **modular arithmetic**.

---

### Modular Arithmetic

\[ x \mod n \] is the **remainder** of \( x \) when divided by \( n \).

- e.g., \( 8 \mod 10 = 8 \), \( 18 \mod 10 = 8 \), \( 24 \mod 10 = 4 \)
- \( 8 \mod 7 = 1 \), \( 18 \mod 7 = 4 \), \( 24 \mod 7 = 3 \)

**Addition:**

**Example:** addition mod 10

- \( 8 + 8 = 6 \),
- \( 1 + 9 = 0 \),
- \( 7 + 6 = 3 \)

See **Fig. 6-1** for addition mod 10 Table:
**Encryption:** Addition mod 10 can be used for encryption of digits.

Add \( k \), a **secret key** between 1-9, to each digit.

**Example:** if \( k = 7 \), then 1987 is **encrypted** to 8654.

**Decryption:** Add \(-k\), the **additive inverse** of \( k \), to each digit.

An additive inverse of \( x \) is the number you'd have to add to \( x \) to get 0.

**Example:** if \( k = 7 \), then \(-k\) is 3 since \( 7 + 3 = 0 \)

Thus 8654 will is decrypted to 1987.

In the above table (Fig. 6-1), each "0" is the intersection of \( k \) and \(-k\), e.g., 0 is the intersection of 3 and 7.

![Addition Modulo 10 Table](image)

*Figure 6-1. Addition Modulo 10*
Multiplication:

Example: multiplication mod 10:

\[ 8 \times 8 = 4, \ 1 \times 9 = 9, \ 7 \times 6 = 2 \]

See Fig. 6-2 for multiplication mod 10 Table:

![Multiplication Modulo 10 Table](image)

Encryption: multiplication by 1, 3, 7, 9 works as a cipher since it performs 1-1 mapping.

Example: if \( k = 7 \), then 1987 is encrypted to 7369

Decryption: is done by multiplying each digit by \( k^{-1} \), the multiplicative inverse of \( k \).

It is the number to multiply by \( k \) to get 1.

Example: if \( k = 7 \), then \( k^{-1} \) is 3 since \( 7 \times 3 = 1 \).
In the above table (Fig. 6-2),

each "1" is the intersection of $k$ and $k^{-1}$.

Note that only \{1,3,7,9\} have multiplicative inverse mod 10.

What is so special about the set \{1,3,7,9\}? These numbers are relatively prime to 10, i.e., they do not share with 10 any common factors other than 1. Note that 9 is not a prime number but it is relatively prime to 10.

How many numbers less than $n$ are relatively prime to $n$?

This quantity is referred to as:

$$\varphi(n)$$ and is called the **totient function**.

- **If $n$ is prime:**
  then \{1,2, ..., $n$-1\} are all relatively prime and thus $\varphi(n) = n-1$.

- **If $n = p.q$ where $p$ and $q$ are two distinct primes,**
  then $\varphi(n) = (p-1)(q-1)$.

**Example:** for $n = 10 = 2.5$, $\varphi(10) = (2-1)(5-1) = 1.4 = 4$, which is the set \{1,3,7,9\}.

**Exponentation:**

**Example:** exponentiation mod 10
$4^2 = 6$, $8^8 = 6$, $1^9 = 1$, $7^6 = 9$

See Fig. 6-3 for exponentiation mod 10 Table:

\[
\begin{array}{cccccccccccc}
\text{x}^\text{y} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 1 & 2 & 4 & 8 & 6 & 2 & 4 & 8 & 6 & 2 & 4 & 8 & 6 \\
3 & 1 & 3 & 9 & 7 & 1 & 3 & 9 & 7 & 1 & 3 & 9 & 7 & 1 \\
4 & 1 & 4 & 6 & 4 & 6 & 4 & 6 & 4 & 6 & 4 & 6 & 4 & 6 \\
5 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
6 & 1 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
7 & 1 & 7 & 9 & 3 & 1 & 7 & 9 & 3 & 1 & 7 & 9 & 3 & 1 \\
8 & 1 & 8 & 4 & 2 & 6 & 8 & 4 & 2 & 6 & 8 & 4 & 2 & 6 \\
9 & 1 & 9 & 1 & 9 & 1 & 9 & 1 & 9 & 1 & 9 & 1 & 9 & 1 \\
\end{array}
\]

Figure 6-3. Exponentiation Modulo 10

Amazing fact about $\phi(n)$:

\[x^m \mod n = x^{m \mod \phi(n)} \mod n\]

Since $\phi(10)=4$, in Fig. 6-3:

\[x^m \mod 10 = x^{m \mod 4} \mod 10\]

the $i^{th}$ column is identical to the $i+4^{th}$ column.

e.g., $1^{st} = 5^{th} = 9^{th}$ and $3^{rd} = 7^{th} = 11^{th}$.

**Special case:**

if $m = 1 \mod \phi(n)$, then for any number $x$,

\[x^m \mod n = x \mod n.\]
Example: For \( n = 10, \Phi(10) = 4 \)

Since \( m = 9 \) is \( 1 \mod 4 \):

\[
3^9 \mod 10 = 3 \mod 10 = 3 & \\
6^9 \mod 10 = 6 \mod 10 = 6
\]

in general:

for any \( x \): \( x^9 \mod 10 = x \mod 10 = x \)

An exponentiative inverse of \( e \) is the number \( d \) such that:

\[
e.d \equiv 1 \mod \Phi(n)
\]

Example: For \( n = 10, \Phi(10) = 4 \):

\( e = 3 \) and \( d = 7 \) are exponentiative inverses since:

\[
3 \cdot 7 = 21 = 1 \mod 4
\]

Encrypt/Decrypt:

- To encrypt \( m \): compute \( c = m^e \mod n \)
- To decrypt \( c \): compute \( m = c^d \mod n \)

Example:

encrypt \( m = 8 \): \( c = 8^3 = 2 \) 

decrypt \( c = 2 \): \( m = 2^7 = 8 \)

Sign/Verify:
To sign \( m \): compute \( s = m^d \mod n \):

To verify \( s \): compute \( m = s^e \mod n \):

Example:

sign \( m = 8 \): \( s = 8^7 = 2 \)
verify \( s = 2 \): \( m = 2^3 = 8 \)

In public cryptography:

\(<e, n>\) is public key

\(<d, n>\) is private key

RSA: Rivest, Shamir & Adleman

Key length:

Variable (long for security, short for efficiency).
Most common values is 512 & 1024 bits.

Block size:

plain text length is variable but less than key length

cipher text length equals key length.
Thus RSA is used for encrypting small amount of data,
e.g., a secret key and then use the secret key cryptography for encrypting/decrypting large amount of data.

**RSA Algorithm:**

**Generate public & private keys pair:**

1. Choose two large primes \( p \) and \( q \).
   (Typically 256 bits each & keep them secret).

2. Compute \( n = p.q \) & \( \phi(n) = (p-1)(q-1) \).
   (It is very hard to factor \( n \) into \( p \) & \( q \)).

3. Choose a number \( e \) that is relatively prime to \( \phi(n) \).

4. Find a number \( d \) that is the exponentiative inverse of \( e \)
   i.e., \( e.d = 1 \) mod \( \phi(n) \).

5. The **public key**: \(<e,n>\) & the **private key**: \(<d,n>\).

**Encrypt/Decrypt:**

To encrypt a message \( m \) (less than \( n \)):
\[
c = m^e \mod n
\]

To decrypt \( c \):
\[
m = c^d \mod n
\]

This works since:
\[
c^d \mod n = (m^e)^d \mod n
= m^{e.d} \mod n
= m \mod n \quad \text{// since } e.d = 1 \text{ mod } \phi(n)
= m \quad \text{// since } m < n
\]

**Sign/Verify:**
To sign a message \( m \) (less than \( n \)):

\[
s = m^d \mod n
\]

To verify \( s \):

\[
m = s^e \mod n
\]

This also works since:

\[
s^e \mod n = m^{e.d} \mod n = m \mod n = m
\]

**Why is RSA Secure:**

Everyone knows the public key: \( <e, n> \).

To find the private key \( <d,n> \)
you need to know \( \phi(n) \) since \( e.d = 1 \mod \phi(n) \).

To know \( \phi(n) \) you need to know \( p \) and \( q \) since

\[
\phi(n) = (p-1).(q-1).
\]

Thus to break RSA you should know:

**how to factor \( n \) to find \( p \) and \( q \).**

Factoring a big number like \( n \) is hard. The best technique to factor 512 bit number takes 30,000 MIPS-years!

**Efficiency of RSA Operations:**

**Exponentiation**

How to compute \( 123^{54} \mod 678 \)?

\[
123^2 = 123.123 = 15129 = 213 \mod 678 \\
123^3 = 123.213 = 26199 = 435 \mod 678 \\
123^4 = 123.435 = 53505 = 621 \mod 678 \\
...... \\
123^{54} = ...... = 87 \mod 678
\]
This requires 54 small number **multiplications** and 54 small number **divisions**.

**How to compute** $123^{32} \text{ mod } 678$?

$123^2 = 123.123 = 15129 = 213 \text{ mod } 678$

$123^4 = 213.213 = 45369 = 621 \text{ mod } 678$

$123^8 = 621.621 = 385641 = 537 \text{ mod } 678$

$123^{16} = 537.537 = 288369 = 219 \text{ mod } 678$

$123^{32} = 219.219 = 47961 = 501 \text{ mod } 678$

This requires **5** **multiplications** and **5** **divisions** instead of **32**.

To efficiently compute $123^{54}$: 54 is represented in binary as:

```
1         1                  0              1                1              0
|              |           |             |          |
((((  (123^2)123          ) 2                    )2123          )2123      )2
```

This requires **8** **multiplications** and **8** **divisions** instead of **32**.

Each 1 requires two multipliactions and two divisions and each 0 requires one multipliaction and one division.

Thus in the above we have three 1s and two 0s and that yields: $3 \times 2 + 2 \times 1 = 8$.

Note that we ignore the leading 1.

**Another example:** $y^{14}$, 14 is represented in binary as:

```
1         1                 1                  0
|                  |                   |
((             ( y^2) y            )2y               )^2
```

This requires **8** **multiplications** and **8** **divisions** instead of **32**.
This requires $5$ multiplications and $5$ divisions instead of $32$.

**Generating RSA Keys**

**Finding e:**

Two popular values for $e$ are: $3$ and $65537$ ($2^{16} + 1$).

These make public key operations on message $m$ faster (encryption and signature verification is $m^e$):

- $m^3$ requires 2 multiplications & 2 divisions.
- $m^{65537}$ requires 17 multiplications & 17 divisions since the binary value of $65537$ is $100..01$ (15 zeros).

**Finding n:**

**If $e = 3$:**

Choose random numbers $x$ and $y$ then:

- $p = (2x+1)*3+2$
- $q = (2y+1)*3+2$

**If $e = 65537$:**

Randomly choose $p$ and $q$ and make sure that they are not $1$ mod $65537$ (the probability of rejection is $1$ in $2^{16}$).

Once we selected $p$ and $q$, then:
\[ n = p \cdot q \quad \& \quad \varnothing(n) = (p-1)(q-1). \]

**Finding d:**

How to find \( d \) such that \( e \cdot d = 1 \mod \varnothing(n) \)? Use *Euclid* algorithm (see Section 7.4, page 187 of textbook).

**The RSA keys:**

- **public key:** \(<3, 65537, n>\)
- **private key:** \(<d, n>\)

---

**Diffie-Hellman**

Alice and Bob agree on: \( p \) (large prime) & \( g < p \).

**Alice**

- Pick \( S_A \) (512-bit random number)
- Compute \( T_A = (g^{S_A}) \mod p \)
- \( T_A \)
- Compute \( X = T_B^{S_A} \mod p \)

**Bob**

- Pick \( S_B \) (512-bit random number)
- Compute \( T_B = (g^{S_B}) \mod p \)
- \( T_B \)
- Compute \( Y = T_A^{S_B} \)

**X is the same as Y!**

*why?*

\[
X = T_B^{S_A} = g^{SBSA} \\
Y = T_A^{SB} = g^{SASB}
\]
No one can compute $g^{(S_A S_B)}$ by knowing $g^{(S_A)}$ & $g^{(S_B)}$

The bucket Brigade/Man-in-the-Middle Attack

Alice | Mr. X | Bob
---|---|---
Pick $S_A$ | Pick $S_X$ | Pick $S_B$

**Compute:**

$$T_A = g^{S_A} \mod p$$

$$T_X = g^{S_X} \mod p$$

$$T_B = g^{S_B} \mod p$$

$T_A >> T_X .. T_X$ >> $T_X$

$T_X << T_X .. T_B$ << $T_B$

**Compute:**

$$K_{AX} = T_X^{S_A} \mod p$$

$$K_{AX} = T_A^{S_X} \mod p$$

$$K_{BX} = T_X$$

$$K_{BX} = T_B^{S_X} \mod p$$

Possible Defense

- Each person $i$ picks $S_i$ and computes $T_i = g^{S_i} \mod p$ and keeps $S_i$ private and makes $T_i$ public.

- If Alice like to communicate with Bob, she finds $T_B$ and computes:

  $$K_{AB} = T_B^{S_A} \mod p$$

  Then tells Bob she likes to communicate with him.

- Bob finds $T_A$ and hen computes:

  $$K_{BA} = T_A^{S_B} \mod p$$

  This requires PKI (public Key Infrastructure) to manage $T_i$. 
ElGamal Signature

- Each person has long-term public/private key pair:
  - Private: $S$
  - Public: $<g, p, T>$ where $T = g^S \mod p$

- For each message $m$ to be signed:
  - Generate a per-message public/private key pair:
    - Private: $S_m$
    - Public: $T_m = g^{S_m} \mod p$
  - Compute the message digest: $d_m = MD(m | T_m)$
  - Compute the message signature: $X = S_m + d_m \cdot S \mod (p - 1)$
  - Send: $m, T_m$ and $X$

Verify:

- Compute the message digest: $d_m = MD(m | T_m)$
- Compute: $Y_1 = g^X$ and $Y_2 = T_m \cdot T_d^m$
  and If $Y_1 = Y_2$ then the signature is correct.

Why?

$Y_1 = g^X = g^{S_m} \cdot T_d^m = g^{S_m} \cdot g^{d_m} = T_m \cdot g^{S \cdot d_m} = T_m \cdot T_d^m = Y_2$
**Digital Signature Standard (DSS)**

Proposed by NIST based on a modified version of ElGamal algorithm.

---

**Zero Knowledge Proof Systems**

**Example:**

**Graph isomorphism problem:**

We consider two graphs *isomorphic* if we can *rename* the vertices of one to get a graph *identical* to the other.

This is a well known NP-complete problem.

![Graph Diagram]
Alice specifies a large graph $G_A$ and renames the vertices to produce another isomorphic graph $G_B$.

Public Key: $(G_A, G_B)$

Private Key: $G_A \leftrightarrow G_B$

To prove to Bob that she is Alice:

- She renames the vertices to produce a set of isomorphic graphs: $G_1, G_2, \ldots, G_k$ and sends them to Bob.

- Bob asks Alice to show him for each $i$ the mapping between: $G_i$ and either $G_A$ or $G_B$ but not both, otherwise Bob may know her private key!