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

The thing that 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:

```
+  0  1  2  3  4  5  6  7  8  9
0  0  1  2  3  4  5  6  7  8  9
1  1  2  3  4  5  6  7  8  9  0
2  2  3  4  5  6  7  8  9  0  1
3  3  4  5  6  7  8  9  0  1  2
4  4  5  6  7  8  9  0  1  2  3
5  5  6  7  8  9  0  1  2  3  4
6  6  7  8  9  0  1  2  3  4  5
7  7  8  9  0  1  2  3  4  5  6
8  8  9  0  1  2  3  4  5  6  7
9  9  0  1  2  3  4  5  6  7  8
```

**Figure 6-1.** Addition Modulo 10
Addition mod 10 can be used for encryption of digits: add \( k \) (between 1-9) to each digit, \( k \) is the secret key.

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

Decryption is done by adding the \(-k\), 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 \( 7 + 3 = 0 \) and thus 8654 will be 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.

**Multiplication:**

**Example:** multiplication mod 10
\[
8 \times 8 = 4, \quad 1 \times 9 = 9, \quad 7 \times 6 = 2
\]

See Fig. 6-2 for multiplication mod 10 Table:
Multiplication by 1, 3, 7 and 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 \( k \). A multiplicative inverse of \( k \) 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 \( k^{-1} \). Only the numbers \( \{1,3,7,9\} \) have multiplicative inverse mod 10.

**Euclid's Algorithm:** efficiently find multiplicative inverses mod \( n \). Given \( x \) and \( n \), it finds a number \( y \) such that \( x \cdot y \mod n = 1 \) (if there is such \( y \)).

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 \( \{1,2, \ldots, n-1\} \) are all relatively prime and thus \( \varphi(n) = n-1 \).

- If \( n = p \cdot q \) where \( p \) and \( q \) are two distinct primes,

  then \( \varphi(n) = (p-1)(q-1) \).

  **Example:** for \( n = 10 = 2 \cdot 5 \), \( \varphi(10) = (2-1)(5-1) = 4 \cdot 4 = 16 \), which is the set \( \{1,3,7,9\} \).

- **Exponentation:**
Example: exponentiation mod 10

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

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

![Table](image)

Amazing fact about \(\varphi(n)\):

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

Fig. 6-3: since \(\varphi(10)=4\), the \(i^{th}\) column is the same as the \((i+4)^{th}\) column, e.g., 3rd = 7th = 11th & 1st = 5th = 9th = 11th & 2nd = 6th = 10th.

Special case: if \(m = 1 \mod \varphi(n)\), then for any number \(x\),

\[x^m \mod n = x \mod n\]

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

\[m = 5 = 1 \mod 4: \ x = 3: \ 3^5 \mod 10 = 3 \& \ x = 6: \ 6^5 \mod 10 = 6 \& \text{in general: } x^5 \mod 10 = x \mod 10\]

\[m = 9 = 1 \mod 4: \ x = 3: \ 3^9 \mod 10 = 3 \& \ x = 6: \ 6^9 \mod 10 = 6 \& \text{in general: } x^9 \mod 10 = x \mod 10\]
An exponentiative inverse of $e$ is the number $d$ such that:

$$e.d = 1 \mod \phi(n)$$

Example: For $n=10$, $\phi(10)=4$:

$e=3$ and $d=7$ are exponentiative inverses since $3.7=21=1 \mod 4$

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 value is 512 bits.
Block size: plain text is variable less than key length & cipher text length equals key length.

Thus RSA is used for encrypting small amount of data like secret key & then we use 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 \( n = p \cdot q \) & \( \varnothing(n) = (p-1)(q-1) \).
   (it is very hard to factor \( n \) into \( p \) & \( q \)).
3. choose a number \( e \) that is relatively prime to \( \varnothing(n) \).
4. find a number \( d \) that is the multiplicative inverse of \( e \mod \varnothing(n) \), i.e., \( e \cdot d = 1 \mod \varnothing(n) \).
5. your public key: \(<e,n>\) & private key: \(<d,n>\).

encrypt/decrypt:
To encrypt a message \( m < n \):
\[
c = m^e \mod n
\]
& To decrypt \( c \):
\[
m = c^d \mod n
\]
This works since:
\[
c^d \mod n = (m^e)^d \mod n
\]
\[
= m^e^d \mod n
\]
\[
= m \mod n \quad \text{// since } e \cdot d = 1 \mod \varnothing(n)
\]
\[
= m \quad \text{// since } m < n
\]

sign/verify:
To sign a message m (<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:**

Every one 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 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 will take 30,000 MIPS-years!)

**Efficiency of RSA Operations:**

**Exponentiation**

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

\[ 123^2 = 123.123 = 15129 = 213 \mod 678 \]
\[ 123^3 = 123.213 = 26199 = 435 \mod 678 \]
\[ 123^4 = 123.435 = 53505 = 621 \mod 678 \]

......
\[ 123^{54} = \ldots \ldots \ldots = 87 \mod 678 \]

This requires 54 small number **multiplications** and 54 small number **divisions**.

How to compute \(123^{32} \mod 678\)?

\[ 123^2 = 123.123 = 15129 = 213 \mod 678 \]
\[ 123^4 = 213.213 = 45369 = 621 \mod 678 \]
\[123^8 = 621.621 = 385641 = 537 \mod 678\]
\[123^{16} = 537.537 = 288369 = 219 \mod 678\]
\[123^{32} = 219.219 = 47961 = 501 \mod 678\]

This requires 5 multiplications and 5 divisions.

To efficiently compute \(123^{54}\): 54 is represented in binary as:

\[
\begin{align*}
1 & \quad 1 & \quad 0 & \quad 1 & \quad 1 & \quad 0 \\
| & \quad | & \quad | & \quad | & \quad | \\
((( & (123^2)123 & )^2 & )^2123 & )^2123 & )^2
\end{align*}
\]

This requires 8 multiplications and 8 divisions.

Each 1 requires two multiplications and two divisions and each 0 requires one multiplication and one division. Thus in the above we have three 1s and two 0s that yields 3.2 + 2.1 = 8 (we ignore the leading 1).

Another example: \(y^{14}\). 14 is represented in binary as:

\[
\begin{align*}
1 & \quad 1 & \quad 1 & \quad 0 \\
| & \quad | & \quad | & \quad | \\
(( & (y^2)y & )^2y & )^2
\end{align*}
\]

This requires 5 multiplication's and 5 divisions.

**Generating RSA Keys**

- **Finding big prime number:**

  Choose a random number \(n\) and test if it is a prime. (for a 100 digit number the chances of success is 1 in 230).

  **test:** pick \(a < n\)

  \[\text{if } (a^{n-1} \mod n) \neq 1\]
then
n is not prime
else
n is prime   // the probability of being wrong is 1 in 10^{13}

• Finding d, n & e:

Finding e:

Instead of selecting p, q and then e (see RSA algorithm), we will select e first then p and q.

Two popular values for e are: 3 and 65537 (2^{16} + 1).
These makes 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 (binary value of e is 1.150s.1).

Finding n:

If e = 3:
how to choose p & q so that 3 is relatively prime to \( \varphi(n) = (p-1)(q-1) \)?
Choose random numbers x and y then:
- \( p = (2x+1)*3+2 \)
- \( q = (2y+1)*3+2 \)

Why?
To ensure that (p-1) relatively prime to 3 we choose
(p-1) = 1 mod 3 or  p = 2 mod 3.
Hence choose p as k*3+2 and to make sure that p is odd let k=2x+1.
Thus p = (2x+1)*3 + 2.
If e = 65537 randomly choose p and q and 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 \( n = p.q \) and \( \varphi(n) = (p-1)(q-1) \).

Finding d:
How to find $d$ such that $e\cdot d = 1 \mod \varphi(n)$?

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 $T_A = (g^{S_A}) \mod p$  
$(g^{S_B}) \mod p$

**Bob**

Pick $S_B$ (512-bit random number)
Compute $T_B =$

$T_A \quad \Rightarrow \quad \leftarrow \quad T_B$

Compute $X = T_B^{S_A} \mod p$

$X$ is the same as $Y$! why?

$$X = T_B^{S_A} = g^{S_A}$$
$$Y = T_A^{S_B} = g^{S_A}$$

No one can compute $g^{(S_A S_B)}$ by knowing $g^{\text{T} A}$ & $g^{\text{T} B}$

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

**Alice**

Pick $S_A$

**Mr. X**

Pick $S_X$

**Bob**

Pick $S_B$
Compute: \[ T_A = g^A \mod p \quad T_X = g^X \mod p \quad T_B = g^B \mod p \]

\[ T_A \gg T_A \cdot T_X \gg T_X \]
\[ T_X \ll T_X \cdot T_B \ll T_B \]

Compute: \[ K_{AX} = T_X^A \mod p \quad K_{AX} = T_A^X \mod p \quad K_{BX} = T_X \]
\[ K_{BX} = T_B^X \mod p \]

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:\n\[ K_{AB} = T_B^A \mod p \]
Then tells Bob she likes to communicate with him.
Bob finds \( T_A \) and then computes:\n\[ K_{BA} = T_A^B \mod p \]

This requires PKI (public Key Infrastructure) to manage \( T_i \)

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.
Alice specify a large graph $G_A$ and rename the vertices to produce another isomorphic graph $G_B$.

**Public Key:** $(G_A, G_B)$

**Private Key:** $G_A <--- G_B$

To prove to Bob that she is Alice:

she renames the vertices to produce a set of isomorphic graphs:

$$G_1, G_2, ..., G_k$$

and send them to Bob.

Bob asks Alice for each $i$ to show him the mapping between $G_i$ and either $G_A$ or $G_B$

(not both, otherwise he knows her private key!)