UNIT-1

Syllabus: Security Goals, Cryptographic Attacks, Services and Mechanisms, Mathematics of Cryptography

INTRODUCTION TO CRYPTOGRAPHY:

- An original message is known as the **plaintext**, while the coded message is called the **ciphertext**.
- The process of converting from plaintext to ciphertext is known as **enciphering or encryption;** restoring the plaintext from the ciphertext is **deciphering or decryption**.
- The many schemes used for encryption constitute the area of study known as **cryptography.** Such a scheme is known as a cryptographic system or a cipher.
- Techniques used for deciphering a message without any knowledge of the enciphering details fall into the area of **cryptanalysis.**
- Cryptanalysis is what the layperson calls "breaking the code."The areas of cryptography and cryptanalysis together are called **cryptology.**

A symmetric encryption scheme has five ingredients:

Plaintext: This is the original intelligible message or data that is fed into the algorithm as input.

Encryption algorithm: The encryption algorithm performs various substitutions and transformations on the plaintext.

Secret key: The secret key is also input to the encryption algorithm.The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time.The exact substitutions and transformations performed by the algorithm depend on the key.

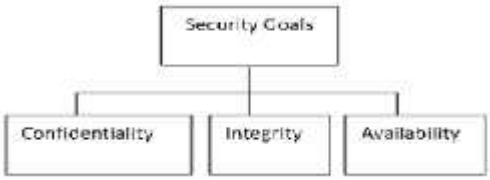
Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the secret key.

Decryption algorithm: This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

TOPIC 01:

❖ SECURITY GOALS:

let us first discuss three security goals: confidentiality, integrity, and availability



Confidentiality:

- It is the most common aspect of information security. we need to protect our confidential information.

- An organization needs to guard against those malicious actions that endanger the confidentiality of its information.
- In the miltrary, concealment of sensitive information is the major concern.
- In industry, hiding some information from competitors is crucial to the operation of the organization.
- In banking, customers accounts need to be kept secret.
- Confidentiality not only applies to the storage of the information, is also applies to the transmission of information.

Integrity:

- Information needs to be changed constantly. In a bank, when a customer deposits or withdraws money, the balance of her account needs to be changed.
- Integrity means that changes need to be done only by authorized entities and through authorized mechanisms.
- Integrity violation is not necessarily the result of a malicious act.

Availability:

- The third component of information security is availability.
- The information created and stored by an organization needs to be available to authorized entities. Information is useless, if it is not available.
- The unavailability of information is just as harmful for an organization as the lack of confidentiality or integrity.

TOPIC 02:

❖ **<u>Cryptographic attacks:</u>**

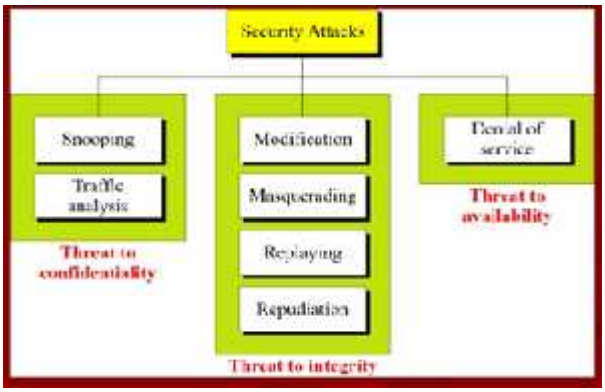
Cryptographic attacks can be broadly categorized into two distinct types: 1.Cryptanalytic and 2.Non-cryptanalytic.

➢ **Cryptanalytic attacks:** These attacks are combination of statistical and algebraic techniques aimed at ascertaining the secret key of a cipher.
  - These methods inspect the mathematical properties of the cryptographic algorithms and aims at finding distinguishers of the output distribution of cryptographic algorithms form uniform distributions.
  - The objective of cryptanalysis is to find properties of the cipher which does not exist in a random function.
  - Here distinguishers means that all attacks are fundamentally distinguishers.The attacker thus guesses the key and looks for the distinguishing property.If the property is detected,the guess is correct otherwise the next guess is tried.
  - The guessing complexity is lesser than the brute force search complexity.

➢ **Non-cryptanalytic attacks:**

- The other types of attacks are non-cryptanalytic attacks, which do not exploit the mathematical weakness of the cryptographic algorithm.

The three goals of security---confidentiality,integrity,and availability---can be threatened by security attacks.



- **Attacks threatening confidentiality**: In general , two types of attacks threaten the confidentiality of information: snooping and traffic analysis.

Snooping:

- It refers to unauthorized access to or interception of data.For example,a file transferred through the internet may contain confidential information.
- An unauthorized entity may interrupt the transmission and use the contents for her own benefit.
- To prevent snooping, the data can be made nonintelligible to the intercepter by using encipherment techniques.

Traffic analysis:

- Although encipherment of data may it non intelligible for the intercepter, she can obtain some other type information by monitoring online traffic.
- For example, she can find the electronic address of the sender or the receiver.

- **Attacks threatening integrity:** The integrity of data can be threatened by several kinds of attacks: modification, ma**sq**uerading, replaying and repudiation.

Modification:

- After intercepting or accessing information, the attacker modifies the information to make it beneficial to herself.
- For example, a customer sends a message to a bank to do some transaction. The attacker intercepts the message and changes the type of transaction to benefit herself.

Masquerading:

- It happens when the attacker impersonates somebody else.
- For example, an attacker might steal the bank card PIN of a bank customer and pretend that she is that customer.

- For example, a user tries to contact a bank, but another site pretends that it is the bank and obtains some information from the user.

Replaying:

- The attacker obtains a copy of a message sent by a user and later tries to replay it.

- For example , a person sends a request to her bank to ask for payment to the attacker, who has done a job for her. The attacker intercepts the message and sends it again to receive another payment from the bank.

Repudiation:

- This type of attack is different from others because it is performed by one of the two parties in the communication:sender and the receiver.
- The sender of the message might later deny that she has sent the message; the receiver of the message might later deny that she has received the message.

➢ **Attacks threatening Availability:** Only one this attack is **denial of service**.

Denial of service:

- It is very common attack.It may slow down or totally interrupt the service of a system.
- The sender sends so many bogus requests to a server that the server crashes because of heavy load.

➢ **Passive versus active attacks:**

active attacks

Attacks	Passive/Active	Threatening
Snooping Traffic Analysis	Passive	Confidentiality
Modification Masquerading Replaying Repudiation	Active	Integrity
Denial of Service	Active	Availability

Passive attacks:

- The attackers goal is just to obtain information.This means that the attack does not modify data or harm the system.
- The system continues with its normal operation.The attack may harm the sender or the receiver of the message.
- Attacks that threaten confidentiality--snooping and traffic analysis ---are passive attacks.

Active atacks:

- An active attack may change the data or harm the system.
- Attacks that threaten the integrity and availability are active attacks.

- These attacks are normally easier to detect than to prevent because an attacker can launch them in a variety of ways.

TOPIC 03:

❖ **Security services and mechanisms:**

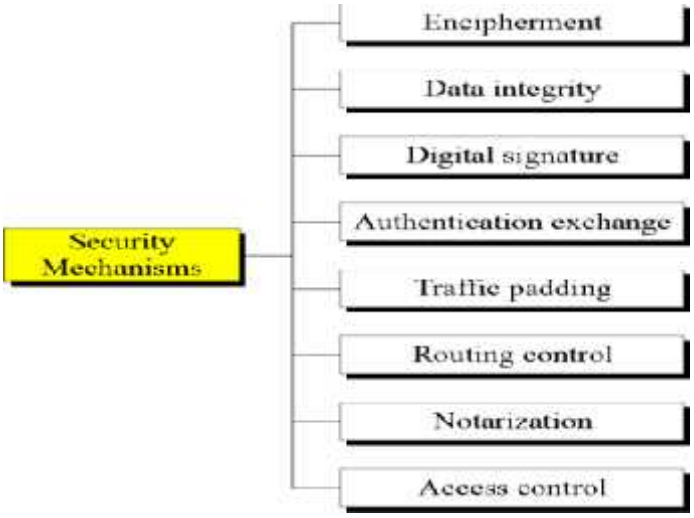
ITU-T (International Telecommunication Union-Telecommunication Standardization Sector) provides some security services and some mechanisms to implement those services.Security services and

mechanisms are closely related because a mechanism or combination of mechanisms are used to provide a service.

➢ **Security services:**

- **Authentication:** In connection oriented communication,it provides authentication of the sender and receiver during the connection establishment.In connection-less communication, it authenticates the source of the data.
- **Access Control:** It provides protection against unauthorized access to data.
- **Data Confidentiality:** information is not made available to unauthorized individual.It is designed to prevent snooping and traffic analysis attack.
- **Data Integrity**: It is designed to protect data from modification,insertion,deletion,and replaying by an adversary, It may protect the whole message or part of the message.
- **Non-Repudiation:** protection against denial of sending or receiving in the communication.

➢ **Security mechanisms:**



Encipherment :

- Encipher ,hiding or covering data,can provide confidentiality.
- It can also be used to complement other mechanisms to provide other services.
- we use two techniques---cryptography and stegnography

Data integrity:

- The data integrity mechanism appends to the data a short checkvalue that has been created by a specific process from the data itself.
- The receiver receives the data and checks value.
- He creates a new checkvalue from the received data and compares the newly created checkvalue with the one received.
- If two checkvalues are same,the integrity of data has been preserved.

Digital signature:

- A digital signature is a means by which the sender can electronically sign the data and receiver can electronically verify the signature.
- The sender uses a process that involves showing that she owns a private key related to the public key that she has announced publicly .
- The receiver uses the sender's public key to prove that the message is indeed signed by the sender who claims to have sent the message.

Authentication exchange:

- In this two entities exchange some messages to prove their identity to each other.
- For example, one entity can prove that she knows a secret that only she is supposed to know

Traffic Padding:

- This means inserting some bogus data into the data traffic to the adversary's attempt to use the traffic analysis.

Routing control:

- It means selecting and continuously changing different available routes between sender and receiver to prevent the opponent from eavesdropping on a particular route.

Notarization:

- It means selecting a third trusted party to control the communication between two entities.
- This can be done, for example,to prevent repudiation.

Access control:

- It uses methods to prove that a user has access right to the data or resources owned by a system.
- Examples of proofs are passwords and PINs

➢ **Relation between Services and Mechanisms:**

Security Service	*Security Mechanism*
Data confidentiality	Encipherment and routing control
Data integrity	Encipherment, digital signature, data integrity
Authentication	Encipherment, digital signature, authentication exchanges
Nonrepudiation	Digital signature, data integrity, and notarization
Access control	Access control mechanism

❖ **<u>Mathematics of Cryptography:</u>**

Cryptography is based on some specific areas of mathematics, including number theory, linear algebra and algebraic structures.

Integer arithmetic: In integer arithmetic, we use a set and few operations.

Set of Integers: The set of integers, denoted by Z, contains all integral numbers(with no fraction)from negative infinity to positive infinity.

z={.....,-2,-1,0,1,2.....}

Binary operations: In cryptography , we are interested in three basic operations applied to the set of integers. A binary operation takes two inputs and creates one output.

- Three basic operations are addition, subtraction and multiplication. Each of these operations takes 2 inputs and creates 1 output.
- The two inputs come from the set of integers; the output goes into the set of integers.

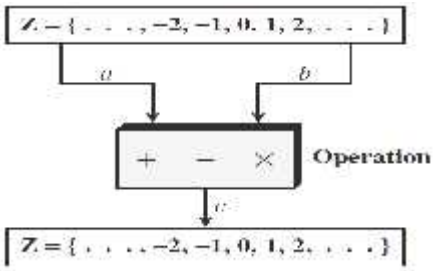


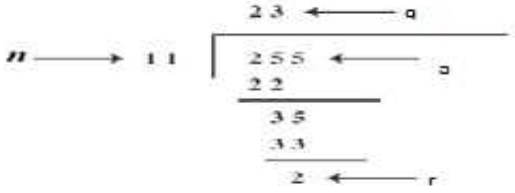
fig: Three binary operations for the set of integers

- **Integer Division**:

  In integer arithmetic, if we divide a by n, we get q and r. The relationship between these four integers can be shown as

  $$a=q\times n+r$$

  In this relation , a is called the dividend; q, the quotient; the divisor; and r, the remainder.



  Two Restrictions: For our purpose,we impose two restrictions.First,we require that the divisor be a positive integer($n>0$). Second,we require that the remainder be a non-negative integer($r \geq 0$).

- **Divisibility:** If a is not zero and we let r=0 in the division relation, we get

  $$a=q\times n$$

  we say that n divides a.we can also say that a is divisible by n.when we are not interested in the value of q,we can write the above relationship as a|n.If the remainder is not zero,then n does not divide a and we can write the relationship as a×n.

Properties:

- If a|1, then a = ±1.
- If a|b and b|a, then a = ±b.
- If a | b and b | c, then a | c
- If a|b and a|c, then a|(m×b + n×c) where m and n are arbitrary integers.

Example:

a. Since 3|15 and 15|45,according to third property, 3|45

b. Since 3|15 and 3|9 , according to fourth property, 3|(15×2+9×4),which means 3|66

- **Greatest Common Divisor:** One integer often needed in cryptography is the greatest common divisor of two positive integers. Two positive integers may have many common divisors, but only one greatest common divisor.

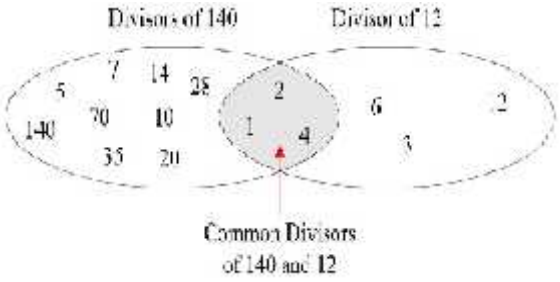


fig:Common divisors of two integers

Note: The greatest common divisor of two positive integers is the largest integer that can divide both integers

- **Euclidean Algorithm:** Finding the greatest common divisor (gcd) of two positive integers by listing all common divisors is not practical when two integers are large.

Fact 1: gcd(a,0)=a Fact 2: gcd(a,b)=gcd(b,r),where r is the remainder of dividing a by b

The first fact tells us that if the second integer is 0, the greatest common divisor is the first one.The second fact allows us to change the value of a,b until b becomes 0.

gcd(36,10)=gcd(10,6)=gcd(6,4)=gcd(4,2)=gcd(2,0)=2

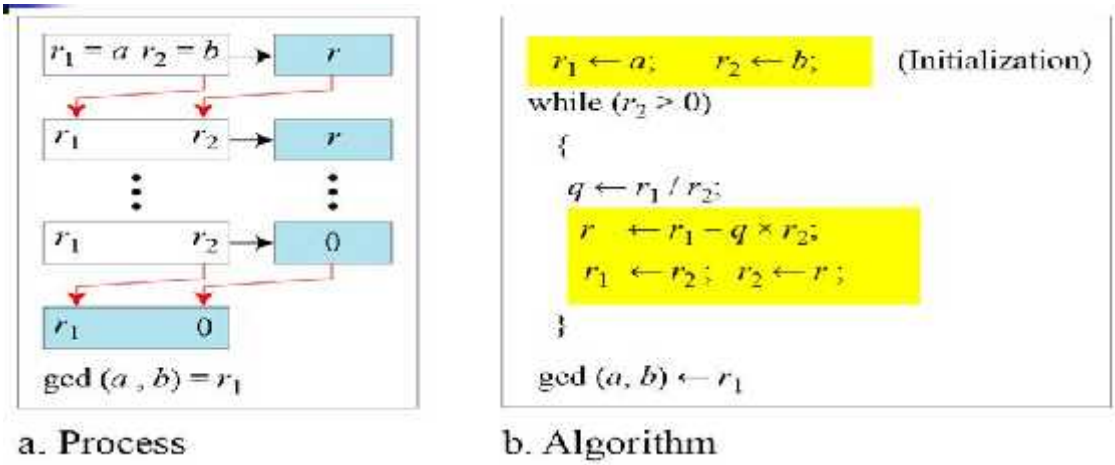


fig: 2.7 Euclidean algorithm

- we use two variables r1 and r2, to hold the changing values during the process of reduction.They are initialized to a and b.
- In each step, we calculate the remainder of r1 divided by r2 and store the result in the variable r.we then replace r1 by r2 and r2 by r.

- The steps are continued until r2 becomes 0.At this moment,we stop. The gcd(a,b) is r1.

  when gcd(a,b) = 1, we say that a and b are relatively prime

  Example : Find the greatest common divisor of 2740,1760

  sol:

We have gcd (2740, 1760) = 20

q	r_1	r_2	r
1	2740	1760	980
1	1760	980	780
1	980	780	200
3	780	200	180
1	200	180	20
9	180	20	0
	20	0	

- **The Extended Euclidean Algorithm:**

Given two integers a and b,we often need to find other two integers, s and t, such that

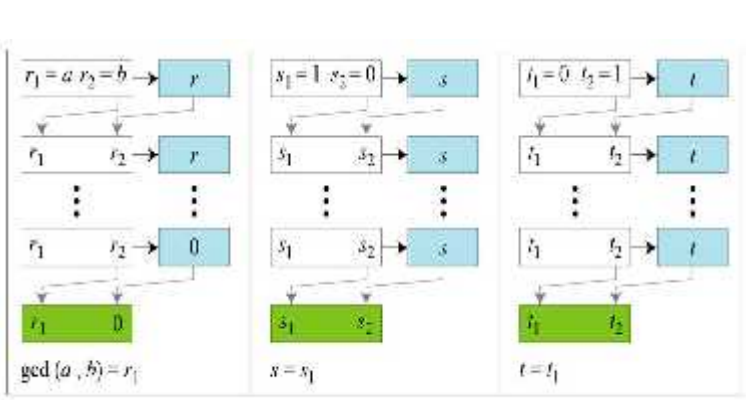
$$s \times a + t \times b = \gcd(a,b)$$

The Extended euclidean algorithm can calculate the gcd(a,b) and at the same time calculate the value of s and t. The algorithm and the process is shown below diagram.

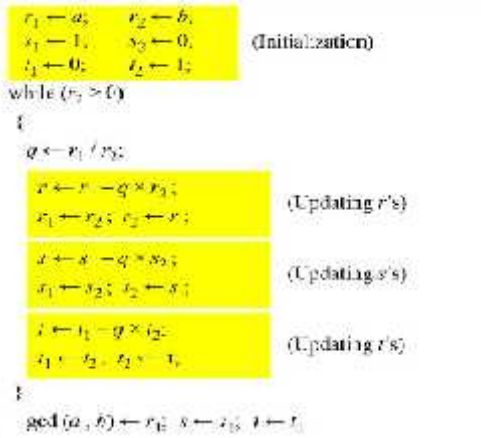
- The extended euclidean algorithm uses the same number of steps as the Euclidean algorithm. However in each step , we use three sets of calculations and exchange instead of one.
- The algorithm uses three sets of variables, r's, s's and t's.
- In each step r_1,r_2 and r have the same values in the Euclidean algorithm.
- The variables r_1 and r_2 are initialized to the values of a and b respectively.
- The variables s_1 and s_2 are initialized to 1 and 0 respectively.

- The variables t_1 and t_2 are initialized to 1 and 0 respectively.
- The calculations of r, s and t are similar, with one warning.

Although r is the remainder of dividing r_1 and r_2, there is no such relationship between the other two sets. There is only one quotient, q, which is calculated $r_1|r_2$ and used for the other two calculations.



a. Process



b.Algorithm

Given $a = 161$ and $b = 28$, find gcd (a, b) and the values of s and t.

Solution

We get gcd $(161, 28) = 7$, $s = -1$ and $t = 6$.

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
5	161	28	21	1	0	1	0	1	−5
1	28	21	7	0	1	−1	1	5	6
3	21	7	0	1	−1	4	−5	6	−23
	7	0		−1	4		6	−23	

$r=r_1-q\times r_2$ $s=s_1-q\times s_2$ $t=t_1-q\times t_2$

- **Linear Diophantine Equations:**

Although we will see a very important application of the extended Euclidean algorithm. One immediate applications is to find the solutions to the linear Diophantine equations of two variables, an equation of type ax+by+c.we need to find integer values for x and y that satisfy the equation.This type of equation has either no solution or an infinite number of solutions.

Let d= gcd(a,b) , If d+c, then the equation has no solution.

If d|c , then we have an infinite number of solutions. one of them is called the particular; the rest, general

A linear Diophantine equation of two variables is ax+by=c.

- **MODULAR ARITHMETIC:**

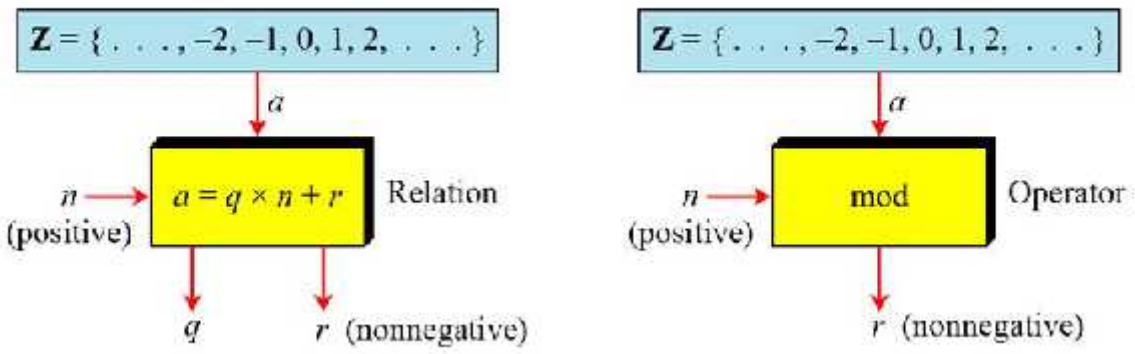
- The division relationship (a=q × n+r) has two inputs (a and n) and two outputs (q and r).
- In modular arithmetic , we are intereted in only one of the outputs, the remainder r.we don't care about the quotient q.
- In other words , we want to know what is the value of r when we divide a by n.
- This implies that we can change the above relation into a binary operator with two inputs a and n and one output r.

Modulo Operator:

- The above mentioned binary operator is called the **modulo operator** and is shown as **mod.**
- The second input (n) is called the modulus. The output r is called the residue.
- The below figure shows , the modulo operator (mod) takes an integer (a) from the set z and a positive modulus (n) .The operator creates a nonnegative residue (r) .we can say

a mod n = r

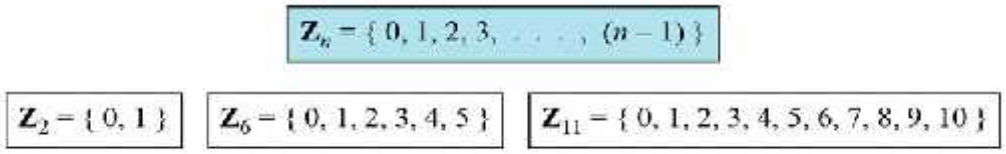
Figure 2.9 *Division algorithm and modulo operator*



Set of Residues: $\mathbf{Z_n}$

- The result of the modulo operation with modulus n is always an integer between 0 and n-1.
- In other words, the result of a mod n is always a nonnegative integer less than n.
- we can say that the modulo operation creates a set, which in modular arithmetic is referred to as the set of least residues modulo n, or Z_n.
- We have infinite instances of the set of residues (Z_n),one for each value of n.
- The below figure shows the set Z_n and three instances, Z_2,Z_6, and Z_{11}.

Figure 2.10 *Some Z_n sets*



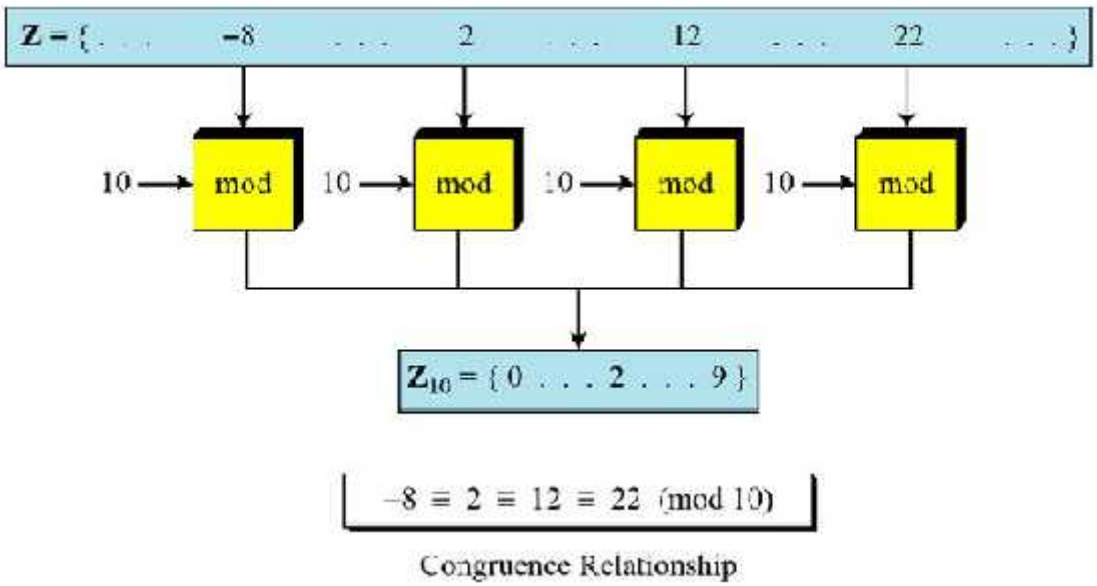
Congruence:

- In Cryptography, we often used the concept of congruence instead of equality.
- Mapping from Z to Z_n is not one-to-one.
- For example, the result of 2 mod 10 = 2,12 mod 10 = 2,22 mod 2= 2,and so on.
- In Modular arithmetic , integers like 2,12, and 22 are called congruent mod 10.
- To show that two integers congruent, we use the congruence operator (≡).
- We add the phrase (mod n) to the right side of the congruence to define the value of modulus that makes the relationship valid. For example ,we write:

$2 \equiv 12 \pmod{10}$	$13 \equiv 23 \pmod{10}$	$34 \equiv 24 \pmod{10}$	$-8 \equiv 12 \pmod{10}$
$3 \equiv 8 \pmod{5}$	$8 \equiv 13 \pmod{5}$	$23 \equiv 33 \pmod{5}$	$-8 \equiv 2 \pmod{5}$

we need to explain several points.

- The congruence operator looks like the equality operator, but there are differences. First, an equality operator maps a member of Z to itself; the congruence operator maps a member from Z to member of Zn. Second, the equality operator is one-to-one ; the congruence operator is many-to-one.
- The phrase (mod n) that we insert at the right-hand-side of the congruence operator is just an indication of the destination set (Zn).



Residue classes:

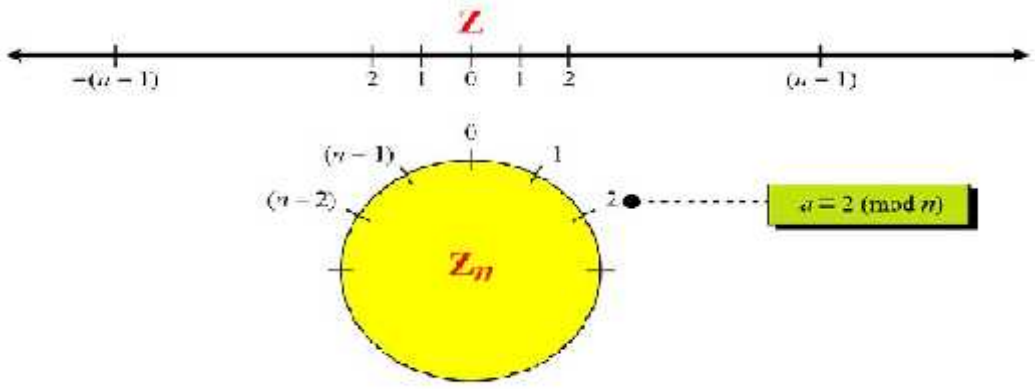
- A residue calss [a] or $[a]_n$ is the set of integers congruent modulo n. In other words, it is the set of all integers such that x = a (mod n). For example, if n=5, we have five sets [0],[1],[2],[3], and [4] as shown below:

$$[0] = \{\ldots, -15, -10, -5, 0, 5, 10, 15, \ldots\}$$
$$[1] = \{\ldots, -14, -9, -4, 1, 6, 11, 16, \ldots\}$$
$$[2] = \{\ldots, -13, -8, -3, 2, 7, 12, 17, \ldots\}$$
$$[3] = \{\ldots, -12, -7, -5, 3, 8, 13, 18, \ldots\}$$
$$[4] = \{\ldots, -11, -6, -1, 4, 9, 14, 19, \ldots\}$$

- The integers in the set [0] are all reduced to 0 when we apply the modulo 5 operation on them. The integers in the set [1] are all reduced to a when we apply the modulo 5 operation, and so on.
- In each set, there is one element called the least(non negative) residue.
- In the set [0], this element is 0; in the set [1], this element is 1; and so on.
- The set of all of these least residues is what we have shown as $Z_5 = \{0,1,2,3,4\}$.
- In other words , the set Z_n is the set of all least residue modulo n.

Circular Notation:

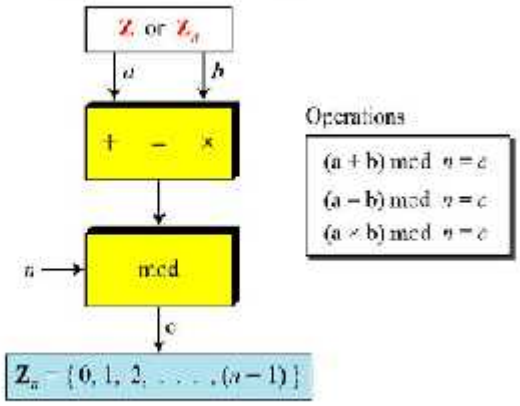
- The concept of congruence can be better understood with the use of a circle.
- we can use a circle to show the distribution of integers in Z_n.
- The below figure shows the comparison between the two. Integers 0 to n-1are spaced evenly around a circle.
- All congruent integers modulo n occupy the same point on the circle.
- Positive and negative integers from Z are mapped to the circle in such a way that there is a symmetry between them.



Operations in Z_n:

- The three binary operations(addition, subtraction and multiplication) that we discussed for the set Z can also be defined for the set Z_n.
- The result may need to be mapped to Z_n using the mod operator as shown

Figure 2.13 *Binary operations in Z_n*



- Actually the two sets of operators are used here.
- The first set is one of the binary operators (+,-,×); the second is the mod operator.
- we need to use paranthesis to emphasize the order of operations.

Perform the following operations (the inputs come from Zn):
a. Add 7 to 14 in Z15.
b. Subtract 11 from 7 in Z13.
c. Multiply 11 by 7 in Z20.

Solution The following shows the two steps involved in each case

$$(14 + 7) \bmod 15 \rightarrow (21) \bmod 15 = 6$$
$$(7 - 11) \bmod 13 \rightarrow (-4) \bmod 13 = 9$$
$$(7 \times 11) \bmod 20 \rightarrow (77) \bmod 20 = 17$$

Properties:

- we mentioned that the two inputs to three binary operations in the modular arithmetic can come from Z or Z_n.
- The following properties allow us to first map the two inputs to Z_n before applying the three basic operations (+,-,×).

First Property: $(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$
Second Property: $(a - b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$
Third Property: $(a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$

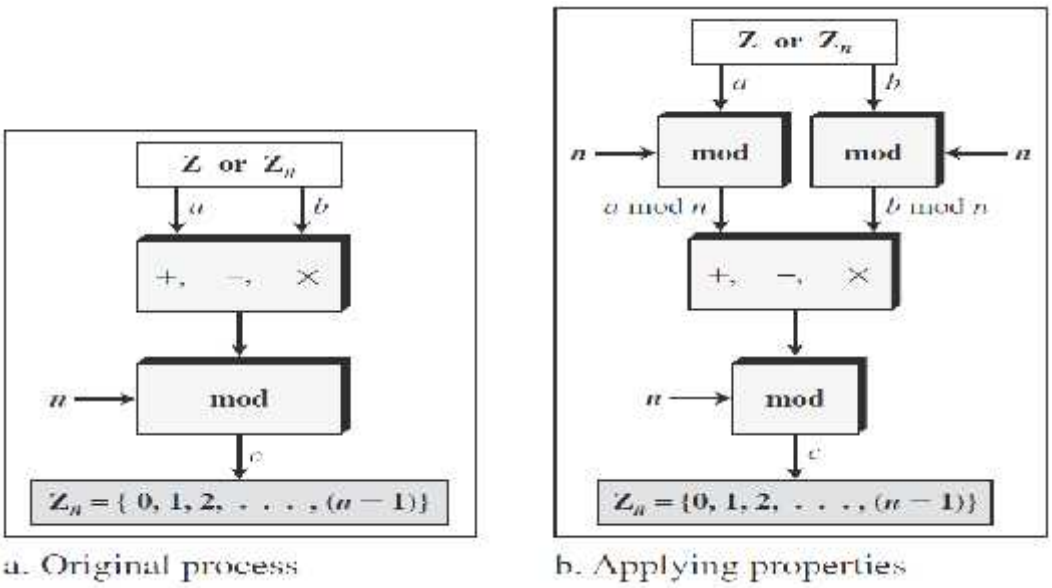


fig: Properties of mod operator

- The above figure shows the process before and after applying the above properties.
- Although the figure shows that the process is longer if we apply the above properties,we should remember that in cryptography we are dealing with very large integers.
- For example,if we multiply a very large integer by another very large integer , we have an integer that is too large to be stored in computer.
- The properties allow us to work with small numbers.

The following shows the application of the above properties:

1. (1,723,345 + 2,124,945) mod 11 = (8 + 9) mod 11 = 6
2. (1,723,345 − 2,124,945) mod 11 = (8 − 9) mod 11 = 10
3. (1,723,345 × 2,124,945) mod 11 = (8 × 9) mod 11 = 6

Inverses:

- when we are working in modular arithmetic, we often need to find the inverse of a number relative to an operation.
- we are normally looking for an **additive inverse** or a **multiplicative inverse.**

Additive inverse:

- In Z_n, two numbers a and b are additive inverses of each other if $a+b \equiv 0 \pmod n$
- In Z_n, the additive inverse of a can be calculated as b=n-a. For example , the additive inverse of 4 in Z_{10} is 10-4=6.

In modular arithmetic, each integer has an additive inverse. The sum of an integer and its additive inverse is congruent to 0 modulo n.

Note that in modular arithmetic, each number has an additive inverse and the inverse is unique; each number has one and only one additive inverse. However the inverse of the number may be the number itself.

Example 2.21 Find all additive inverse pairs in Z_{10}.

Solution The six pairs of additive inverses are (0, 0), (1, 9), (2, 8), (3, 7), (4, 6), and (5, 5). In this list, 0 is the additive inverse of itself; so is 5. Note that the additive inverses are reciprocal; if 4 is the additive inverse of 6, then 6 is also the additive inverse of 4.

Multiplicative Inverse:

- In Z_n, two numbers a and b are multiplicative inverses of each other if
$$a \times b \equiv 1 \pmod n$$
- For example, if the modulus is 10,then the multiplicative inverse of 3 is 7.In other words,we have $(3 \times 7) \bmod 10 = 1$.

In modular arithmetic, an integer may or may not have a multiplicative inverse. when it does, the product of the integer and its multiplicative inverse is congruent to 1 modulo n.

It Can be proved that a has a multiplicative inverse in Z_n if and only if gc(n,a)=1.In this case, a and n are said to be **relatively prime**.

Example 2.22

Find the multiplicative inverse of 8 in Z_{10}.

Solution

There is no multiplicative inverse because gcd (10, 8) = 2 ≠ 1. In other words, we cannot find any number between 0 and 9 such that when multiplied by 8, the result is congruent to 1.

Example 2.23

Find all multiplicative inverses in Z_{10}.

Solution

There are only three pairs: (1, 1), (3, 7) and (9, 9). The numbers 0, 2, 4, 5, 6, and 8 do not have a multiplicative inverse.

The integer a in Z_n has a multiplicative inverse if and only if gcd(n,a) ≡ 1(mod n)

The extended Euclidean algorithm we can find the multiplicative inverse of b in Zn when n and b are given and inverse exists. when n and b are given and the inverse exists.

To show this, let us replace the first integer a with n(the modulus). we can say that the algorithm can find s and t such $s \times n + b \times t = \gcd(n,b)$.

However, if the multiplicative inverse of b exists, gcd(n,b) must be 1. so the relationship is

$(s \times n) + (b \times t) = 1$

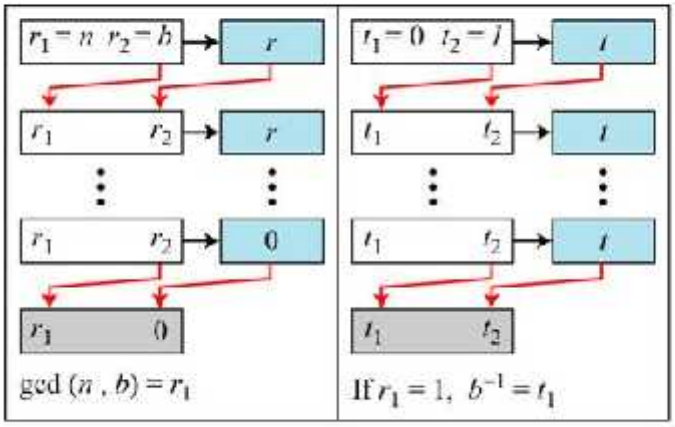
Now we apply the modulo operator to both sides. In other words, we map each side to Z_n. We will have

$$(s \times n + b \times t) \bmod n = 1 \bmod n$$
$$[(s \times n) \bmod n] + [(b \times t) \bmod n] = 1 \bmod n$$
$$0 + [(b \times t) \bmod n] = 1$$
$$(b \times t) \bmod n = 1 \quad \rightarrow \text{This means } t \text{ is the multiplicative inverse of } b \text{ in } Z_n$$

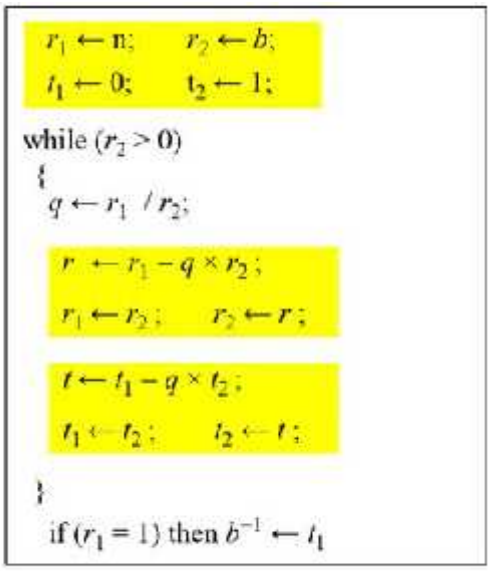
Note that $[(s \times n) \bmod n]$ in the third line is 0 because if we divide $(s \times n)$ by n, the quotient is s but the remainder is 0.

The extended Euclidean algorithm finds the multiplicative inverses of b in Z_n when n and b are given and gcd(n,b)= 1.The multiplicative inverse of b is the value of t after being mapped to Z_n.

fig: using the extended Euclidean algorithm to find the multiplicative inverse



a. Process



b. Algorithm

Example:

Find the multiplicative inverse of 11 in Z_{26}.

Solution

q	r_1	r_2	r	t_1	t_2	t
2	26	11	4	0	1	−2
2	11	4	3	1	−2	5
1	4	3	1	−2	5	−7
3	3	1	0	5	−7	26
	1	0		−7	26	

The gcd(26,11) is 1, which means that the multiplicative inverse of 11 exists. The extended Euclidean algorithm gives t_1 = -7. The multiplicative inverse is (-7)mod 26=19.In other words, 11 and 19 are multiplicative inverse in Z_{26}.we can see that (11 × 19)mod 26=209 mod 26=1.

Addition And Multiplication Tables:

- In addition table, each integer has an additive inverse. The inverse pairs can be found when the result of addition is zero.
- In multiplication table, we have only three multiplicative pairs(1,1),(3,7),(9,9).The pairs can be found whenever the result of multiplication is 1.
- Both tables are symmetric with respect to the diagonal of elements that moves from the top left to bottom right, revealing the commutative property for addition and multiplication (a+b=b+a and a × b =b × a).
- The addition table also shows that each row or column is a permutation of another row or column.This is not true for the multiplication table.

Addition and multiplication tables for Z_{10}

+	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

Addition Table in $\mathbf{Z_{10}}$

×	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	0	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

Multiplication Table in $\mathbf{Z_{10}}$

Different Sets For Addition And Multipliation:

- In cryptography, we often work with inverses.
- If the sender uses an integer, the receiver uses the inverse of that integer .
- If the operation is addition, Z_n can be used as the set of possible keys because each integer in this set has an additive inverse.
- If the operation is multiplication, Z_n cannot be the set of possible keys because only some members of this set have a multiplicative inverse.

We need to use Z_n when additive inverses are needed; we need to use Z_n^* when multiplicative inverses are needed.

fig: some Z_n and Z_n^* sets

$\mathbf{Z_6} = \{0, 1, 2, 3, 4, 5\}$

$\mathbf{Z_6^*} = \{1, 5\}$

$\mathbf{Z_7} = \{0, 1, 2, 3, 4, 5, 6\}$

$\mathbf{Z_7^*} = \{1, 2, 3, 4, 5, 6\}$

$\mathbf{Z_{10}} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$\mathbf{Z_{10}^*} = \{1, 3, 7, 9\}$

Two more sets:

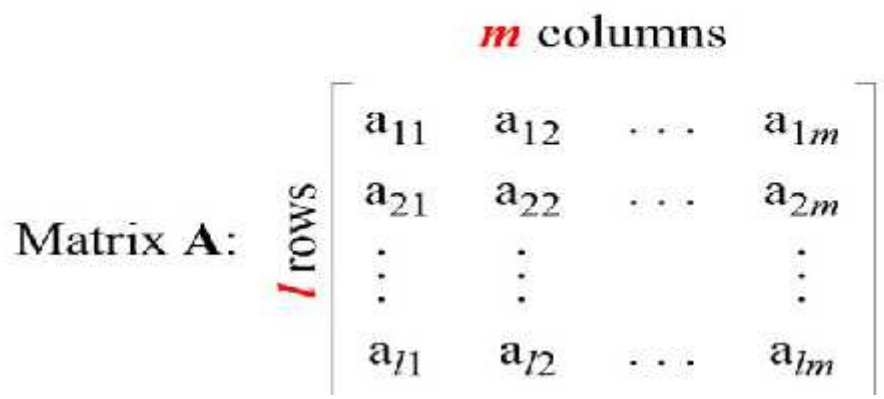
- The set Z_p is same as Z_n except that n is prime. Z_p contains all integers from 0 to p-1.Each member in Z_p has an additive inverse; each member except 0 has a multiplicative inverse.
- The set Z_p^* is same as Z_n^* except that n is prime. Z_p^* contains all integers from 1 to p-1.Each member in Z_p^* has an additive and multicative inverse.
- The following shows these two sets when p=13
  $Z_{13} = \{0,1,2,3,4,5,6,7,8,9,10,11,12\}$
  $Z_{13}^* = \{1,2,3,4,5,6,7,8,9,10,11,12\}$

➢ **MATRICES:**

Def: A matrix is a rectangular array of $l \times m$ elements, in which l is the number of rows and m is the number of columns.

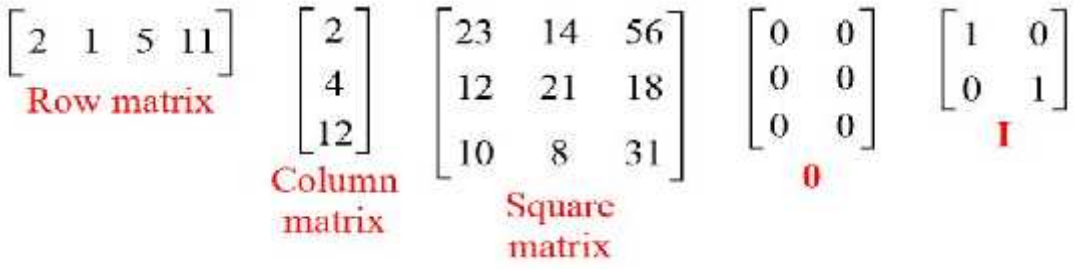
A matrix is normally denoted with a boldface uppercase letter such as **A.** The element a_{ij} is located in the ith row and jth column.Although the elements can be a set of numbers.

Figure 2.18 *A matrix of size l x m*



- If a matrix has only one row(l=1),it is called a row matrix; if it has only one column (m=1) it is called column matrix.
- In a square matrix, in which there is the same number of rows and columns (l=m)
- An additive identity matrix, denoted as 0,is a matrix with all rows and columns set to 0's.
- An identity matrix, denoted as I, is a square matrix with 1sbon the main diagonal and 0s elsewhere.

Figure 2.19 *Examples of matrices*



Operations and Relations:

In linear algebra,one relation and four operations(addition,subtraction,multiplication and scalar multiplication) are defined for matrices.

- **Equality:** Two matrices are equal if they have the same number of rows and columns and the corresponding elements are equal. In other words, A=B if we have $a_{ij}=b_{ij}$ for all i'a and j's.
- **Addition and Subtraction:** Two matrices can be added if they have the same number of columns and rows. This addition is shown as C=A+B.
- In this case,the resulting matrix C has also the same number of rows and columns as A or B.
- Each element of C is the sum of two correponding elements of A and B: $C_{ij} = a_{ij} + b_{ij}$.
- Subtraction is the same except that each element of B is subtracted from the corresponding element of A:dij = a_{ij} - b_{ij}

Example 2.28

Figure 2.20 shows an example of addition and subtraction.

Figure 2.20 *Addition and subtraction of matrices*

$$\begin{bmatrix} 12 & 4 & 4 \\ 11 & 12 & 30 \end{bmatrix} = \begin{bmatrix} 5 & 2 & 1 \\ 3 & 2 & 10 \end{bmatrix} + \begin{bmatrix} 7 & 2 & 3 \\ 8 & 10 & 20 \end{bmatrix}$$

C = A + B

- **Multiplication:**
  we can multiply two matrices of different sizes if the number of columns of the first matrix is the same as the number of rows of the second matrix.
  if A is an l × m matrix and B is m × p matrix, the product of the two is a matrix C of size l × p.
  If each element of matrix A is called aij ,each element of matrix B is called bjk,then each element of matrix C , Cij, can be calculated as
  cik = ∑ aij × bij =ai1 × b ij + ai2 × b 2j +.....+ aim × b mj

Example:

shows the product of a row matrix (1 × 3) by a column matrix (3 × 1). The result is a matrix of size 1 × 1.

Figure 2.21 *Multiplication of a row matrix by a column matrix*

$$\underset{C}{\begin{bmatrix}53\end{bmatrix}} = \underset{A}{\begin{bmatrix}5 & 2 & 1\end{bmatrix}} \times \underset{B}{\begin{bmatrix}7\\8\\2\end{bmatrix}}$$

- Scalar Multiplication:

We can also multiply a matrix by a number(called a scalar).If A is an l × m matrix and x is scalar, C=xA is a matrix of size l × m,in which cij= x * aij.

$$\underset{B}{\begin{bmatrix}15 & 6 & 3\\9 & 6 & 12\end{bmatrix}} = 3 \times \underset{A}{\begin{bmatrix}5 & 2 & 1\\3 & 2 & 4\end{bmatrix}}$$

Determinant: The determinant of a square matrix A of size m x m denoted as det(A) is scalar calculated recursively as shown below:

1. If m=1 , $\det(A)=a_{11}$
2. If m>1, $\det(A) = \sum(-1)^{i+j} \times a_{ij} \times \det(A_{ij})$

where A_{ij} is a matrix obtained from A by deleting the i^{th} row and j^{th} column.

example: we can calcualte the deteminant of a 2X 2 matrix

$$\det\begin{bmatrix}5 & 2\\3 & 4\end{bmatrix} = (-1)^{1+1} \times 5 \times \det[4] + (-1)^{1+2} \times 2 \times \det[3] \longrightarrow 5 \times 4 - 2 \times 3 = 14$$

or $$\det\begin{bmatrix}a_{11} & a_{12}\\a_{21} & a_{22}\end{bmatrix} = a_{11} \times a_{22} - a_{12} \times a_{21}$$

Inver

Inverses:

Matrices have both additive and multiplicative inverses.

Additive inverse:

The additive inverse of matrix A is another matrix B such that A+ B = 0.In other words, we have $b_{ij} = -a_{ij}$ for all values of i and j. Normally the additive inverse of A is defined by -A.

Multiplicative inverse:

The multiplicative inverse is defined only for square matrices. The multiplicative inverse of a square matrix A is a square matrix B such that AxB=BxA=I.

Normally the multiplicative inverse of A is defined by A^{-1}.However , matrices with real elements have inverses only if det(A) ≠ 0.

Multiplicative inverses are only definded for square matrix

Residue Matrices:

Cryptography uses residue matrices: matrices with all elements are in Z_n . All operations on residue matrices are performed the same as for the integer matrices except that the operations are done in modular arithmetic. The residue matrix has a multiplicative inverse if gcd(det(A),n)=1.

Example

A residue matrix and its multiplicative inverse in Z_{26}

$$\mathbf{A} = \begin{bmatrix} 3 & 5 & 7 & 2 \\ 1 & 4 & 7 & 2 \\ 6 & 3 & 9 & 17 \\ 13 & 5 & 4 & 16 \end{bmatrix} \qquad \mathbf{A}^{-1} = \begin{bmatrix} 15 & 21 & 0 & 15 \\ 23 & 9 & 0 & 22 \\ 15 & 16 & 18 & 3 \\ 24 & 7 & 15 & 3 \end{bmatrix}$$

$$\det(\mathbf{A}) = 21 \qquad \det(\mathbf{A}^{-1}) = 5$$

Congruence: Two matrices are congruent modulo n, written as A≡ B(mod n) , if they jave the same number of rows and columns and all corresponding elements are congruent modulo n. In other words A≡ B(mod n) if $a_{ij} \equiv b_{ij}$ for all i's and j's.

- **Linear Congruence:**

Single Variable Linear Equations:

Let us see how we can solve equations involving a single variable—that is, equations of the form $ax \equiv b \pmod{n}$. An equation of this type might have no solution or a limited number of solutions. Assume that the gcd $(a, n) = d$. If d + b, there is no solution. If $d \mid b$, there are d solutions.

If $d \mid b$, we use the following strategy to find the solutions:

1. Reduce the equation by dividing both sides of the equation (including the modulus) by d.
2. Multiply both sides of the reduced equation by the multiplicative inverse of a to find the particular solution x_0.
3. The general solutions are $x = x_0 + k\,(n / d)$ for $k = 0, 1, \ldots, (d - 1)$.

Example 2.35 Solve the equation $10x \equiv 2 \pmod{15}$.

Solution First we find the gcd (10 and 15) = 5. Since 5 does not divide 2, we have no solution.

Example 2.36 Solve the equation $14x \equiv 12 \pmod{18}$.

Solution Note that gcd (14 and 18) = 2. Since 2 divides 12, we have exactly two solutions, but first we reduce the equation.

$$14x \equiv 12 \pmod{18} \rightarrow 7x \equiv 6 \pmod{9} \rightarrow x \equiv 6\,(7^{-1}) \pmod{9}$$
$$x_0 = (6 \times 7^{-1}) \bmod 9 = (6 \times 4) \pmod{9} = 6$$
$$x_1 = x_0 + 1 \times (18/2) = 15$$

Both solutions, 6 and 15 satisfy the congruence relation, because $(14 \times 6) \bmod 18 = 12$ and also $(14 \times 15) \bmod 18 = 12$.

Set of linear equations:

- we can also solve a set of linear equations with the same modulus if the matrix formed from the coefficients of the variables is invertible.
- we make three matrices. The first is the square matrix made from the coeffients of varibles
- The second is a column matrix made from the variables.
- The third is a column matrix made from the values at the right - hand side of the congruence operator.
- We can interpret the set of equations as matrix multiplication.
- If both sides of congruence are multiplied by the multiplicative inverse of the first matrix, the result is the variable matrix at the right hand side, which means the problem can be solved by a matrix multiplication as shown below

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\equiv b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\equiv b_2 \\
\vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad &\quad \vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &\equiv b_n
\end{aligned}$$

a. Equations

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \equiv \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \qquad \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \equiv \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}^{-1} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

b. Interpretation c. Solution

Fig. 2.27 *Set of linear equations*

Example 2.38 Solve the set of following three equations:

$3x + 5y + 7z \equiv 3 \pmod{16}$
$x + 4y + 13z \equiv 5 \pmod{16}$
$2x + 7y + 3z \equiv 4 \pmod{16}$

Solution Here x, y, and z play the roles of x_1, x_2, and x_3. The matrix formed by the set of equations is invertible. We find the multiplicative inverse of the matrix and multiply it by the column matrix formed from 3, 5, and 4. The result is $x \equiv 15 \pmod{16}$, $y \equiv 4 \pmod{16}$, and $z \equiv 14 \pmod{16}$. We can check the answer by inserting these values into the equations.

UNIT- II:

Syllabus:

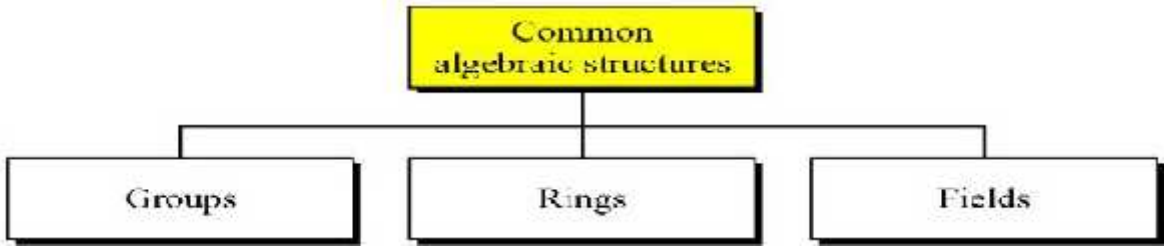
Symmetric Encryption: Mathematics of Symmetric Key Cryptography, Introduction to Modern Symmetric Key Ciphers, Data Encryption Standard, Advanced Encryption Standard.

TOPIC 01:

Algebraic structures:

In previous chapter we discuss some set of numbers,such as Z,Z_n, Z_n^*, Zp, Zp^*.

- Cryptography requires sets of integers and specific operations that are defined for those sets.
- The combination of the set and the operations that are applied to that elements of the set is called an algebraic structure.
- In this we will define three common algebraic structures: groups,rings, and fields.



- **GROUPS:**

A Group (G) is a set of elements with a binary operation "•" that satisfies four properties(or axioms). A commutative group , also called an abelian group,is a group in which the operator satisfies the four properties for group plus an extra property, commutativity. The four properties for groups plus commutativity are defined as follows:

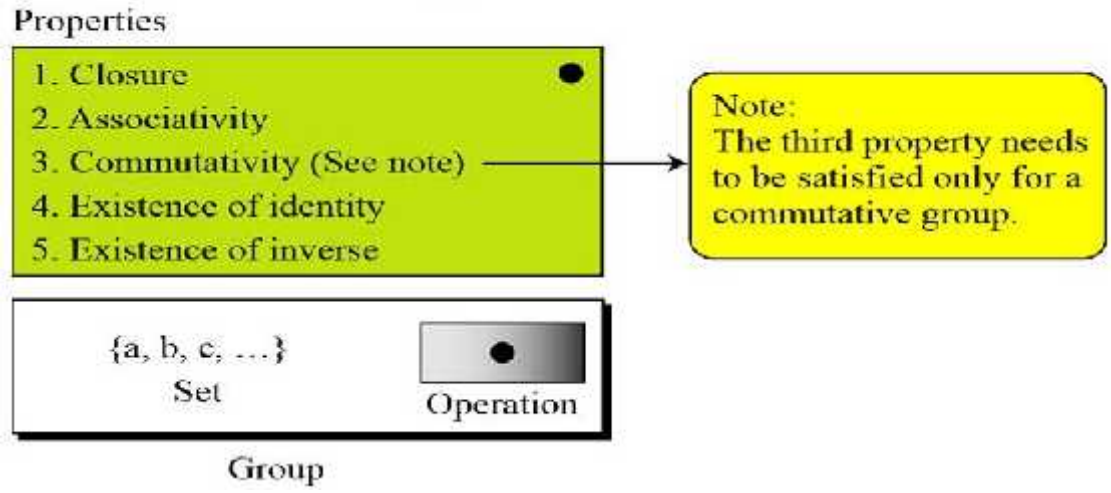
- **Closure:** if a and b are elements of G, then c= a •b is also an element of G. This means that the result of applying the operation on any two elements in the set is another element in the set.

- **Associativity:** If a,b,c are elements of G, then (a • b) •c = a • (b •c).

- **Commutativity:** For all a and b in G, we have (a • b)=(b.a).
- **Existence of Identity:** For all a in G, there exists an element e, called the identity element, such that e.a = a.e = a.

- **Existence of Inverse:** For each a in G,there exists an element a', called the inverse of a, such that a.a'=a'.a=e.

Figure 4.2 *Group*



Application:

- Although a group involves a single operation, the properties imposed on the operation allow the use of a pair of operaions as long as they are inverses of each other.
- For example, if the defined operation is addition, the group supports both addition and subtraction , because subtraction is addition using the additive inverse. This is also true for multiplication and division. However, a group can support only addition/subtraction or multiplication/division operations, but not the both at the same time.

Example 4.1 The set of residue integers with the addition operator, $G = \langle Z_n, + \rangle$, is a commutative group. We can perform addition and subtraction on the elements of this set without moving out of the set. Let us check the properties.

1. Closure is satisfied. The result of adding two integers in Z_n is another integer in Z_n.
2. Associativity is satisfied. The result of $4 + (3 + 2)$ is the same as $(4 + 3) + 2$.
3. Commutativity is satisfied. We have $3 + 5 = 5 + 3$.
4. The identify element is 0. We have $3 + 0 = 0 + 3 = 3$.
5. Every element has an additive inverse. The inverse of an element is its complement. For example, the inverse of 3 is -3 (n - 3 in Z_n) and the inverse of -3 is 3. The inverse allows us to perform subtraction on the set.

Finite Group:

A group is called a finite group if the set has a finite number of elements; otherwise, it is an infinite group.

Order of a group:

The order of a group , |G| , is the number of elements in the group. If the group is not finite, its order is infinite; if the group is finite, the order is finite.

Subgroups:

A subset H of a group G is a subgroup of G if H itself is a group with respect to the operation on

G. In other words , if G = < S, •> is a group , H = <T, •> is a group under the same operation, and T is a nonempty subset of S, then H is a subgroup of G. The above definition implies that:

1. if a and b are members of both groups, then c=a.b is also a member of both groups.
2. The group share the same identity element.
3. If a is a member of both groups, the inverse of a is also a member of both groups.
4. The group made of the identity element of G, H=< {e}, . >, is a subgroup of G.
5. Each group is a subgroup of itself.

Example 4.6 Is the group $H = \langle Z_{10}, + \rangle$ a subgroup of the group $G = \langle Z_{12}, + \rangle$?

Solution The answer is no. Although **H** is a subset of **G**, the operations defined for these two groups are different. The operation in **H** is addition modulo 10; the operation in **G** is addition modulo 12.

Cyclic Subgroups:

If a subgroup of a group can be generated using the power of an element, the subgroup is called the **cyclic subgroup.** The term *power* here means repeatedly applying the group operation to the element:

$a^n \rightarrow a \cdot a \ldots\ldots \cdot a$ (n times)

The set made from this process is referred to as <a>. Note that the duplicate elements must be discarded. Note also that a^0=e.

Example:

Four cyclic subgroups can be made from the group $G = \langle Z_6, + \rangle$. They are $H_1 = \langle \{0\}, + \rangle$, $H_2 = \langle \{0, 2, 4\}, + \rangle$, $H_3 = \langle \{0, 3\}, + \rangle$, and $H_4 = G$.

$0^0 \bmod 6 = 0$

$1^0 \bmod 6 = 0$
$1^1 \bmod 6 = 1$
$1^2 \bmod 6 = (1 + 1) \bmod 6 = 2$
$1^3 \bmod 6 = (1 + 1 + 1) \bmod 6 = 3$
$1^4 \bmod 6 = (1 + 1 + 1 + 1) \bmod 6 = 4$
$1^5 \bmod 6 = (1 + 1 + 1 + 1 + 1) \bmod 6 = 5$

$2^0 \bmod 6 = 0$
$2^1 \bmod 6 = 2$
$2^2 \bmod 6 = (2 + 2) \bmod 6 = 4$

$3^0 \bmod 6 = 0$
$3^1 \bmod 6 = 3$

$4^0 \bmod 6 = 0$
$4^1 \bmod 6 = 4$
$4^2 \bmod 6 = (4 + 4) \bmod 6 = 2$

$5^0 \bmod 6 = 0$
$5^1 \bmod 6 = 5$
$5^2 \bmod 6 = 4$
$5^3 \bmod 6 = 3$
$5^4 \bmod 6 = 2$
$5^5 \bmod 6 = 1$

Cyclic Groups:

- A Cyclic group is a group that is its own cyclic subgroup. The group **G** has a cyclic subgroup H_5 = G. This means that the group G is a cyclic group.

- In this case , the element that generates the cyclic subgroup can also generate the group itself.
- This element is referred to as a generator.
- If g is a generator, the element in a finite cyclic group can be written as, $\{e,g,g^2,g^3,\ldots\ldots,g^{n-1}\}$,where $g^n = e$

  Note that a cyclic group can have many generators.

Lagrange's Theorem:

- Lagrange's theorem relates the order of a group to the order of its subgroup.
- Assume that G is a group, and H is a subgroup of G. If the order of G and H are |G| and |H| ,respectively, then based on this theorem, |H| divides |G|.

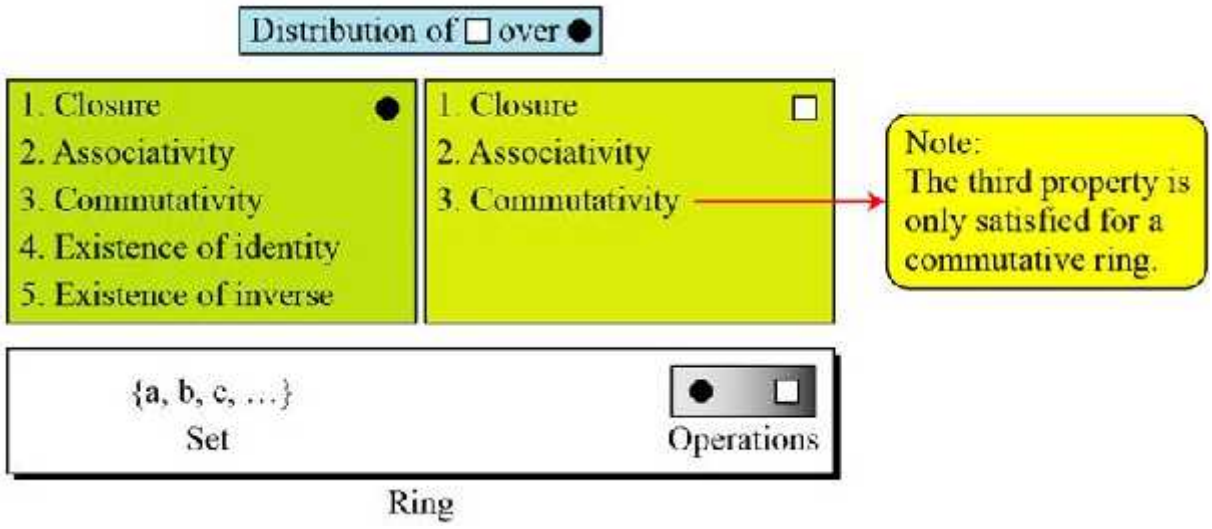
- In this example , H_1=<{0},+>,H_2 = <{0,2,4}, +>, H_3=<{0,3}, +>, then $|H_1|=1, |H_2|=3, |H_3|=2$.

Order of an element:

- The order of an element a in a group, ord(a),is the smallest integer n such that $a^n = e$.

➢ **Ring:**

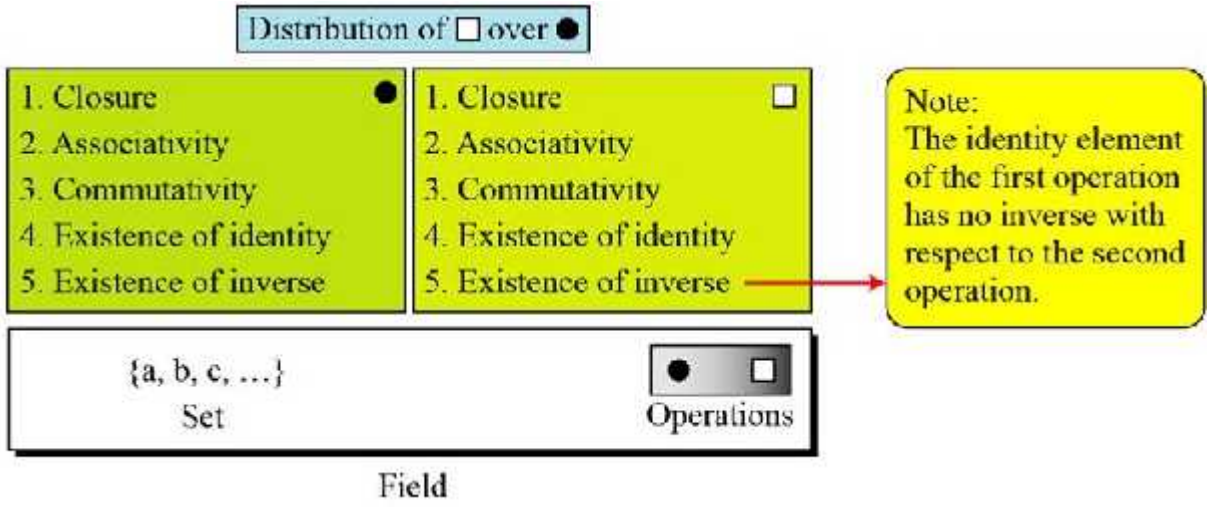
- A ring, denoted as R = <{ ……}, $\bullet$, □ **>** , is an algebraic structure with two operations.
- The first operation must satisfy all 5 properties required for an abelian group.
- The second operation must satisfy only the first two.
- In addition, the second operation must be distributed over the first.
- **Distributivity** means that for all a,b, and c elements of R, we have
  a □ (b ● c) = (a □ b) ●(a □c) and (a ● b) □ c =(a□ c) ● (b □ c).
- A **commutative ring** is a ring in which the commutative property is also satisfied for the second operation.



Application: A ring involves two operations. However the second operation can fail to satisfy the third and fourth properties.

- **Field:**

- A **field**, denoted by F= <{ ……}, ● , □ **>** is a commutative ring in which the second operation satisfies all five properties defined for the first operation except that the identity of the first operation has no inverse.



Application:

A field is a structure that supports two pairs of operations that we have used in mathematics: addition/subtraction and multiplication/division. There is an exception: division by zero is not allowed.

- **Finite fields:**

- Although we have fields of infinite order, only finite fields extensively used in cryptography. A finite field, a field with a finite number of elements,are very important structures in cryptography.

- Galios showed that for a field to be finiter,the number of elements should be p^n,where p is prime and n is a positive integer.
- The finite fields are usually called as Galios fields and denoted as **GF(p^n).**-

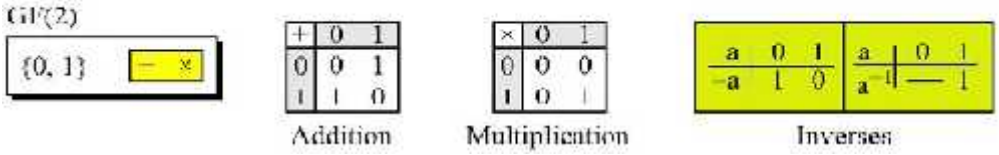
GF(P) Fields:

- When n=1, we have GF(p)field.The field can be the set Z_p,{0,1,2,…….p-1},with two arithmetic operations.

- Recall that each element has an additive inverse and that nonzero elements have a multiplicative inverse(no multiplicative inverse for 0)

A very common field in this category is GF(2) with the set {0, 1} and two operations, addition and multiplication, as shown in Figure 4.6.

Figure 4.6 *GF(2) field*

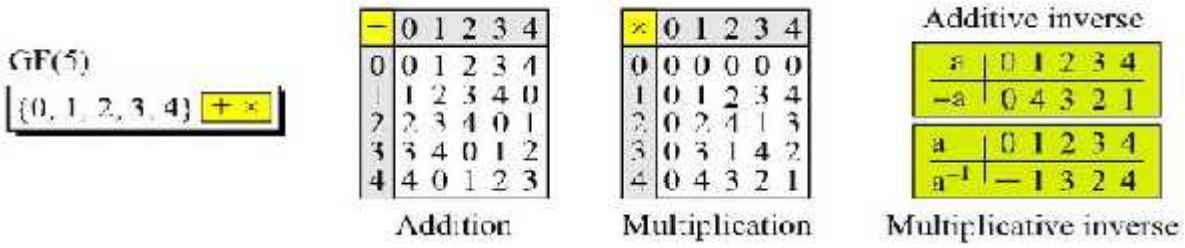


Example:1

There are several things to notice about this field.First, the set has only two elements, which are binary digits or bits(0 and 1) . second, the addition operation is actually the exclusive-or(XOR) operation we use on two binary digits.Third , the multiplication operation is the AND operation we use on two binary digits. Fourth, addition and subtraction are the same(XOR). Fifth, multiplication and division are the same(AND operation).

Example :2

- We can define GF(5) on the set Z_5 (5 is a prime) with addition and multiplication operators



Although we can use the extended Euclidean algorithm to find the multiplicative inverses of elements in GF(5), it is simpler to find each pair with the product equal to 1.They are (1,1),(2,3),(3,2),(4,4).

GF(p^n) Fields:

In addition to GF(p) fields,we have to know GF(p^n) fields in cryptography.

Algebraic Structure	*Supported Typical Operations*	*Supported Typical Sets of Integers*
Group	$(+ \ -)$ or $(\times \ \div)$	$\mathbf{Z}_n$ or $\mathbf{Z}_n^*$
Ring	$(+ \ -)$ and $(\times)$	$\mathbf{Z}$
Field	$(+ \ -)$ and $(\times \ \div)$	$\mathbf{Z}_p$

Summary:

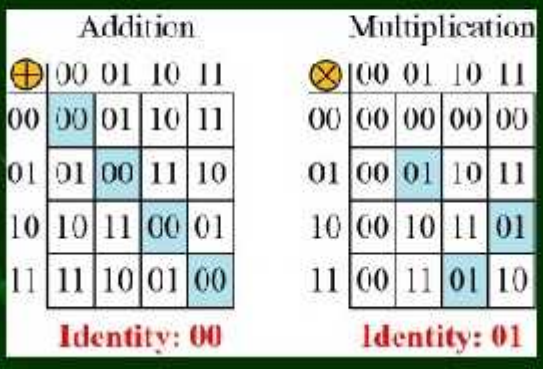
- **GF(2^n) FIELDS:**

In cryptography, we often need to use four operations (addition, subtraction, multiplication, and division). In other words, we need to use fields. We can work in GF(2^n) and uses a set of 2^n elements. The elements in this set are n-bit words.

1. We can use GF(p) with the set Zp, where p is the largest prime number less than 2^n. For example, if n=4, the largest prime less than 2^4 is 13.This means that we cannot use integers 13,14,15.
2. We can work in GF(2^n) and uses a set of 2^n elements. The elements in this setare n-bit words, for example, if n=3, the set is
   {000,001,010,011,100,101,110,111}

Let us define a GF(2^2) field in which the set has four 2-bit words: {00, 01, 10, 11}. We can redefine addition and multiplication for this field in such a way that all properties of these operations are satisfied, as shown in Figure 4.8.

Figure 4.8 *An example of GF(2^2) field*



Polynomials:

Although we can directly define the rules for addition and multiplication operations on n-bit words that satisfy the properties in GF(2^n). It is easier to work with a representation of n-bit words, a polynomial of degree n-1, A polynomial of degree n-1 is an expression of the form

$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = \sum a_i x^i$

where x^i is called the ith term and a_i is called coefficient of the ith term. Although we are familiar with polynomials in algebra, to represent an n-bit word by a polynomial we need to follow some rules:

a. The power of x defines the position of the bit in the n-bit word. This means the leftmost bit is at position zero (related to x^0); the rightmost bit is at position $n - 1$ (related to x^{n-1}).
b. The coefficients of the terms define the value of the bits. Because a bit can have only a value of 0 or 1, our polynomial coefficients can be either 0 or 1.

Example 4.15 Figure 4.9 shows how we can represent the 8-bit word (10011001) using a polynomials.

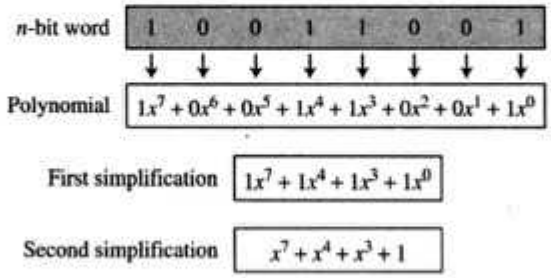


Fig. 4.9 *Representation of an 8-bit word by a polynomial*

Note that the term is totally omitted if the coefficient is 0, and the coefficient is omitted if it is 1. Also note that x^0 is 1.

Example 4.16 To find the 8-bit word related to the polynomial $x^5 + x^2 + x$, we first supply the omitted terms. Since $n = 8$, it means the polynomial is of degree 7. The expanded polynomial is

$$0x^7 + 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 1x^1 + 0x^0$$

This is related to the 8-bit word 00100110.

Operations Note that any operation on polynomials actually involves two operations: operations on coefficients and operations on two polynomials. In other words, we need to define two fields: one for the coefficients and one for the polynomials. Coefficients are made of 0 or 1; we can use the **GF**(2) field for this purpose. We discusses this field before (see Example 4.14). For the polynomials we need the field GF(2^n), which we will discuss shortly.

Polynomials representing *n*-bit words use two fields: GF(2) and GF(2^n).

Modulus Before defining the operations on polynomials, we need to talk about the modulus polynomials. Addition of two polynomials never creates a polynomial out of the set. However, multiplication of two polynomials may create a polynomial with degrees more than $n - 1$. This means

- We need to divide the result by a modulus and keep only the remainder, as in modular arithmetic.For the sets of polynomials in GF(2^n) , a group of polynomials of degree n is defined as the modulus.
- The modulus in this case acts as a prime polynomial, which means that no polynomials in the set can divide this polynomial.
- A prime polynomial cannot be factored into a polynomial with degree of less than n.such polynomials are referred to as irreducible polynomial.

Degree	*Irreducible Polynomials*
1	$(x + 1), (x)$
2	$(x^2 + x + 1)$
3	$(x^3 + x^2 + 1), (x^3 + x + 1)$
4	$(x^4 + x^3 + x^2 + x + 1), (x^4 + x^3 + 1), (x^4 + x + 1)$
5	$(x^5 + x^2 + 1), (x^5 + x^3 + x^2 + x + 1), (x^5 + x^4 + x^3 + x + 1),$ $(x^5 + x^4 + x^3 + x^2 + 1), (x^5 + x^4 + x^2 + x + 1)$

Addition: Now let us define the addition operation for polynomials with coeffieients in GF(2). Addition is very easy. We add the coefficients of the corresponding terms in GF(2).

Note that adding two polynomials of degree n-1 always create a polynomial with degree n-1,which means that we do not need to reduce the result using the modulus.

Example 4.17 Let us do $(x^5 + x^2 + x) \oplus (x^3 + x^2 + 1)$ in GF(2^8). We use the symbol $\oplus$ to show that we mean polynomial addition. The following shows the procedure:

$$\begin{array}{ll} 0x^7 + 0x^6 + 1x^5 + 0x^4 + 0x^3 + 1x^2 + 1x^1 + 0x^0 & \oplus \\ 0x^7 + 0x^6 + 0x^5 + 0x^4 + 1x^3 + 1x^2 + 0x^1 + 1x^0 & \\ \hline 0x^7 + 0x^6 + 1x^5 + 0x^4 + 1x^3 + 0x^2 + 1x^1 + 1x^0 & \rightarrow \quad x^5 + x^3 + x + 1 \end{array}$$

There is a short cut: keeps the uncommon terms and delete the common terms. In other words, x^5, x^3, x, and 1 are kept and x^2, which is common in the two polynomials, is deleted.

Additive Identity:

The additive identity in a polynomial is a zero polynomial because adding the polynomial with itself results in a zero polynomial.

Additive Inverse:

The additive inverse of a polynomial with coefficients in GF(2) is the polynomial itself. This means that the subtraction operation is the same as the addition operation.

Multiplication:

Multiplication in polynomial is the sum of the multiplication of each term of the first polynomial with each term of the second polynomial.we have to consider three points.

Example 4.19 Find the result of $(x^5 + x^2 + x) \otimes (x^7 + x^4 + x^3 + x^2 + x)$ in GF(2^8) with irreducible polynomial $(x^8 + x^4 + x^3 + x + 1)$. Note that we use the symbol $\otimes$ to show the multiplication of two polynomials.

Solution We first multiply the two polynomials as we have learned in algebra. Note that in this process, a pair of terms with equal power of x are deleted. For example, $x^9 + x^9$ is totally deleted because the result is a zero polynomial, as we discussed above.

$$P_1 \otimes P_2 = x^5(x^7 + x^4 + x^3 + x^2 + x) + x^2(x^7 + x^4 + x^3 + x^2 + x) + x(x^7 + x^4 + x^3 + x^2 + x)$$
$$P_1 \otimes P_2 = x^{12} + x^9 + x^8 + x^7 + x^6 + x^9 + x^6 + x^5 + x^4 + x^3 + x^8 + x^5 + x^4 + x^3 + x^2$$
$$P_1 \otimes P_2 = (x^{12} + x^7 + x^2) \bmod (x^8 + x^4 + x^3 + x + 1) = x^5 + x^3 + x^2 + x + 1$$

To find the final result, divide the polynomial of degree 12 by the polynomial of degree 8 (the modulus) and keep only the remainder. The process is the same as we have learned in algebra, but we need to remember that subtraction is the same as addition here. Figure 4.10 shows the process of division.

$$\begin{array}{r|l} & x^4 + 1 \\ \hline x^8 + x^4 + x^3 + x + 1 & x^{12} + x^7 + x^2 \\ & x^{12} + x^8 + x^7 + x^5 + x^4 \\ \hline & x^8 + x^5 + x^4 + x^2 \\ & x^8 + x^4 + x^3 + x + 1 \\ \hline \text{Remainder} & \boxed{x^5 + x^3 + x^2 + x + 1} \end{array}$$

Multiplicative Identity The multiplicative identity is always 1. For example, in GF(2^8), the multiplicative inverse is the bit pattern 00000001.

Multiplicative Inverse Finding the multiplicative inverse is a little more involved. The extended Euclidean algorithm must be applied to the modulus and the polynomial. The process is exactly the same as for integers.

Example 4.20 In GF(2^4), find the inverse of $(x^2 + 1)$ modulo $(x^4 + x + 1)$.

Solution We use the extended Euclidean algorithm as in Table 4.5:

Table 4.5 *Euclidean algorithm for Exercise 4.20*

q	r_1	r_2	r	t_1	t_2	t
$(x^2 + 1)$	$(x^4 + x + 1)$	$(x^2 + 1)$	(x)	(0)	(1)	$(x^2 + 1)$
(x)	$(x^2 + 1)$	(x)	(1)	(1)	$(x^2 + 1)$	$(x^3 + x + 1)$
(x)	(x)	(1)	(0)	$(x^2 + 1)$	$(x^3 + x + 1)$	(0)
	(1)	(0)		$(x^3 + x + 1)$	(0)	

This means that $(x^2 + 1)^{-1}$ modulo $(x^4 + x + 1)$ is $(x^3 + x + 1)$. The answer can be easily proved by multiplying the two polynomials and finding the remainder when the result is divided by the modulus.

$$[(x^2 + 1) \otimes (x^3 + x + 1)] \bmod (x^4 + x + 1) = 1$$

Example 4.21 In GF(2^8), find the inverse of (x^5) modulo ($x^8 + x^4 + x^3 + x + 1$).

Solution Use the Extended Euclidean algorithm as shown in Table 4.6:

Table 4.6 *Euclidean algorithm for Example 4.21*

q	r_1	r_2	r	t_1	t_2	t
(x^3)	$(x^8+x^4+x^3+x+1)$	(x^5)	(x^4+x^3+x+1)	(0)	(1)	(x^3)
$(x+1)$	(x^5)	(x^4+x^3+x+1)	(x^3+x^2+1)	(1)	(x^3)	(x^4+x^3+1)
(x)	(x^4+x^3+x+1)	(x^3+x^2+1)	(1)	(x^3)	(x^4+x^3+1)	$(x^5+x^4+x^3+x)$
(x^3+x^2+1)	(x^3+x^2+1)	(1)	(0)	(x^4+x^3+1)	$(x^5+x^4+x^3+x)$	(0)
	(1)	(0)		$(x^5+x^4+x^3+x)$	(0)	

This means that $(x^5)^{-1}$ modulo $(x^8 + x^4 + x^3 + x + 1)$ is $(x^5 + x^4 + x^3 + x)$. The answer can be easily proved by multiplying the two polynomials and finding the remainder when the result is divided by the modulus.

$$[(x^5) \otimes (x^5 + x^4 + x^3 + x)] \text{ modulo } (x^8 + x^4 + x^3 + x + 1) = 1$$

Multiplication Using a Computer Because of the division operation, there is an efficiency problem involved in writing a program to multiply two polynomials. The computer implementation uses a better algorithm, repeatedly multiplying a reduced polynomial by x. For example, instead of finding the result of $(x^2 \otimes P_2)$, the program finds the result of $(x \otimes (x \otimes P_2))$. The benefit of this strategy will be discussed shortly, but first let us use an example to show the process.

Example 4.22 Find the result of multiplying $P_1 = (x^5 + x^2 + x)$ by $P_2 = (x^7 + x^4 + x^3 + x^2 + x)$ in $GF(2^8)$ with irreducible polynomial $(x^8 + x^4 + x^3 + x + 1)$ using the algorithm described above.

Solution The process is shown in Table 4.7. We first find the partial result of multiplying x^0, x^1, x^2, x^3, x^4, and x^5 by P_2. Note that although only three terms are needed, the product of $x^m \otimes P_2$ for m from 0 to 5 because each calculation depends on the previous result.

Table 4.7 *An efficient algorithm for multiplication using polynomials (Example 4.22)*

Powers	*Operation*	*New Result*	*Reduction*
$x^0 \otimes P_2$		$x^7 + x^4 + x^3 + x^2 + x$	No
$x^1 \otimes P_2$	$x \otimes (x^7 + x^4 + x^3 + x^2 + x)$	$x^5 + x^2 + x + 1$	**Yes**
$x^2 \otimes P_2$	$x \otimes (x^5 + x^2 + x + 1)$	$x^6 + x^3 + x^2 + x$	No
$x^3 \otimes P_2$	$x \otimes (x^6 + x^3 + x^2 + x)$	$x^7 + x^4 + x^3 + x^2$	No
$x^4 \otimes P_2$	$x \otimes (x^7 + x^4 + x^3 + x^2)$	$x^5 + x + 1$	**Yes**
$x^5 \otimes P_2$	$x \otimes (x^5 + x + 1)$	$x^6 + x^2 + x$	No
$P_1 \times P_2 = (x^6 + x^2 + x) + (x^6 + x^3 + x^2 + x) + (x^5 + x^2 + x + 1) = x^5 + x^3 + x^2 + x + 1$			

The above algorithm has two benefits. First, multiplication of a polynomial by x can be easily achieved by one-bit shifting of the n-bit word; an operation provided by common programming languages. Second, the result needed to be reduced only if the polynomial maximum power is $n - 1$. In this case, reduction can be easily done by an XOR operation with the modulus because the highest power in the result is only 8. We can then design a simple algorithm to find each partial result:

1. If the most significant bit of the previous result is 0, just shift the previous result one bit to the left.
2. If the most significant bit of the previous result is 1,
   a. shift it one bit to the left, and
   b. exclusive-or it with the modulus without the most significant bit

Example 4.23 Repeat Example 4.22 using bit patterns of size 8.

Solution We have $P_1 = 000100110$, $P_2 = 10011110$, modulus = 100011010 (nine bits). We show the exclusive-or operation by $\oplus$. See Table 4.8.

Table 4.8 *An efficient algorithm for multiplication using n-bit words*

Powers	*Shift-Left Operation*	*Exclusive-Or*
$x^0 \otimes P_2$		10011110
$x^1 \otimes P_2$	00111100	(00111100) $\oplus$ (00011010) = **00100111**
$x^2 \otimes P_2$	01001110	**01001110**
$x^3 \otimes P_2$	10011100	10011100
$x^4 \otimes P_2$	00111000	(00111000) $\oplus$ (00011010) = 00100011
$x^5 \otimes P_2$	01000110	**01000110**
$P_1 \otimes P_2 = (00100111) \oplus (01001110) \oplus (01000110) = 00101111$		

In this case, we need only five shift-left operations and four exclusive-or operations to multiply the two polynomials. In general, a maximum of $n - 1$ shift-left operations and $2n$ exclusive-or operations are needed to multiply two polynomial of degree $n - 1$.

Multiplication of polynomials in GF(2^n) can be achieved using shift-left and exclusive-or operations.

Example 4.24 The GF(2^3) field has 8 elements. We use the irreducible polynomial ($x^3 + x^2 + 1$) and show the addition and multiplication tables for this field. We show both 3-bit words and the polynomials. Note that there are two irreducible polynomials for degree 3. The other one, ($x^3 + x + 1$), yields a totally different table for multiplication. Table 4.9 shows addition. The shaded boxes easily give us the additive inverses pairs.

Table 4.9 *Addition table for GF(2^3)*

⊕	000 (0)	001 (1)	010 (x)	011 ($x+1$)	100 (x^2)	101 x^2+1	110 (x^2+x)	111 (x^2+x+1)
000 (0)	000 (0)	001 (1)	010 (x)	011 ($x+1$)	100 (x^2)	101 (x^2+1)	110 (x^2+x)	111 (x^2+x+1)
001 (1)	001 (1)	000 (0)	011 ($x+1$)	010 (x^2)	101 (x^2+1)	100 (x^2+x)	111 (x^2+x+1)	110 (x^2+x)
010 (x)	010 (x)	011 ($x+1$)	000 (0)	001 (1)	110 (x^2+x)	111 (x^2+x+1)	100 (x^2+x)	101 (x^2+1)
011 ($x+1$)	011 ($x+1$)	010 (x)	001 (1)	000 (0)	111 (x^2+x+1)	110 (x^2+x)	101 (x^2+1)	100 (x^2)
100 (x^2)	100 (x^2)	101 (x^2+1)	110 (x^2+x)	111 (x^2+x+1)	000 (0)	001 (1)	010 (x)	011 ($x+1$)
101 (x^2+1)	101 (x^2+1)	100 (x^2)	111 (x^2+x+1)	110 (x^2+x)	001 (1)	000 (0)	011 ($x+1$)	010 (x)
110 (x^2+x)	110 (x^2+x)	111 (x^2+x+1)	100 (x^2)	101 (x^2+1)	010 (x)	011 ($x+1$)	000 (0)	001 (1)
111 (x^2+x+1)	111 (x^2+x+1)	110 (x^2+x)	101 (x^2+1)	100 (x^2)	011 ($x+1$)	010 (x)	001 (1)	000 (0)

Multiplication using a generator:

- Sometimes it is easier to define the elements of GF(2^n) using a generator. In this field with the irreducible polynomial f(x), an element in the field, a, must satisfy the relation f(a)=0.

Inverses

Finding inverses using the above representation is simple.

Additive Inverses The additive inverse of each element is the element itself because addition and subtraction in this field are the same: $-g^3 = g^3$

Multiplicative Inverses Finding the multiplicative inverse of each element is also very simple. For example, we can find the multiplicative inverse of g^3 as shown below:

$$(g^3)^{-1} = g^{-3} = g^{12} = g^3 + g^2 + g + 1 \rightarrow (1111)$$

Note that the exponents are calculated modulo $2^n - 1$, 15 in this case. Therefore, the exponent –3 mod 15 = 12 mod 15. It can be easily proved that g^3 and g^{12} are inverses of each other because $g^3 \times g^{12} = g^{15} = g^0 = 1$.

Operations The four operations defined for the field can also be performed using this representation.

Addition and Subtraction Addition and subtraction are the same operation. The intermediate results can be simplified as shown in the following example.

Example 4.26 The following show the results of addition and subtraction operations:

a. $g^3 + g^{12} + g^7 = g^3 + (g^3 + g^2 + g + 1) + (g^3 + g + 1) = g^3 + g^2 \rightarrow (1100)$
b. $g^3 - g^6 = g^3 + g^6 = g^3 + (g^3 + g^2) = g^2 \rightarrow (0100)$

$\{0, g, g, g^2, \ldots, g^N\}$, where $N = 2^n - 2$

⊗	000 (0)	001 (1)	010 (x)	011 ($x+1$)	100 (x^2)	101 (x^2+1)	110 (x^2+x)	111 (x^2+x+1)
000 (0)	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)
001 (1)	000 (0)	001 (1)	010 (x)	011 ($x+1$)	100 (x^2)	101 (x^2+1)	110 (x^2+x)	111 (x^2+x+1)
010 (x)	000 (0)	010 (x)	100 (x^2)	110 (x^2+x)	101 (x^2+1)	111 (x^2+x+1)	001 (1)	011 ($x+1$)

Multiplication and Division Multiplication is the addition of powers modulo $2^n - 1$. Division is multiplication using the multiplicative inverse.

Example 4.27 The following show the result of multiplication and division operations:

a. $g^9 \times g^{11} = g^{20} = g^{20 \bmod 15} = g^5 = g^2 + g \rightarrow (0110)$

b. $g^3 / g^8 = g^3 \times g^7 = g^{10} = g^2 + g + 1 \rightarrow (0111)$

Conclusion

The finite field GF(2^n) can be used to define four operations of addition, subtraction, multiplication and division over n-bit words. The only restriction is that division by zero is not defined. Each n-bit word can also be represented as a polynomial of degree $n - 1$ with coefficients in GF(2), which means that the operations on n-bit words are the same as the operations on this polynomial. To make it modular, we need to define an irreducible polynomial of degree n when we multiply two polynomials. The extended Euclidean algorithm can be applied to polynomials to find the multiplicative inverses.

TOPIC 02:

- **Introduction to modern block ciphers:**

A symmetric-key **modern block cipher** encrypts an n-bit block of plaintext or decrypts an n-bit block of ciphertext. The encryption or decryption algorithm uses a k-bit key. The decryption algorithm must be the inverse of the encryption algorithm, and both operations must use the same secret key so that Bob can retrieve the message sent by Alice. Figure 5.1 shows the general idea of encryption and decryption in a modern block cipher.

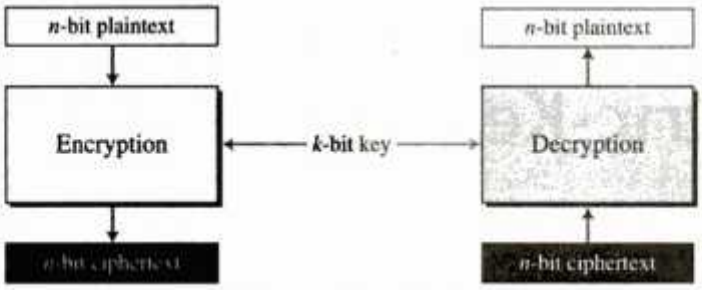


Fig. 5.1 *A modern block cipher*

If the message has fewer than n bits, padding must be added to make it an n-bit block; if the message has more than n bits, it should be divided into n-bit blocks and the appropriate padding must be added to the last block if necessary. The common values for n are 64, 128, 256, or 512 bits.

- **Substitution or Transposition:-**

A modern block cipher can be designed to act as a substitution cipher or a transposition cipher. This is the same idea as is used in traditional ciphers, except that the symbols to be substituted or transposed are bits instead of characters.

If the cipher is designed as a substitution cipher, a 1-bit or a 0-bit in the plaintext can be replaced by either a 0 or a 1. This means that the plaintext and the ciphertext can have a different number of 1's. A 64-bit plaintext block of 12 0's and 52 1's can be encrypted to a ciphertext of 34 0's and 30 1's. If the cipher is designed as a transposition cipher, the bits are only reordered (transposed); there is the same number of 1's in the plaintext and in the ciphertext. In either case, the number of n-bit possible plaintexts or ciphertexts is 2^n, because each of the n bits in the block can have one of the two values, 0 or 1.

- **Block ciphers as permutation groups:-**

As we will see in later chapters, we need to know whether a modern block cipher is a group (see Chapter 4). To answer this question, first assume that the key is long enough to choose every possible mapping from the input to the output. Call this a full-size key cipher. In practice, however, the key is smaller; only some mappings from the input to the output are possible. Although a block cipher needs to have a key that is a secret between the sender and the receiver, there are also keyless components that are used inside a cipher.

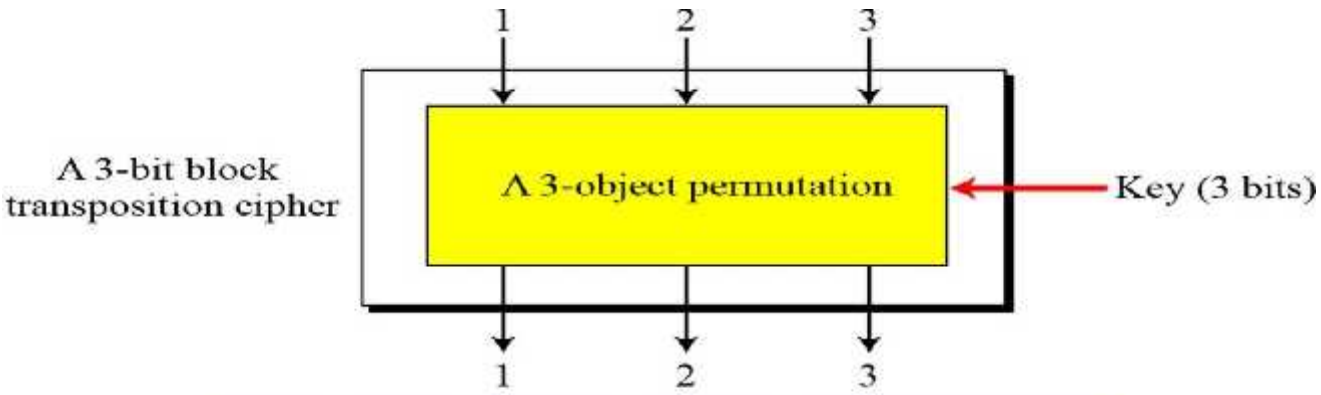
Full-Size Key Ciphers

Although full-size key ciphers are not used in practice, we first discuss this category to make the discussion of partial-size key ciphers understandable.

Full-Size Key Transposition Block Ciphers A full-size key transposition cipher only transposes bits without changing their values, so it can be modeled as an n-object permutation with a set of $n!$ permutation tables in which the key defines which table is used by Alice and Bob. We need to have $n!$ possible keys, so the key should have $\lceil \log_2 n! \rceil$ bits.

Example 5.3 Show the model and the set of permutation tables for a 3-bit block transposition cipher where the block size is 3 bits.

Solution The set of permutation tables has $3! = 6$ elements, as shown in Fig. 5.2. The key should be $\lceil \log_2 6 \rceil = 3$ bits long. Note that, although a 3-bit key can select $2^3 = 8$ different mappings, we use only 6 of them.



The set of permutation tables with $3! = 6$ elements

Full-Size Key Substitution Block Ciphers A full-size key substitution cipher does not transpose bits; it substitutes bits. At first glance, it appears that a full-size key substitution cipher cannot be modeled as a permutation. However, we can model the substitution cipher as a permutation if we can decode the input and encode the output. **Decoding** means transforming an n-bit integer into a 2^n-bit string with only a single 1 and $2^n - 1$ 0's. The position of the single 1 is the value of the integer, in which the positions range from 0 to $2^n - 1$. **Encoding** is the reverse process. Because the new input and output have always a single 1, the cipher can be modeled as a permutation of $2^n!$ objects.

Example 5.4 Show the model and the set of permutation tables for a 3-bit block substitution cipher.

Solution The three-input plaintext can be an integer between 0 to 7. This can be decoded as an 8-bit string with a single 1. For example, 000 can be decoded as 00000001; 101 can be decoded as 00100000. Figure 5.3 shows the model and the set of permutation tables. Note that the number of elements in the set is much bigger than the number of elements in the transposition cipher (8! = 40,320). The key is also much longer, $\lceil \log_2 40{,}320 \rceil = 16$ bits. Although a 16-bit key can define 65,536 different mappings, only 40,320 are used.

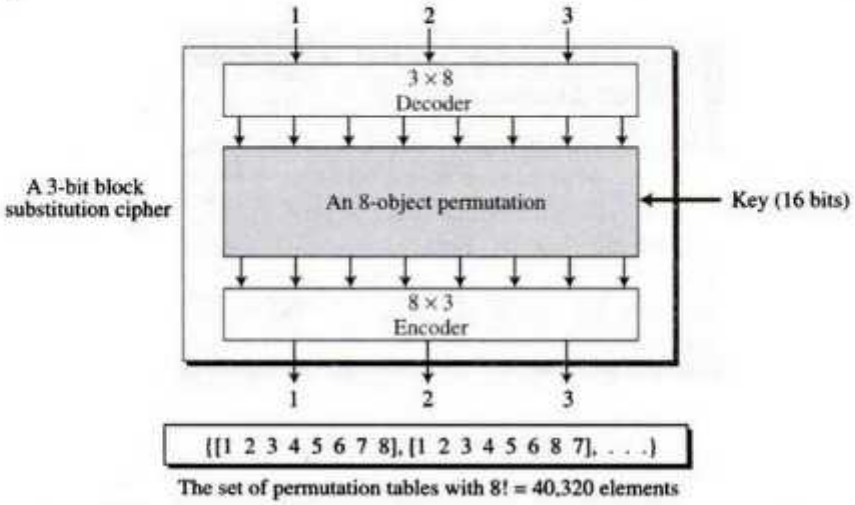


Fig. 5.3 *A substitution block cipher model as a permutation*

A full-size key n-bit transposition cipher or a substitution block cipher can be modeled as a permutation, but their key sizes are different:
For a transposition cipher, the key is $\lceil \log_2 n! \rceil$ bits long.
For a substitution cipher, the key is $\lceil \log_2 (2^n)! \rceil$ bits long.

Partial-Size Key Ciphers

Actual ciphers cannot use full-size keys because the size of the key becomes so large, especially for a substitution block cipher. For example, a common substitution cipher is DES (see Chapter 6), which uses a 64-bit block cipher. If the designers of DES had used a full-size key, the key would have been $\log_2(2^{64}!) \approx 2^{70}$ bits. The key size for DES is only 56 bits, which is a very small fraction of the full-size key. This means that DES uses only 2^{56} mappings out of approximately $2^{2^{70}}$ possible mappings.

A partial-key cipher is a group under the composition operation if it is a subgroup of the corresponding full-size key cipher.

Keyless Ciphers

Although a keyless cipher is practically useless by itself, keyless ciphers are used as components of keyed ciphers.

Keyless Transposition Ciphers A keyless (or fixed-key) transposition cipher (or unit) can be thought of as a prewired transposition cipher when implemented in hardware. The fixed key (single permutation rule) can be represented as a table when the unit is implemented in software. The next section of this chapter discusses keyless transposition ciphers, called D-boxes, which are used as building blocks of modern block ciphers.

Keyless Substitution Ciphers A keyless (or fixed-key) substitution cipher (or unit) can be thought of as a predefined mapping from the input to the output. The mapping can be defined as a table, a mathematical function, and so on. The next section of this chapter discusses keyless substitution ciphers, called S-boxes, which are used as building blocks of modern block ciphers.

- **Components of Modern block cipher:-**

Modern block ciphers normally are keyed substitution ciphers in which the key allows only partial mappings from the possible inputs to the possible outputs.
Modern block ciphers normally are keyed substitution ciphers in which the key allows only partial mappings from the possible inputs to the possible outputs**.**

Keyless transposition ciphers : D-boxes

Keyless substitution ciphers : S-boxes

D-boxes

A **D-box** (diffusion box) parallels the traditional transposition cipher for characters. It transposes bits. We can find three types of D-boxes in modern block ciphers: straight D-boxes, expansion D-boxes, and compression D-boxes, as shown in Fig. 5.4. It helps in the spreading or diffusion of the input disturbances.

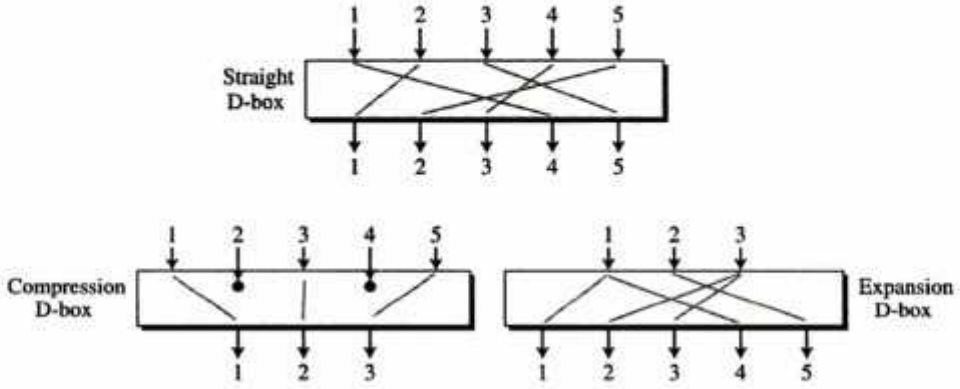


Fig. 5.4 *Three types of D-boxes*

Figure 5.4 shows a 5 × 5 straight D-box, a 5 × 3 compression D-box, and a 3 × 5 expansion D-box.

Straight D-Boxes: A straight D-box with n-inputs and n outputs is a permutation. There are n! possible mappings.

Example 5.5 Figure 5.5 shows all 6 possible mappings of a 3 × 3 D-box.

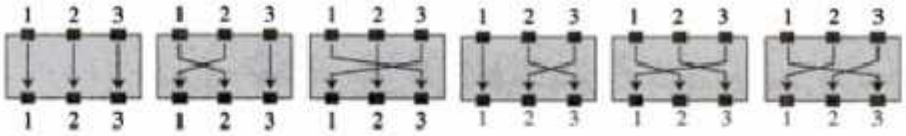


Fig. 5.5 *The possible mappings of a 3 × 3 D-box*

Although a D-box can use a key to define one of the $n!$ mappings, D-boxes are normally keyless, which means that the mapping is predetermined. If the D-box is implemented in hardware, it is prewired; if it is implemented in software, a permutation table shows the rule of mapping. In the second case, the entries in the table are the inputs and the positions of the entries are the outputs. Table 5.1 shows an example of a straight permutation table when n is 64.

Table 5.1 *Example of a permutation table for a straight D-box*

58	50	42	34	26	18	10	02	60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06	64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01	59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05	63	55	47	39	31	23	15	07

Table 5.1 has 64 entries, corresponding to the 64 inputs. The position (index) of the entry corresponds to the output. Because the first entry contains the number 58, we know that the first output comes from the 58th input. Because the last entry is 7, we know that the 64th output comes from the 7th input, and so on.

Compression D-boxes **A compression D-box** is a D-box with n inputs and m outputs where $m < n$. Some of the inputs are blocked and do not reach the output (see Fig. 5.4). The compression D-boxes used in modern block ciphers normally are keyless with a table showing the rule for transposing bits. We need to know that a table for a compression D-box has m entries, but the content of each entry is from 1 to n with some missing values (those inputs that are blocked). Table 5.2 shows an example of a table for a 32 × 24 compression D-box. Note that inputs 7, 8, 9, 15, 16, 23, 24, and 25 are blocked.

Table 5.2 *Example of a 32 × 24 D-box*

01	02	03	21	22	26	27	28	29	13	14	17
18	19	20	04	05	06	10	11	12	30	31	32

Compression D-boxes are used when we need to permute bits and the same time decrease the number of bits for the next stage.

Expansion D-boxes An **expansion D-box** is a D-box with n inputs and m outputs where $m > n$. Some of the inputs are connected to more than one input (see Fig. 5.4). The expansion D-boxes used in modern block ciphers normally are keyless, where a table shows the rule for transposing bits. We need to know that a table for an expansion D-box has m entries, but $m - n$ of the entries are repeated (those inputs mapped to more than one output). Table 5.3 shows an example of a table for a 12×16 expansion D-box. Note that each of the inputs 1, 3, 9, and 12 is mapped to two outputs.

Table 5.3 *Example of a 12 × 16 D-box*

01	09	10	11	12	01	02	03	03	04	05	06	07	08	09	12

Expansion D-boxes are used when we need to transpose bits and the same time increase the number of bits for the next stage.

Invertibility A straight D-box is invertible. This means that we can use a straight D-box in the encryption cipher and its inverse in the decryption cipher. The mapping defined by a straight D-box is a permutation, and thus may be referred to as P-box. The permutation tables, however, need to be the inverses of each other. In Chapter 3, we saw how we can make the inverse of a permutation table.

Example 5.7 Figure 5.6 shows how to invert a permutation table represented as a one-dimensional table.

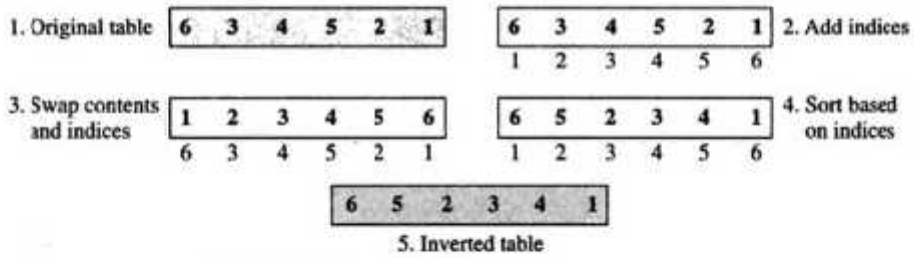


Fig. 5.6 *Inverting a permutation table*

Compression and expansion D-boxes have no inverses. In a compression D-box, an input can be dropped during encryption; the decryption algorithm does not have a clue how to replace the dropped bit (a choice between a 0-bit or a 1-bit). In an expansion D-box, an input may be mapped to more than one output during encryption; the decryption algorithm does not have a clue which of the several inputs are mapped to an output. Figure 5.7 demonstrates both cases.

Figure 5.7 also shows that a compression D-box is not the inverse of an expansion D-box or vice versa. This means that if we use a compression D-box in the encryption cipher, we cannot use an expansion D-box in the decryption cipher; or vice versa. However, as will be shown later in this chapter, there are ciphers that use compression or expansion D-boxes in the encryption cipher; the effects of these are canceled in some other ways in the decryption cipher.

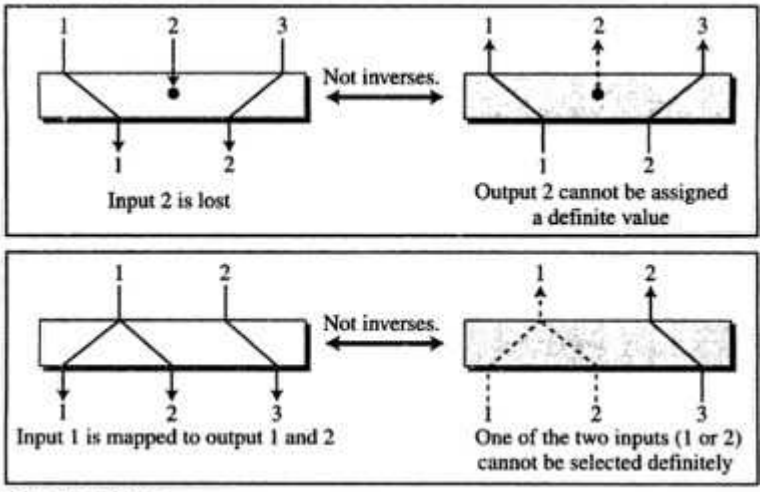
A straight D-box is invertible, but compression and expansion D-boxes are not.

5.1.4 S-Boxes

An **S-box** (substitution box) can be thought of as a miniature substitution cipher. However, an S-box can have a different number of inputs and outputs. In other words, the input to an S-box could be an n-bit word, but the output can be an m-bit word, where m and n are not necessarily the same. Although an S-box can be keyed or keyless, modern block ciphers normally use keyless S-boxes, where the mapping from the inputs to the outputs is predetermined.

An S-box is an $m \times n$ substitution unit, where m and n are not necessarily the same.

Compression D-box



Expansion D-box

Example 5.9 In an S-box with three inputs and two outputs, we have

$$y_1 = x_1x_2 \qquad y_2 = x_1 + x_2x_3$$

where multiplication and addition is in GF(2). The S-box is nonlinear because there is no linear relationship between the inputs and the outputs. Note the 'and' term(s) x_1x_2 and x_2x_3.

Linear Versus Nonlinear S-Boxes

In an S-box with n inputs and m outputs, we call the inputs $x_1, \ldots, x_n$ and the outputs $y_1, \ldots, y_m$. The relationship between the inputs and the outputs can be represented as a set of equations

$$\begin{aligned} y_1 &= f_1(x_1, x_2, \ldots, x_n) \\ y_2 &= f_2(x_1, x_2, \ldots, x_n) \\ &\ldots \\ y_m &= f_m(x_1, x_2, \ldots, x_n) \end{aligned}$$

In a **linear S-box,** the above relations can be expressed as

$$\begin{aligned} y_1 &= a_{1,1}x_1 \oplus a_{1,2}x_1 \oplus \ldots \oplus a_{1,n}x_n \\ y_2 &= a_{2,1}x_1 \oplus a_{2,2}x_1 \oplus \ldots \oplus a_{2,n}x_n \\ &\ldots \\ y_m &= a_{m,1}x_1 \oplus a_{m,2}x_1 \oplus \ldots \oplus a_{m,n}x_n \end{aligned}$$

In a **nonlinear S-box** we cannot have the above relations for every output. Such an S-box will have 'and' terms, like x_1x_2, x_3x_7 etc. in their expressions.

Example 5.8 In an S-box with three inputs and two outputs, we have

$$y_1 = x_1 \oplus x_2 \oplus x_3 \quad y_2 = x_1$$

The S-box is linear because $a_{1,1} = a_{1,2} = a_{1,3} = a_{2,1} = 1$ and $a_{2,2} = a_{2,3} = 0$. The relationship can be represented by matrices, as shown below:

Invertibility S-boxes are substitution ciphers in which the relationship between input and output is defined by a table or mathematical relation. An S-box may or may not be invertible. In an invertible S-box, the number of input bits should be the same as the number of output bits.

Example 5.11 Figure 5.8 shows an example of an invertible S-box. One of tables is used in the encryption algorithm; the other table is used in the decryption algorithm. In each table, the leftmost bit of the input defines the row; the next two bits define the column. The output is the value where the input row and column meet.

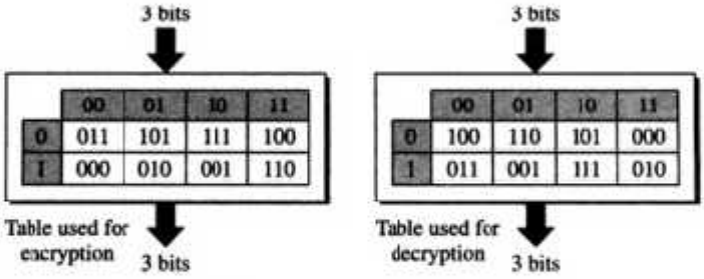


Fig. 5.8 *S-box tables for Example 5.11*

For example, if the input to the left box is 001, the output is 101. The input 101 in the right table creates the output 001, which shows that the two tables are inverses of each other.

Exclusive-Or

An important component in most block ciphers is the *exclusive-or* operation. As we discussed in Chapter 4, addition and subtraction operations in the $GF(2^n)$ field are performed by a single operation called the *exclusive-or* (XOR).

Inverse The inverse of a component in a cipher makes sense if the component represents a unary operation (one input and one output). For example, a keyless D-box or a keyless S-box can be made invertible because they have one input and one output. An exclusive operation is a binary operation. The inverse of an exclusive-or operation can make sense only if one of the inputs is fixed (is the same in encryption and decryption). For example, if one of the inputs is the key, which normally is the same in encryption and decryption, then an exclusive-or operation is self-invertible, as shown in Fig. 5.9.

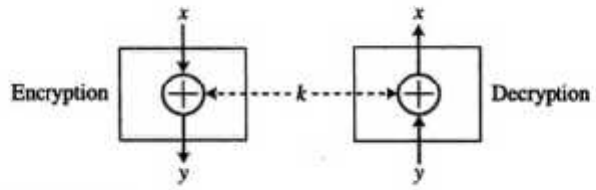


Fig. 5.9 *Invertibility of the exclusive-or operation*

Properties The five properties of the exclusive-or operation in the $\mathbf{GF}(2^n)$ field makes this operation a very interesting component for use in a block cipher.

1. *Closure:* This property guarantees that the result of exclusive-oring two n-bit words is another n-bit word.
2. *Associativity:* This property allows us to use more than one exclusive-or operator in any order.

$$x \oplus (y \oplus z) \quad \leftrightarrow \quad (x \oplus y) \oplus z$$

3. *Commutativity:* This property allows us to swap the inputs without affecting the output.

$$x \oplus y \quad \leftrightarrow \quad y \oplus x$$

4. *Existence of identity:* The identity element for the exclusive-or operation is an n-bit word that consists of all 0's, or (00...0). This implies that exclusive-oring of a word with the identity element does not change that word.

Invertibility A circular left-shift operation is the inverse of the circular right-shift operation. If one is used in the encryption cipher, the other can be used in the decryption cipher.

Property The circular shift operation has two properties that we need to be aware of. First, the shifting is modulo n. In other words, if $k = 0$ or $k = n$, there is no shifting. If k is larger than n, then the input is shifted k mod n bits. Second, the circular shift operation under the composition operation is a group. This means that shifting a word more than once is the same as shifting it only once.

Swap The **swap operation** is a special case of the circular shift operation where $k = n/2$. This means this operation is valid only if n is an even number. Because left-shifting $n/2$ bits is the same as right-shifting $n/2$, this component is self-invertible. A swap operation in the encryption cipher can be totally canceled by a swap operation in the decryption cipher. Figure 5.11 shows the swapping operation for an 8-bit word.

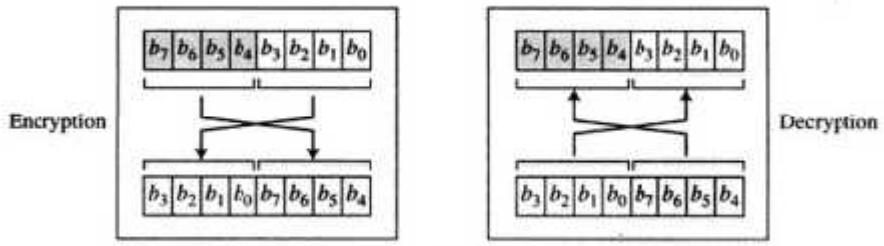
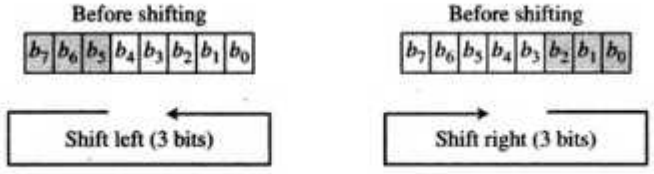


Fig. 5.11 *Swap operation on an 8-bit word*

Circular Shift

Another component found in some modern block ciphers is the **circular shift operation.** Shifting can be to the left or to the right. The circular left-shift operation shifts each bit in an n-bit word k positions to the left; the leftmost k bits are removed from the left and become the rightmost bits. The circular right-shift operation shifts each bit in an n-bit word k positions to the right; the rightmost k bits are removed from the right and become the leftmost bits. Figure 5.10 shows both left and right operations in the case where $n = 8$ and $k = 3$.



Split and Combine Two other operations found in some block ciphers are split and combine. The **split operation** normally splits an n-bit word in the middle, creating two equal-length words. The **combine operation** normally concatenates two equal-length words to create an n-bit word. These two operations are inverses of each other and can be used as a pair to cancel each other out. If one is used in the encryption cipher, the other is used in the decryption cipher. Figure 5.12 shows the two operations in the case where $n = 8$.

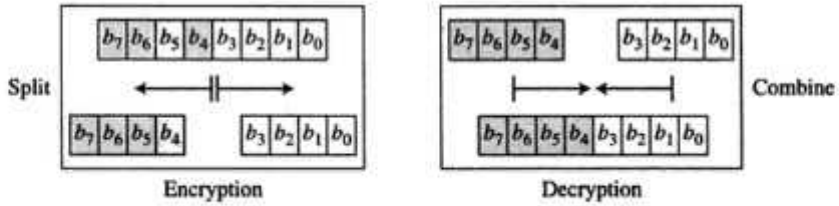


Fig. 5.12 *Split and combine operations on an 8-bit word*

- **Product ciphers:**

Shannon introduced the concept of a **product cipher.** A product cipher is a complex cipher combining substitution, permutation, and other components discussed in previous sections.

Diffusion and Confusion Shannon's idea in introducing the product cipher was to enable the block ciphers to have two important properties: diffusion and confusion. The idea of **diffusion** is to hide the relationship between the ciphertext and the plaintext. This will frustrate the adversary who uses ciphertext statistics to find the plaintext. Diffusion implies that each symbol (character or bit) in the ciphertext is dependent on some or all symbols in the plaintext. In other words, if a single symbol in the plaintext is changed, several or all symbols in the ciphertext will also be changed.

Diffusion hides the relationship between the ciphertext and the plaintext.

The idea of **confusion** is to hide the relationship between the ciphertext and the key. This will frustrate the adversary who tries to use the ciphertext to find the key. In other words, if a single bit in the key is changed, most or all bits in the ciphertext will also be changed.

Confusion hides the relationship between the ciphertext and the key.

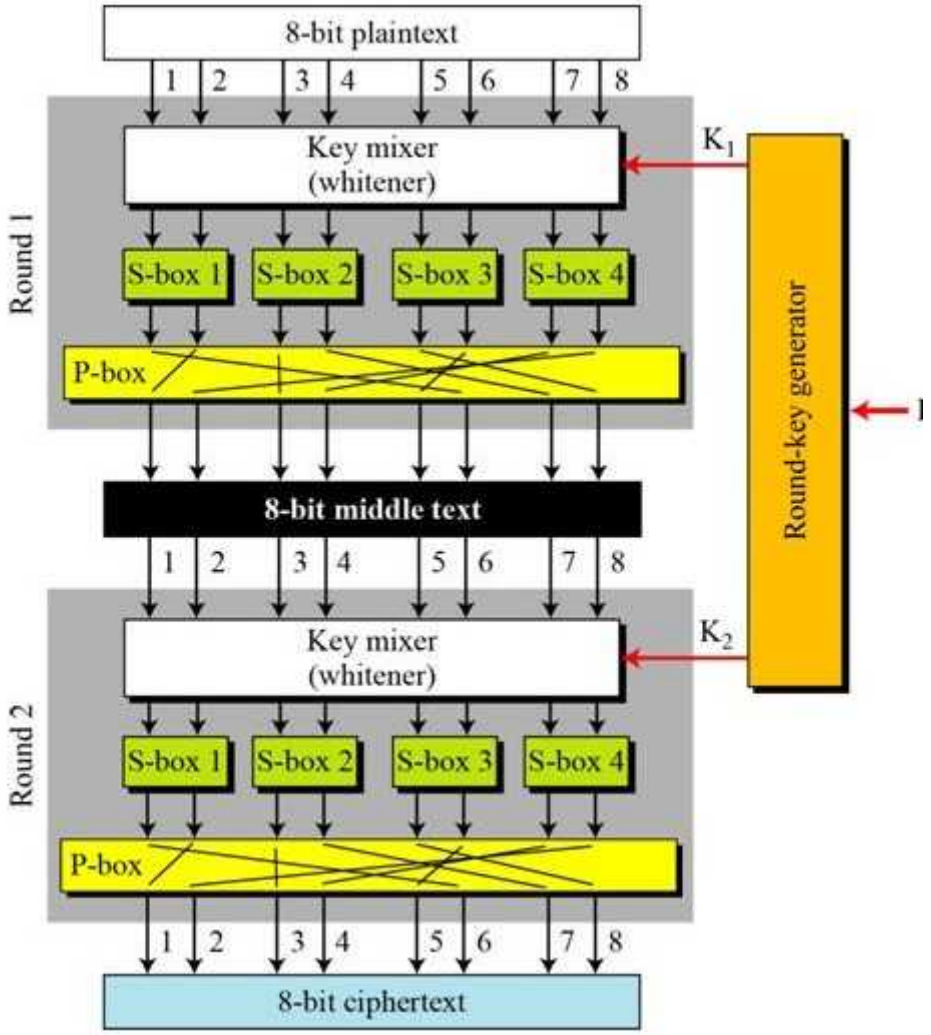
Rounds Diffusion and confusion can be achieved using iterated product ciphers where each iteration is a combination of S-boxes, D-boxes, and other components. Each iteration is referred to as a **round.** The block cipher uses a **key schedule** or **key generator** that creates different keys for each round from the cipher key. In an N-round cipher, the plaintext is encrypted N times to create the ciphertext; the ciphertext is decrypted N times to create the plaintext. We refer to the text created at the intermediate levels (between two rounds) as the middle text. Figure 5.13 shows a simple product cipher with two rounds. In practice, product ciphers have more than two rounds.

In Fig. 5.13, three transformations happen at each round:

a. The 8-bit text is mixed with the key to whiten the text (hide the bits using the key). This is normally done by exclusive-oring the 8-bit word with the 8-bit key.

b. The outputs of the whitener are organized into four 2-bit groups and are fed into four S-boxes. The values of bits are changed based on the structure of the S-boxes in this transformation.

c. The outputs of S-boxes are passed through a D-box to transpose the bits so that in the next round each box receives different inputs. Since this is a straight D-box and has to be invertible (for the purpose of decryption), so it is referred to as a permutation (P-box) and is denoted as P

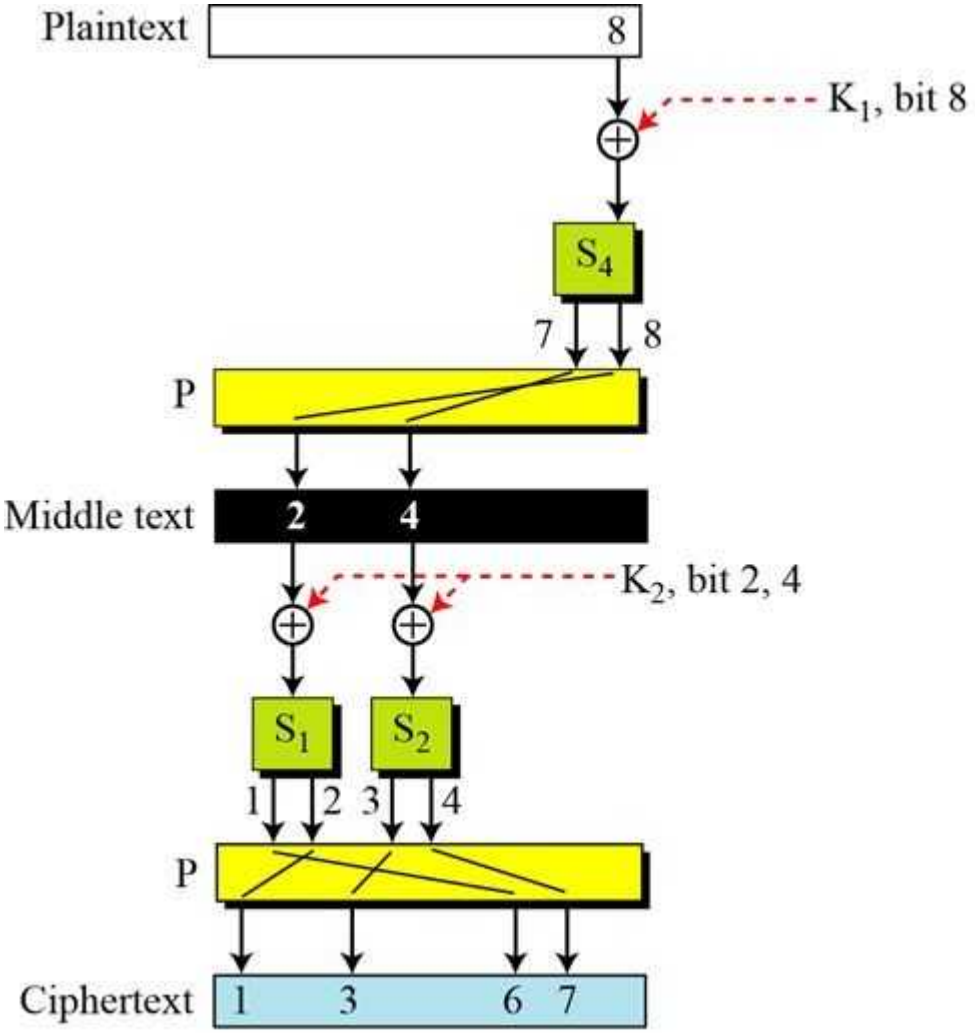


A product cipher mode of two rounds

Diffusion The primitive design of Fig. 5.13 shows how a product with the combination of S-boxes and D-boxes can guarantee diffusion. Figure 5.14 shows how changing a single bit in the plaintext affects many bits in the ciphertext.

a. In the first round, bit 8, after being exclusive-ored with the corresponding bit of K_1, affects two bits (bits 7 and 8) through S-box 4. Bit 7 is permuted and becomes bit 2; bit 8 is permuted and becomes bit 4. After the first round, bit 8 has affected bits 2 and 4. In the second round, bit 2, after being exclusive-ored with the corresponding bit of K_2, affects two bits (bits 1 and 2) through S-box 1. Bit 1 is permuted and becomes bit 6; bit 2 is permuted and becomes bit 1. Bit 4, after being exclusive-ored with the corresponding bit in K_2, affects bits 3 and 4. Bit 3 remains the same; bit 4 is permuted and becomes bit 7. After the second round, bit 8 has affected bits 1, 3, 6, and 7.

b. Going through these steps in the other direction (from ciphertext to the plaintext) shows that each bit in the ciphertext is affected by several bits in the plaintext.



Diffusion and confusion in a block cipher

Confusion Figure 5.14 also shows us how the confusion property can be achieved through the use of a product cipher. The four bits of ciphertext, bits 1, 3, 6, and 7, are affected by three bits in the key (bit 8 in K_1 and bits 2 and 4 in K_2). Going through the steps in the other direction shows that each bit in each round key affects several bits in the ciphertext. The relationship between ciphertext bits and key bits is obscured.

Practical Ciphers To improve diffusion and confusion, practical ciphers use larger data blocks, more S-boxes, and more rounds. With some thought, it can be seen that increasing the number of rounds using more S-boxes may create a better cipher in which the ciphertext looks more and more like a random *n*-bit word. In this way, the relationship between ciphertext and plaintext is totally hidden (diffusion). Increasing the number of rounds increases the number of round keys, which better hides the relationship between the ciphertext and the key.

5.1.6 Two Classes of Product Ciphers

Modern block ciphers are all product ciphers, but they are divided into two classes. The ciphers in the first class use both invertible and noninvertible components. The ciphers in this class are normally referred to as Feistel ciphers. The block cipher DES discussed in Chapter 6 is a good example of a Feistel cipher. The ciphers in the second class use only invertible components. We refer to ciphers in this class as non-Feistel ciphers (for the lack of another name). The block cipher AES discussed in Chapter 7 is a good example of a non-Feistel cipher.

Feistel Ciphers

Feistel designed a very intelligent and interesting cipher that has been used for decades. A **Feistel cipher** can have three types of components: self-invertible, invertible, and noninvertible. A Feistel cipher combines all noninvertible elements in a unit and uses the same unit in the encryption and decryption algorithms. The question is how the encryption and decryption algorithms are inverses of each other if each has a non-invertible unit. Feistel showed that they can be canceled out.

First Thought To better understand the Feistel cipher, let us see how we can use the same noninvertible component in the encryption and decryption algorithms. The effects of a noninvertible component in the encryption algorithm can be canceled in the decryption algorithm if we use an exclusive-or operation, as shown in Fig. 5.15.

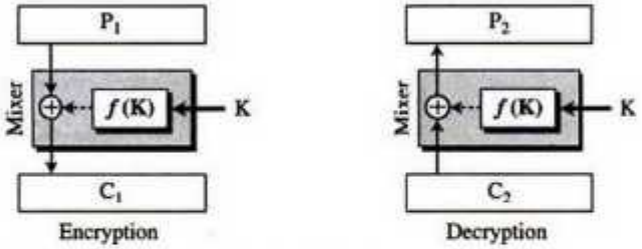


Fig. 5.15 *The first thought in Feistel cipher design*

In the encryption, a noninvertible function, $f(K)$, accepts the key as the input. The output of this component is exclusive-ored with the plaintext. The result becomes the ciphertext. We call the combination of the function and the exclusive-or operation the **mixer** (for lack of another name). The mixer plays an important role in the later development of the Feistel cipher.

Because the key is the same in encryption and decryption, we can prove that the two algorithms are inverses of each other. In other words, if $C_2 = C_1$ (no change in the ciphertext during transmission), then $P_2 = P_1$.

Encryption: $C_1 = P_1 \oplus f(K)$

Decryption: $P_2 = C_2 \oplus f(K) = C_1 \oplus f(K) = P_1 \oplus f(K) \oplus f(K) = P_1 \oplus (00...0) = P_1$

Note that two properties of exclusive-or operation have been used (existence of inverse and existence of identity).

The above argument proves that, although the mixer has a noninvertible element, the mixer itself is self-invertible.

The mixer in the Feistel design is self-invertible.

Example 5.12 This is a trivial example. The plaintext and ciphertext are each 4 bits long and the key is 3 bits long. Assume that the function takes the first and third bits of the key, interprets these two bits as a decimal number, squares the number, and interprets the result as a 4-bit binary pattern. Show the results of encryption and decryption if the original plaintext is 0111 and the key is 101.

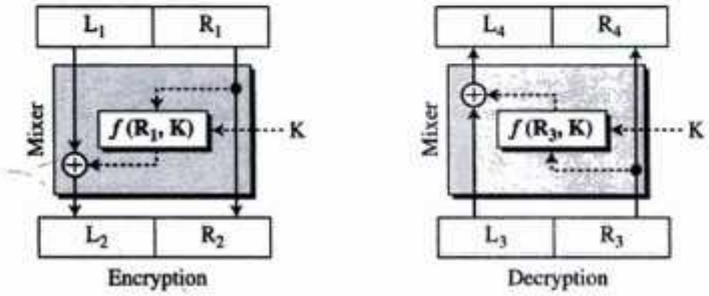
Solution The function extracts the first and second bits to get 11 in binary or 3 in decimal. The result of squaring is 9, which is 1001 in binary.

Encryption: $C = P \oplus f(K) = 0111 \oplus 1001 = 1110$

Decryption: $P = C \oplus f(K) = 1110 \oplus 1001 = 0111$ Same as the original P

The function $f(101) = 1001$ is noninvertible, but the exclusive-or operation allows us to use the function in both encryption and decryption algorithms. In other words, the function is noninvertible, but the mixer is self-invertible.

Improvement Let us improve on our first thought to get closer to the Feistel cipher. We know that we need to use the same input to the noninvertible element (the function), but we don't want to use only the key. We want the input to the function to also be part of the plaintext in the encryption and part of the ciphertext in the decryption. The key can be used as the second input to the function. In this way, our function can be a complex element with some keyless elements and some keyed elements. To achieve this goal, divide the plaintext and the ciphertext into two equal-length blocks, left and right. We call the left block L and the right block R. Let the right block be the input to the function, and let the left block be exclusive-ored with the function output. We need to remember one important point: the inputs to the function must be exactly the same in encryption and decryption. This means that the right section of plaintext in the encryption and the right section of the ciphertext in the decryption must be the same. In other words, the right section must go into and come out of the encryption and decryption processes unchanged. Figure 5.16 shows the idea.



The encryption and decryption algorithms are still inverses of each other. Assume that $L_3 = L_2$ and $R_3 = R_2$ (no change in the ciphertext during transmission).

$$R_4 = R_3 = R_2 = R_1$$
$$L_4 = L_3 \oplus f(R_3, K) = L_2 \oplus f(R_2, K) = L_1 \oplus f(R_1, K) \oplus f(R_1, K) = L_1$$

Final Design The preceding improvement has one flaw. The right half of the plaintext never changes. Eve can immediately find the right half of the plaintext by intercepting the ciphertext and extracting the right half of it. The design needs more improvement. First, increase the number of rounds. Second, add a new element to each round: a **swapper.** The effect of the swapper in the encryption round is canceled by the effect of the swapper in the decryption round. However, it allows us to swap the left and right halves in each round. Figure 5.17 shows the new design with two rounds.

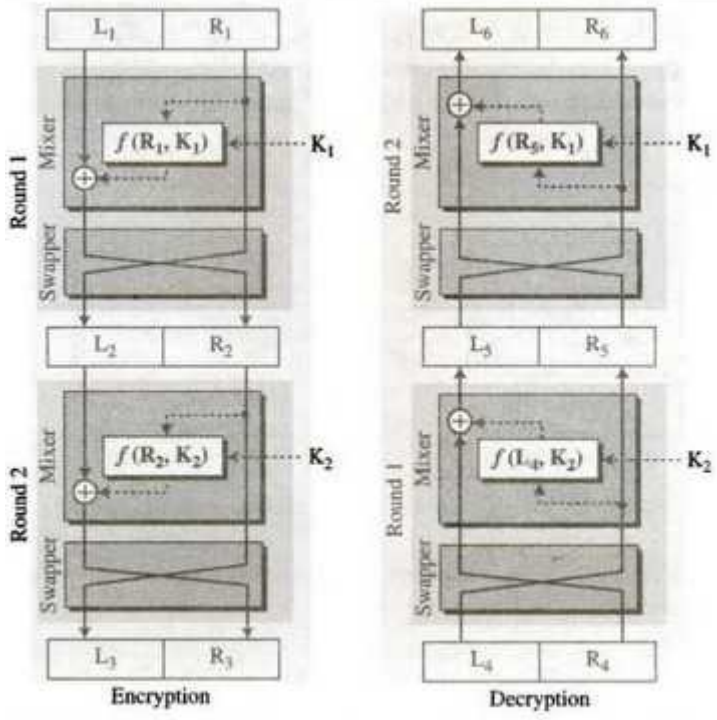
Note that there are two round keys, K_1 and K_2. The keys are used in reverse order in the encryption and decryption.

Because the two mixers are inverses of each other, and the swappers are inverses of each other, it should be clear that the encryption and decryption ciphers are inverses of each other. However, let us see if we can prove this fact using the relationship between the left and right sections in each cipher. In other words, let us see if $L_6 = L_1$ and $R_6 = R_1$, assuming that $L_4 = L_3$ and $R_4 = R_3$ (no change in the ciphertext during transmission). We first prove the equality for the middle text.

$$L_5 = R_4 \oplus f(L_4, K_2) = R_3 \oplus f(R_2, K_2) = L_2 \oplus f(R_2, K_2) \oplus f(R_2, K_2) = L_2$$
$$R_5 = L_4 = L_3 = R_2$$

Then it is easy to prove that the equality holds for two plaintext blocks.

$$L_6 = R_5 \oplus f(L_5, K_1) = R_2 \oplus f(L_2, K_1) = L_1 \oplus f(R_1, K_1) \oplus f(R_1, K_1) = L_1$$
$$R_6 = L_5 = L_2 = R_1$$



Non-Feistel Ciphers

A **non-Feistel cipher** uses only invertible components. A component in the plaintext has the corresponding component in the cipher. For example, S-boxes need to have an equal number of inputs and outputs to be compatible. No compression or expansion D-boxes are allowed, because they are not invertible. In a non-Feistel cipher, there is no need to divide the plaintext into two halves as we saw in the Feistel ciphers. Figure 5.13 can be thought of as a non-Feistel cipher because the only components in each round are the exclusive-or operation (self-invertible), 2×2 S-boxes that can be designed to be invertible, and a straight D-box that is invertible using the appropriate permutation table. Because each component is invertible, it can be shown that each round is invertible. We only need to use the round keys in the reverse order. The encryption uses round keys K_1 and K_2. The decryption algorithm needs to use round keys K_2 and K_1.

Differential Cryptanalysis

Eli Biham and Adi Shamir introduced the idea of **differential cryptanalysis.** This is a *chosen-plaintext* attack; Eve can somehow access Alice's computer, submitting chosen plaintext and obtaining the corresponding ciphertext. The goal is to find Alice's cipher key.

Algorithm Analysis Before Eve uses the chosen-plaintext attack, she needs to analyze the encryption algorithm in order to collect some information about plaintext-ciphertext relationships. Obviously, Eve does not know the cipher key. However, some ciphers have weaknesses in their structures that can allow Eve to find a relationship between the plaintext differences and ciphertext differences leaking information about the key.

Launching a Chosen-Plaintext Attack After the analysis, which can be done once and kept for future uses as long as the structure of the cipher does not change, Eve can choose the plaintexts for attacks. The differential probability distribution table (Table 5.5) helps Eve choose plaintexts that have the highest probability in the table.

Guessing the Key Value After launching some attacks with appropriate chosen plaintexts, Eve can find some plaintext-ciphertext pairs that allow her to guess the value of the key. This step starts from C and makes toward P.

General Procedure Modern block ciphers have more complexity than we discussed in this section. In addition, they are made from different rounds. Eve can use the following strategy:

1. Because each round is the same, Eve can create a differential distribution table (XOR profile) for each S-box and combine them to create the distribution for each round.
2. Assuming that each round is independent (a fair assumption), Eve can create a distribution table for the whole cipher by multiplying the corresponding probabilities.
3. Eve now can make a list of plaintexts for attacks based on the distribution table in step 2. Note that the table in step 2 only helps Eve choose a smaller number of ciphertext-plaintext pairs.

Differential cryptanalysis is based on a nonuniform differential distribution table of the S-boxes in a block cipher.

A more detailed differential cryptanalysis is given in Appendix N.

Attack on Block Ciphers:

Linear Cryptanalysis

Linear cryptanalysis was presented by Mitsuru Matsui in 1993. The analysis uses *known-plaintext* attacks (versus the chosen-plaintext attacks in differential cryptanalysis). To see the main idea behind the attack, assume that the cipher is made of a single round, as shown in Fig. 5.20, where c_0, c_1, and c_2 represent the three bits in the output and x_0, x_1, and x_2 represent the three bits in the input of the S-box.

The S-box is a linear transformation in which each output is a linear function of input, as we discussed earlier in this chapter. With this linear component, we can create three linear equations between plaintext and ciphertext bits, as shown below:

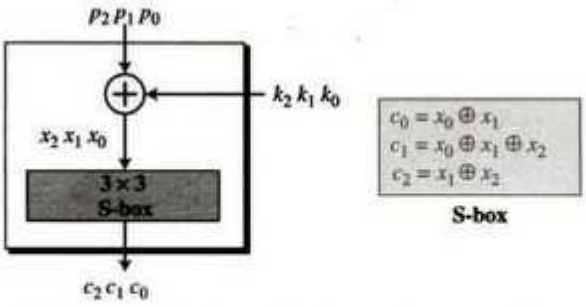


Fig. 5.20 *A simple cipher with a linear S-box*

$$c_0 = p_0 \oplus k_0 \oplus p_1 \oplus k_1$$
$$c_1 = p_0 \oplus k_0 \oplus p_1 \oplus k_1 \oplus p_2 \oplus k_2$$
$$c_2 = p_1 \oplus k_1 \oplus p_2 \oplus k_2$$

Solving for three unknowns, we get

$$k_1 = (p_1) \oplus (c_0 \oplus c_1 \oplus c_2)$$
$$k_2 = (p_2) \oplus (c_0 \oplus c_1)$$
$$k_0 = (p_0) \oplus (c_1 \oplus c_2)$$

- **Modern Stream ciphers:-**

In Chapter 3 we briefly discussed the difference between traditional stream ciphers and tradition block ciphers. Similar differences exist between modern stream ciphers and modern block ciphers. In a **modern stream cipher,** encryption and decryption are done r bits at a time. We have a plaintext bit stream P = $p_n \ldots p_2 p_1$, a ciphertext bit stream C = $c_n \ldots c_2 c_1$, and a key bit stream K = $k_n \ldots k_2 k_1$, in which p_i , c_i , and k_i are r-bit words. Encryption is $c_i = \mathrm{E}\ (k_i, \mathrm{p_i})$, and decryption is $p_i = \mathrm{D}\ (k_i, c_i)$, as shown in Fig. 5.21.

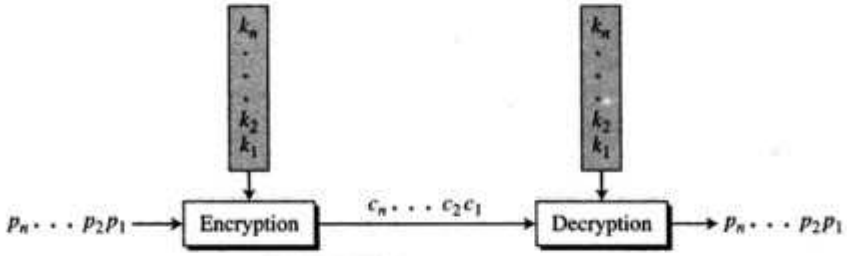


Fig. 5.21 *Stream cipher*

Stream ciphers are faster than block ciphers. The hardware implementation of a stream cipher is also easier. When we need to encrypt binary streams and transmit them at a constant rate, a stream cipher is the better choice to use. Stream ciphers are also more immune to the corruption of bits during transmission.

5.2.1 Synchronous Stream Ciphers

In a **synchronous stream cipher,** the key stream is independent of the plaintext or ciphertext stream. The key stream is generated and used with no relationship between key bits and the plaintext or ciphertext bits.

One-Time Pad The simplest and the most secure type of synchronous stream cipher is called the **one-time pad,** which was invented and patented by Gilbert Vernam. A one-time pad cipher uses a key stream that is randomly chosen for each encipherment. The encryption and decryption algorithms each use a single exclusive-or operation. Based on properties of the exclusive-or operation discussed earlier, the encryption and decryption algorithms are inverses of each other. It is important to note that in this cipher the exclusive-or operation is used one bit at a time. In other words, the operation is over 1-bit word and the field is **GF**(2). Note also that there must be a secure channel so that Alice can send the key stream sequence to Bob (Fig. 5.22).

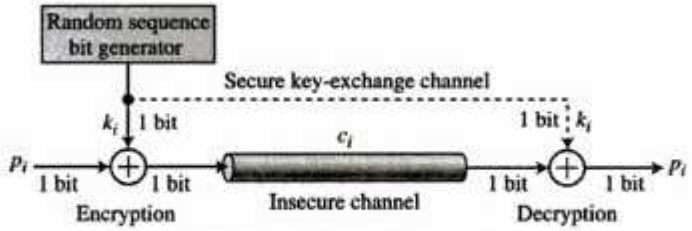


Fig. 5.22 *One-time pad*

The one-time pad is an ideal cipher. It is perfect. There is no way that an adversary can guess the key or the plaintext and ciphertext statistics. There is no relationship between the plaintext and ciphertext, either. In other words, the ciphertext is a true random stream of bits even if the plaintext contains some patterns. Eve cannot break the cipher unless she tries all possible random key streams, which would be 2^n if the size of the plaintext is n bits. However, there is an issue here. How can the sender and the receiver share a one-time pad key each time they want to communicate? They need to somehow agree on the random key. If there exists such a secured channel, they can use that to communicate the plaintext itself. So this perfect and ideal cipher is impractical.

Feedback Shift Register One compromise to the one-time-pad is the **feedback shift register (FSR).** An FSR can be implemented in either software or hardware, but the hardware implementation is easier to discuss. A feedback shift register is made of a **shift register** and a **feedback function,** as shown in Fig. 5.23.

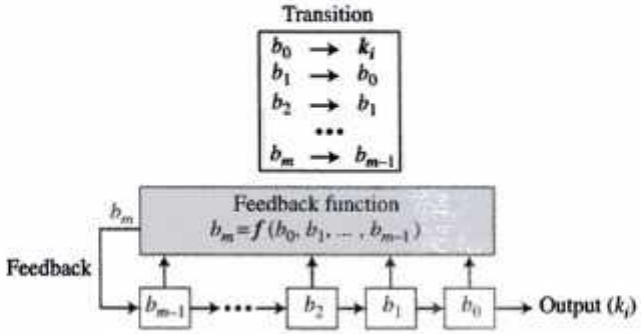


Fig. 5.23 *Feedback shift register (FSR)*

The shift register is a sequence of m cells, b_0 to b_{m-1}, where each cell holds a single bit. The cells are initialized to an m-bit word, called the initial value or the **seed.** Whenever an output bit is needed (for example, in a click of time), every bit is shifted one cell to the right, which means that each cell gives its value to the cell to its right and receives the value of the cell to its left. The rightmost cell, b_0, gives its value as output (k_i); the leftmost cell, b_{m-1}, receives its value from the feedback function. We call the output of the feedback function b_m. The feedback function defines how the values of cells are combined to calculate b_m. A feedback shift register can be linear or nonlinear.

Linear Feedback Shift Register In a **linear feedback shift register (LFSR),** b_m is a linear function of $b_0, b_1, \ldots, b_{m-1}$.

$$b_m = c_{m-1}\, b_{m-1} + \cdots + c_2\, b_2 + c_1\, b_1 + c_0\, b_0 \; (c_0 \neq 0)$$

However, we are dealing with binary digits because the multiplication and addition are in the **GF**(2) field, so the value of c_i is either 1 or 0, but c_0 should be 1 to get a feedback from the output. The addition operation is also the exclusive-or operation. In other words,

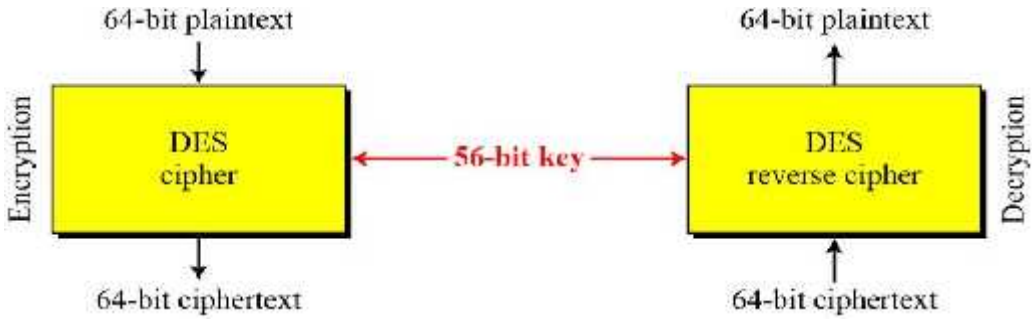
$$b_m = c_{m-1}\, b_{m-1} \oplus \cdots \oplus c_2\, b_2 \oplus c_1\, b_1 \oplus c_0\, b_0 \; (c_0 \neq 0)$$

- **Topic :03**

DES(DATA ENCRYPTION STANDARD) :

- The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).
- DES was published in the *Federal Register* in March 1975 as a draft of the **Federal Information Processing Standard (FIPS).**
- DES has been the most widely used symmetric-key block cipher since its publication. NIST later issued a new standard (FIPS 46-3) that recommends the use of triple DES (repeated DES cipher three times) for future applications.

- **Overview**

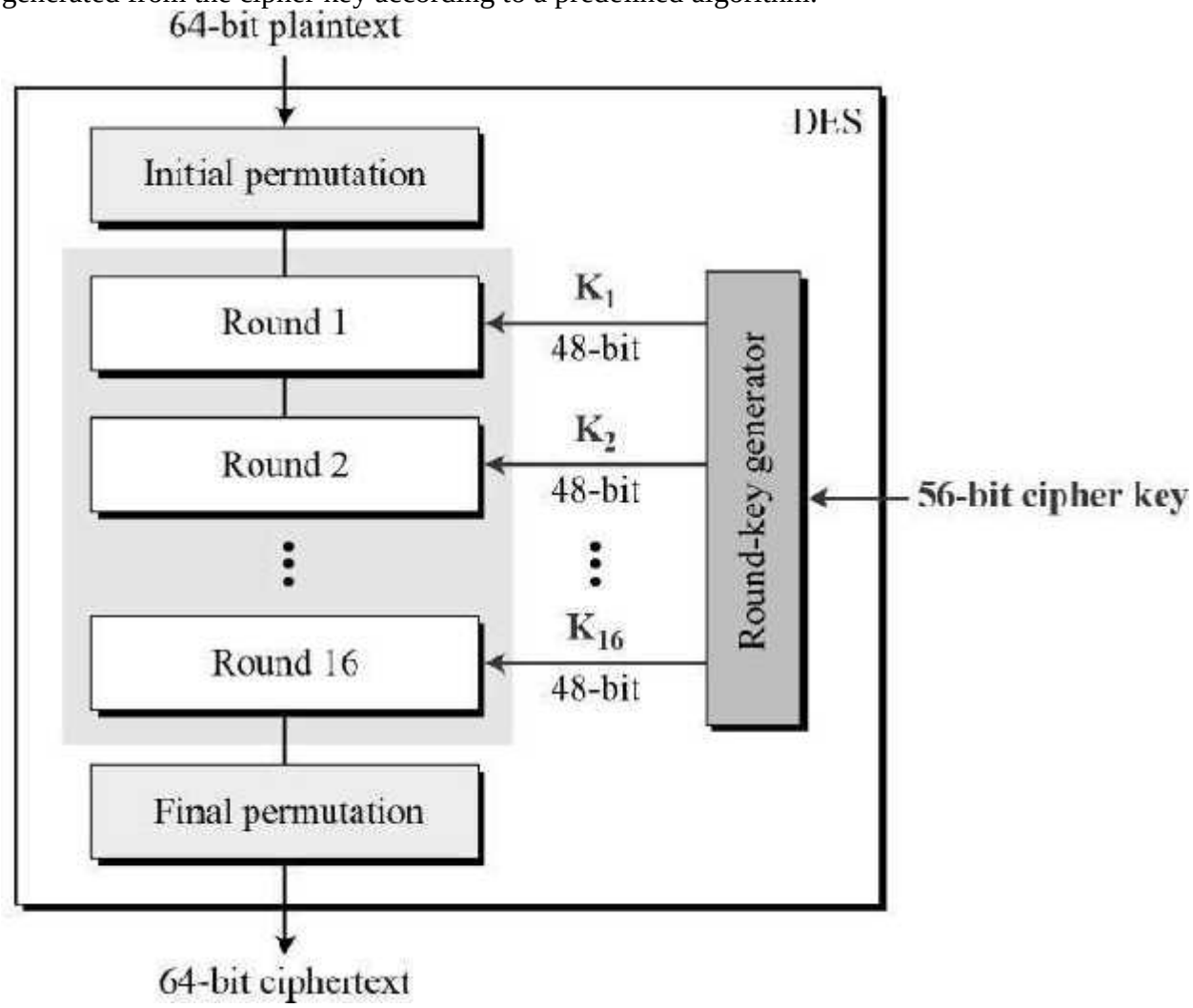


At the encryption site, DES takes a 64-bit plaintext and creates a 64-bit ciphertext; at the decryption site, DES takes a 64-bit ciphertext and creates a 64-bit block of plaintext. The same 56-bit cipher key is used for both encryption and decryption.

DES structure:

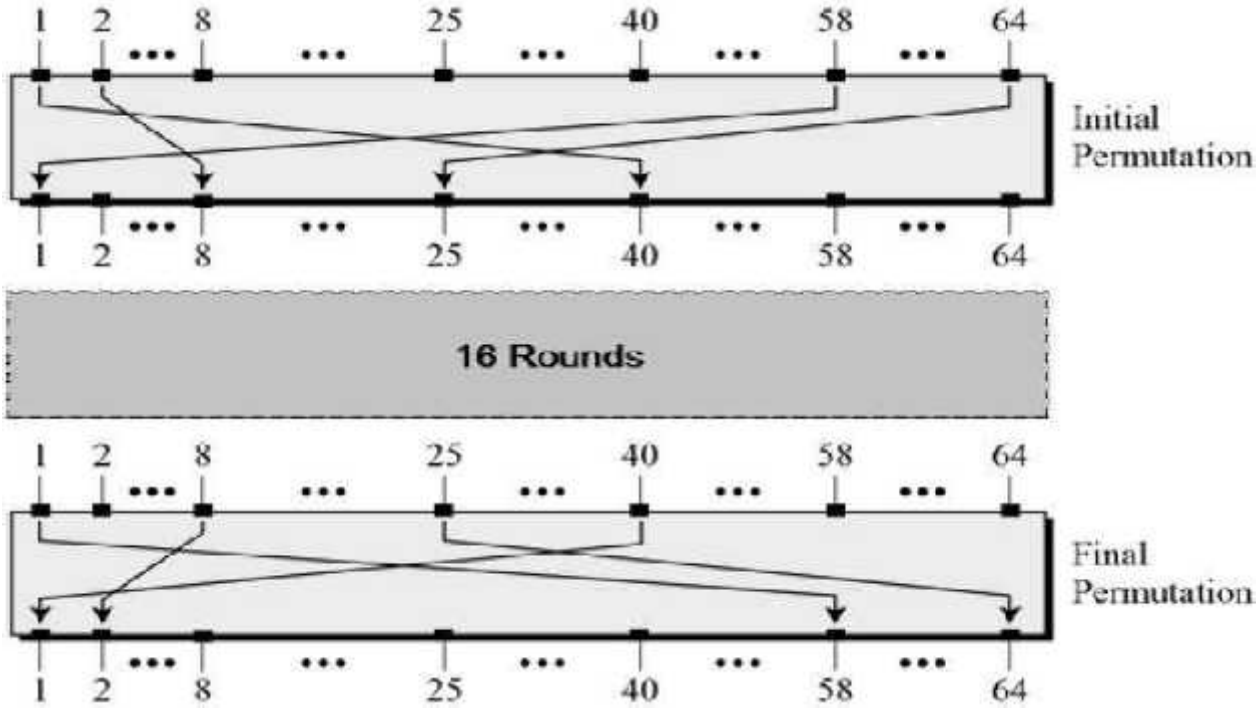
The encryption process is made of two permutations (P-boxes), which we call initial and final

permutations, and sixteen Feistel rounds. Each round uses a different 48-bit round key generated from the cipher key according to a predefined algorithm.



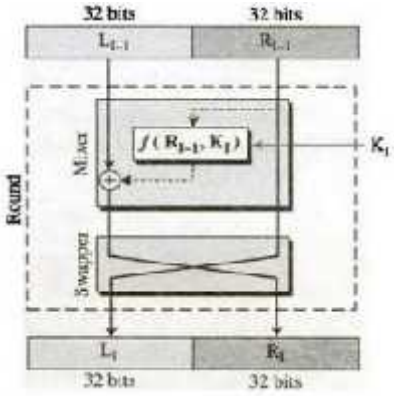
Initial and final permutations:

- The below diagram shows the initial and final permutations (P-boxes). Each of these permutations takes a 64-bit input and permutes them according to a predefined rule.
- We have shown only a few input ports and the corresponding output ports.
- These permutations are keyless straight permutations that are the inverse of each other.
- For example, in the initial permutation, the 58th bit in the input becomes the first bit in the output.
- Similarly, in the final permutation, the first bit in the input becomes the 58th bit in the output. In other words, if the rounds between these two permutations do not exist, the 58th bit entering the initial permutation is the same as the 58th bit leaving the final permutation.



Rounds:

- DES uses 16 rounds. Each round of DES is a Feistel cipher.
- The round takes L_{i-1} and R_{i-1} from previous round (or the initial permutation box) and creates L_i and R_i, which go to the next round (or final permutation box).
- we can assume that each round has two cipher elements (mixer and swapper). Each of these elements is invertible.
- The swapper is obviously invertible. It swaps the left half of the text with the right half. The mixer is invertible because of the XOR operation.
- All noninvertible elements are collected inside the function $\boldsymbol{f}(R_{i-1}, K_i)$.

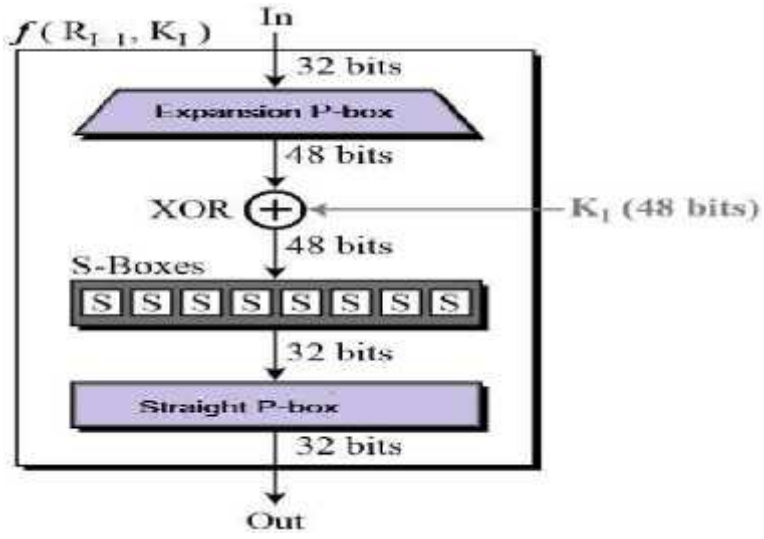


DES function:

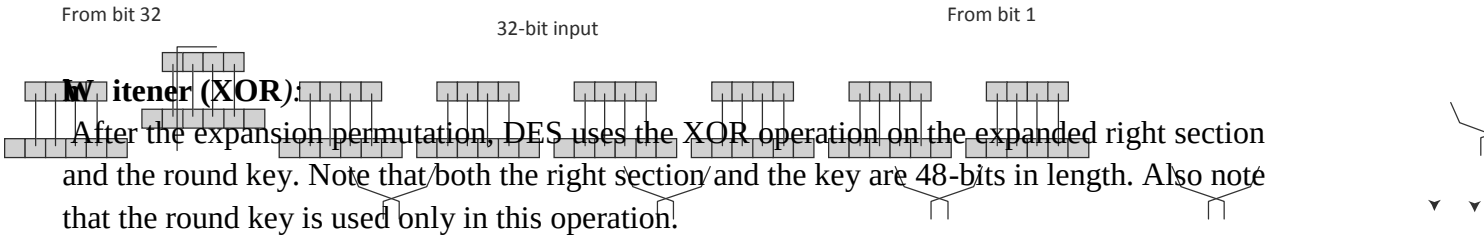
- The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits (R_{i-1}) to produce a 32-bit output.

- This function is made up of four sections: an expansion D-box, a whitener (that adds key), a group of S-boxes, and a straight D-box.

Expansion D-box:



- Since RI-1 is a 32-bit input and KI is a 48-bit key, we first need to expand RI-1 to 48 bits. RI-1 is divided into 8 4-bit sections. Each 4-bit section is then expanded to 6 bits. This expansion permutation follows a predetermined rule.
- For each section, input bits 1, 2, 3, and 4 are copied to output bits 2, 3, 4, and 5, respectively.
- Output bit 1 comes from bit 4 of the previous section; output bit 6 comes from bit 1 of the next section.
- If sections 1 and 8 can be considered adjacent sections, the same rule ap- plies to bits 1 and 32. The below figure shows the input and output in the expansion permutation.

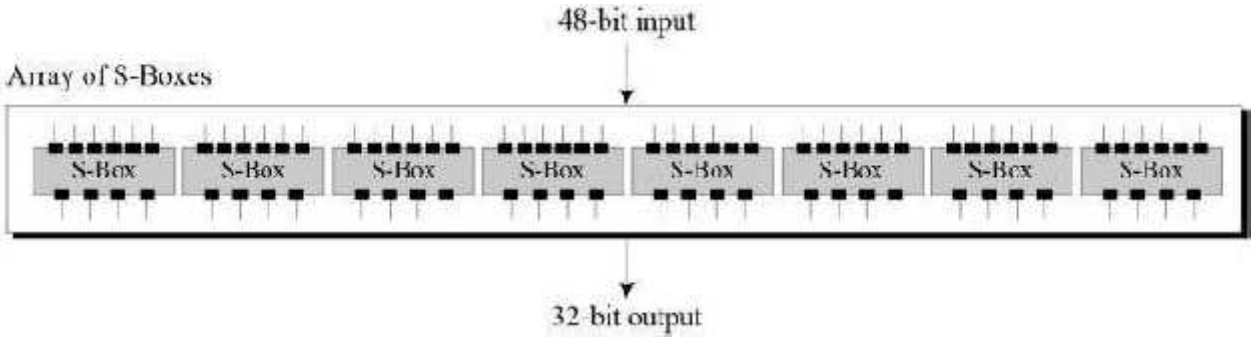


Whitener (XOR):

After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key. Note that both the right section and the key are 48-bits in length. Also note that the round key is used only in this operation.

S-Boxes:

The S-boxes do the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output.



- The 48-bit data from the second operation is divided into eight 6-bit chunks, and each chunk is fed into a box.
- The result of each box is a 4-bit chunk; when these are combined the result is a 32-bit text.
- The substitution in each box follows a pre-determined rule based on a 4-row by 16- column table.
- The combination of bits 1 and 6 of the input defines one of four rows; the combination of bits 2 through 5 defines one of the sixteen columns as shown in Fig s-box 1:

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

s-box 2:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

s-box 3:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

s-box 4:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	07	13	14	03	00	6	09	10	1	02	08	05	11	12	04	15
1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04

3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

s-box 5:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

s-box 6:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
3	04	03	02	12	09	05	15	10	11	14	01	07	10	00	08	13

s-box 7:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	00	08	13	03	12	09	07	05	10	06	01
1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

ox 8:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	10	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	10	15	03	05	08
3	02	01	14	07	04	10	8	13	15	12	09	09	03	05	06	11

Final Permutation:

- The last operation in the DES function is a permutation with a 32-bit input and a 32-bit output.
- The input/output relationship for this operation is shown in Table 6.11 and follows the same general rule as previous tables.
- For example, the seventh bit of the input becomes the second bit of the output.

Table 6.11 *Straight permutation table*

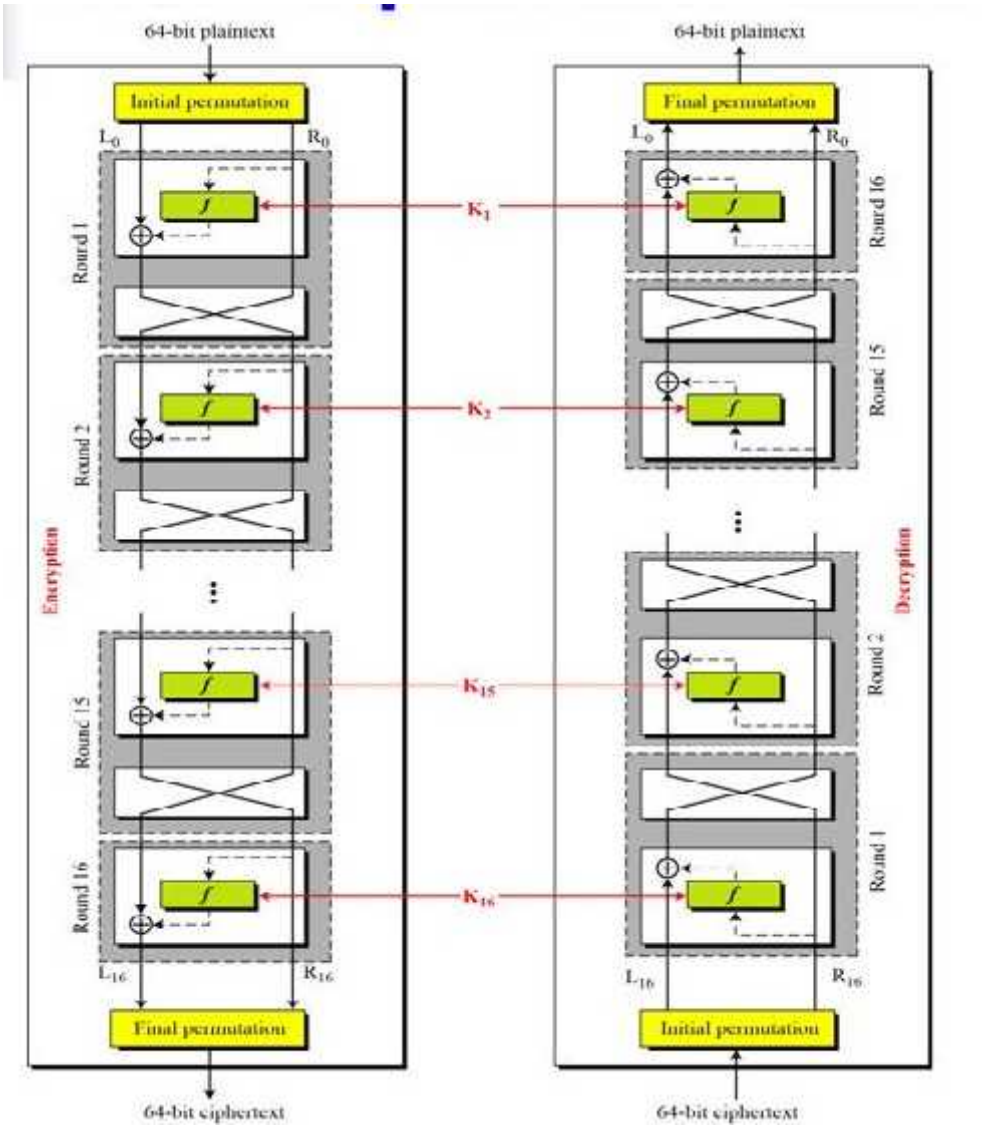
16	**07**	**20**	**21**	**29**	**12**	**28**	**17**
01	**15**	**23**	**26**	**05**	**18**	**31**	**10**
02	**08**	**24**	**14**	**32**	**27**	**03**	**09**

19	**13**	**30**	**06**	**22**	**11**	**04**	**25**

Cipher and Reverse Cipher:

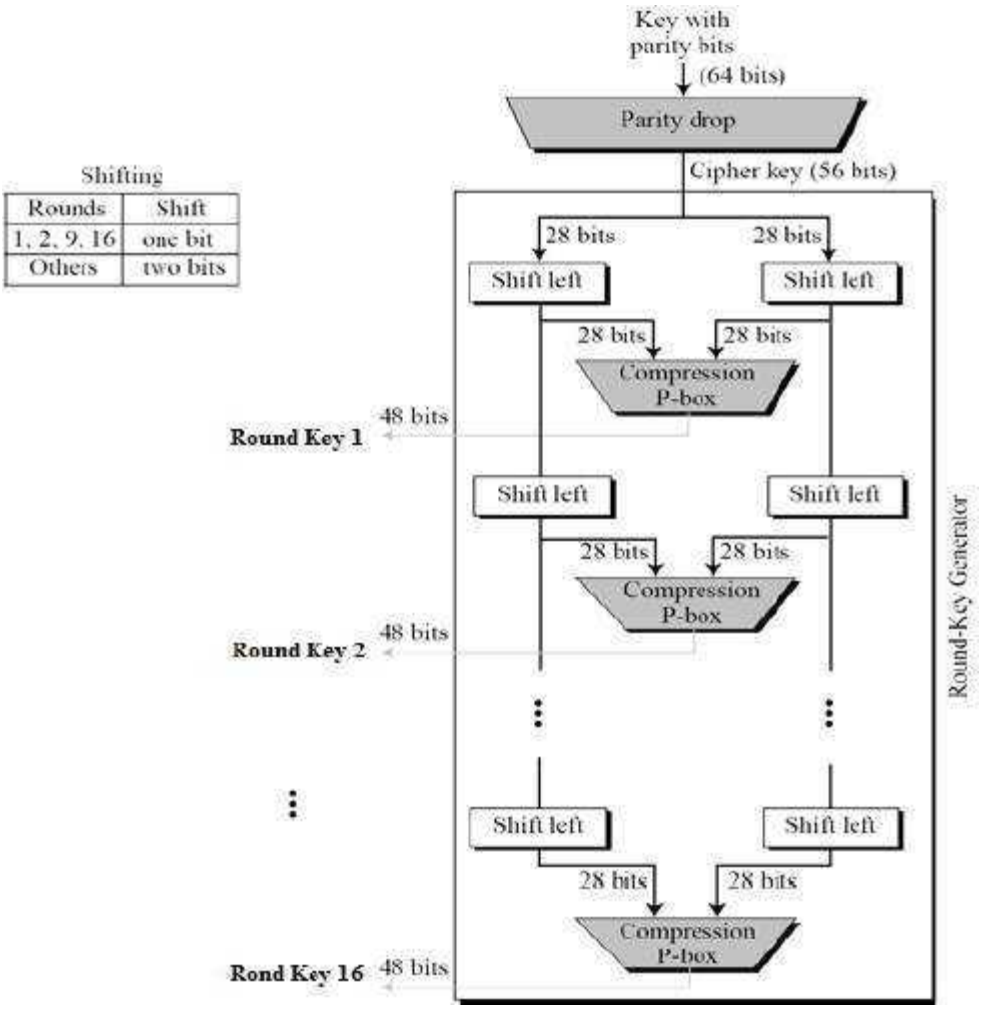
First Approach:

- To achieve this goal, one approach is to make the last round (round 16) different from the others; it has only a mixer and no swapper.
- Although the rounds are not aligned, the elements (mixer or swapper) are aligned.
- We proved in Chapter 5 that a mixer is a self-inverse; so is a swapper.
- The final and initial permutations are also inverses of each other.
- The left section of the plaintext at the encryption site, L_0, is enciphered as L_{16} at the encryption site; L_{16} at the decryption is deciphered as L_0 at the decryption site. The situation is the same with R_0 and R_{16}.



Key Generation :

The **round-key generator** creates sixteen 48-bit keys out of a 56-bit cipher key. However, the cipher key is normally given as a 64-bit key in which 8 extra bits are the parity bits, which are dropped before the actual key-generation process, as shown in Fig.



Parity Drop :

- The preprocess before key expansion is a compression transposition step that we call parity bit drop.
- It drops the parity bits (bits 8, 16, 24, 32, …64) from the 64-bit key and permutes the rest of the bits according to Table
- The remaining 56-bit value is the actual cipher key which is used to generate round keys. The parity drop step (a compression D-box) is shown in Table

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Shift Left :

After the straight permutation, the key is divided into two 28-bit parts. Each part is shifted left (circular shift) one or two bits. In rounds 1, 2, 9, and 16, shifting is one bit; in the other rounds, it is two bits. The two parts are then combined to form a 56-bit part. Table 6.13 shows the number of shifts for each round.

Number of bit shifts

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Compression D-box :

The compression D-box changes the 58 bits to 48 bits, which are used as a key for a round. The compression step is shown in Table 6.14.

Key-compression table

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

- **DES Analysis:**

Critics have used a strong magnifier to analyze DES. Tests have been done to measure the strength of some desired properties in a block cipher. The elements of DES have gone through scrutinies to see if they have met the established criteria. We discuss some of these in this section.

Properties

Two desired properties of a block cipher are the avalanche effect and the completeness.

Avalanche Effect:

Avalanche effect means a small change in the plaintext (or key) should create a significant change in the ciphertext. DES has been proved to be strong with regard to this property.

Completeness Effect:

It means that each bit of the ciphertext needs to depend on many bits on the plaintext. The diffusion and confusion produced by D-boxes and S-boxes in DES, show a very strong completeness effect.

- **Design Criteria**

The design of DES was revealed by IBM in 1994. Many tests on DES have proved that it satisfies some of the required criteria as claimed. We briefly discuss some of these design issues.

S-Boxes : We have discussed the general design criteria for S-boxes in Chapter 5; we only discuss the criteria selected for DES here. The design provides confusion and diffusion of bits from each round to the next. According to this revelation and some research, we can mention several properties of S-boxes.

1. The entries of each row are permutations of values between 0 and 15.
2. S-boxes are nonlinear. In other words, the output is not an affine transformation of the input. See Chapter 5 for discussion on the linearity of S-boxes.
3. If we change a single bit in the input, two or more bits will be changed in the output.
4. If two inputs to an S-box differ only in two middle bits (bits 3 and 4), the output must differ in at least two bits. In other words, S(x) and S(x $\oplus$ 001100) must differ in at least two bits where x is the input and S(x) is the output.
5. If two inputs to an S-box differ in the first two bits (bits 1 and 2) and are the same in the last two bits (5 and 6), the two outputs must be different. In other words, we need to have the following relation S(x) $\neq$ S(x $\oplus$ 11bc00), in which b and c are arbitrary bits.
6. There are only 32 6-bit input-word pairs (xi and xj), in which xi $\oplus$ xj $\neq$ (000000)2. These 32 input pairs create 32 4-bit output-word pairs. If we create the difference between the 32 output pairs, d = yi $\oplus$yj, no more than 8 of these d's should be the same.
7. A criterion similar to # 6 is applied to three S-boxes.
8. In any S-box, if a single input bit is held constant (0 or 1) and the other bits are changed randomly, the differences between the number of 0s and 1s are minimized.

D-Boxes:

Between two rows of S-boxes (in two subsequent rounds), there are one straight D-box (32 to 32) and one expansion D-box (32 to 48). These two D-boxes together provide diffusion of bits. We have discussed the general design principle of D-boxes in Chapter

5. Here we discuss only the ones applied to the D-boxes used inside the DES function. The following criteria were implemented in the design of D-boxes to achieve this goal:

1. Each S-box input comes from the output of a different S-box (in the previous round).
2. No input to a given S-box comes from the output from the same box (in the previous round).
3. The four outputs from each S-box go to six different S-boxes (in the next round).
4. No two output bits from an S-box go to the same S-box (in the next round).
5. If we number the eight S-boxes, S_1, S_2, …, S_8,
   a. An output of S_{j-2} goes to one of the first two bits of S_j (in the next round).
   b. An output bit from S_{j-1} goes to one of the last two bits of S_j (in the next round).
   c. An output of S_{j+1} goes to one of the two middle bits of S_j (in the next round).
6. For each S-box, the two output bits go to the first or last two bits of an S-box in the next round. The other two output bits go to the middle bits of an S-box in the next round.
7. If an output bit from S_j goes to one of the middle bits in S_k (in the next round), then an output bit from S_k cannot go to the middle bit of S_j. If we let $j = k$, this implies that none of the middle bits of an S-box can go to one of the middle bits of the same S-box in the next round.

Number of Rounds :

- DES uses sixteen rounds of Feistel ciphers.
- It has been proved that after eight rounds, each ciphertext is a function of every plaintext bit and every key bit; the ciphertext is thoroughly a random function of plaintext and ciphertext.
- Therefore, it looks like eight rounds should be enough. However, experiments have found that DES versions with less than sixteen rounds are even more vul- nerable to known- plaintext attacks than brute-force attack, which justifies the use of sixteen rounds by the designers of DES.

➢ **DES Weaknesses**

During the last few years critics have found some weaknesses in DES.

Weaknesses in Cipher Design

We will briefly mention some weaknesses that have been found in the design of the cipher

S-boxes:

At least three weaknesses are mentioned in the literature for S-boxes.

1. In S-box 4, the last three output bits can be derived in the same way as the first output bit by complementing some of the input bits.
2. Two specifically chosen inputs to an S-box array can create the same output.
3. It is possible to obtain the same output in a single round by changing bits in only three neighboring S-boxes.

D-boxes:

One mystery and one weakness were found in the design of D-boxes:

1. It is not clear why the designers of DES used the initial and final permutations; these have no security benefits.
2. In the expansion permutation (inside the function), the first and fourth bits of every 4- bit series are repeated.

- **Security of DES:**

DES, as the first important block cipher, has gone through much scrutiny. Among the attempted attacks, three are of interest: brute-force, differential cryptanalysis, and linear cryptanalysis.

- **Brute-Force Attack**

We have discussed the weakness of short cipher key in DES. Combining this weakness with the key complement weakness, it is clear that DES can be broken using 255 encryptions. However, today most applications use either 3DES with two keys (key size of 112) or 3DES with three keys (key size of 168). These two multiple-DES versions make DES resistant to brute-force attacks.

Differential cryptanalysis:

- DES is not immune to that kind of attack. However, it has been revealed that the designers of DES already knew about this type of attack and designed S-boxes and chose 16 as the number of rounds to make DES specifically resistant to this type of attack.
- Today, it has been shown that DES can be broken using differential cryptanalysis if we have 247 chosen plaintexts or 255 known plaintexts.
- Although this looks more efficient than a brute-force attack, finding 247 chosen plaintexts or 255 know plaintexts is impractical.
- Therefore, we can say that DES is resistant to differential cryptanalysis. It has also been shown that increasing the number of rounds to 20 require more than 264 chosen plaintexts for this attack, which is impossible because the possible number of plaintext blocks in DES is only 264.

- **Linear Cryptanalysis**

We discussed the technique of linear cryptanalysis on modern block ciphers in Chapter 5. Linear cryptanalysis is newer than differential cryptanalysis.

DES is more vulnerable to linear cryptanalysis than to differential cryptanalysis, probably because this type of attack was not known to the designers of DES. S-boxes are not very resistant to linear cryptanalysis. It has been shown that DES can be broken using 243 pairs of known plaintexts. However, from the practical point of view, finding so many pairs is very unlikely.

- **The CAST Block Cipher**

The CAST Block Cipher is an improvement of the DES block cipher, invented in Canada by Carlisle Adams and Stafford Tavares. The name of the cipher seems to be after the initials of the inventors. The CAST algorithm has 64 bit block size and has a key of size 64 bits.

CAST is based on the Feistel structure to implement the substitution permutation network. The authors state that they use the Feistel structure, as it is well studied and free of basic structural weaknesses.

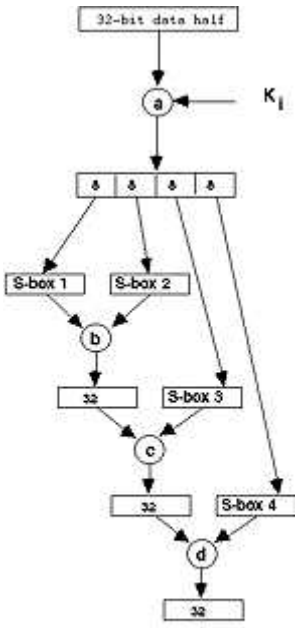


Fig. 2:
CAST Round Function

Encryption Function :

The plaintext block is di- vided into a left half and a right half. The algorithm has 8 rounds. Each round is essentially a Feistel structure. In each round the right half is combined with the round

key using a function *f* and then XOR-ed with the left half. The new left half after the round is the same as the right half before the round. After 8 iterations of the rounds, the left and the right half are concatenated to form the ciphertext.

Key Scheduling of CAST The key scheduling in CAST has three main components:

1. A key transformation step which converts the primary key (input key) to an intermediate key
2. A relatively simple bit-selection algorithm mapping the primary key and the intermediate key to a form, referred as partial key bits.
3. A set of key-schedule S-Boxes which are used to create subkeys from the partial key bits.

Let the input key be denoted by KEY $= k_1k_2k_3k_4k_5k_6k_7k_8$, where k_i is the i^{th} byte of the primary key. The key transformation step generates the intermediate key, KEY' $= k'_1k'_2k'_3k'_4k'_5k'_6k'_7k'_8$ as follows:

$$k'_1k'_2k'_3k'_4 = k_1k_2k_3k_4 \approx S_1[k_5] \approx S_2[k_7]$$
$$k'_5k'_6k'_7k'_8 = k_5k_6k_7k_8 \approx S_1[k'_1] \approx S_2[k'_3]$$

Here, S_1 and S_2 are key-schedule S-Boxes of dimension 8 ¥ 32.

Subsequently, there is a bit-selection step which operates as shown below:

$$K'_1 = k'_1k'_2$$
$$K'_2 = k'_3k'_4$$
$$K'_3 = k'_5k'_6$$
$$K'_4 = k'_7k'_8$$
$$K'_5 = k'_3k'_5$$
$$K'_6 = k'_4k'_6$$
$$K'_7 = k'_8k'_7$$
$$K'_8 = k'_6k'_5$$

The partial key bits are used to obtain the subkeys, K_i. The subkeys are 32 bits, and are obtained as follows:

$$K_i = S_1(K'_{i,1}) \approx S_2(K'_{i,2})$$

Here, K_{ij} is the j^{th} byte of K_i. Thus the 8 round subkeys are obtained.

The CAST block cipher can also be implemented with 128 bits, and is referred to as CAST-128. The essential structure of the cipher is still the same as discussed above.

- **BLOWFISH:**

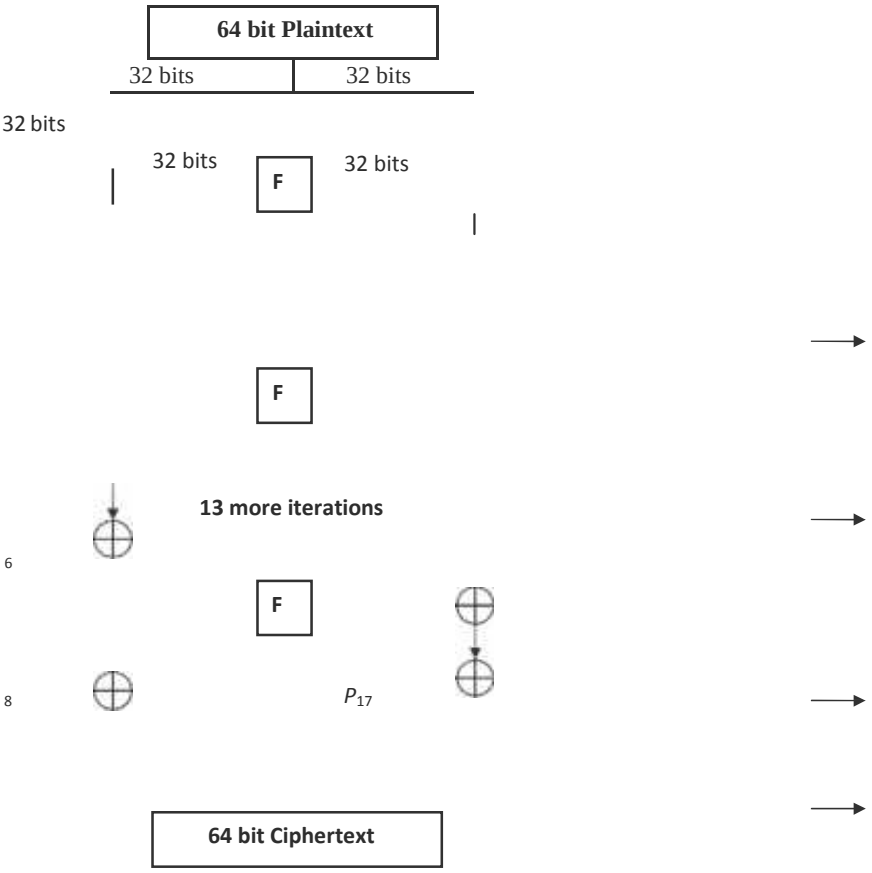
Blowfish is a 64-bit block cipher invented by Bruce Schneier. Blowfish was designed for fast ciphering on 32-bit microprocessors. Blowfish is also compact and has a variable key length which can be increased to 448 bits.

Blowfish is suitable for applications where the key does not change frequently like communication links or file encryptors. However for applications like packet switching or as an

one-way hash function, it is unsuitable. Blowfish is not ideal for smart cards, which requires even more compact ciphers. Blowfish is faster than DES when implemented on 32-bit microprocessors.

Round Structure:

The algorithm is based on the Feistel structure and has two important parts: the round structure and the key expansion func- tion.



Divide x into two 32-bit halves: x_L, x_R

For $i = 1$ to 16:

$x_L = x_L \approx P_i$

$x_R = F[x_L] \approx x_R$

Swap x_L and x_R

(undo the last swap)

$x_R = x_R \approx P_{17}$

$x_L = x_L \approx P_{18}$

Ciphertext = Concatenation of x_L and x_R

The function F is central to the security of the block cipher and is defined as below:
Divide x_L into four 8-bit parts: a, b, c, d

$$F[x_L] = ((S_1[a] + S_2[b] \bmod 2^{32}) \approx S_3[c]) + S_4[d] \bmod 2^{32}$$

Key Scheduling Algorithm:

The subkeys are computed using the following method:

1. The P-array and then the four *S*-Boxes are initialized with a fixed string. The string is the hexadecimal digits of π.

2. P_1 is XOR-ed with 32 bits of the key, P_2 is XOR-ed with the next 32 bits of the key, and so on for all the bits of the key. If needed the key bits are cycled to ensure that all the P- array elements are XOR-ed.

1. An all-zero string is encrypted with the Blowfish algorithm, with the subkeys P_1 to P_{18} obtained so far in steps 1 and 2.
2. P_1 and P_2 are replaced by the 64 bit output of step 3.
3. The output of step 3 is now encrypted with the updated subkeys to replace P_3 and P_4 with the ciphertext of step 4.
4. This process is continued to replace all the *P*-arrays and the *S*-Boxes in order.

This complex key-scheduling implies that for faster operations the subkeys should be precomputed and stored in the cache for faster access.
Security analysis by Serge Vaudenay shows that for a Blowfish algorithm implemented with known S-Boxes (note that in the original cipher the S-Boxes are generated during the encryption process) and with r-rounds, a differential attack can recover the P-array with 28r+1 chosen plaintexts.

- **IDEA**

IDEA is another block cipher. It operates on 64 bit data blocks and the key is 128 bit long. It was invented by Xuejia Lai and James Massey, and named IDEA (International Data Encryption Algorithm) in 1990, after modifying and improving the initial proposal of the cipher based on the seminal work on Differential cryptanalysis by Biham and Shamir.

The design principle behind IDEA is the "mixing of arithmetical operations from different algebraic groups". These arithmetical operations are easily implemented both in hardware and software.

The underlying operations are XOR, addition modulo 216, multiplication modulo 210+1.

The cipher obtains the much needed non-linearity from the later two arithmetical operations and

does not use an explicit S-Box.

Round Transformation of IDEA The 64-bit data is divided into four 16 bit blocks: X_1, X_2, X_3, X_4. These four blocks are processed through eight rounds and transformed by the above arithmetical operations among each other and with six 16 bit subkeys. In each round the sequence of operations is as follows:

1. Multiply X_1 and the first subkey.
2. Add X_2 and the second subkey.
3. Add X_3 and the third subkey.
4. Multiply X_4 and the fourth subkey.
5. XOR the results of step 1 and 3.
6. XOR the results of step 2 and 4.
7. Multiply the results of steps 5 with the fifth subkey.
8. Add the results of steps 6 and 7.
9. Multiply the results of steps 8 with the sixth subkey.
10. Add the results of steps 7 and 9.
11. XOR the results of steps 1 and 9.
12. XOR the results of steps 3 and 9.
13. XOR the results of steps 2 and 10.
14. XOR the results of steps 4 and 10.

The outputs of steps 11, 12, 13 and 14 are stored in four words of 16 bits each, namely Y_1, Y_2, Y_3 and Y_4. The blocks Y_2 and Y_3 are swapped, and the resultant four blocks are the output of a round of IDEA. It may be noted that the last round of IDEA does not have the swap step.

Instead the last round has the following additional transformations:

1. Multiply Y_1 and the first subkey.
2. Add Y_2 and the second subkey.
3. Add Y_3 and the third subkey.
4. Multiply Y_4 and the fourth subkey.

Finally, the ciphertext is the concatenation of the blocks Y_1, Y_2, Y_3 and Y_4.

Key Scheduling of IDEA :

- IDEA has a very simple key scheduling. It takes the 128 bit key and divides it into eight

16 bit blocks.

- The first six blocks are used for the first round, while the remaining two are to be used for the second round.
- Then the entire 128 bit key is given a rotation for 25 steps to the left and again divided into eight blocks.
- The first four blocks are used as the remaining subkeys for the second round, while the last four blocks are to be used for the third round.
- The key is then again given a left shift by 25 bits, and the other subkeys are obtained. The process is continued till the end of the algorithm.
- For decryption, the subkeys are reversed and are either the multiplicative or additive inverse of the encryption subkeys. The all zero subkey is considered to represent 216=–1 for the modular multiplication operation, mod 216+1. Thus the multiplicative inverse of 0 is itself, as –1 multiplied with –1 gives 1, the multiplicative identity in the group.
- Computing these keys may have its overhead, but it is a one time operation, at the beginning of the decryption process.
- IDEA has resisted several cryptanalytic efforts. The designers gave argument to justify that only 4 rounds of the cipher makes it immune to differential cryptanalysis.
- Joan Daemen, Rene Govaerts and Joos Vandewalle showed that the cipher had certain keys which can be easily discovered in a chosen plaintext attack.
- They used the fact that the use of multiplicative subkeys with the value of 1 or -1 gives rise to linear factors in the round function.
- A linear factor is a linear equation in the key, input and output bits that hold for all possible input bits.
- The linear factors can be revealed by expressing the modulo 2 sum of LSBs of the output subblocks of an IDEA round in terms of inputs and key bits.
- From the round structure of IDEA, the XOR of the LSBs of the first and second output subblock of a round are represented by y_1 and y_2.

$$y_1 \approx y_2 = (X_1.Z_1) \; |_0 \approx 1 \approx x_3 \approx z_3$$

If Z_1=(-)1=0…01 (i.e if the 15 MSB bits of the Z_1 are 0), we have the following linear equation

$$y_1 \approx y_2 = x_1 \approx x_3 \approx z_1 \approx z_3 \approx 1$$

TOPIC 4:

ADVANCED ENCRYPTION STANDARD:

The **Advanced Encryption Standard (AES)** is a symmetric-key block cipher published by the **National Institute of Standards and Technology** (NIST) in December 2001.

7.1.1 History

In 1997, NIST started looking for a replacement for DES, which would be called the *Advanced Encryption Standard* or *AES*. The NIST specifications required a block size of 128 bits and three different key sizes of 128, 192, and 256 bits. The specifications also required that AES be an open algorithm, available to the public worldwide. The announcement was made internationally to solicit responses from all over the world.

The AES cipher:

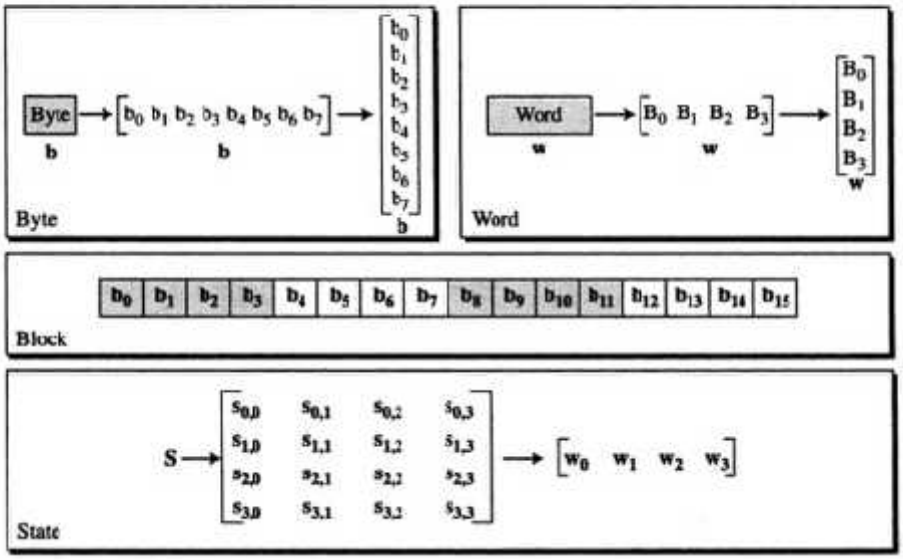
- Like DES, AES is a symmetric block cipher.
- This means that it uses the same key for both encryption and decryption.
- However, AES is quite different from DES in a number of ways.
- The algorithm Rijndael allows for a variety of block and key sizes and not just the 64 and 56 bits of DES' block and key size.
- The block and key can in fact be chosen independently from 128, 160, 192, 224, 256 bits and need not be the same.
- However, the AES standard states that the algorithm can only accept a block size of 128 bits and a choice of three keys - 128, 192, 256 bits.
- Depending on which version is used, the name of the standard is modified to AES-128, AES-192 or AES- 256 respectively.
- As well as these differences AES differs from DES in that it is not a feistel structure.
- Recall that in a feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. In this case the entire data block is processed in parallel during each round using substitutions and permutations.
- A number of AES parameters depend on the key length. For example, if the key size used is 128 then the number of rounds is 10 whereas it is 12 and 14 for 192 and 256 bits respectively.
- At present the most common key size likely to be used is the 128 bit key. This description of the AES algorithm therefore describes this particular implementation.

Rijndael was designed to have the following characteristics:

- Resistance against all known attacks.
- Speed and code compactness on a wide range of platforms.
- Design Simplicity.

DATA UNITS IN AES:

AES uses five units of measurement to refer to data: bits, bytes, words, blocks, and state. The bit is the smallest and atomic unit; other units can be expressed in terms of smaller ones. Figure 7.2 shows the non-atomic data units: byte, word, block, and state.



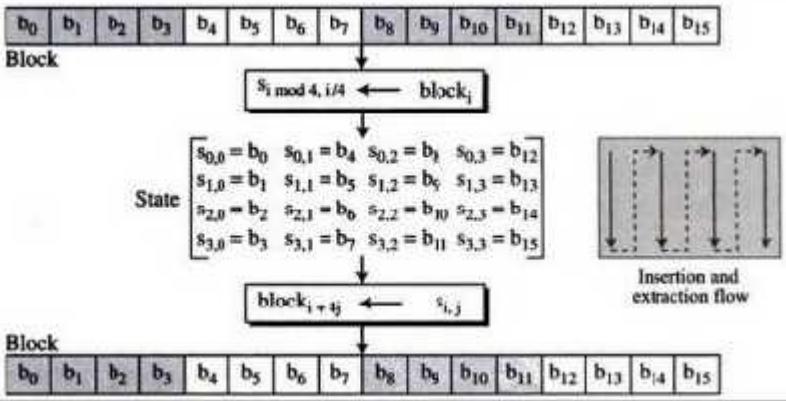
Bit In AES, a **bit** is a binary digit with a value of 0 or 1. We use a lowercase letter to refer to a bit.

Byte A **byte** is a group of eight bits that can be treated as a single entity, a row matrix (1×8) of eight bits, or a column matrix (8×1) of eight bits. When treated as a row matrix, the bits are inserted to the matrix from left to right; when treated as a column matrix, the bits are inserted into the matrix from top to bottom. We use a lowercase bold letter to refer to a byte.

Word A **word** is a group of 32 bits that can be treated as a single entity, a row matrix of four bytes, or a column matrix of four bytes. When it is treated as a row matrix, the bytes are inserted into the matrix from left to right; when it is considered as a column matrix, the bytes are inserted into the matrix from top to bottom. We use the lowercase bold letter w to show a word.

Block AES encrypts and decrypts data blocks. A **block** in AES is a group of 128 bits. However, a block can be represented as a row matrix of 16 bytes.

State AES uses several rounds in which each round is made of several stages. Data block is transformed from one stage to another. At the beginning and end of the cipher, AES uses the term *data block*, before and after each stage, the data block is referred to as a **state.** We use an uppercase bold letter to refer to a state. Although the states in different stages are normally called **S,** we occasionally use the letter **T** to refer to a temporary state. States, like blocks, are made of 16 bytes, but normally are treated as matrices of 4 × 4 bytes. In this case, each element of a state is referred to as $\mathbf{s}_{r,c}$, where r (0 to 3) defines the row and the c (0 to 3) defines the column. Occasionally, a state is treated as a row matrix (1 × 4) of words. This makes sense, if we think of a word as a column matrix. At the beginning of the cipher, bytes in a data block are inserted into a state column by column, and in each column, from top to bottom. At the end of the cipher, bytes in the state are extracted in the same way, as shown in Fig. 7.3.



The overall structure of AES can be seen in below figure.

- The input is a single 128 bit block both for decryption and encryption and is known as the in matrix.
- This block is copied into a state array which is modified at each stage of the algorithm and then copied to an output matrix.
- Both the plaintext and key are depicted as a 128 bit square matrix of bytes. This key is then expanded into an array of key schedule words (the w matrix).
- It must be noted that the ordering of bytes within the in matrix is by column. The same applies to the w matrix.

Inner Workings of a Round:

The algorithm begins with an Add round key stage followed by 9 rounds of four stages and a tenth round of three stages. This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of it's counterpart in the encryption algorithm. The four stages are as follows:

1. Substitute bytes
2. Shift rows
3. Mix Columns

4. Add Round Key

The tenth round simply leaves out the Mix Columns stage. The first nine rounds of the decryption algorithm consist of the following:

1. Inverse Shift rows
2. Inverse Substitute bytes
3. Inverse Add Round Key
4. Inverse Mix Columns

Again, the tenth round simply leaves out the Inverse Mix Columns stage. Each of these stages will now be considered in more detail.

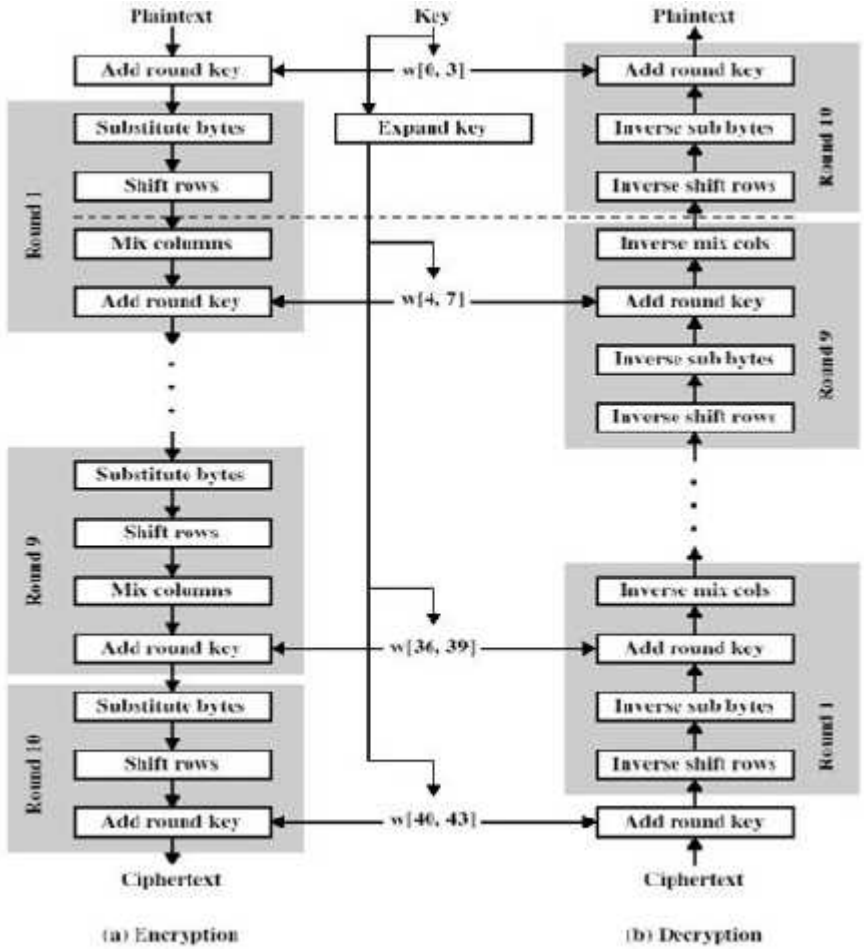


Figure: Overall structure of the AES algorithm.

Substitute Bytes:

- This stage (known as SubBytes) is simply a table lookup using a 16 16 matrix of byte values called an s-box.
- This matrix consists of all the possible combinations of an 8 bit sequence (2^8 = 16 16 = 256).
- However, the s-box is not just a random permutation of these values and there is a well defined method for creating the s-box tables.
- The designers of Rijndael showed how this was done unlike the s-boxes in DES for which no rationale was given.
- We will not be too concerned here how the s-boxes are made up and can simply take them as table lookups.
- Again the matrix that gets operated upon throughout the encryption is known as state.
- We will be concerned with how this matrix is effected in each round. For this particular

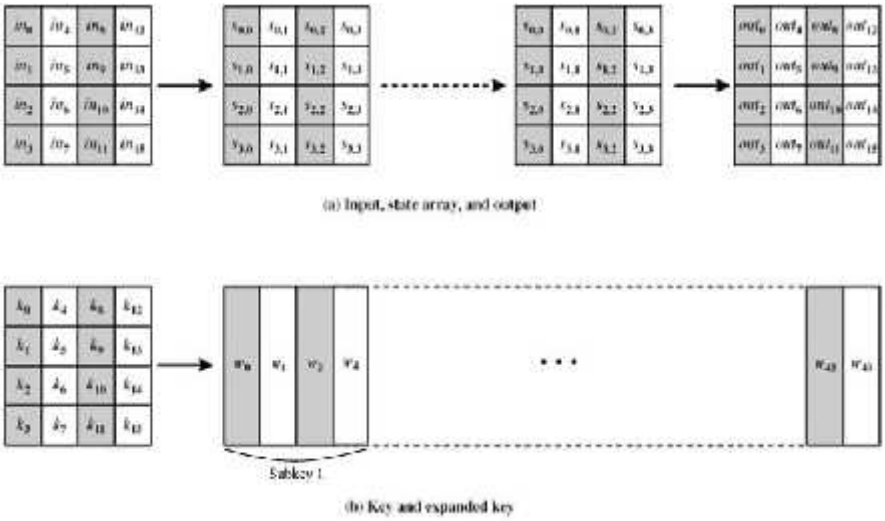


Figure : Data structures in the AES algorithm.

round each byte is mapped into a new byte in the following way: the leftmost nibble of the byte is used to specify a particular row of the s-box and the rightmost nibble specifies a column.

- For example, the byte 95 (curly brackets represent hex values in FIPS PUB 197) selects row 9 column 5 which turns out to contain the value 2*A* . This is then used to update the state matrix. Figure 7.3 depicts this idea.

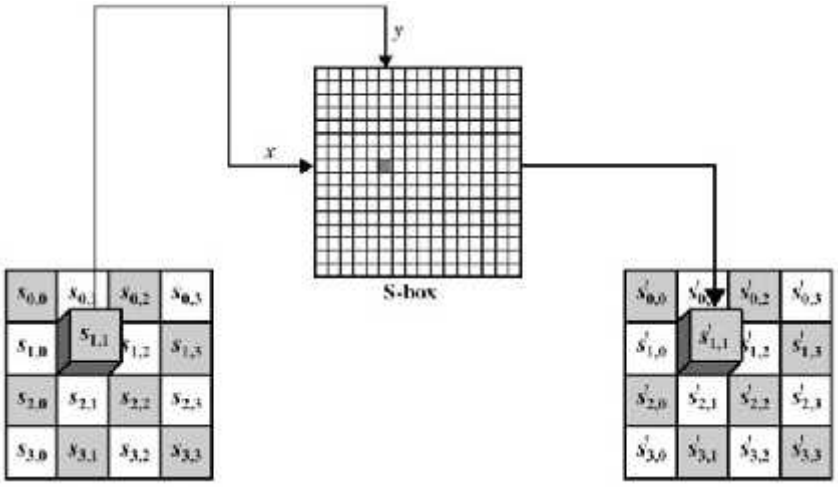


Figure: Substitute Bytes Stage of the AES algorithm.

The Inverse substitute byte transformation (known as InvSubBytes) makes use of an inverse s-box. In this case what is desired is to select the value {2*A*} and get the value

{95}. Table 7.4 shows the two s-boxes and it can be verified that this is in fact the case.

- The s-box is designed to be resistant to known cryptanalytic attacks.
- Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits, and the property that the output cannot be described as a simple mathemat- ical function of the input.
- In addition, the s-box has no fixed points (s-box(*a*) = *a*) and no opposite fixed points (s-box(*a*) =−*a*) where −*a* is the bitwise compliment of *a*. The s-box must be invertible if decryption is to be possible (Is-box[s-box(*a*)]= *a*) however it should not be its self inverse i.e. s-box(*a*) ƒ= Is-box(*a*)

Figure : AES s-boxes both forward and inverse.

(a) S-box

	y															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(b) Inverse S-box

	y															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Shift Row Transformation:

This stage (known as ShiftRows) is shown in figure 7.5. This is a simple permutation an nothing more. It works as follow:

- The first row of state is *not* altered.
- The second row is shifted 1 bytes to the left in a circular manner.
- The third row is shifted 2 bytes to the left in a circular manner.
- The fourth row is shifted 3 bytes to the left in a circular manner.

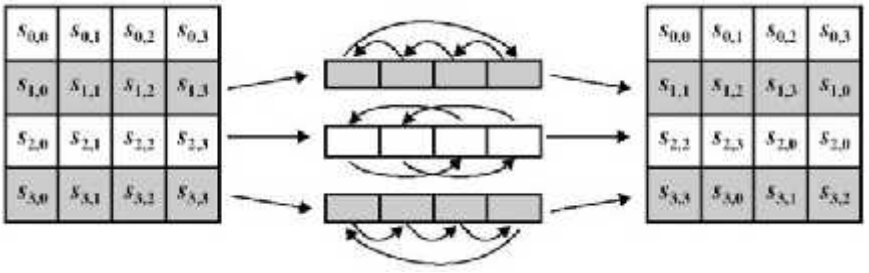
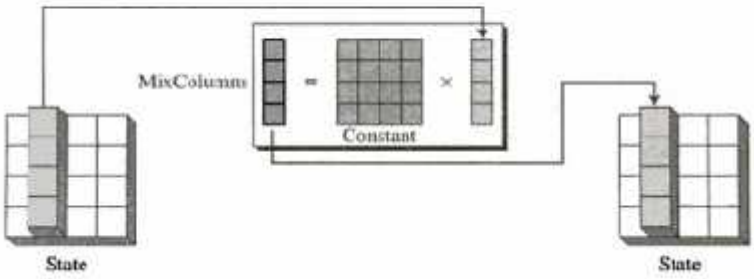


Figure : ShiftRows stage.

- The Inverse Shift Rows transformation (known as InvShiftRows) performs these cir- cular shifts in the opposite direction for each of the last three rows (the first row was unaltered to begin with).
- This operation may not appear to do much but if you think about how the bytes are ordered within state then it can be seen to have far more of an impact. Remember that state is treated as an array of four byte columns, i.e. the first column actually represents bytes 1, 2, 3 and 4.
- A one byte shift is therefore a linear distance of four bytes. The transformation also ensures that the four bytes of one column are spread out to four different columns.

Mix Column Transformation:

MixColumns The **MixColumns** transformation operates at the column level; it transforms each column of the state to a new column. The transformation is actually the matrix multiplication of a state column by a constant square matrix. The bytes in the state column and constants matrix are interpreted as 8-bit words (or polynomials) with coefficients in GF(2). Multiplication of bytes is done in $GF(2^8)$ with modulus (10001101) or $(x^8 + x^4 + x^3 + x + 1)$. Addition is the same as XORing of 8-bit words. Figure 7.13 shows the MixColumns transformations.



InvMixColumns The **InvMixColumns** transformation is basically the same as the MixColumns transformation. If the two constant matrices are inverses of each other, it is easy to prove that the two transformations are inverses of each other.

The MixColumns and InvMixColumns transformations are inverses of each other.

Add Round Key Transformation:

In this stage (known as AddRoundKey) the 128 bits of state are bitwise XORed with the 128 bits of the round key. The operation is viewed as a columnwise operation between the 4 bytes of a state column and one word of the round key. This transfor- mation is as simple as possible which helps in efficiency but it also effects every bit of state.

AES Key Expansion:

The AES key expansion algorithm takes as input a 4-word key and produces a linear array of 44 words. Each round uses 4 of these words as shown in figure 7.2. Each word contains 32 bytes which means each subkey is 128 bits long. Figure 7.7 show pseudocode for generating the expanded key from the actual key.

```
KeyExpansion (byte key[16], word w[44])
{
  word temp
  for (i = 0; i < 4; i++)    w[i] = (key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3]);
  for (i = 4; i < 44; i + +)
  {
    temp = w[i − 1];
    if (i mod 4 = 0)    temp = SubWord (RotWord(temp)) ⊕ Rcon[i/4];
    w[i] = w[i − 4] ⊕ temp;
  }
}
```

Figure : Key expansion pseudocode.

- The key is copied into the first four words of the expanded key.
- The remainder of the expanded key is filled in four words at a time.
- Each added word w[*i*] depends on the immediately preceding word, w[*i* 1], and the word four positions back w[*i* 4].
- In three out of four cases, a simple XOR is used. For a word whose position in the w array is a multiple of 4, a more complex function is used. Figure illustrates the generation of the first eight words of the expanded key using the symbol *g* to represent that complex function. The function *g* consists of the following subfunctions:

  1. RotWord performs a one-byte circular left shift on a word. This means that an input word [b_0, b_1, b_2, b_3] is transformed into [b_1, b_2, b_3, b_0].
  2. SubWord performs a byte substitution on each byte of its input word, using the s-box described earlier.
  3. The result of steps 1 and 2 is XORed with round constant, Rcon[*j*].

- The round constant is a word in which the three rightmost bytes are always 0. Thus the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word.
- The round constant is different for each round and is defined as Rcon[*j*] = (RC[*J*], 0,0,0), with RC[1]= 1, RC[*j*]= 2 RC[*j* 1] and with multiplication defined over the field GF(2^8).

The key expansion was designed to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the way in which round keys are generated in different rounds.

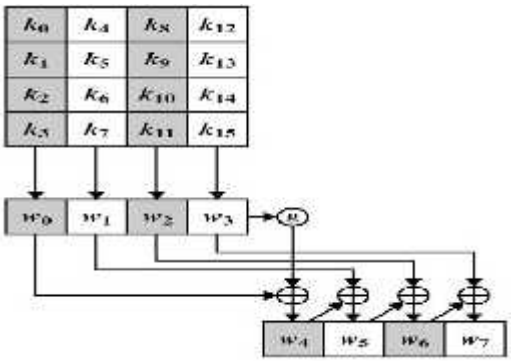
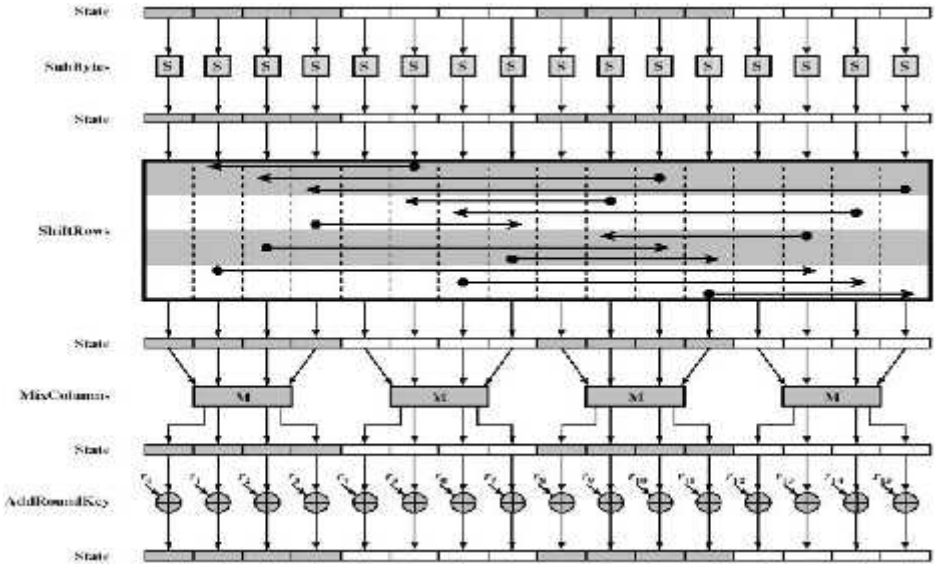


Figure : AES key expansion.

The below Figure give a summary of each of the rounds. The ShiftRows column is depicted here as a linear shift which gives a better idea how this section helps in the encryption.

Figure : AES encryption round.



Equivalent Inverse Cipher

As can be seen from figure 7.1 the decryption ciphers is not identical to the encryption ciphers. However the form of the key schedules is the same for both. This has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption. As well as that, decryption is slightly less efficient to implement. However, encryption was deemed more important than decryption for two reasons:

4. For the CFB and OFB cipher mode (which we have seen before but will study in more detail next) only encryption is used.
5. As with any block cipher, AES can be used to construct a message authentication code (to be described later), and for this only encryption is used.

However, if desired it is possible to create an equivalent inverse cipher. This means that decryption has the same structure as the encryption algorithms. However, to achieve this, a change of key schedule is needed. We will not be concerned with this alternate form but you should be aware that is exists.

UNIT-III

Public Key Cryptography

Introduction to Public key Cryptography:

- Public key cryptography also called as **asymmetric cryptography**.
- It was invented by whitfield **Diffie** and Martin **Hellman** in 1976. Sometimes this cryptography also called as **Diffie-Helman Encryption**.
- Public key algorithms are based on mathematical problems which admit no efficient solution that are inherent in certain integer factorization, discrete logarithm and Elliptic curve relations.

Public key Cryptosystem Principles:

- The concept of public key cryptography in invented for two most difficult problems of Symmetric key encryption.

- The Key Exchange Problem
- The Trust Problem

The Key Exchange Problem: The key exchange problem arises from the fact that communicating parties must somehow share a secret key before any secure communication can be initiated, and both parties must then ensure that the key remains secret. Of course, direct key exchange is not always feasible due to risk, inconvenience, and cost factors.

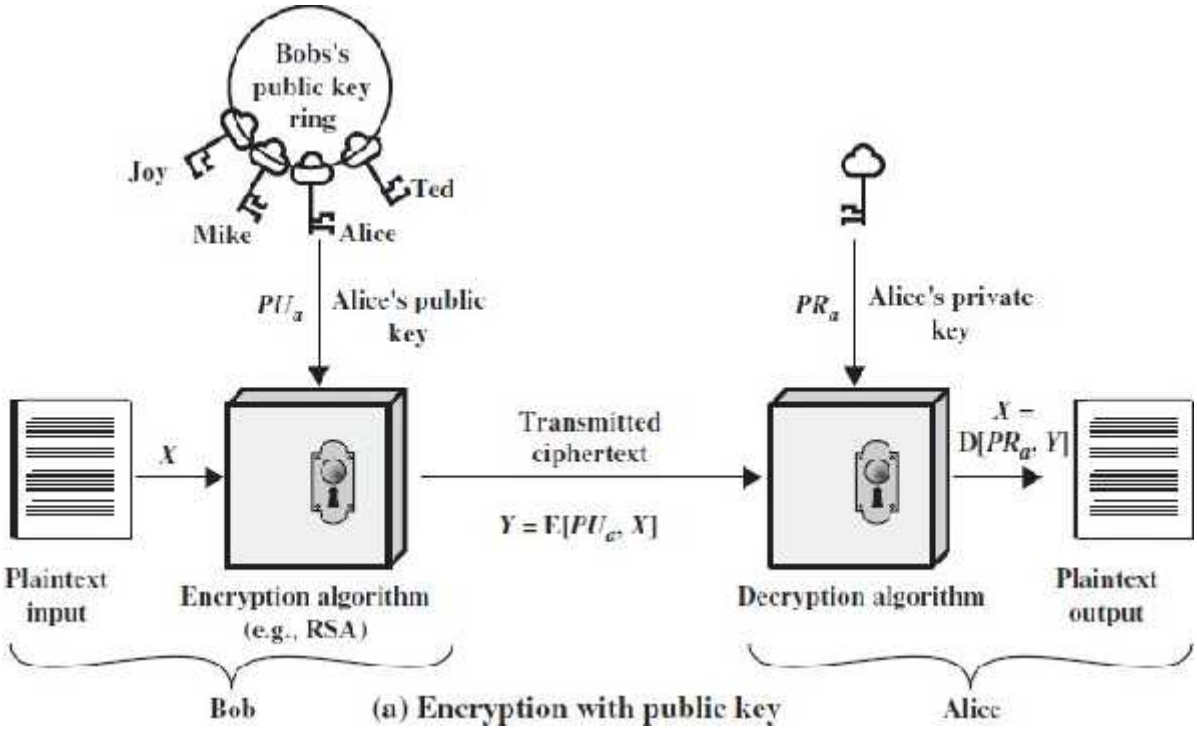
The Trust Problem: Ensuring the integrity of received data and verifying the identity of the source of that data can be very important. Means in the symmetric key cryptography system, receiver doesn‟t know whether the message is coming for particular sender.

- This public key cryptosystem uses two keys as pair for encryption of plain text and Decryption of cipher text.
- These two keys are names as "**Public key**" and "**Private key**". The private key is kept secret where as public key is distributed widely.
- A message or text data which is encrypted with the public key can be decrypted only with the corresponding private-key
- This two key system very useful in the areas of confidentiality (secure) and authentication

A **public-key encryption** scheme has six ingredients		
1	**Plaintext**	This is the readable message or data that is fed into the algorithm as input.
2	**Encryption algorithm**	The encryption algorithm performs various transformations on the plaintext.
3	**Public key**	This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input
4	**Private key**	

5	**Ciphertext**	This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
6	**Decryption algorithm**	This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

Public key cryptography for providing confidentiality (secrecy)



The essential steps are the following.

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key.The companion key is kept private.As Figure 9.1a suggests, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice‟s public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice‟s private key.

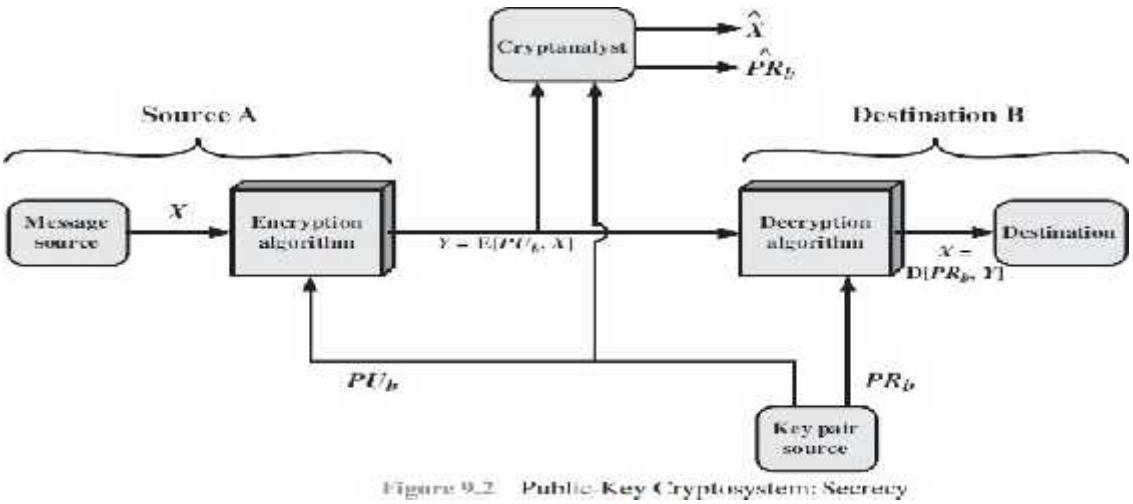


Figure 9.2 Public-Key Cryptosystem: Secrecy

There is some source A that produces a message in plaintext ***X* = [*X*1, *X*2, . . . ,*XM*].**

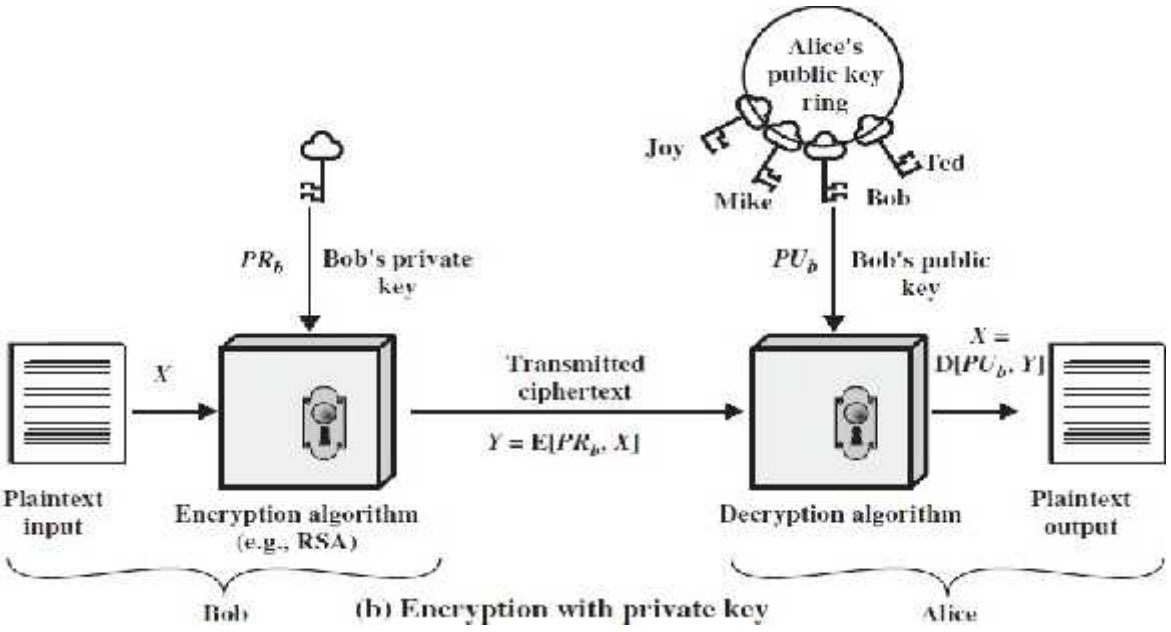
The ***M*** elements of ***X*** are letters in some finite alphabet. The message is intended for destination **B**.
B generates a related pair of keys: a public key, PU_b, and a private key, PR_b.
PR_b is known only to B, whereas PU_b is publicly available and therefore accessible by A.
With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = [Y1, Y2, \ldots, Y_N]$:

$$Y = E(PU_b, X)$$

$$X = D(PR_b, Y)$$

The intended receiver, in possession of the matching private key, is able to invert the transformation:

Public key cryptography for proving Authentication:



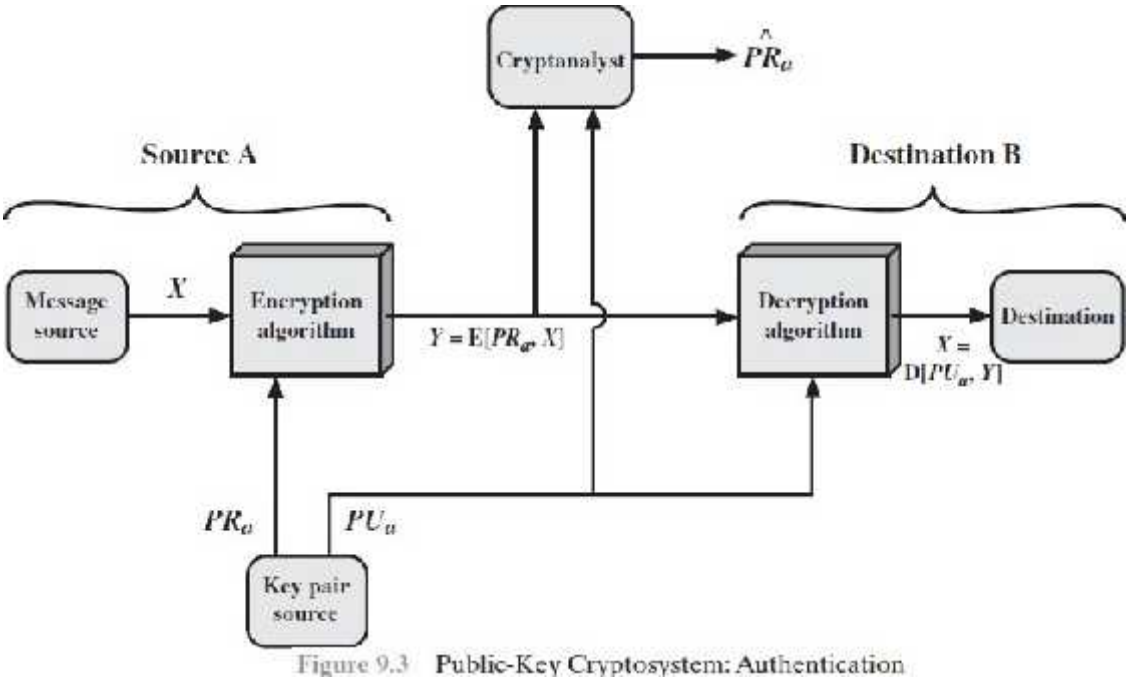


Figure 9.3 Public-Key Cryptosystem: Authentication

The above diagrams show the use of public-key encryption to provide authentication:

$$Y = E(PR_a, X)$$

$$X = D(PU_a, Y)$$

- In this case,A prepares a message to B and encrypts it using A‟s private key before transmitting it. B can decrypt the message using A‟s public key. Because the message was encrypted using A‟s private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a **digital signature.**

- It is impossible to alter the message without access to A‟s private key, so the message is authenticated both in terms of source and in terms of data integrity.

Public key cryptography for both authentication and confidentiality (Secrecy)

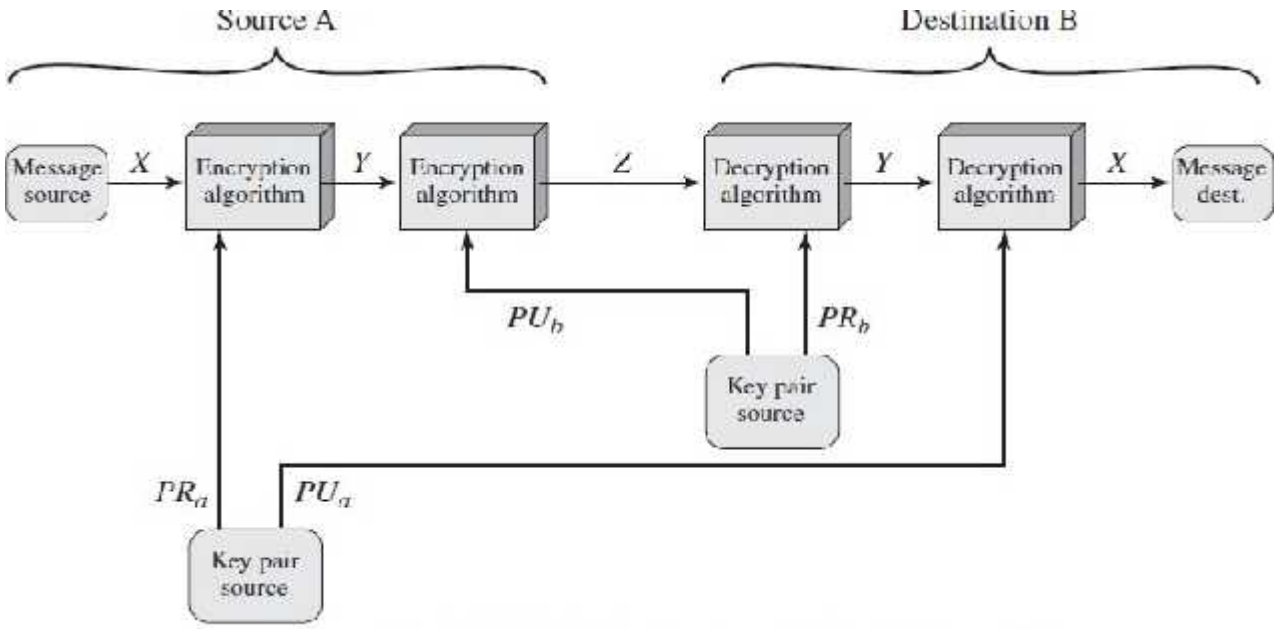


Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (above figure):

$$Z = E(PU_b, E(PR_a, X))$$

$$X = D(PU_a, D(PR_b, Z))$$

In this case, we begin as before by encrypting a message, using the sender‟s private key. This provides the digital signature. Next, we encrypt again, using the receiver‟s public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided.

Applications for Public-Key Cryptosystems

Public-key systems are characterized by the use of a cryptographic algorithm with two keys, one held private and one available publicly. Depending on the application, the sender uses either the sender‟s private key or the receiver‟s public key, or both, to perform some type of cryptographic function. the use of **public-key cryptosystems** into three categories

• Encryption /decryption: The sender encrypts a message with the recipient‟s public key.

• Digital signature: The sender “signs” a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.

• Key exchange: Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Applications for Public-Key Cryptosystems

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Public-Key Cryptanalysis

As with symmetric encryption, a public-key encryption scheme is vulnerable to a brute-force attack. The countermeasure is the same: Use large keys. However, there is a tradeoff to be considered. Public-key systems depend on the use of some sort of invertible mathematical function. The complexity of calculating these functions may not scale linearly with the number of bits in the key but grow more rapidly than that. Thus, the key size must be large enough to make brute-force attack impractical but small enough for practical encryption and decryption. In practice, the key sizes that have been proposed do make brute-force attack impractical but result in encryption/decryption speeds that are too slow for general-purpose use. Instead, as was mentioned earlier, public-key encryption is currently confined to key management and signature applications.

RSA Algorithm

- It is the most common public key algorithm.
- This RSA name is get from its inventors first letter (Rivest (R), Shamir (S) and Adleman (A)) in the year 1977.
- The RSA scheme is a block cipher in which the plaintext & ciphertext are integers between 0 and n-1 for some „n".
- A typical size for „n" is 1024 bits or 309 decimal digits. That is, n is less than 2^{1024}

Description of the Algorithm:

- RSA algorithm uses an expression with exponentials.
- In RSA plaintext is encrypted in blocks, with each block having a binary value less than some number n. that is, the block size must be less than or equal to $\log_2(n)$
- **RSA** uses two exponents „e" and „d" where e→public and d→private.
- Encryption and decryption are of following form, for some PlainText „M" and CipherText block „C"

$$C = M^e \bmod n$$

$$M = C^d \bmod n$$

$$M = C^d \bmod = (M^e \bmod n)^d \text{ mon } n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Both sender and receiver must know the value of n.

The sender knows the value of „e" & only the reviver knows the value of „d" thus this is a public key encryption algorithm with a

Public key PU={e, n}

Private key PR={d, n}

Requirements:

The RSA algorithm to be satisfactory for public key encryption, the following requirements must be met:

1. It is possible to find values of e, d n such that " $\mathbf{M^{ed} \bmod n = M}$ " for all M<n

2. It is relatively easy to calculate " $\mathbf{M^e \bmod n}$ " and " $\mathbf{C^d \bmod n}$ "for M<n
3. It is infeasible to determine "d" given „e" & „n". The " $\mathbf{M^{ed} \bmod n = M}$ " relationship holds if „e" & „d" are multiplicative inverses modulo Ø(n).
   Ø(n)→ Euler Totient function
   For p,q primes where p*q and p≠q.
   Ø(n)= Ø(pq)=(p-1)(q-1)

Then the relation between „e" & „d" can be expressed as " $\boxed{ed \bmod Ø(n) = 1}$ "
this is equivalent to saying

$$\boxed{ed \equiv 1 \bmod Ø(n)}$$
$$\boxed{d \equiv e^{-1} \bmod Ø(n)}$$

That is „e" and „d" are multiplicative inverses mod Ø(n).
Note: according to the rules of modular arithmetic, this is true only if „d" (and „e") is relatively prime to Ø(n).
Equivalently **gcd(Ø(n), d)=1.**

Steps of RSA algorithm:

Step 1→Select 2 prime numbers p & q
Step 2→Calculate n=pq
Step 3→Calculate Ø(n)=(p-1)(q-1)
Step 4→ Select or find integer e (public key) which is relatively prime to Ø(n).
ie., e with gcd (Ø(n), e)=1 where 1<e< Ø(n).
Step 5→ Calculate "d" (private key) by using following condition. $\boxed{ed = 1 \bmod Ø(n)}$ d< Ø(n).
Step 6→ Perform encryption by using $\boxed{C = M^e \bmod n}$
Step 7→ perform Decryption by using $\boxed{M = C^d \bmod n}$

Example:

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 \times 11 = 187$.
3. Calculate $Ø(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.
4. Select e such that e is relatively prime to $Ø(n) = 160$ and less than $Ø(n)$; we choose $e = 7$.
5. Determine d such that $\mathbf{de \equiv 1 \pmod{160}}$ and $d < 160$.The correct value is $d = 23$, because $23 * 7 = 161 = (1 \times 160) + 1$; d can be calculated using the extended Euclid"s algorithm

The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$.
The example shows the use of these keys for a plaintext input of $M = 88$. For encryption,
we need to calculate $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows.

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$
$$88^1 \bmod 187 = 88$$
$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$
$$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$$
$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11$$

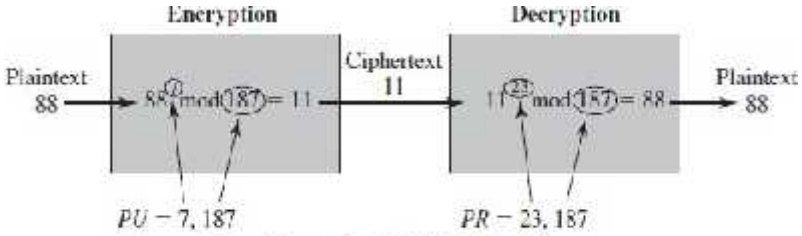


Figure 9.6 Example of RSA Algorithm

For decryption, we calculate $M = 11^{23} \bmod 187$:

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14{,}641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79{,}720{,}245 \bmod 187 = 88$$

The Security of RSA

Four possible approaches to attacking the RSA algorithm are

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

Diffie-Hellman Key Exchange

- Diffie-Hellman key exchange is the first published public key algorithm
- This Diffie-Hellman key exchange protocol is also known as exponential key agreement. And it is based on mathematical principles.
- The purpose of the algorithm is to enable two users to exchange a key securely that can then be used for subsequent encryption of messages.
- This algorithm itself is limited to exchange of the keys.
- This algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.
- The discrete logarithms are defined in this algorithm in the way of define a primitive root of a prime number.
- Primitive root: we define a primitive root of a prime number P as one whose power generate all the integers form 1 to P-1 that is if „a" is a primitive root of the prime number P, then the

  $a \bmod P, a^2 \bmod P, a^3 \bmod P, \ldots\ldots a^{P-1} \bmod P$

  numbers

are distinct and consist of the integers form 1 through P-1 in some permutation.

For any integer „b" and „a", here „a" is a primitive root of prime number P, then

$$b \equiv a^i \bmod P \qquad 0 \le i \le (P-1)$$

The exponent i → is refer as discrete logarithm or index of b for the base a, mod P.

The value denoted as $\mathbf{ind}_{a,p}(b)$

Algorithm for Diffie-Hellman Key Exchange:

Step 1→ two public known numbers q, α

q→ Prime number

α→ primitive root of q and α< q.

Step 2 → if A & B users wish to exchange a key

a) User A select a random integer X_A<q and computes $Y_A = \alpha^{X_A} \bmod q$
b) User B independently select a random integer X_B <q and computes $Y_B = \alpha^{X_B} \bmod q$
c) Each side keeps the X value private and Makes the Y value available publicly to the outer side.

Step 3→ User A Computes the key as $K = (Y_B)^{X_A} \bmod q$

User B Computes the key as $K = (Y_A)^{X_B} \bmod q$

Step 4→ two calculation produce identical results

$K = (Y_B)^{X_A} \bmod q$

$K = (\alpha^{X_B} \bmod q)^{X_A} \bmod q$ (We know that $Y_B = \alpha^{X_B} \bmod q$)

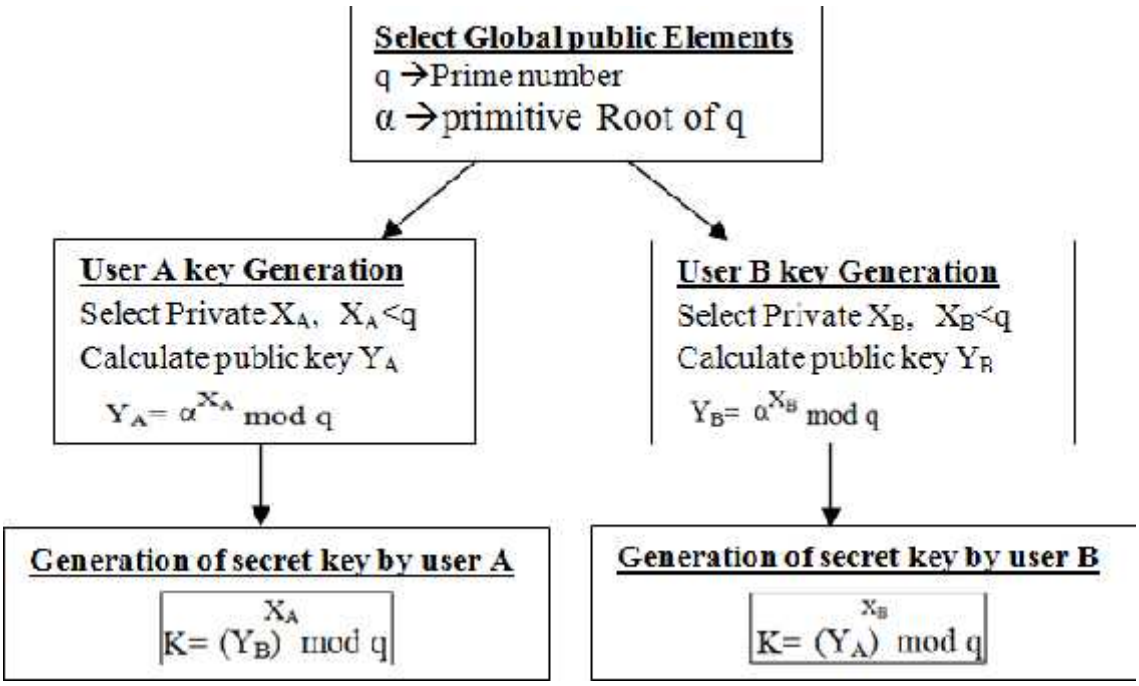
$= (\alpha^{X_B})^{X_A} \bmod q$

$= (\alpha^{X_A})^{X_B} \bmod q$

$= (\alpha^{X_A} \bmod q)^{X_B} \bmod q$

$= (Y_A)^{X_B} \bmod q$ (We know that $Y_A = \alpha^{X_A} \bmod q$)

The result is that the two sides have exchanged a secret key.



Example:

Here is an example. Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case $\alpha = 3$. A and B select secret keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its public key:

A computes $Y_A = 3^{97} \bmod 353 = 40$.

B computes $Y_B = 3^{233} \bmod 353 = 248$.

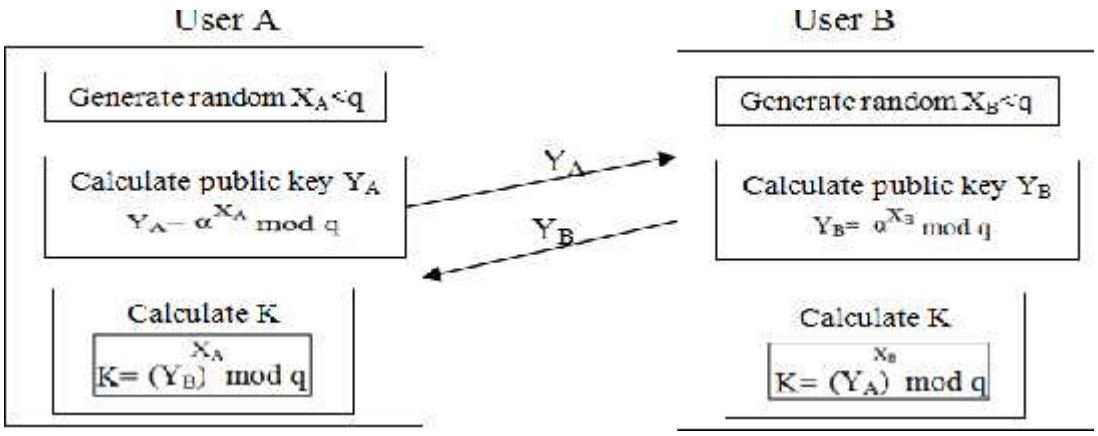
After they exchange public keys, each can compute the common secret key:

A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.

B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$.

We assume an attacker would have available the following information:

$$q = 353; \alpha = 3; Y_A = 40; Y_B = 248$$



MAN-in the Middle Attack (MITM)

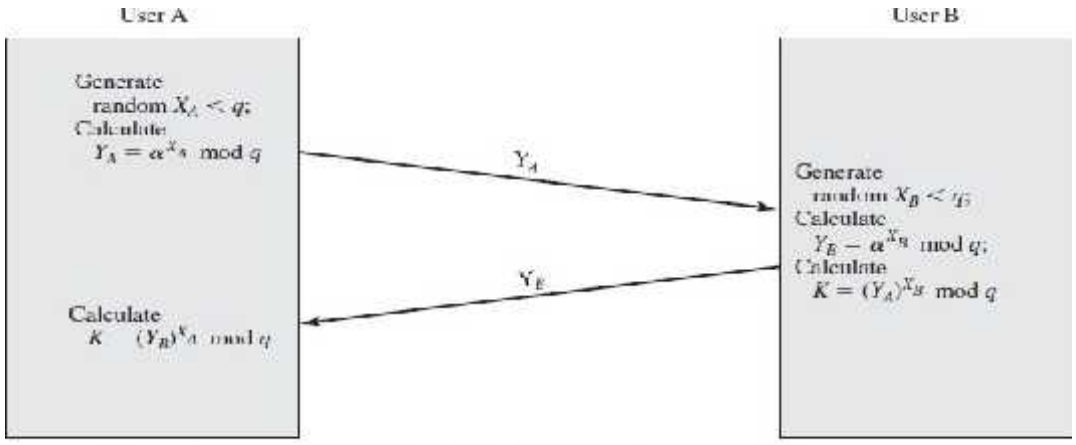


Figure 10.2 **Diffie-Hellman Key Exchange**

Definition: A man in the middle attack is a form of eavesdropping where communication between two users is monitored and modified by an unauthorized party.

Generally the attacker actively eavesdrops by intercepting (stoping) a public key message exchange.

The Diffie- Hellman key exchange is insecure against a "Man in the middle attack".

Suppose user „A" & „B" wish to exchange keys, and D is the adversary (opponent). The attack proceeds as follows.

1. „D" prepares for the attack by generating two random private keys X_{D1} & X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. „A" transmits „Y_A" to „B"
3. „D" intercepts Y_A and transmits Y_{D1} to „B". and D also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
4. „B" receives Y_{D1} & calculate $K1 = (Y_{D1})^{X_B} \bmod q$.
5. „B" transmits „Y_B" to „A"
6. „D" intercepts „Y_B" and transmits Y_{D2} to „A" and „D" calculate K1 $K1 = (Y_B)^{X_{D1}} \bmod q$
7. A receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key *K*1 and Alice and Darth share secret key *K*2. All future communication between Bob and Alice is compromised in the following way.

1. A sends an encrypted message M: $E(K2, M)$.
2. D intercepts the encrypted message and decrypts it to recover M.
3. D sends B $E(K1, M)$ or $E(K1, M')$, where M' is any message. In the first case, D simply wants to eavesdrop on the communication without altering it. In the second case, D wants to modify the message going to B

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

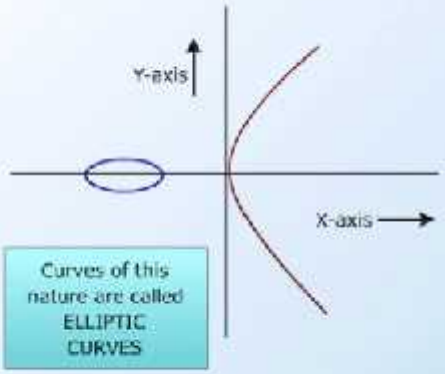
Elliptic Curve Cryptography

- **Definition: Elliptic curve cryptography (ECC)** is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. These are analogy of existing public key cryptosystem in which modular arithmetic is replaced by operations defined over elliptic curve.
- The use of elliptic curves in cryptography was suggested independently by **Neal Koblitz** and **Victor S. Miller** in **1985.**
- Elliptic curve cryptography (ECC) is one of the most powerful but least understood types of cryptography in wide use today. An increasing number of websites make extensive use of ECC to secure everything from customers' HTTPS connections to how they pass data between data centers.

An elliptic curve is defined by an equation in two variables with coefficients. For cryptography, the variables and coefficients are restricted to elements in a finite field, which results in the definition of a finite abelian group.

Elliptic Curves over Real Numbers

- Users A and B wish to exchange keys to communicate and User C is the adversary (attacker)
- It is an approach to public-key cryptography is based on the algebraic structure of elliptic curves
- The principle attraction of ECC compared to RSA is, it appears to offer equal security for a far smaller bit size, thereby reducing processing overhead
- The ECC algorithm is faster than all public key algorithms



Elliptic Curves

ECC - Key Exchange

Global Public elements.

$E_q(a,b)$: Elliptic curve with parameters a, b & q
q is a prime or integer of the form 2^m

G : Point on elliptic curve whose order is large value n.

Alice key Generation.

Select private key n_A ; $n_A < n$

Calculate public key P_A ; $P_A = n_A \times G$

Bob key Generation.

Select private key n_B ; $n_B < n$

Calculate public key P_B ; $P_B = n_B \times G$

Secret key calculation by Alice

$K = n_A \times P_B$

Secret key calculation by Bob

$K = n_B \times P_A$

ECC - Encryption & Decryption

- Let the message be M.
- First encode the message M into a point on the elliptic curve.
- Let this point be P_m.
- Now this point is encrypted.
- For encrypting choose a random positive integer k.
- Then $C_m = \{kG, P_m + kP_B\}$ where G is the base point.

- For decryption, multiply first point in the pair with receiver's secret key.
  i.e, $kG \times n_B$
- Then subtract it from second point in the pair.
  i.e, $P_m + kP_B - (kG \times n_B)$

ELGAMAL CRYPTOGRAPHIC SYSTEM:

- In 1984, T. Elgamal announced a public-key scheme based on discrete logarithms, closely related to the Diffie-Hellman technique.
- EIGamal Algorithms are used for both digital signatures as well as encryption.

EIGamal Algorithm:-

Global Public Elements	
q	prime number
α	$\alpha < q$ and α a primitive root of q

Key Generation by Alice	
Select private X_A	$X_A < q-1$
Calculate Y_A	$Y_A = \alpha^{X_A} \bmod q$
Public key	$PU = \{q, \alpha, Y_A\}$
Private key	X_A

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < q$
Select random integer k	$k < q$
Calculate K	$K = (Y_A)^k \bmod q$
Calculate C_1	$C_1 = \alpha^k \bmod q$
Calculate C_2	$C_2 = KM \bmod q$
Ciphertext:	(C_1, C_2)

Decryption by Alice with Alice's Private Key	
Ciphertext:	(C_1, C_2)
Calculate K	$K = (C_1)^{X_A} \bmod q$
Plaintext:	$M = (C_2 K^{-1}) \bmod q$

Figure 10.3 The ElGamal Cryptosystem

Thus, functions as a one-time key, used to encrypt and decrypt the message. For example, let us start with the prime field GF(19); that is, q = 19. It has primitive roots {2, 3, 10, 13, 14, 15 }. We choose α = 10.

Alice generates a key pair as follows:

1. Alice chooses $X_A = 5$.
2. Then $Y_A = \alpha^{X_A} \bmod q = \alpha^5 \bmod 19 = 3$ (see Table 8.3).
3. Alice's private key is 5; Alice's pubic key is $\{q, \alpha, Y_A\} = \{19, 10, 3\}$.

Suppose Bob wants to send the message with the value $M = 17$. Then,

1. Bob chooses $k = 6$.
2. Then $K = (Y_A)^k \bmod q = 3^6 \bmod 19 = 729 \bmod 19 = 7$.
3. So
   $C_1 = \alpha^k \bmod q = \alpha^6 \bmod 19 = 11$
   $C_2 = KM \bmod q = 7 \times 17 \bmod 19 = 119 \bmod 19 = 5$
4. Bob sends the ciphertext (11, 5).

For decryption:

1. Alice calculates $K = (C_1)^{X_A} \bmod q = 11^5 \bmod 19 = 161051 \bmod 19 = 7$.
2. Then K^{-1} in GF(19) is $7^{-1} \bmod 19 = 11$.
3. Finally, $M = (C_2 K^{-1}) \bmod q = 5 \times 11 \bmod 19 = 55 \bmod 19 = 17$.

UNIT-IV

HASH FUNCTION:

It is a one of the authentication function; it accepts a variable size message M as input and produces a fixed size output.

A hash value 'h' is generated by a function H of the form

h=H (M)

M→ variable length message

H(M)→ fixed length hash value.

The hash code is also referred as Message Digest (MD) or hash value.

The main difference between Hash Function and MAC is a hash code does not use a key but is a function only of the input message.

The hash value is appended to the message at the source at a time when the message is assumed or known to be correct.

The receiver authenticates that message by re-computing the hash value.

Hash functions are often used to determine whether or not data has changed.

Figure 11.1 depicts the general operation of a cryptographic hash function

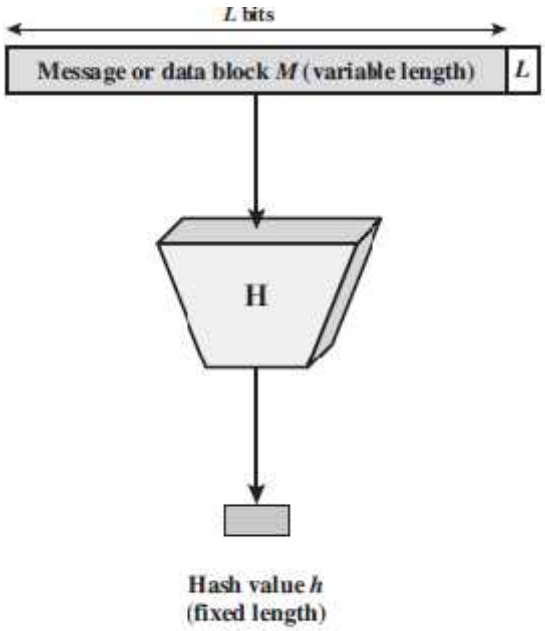


Figure 11.1 Black Diagram of Cryptographic Hash Function; $h = H(M)$

APPLICATIONS OF CRYPTOGRAPHIC HASH FUNCTIONS

It is used in a wide variety of security applications and Internet protocols

Message Authentication

Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent (i.e., contain no modification, insertion, deletion, or replay)

When a hash function is used to provide message authentication, the hash function value is often referred to as a message digest.

Figure 11.2 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows.

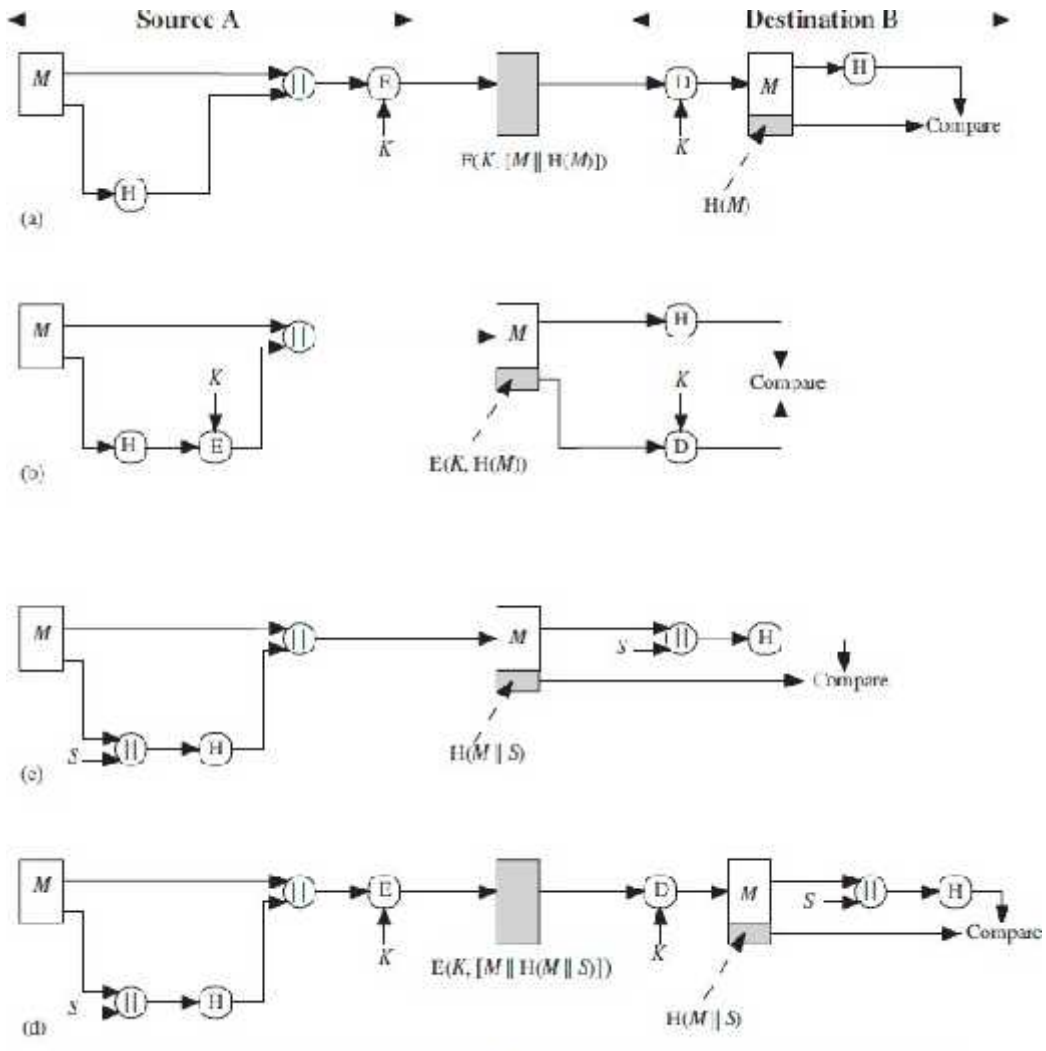


Figure 11.2 Simplified Examples of the Use of a Hash Function for Message Authentication

(a) The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered.

The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

(b) Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality

(c) It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S.A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses, it can recomputed the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

(d) Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.

Digital Signatures

Another important application, which is similar to the message authentication application, is the digital signature.

The operation of the digital signature is similar to that of the MAC. In the case of the digital signature, the hash value of a message is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature.

Figure 11.3 illustrates, in a simplified fashion, how a hash code is used to provide a digital signature.

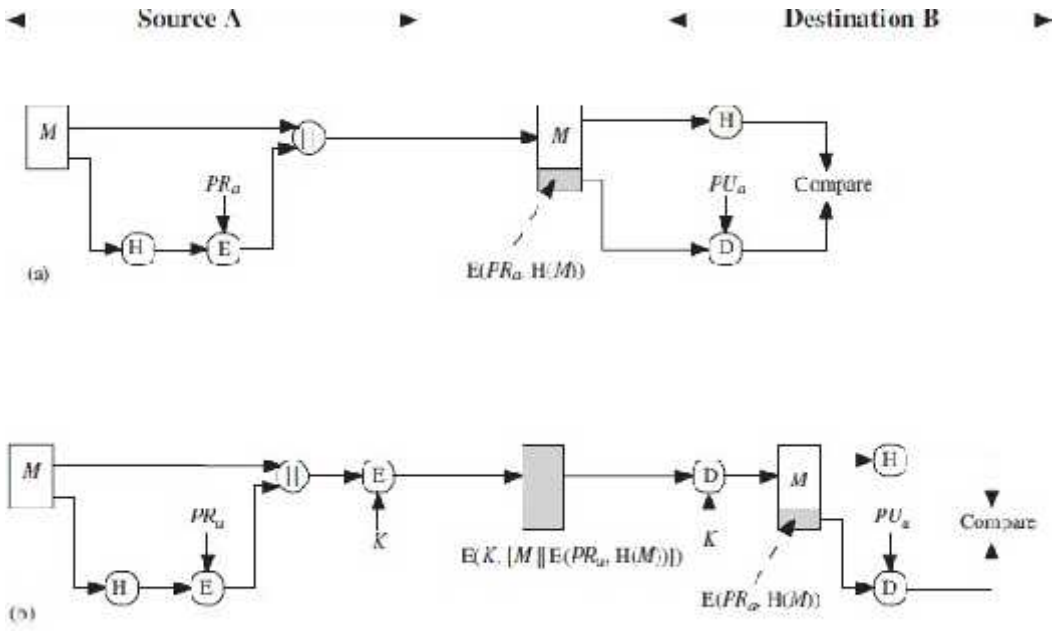


Figure 11.3 Simplified Examples of Digital Signatures

REQUIREMENTS& SECURITY FOR A HASH FUNCTION:

The purpose of a hash function is to produce a "fingerprint" of a file, message or other block of data. To be useful for message authentication, a hash function H must have the following properties:

H can be applied to a block of data of any size

H produces a fixed length output.

H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.

One-way property: - for any given value h, it is computationally infeasible to find x such that H(x)=h. this sometimes referred to in the literature as the one way property.

Weak collision resistance:- for any given block x. it is computationally infeasible to find y≠x with H(y)=H(x). this is referred as weak collision resistance.

Strong collision resistance:- it is computationally infeasible to find any pair (X,Y) such that H(x)=H(y). this is referred as strong collision resistance.

Requirements for a Cryptographic Hash Function H

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one way property)	For any given hash value h, it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

A hash function that satisfies the first five properties in Table 11.1 is referred to as a weak hash function. If the sixth property, collision resistant, is also satisfied, then it is referred to as a strong hash function.

As with encryption algorithms, there are two categories of attacks on hash functions: brute-force attacks and cryptanalysis

Brute-Force Attacks

A brute-force attack does not depend on the specific algorithm but depends only on bit length. In the case of a hash function, a brute-force attack depends only on the bit length of the hash value. A cryptanalysis, in contrast, is an attack based on weaknesses in a particular cryptographic algorithm.

Cryptanalysis

As with encryption algorithms, cryptanalytic attacks on hash functions seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a hash algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack.

That is, an ideal hash algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

SHA(Secure Hash Algorithm):

In recent years, the most widely used hash function has been the Secure Hash Algorithm (SHA).

Introduction:

The Secure Hash Algorithm is a family of cryptographic hash functions developed by the NIST (National Institute of Standards & Technology).

SHA is based on the MD4 algorithm and its design closely models MD5.

SHA-1 is specified in RFC 3174.

Purpose: Authentication, not encryption.

SHA-1 produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512, respectively.

SHA-1 logic:

The algorithm takes a message with maximum of length of less than 264 bits.

Produce output is 160 bits message digest.

The input is processed 512 bits block.

Processed Steps:

Algorithm processing Steps:

Step1: Append Padding Bits

Step 2: Append Length

Step 3: Initialize MD Buffer

Step 4: Process Message in 512 bit (16-Word) Blocks

Step 5: Output

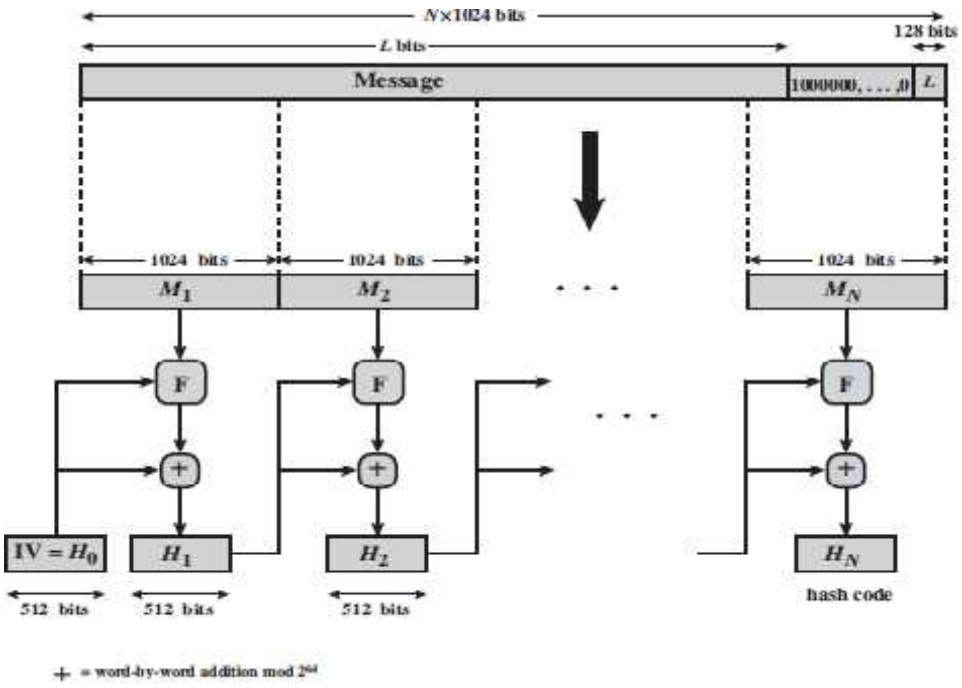


Figure 11.8 Message Digest Generation Using SHA-512

Step-1: **Appending Padding Bits**: The original message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. The padding rules are:

The original message is always padded with one bit "1" first.

Then zero or more bits "0" are padded to bring the length of the message up to 64 bits fewer than a multiple of 512.

Step-2: **append length**: a block of 64 bits is appended to the message. This block is treated as unsigned 64 bit integers (most significant byte first) and contains the length of the original message.

Step-3: **Initialize MD buffer**: 160 bit buffer is used to hold intermediate and final results of the hash function. This buffer can be represented as five 32 bit registers (A, B,C,D,E).

Step 4: Process message in 1024-bit (128-word) blocks. The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure 11.8. The logic is illustrated in Figure 11.9.

Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}. Each round t makes use of a 64-bit value W_t, derived from the current 1024-bit block being processed (M_i). These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant K_t, where $0 \leq t \leq 79$ indicates one of the 80 rounds. These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. The constants provide a "randomized" set of 64-bit patterns, which should eliminate any regularities in the input data. Table 11.4 shows these constants in hexadecimal format (from left to right).

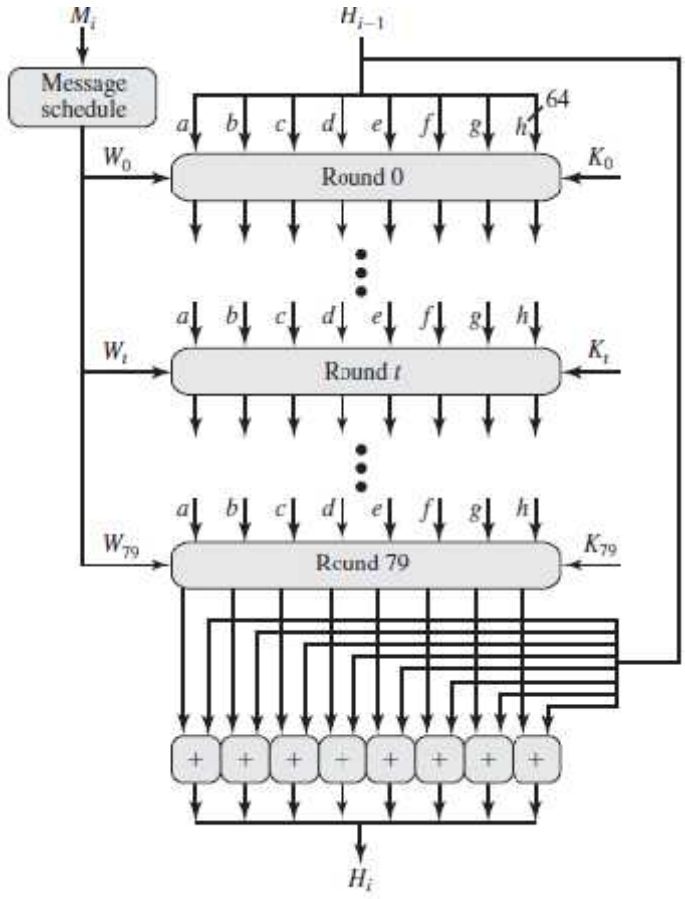


Figure 11.9 SHA-512 Processing of a Single 1024-Bit Block

Step 5 Output. After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.

We can summarize the behavior of SHA-512 as follows:

$$H_0 = \text{IV}$$
$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$
$$MD = H_N$$

where

IV = initial value of the abcdefgh buffer, defined in step 3

abcdefgh_i = the output of the last round of processing of the ith message block

N = the number of blocks in the message (including padding and length fields)

SUM_{64} = addition modulo 2^{64} performed separately on each word of the pair of inputs

MD = final message digest value

MESSAGE AUTHENTICATION

Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent by (i.e., contain no modification, insertion, deletion, or replay) and that the purported identity of the sender is valid.

MESSAGE AUTHENTICATION REQUIREMENTS

In the context of communications across a network, the following attacks can be identified

1. Disclosure: Release of message contents to any person or process not possessing the appropriate cryptographic key.

2. Traffic analysis: Discovery of the pattern of traffic between parties. In a connection oriented application, the frequency and duration of connections could be determined.

3. Masquerade: Insertion of messages into the network from a fraudulent source.

4. Content modification: Changes to the contents of a message, including insertion, deletion, transposition, and modification.

5. Sequence modification: Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

6. Timing modification: Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed.

7. Source repudiation: Denial of transmission of message by source.

8. Destination repudiation: Denial of receipt of message by destination.

MESSAGE AUTHENTICATION FUNCTIONS

Any message authentication or digital signature mechanism has two levels of functionality. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message. there are 3 types of functions that may be used to produce an authenticator.

- **Hash function:** A function that maps a message of any length into a fixed length hash value, which serves as the authenticator
- **Message encryption:** The cipher text of the entire message serves as its authenticator
- **Message authentication code (MAC):** A function of the message and a secret key that produces a fixed-length value that serves as the authenticator

Message Encryption

Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

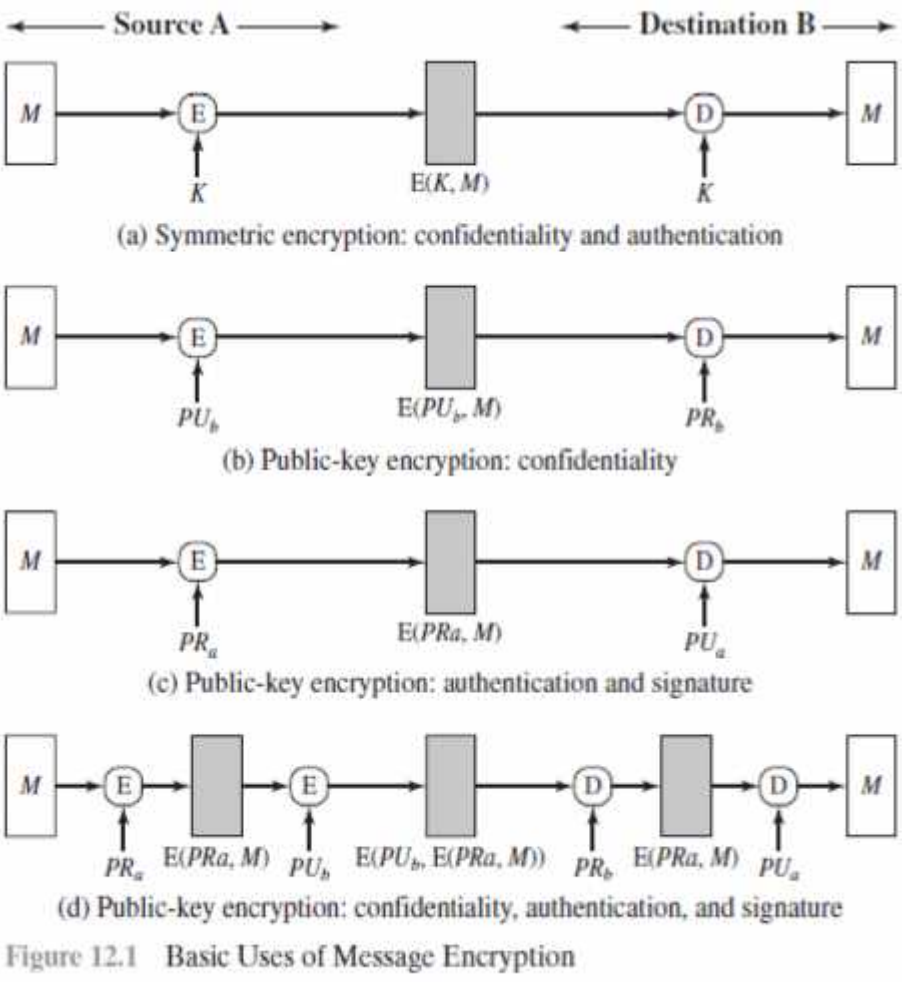


Figure 12.1 Basic Uses of Message Encryption

MESSAGE AUTHENTICATION CODE (MAC)

This authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum** or MAC, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key

When A has a message to send to B, it calculates the MAC as a function of the message and the key

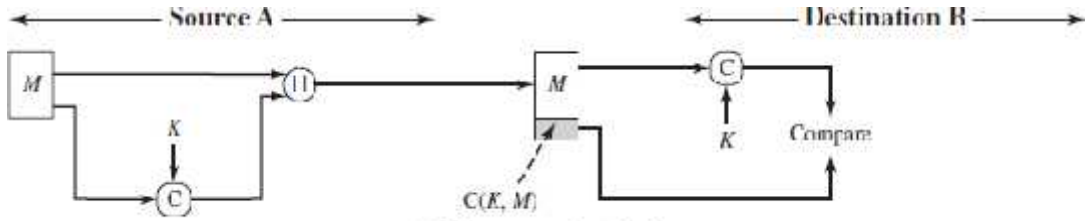
$$MAC = MAC(K, M)$$

where

M = input message
C = MAC function
K = shared secret key
MAC = message authentication code

The message plus MAC are transmitted to the intended recipient.The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC (Figure 12.4a). If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC.
2. The receiver is assured that the message is from the alleged sender. Because no one else



(a) Message authentication

knows the secret key.

SECURITY OF MACS:

Just as with symmetric and public-key encryption, we can group attacks on hash functions and MACs into two categories: **brute-force attacks and cryptanalysis.**

brute-force attacks

A brute-force attack on a MAC is a more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs. The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm, with cost (2m/2). A brute-force attack on a MAC has cost related to min(2k, 2n), similar to symmetric encryption algorithms. It would appear reasonable to require that the key length and MAC length satisfy a relationship such as min(k, n) >= N, where N is perhaps in the range of 128 bits.

cryptanalysis.

As with encryption algorithms, cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a hash or MAC algorithm to cryptanalysis is to compare its strength to the effort required for a bruteforce attack. That is, an ideal hash or MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

HMAC:

In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash function, because they generally execute faster than symmetric block ciphers, and because code for cryptographic hash functions is widely available.

A hash function such as SHA was not designed for use as a MAC and cannot be used directly for that purpose because it does not rely on a secret key. There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm,originally by just pre-pending a key to the message. Problems were found with these earlier, simpler proposals, but they resulted in the development of HMAC.

HMAC Design Objectives:

- To use, without modifications, available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

HMAC Algorithm:

HMAC Algorithm

Figure 12.5 illustrates the overall operation of HMAC. Define the following terms.

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

IV = initial value input to hash function

M = message input to HMAC (including the padding specified in the embedded hash function)

Y_i = i th block of M, $0 \leq i \leq (L-1)$

L = number of blocks in M

b = number of bits in a block

n = length of hash code produced by embedded hash function

K = secret key; recommended length is $\geq n$; if key length is greater than b, the key is input to the hash function to produce an n-bit key

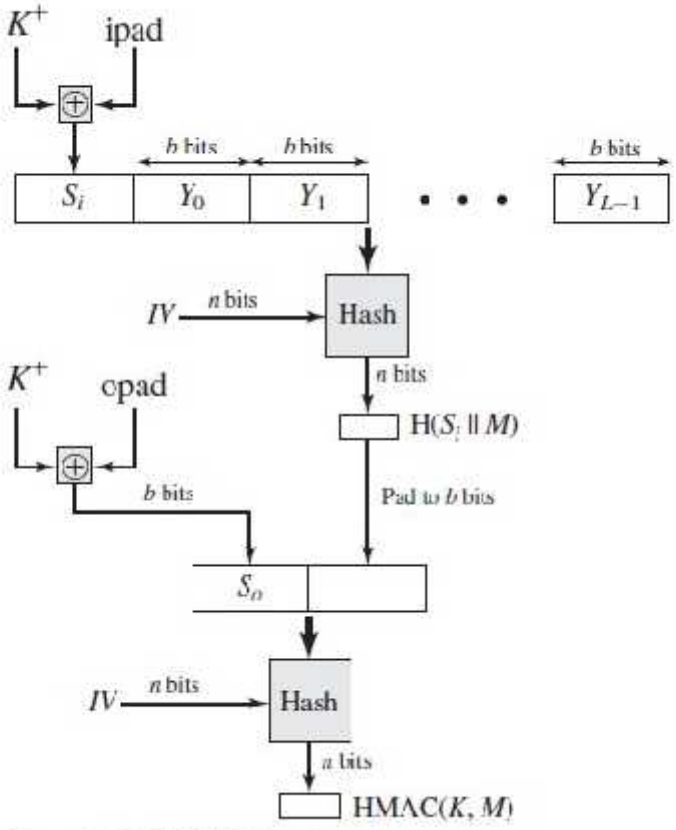
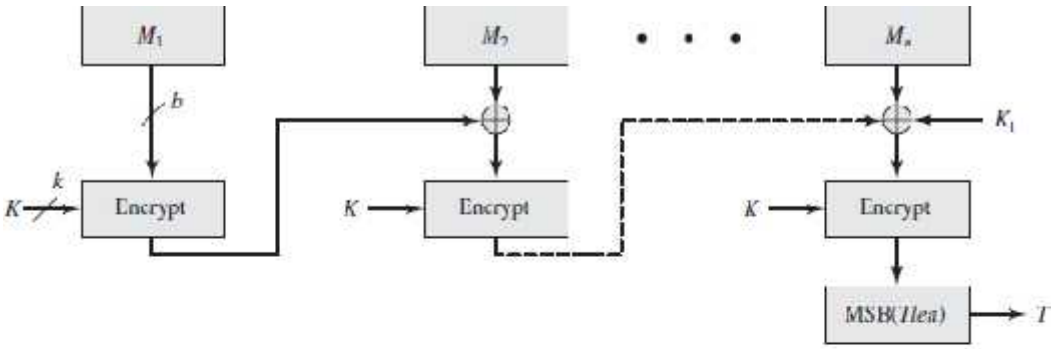


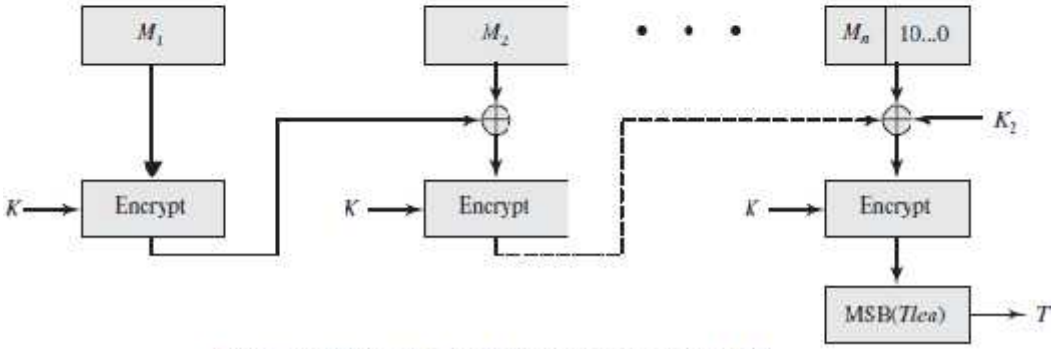
Figure 12.5 HMAC Structure

Cipher-Based Message Authentication Code (CMAC)

First, let us define the operation of CMAC when the message is an integer multiple n of the cipher block length b. For AES, $b = 128$, and for triple DES, $b = 64$. The message is divided into n blocks $(M_1, M_2, \ldots, M_n)$. The algorithm makes use of a k-bit encryption key K and an n-bit constant, K_1. For AES, the key size k is 128, 192, or 256 bits; for triple DES, the key size is 112 or 168 bits. CMAC is calculated as follows (Figure 12.8).



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

Figure 12.8 Cipher-Based Message Authentication Code (CMAC)

$$
\begin{aligned}
C_1 &= \mathrm{E}(K, M_1) \\
C_2 &= \mathrm{E}(K, [M_2 \oplus C_1]) \\
C_3 &= \mathrm{E}(K, [M_3 \oplus C_2]) \\
&\bullet \\
&\bullet \\
&\bullet \\
C_n &= \mathrm{E}(K, [M_n \oplus C_{n-1} \oplus K_1]) \\
T &= \mathrm{MSB}_{Tlen}(C_n)
\end{aligned}
$$

where

T = message authetication code, also referred to as the tag
$Tlen$ = bit length of T
$\mathrm{MSB}_s(X)$ = the s leftmost bitsof the bit string X

If the message is not an integer multiple of the cipher block length, then the final block is padded to the right (least significant bits) with a 1 and as many 0s as necessary so that the final block is also of length b. The CMAC operation then proceeds as before, except that a different n-bit key K_2 is used instead of K_1.

DIGITAL SIGNATURES

A digital signature is an authentication mechanism that enables thecreator of a message to attach a code that acts as a signature. Typicallythe signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees thesource and integrity of the message.

The digital signature standard (DSS) is an NIST standard that uses thesecure hash algorithm (SHA).

Properties

Message authentication protects two parties who exchange messages from any thirdparty. However, it does not protect the two parties against each other. Several formsof dispute between the two are possible.

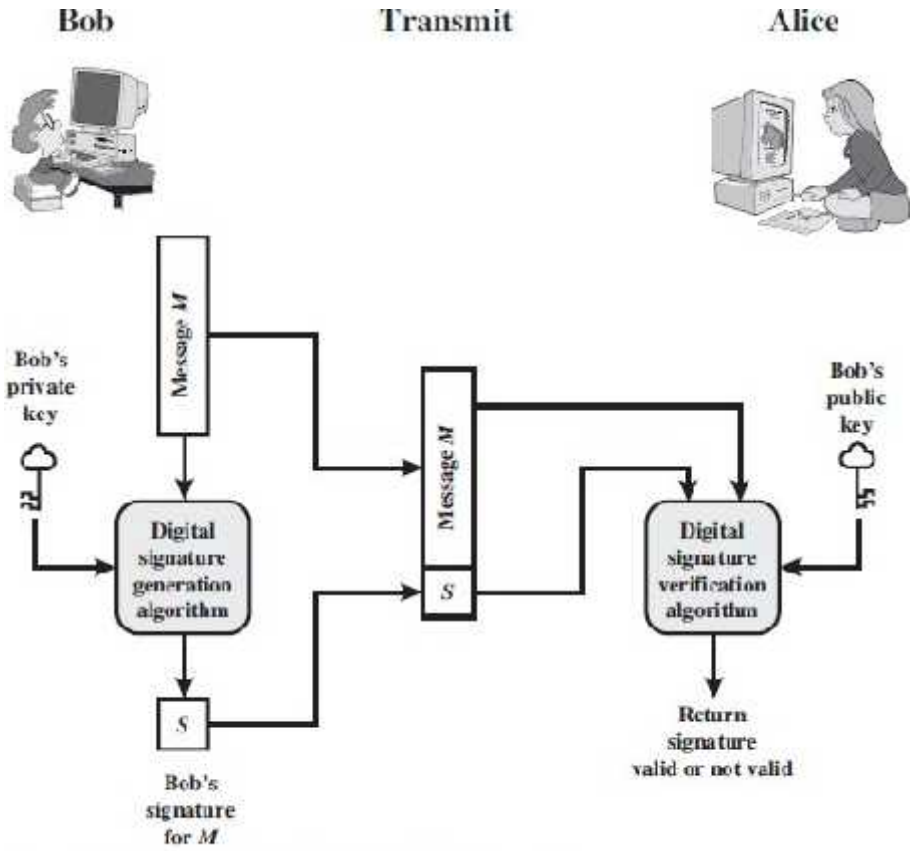


Figure 13.1 Generic Model of Digital Signature Process

DIGITAL SIGNATURE STANDARD

The Digital Signature Standard (DSS) makes use of the Secure Hash Algorithm (SHA) described and presents a new digital signature technique, the Digital Signature

Algorithm (DSA).

This latest version incorporates digital signature algorithms based on RSA and on elliptic curve cryptography. In this section, we discuss the original DSS algorithm. The DSS uses an algorithm that is designed to provide only the digital signature function.

Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

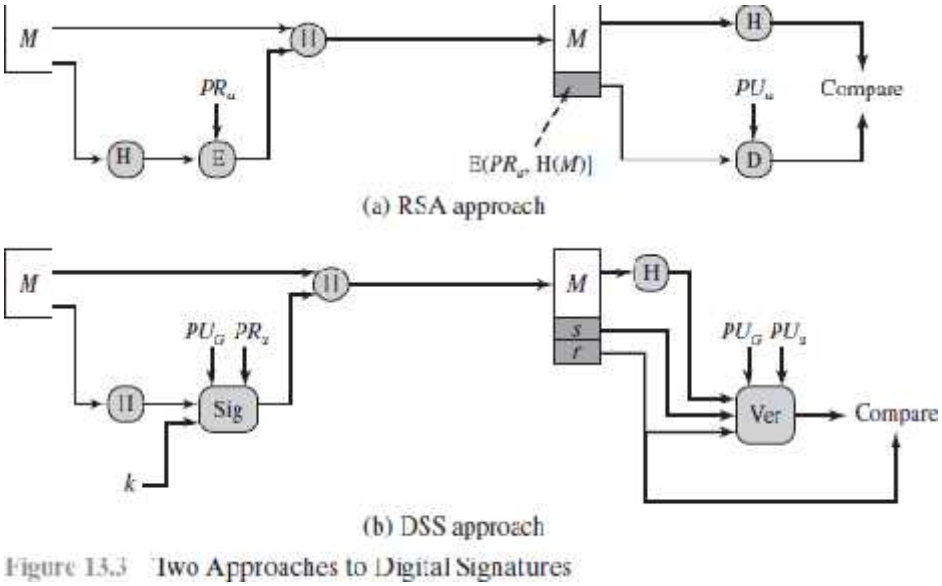


Figure 13.3 Two Approaches to Digital Signatures

In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are thentransmitted. The recipient takes the message and produces a hash code.

The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

Digital Signature Algorithm

The DSA is based on the difficulty of computing discrete logarithms and is based on schemes originally presented by ElGamal and Schnorr. The DSA signature scheme has advantages, being both smaller (320 vs 1024bit) and faster over RSA. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique

DSA typically uses a common set of global parameters (p,q,g) for a community of clients, as shown. A 160-bit prime number q is chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p – 1). Finally, g is chosen to be of the form $h^{(p-1)/q}$ *mod p* where h is an integer between 1 and (p – 1) withthe restriction that g must be greater than 1. Thus, the global public key components of DSA have the same for as in the Schnorr signature scheme.

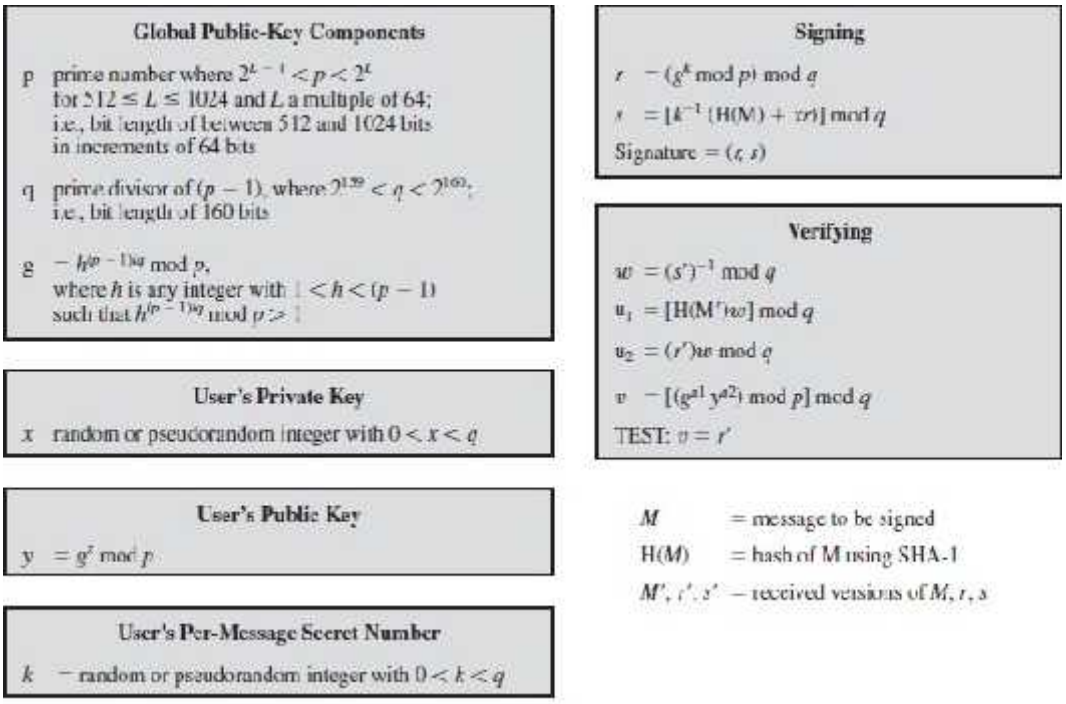
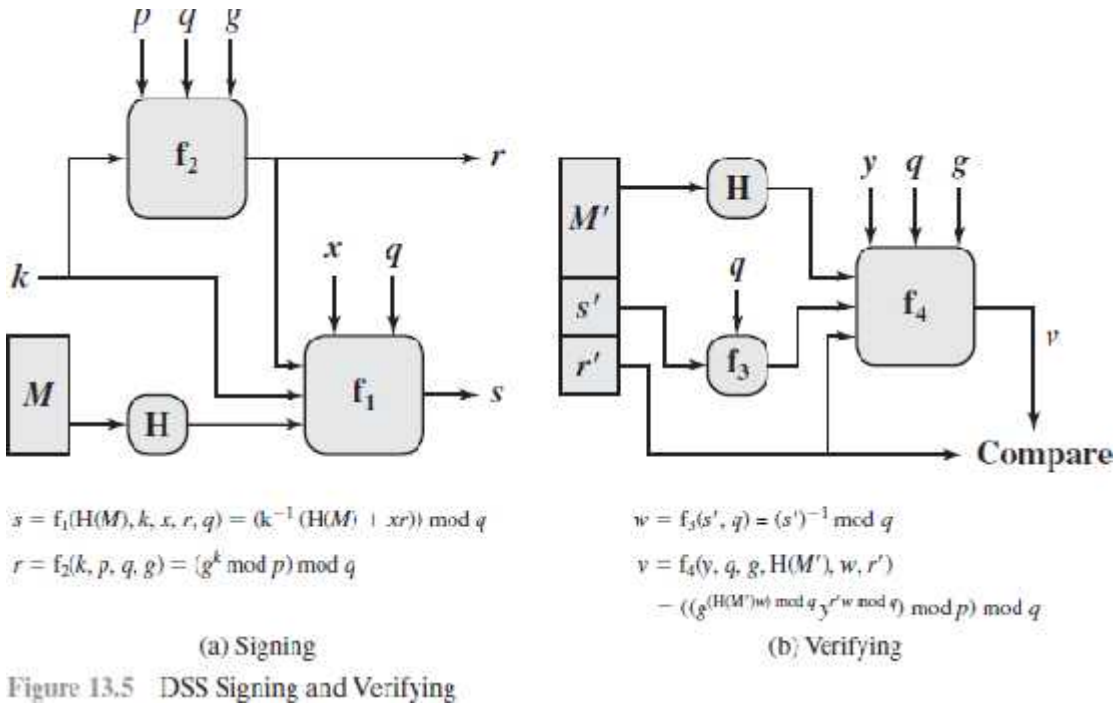


Figure 13.4 The Digital Signature Algorithm (DSA)

Signing and Verifying

The structure of the algorithm, as revealed here is quite interesting. Note that the test at the end is on the value r, which does not depend on the message at all. Instead, r is a function of k and the three global public-key components. The multiplicative inverse of k (mod q) is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key.



$s = f_1(H(M), k, x, r, q) = (k^{-1}(H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

(a) Signing

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

$= ((g^{(H(M'))w \bmod q} y^{r'w \bmod q}) \bmod p) \bmod q$

(b) Verifying

Figure 13.5 DSS Signing and Verifying

KEY MANAGEMENT AND DISTRIBUTION

Symmetric Key Distribution Using Symmetric Encryption For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Therefore, the term that refers to the means of delivering a key to two parties who wish to exchange data, without allowing others to see the key. For two parties A and B, key distribution can be achieved in a number of ways, as follows:

1. A can select a key and physically deliver it to B.
2. A third party can select the key and physically deliver it to A and B.
3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.
4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.

Physical delivery (1 & 2) is simplest - but only applicable when there is personal contact between recipient and key issuer. This is fine for link encryption where devices & keys occur in pairs, but does not scale as number of parties who wish to communicate grows.

3 is mostly based on 1 or 2 occurring first.A third party, whom all parties trust, can be used as a trusted intermediary to mediate the establishment of secure communications between them (4). Must trust intermediary not to abuse the knowledge of all session keys. As number of parties grow,some variant of 4 is only practical solution to the huge growth in number of keys potentially needed.

Key distribution centre:

The use of a key distribution centeris based on the use of a hierarchy of keys. At a minimum, two levels of keys are used.

Communication between end systems is encrypted using a temporary key, often referred to as a Session key.

Typically, the session key is used for the duration of a logical connection and then discarded

Master key is shared by the key distribution center and an end system or user and used to encrypt the session key.

A Key Distribution Scenario

The key distribution concept can be deployed in a number of ways. A typical scenario is illustrated in Figure 14.3, which is based on a figure in [POPE79]. The scenario assumes that each user shares a unique master key with the key distribution center (KDC).

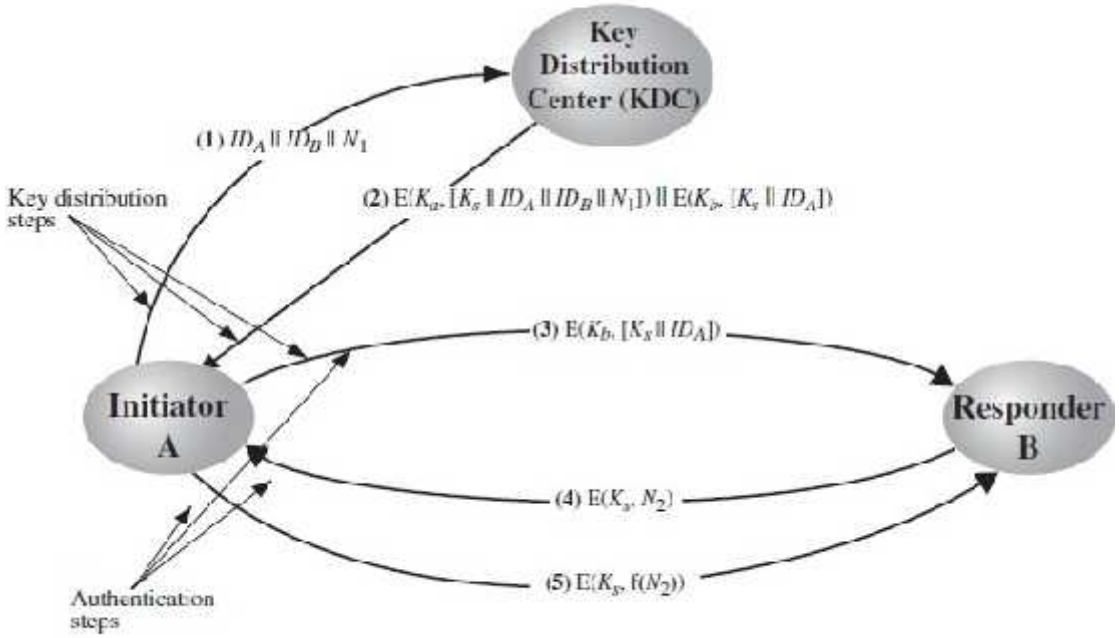


Figure 14.3 Key Distribution Scenario

Symmetric Key Distribution Usingasymmetric Encryption

Because of the inefficiency of public key cryptosystems, they are almost never usedfor the direct encryption of sizable block of data, but are limited to relatively smallblocks. One of the most important uses of a public-key cryptosystem is to encryptsecret keys for distribution.Wesee many specific examples of this in Part Five. Here,we discuss general principles and typical approaches.

Simple Secret Key Distribution

An extremely simple scheme was put forward by Merkle [MERK79], as illustrated in Figure 14.7. If A wishes to communicate with B, the following procedure is employed:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B consisting of PU_a and an identifier of A, ID_A.
2. B generates a secret key, K_s, and transmits it to A, which is encrypted with A's public key.
3. A computes $D(PR_a, E(PU_a, K_s))$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s.
4. A discards PU_a and PR_a and B discards PU_a.

A and B can now securely communicate using conventional encryption and the session key K_s. At the completion of the exchange, both A and B discard K_s.

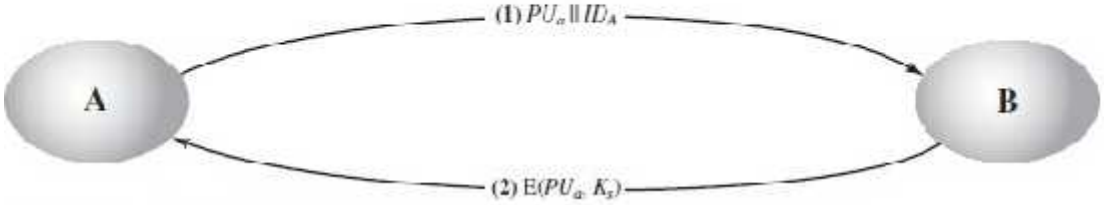


Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key

UNIT-V

SECURITY AT APPLICATION LAYER

Syllabus:

Network Security-I Security at application layer: PGP and S/MIME, Security at the Transport Layer: SSL and TLS

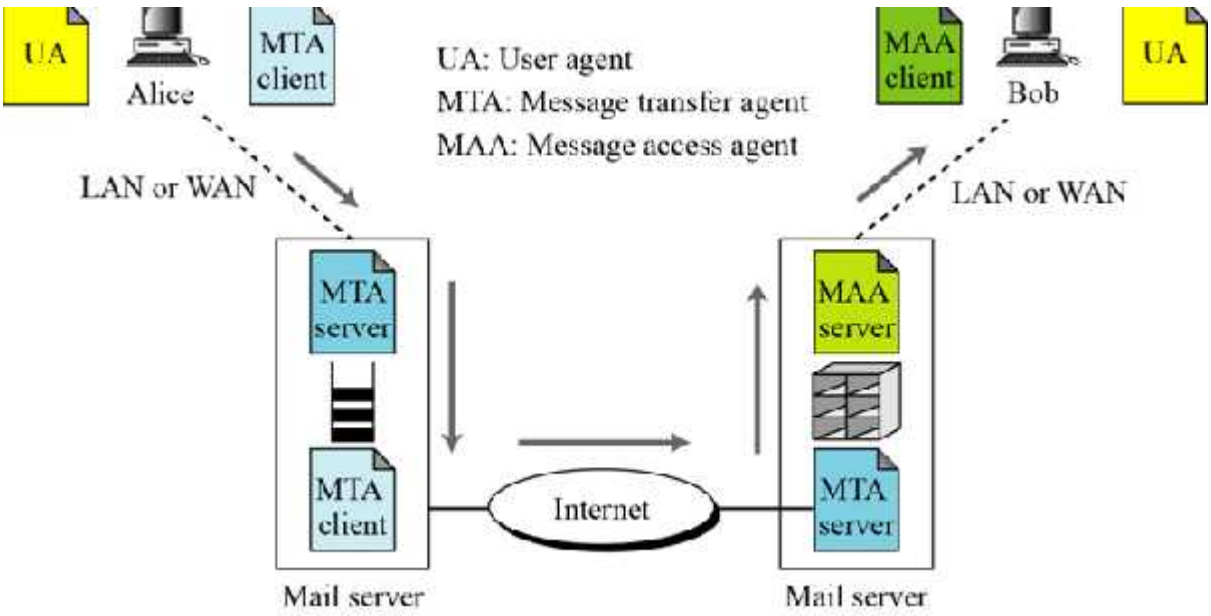
Topic 01:

Assume that Alice is working in an organization that runs an e-mail server; every employee is connected to the e-mail server through a LAN. Or alternatively, Alice could be connected to the e-mail server of an ISP through a WAN (telephone line or cable line). Bob is also in one of the above two situations.

The administrator of the e-mail server at Alice's site has created a queuing system that sends e-mail to the Internet one by one. The administrator of the e-mail server at Bob's site has created a mailbox for every user connected to the server; the mailbox holds the received messages until they are retrieved by the recipient.

E-Mail :

Architecture:



E-Mail Security:

- **Sending an email is an one time activity.**

- In IP Sec or SSl, We assume that two parties create a session between themselves and exchange data in both the directions.
- In e-mail, there is no session. Alice and Bob cannot create a session.
- Alice sends a message to bob, sometimes later, bob read the message, and may or maynot send a reply.

Cryptographic Algorithms:

- If e-mail is one time activity, how can the sender and receiver agree on a cryptographic algorithm to use for email-security.
- If there is no session and no handshaking to negotiate algorithms for encryption/decryption and hashing.
- A better solution is to define a set of algorithms.
- For example, Alice can choose triple DES for encryption/decryption and MD5 for hashing.
- In e-mail security, the sender of the message needs to include the name or identifiers of the algorithms used in the message.

❑ ***Cryptographic Secrets*** The same problem for the cryptographic algorithms applies to thecryptographic secrets (keys). If there is no negotiation, how can the two parties establish secrets betweenthemselves? Alice and Bob could use asymmetric-key algorithms for authentication and encryption, whichdo not require the establishment of a symmetric key. However, as we have discussed, the use of asymmetric-key algorithms is very inefficient for the encryption/decryption of a long message.

Most e-mail security protocols today require that encryption/decryption be done using a symmetric-key algorithm and a one-time secret key sent with the message. Alice can create a secret key and send it with the message she sends to Bob. To protect the secret key from interception by Eve, the secret key is encrypted with Bob's public key. In other words, the secret key itself is encrypted.

❑ ***Cryptographic Secrets*** The same problem for the cryptographic algorithms applies to thecryptographic secrets (keys). If there is no negotiation, how can the two parties establish secrets betweenthemselves? Alice and Bob could use asymmetric-key algorithms for authentication and encryption, whichdo not require the establishment of a symmetric key. However, as we have discussed, the use of asymmetric-key algorithms is very inefficient for the encryption/decryption of a long message.

Most e-mail security protocols today require that encryption/decryption be done using a symmetric-key algorithm and a one-time secret key sent with the message. Alice can create a secret key and send it with the message she sends to Bob. To protect the secret key from interception by Eve, the secret key is encrypted with Bob's public key. In other words, the secret key itself is encrypted.

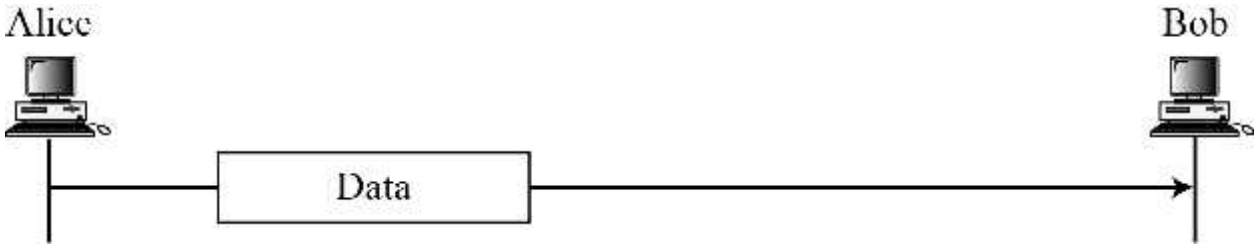
- **PGP: Pretty Good Privacy**

Privacy (PGP). PGP was invented by Phil Zimmermann to provide e-mail with privacy, integrity, and authentication. PGP can be used to create a secure e-mail message or to store a file securely for future retrieval.

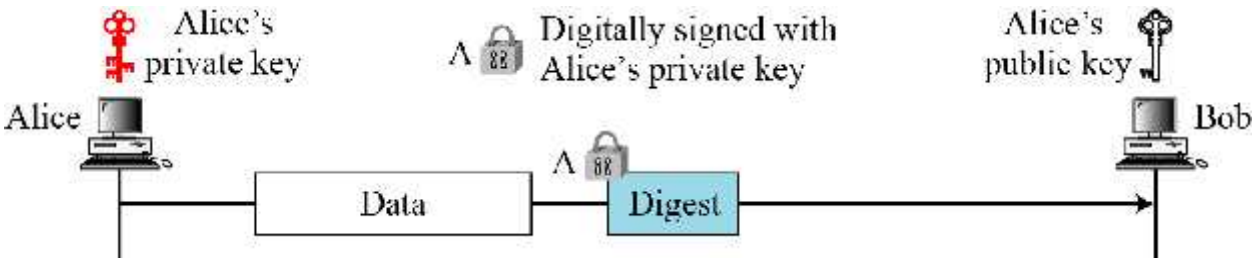
2.1 Scenarios

Let us first discuss the general idea of PGP, moving from a simple scenario to a complex one. We use the term "Data" to show the message or file prior to processing.

❑ ***Plaintext*** The simplest scenario is to send the e-mail message (or store the file) in plaintext as shown in Fig. 16.2. There is no message integrity or confidentiality in this scenario. Alice, the sender, composes a message and sends it to Bob, the receiver. The message is stored in Bob's mailbox until it is retrieved by him.

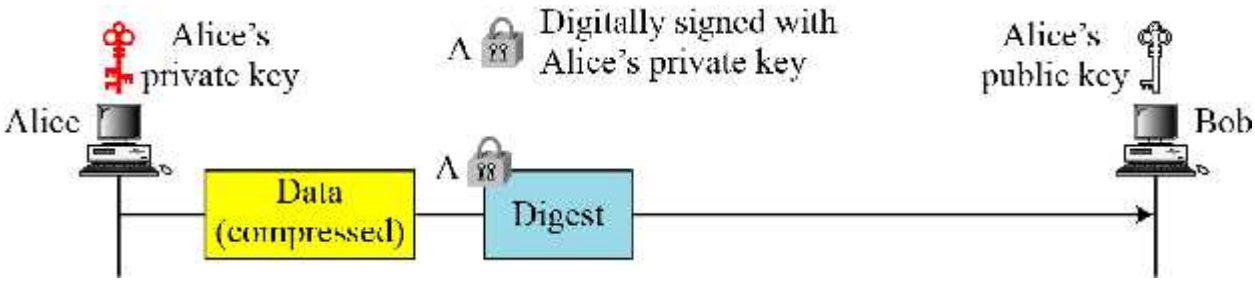


❑ ***Message Integrity*** Probably the next improvement is to let Alice sign the message. Alice creates a digest of the message and signs it with her private key. When Bob receives the message, he verifies the message by using Alice's public key. Two keys are needed for this scenario. Alice needs to know her private key; Bob needs to know Alice's public key. Figure 16.3 shows the situation.



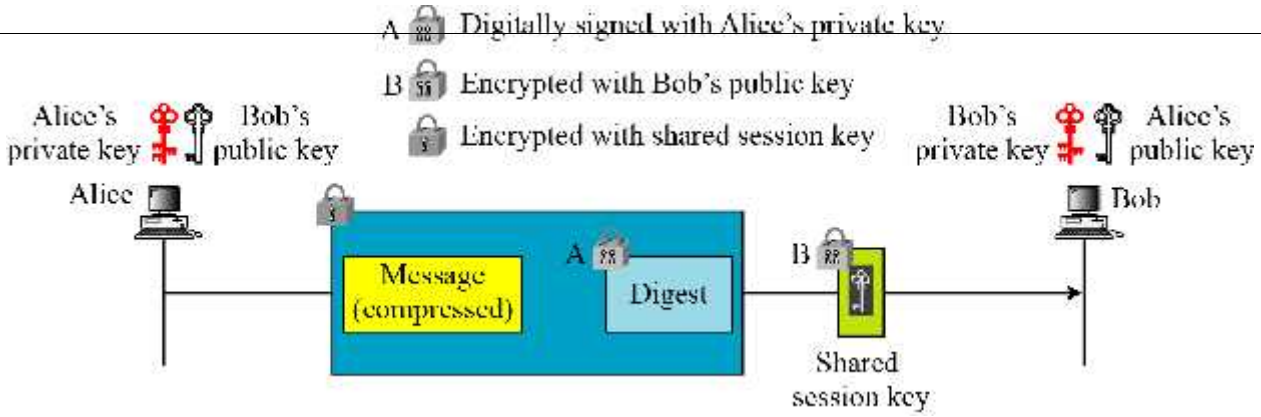
Compression:

- A further improvement is to compress the message and digest to make the packet more compact.
- This improvement has no security benefit, but it eases the traffic.



Confidentiality with onetime session key:

- Confidentiality in an e-mail system can be achieved using conventional encryption with a one-time session key.
- Alice can create a session key,use the session key to encrypt the messge and the digest , and send the key itself with the message.
- However to protect the session key Alice encrypts it with Bob's public key.
- When Bob receives the packet, he first decrypts the key, using his private key to remove the key.
- He then uses the session key to decrypt the rest of the message.
- After, decompressing the rest of the message, Bob creates the digest of the message and checks to see if it is equal to the digest sent by Alice.
- If it is, then message is authentic.



Code Conversion:

- Another service provided by PGP is code conversion.
- PGP uses Radix-64 conversion

Segmentation:

- PGP allows segmentation of the message after it has been converted to Radix-64 to make each transmitted unit the uniform size as allowed by the underlying e-mail protocol.

Key rings:

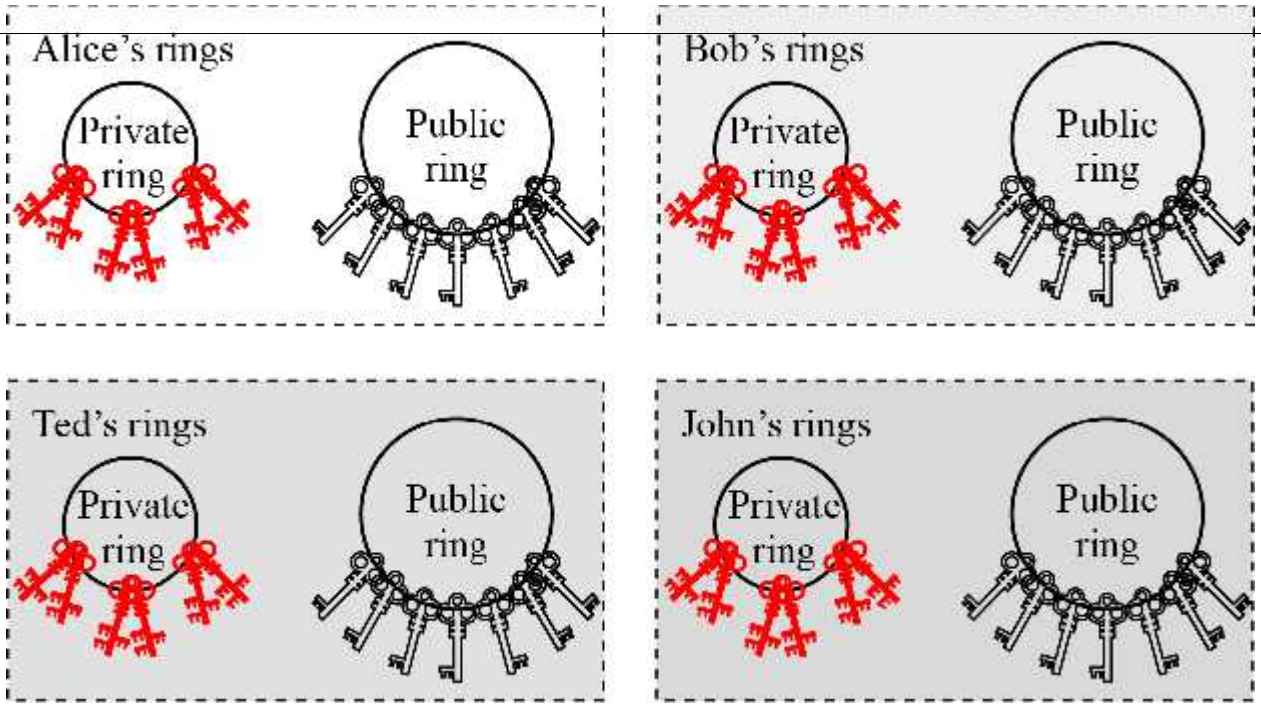
- Alice needs to send messages to many people. She needs **key rings.**
- In this case , Alice needs a ring of public keys, with a key belonging to each person with whom Alice needs to correspond .
- In addition, the PGP designers specified a ring of private/public keys.
- One reason that,Alice may wish to change her pair of keys from time to time.
- Another reason is that Alice may need to correspond with different groups of people.
- Alice may wish to use a different pair for each group.

1. Alice needs to send a message to another person in the community.

    a) She uses her private key to sign the digest.
    b) She uses the receiver's public key to encrypt a newly created session key.
    c) She encrypts the message and signed digest with the session key created.

2. Alice receives a message from another person in the community

    a) She uses her private key to decrypt the session key.
    b) She uses the session key to decrypt the message and digest.
    c) She uses her public key to verify the digest.



PGP Algorithms:

The following algorithms are used in PGP.

Public key algorithms:

Table 16.1 *Public-key algorithms*

ID	*Description*
1	RSA (encryption or signing)
2	RSA (for encryption only)
3	RSA (for signing only)
16	ElGamal (encryption only)
17	DSS
18	Reserved for elliptic curve
19	Reserved for ECDSA
20	ElGamal (for encryption or signing)
21	Reserved for Diffie-Hellman
100 - 110	Private algorithms

Symmetric key Algorithms:

The symmetric-key algorithms that are used for conventional encrypting are shown below

Table 16.2 *Symmetric-key algorithms*

ID	*Description*
0	No Encryption
1	IDEA
2	Triple DES
3	CAST-128
4	Blowfish
5	SAFER-SK128
6	Reserved for DES/SK
7	Reserved for AES-128
8	Reserved for AES-192
9	Reserved for AES-256
100–110	Private algorithms

Hash Algorithm:

Table 16.3 *Hash Algorithms*

ID	*Description*
1	MD5
2	SHA-1
3	RIPE-MD/160
4	Reserved for double-width SHA
5	MD2
6	TIGER/192
7	Reserved for HAVAL
100–110	Private algorithms

Compression Algorithms:

Table 16.4 *Compression methods*

ID	*Description*
0	Uncompressed
1	ZIP
2	ZLIP
100–110	Private methods

PGP Certificates:

X.509 Certificates

- Protocols that use X.509 certificates depend on the hierarchical structure of the trust.
- In X.509, there is a single path from the fully trusted authority to any certificate.

➢ **PGP Certificates**

- In PGP, there is no need for CAs; anyone in the ring can sign a certificate for anyone else in the ring.

- In PGP, there can be multiple paths from fully or partially trusted authorities to any subject.

Trusts and Legitimacy

- The entire operation of PGP is based on introducer trust, the certificate trust, and the legitimacy of the public keys.

Introducer Trust Levels:

- With the lack of a central authority, it is obvious that the ring cannot be very large if every user in the PGP ring of users has to fully trust everyone else.
- To solve problem, PGP allows different levels of trust.
- The number of levels is mostly implementation dependent, but for simplicity, let us assign three levels of trust to any introducer: none, partial and full.

Certificate Trust Levels:

- When Alice receives a certificate from an introducer, she stores the certificate under the name of the subject.
- She assigns a level of trust to this certificate.

1. Bob issues two certificates, one for Linda and one for Lesely.
2. Anne issues a certificate for John. Alice stores this certificate and public key under John's name , but assigns a partial level for this certificate.
3. Janette issues two certificates, one for John, and one for Lee. Alice stores John's certificate.
4. John issues a certificate for Lliz. Alice can discard or keep this certificate with a signature trust of none.

Key Legitimacy:

The purpose of using introducer and certificate trusts is to determine the legitimacy of a public key.

For example, suppose we assign the following weights to certificate trust levels:

1. A weight of 0 to a nontrusted certificate.
2. A weight of ½ to a certificate with partial trust
3. A weight of 1 to a certificate with full trust

Starting the ring:

1. Alice can physically obtain Bob's public key. For example ,Alice and Bob can meet personally and exchange a public key written on a piece of paper or to a disk.
2. If bob's voice recognizable to Alice , Alice can call him and obtain his public key on the phone.
3. A better solution proposed by PGP is for Bob to send his public key to Alice by e-mail.
4. In PGP, nothing prevents Alice from getting Bob's public key from a CA in a separate procedure. She can then insert the public key in a public key ring.

- **Key Ring tables:**

User Id: It is usually the e-mail address of the user.

Key Id: This column uniquely defines a public key among user's public keys. It is calculated as (key mod 2^{64})

Public key: This column just lists the public key belonging to a particular private key/public key pair.

Encrypted private key: This column shows the encrypted value of the private key in the private/public key.

Timestamp: This column holds the time and date of the key pair creation.

Fig: Format of private key ring table

User ID	Key ID	Public key	Encrypted private key	Timestamp
⋮	⋮	⋮	⋮	⋮

Public key ring table:

User Id: As in the private key ring table,the user ID is usually the email address of the entity.

Key Id: As in the private key ring table, the key ID is the first(least significant) 64 bits of the public key.

Public key: This is the public key of the entity**.**

Producer Trust: This column defines the producer lever of trust.

Certificate: This column holds the certificate or certificates signed by other entities for this entity. A user ID may have more than one certificate.

Certificate Trust: This column represents the certificate trust or trusts.

Key Legitimacy: This value is calculated by PGP based on the value of the certificate trust and the predefined weight for each certificate trust.

Timestamp: This column holds the time and date of the column creation

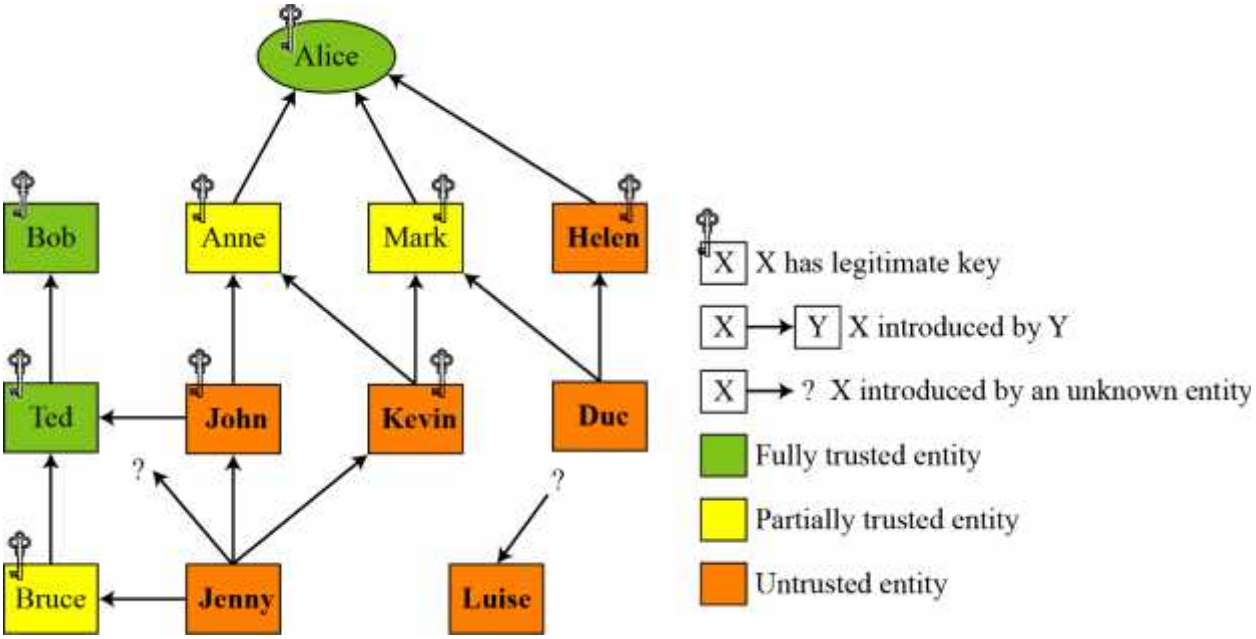
Fig: Format of public key ring table:

User ID	Key ID	Public key	Producer trust	Certificate(s)	Certificate trust(s)	Key Legitimacy	Timestamp
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Trusted model in PGP:

As Zimmermann has proposed, we can create a trust model for any user in a ring with the user as the center of activity. Such a model can look like the one shown in Fig. 16.9. The figure shows the trust model for Alice at some moment. The diagram may change with any changes in the public key ring table.

Let us elaborate on the figure. Figure 16.9 shows that there are three entities in Alice's ring with full trust (Alice herself, Bob, and Ted). The figure also shows three entities with partial trust (Anne, Mark, and Bruce). There are also six entities with no trust. Nine entities have a legitimate key. Alice can encrypt a message to any one of these entities or verify a signature received from one of these entities (Alice's key is never used in this model). There are also three entities that do not have any legitimate keys with Alice.

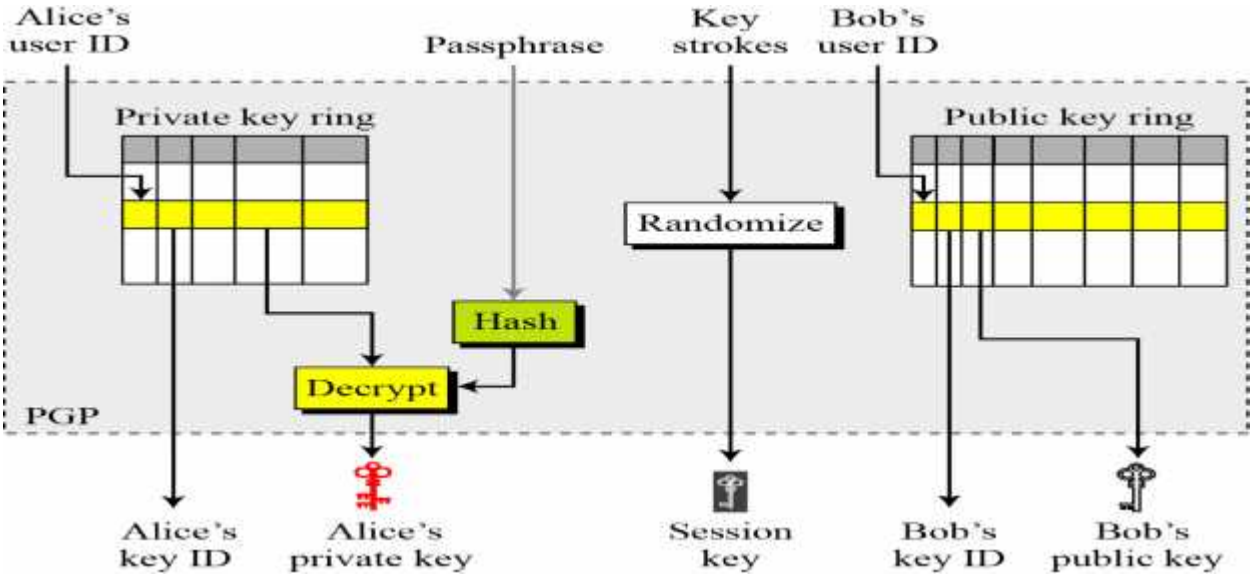


Key invocation:

- It may become necessary for an entity to revoke his or her public key from the ring.
- This may happen if the owner of the key feels that the key is compromised (stolen, for example) or just too old to be safe.

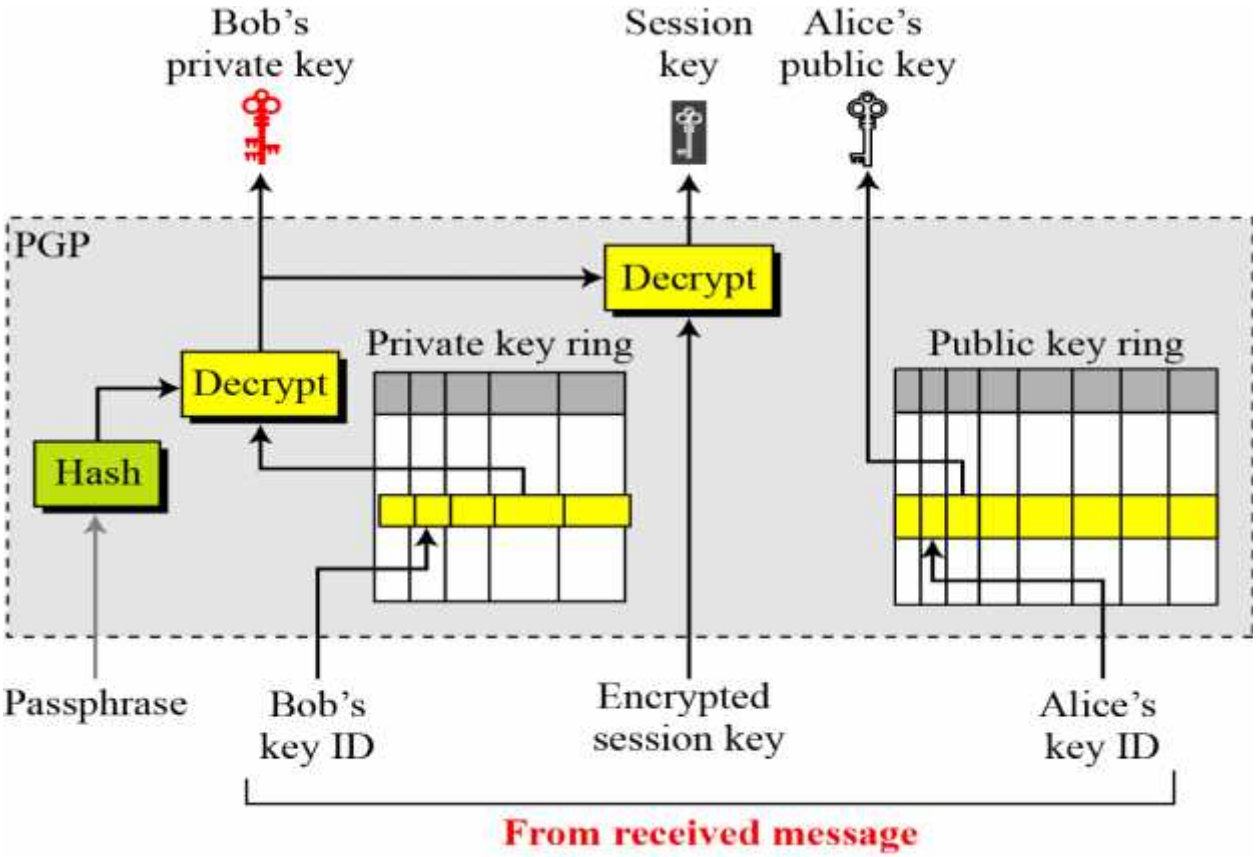
- **Extracting information from rings:**

Sender Site:



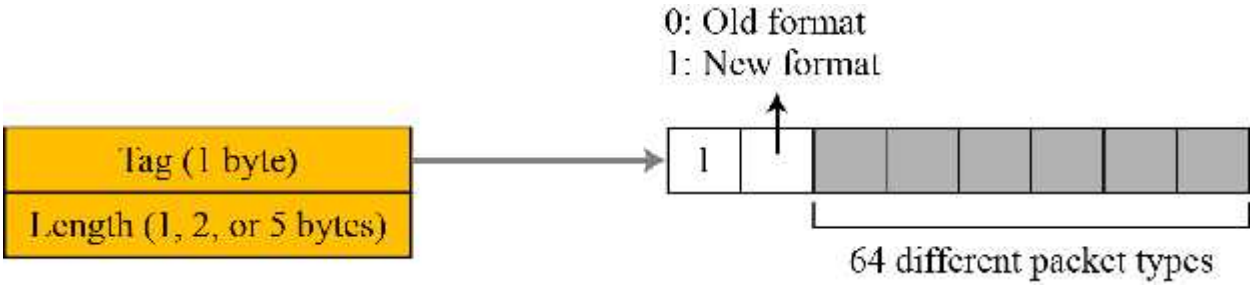
Receiver site:

Extracting information at the receiver site:



- **PGP Packets:**
  - A message in PGP consists of one or more packets.

Fomat of packet header



Tag: The recent format for this field defines a tag as an 8-bit flag; the first bit is always 1. The second bit is 1 If we are using the latest version.

Table 16.12 *Some commonly used packet types*

Value	*Packet type*
1	Session key packet encrypted using a public key
2	Signature packet
5	Private-key packet
6	Public-key packet
8	Compressed data packet
9	Data packet encrypted with a secret key
11	Literal data packet
13	User ID packet

Literal data packet:

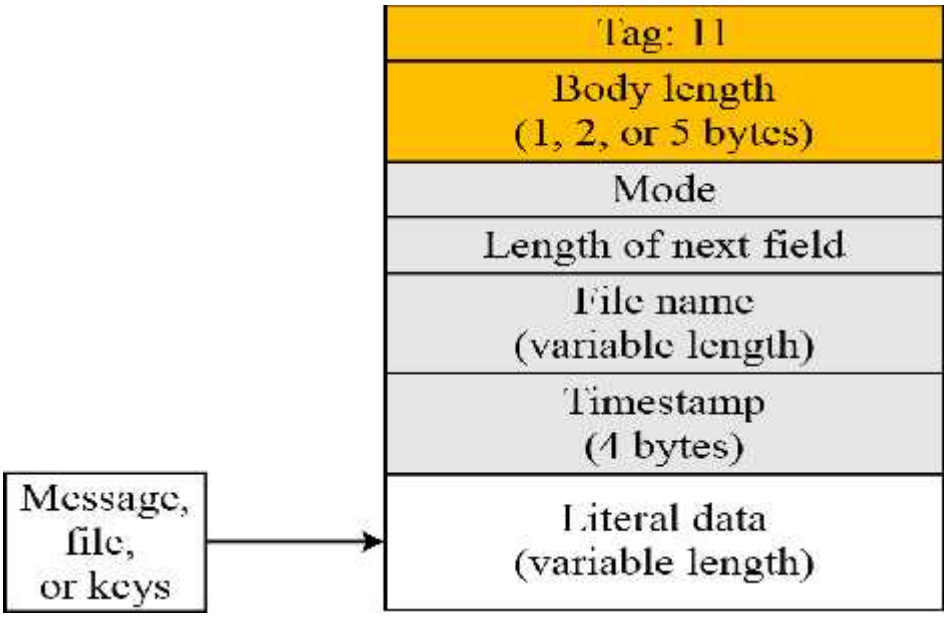
Mode: This one byte field defines how data is written to the packet.

Length of next field: This one byte field defines the length of the next field.

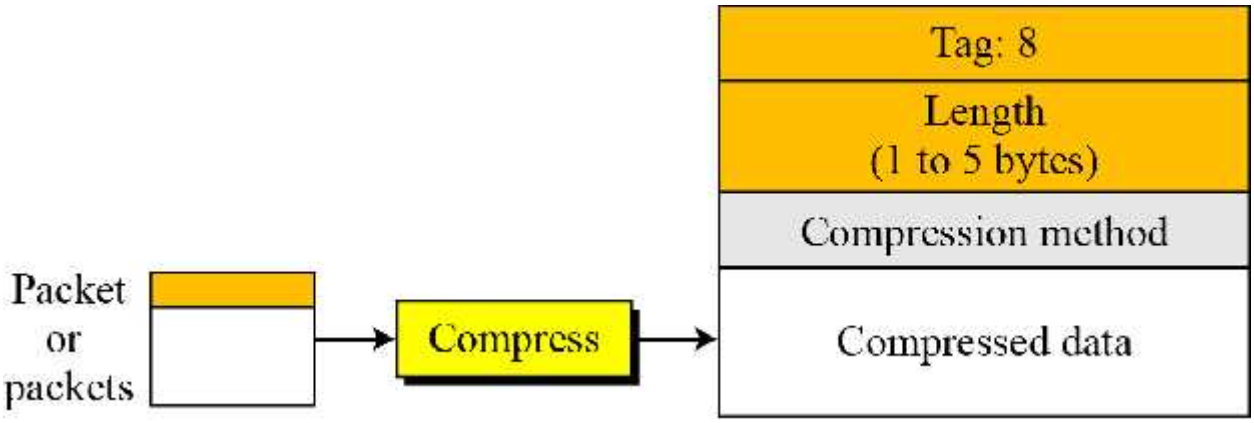
File name: This variable length field defines the name of the file or message as an ASCII string.

Time stamp: This four byte field defines the time of creation or last modification of the message. The value can be 0,which means that the user chooses not to specify a time.

Literal data: This variable length field carries the actual data in text or binary (depending on the value of the mode field).



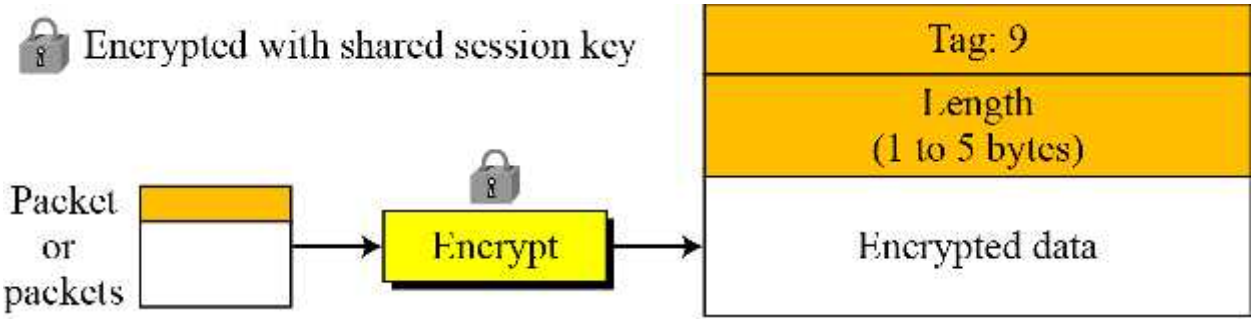
Compressed data packet:



Compressed method: This one byte field defines the compression method used to compress the data.The values are 1 and 2.

Compressed data: This variable length field carries the data after compression.

Encrypted data packet:



Signature packet:

Version: This one byte field defines the PGP version that is being used.

Length: This field was originally designed to show the length of the next two fields, but because the size of these fields is now fixed, the value of this field is 5.

Signature type: This one byte field defines the purpose of the signature, the document it signs.

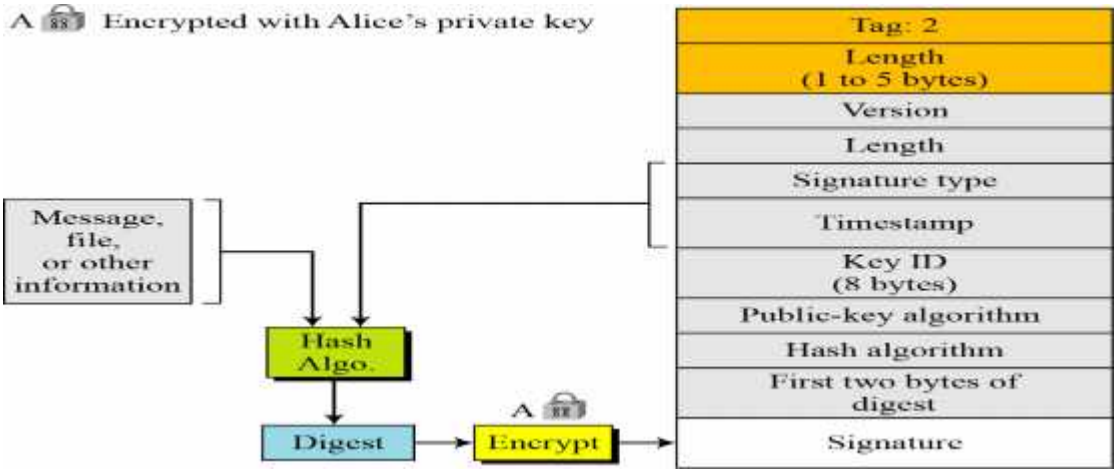
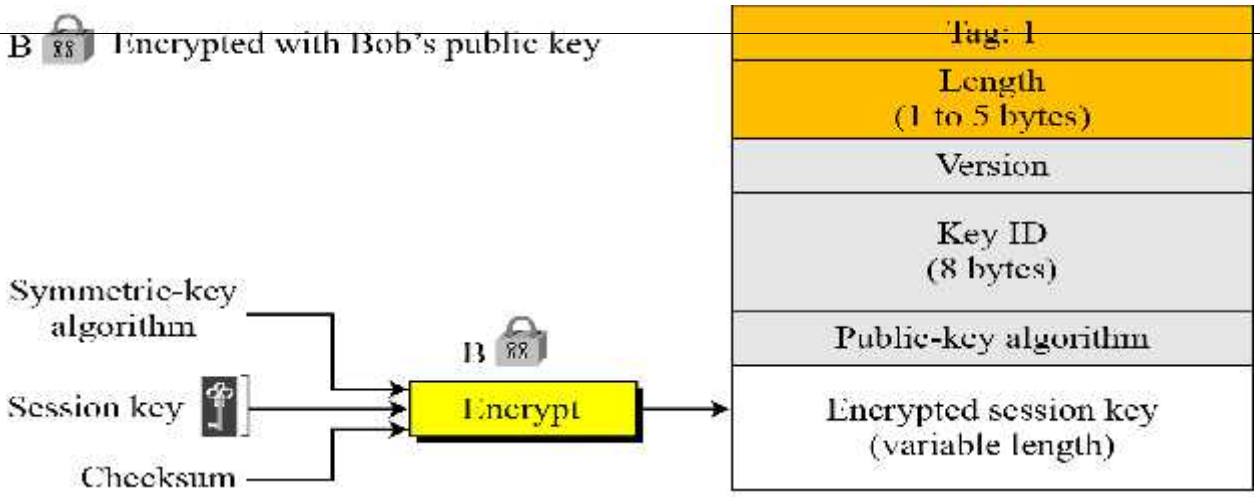


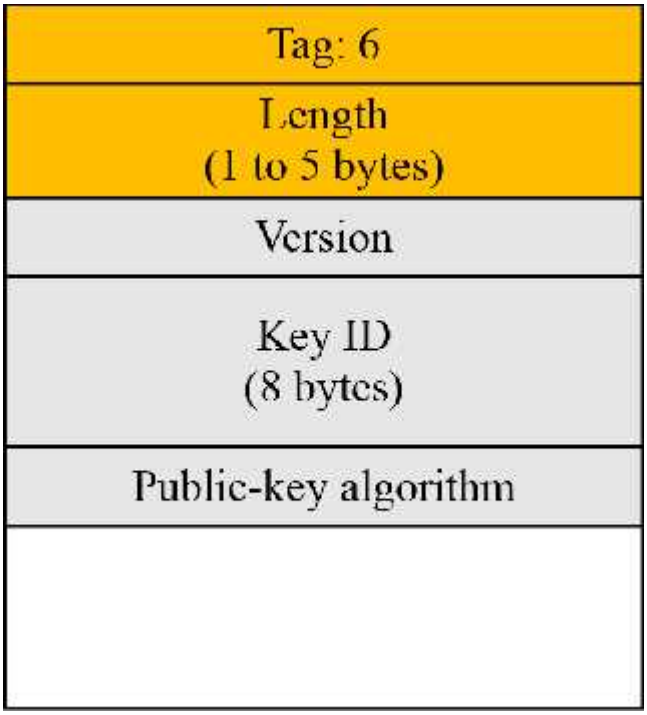
Table 16.13 *Some signature values*

Value	*Signature*
0x00	Signature of a binary document (message or file).
0x01	Signature of a text document (message or file).
0x10	Generic certificate of a user ID and public-key packet. The signer does not make any particular assertion about the owner of the key.
0x11	Personal certificate of a user ID and public-key packet. No verification is done on the owner of the key.
0x12	Casual certificate of a User ID and public-key packet. Some casual verification done on the owner of the key.
0x13	Positive certificate of a user ID and public-key packet. Substantial verification done.
0x30	Certificate revocation signature. This removes an earlier certificate (0x10 through 0x13).

Session key packet Encrypted with public key:

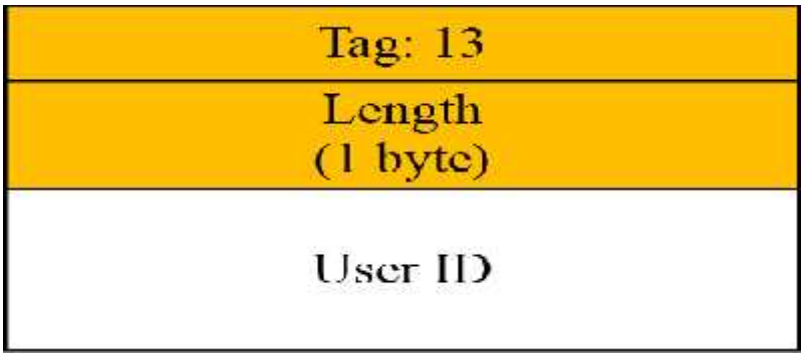


Public key packet:



User id packet:

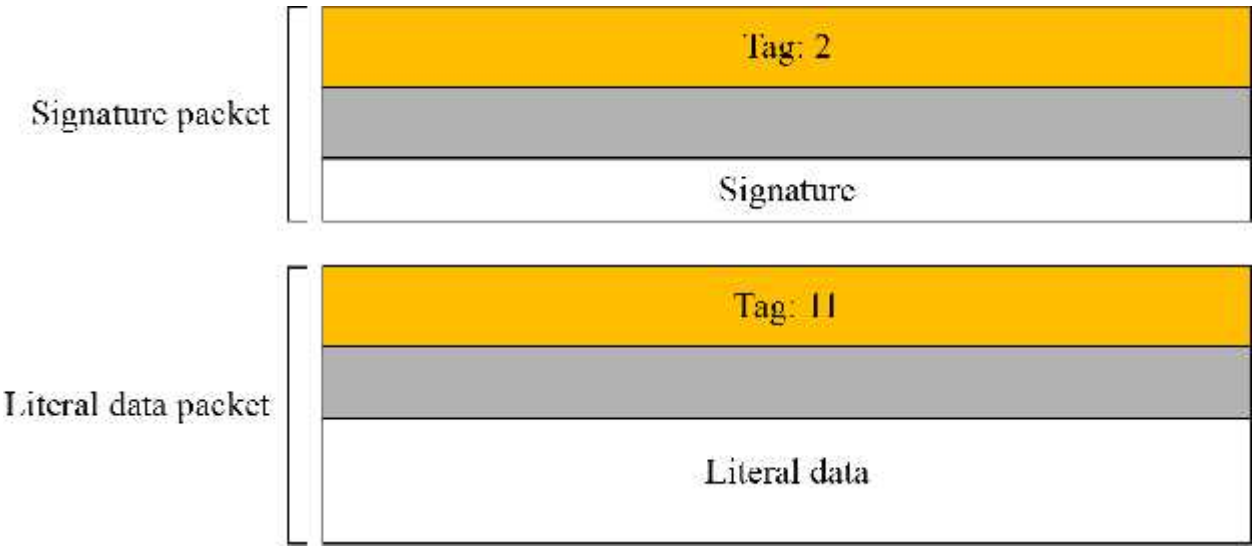
User Id: This variable length defines the user ID of the sender. It is normally the name of the user followed by an email address.



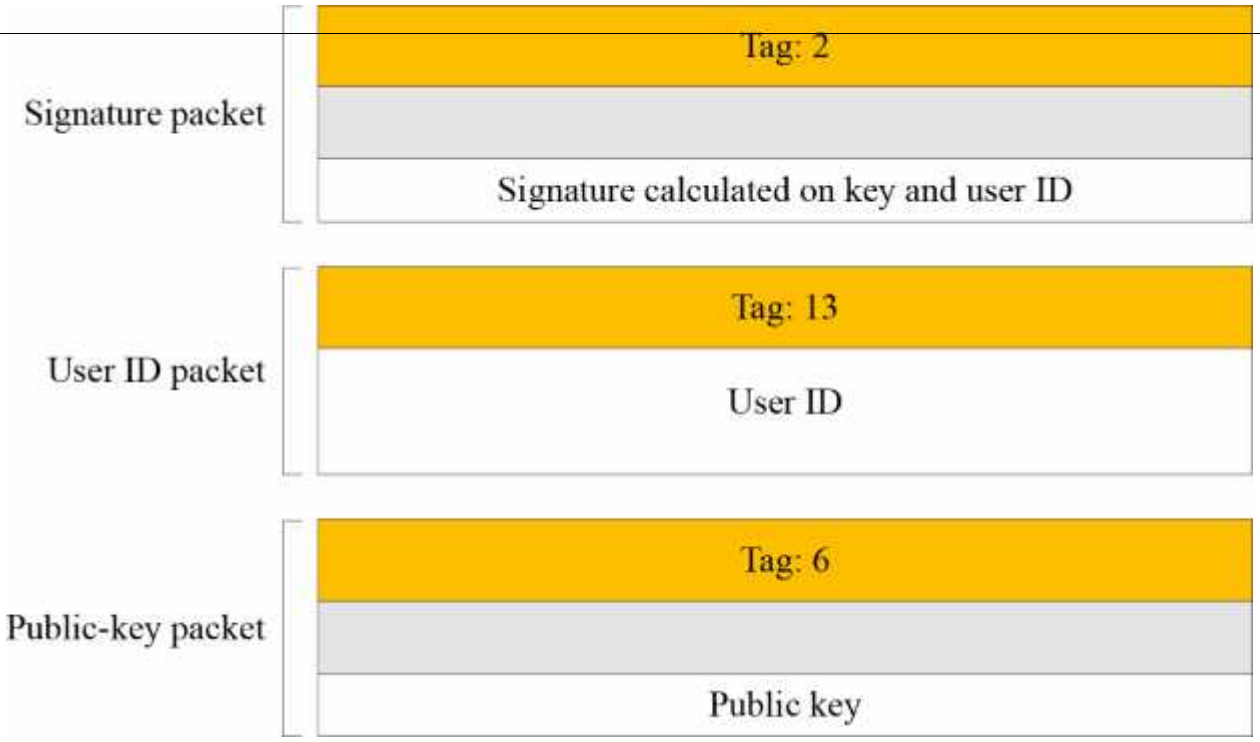
PGP Messages:

Encrypted message:

Signed message:



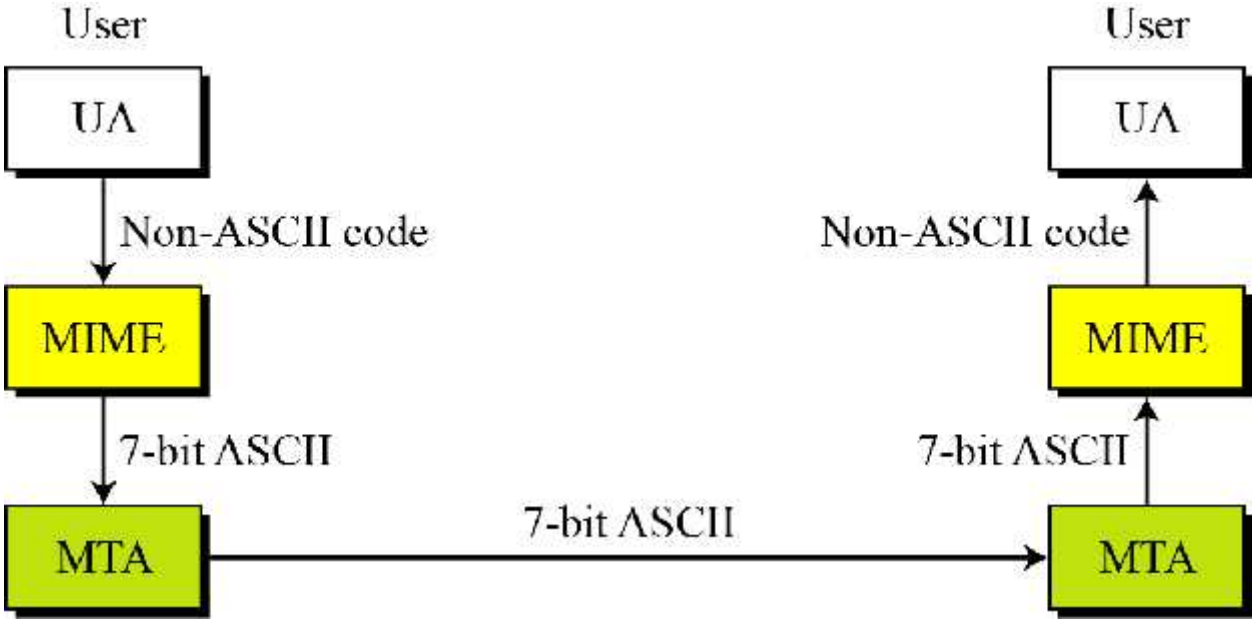
Certificate message:



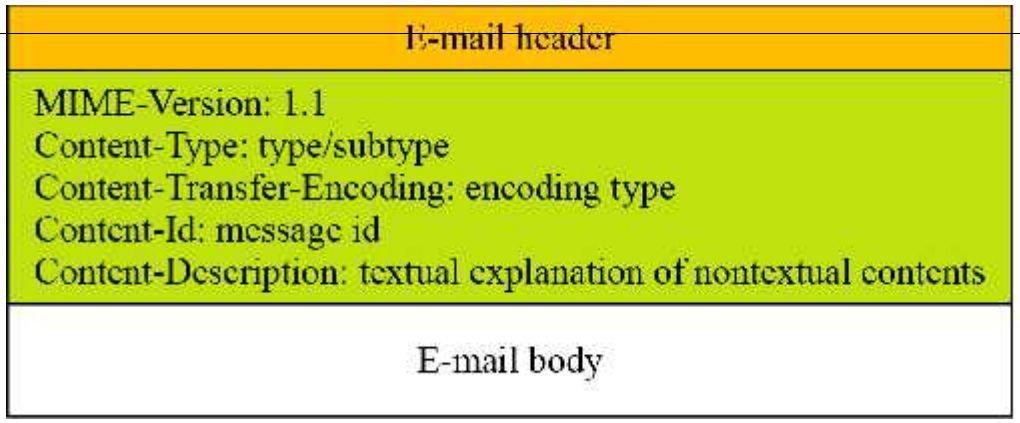
Applications of PGP: PGP has been extensively used for personal e-mails.It will probably continue to be.

- **S/MIME:**

  - Another security service designed for electronic mail is Secure/Multipurpose Internet Mail Extension (S/MIME).
  - The protocol is an enhancement of the Multipurpose Internet Mail Extension (MIME) protocol.



MIME defines five headers that can be added to the original e-mail header section to define the transformation parameters:



MIME-Version:

- This header defines the version of MIME used. The current version is 1.1.

```
MIME-Version: 1.1
```

Content-Type:

- The content type and the content subtype are separated by a slash. Depending on the subtype, the header may contain other parameters.

```
Content-Type: <type / subtype; parameters>
```

MIME allows seven different types of data. These are listed below

Table 16.14 *Data types and subtypes in MIME*

Type	*Subtype*	*Description*
	Plain	Unformatted.
	HTML	HTML format.
Multipart	Mixed	Body contains ordered parts of different data types.
	Parallel	Same as above, but no order.
	Digest	Similar to Mixed, but the default is message/RFC822.
	Alternative	Parts are different versions of the same message.
Message	RFC822	Body is an encapsulated message.
	Partial	Body is a fragment of a bigger message.
	External-Body	Body is a reference to another message.
Image	JPEG	Image is in JPEG format.
	GIF	Image is in GIF format.
Video	MPEG	Video is in MPEG format.
Audio	Basic	Single channel encoding of voice at 8 KHz.
Application	PostScript	Adobe PostScript.
	Octet-stream	General binary data (eight-bit bytes).

Text: The original message is in 7-bit ASCII format and no transformation by MIME is needed.

MultiPart: The body contains multiple, independent parts.

Messages: In message type, the body is itself an entire mail message, a part of a mail message or a pointer to a messagg. Three subtypes are currently used: RFC 822,partial, and external body.

Content - Transfer – Encoding:

Content-Transfer-Encoding: <Type>

Table 16.15 *Content-transfer-encoding*

Type	*Description*
7bit	NVT ASCII characters and short lines.
8bit	Non-ASCII characters and short lines.
Binary	Non-ASCII characters with unlimited-length lines.
Radix-64	6-bit blocks of data are encoded into 8-bit ASCII characters using Radix-64 conversion.
Quoted-printable	Non-ASCII characters are encoded as an equal sign followed by an ASCII code.

7-bit: This is 7-bit NVTASCII encoding.

8bit: This is 8-bit encoding**.**

Binary: This is 8-bit encoding. Non-ASCII characters can be sent, and the length of the line can exceed 1000 characters.

Radix -64 conversion

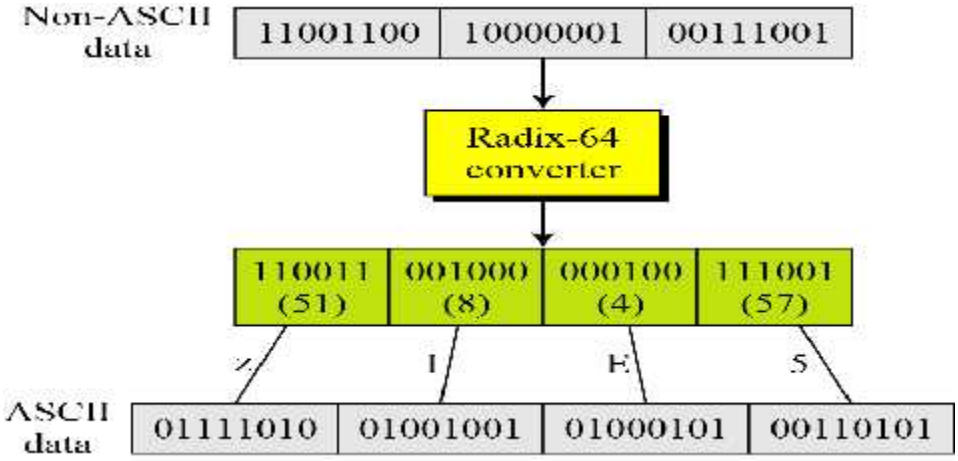
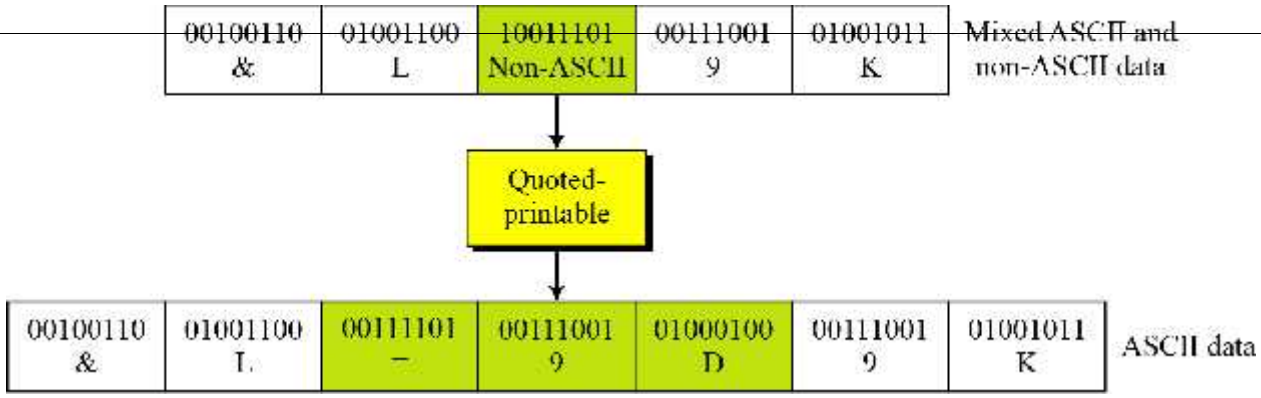


Table 16.16 *Radix-64 encoding table*

Value	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*	*Value*	*Code*
0	**A**	11	**L**	22	**W**	33	**h**	44	**s**	55	**3**
1	**B**	12	**M**	23	**X**	34	**i**	45	**t**	56	**4**
2	**C**	13	**N**	24	**Y**	35	**j**	46	**u**	57	**5**
3	**D**	14	**O**	25	**Z**	36	**k**	47	**v**	58	**6**
4	**E**	15	**P**	26	**a**	37	**l**	48	**w**	59	**7**
5	**F**	16	**Q**	27	**b**	38	**m**	49	**x**	60	**8**
6	**G**	17	**R**	28	**c**	39	**n**	50	**y**	61	**9**
7	**H**	18	**S**	29	**d**	40	**o**	51	**z**	62	**+**
8	**I**	19	**T**	30	**e**	41	**p**	52	**0**	63	**/**
9	**J**	20	**U**	31	**f**	42	**q**	53	**1**		
10	**K**	21	**V**	32	**g**	43	**r**	54	**2**		

Quoted-printable



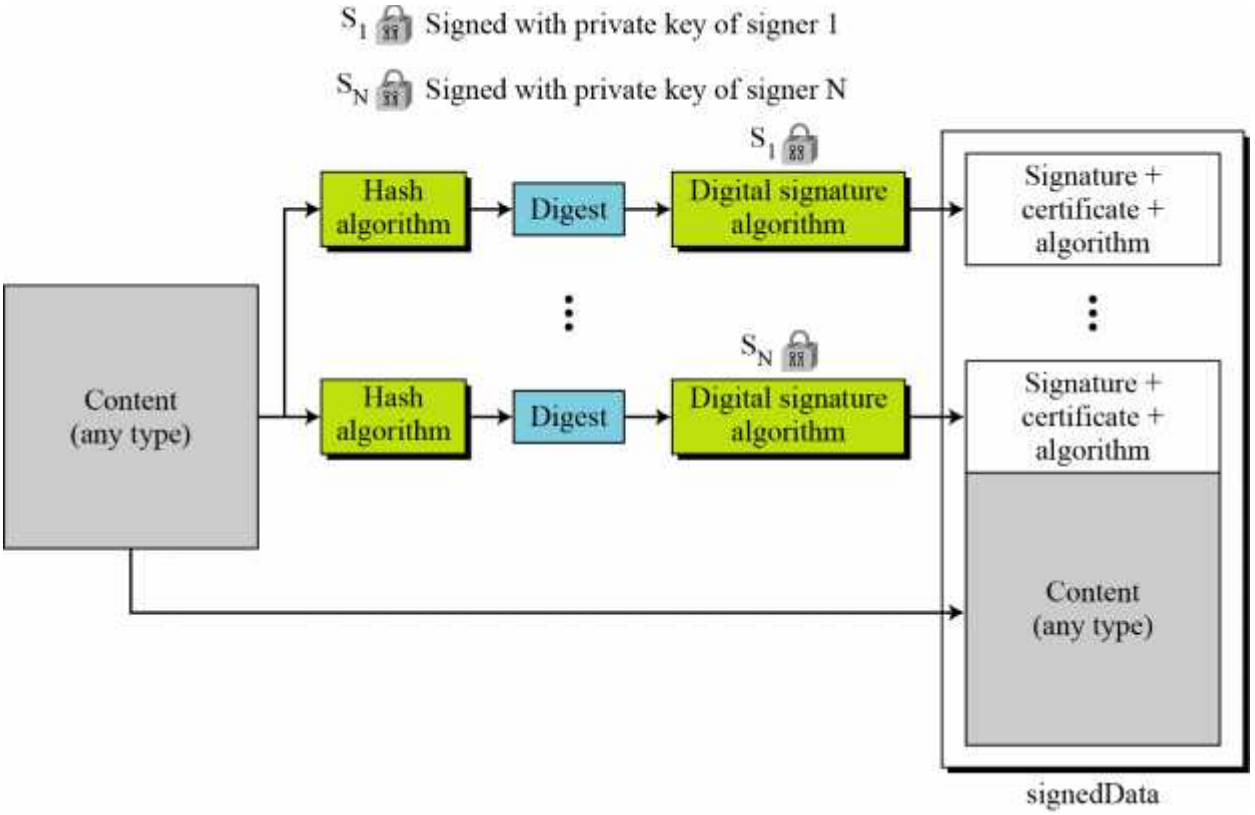
S/MIME:

- S/MIME adds some new content types to include security services to the MIME.
- All of these new types include the parameter "application/pkcs7-mime," in which "pkcs" defines "Public Key Cryptography Specification."

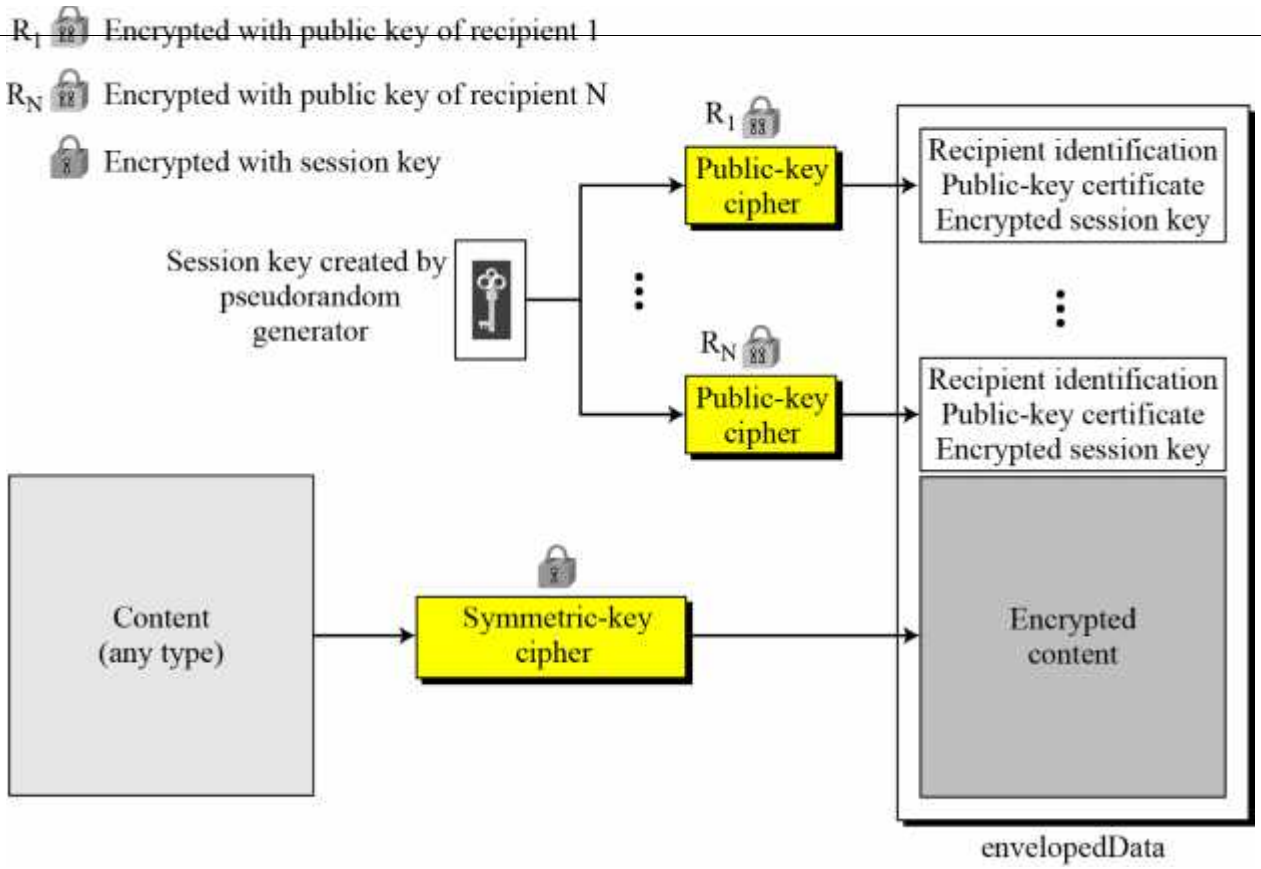
Cryptographic Message Syntax (CMS)

- To define how security services, such as confidentiality or integrity, can be added to MIME content types, S/MIME has defined Cryptographic Message Syntax (CMS).

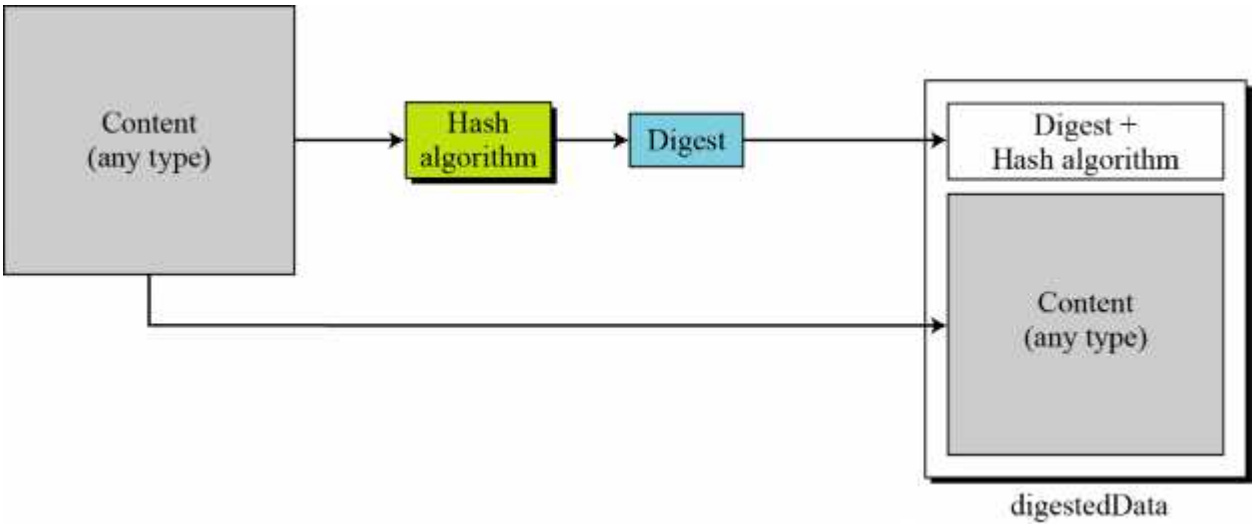
- The syntax in each case defines the exact encoding scheme for each content type. For details, the reader is referred to RFC 3369 and 3370.



Enveloped data content type:



Digest data content type:

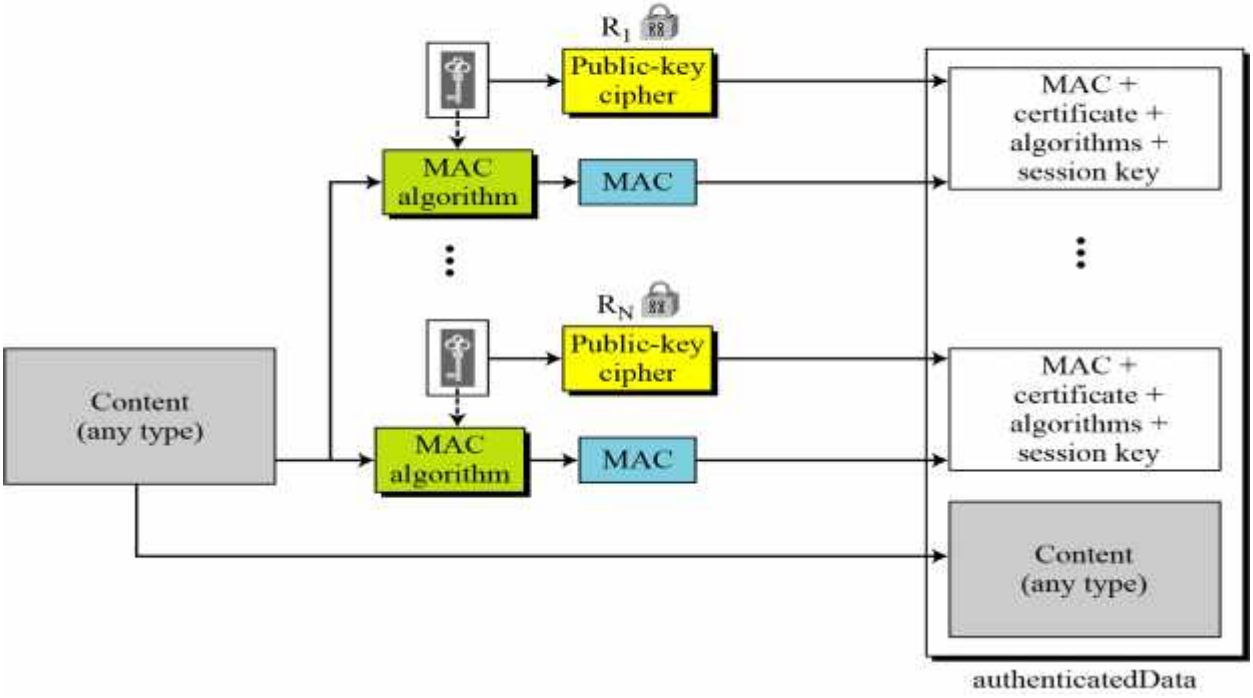


Encrypted data content type: This type is used to create an encrypted version of any content type.

Authenticated data content type

R_1 Encrypted with public key of recipient 1

R_N Encrypted with public key of recipient N



Cryptographic Algorithms:

- S/MIME defines several cryptographic algorithms. The term “must” means an absolute requirement; the term “should” means recommendation.

Table 16.17 *Cryptographic algorithm for S/MIME*

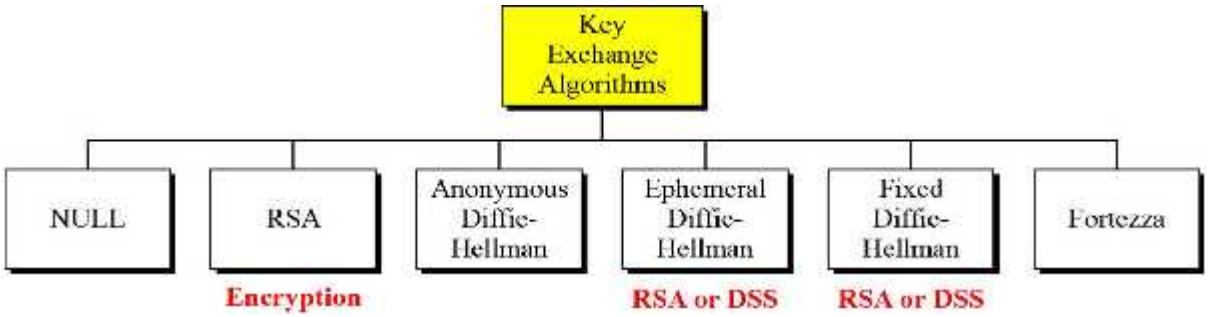
Algorithm	*Sender must support*	*Receiver must support*	*Sender should support*	*Receiver should support*
Content-encryption algorithm	Triple DES	Triple DES		1. AES 2. RC2/40
Session key encryption algorithm	RSA	RSA	Diffie Hellman	Diffie Hellman
Hash algorithm	SHA-1	SHA-1		MD5
Digest-encryption algorithm	DSS	DSS	RSA	RSA
Message authentication algorithm		HMAC with SHA-1		

The following shows an example of an enveloped-data in which a small message is encrypted using triple DES.

Content-Type: application/pkcs7-mime; mime-type=enveloped-data
Content-Transfer-Encoding: Radix-64
Content-Description: attachment
name="report.txt";
cb32ut67f4bhijHU21oi87eryb0287hmnklsgFDoY8bc659GhIGfH6543mhjkdsaH23YjBnmN
ybmlkzjhgfdyhGe23Kjk34XiuD678Es16se09jy76jHuytTMDcbnmlkjgfFdiuyu678543m0n3h
G34un12P2454Hoi87e2ryb0H2MjN6KuyrlsgFDoY897fk923jljk1301XiuD6gh78EsUyT23y

- **SSL(Secure socket layer):**

- SSL is designed to provide security and compression services to data generated from the application layer.



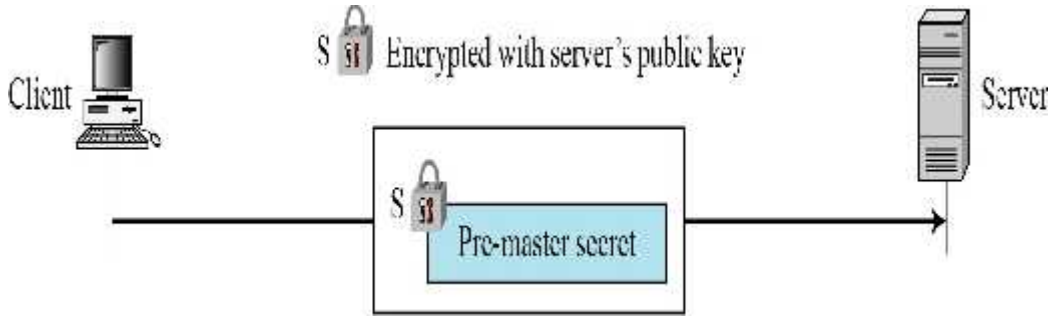
NULL:

- There is no key exchange in this method. No pre- master secret is established between the client and the server.
- Both client and server need to know the value of the pre-master secret.

RSA:

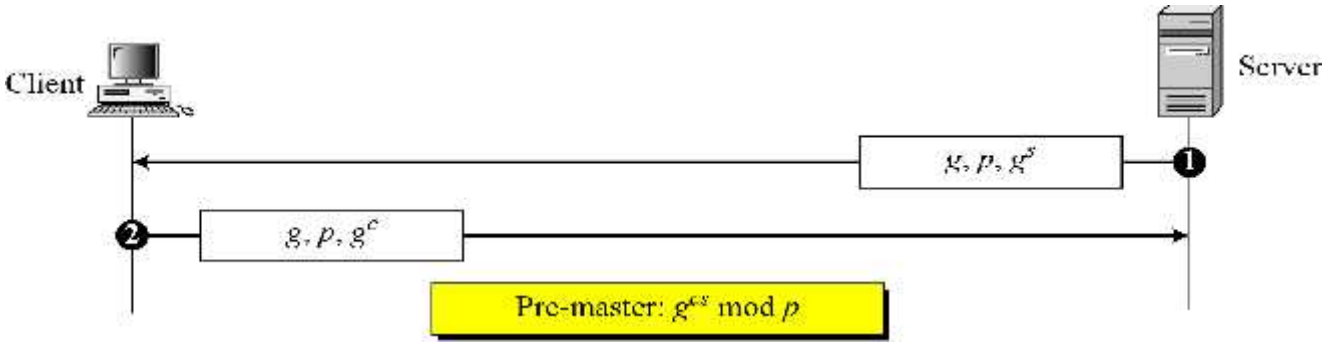
- In this method, the pre-master secret is a 48-byte random number created by the client, encrypted with the server's RSA public key, and sent to the server.
- The server needs to send its RSA encryption/decryption certificate.

RSA key exchange; server public key



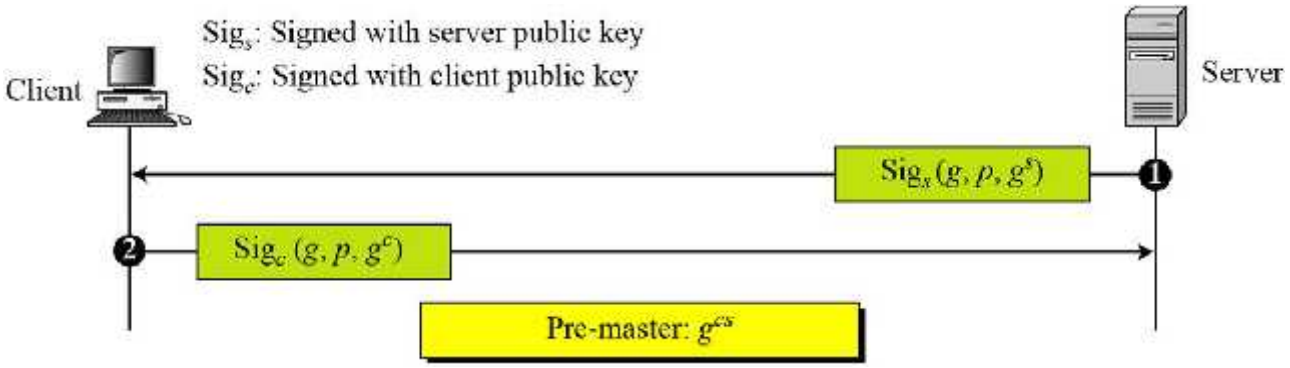
Anonymous Diffie-Hellman key exchange :

- This is the simplest and most insecure method.
- The pre-master secret is established between the client and server using the Diffie-Hellman protocol.
- The Diffie-Hellman half keys are sent in plaintext.
- It is called anonymous Diffie-Hellman because neither party is known to the other.



Ephemeral Diffie-Hellman key exchange:

- To the man-in-the-middle attack, the ephemeral Diffie-Hellman key exchange can be used.
- Each party sends a Diffie-Hellman key signed by its private key.
- The receiving party needs to verify the signature using the public key of the sender.



Fixed Diffie-Hellman:

- Another solution is the fixed **Diffie-Hellman** method.
- All entities in a group can prepare fixed **Diffie-Hellman** parameters(g and p).
- Then each entity can create a fixed **Diffie-Hellman** half-key(g^x).

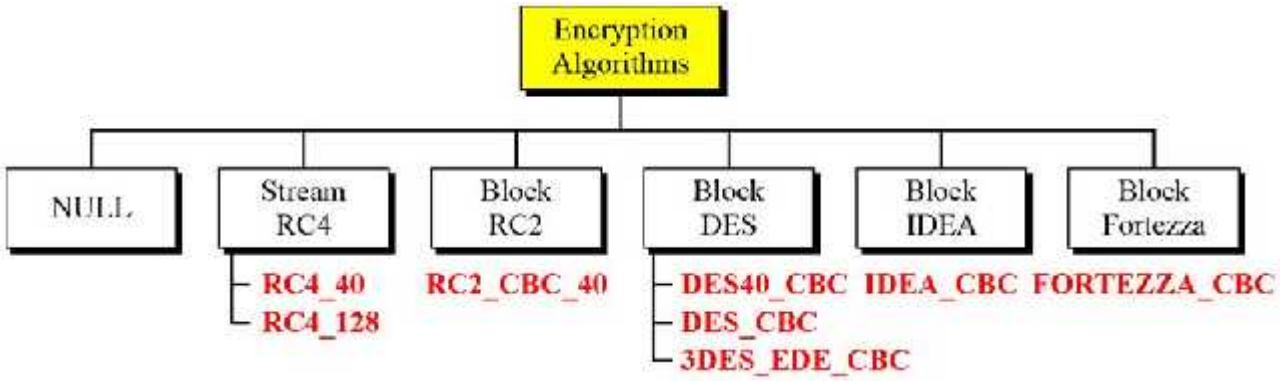
Fortezza:

- It is a registered trademark of the U.S National Security Agency(NSA).

- It is a family of security protocols developed for the Defense Department.

Encryption/Decryption algorithms:

- There are several choices for the encryption/decryption algorithm.



NULL: The NULL category simply defines the lack of an encryption/decryption algorithm.

Stream RC: Two RC algorithms are defined in stream mode.

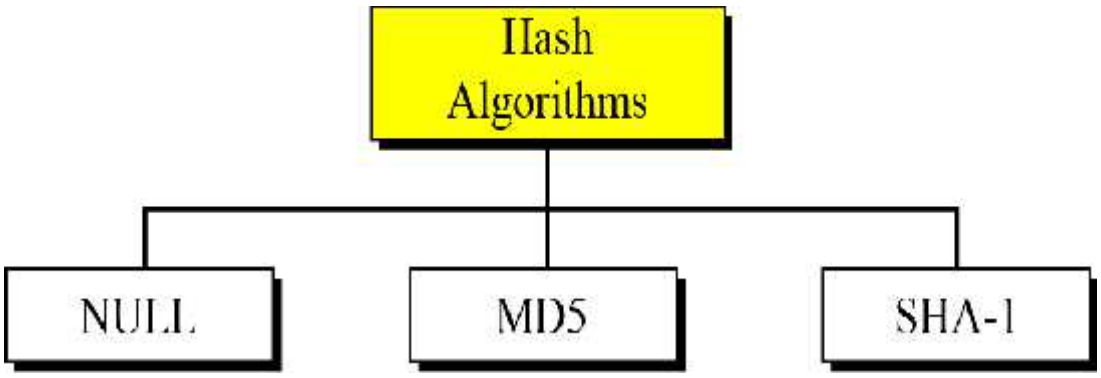
Block RC: One RC algorithm is defined in block mode.

DES: All DES algorithms are defined in block mode.DES 40_CBC uses a 40-bit key. Standard DES is defined as DES_CBC.3DES_EDE_CBC uses a 168-bit key.

IDEA: The one IDEA algorithm defined in block mode is IDEA_CBC , with a 128-bit key.

Fortezza: The one Fortezza algorithm defined in block mode is FORTEZZA_CBC , with a 96-bit key.

Hash Algorithms: SSL uses hash algorithms to provide message integrity.Three hash functions are defined.



NULL: The two parties may decline to use an algorithm. In this case, there is no hash function and the message is not authenticated.

MD5: The two parties may choose MD5 as the hash algorithm. In this case, a 128-bit key MD5 hash algorithm is used.

SHA-1: The two parties may choose SHA as the hash algorithm. In this case, a 160-bit SHA-1 hash algorithm is used.

Cipher Suite:

The combination of key exchange, hash, and encryption algorithms defines a cipher suite for each SSL session.

- Each suite starts with the term "SSL" followed by the key exchange algorithm.
- The word "WITH" separates with the key exchange algorithm from the encryption and hash algorithm. For example,

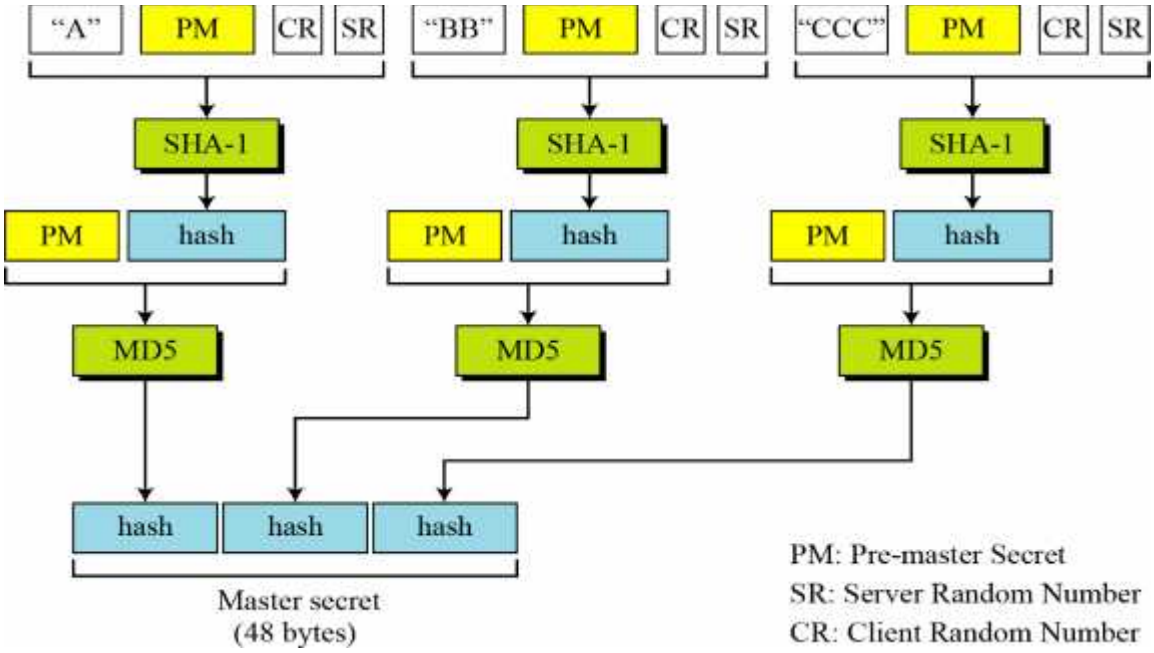
SSL_DHE_RSA_WITH_DES_CBC_SHA

Cipher suite	*Key Exchange*	*Encryption*	*Hash*
SSL_NULL_WITH_NULL_NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA-1
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4	SHA-1
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-1
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-1
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES	SHA-1
SSL_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4	MD5
SSL_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES	SHA-1
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES	SHA-1
SSL_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA-1
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES	SHA-1
SSL_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES	SHA-1
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES	SHA-1
SSL_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES	SHA-1
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES	SHA-1
SSL_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES	SHA-1
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES	SHA-1
SSL_FORTEZZA_DMS_WITH_NULL_SHA	Fortezza	NULL	SHA-1
SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA	Fortezza	Fortezza	SHA-1
SSL_FORTEZZA_DMS_WITH_RC4_128_SHA	Fortezza	RC4	SHA-1

Compression Algorithms:

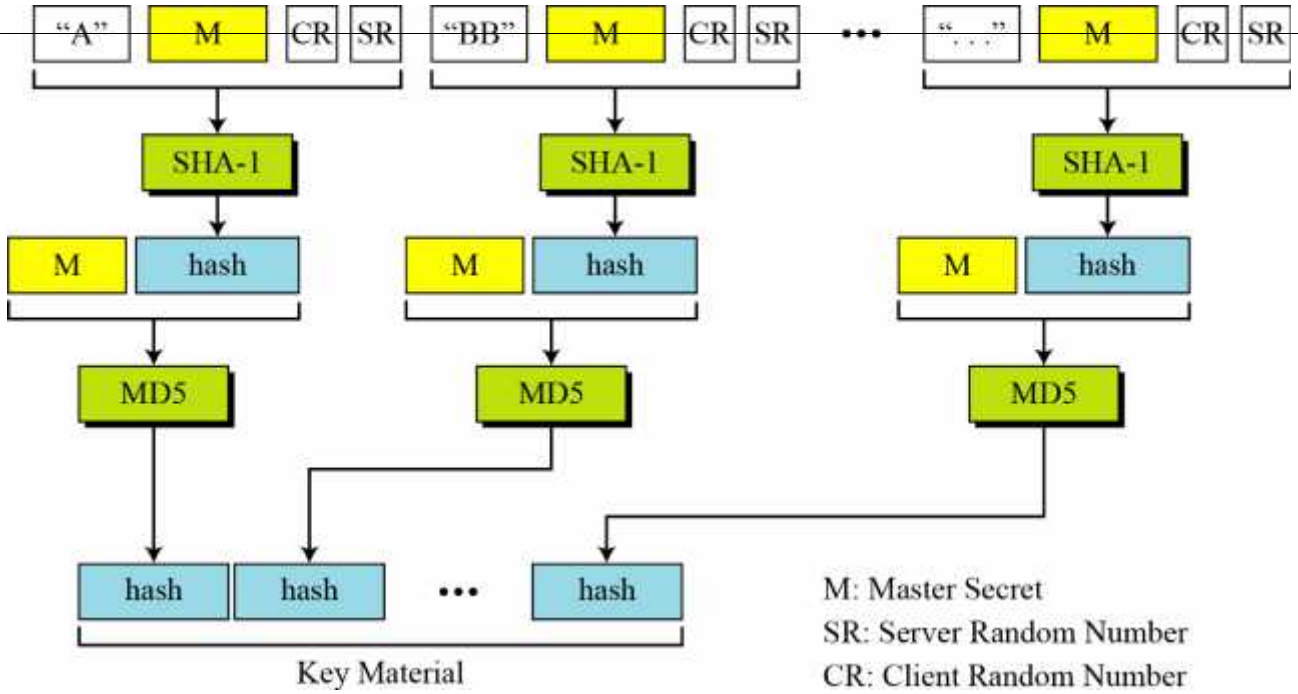
- Compression is optional in SSLv3
- No specific algorithm is specified for SSLv3.

Cryptographic parameter generation:

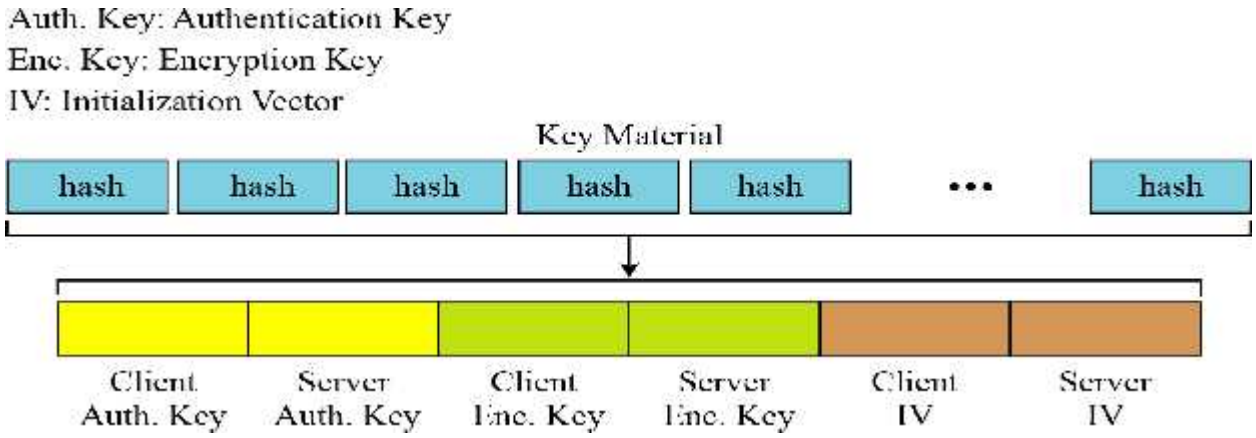


The parameters are generated using the following properties

1. The client and server exchange two random numbers; one is created by the client and the other by the server.
2. The client and server exchange one pre-master secret using one of the key-exchange algorithms.
3. A 48-byte master secret is created from the pre-master secret by applying two hash functions as shown above diagram.
4. The master secret is used to create variable-length key material by applying the same set of hash function and prepending with different constants as shown in below figure



5. Six different keys are extracted from the key material , as shown

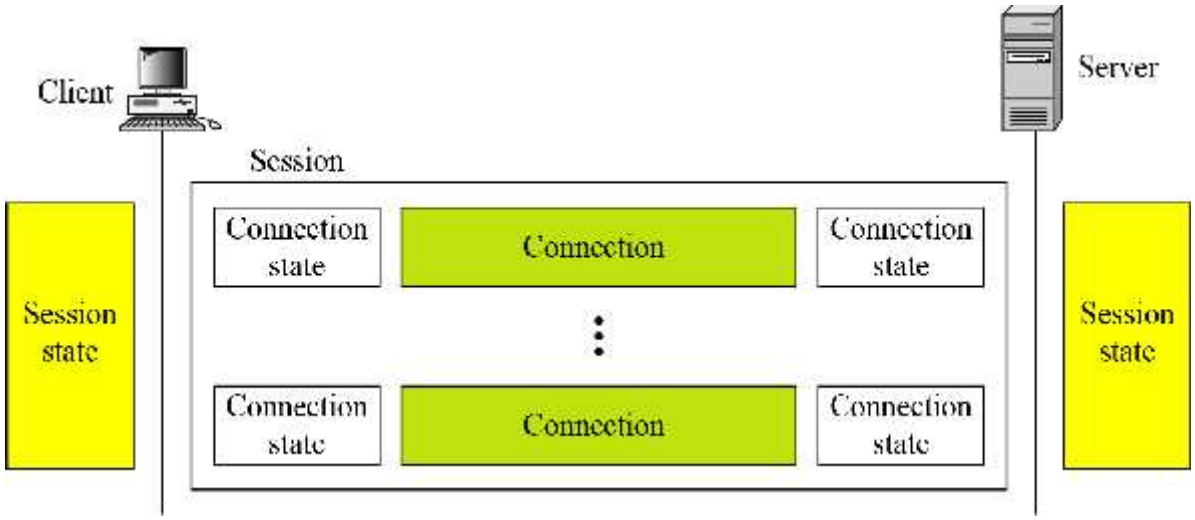


Sessions and connections:

- SSL differentiates a connection from a session.
- A session is an association between a client and server.

- After a session is established , the two parties have communication information such as the session identifier, the certificate authenticating each of them, the compression method , the cipher suite, and a master secret that is used to create keys for message authentication encryption.

A session and connections



- For two entities to exchange data, the establishment of a session is necessary, but not sufficient; they need to create a connection between themselves.
- A connection can consists of many connections. A connection between two parties can be terminated and re-established within the same session.
- When a connection is terminated , the two parties can also terminate the session, bit is not mandatory.
- To create a new session,the two parties need to go through a negotiation process.

Session state: A session is defined by the following parameters.

Parameter	*Description*
Session ID	A server-chosen 8-bit number defining a session.
Peer Certificate	A certificate of type X509.v3. This parameter may by empty (null).
Compression Method	The compression method.
Cipher Suite	The agreed-upon cipher suite.
Master Secret	The 48-byte secret.
Is resumable	A yes-no flag that allows new connections in an old session.

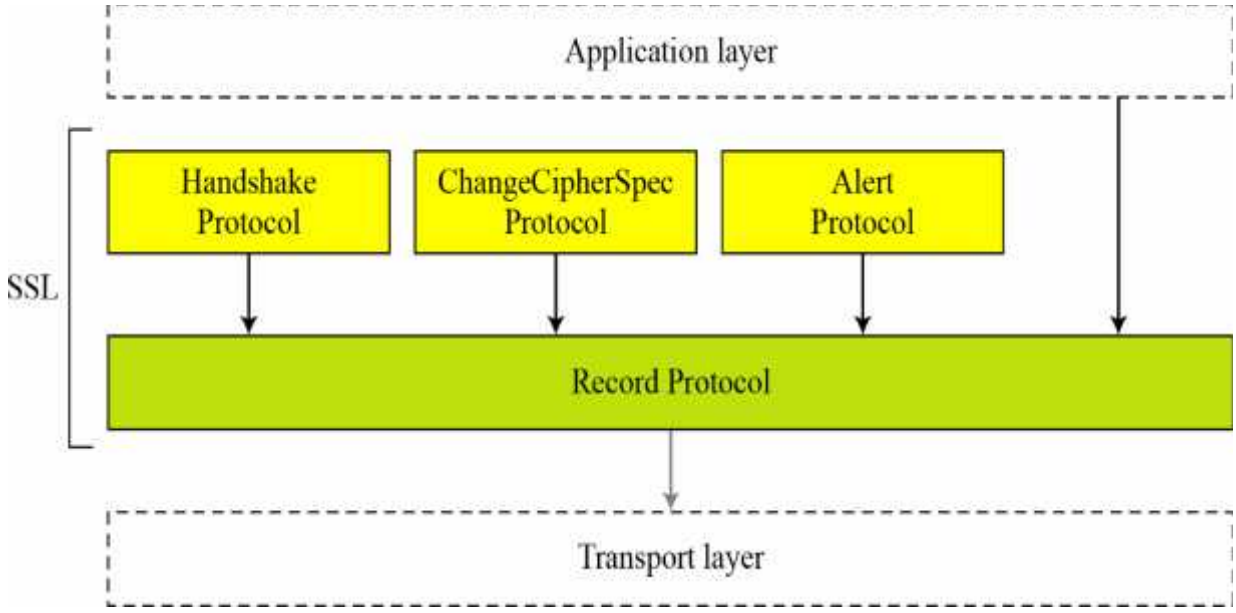
Connection state: A connection is defined by a connection state, a set of parameters established between two peers.

Parameter	*Description*
Server and client random numbers	A sequence of bytes chosen by the server and client for each connection.
Server write MAC secret	The outbound server MAC key for message integrity. The server uses it to sign; the client uses it to verify.
Client write MAC secret	The outbound client MAC key for message integrity. The client uses it to sign; the server uses it to verify.
Server write secret	The outbound server encryption key for message integrity.
Client write secret	The outbound client encryption key for message integrity.
Initialization vectors	The block ciphers in CBC mode use initialization vectors (IVs). One initialization vector is defined for each cipher key during the negotiation, which is used for the first block exchange. The final cipher text from a block is used as the IV for the next block.
Sequence numbers	Each party has a sequence number. The sequence number starts from 0 and increments. It must not exceed $2^{64} - 1$.

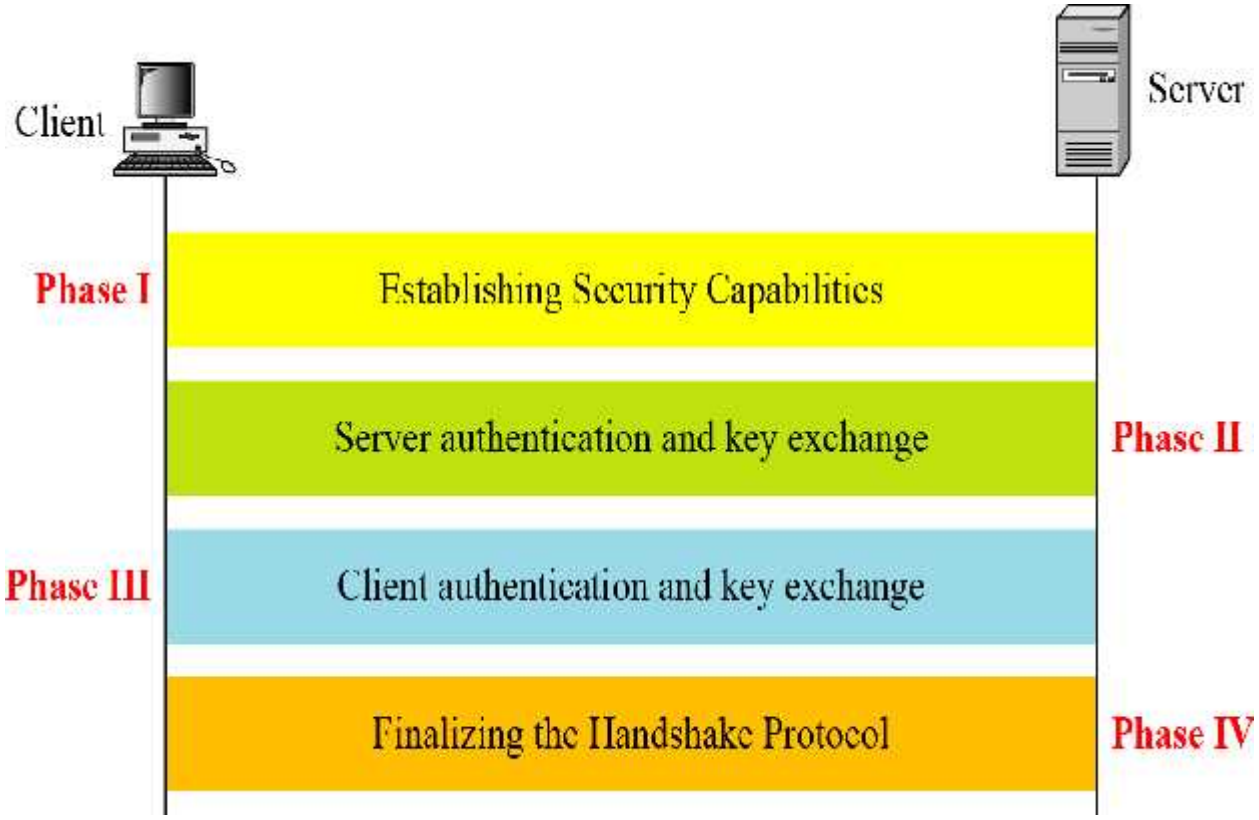
- The client and the server have six different cryptography secrets: three read secrets and three write secrets.
- The read secrets for the client are the same as the write secrets for the server and vice versa.

➢ FOUR protocols:

- We have discussed the idea of SSL without showing how SSL accomplishes its tasks. SSL defines four protocols in two layers, as shown in Figure



Handshake protocol:



Phase I: Establishing Security Capability

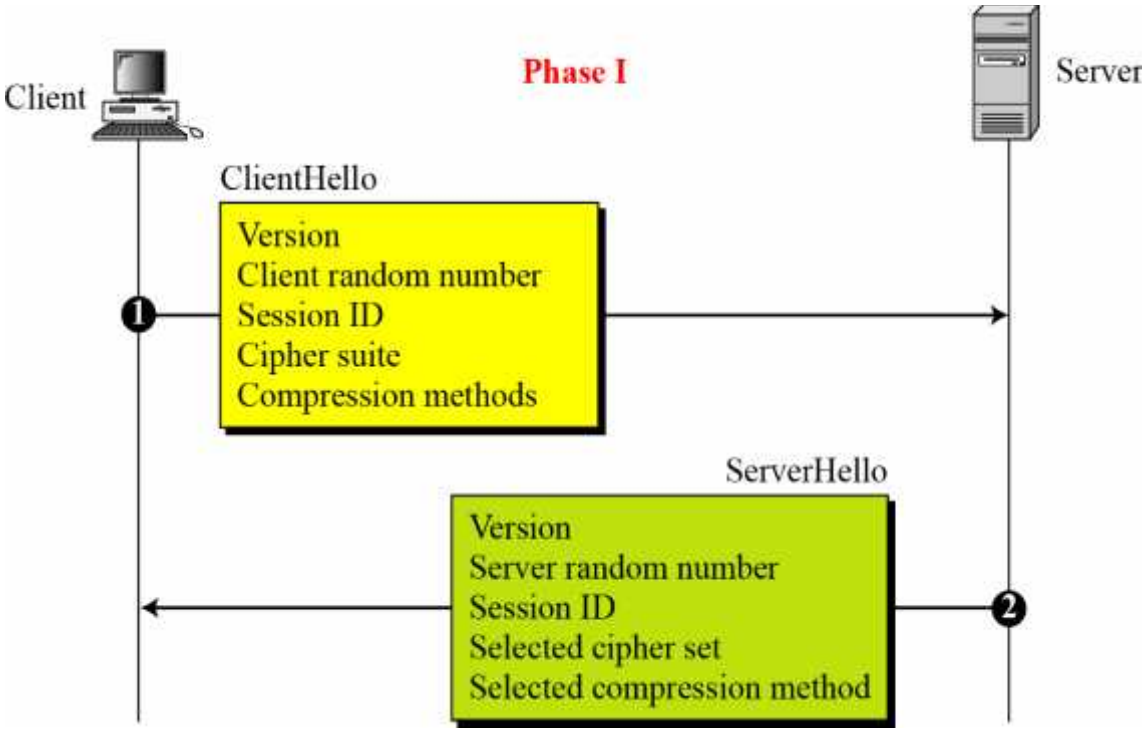
In Phase I, the client and the server announce their security capabilities and choose those that are convenient for both. In this phase, a session ID is established and the cipher suite is chosen. The parties agree upon a particular compression method. Finally, two random numbers are selected, one by the client and one by the server, to be used for creating a master secret as we saw before. Two messages are exchanged in this phase: ClientHello and ServerHello messages. Figure 17.14 gives additional details about Phase I.

ClientHello The client sends the ClientHello message. It contains the following:

a. The highest SSL version number the client can support.
b. A 32-byte random number (from the client) that will be used for master secret generation.
c. A session ID that defines the session.
d. A cipher suite that defines the list of algorithms that the client can support.
e. A list of compression methods that the client can support.

ServerHello The server responds to the client with a ServerHello message. It contains the following:

a. An SSL version number. This number is the lower of two version numbers: the highest supported by the client and the highest supported by the server.
b. A 32-byte random number (from the server) that will be used for master secret generation.



c. A session ID that defines the session.

d. The selected cipher set from the client side

e. The selected compression method from the client list.

After Phase I, the client and server know the following:

- The version of SSL
- The algorithms for key exchange, message authentication, and encryption

- The compression method
- The two random numbers for key generation

Phase II: Server key exchange and Authentication

- In phase II the server authenticates itself if needed. The sender may send its certificate, its public key, and also request certificates from the client.
- At the end, the server announces that the serverhello process is done.

Certificate: If it is required, the server sends a certificate message to authenticate itself.

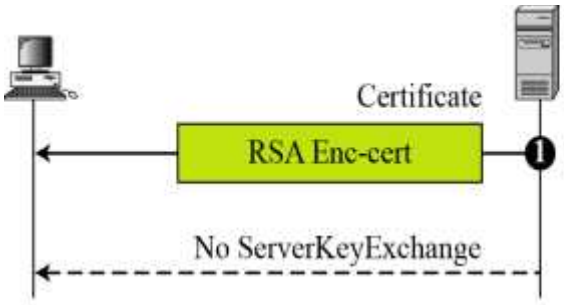
ServerKeyExchange: After the certificate message, the server sends a ServerKey-Exchange message that includes its contribution to the pre-master secret.

CertificateRequest: The server may require the client to authenticate itself.

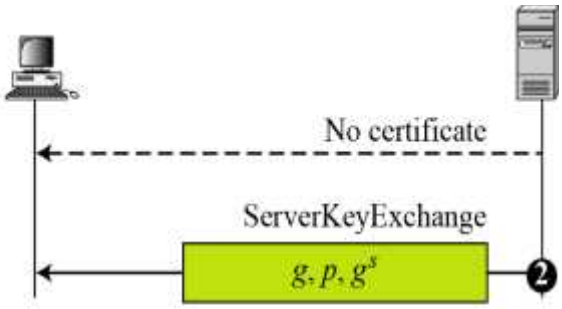
ServerHelloDone: The last message in Phase II is the ServerHelloDone message, which is a signal to the client that Phase II is over and that the client needs to start Phase III.

After Phase II,

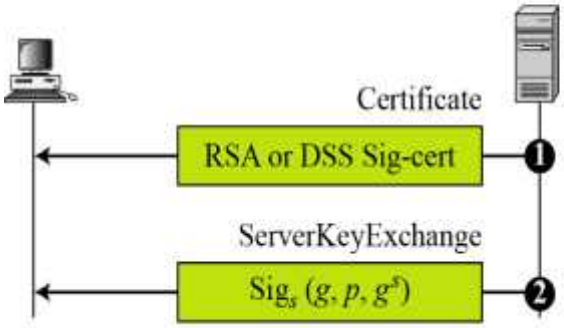
- The server is authenticated to the client.
- The client knows the public key of the server if required.



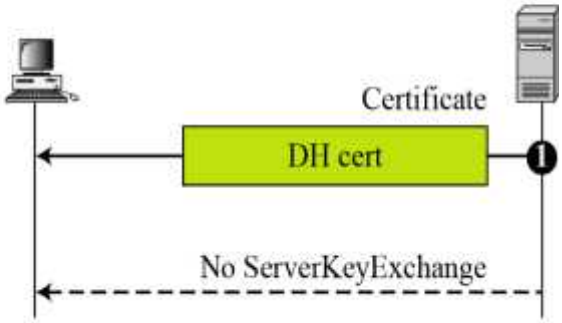
a. RSA



b. Anonymous DH



c. Ephemeral DH

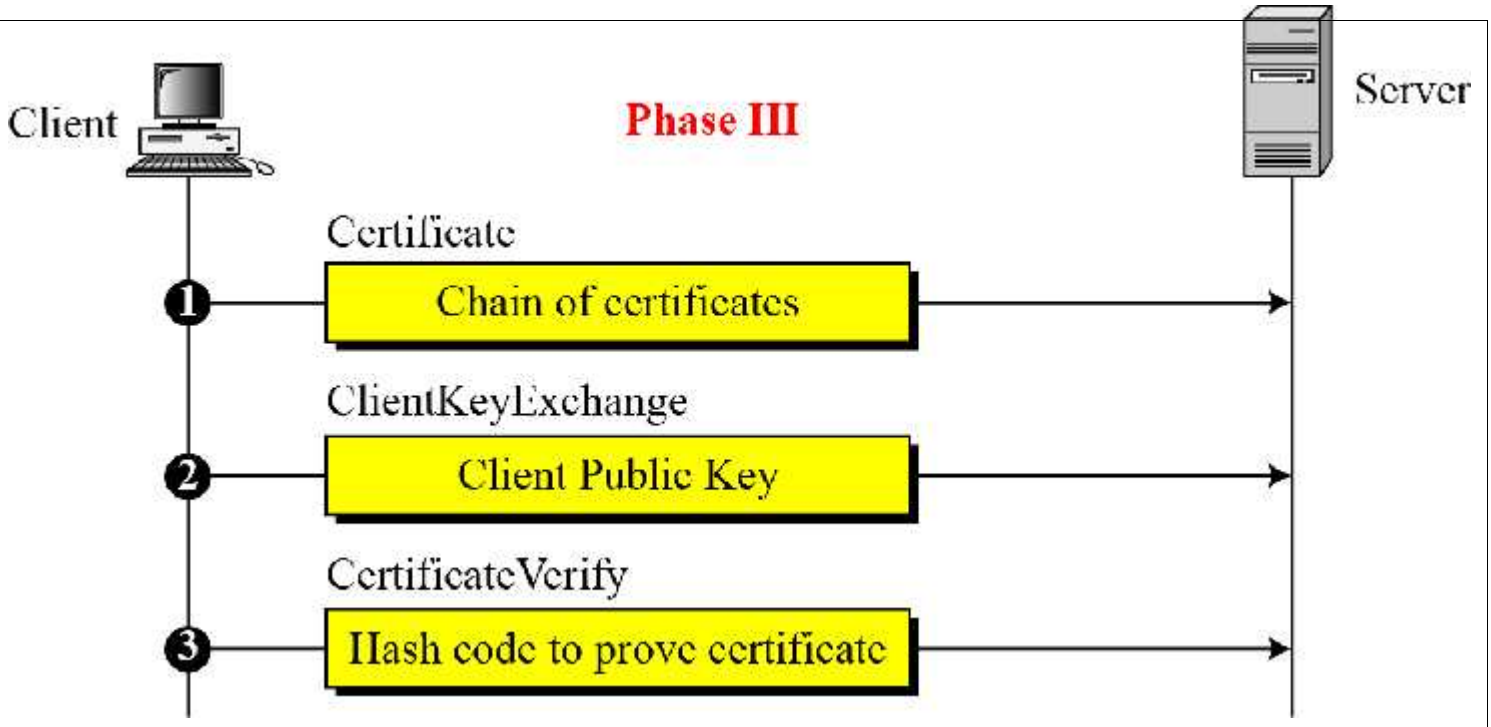


d. Fixed DH

- **RSA.** In this method, the server sends its RSA encryption/decryption public-key certificate in the first message. The second message, however, is empty because the pre-master secret is generated and sent by the client in the next phase. Note that the public-key certificate authenticates the server to the client. When the server receives the pre-master secret, it decrypts it with its private key. The possession of the private key by the server is proof that the server is the entity that it claims to be in the public-key certificate sent in the first message.

- **Anonymous DH.** In this method, there is no Certificate message. An anonymous entity does not have a certificate. In the ServerKeyExchange message, the server sends the Diffie-Hellman parameters and its half-key. Note that the server is not authenticated in this method.
- **Ephemeral DH.** In this method, the server sends either an RSA or a DSS digital signature certificate. The private key associated with the certificate allows the server to sign a message; the public key allows the recipient to verify the signature. In the second message, the server sends the Diffie-Hellman parameters and the half-key signed by its private key. Other text is also sent. The server is authenticated to the client in this method, not because it sends the certificate, but because it signs the parameters and keys with its private key. The possession of the private key is proof that the server is the entity that it claims to be in the certificate. If an impostor copies and sends the certificate to the client, pretending that it is the server claimed in the certificate, it cannot sign the second message because it does not have the private key.
- **Fixed DH.** In this method, the server sends an RSA or DSS digital signature certificate that includes its registered DH half-key. The second message is empty. The certificate is signed by the CA's private key and can be verified by the client using the CA's public key. In other words, the CA is authenticated to the client and the CA claims that the half-key belongs to the server.

Phase III: Client key exchange and authentication:



Certificate To certify itself to the server, the client sends a Certificate message. Note that the format is the same as the Certificate message sent by the server in Phase II, but the contents are different. It includes the chain of certificates that certify the client. This message is sent only if the server has requested a certificate in Phase II. If there is a request and the client has no certificate to send, it sends an Alert message (part of the Alert Protocol to be discussed later) with a warning that there is no certificate. The server may continue with the session or may decide to abort.

ClientKeyExchange After sending the Certificate message, the client sends a ClientKeyExchange message, which includes its contribution to the pre-master secret. The contents of this message are based on the key-exchange algorithm used. If the method is RSA, the client creates the entire pre-master secret and encrypts it with the RSA public key of the server. If the method is anonymous or ephemeral Diffie-Hellman, the client sends its Diffie-Hellman half-key. If the method is Fortezza, the client sends the Fortezza parameters. The contents of this message are empty if the method is fixed Diffie-Hellman.

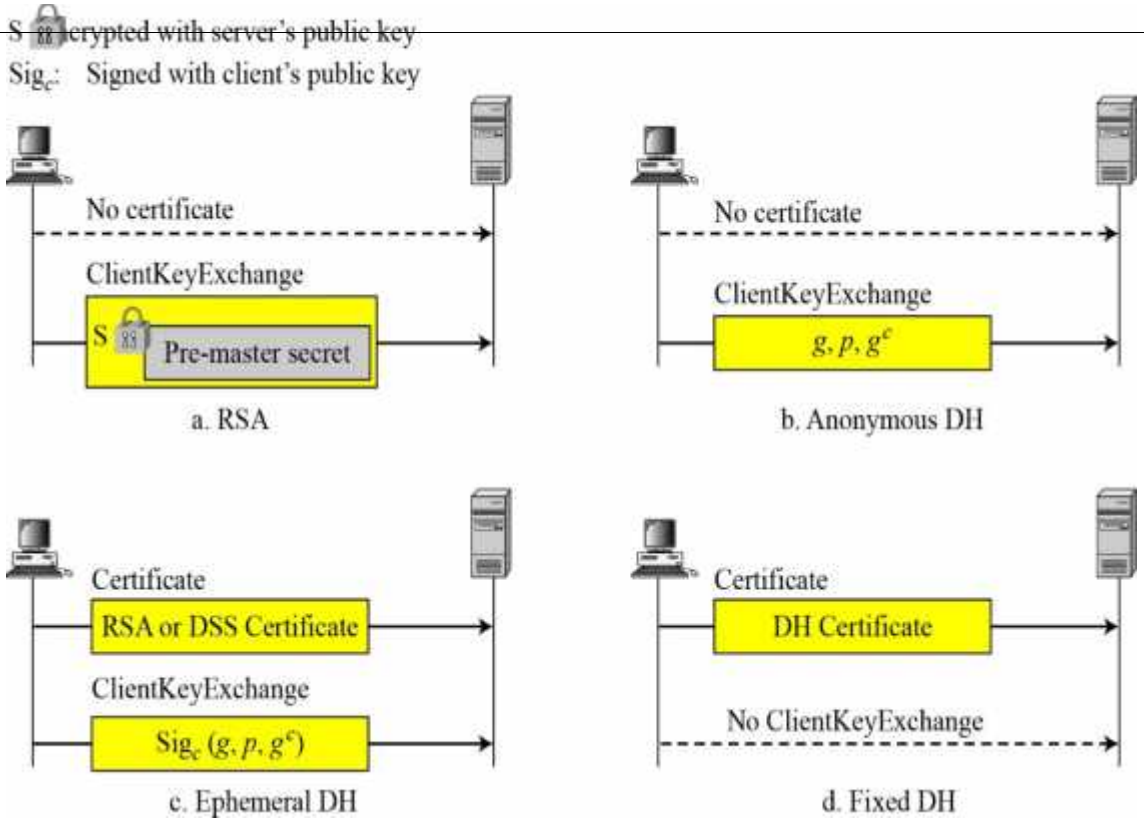
CertificateVerify:

- If the client has sent a certificate declaring that it owns the public key in the certificate,it needs to prove that it knows the corresponding private key.

After Phase III,

- The client is authenticated for the server.
- Both the client and the server know the pre-master secret.

S: Encrypted with server's public key
Sig_c: Signed with client's public key



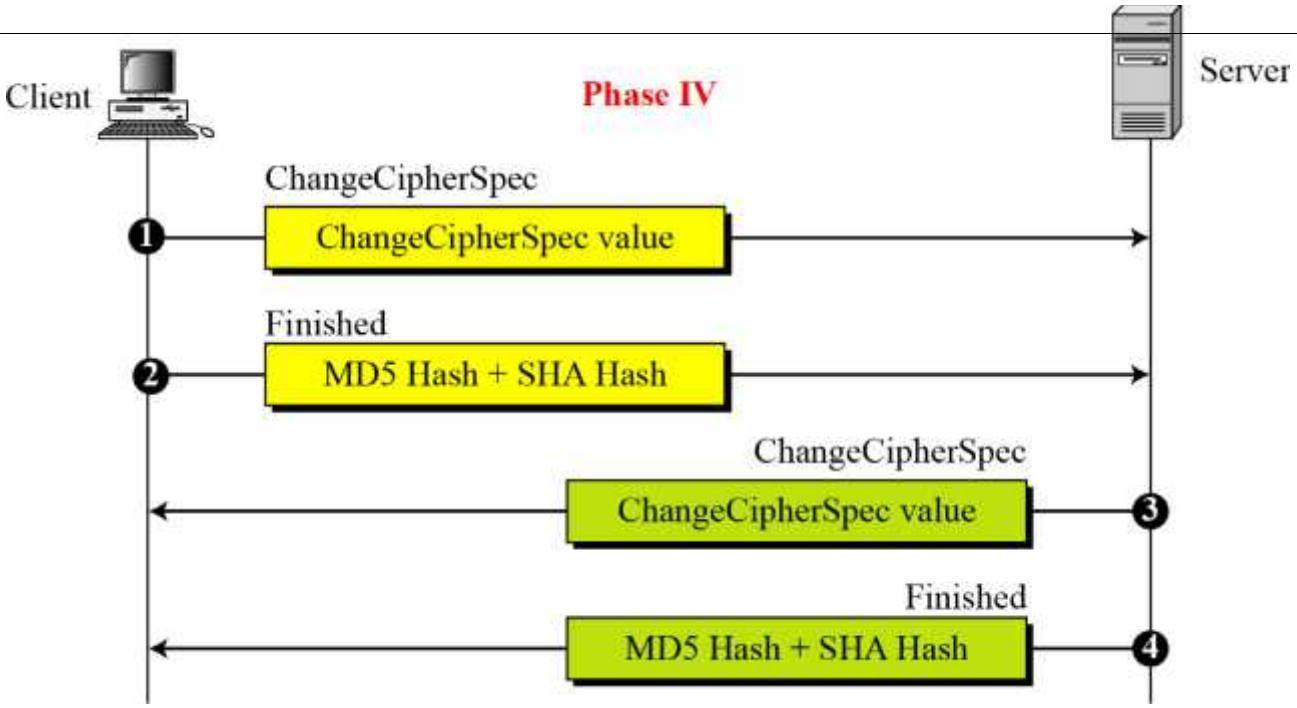
- RSA:
- Anonymous DH:
- Ephemeral DH
- Fixed DH

Phase IV Finalizing and Finishing:

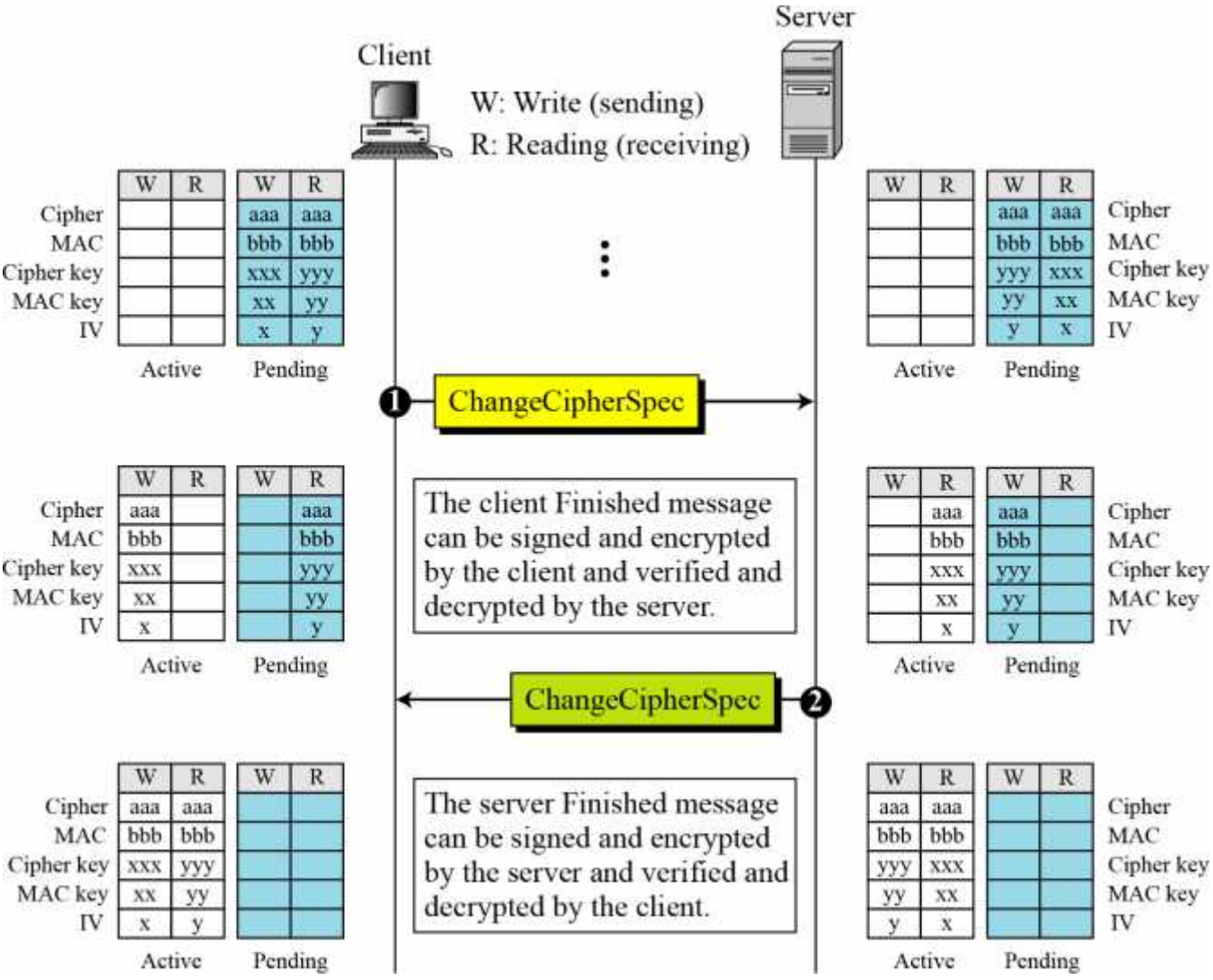
- In Phase IV, the client and server send messages to change cipher specification and to finish the handshaking protocol. Four messages are exchanged in this phase.

  ChangeCipherSpec:
  Finished:
  ChangeCipherSpec:

  After Phase IV, the client and server are ready to exchange data.



ChangeCipherSpec Protocol:



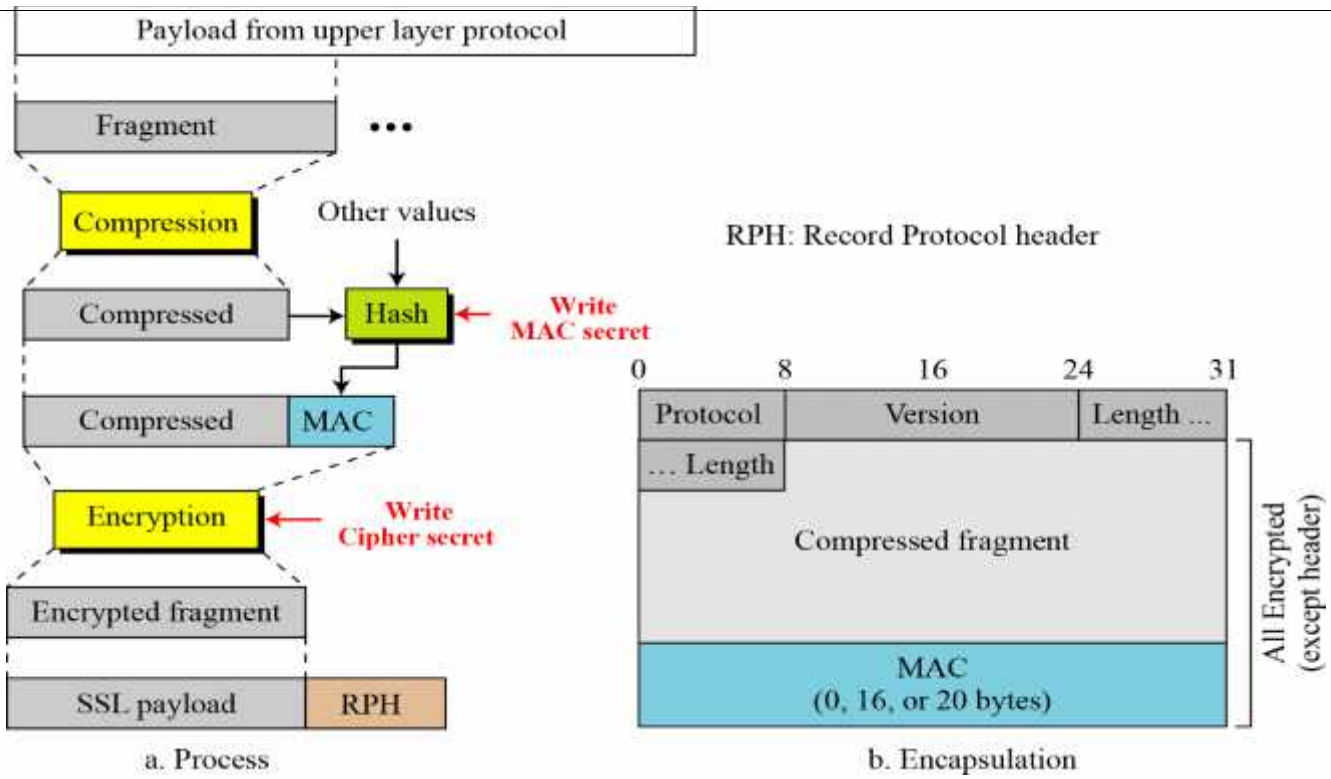
- SSL uses the **Alert Protocol** for reporting errors and abnormal conditions.It has onlyone message type, , the Alert message, that describes the problem and its level.

Value	*Description*	*Meaning*
0	*CloseNotify*	Sender will not send any more messages.
10	*UnexpectedMessage*	An inappropriate message received.
20	*BadRecordMAC*	An incorrect MAC received.
30	*DecompressionFailure*	Unable to decompress appropriately.
40	*HandshakeFailure*	Sender unable to finalize the handshake.
41	*NoCertificate*	Client has no certificate to send.
42	*BadCertificate*	Received certificate corrupted.
43	*UnsupportedCertificate*	Type of received certificate is not supported.
44	*CertificateRevoked*	Signer has revoked the certificate.
45	*CertificateExpired*	Certificate expired.
46	*CertificateUnknown*	Certificate unknown.
47	*IllegalParameter*	An out-of-range or inconsistent field.

Record Protocol:

- The Record Protocol carries messages from the upper layer.
- The message is fragmented and optionally compressed.
- **A MAC** is added to the compressed message using the negotiated hash algorithm.
- **Finally the SSL** header is added to the encrypted message.

Fragmentation/Combination **At the sender, a message from the application layer is fragmented into blocks of 2^{14} bytes, with the last block possibly less than this size. At the receiver, the fragments are combined together to make a replica of the original message.**



Compression/Decompression At the sender, all application layer fragments are compressed by the compression method negotiated during the handshaking. The compression method needs to be lossless (the decompressed fragment must be an exact replica of the original fragment). The size of the fragment must not exceed 1024 bytes. Some compression methods work only on a predefined block size and if the size of the block is less than this, some padding is added. Therefore, the size of the compressed fragment may be greater than the size of the original fragment. At the receiver, the compressed fragment is decompressed to create a replica of the original. If the size of the decompressed fragment exceeds 2^{14}, a fatal decompression Alert message is issued. Note that compression/decompression is optional in SSL.

Signing/Verifying At the sender, the authentication method defined during the handshake (NULL, MD5, or SHA-1) creates a signature (MAC), as shown in Fig. 17.22.

The hash algorithm is applied twice. First, a hash is created from the concatenations of the following values:

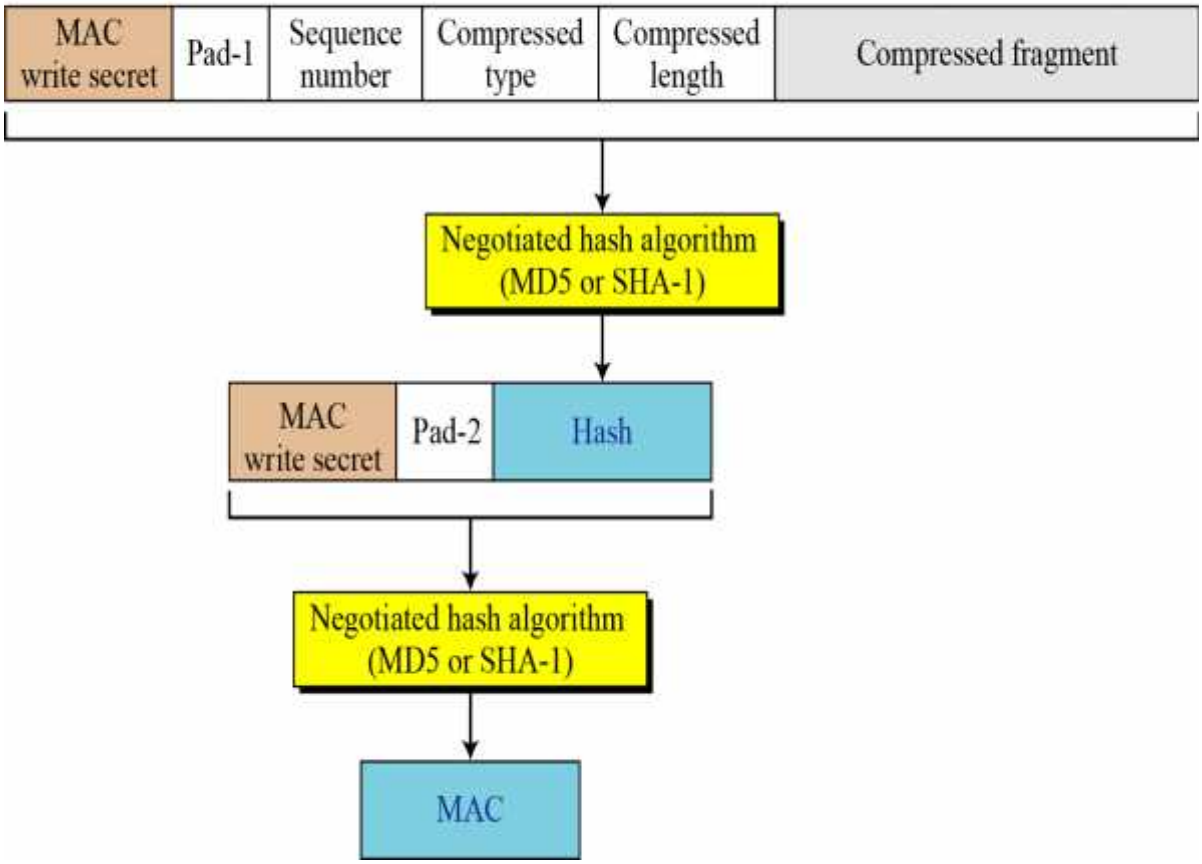
a. The MAC write secret (authentication key for the outbound message)
b. Pad-1, which is the byte 0x36 repeated 48 times for MD5 and 40 times for SHA-1
c. The sequence number for this message
d. The compressed type, which defines the upper-layer protocol that provided the compressed fragment
e. The compressed length, which is the length of the compressed fragment
f. The compressed fragment itself

Second, the final hash (MAC) is created from the concatenation of the following values:

a. The MAC write secret
b. Pad-2, which is the byte 0x5C repeated 48 times for MD5 and 40 times for SHA-1
c. The hash created from the first step

Pad-1: Byte 0x36 (00110110) repeated 48 times for MD5 and 40 times for SHA-1

Pad-2: Byte 0x5C (01011100) repeated 48 times for MD5 and 40 times for SHA-1



Encryption/Decryption At the sender, the compressed fragment and the hash are encrypted using the cipher write secret. At the receiver, the received message is decrypted using the cipher read secret. For block encryption, padding is added to make the size of the encryptable message a multiple of the block size.

Framing/Deframing After the encryption, the Record Protocol header is added at the sender. The header is removed at the receiver before decryption.

SSL message Protocol:

- As we have discussed, messages from three protocols and data from the application layer are encapsulated in the Record Protocol messages.

The fields in this header are listed below.

- **Protocol.** This 1-byte field defines the source or destination of the encapsulated message. It is used for multiplexing and demultiplexing. The values are 20 (ChangeCipherSpec Protocol), 21 (Alert Protocol), 22 (Handshake Protocol), and 23 (data from the application layer).
- **Version.** This 2-byte field defines the version of the SSL; one byte is the major version and the other is the minor. The current version of SSL is 3.0 (major 3 and minor 0).
- **Length.** This 2-byte field defines the size of the message (without the header) in bytes.

ChangeCipherSpecProtocol:

As we said before, the ChangeCipherSpec Protocol has one message, the Change-CipherSpec message. The message is only one byte, encapsulated in the Record Protocol message with protocol value 20, as shown in Fig. 17.24.

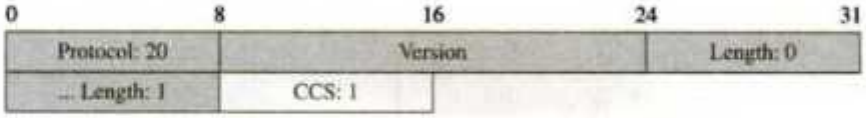


Fig. 17.24 *ChangeCipherSpec message*

The one-byte field in the message is called the CCS and its value is currently 1.

Alert Protocol:

The Alert Protocol, as we discussed before, has one message that reports errors in the process. Figure 17.25 shows the encapsulation of this single message in the Record Protocol with protocol value 21.

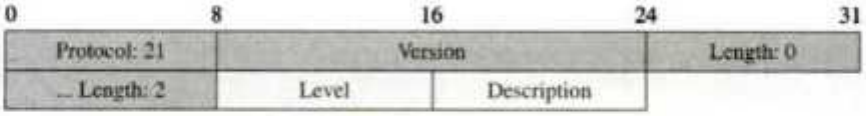


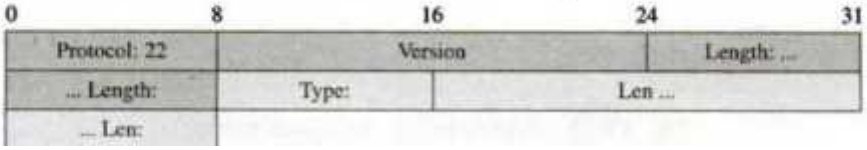
Fig. 17.25 *Alert message*

The two fields of the Alert message are listed below.

- **Level.** This one-byte field defines the level of the error. Two levels have been defined so far: warning and fatal.
- **Description.** The one-byte description defines the type of error.

HandShake Protocol:

Several messages have been defined for the Handshake Protocol. All of these messages have the four-byte generic header shown in Fig. 17.26. The figure shows the Record Protocol header and the generic header for the Handshake Protocol. Note that the value of the protocol field is 22.



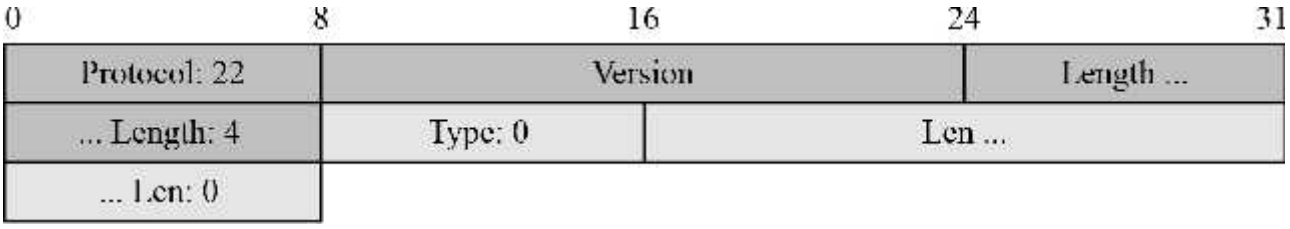
- **Type.** This one-byte field defines the type of message. So far ten types have been defined as listed in Table 17.5.

Table 17.5 *Types of Handshake messages*

Type	*Message*
0	HelloRequest
1	ClientHello
2	ServerHello
11	Certificate
12	ServerKeyExchange
13	CertificateRequest
14	ServerHelloDone
15	CertificateVerify
16	ClientKeyExchange
20	Finished

- **Length (Len).** This three-byte field defines the length of the message (excluding the length of the type and length field). The reader may wonder why we need two length fields, one in the general Record header and one in the generic header for the Handshake messages. The answer is that a Record message may carry two Handshake messages at the same time if there is no need for another message in between.

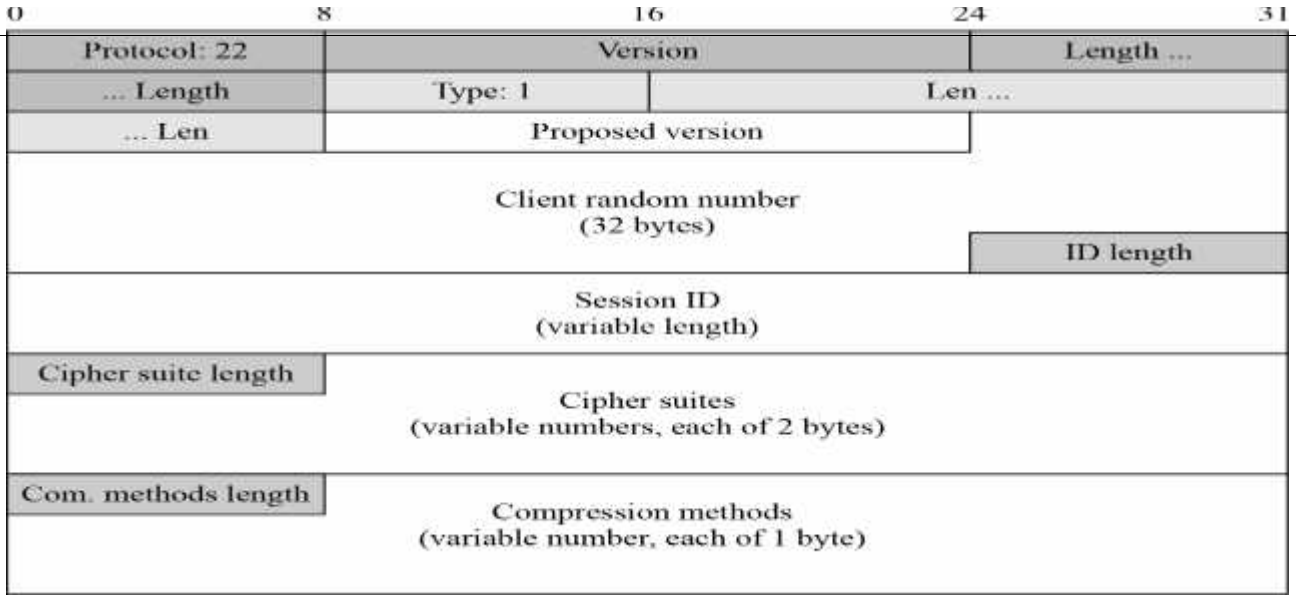
HelloRequest Message The HelloRequest message, which is rarely used, is a request from the server to the client to restart a session. This may be needed if the server feels that something is wrong with the session and a fresh session is needed. For example, if the session becomes so long that it threatens the security of the session, the server may send this message. The client then needs to send a ClientHello message and negotiate the security parameters. Figure 17.27 shows the format of this message. It is four bytes with a type value of 0. The message has no body, so the value of the length field is also 0.



ClientHello Message The ClientHello message is the first message exchanged during handshaking. Figure 17.28 shows the format of the message.

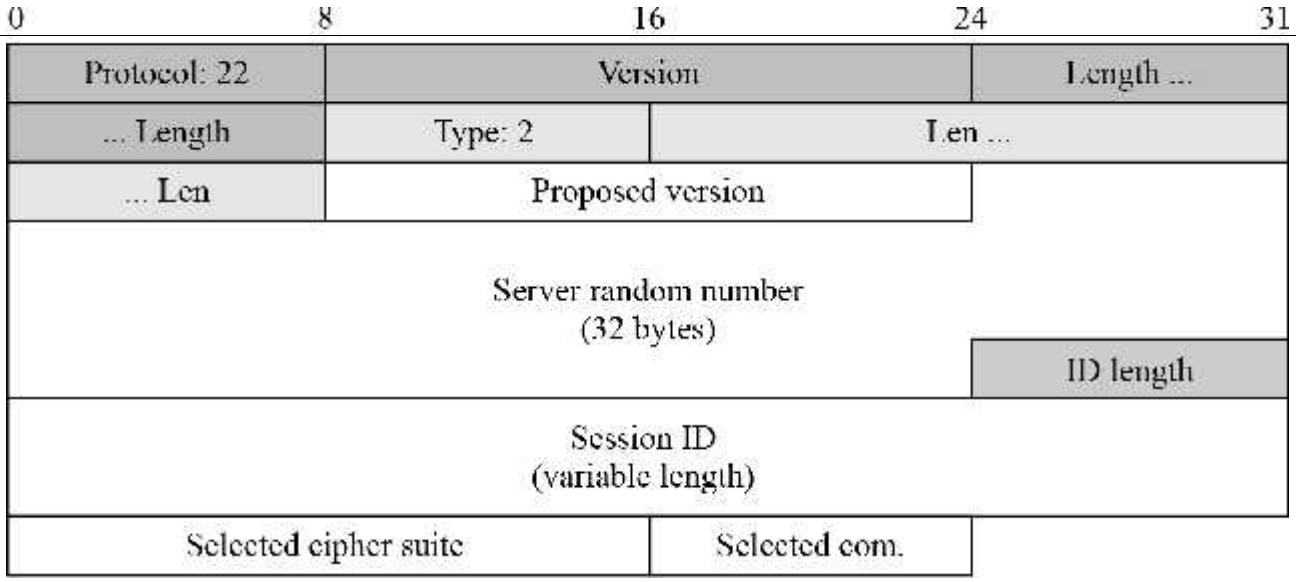
The type and length fields are as discussed previously. The following is a brief description of the other fields.

- **Version.** This 2-byte field shows the version of the SSL used. The version is 3.0 for SSL and 3.1 for TLS. Note that the version value, for example, 3.0, is stored in two bytes: 3 in the first byte and 0 in the second.

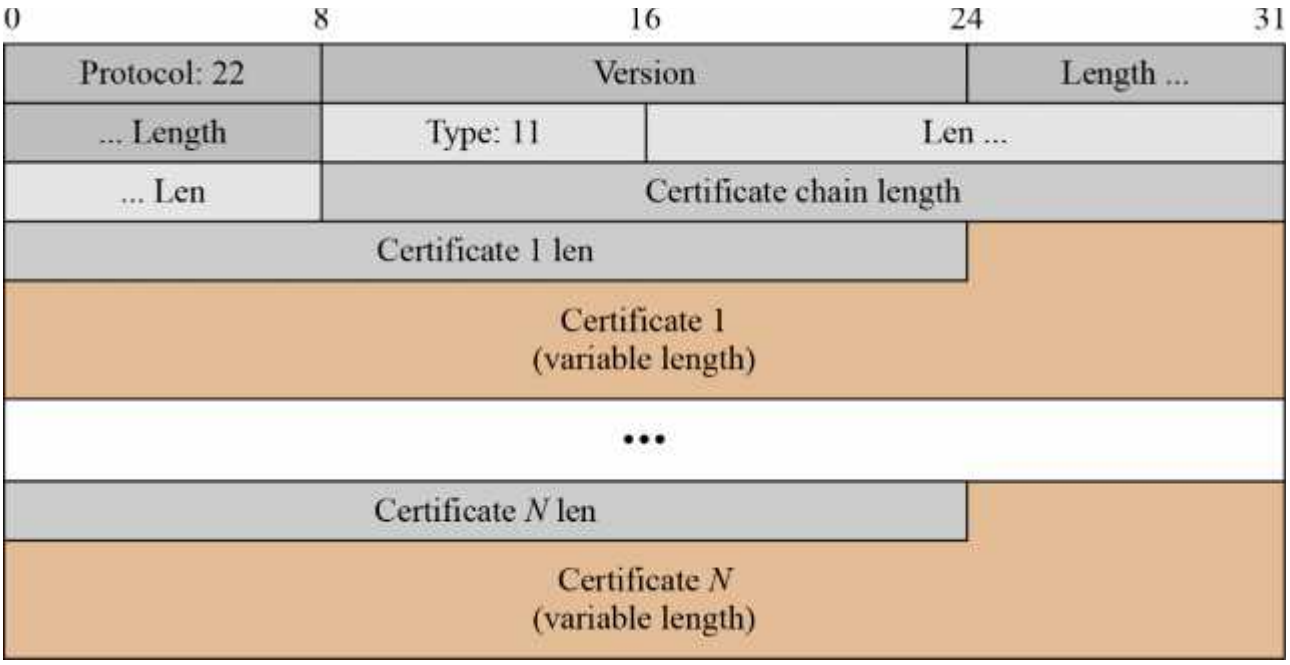


- **Client Random Number.** This 32-byte field is used by the client to send the client random number, which creates security parameters.
- **Session ID Length.** This 1-byte field defines the length of the session ID (next field). If there is no session ID, the value of this field is 0.
- **Session ID.** The value of this variable-length field is 0 when the client starts a new session. The session ID is initiated by the server. However, if a client wants to resume a previously stopped session, it can include the previously-defined session ID in this field. The protocol defines a maximum of 32 bytes for the session ID.
- **Cipher Suite Length.** This 2-byte field defines the length of the client-proposed cipher suite list (next field).
- **Cipher Suite List.** This variable-length field gives the list of cipher suites that the client supports. The field lists the cipher suites from the most preferred to the least preferred. Each cipher suite is encoded as a two-byte number.
- **Compression Methods Length.** This 1-byte field defines the length of client-proposed compression methods (next field).
- **Compression Method List.** This variable-length field gives the list of compression methods that the client supports. The field lists the methods from the most preferred to the least preferred. Each method is encoded as a one-byte number. So far, the only method is the *NULL* method (no compression). In this case, the value of the compression method length is 1 and the compression method list has only one element with the value of 0.

ServerHello Message The ServerHello message is the server response to the ClientHello message. The format is similar to the ClientHello message, but with fewer fields. Figure 17.29 shows the format of the message.



Certificate Message:



- **Certificate Chain Length.** This three-byte field shows the length of the certificate chain. This field is redundant because its value is always 3 less than the value of the length field.

- **Certificate Chain.** This variable-length field lists the chain of public-key certificates that the client or the server carries. For each certificate, there are two sub-fields:
  a. A three-byte length field
  b. The variable-size certificate itself

ServerKeyExchange Message The ServerKeyExchange message is sent from the server to the client. Figure 17.31 shows the general format.

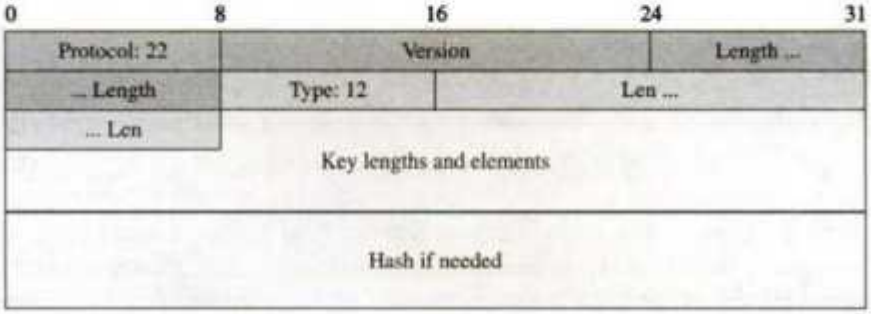
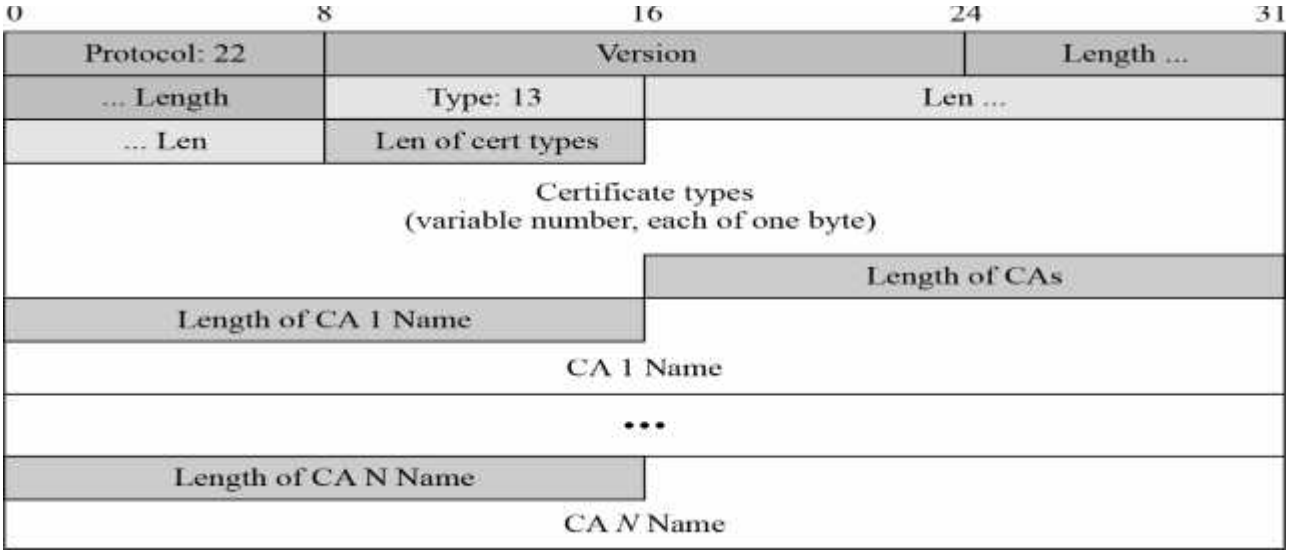


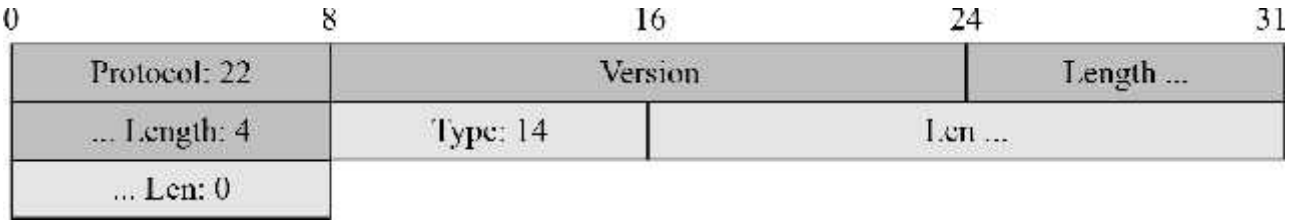
Fig. 17.31 *ServerKeyExchange message*

The message contains the keys generated by the server. The format of the message is dependent on the cipher suite selected in the previous message. The client that receives the message needs to interpret the message according to the previous information. If the server has sent a certificate message, then the message also contains a signed parameter.

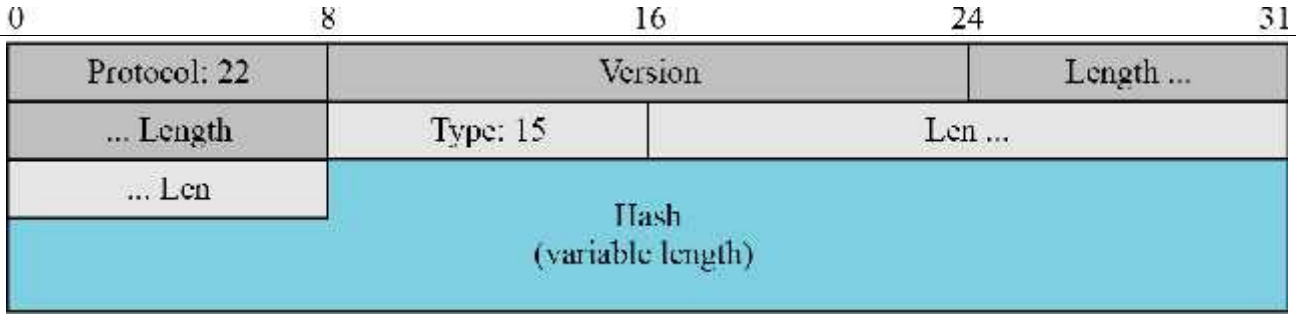
CertificateRequest Message The CertificateRequest message is sent from the server to the client. The message asks the client to authenticate itself to the server using one of the acceptable certificates and one of the certificate authorities named in the message. Figure 17.32 shows the format.



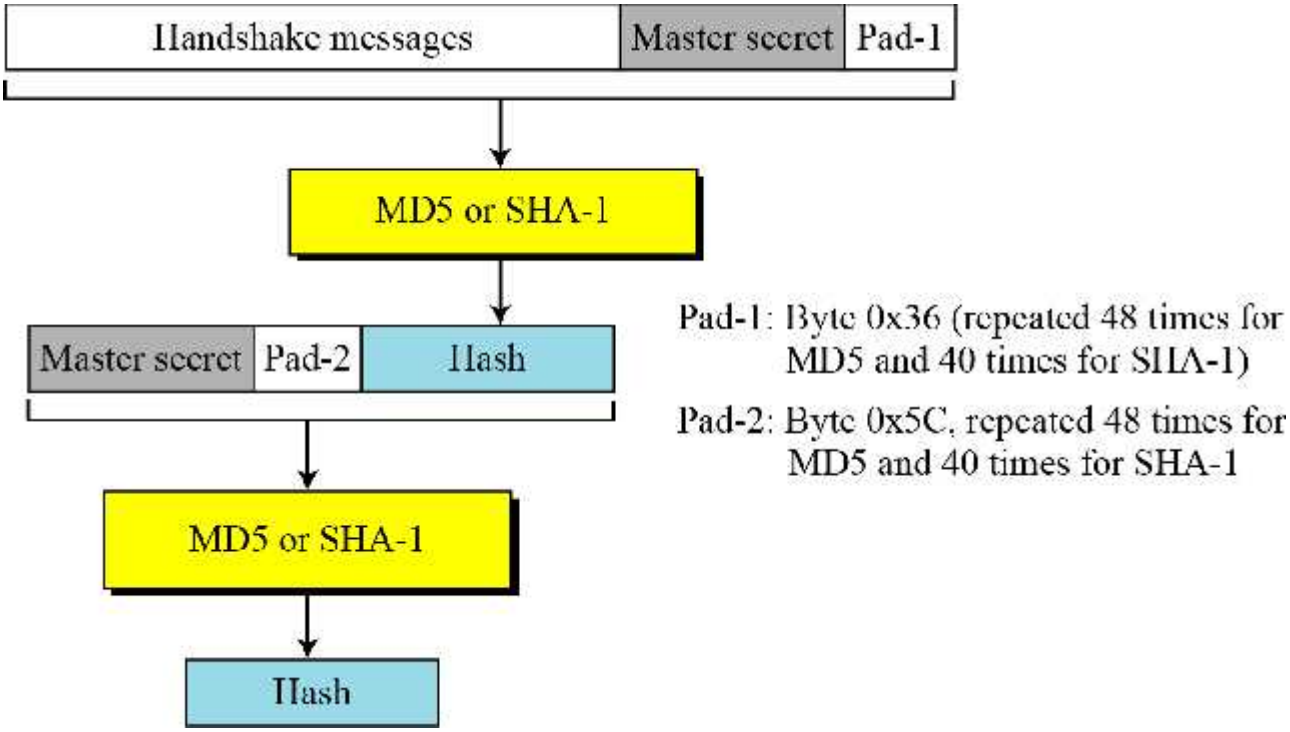
ServerHelloDone Message:

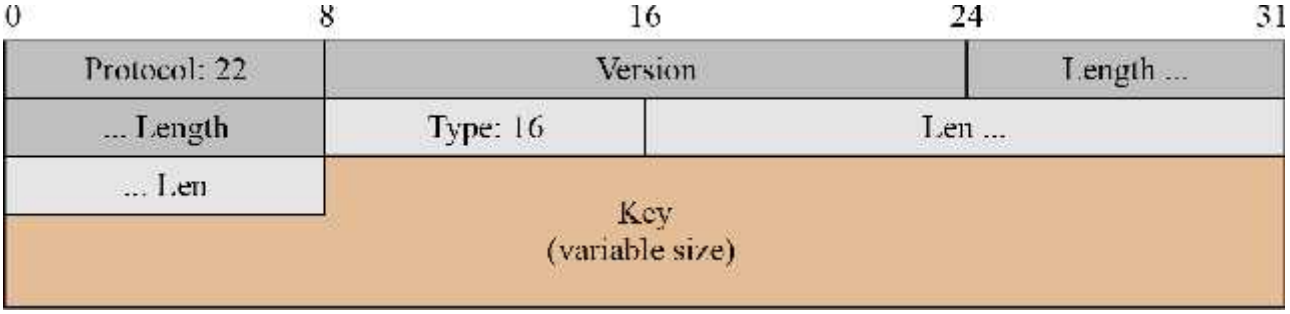


CertificateVerify Message:

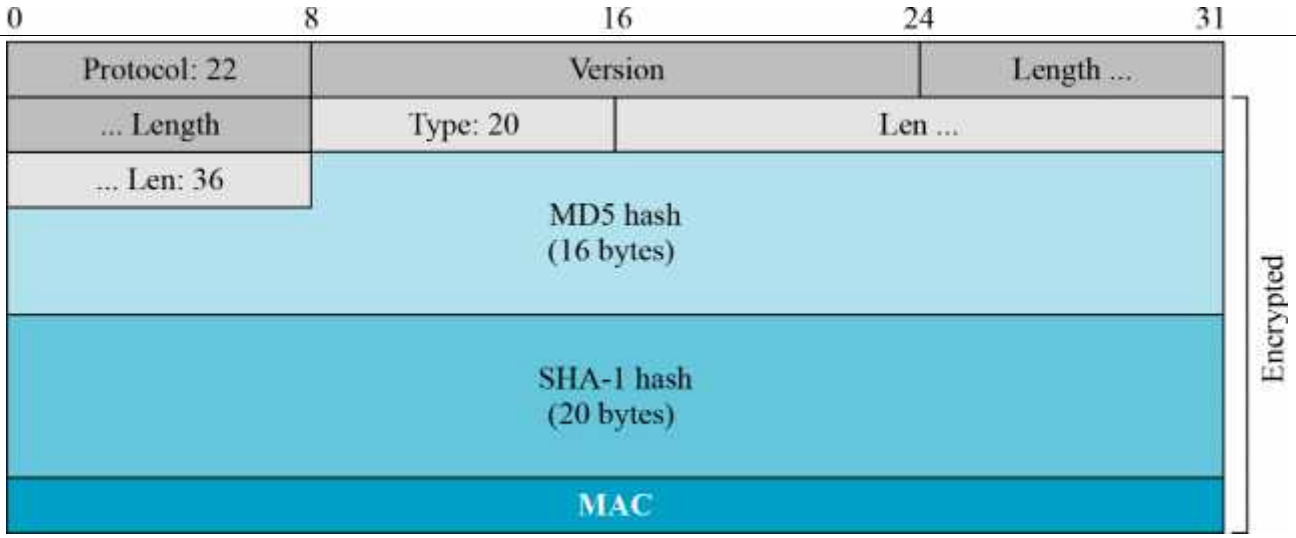


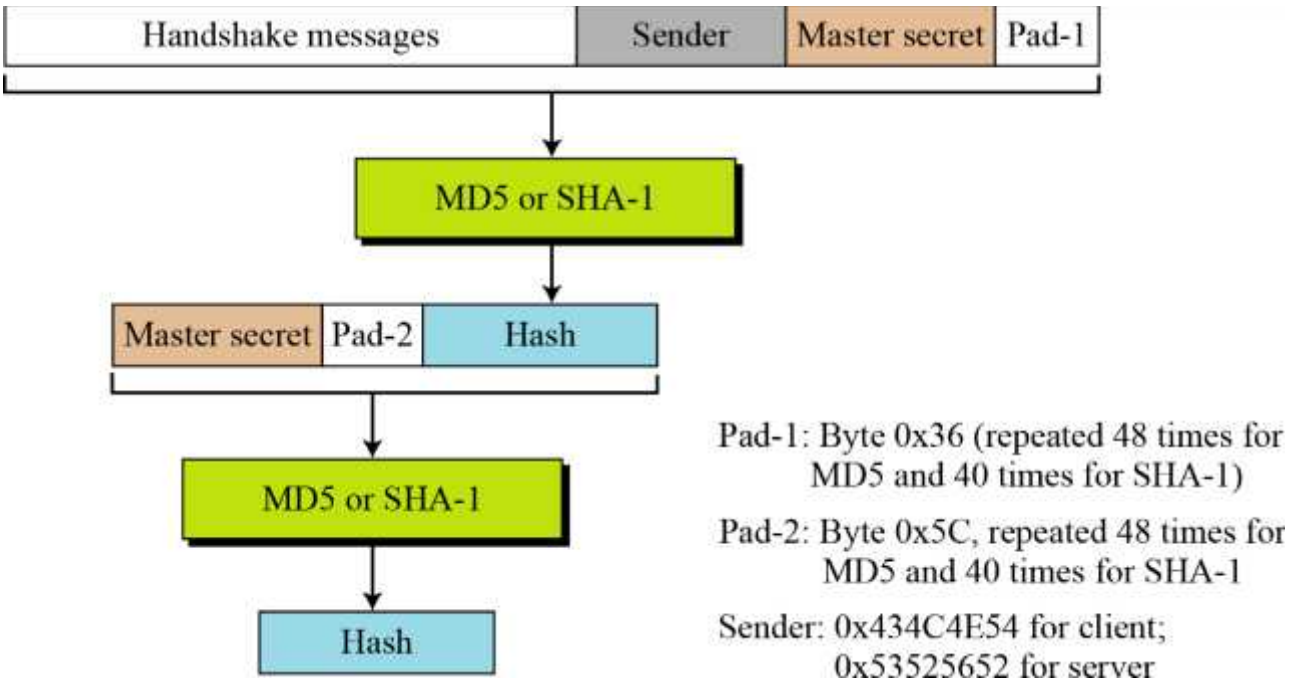
ClientKeyExchange Message:



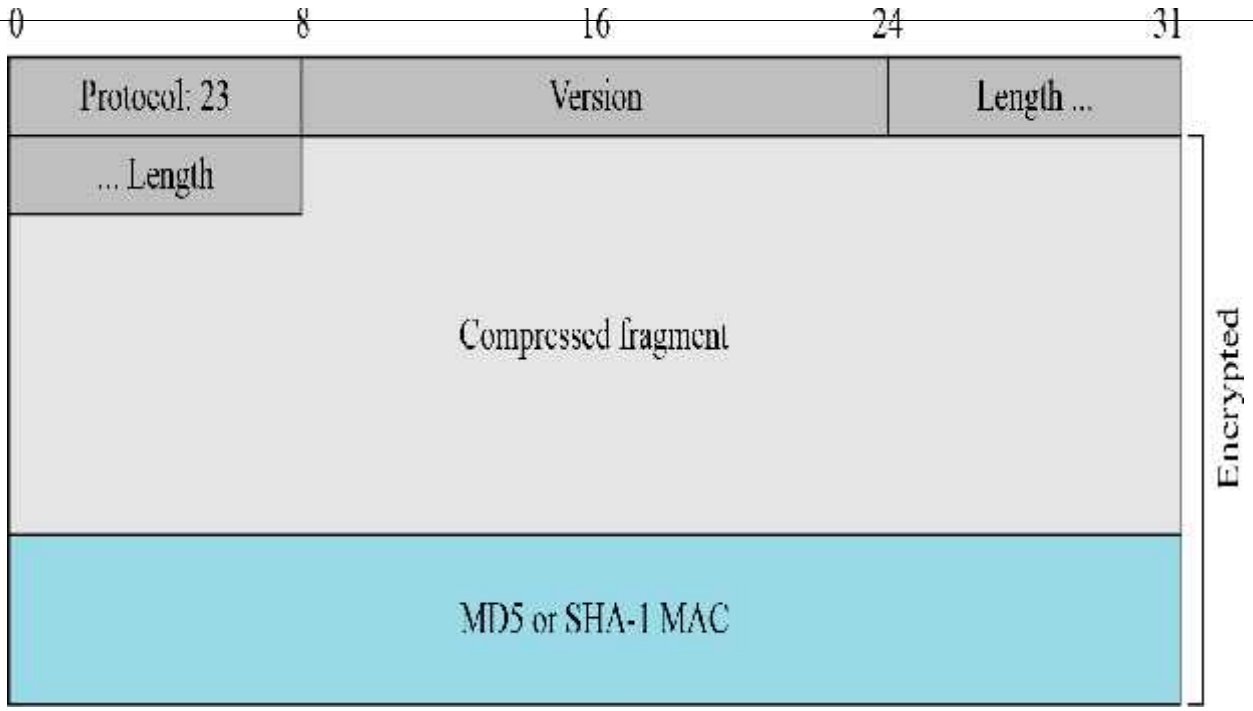


Finished Message:





Application Data:



- **TRANSPORT LAYER SECURITY**

- The Transport Layer Security (TLS) is the IETF standard version of the SSL protocol.
- The two are very similar, with slight differences.
- Instead of describing TLS in full , we highlight the differences between TLS and SSL protocols.

17.4.1 Version

The first difference is the version number (major and minor). The current version of SSL is 3.0; the current version of TLS is 1.0. In other words, SSLv3.0 is compatible with TLSv1.0.

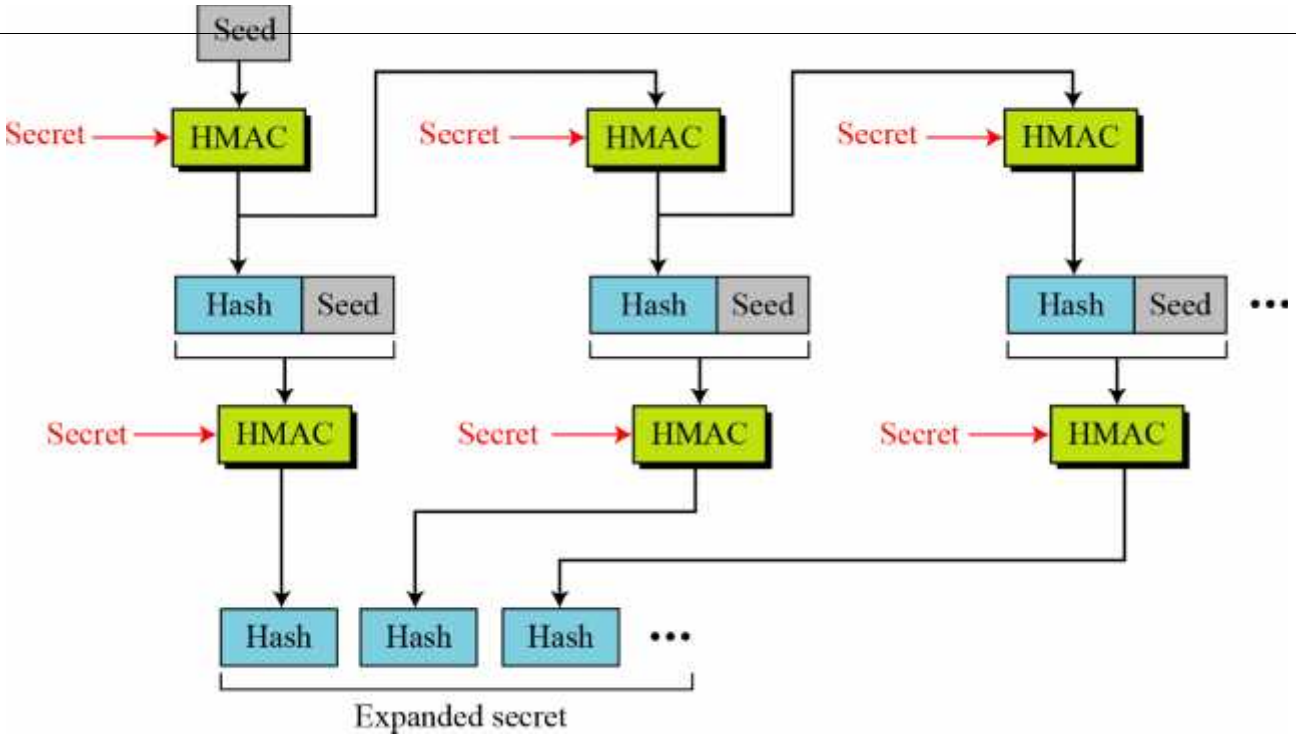
17.4.2 Cipher Suite

Another minor difference between SSL and TLS is the lack of support for the Fortezza method. TLS does not support Fortezza for key exchange or for encryption/decryption. Table 17.6 shows the cipher suite list for TLS (without export entries).

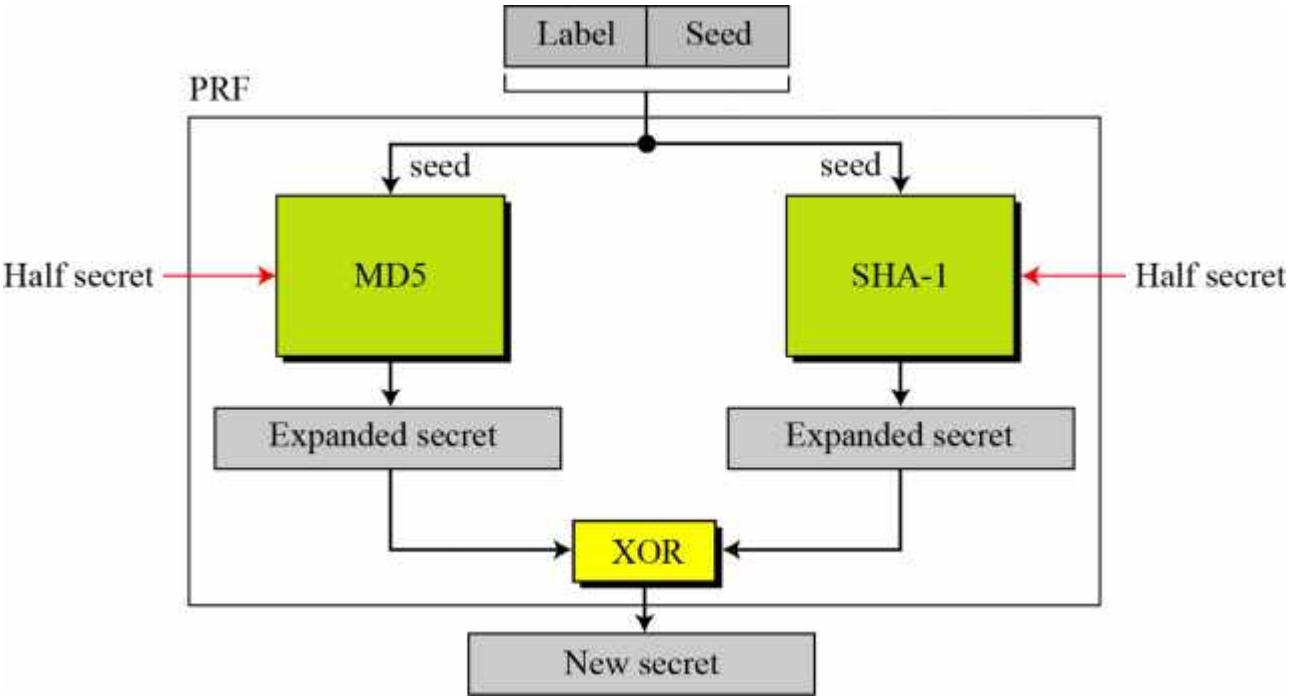
Cipher suite	*Key Exchange*	*Encryption*	*Hash*
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA-1
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4	SHA-1
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-1
TLS_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-1
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES	SHA-1
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4	MD5
TLS_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES	SHA-1
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES	SHA-1
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA-1
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES	SHA-1
TLS_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES	SHA-1
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES	SHA-1
TLS_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES	SHA-1
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES	SHA-1
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES	SHA-1
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES	SHA-1

The generation of cryptographic secrets is more complex in TLS than in SSL. TLS first defines two functions: the data-expansion function and the pseudorandom function. Let us discuss these two functions.

Data-Expansion Function The **data-expansion function** uses a predefined HMAC (either MD5 or SHA-1) to expand a secret into a longer one. This function can be considered a multiple-section function, where each section creates one hash value. The extended secret is the concatenation of the hash values. Each section uses two HMACs, a secret and a seed. The data-expansion function is the chaining of as many sections as required. However, to make the next section dependent on the previous, the second seed is actually the output of the first HMAC of the previous section as shown in Fig. 17.40.



TLS defines a **pseudorandom function (PRF)** to be the combination of two data-expansion functions, one using MD5 and the other SHA-1. PRF takes three inputs, a secret, a label, and a seed. The label and seed are concatenated and serve as the seed for each data-expansion function. The secret is divided into two halves; each half is used as the secret for each data-expansion function. The output of two data-expansion functions is exclusive-ored together to create the final expanded secret. Note that because the hashes created from MD5 and SHA-1 are of different sizes, extra sections of MD5-based functions must be created to make the two outputs the same size. Figure 17.41 shows the idea of PRF.



Pre master secret:

The generation of the pre-master secret in TLS is exactly the same as in SSL.

Master Secret TLS uses the PRF function to create the master secret from the pre-master secret. This is achieved by using the pre-master secret as the secret, the string "master secret" as the label, and concatenation of the client random number and server random number as the seed. Note that the label is actually the ASCII code of the string "master secret". In other words, the label defines the output we want to create, the master secret. Figure 17.42 shows the idea.

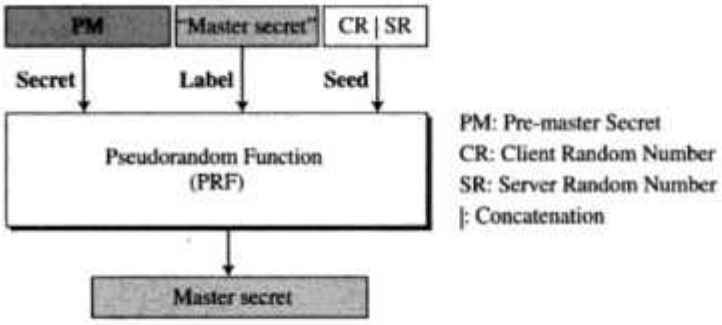
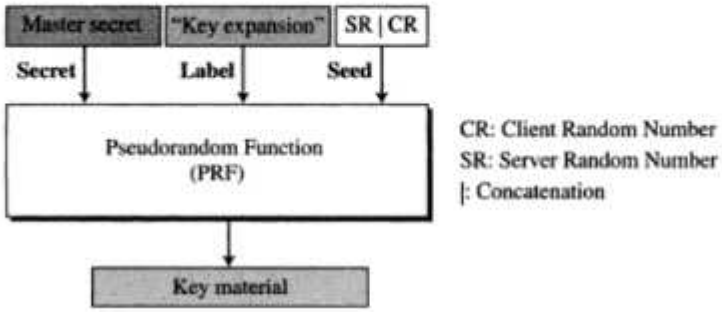


Fig. 17.42 *Master secret generation*

Key Material TLS uses the PRF function to create the key material from the master secret. This time the secret is the master secret, the label is the string "key expansion", and the seed is the concatenation of the server random number and the client random number, as shown in Fig. 17.43.



17.4.5 Alert Protocol

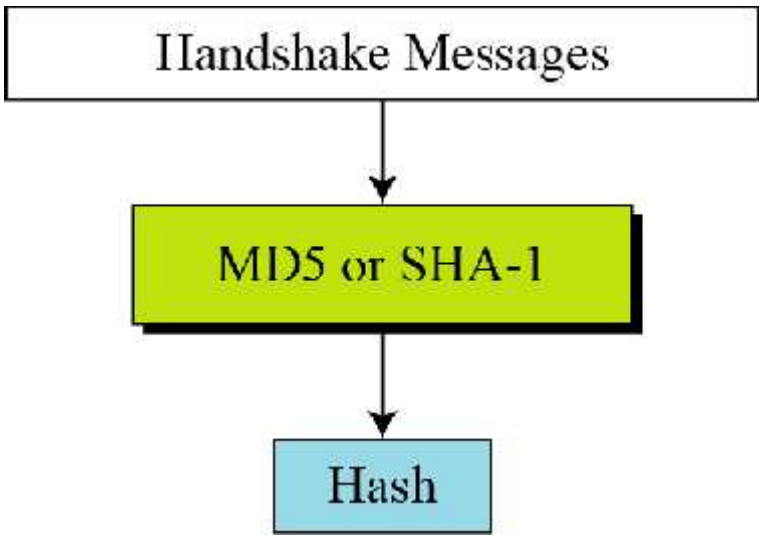
TLS supports all of the alerts defined in SSL except for *NoCertificate*. TLS also adds some new ones to the list. Table 17.7 shows the full list of alerts supported by TLS.

17.4.6 Handshake Protocol

TLS has made some changes in the Handshake Protocol. Specifically, the details of the CertificateVerify message and the Finished message have been changed.

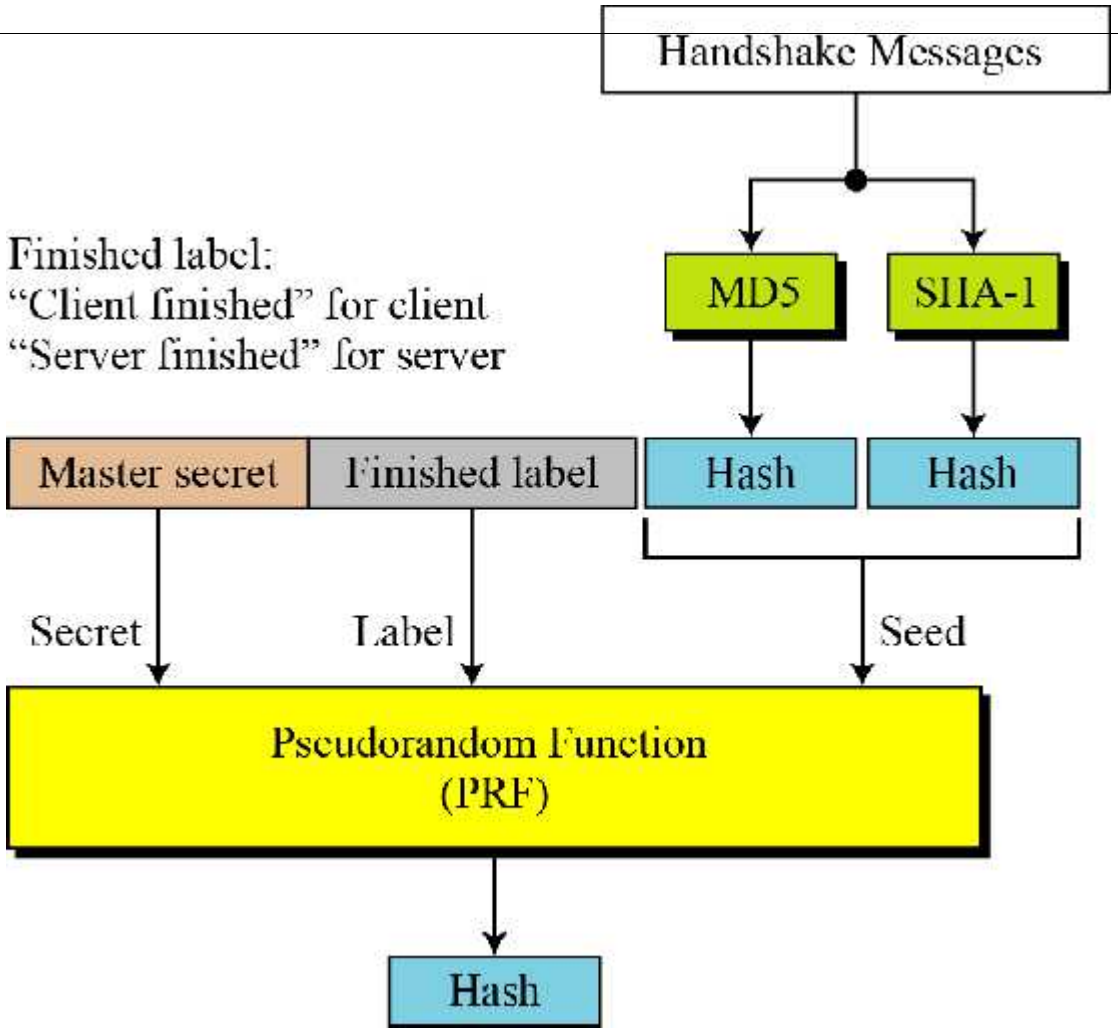
CertificateVerify Message In SSL, the hash used in the CertificateVerify message is the two-step hash of the handshake messages plus a pad and the master secret. TLS has simplified the process. The hash in the TLS is only over the handshake messages, as shown in Fig. 17.44.

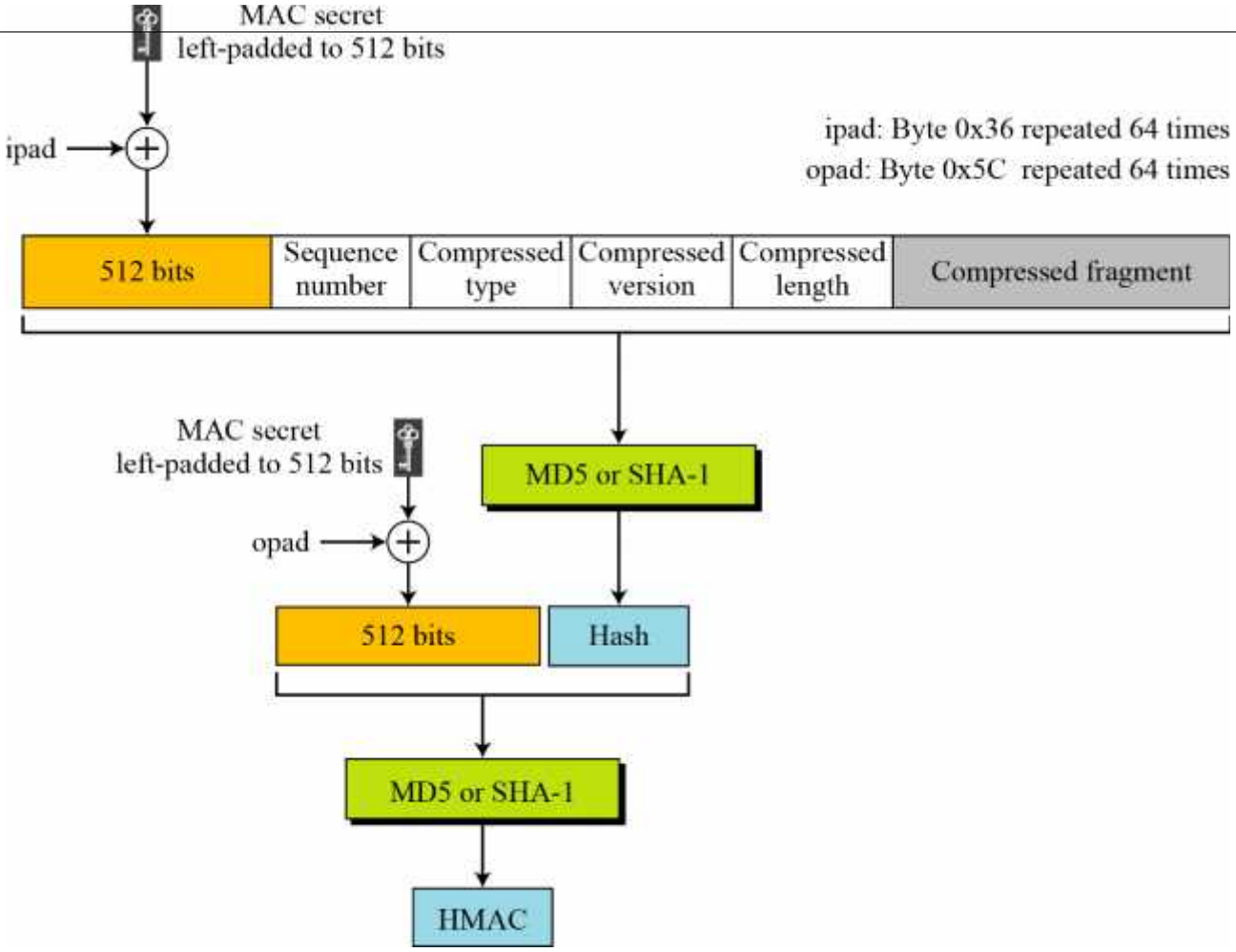
Value	Description	Meaning
0	*CloseNotify*	Sender will not send any more messages.
10	*UnexpectedMessage*	An inappropriate message received.
20	*BadRecordMAC*	An incorrect MAC received.
21	*DecryptionFailed*	Decrypted message is invalid.
22	*RecordOverflow*	Message size is more than $2^{14} + 2048$.
30	*DecompressionFailure*	Unable to decompress appropriately.
40	*HandshakeFailure*	Sender unable to finalize the handshake.
42	*BadCertificate*	Received certificate corrupted.
43	*UnsupportedCertificate*	Type of received certificate is not supported.
44	*CertificateRevoked*	Signer has revoked the certificate.
45	*CertificateExpired*	Certificate has expired.
46	*CertificateUnknown*	Certificate unknown.
47	*IllegalParameter*	A field out of range or inconsistent with others.
48	*UnknownCA*	CA could not be identified.



Finished Message: The calculation of the hash for the finished message has also been changed.

Record Protocol:


Handshake Messages
Finished label:
"Client finished" for client
"Server finished" for server
MD5
SHA-1
Master secret
Finished label
Hash
Hash
Secret
Label
Seed
Pseudorandom Function
(PRF)
Hash


MAC secret
left-padded to 512 bits
ipad
ipad: Byte 0x36 repeated 64 times
opad: Byte 0x5C repeated 64 times
512 bits
Sequence number
Compressed type
Compressed version
Compressed length
Compressed fragment
MAC secret
left-padded to 512 bits
MD5 or SHA-1
opad
512 bits
Hash
MD5 or SHA-1
HMAC

UNIT-VI

Syllabus:

Network Security-II

Security at the Network Layer: IPSec, System Security

Topic 1:

- **Security at the Network Layer:**

IPSec operates in one of two different modes: transport mode or tunnel mode.

18.1.1 Transport Mode

In **transport mode,** IPSec protects what is delivered from the transport layer to the network layer. In other words, transport mode protects the network layer payload, the payload to be encapsulated in the network layer, as shown in Fig. 18.2.

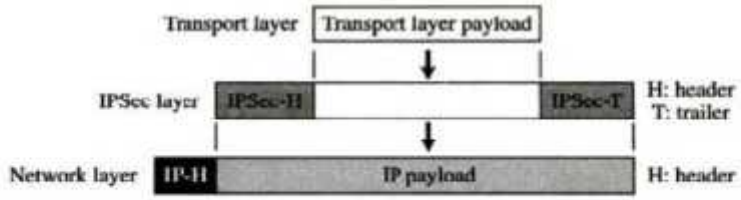


Fig. 18.2 *IPSec in transport mode*

Note that transport mode does not protect the IP header. In other words, transport mode does not protect the whole IP packet; it protects only the packet from the transport layer (the IP layer payload). In this mode, the IPSec header (and trailer) are added to the information coming from the transport layer. The IP header is added later.

IPSec in transport mode does not protect the IP header; it only protects the information coming from the transport layer.

Transport mode is normally used when we need host-to-host (end-to-end) protection of data. The sending host uses IPSec to authenticate and/or encrypt the payload delivered from the transport layer. The receiving host uses IPSec to check the authentication and/or decrypt the IP packet and deliver it to

the transport layer.

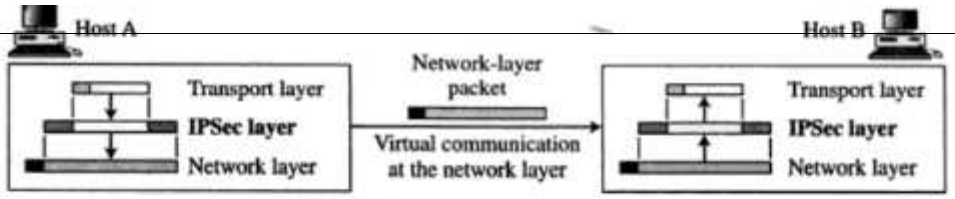


Fig. 18.3 *Transport mode in action*

18.1.2 Tunnel Mode

In **tunnel mode,** IPSec protects the entire IP packet. It takes an IP packet, including the header, applies IPSec security methods to the entire packet, and then adds a new IP header, as shown in Fig. 18.4.

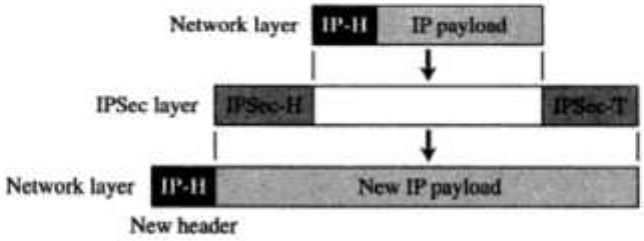


Fig. 18.4 *IPSec in tunnel mode*

The new IP header, as we will see shortly, has different information than the original IP header. Tunnel mode is normally used between two routers, between a host and a router, or between a router and a host, as shown in Fig. 18.5. In other words, tunnel mode is used when either the sender or the receiver is not a host. The entire original packet is protected from intrusion between the sender and the receiver, as if the whole packet goes through an imaginary tunnel.

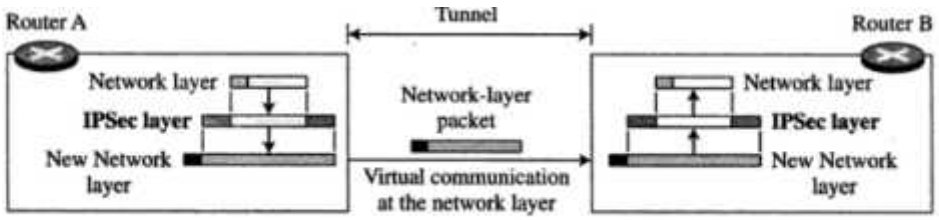
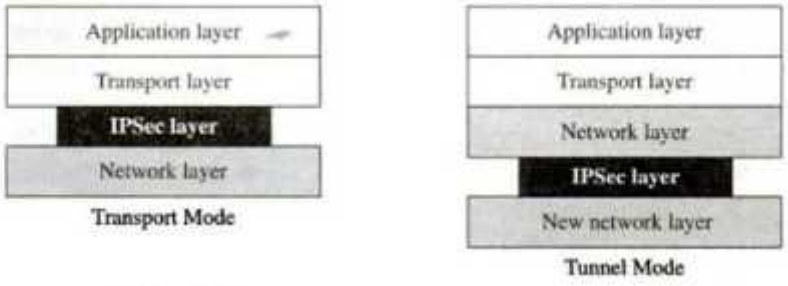


Fig. 18.5 *Tunnel mode in action*

IPSec in tunnel mode protects the original IP header.

18.1.3 Comparison

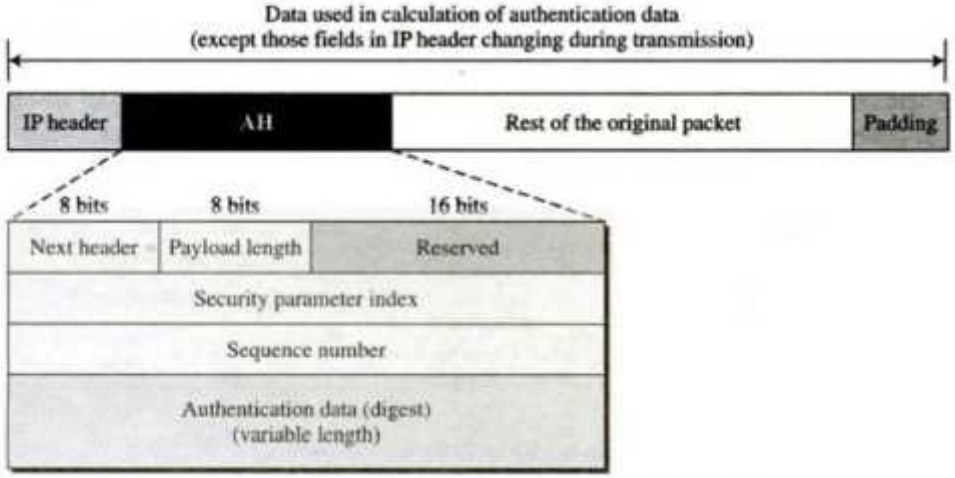
In transport mode, the IPSec layer comes between the transport layer and the network layer. In tunnel mode, the flow is from the network layer to the IPSec layer and then back to the network layer again.



TWO SECURITY PROTOCOLS:

It defines two protocols—the Authentication header(AH) Protocol and the Encapsulating Security Payload(ESP) –to provide authentication and/or encryption for packets at the IP level.

The **Authentication Header (AH) Protocol** is designed to authenticate the source host and to ensure the integrity of the payload carried in the IP packet. The protocol uses a hash function and a symmetric key to create a message digest; the digest is inserted in the authentication header. The AH is then placed in the appropriate location, based on the mode (transport or tunnel). Figure 18.7 shows the fields and the position of the authentication header in transport mode.



When an IP datagram carries an authentication header, the original value in the protocol field of the IP header is replaced by the value 51. A field inside the authentication header (the next header field) holds the original value of the protocol field (the type of payload being carried by the IP datagram). The addition of an authentication header follows these steps:

1. An authentication header is added to the payload with the authentication data field set to 0.

2. Padding may be added to make the total length even for a particular hashing algorithm.
3. Hashing is based on the total packet. However, only those fields of the IP header that do not change during transmission are included in the calculation of the message digest (authentication data).
4. The authentication data are inserted in the authentication header.
5. The IP header is added after changing the value of the protocol field to 51.

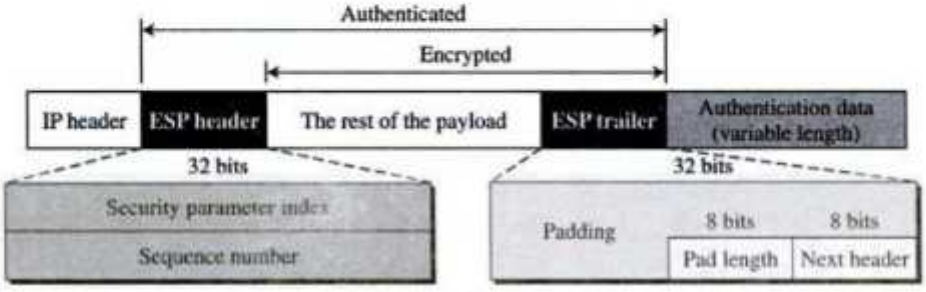
A brief description of each field follows:

- **Next header.** The 8-bit next header field defines the type of payload carried by the IP datagram (such as TCP, UDP, ICMP, or OSPF). It has the same function as the protocol field in the IP header before encapsulation. In other words, the process copies the value of the protocol field in the IP datagram to this field. The value of the protocol field in the new IP datagram is now set to 51 to show that the packet carries an authentication header.
- **Payload length.** The name of this 8-bit field is misleading. It does not define the length of the payload; it defines the length of the authentication header in 4-byte multiples, but it does not include the first 8 bytes.
- **Security parameter index.** The 32-bit security parameter index (SPI) field plays the role of a virtual circuit identifier and is the same for all packets sent during a connection called a Security Association (discussed later).
- **Sequence number.** A 32-bit sequence number provides ordering information for a sequence of datagrams. The sequence numbers prevent a playback. Note that the sequence number is not repeated even if a packet is retransmitted. A sequence number does not wrap around after it reaches 2^{32}; a new connection must be established.
- **Authentication data.** Finally, the authentication data field is the result of applying a hash function to the entire IP datagram except for the fields that are changed during transit (e.g., time-to-live).

The AH protocol provides source authentication and data integrity, but not privacy.

Encapsulating Security Payload(ESP):

The AH protocol does not provide privacy, only source authentication and data integrity. IPSec later defined an alternative protocol, **Encapsulating Security Payload (ESP),** that provides source authentication, integrity, and privacy. ESP adds a header and trailer. Note that ESP's authentication data are added at the end of the packet, which makes its calculation easier. Figure 18.8 shows the location of the ESP header and trailer.



When an IP datagram carries an ESP header and trailer, the value of the protocol field in the IP header is 50. A field inside the ESP trailer (the next-header field) holds the original value of the protocol field (the type of payload being carried by the IP datagram, such as TCP or UDP). The ESP procedure follows these steps:

1. An ESP trailer is added to the payload.
2. The payload and the trailer are encrypted.
3. The ESP header is added.
4. The ESP header, payload, and ESP trailer are used to create the authentication data.
5. The authentication data are added to the end of the ESP trailer.
6. The IP header is added after changing the protocol value to 50.

The fields for the header and trailer are as follows:

- **Security parameter index.** The 32-bit security parameter index field is similar to that defined for the AH protocol.
- **Sequence number.** The 32-bit sequence number field is similar to that defined for the AH protocol.
- **Padding.** This variable-length field (0 to 255 bytes) of 0s serves as padding.
- **Pad length.** The 8-bit pad-length field defines the number of padding bytes. The value is between 0 and 255; the maximum value is rare.
- **Next header.** The 8-bit next-header field is similar to that defined in the AH protocol. It serves the same purpose as the protocol field in the IP header before encapsulation.
- **Authentication data.** Finally, the authentication data field is the result of applying an authentication scheme to parts of the datagram. Note the difference between the authentication data in AH and ESP. In AH, part of the IP header is included in the calculation of the authentication data; in ESP, it is not.

ESP provides source authentication, data integrity, and privacy.

IPV4 and IPV6:

IPSec supports both IPv4 and IPv6 . In IPv6 however, AH and ESP are part of the extension header.

AH versus ESP:

The ESP protocol was designed after the AH protocol was already in use. ESP does whatever AH does with additional functionality (privacy). The question is, Why do we need AH? The answer is that we don't. However, the implementation of AH is already included in some commercial products, which means that AH will remain part of the Internet until these products are phased out.

18.2.5 Services Provided by IPSec

The two protocols, AH and ESP, can provide several security services for packets at the network layer. Table 18.1 shows the list of services available for each protocol.

Services	AH	ESP
Access control	yes	yes
Message authentication (message integrity)	yes	yes
Entity authentication (data source authentication)	yes	yes
Confidentiality	no	yes
Replay attack protection	yes	yes

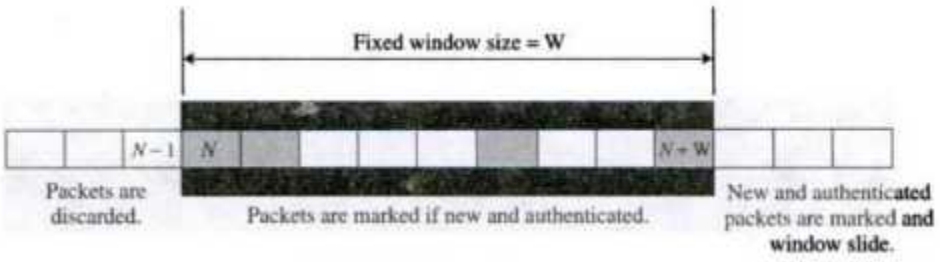
Access Control IPSec provides access control indirectly using a Security Association Database (SAD), as we will see in the next section. When a packet arrives at a destination, and there is no Security Association already established for this packet, the packet is discarded.

Message Integrity Message integrity is preserved in both AH and ESP. A digest of data is created and sent by the sender to be checked by the receiver.

Entity Authentication The Security Association and the keyed-hash digest of the data sent by the sender authenticate the sender of the data in both AH and ESP.

Confidentiality The encryption of the message in ESP provides confidentiality. AH, however, does not provide confidentiality. If confidentiality is needed, one should use ESP instead of AH.

Replay attack Protection:



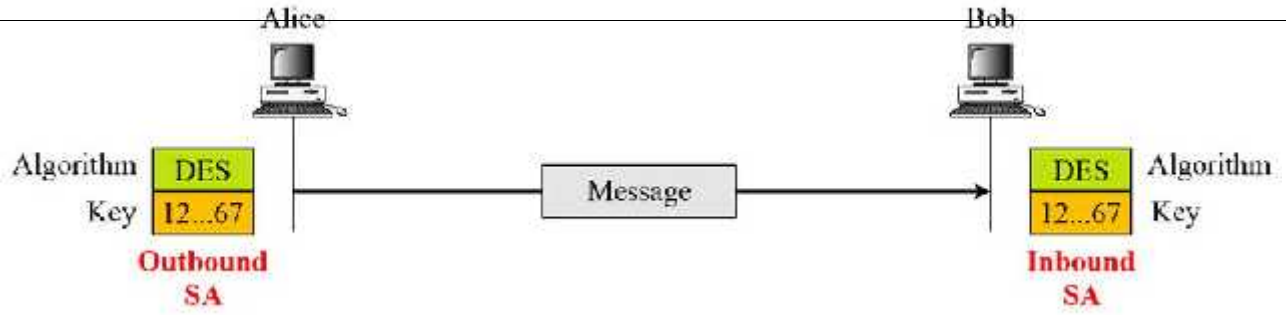
When a packet arrives at the receiver, one of three things can happen, depending on the value of the sequence number.

- **SECURITY ASSOCIATION:**

Security Association is a very important aspect of IPSec.

Idea of Security Association:

- A security association is a contract between two parties; it creates a secure channel between them.
- Let us assume that Alice needs to unidirectionally communicate with bob.
- If Alice and Bob are interested in the confidentiality aspect of security, they can get a shared secret key between themselves.
- We can say that there are two SAs between Alice and Bob; one outbound SA and one Inbound SA.



Security Association Database(SAD):

- A Security Association can be very complex.
- This is particularly true if Alice wants to send messages to many people and bob needs to receive messages from many people.

Index	SN	OF	ARW	AH/ESP	LT	Mode	MTU
< SPI, DA, P >							
< SPI, DA, P >							
< SPI, DA, P >							
< SPI, DA, P >							

Security Association Database

Legend:

SPI: Security Parameter Index	SN: Sequence Number
DA: Destination Address	OF: Overflow Flag
AH/ESP: Information for either one	ARW: Anti-Replay Window
P: Protocol	LT: Lifetime
Mode: IPSec Mode Flag	MTU: Path MTU

Security Parameter Index:

Destination Address:

Protocol:

Parameters	*Description*
Sequence Number Counter	This is a 32-bit value that is used to generate sequence numbers for the AH or ESP header.
Sequence Number Overflow	This is a flag that defines a station's options in the event of a sequence number overflow.
Anti-Replay Window	This detects an inbound replayed AH or ESP packet.
AH Information	This section contains information for the AH protocol: 1. Authentication algorithm 2. Keys 3. Key lifetime 4. Other related parameters
ESP Information	This section contains information for the ESP protocol: 1. Encryption algorithm 2. Authentication algorithm 3. Keys 4. Key lifetime 5. Initiator vectors 6. Other related parameters
SA Lifetime	This defines the lifetime for the SA.
IPSec Mode	This defines the mode, transport or tunnel.
Path MTU	This defines the path MTU (fragmentation).

SECURITY POLICY:

It defines the type of security applied to a packet when it is to be sent or when it has arrived.

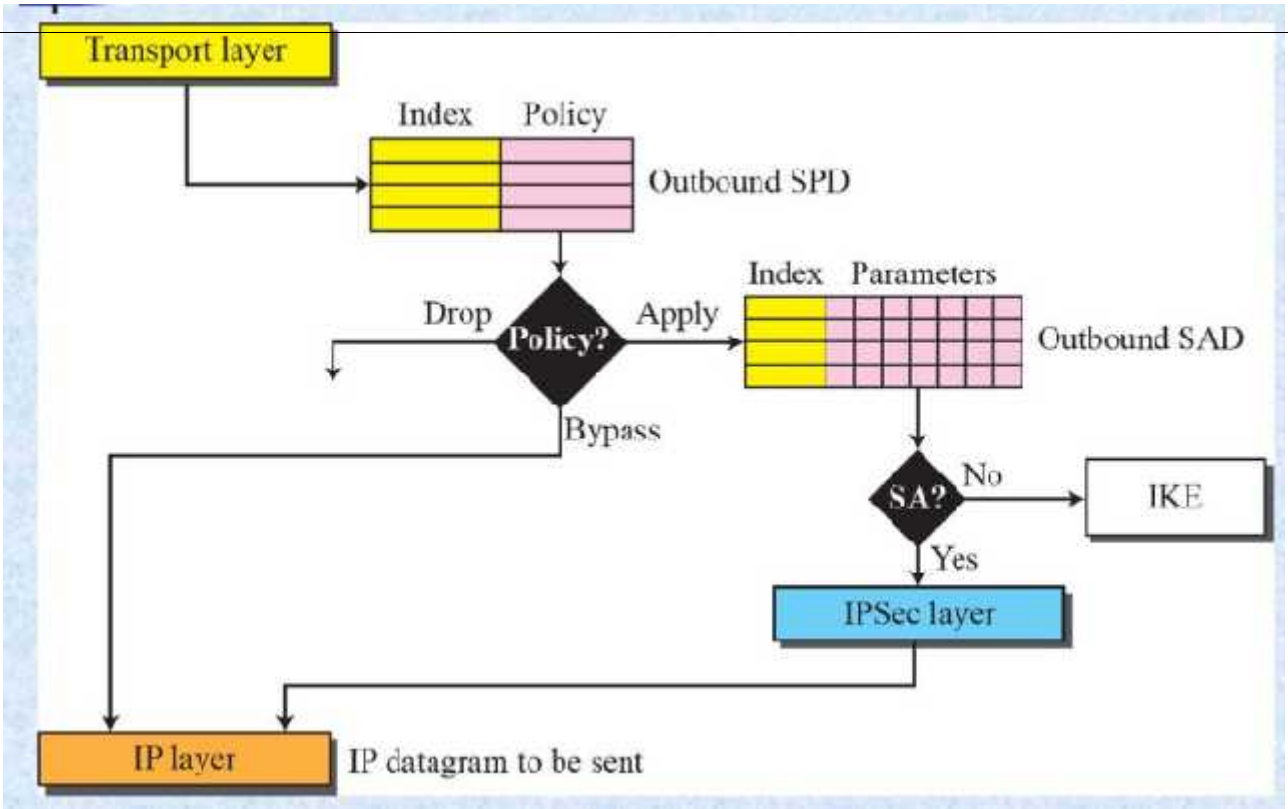
Security Policy Database:

Index	Policy
< SA, DA, Name, P, SPort, DPort >	
< SA, DA, Name, P, SPort, DPort >	
< SA, DA, Name, P, SPort, DPort >	
< SA, DA, Name, P, SPort, DPort >	

Legend:

SA: Source Address
DA: Destination Address
P: Protocol
SPort: Source Port
DPort: Destination Port

OUTBOUND SPD: When a packet is to be sent out, the outbound SPD is consulted.



1.Drop. This means that the packet defined by the index cannot be sent; it is dropped.

2. Bypass. This means that there is no policy for the packet with this policy index; the packet is sent, bypassing the security header application.

3.Apply: In this bound SA is already established, the triple SA index is returned that selects the corresponding SA from the outbound SAD.

❑ ***Inbound SPD*** When a packet arrives, the inbound SPD is consulted. Each entry in the inbound SPD is also accessed using the same sextuple index. Figure 18.14 shows the processing of a packet by a receiver.

The input to the inbound SPD is the sextuple index; the output is one of the three following cases:

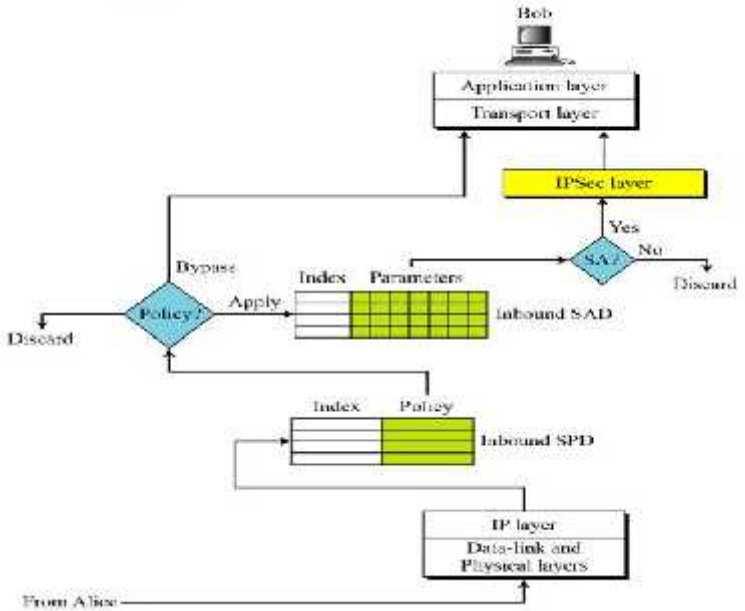
• *Discard* This means that the packet defined by that policy must be dropped.

• *Bypass* This means that there is no policy for a packet with this policy index; the packet is processed, ignoring the information from AH or ESP header. The packet is delivered to the transport layer.

• *Apply* In this case, the security header must be processed. Two situations may occur:

a. If an inbound SA is already established, the triple SA index is returned that selects the correspondinginbound SA from the inbound SAD. Decryption, authentication, or both are applied. If the packetpasses the security criteria, the AH or ESP header is discarded and the packet is delivered to thetransport layer.

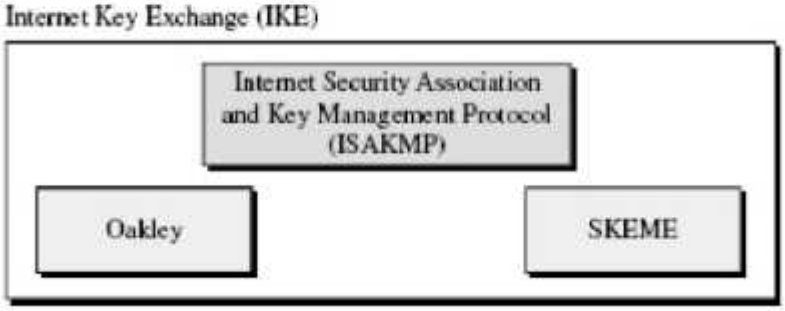
b. If an SA is not yet established, the packet must be discarded.



➢ **INTERNET KEY EXCHANGE:**

The **Internet Key Exchange (IKE)** is a protocol designed to create both inbound and outbound Security Associations. As we discussed in the previous section, when a peer needs to send an IP packet, it consults the Security Policy Database (SPDB) to see if there is an SA for that type of traffic. If there is no SA, IKE is called to establish one.

The **Oakley** protocol was developed by Hilarie Orman. It is a key creation protocol based on the Diffie-Hellman key-exchange method, but with some improvements as we shall see shortly. Oakley is a free-formatted protocol in the sense that it does not define the format of the message to be exchanged. We do not discuss the Oakley protocol directly in this chapter, but we show how IKE uses its ideas.

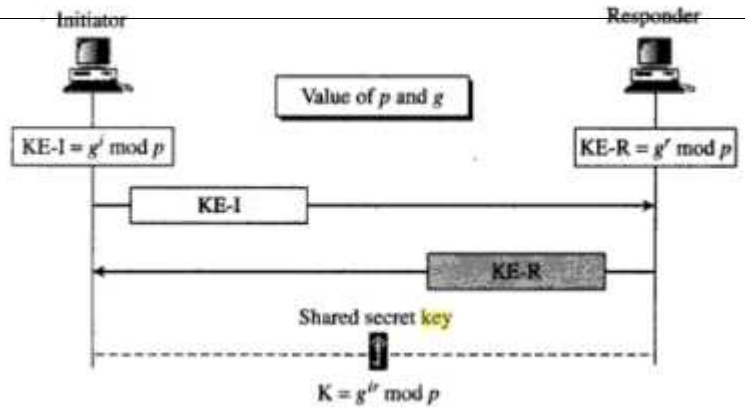


SKEME, designed by Hugo Krawcyzk, is another protocol for key exchange. It uses public-key encryption for entity authentication in a key-exchange protocol. We will see shortly that one of the methods used by IKE is based on SKEME.

The **Internet Security Association and Key Management Protocol (ISAKMP)** is a protocol designed by the National Security Agency (NSA) that actually implements the exchanges defined in IKE. It defines several packets, protocols, and parameters that allow the IKE exchanges to take place in standardized, formatted messages to create SAs. We will discuss ISAKMP in the next section as the carrier protocol that implements IKE.

Improved Diffie-Hellman key exchange:

- The key-exchange idea in ike is based on the Diffie-Hellman protocol.
- This protocol provides a session key between two peers without the need for the existence of any previous secret.



In the original Diffie-Hellman key exchange, two parties create a symmetric session key to exchange data without having to remember or store the key for future use. Before establishing a symmetric key, the two parties need to choose two numbers p and g. The first number, p, is a large prime on the order of 300 decimal digits (1024 bits). The second number, g, is a generator in the group $<\mathbf{Z}_p^*, \times>$. Alice chooses a large random number i and calculates KE-I = g^i mod p. She sends KE-I to Bob. Bob chooses another large random number r and calculates KE-R = g^r mod p. He sends KE-R to Alice. We refer to KE-I and KE-R as Diffie-Hellman half-keys because each is a half-key generated by a peer. They need to be combined together to create the full key, which is K = g^{ir} mod p. K is the symmetric key for the session.

The Diffie-Hellman protocol has some weaknesses that need to be eliminated before it is suitable as an Internet key exchange.

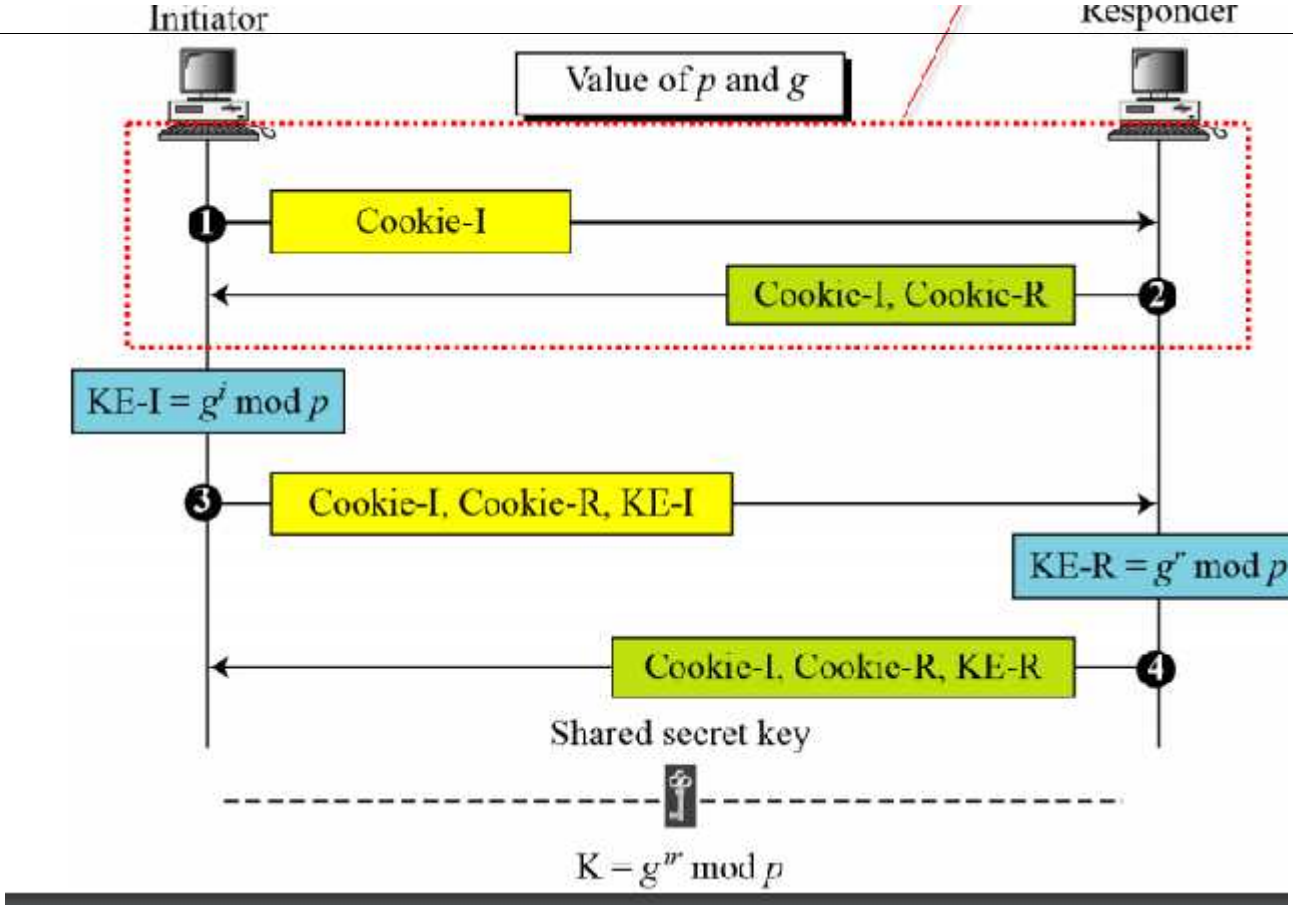
Clogging Attack

The first issue with the Diffie-Hellman protocol is the **clogging attack** or *denial-of-service attack*. A malicious intruder can send many half-key (g^x mod q) messages to Bob, pretending that they are from different sources. Bob then needs to calculate different responses (g^y mod q) and at the same time calculate the full-key (g^{xy} mod q). This keeps Bob so busy that he may stop responding to any other messages. He denies services to clients. This can happen because the Diffie-Hellman protocol is computationally intensive.

To prevent this clogging attack, we can add two extra messages to the protocol to force the two parties to send **cookies.** Figure 18.17 shows the refinement that can prevent a clogging attack. The cookie is the result of hashing a unique identifier of the peer (such as IP address, port number, and protocol), a secret random number known to the party that generates the cookie, and a timestamp.

:

Replay attack:



Man-In The Middle Attack:

Authentication of the messages exchanged (message integrity) and the authentication of the parties involved (entity authentication) require that each party proves his/her claimed identity. To do this, each must prove that it possesses a secret.

To protect against man-in-the-middle attack, IKE requires that each party shows that it possesses a secret.

In IKE, the secret can be one of the following:

a. A preshared secret key
b. A preknown encryption/decryption public-key pair. An entity must show that a message encrypted with the announced public key can be decrypted with the corresponding private key.
c. A preknown digital signature public-key pair. An entity must show that it can sign a message with its private key which can be verified with its announced public key.

IKE Phases

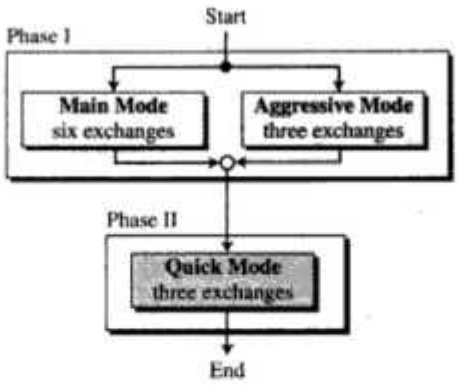
IKE creates SAs for a message-exchange protocol such as IPSec. IKE, however, needs to exchange confidential and authenticated messages. What protocol provides SAs for IKE itself? The reader may realize that this requires a never-ending chain of SAs: IKE must create SAs for IPSec, protocol X must create SAs for IKE, protocol Y needs to create SAs for protocol X, and so on. To solve this dilemma and, at the same time, make IKE independent of the IPSec protocol, the designers of IKE divided IKE into two phases. In phase I, IKE creates SAs for phase II. In phase II, IKE creates SAs for IPSec or some other protocol. Phase I is generic; phase II is specific for the protocol.

IKE is divided into two phases: phase I and phase II. Phase I creates SAs for phase II; phase II creates SAs for a data exchange protocol such as IPSec.

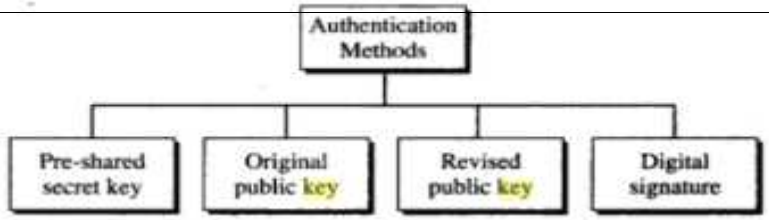
Still, the question remains: How is phase I protected? In the next sections we show how phase I uses an SA that is formed in a gradual manner. Earlier messages are exchanged in plaintext; later messages are authenticated and encrypted with the keys created from the earlier messages.

Phases and Modes

To allow for a variety of exchange methods, IKE has defined modes for the phases. So far, there are two modes for phase I: the *main mode* and the *aggressive mode*. The only mode for phase II is the *quick mode*. Figure 18.18 shows the relationship between phases and modes.



Based on the nature of the pre-secret between the two parties, the phase I modes can use one of four different authentication methods: the preshared secret key method, the original public-key method, the revised public-key method, or the digital signature method, as shown in Figure 18.19.



Phase I: Main Mode

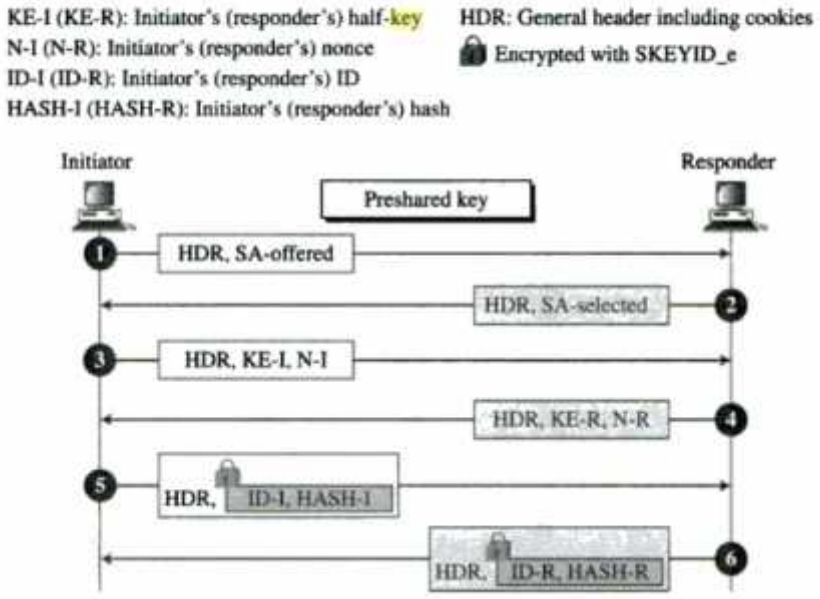
In the **main mode,** the initiator and the responder exchange six messages. In the first two messages, they exchange cookies (to protect against a clogging attack) and negotiate the SA parameters. The initiator sends a series of proposals; the responder selects one of them. When the first two messages are exchanged, the initiator and the responder know the SA parameters and are confident that the other party exists (no clogging attack occurs).

In the third and fourth messages, the initiator and responder usually exchange their half-keys (g^i and g^r of the Diffie-Hellman method) and their nonces (for replay protection). In some methods other information is exchanged; that will be discussed later. Note that the half-keys and nonces are not sent with the first two messages because the two parties must first ensure that a clogging attack is not possible.

Preshared Secret-Key Method

In the preshared secret-key method, a symmetric key is used for authentication of the peers to each other. Figure 18.20 shows shared-key authentication in the main mode.

Figure 18.20 *Main mode, preshared secret-key method*



$$SKEYID = prf(\text{preshared-key}, N\text{-}I \mid N\text{-}R) \quad \text{(preshared-key method)}$$

$$SKEYID = prf(N\text{-}I \mid N\text{-}R, g^{ir}) \quad \text{(public-key method)}$$

$$SKEYID = prf(\text{hash}(N\text{-}I \mid N\text{-}R), \text{Cookie-}I \mid \text{Cookie-}R) \quad \text{(digital signature)}$$

Other common secrets are calculated as follows:

$$SKEYID\_d = prf(SKEYID, g^{ir} \mid \text{Cookie-}I \mid \text{Cookie-}R \mid 0)$$

$$SKEYID\_a = prf(SKEYID, SKEYID\_d \mid g^{ir} \mid \text{Cookie-}I \mid \text{Cookie-}R \mid 1)$$

$$SKEYID\_e = prf(SKEYID, SKEYID\_a \mid g^{ir} \mid \text{Cookie-}I \mid \text{Cookie-}R \mid 2)$$

In the first two messages, the initiator and responder exchange cookies (inside the general header) and SA parameters. In the next two messages, they exchange the half-keys and the nonces (see Chapter 15). Now the two parties can create SKEYID and the two keyed hashes (HASH-I and HASH-R). In the fifth and sixth messages, the two parties exchange the created hashes and their IDs. To protect the IDs and hashes, the last two messages are encrypted with SKEYID_e.

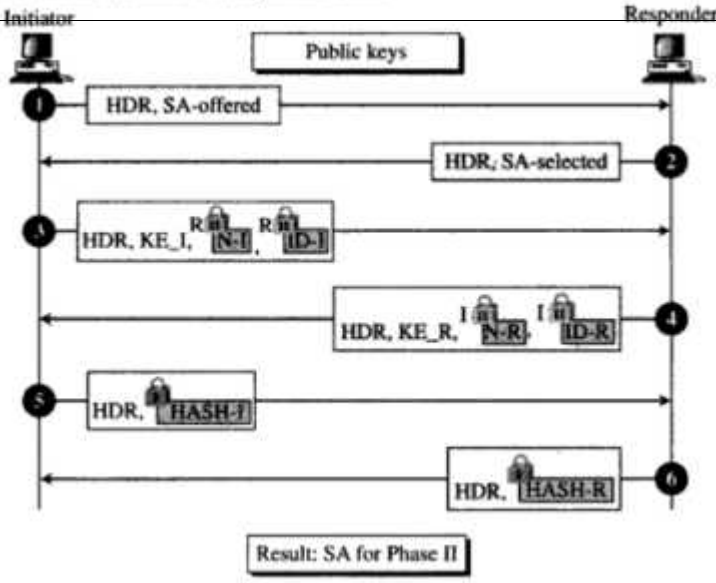
Note that the pre-shared key is the secret between Alice (initiator) and Bob (responder). Eve (intruder) does not have access to this key. Eve cannot create SKEYID and therefore cannot create either HASH-I or HASH-R. Note that the IDs need to be exchanged in messages 5 and 6 to allow the calculation of the hash.

There is one problem with this method. Bob cannot decrypt the message unless he knows the preshared key, which means he must know who Alice is (know her ID). But Alice's ID is encrypted in message 5. The designer of this method has argued that the

Original Public-Key Method

In the original public-key method, the initiator and the responder prove their identities by showing that they possess a private key related to their announced public key. Figure 18.21 shows the exchange of messages using the original public-key method.

Figure 18.21 *Main mode, original public-key method*



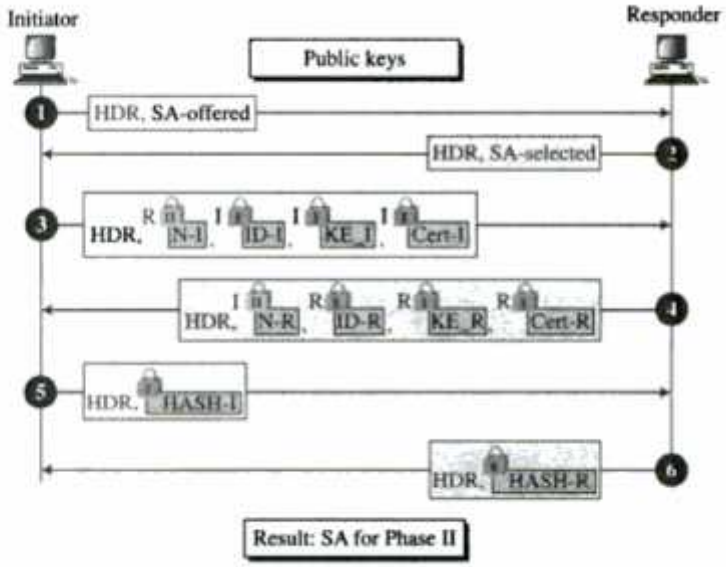
Revised Public-Key Method

The original public-key method has some drawbacks. First, two instances of public-key encryption/decryption place a heavy load on the initiator and responder. Second, the initiator cannot send its certificate encrypted by the public key of the responder, since anyone could do this with a false certificate. The method was revised so that the public key is used only to create a temporary secret key, as shown in Figure 18.22.

Figure 18.22 *Main mode, revised public-key method*

HDR: General header including cookies
KE-I (KE-R): Initiator's (responder's) half-key
Cert-I (Cert-R): Initiator's (responder's) certificate
N-I (N-R): Initiator's (responder's) nonce
ID-I (ID-R): Initiator's (responder's) ID
HASH-I (HASH-R): Initiator's (responder's) hash

I — Encrypted with initiator's public key
R — Encrypted with responder's public key
R — Encrypted with responder's secret key
I — Encrypted with initiator's secret key
Encrypted with SKEYID_e



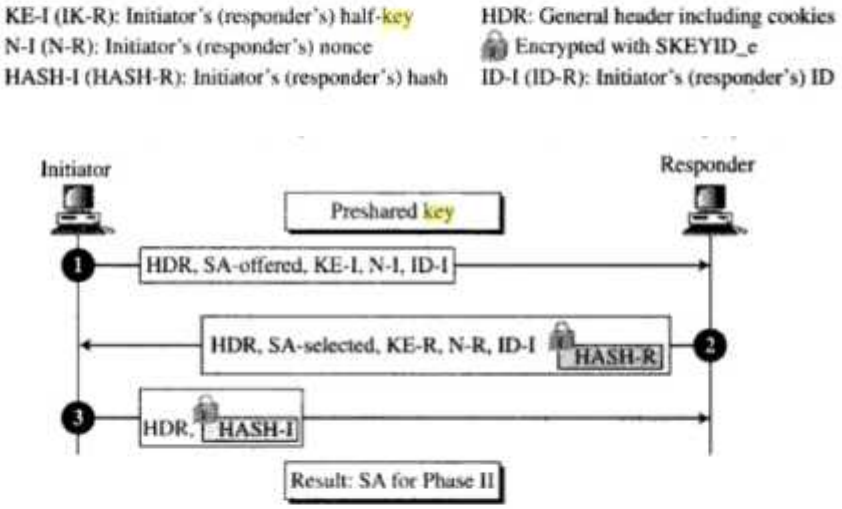
Phase I: Aggressive Mode

Each **aggressive mode** is a compressed version of the corresponding main mode. Instead of six messages, only three are exchanged. Messages 1 and 3 are combined to make the first message. Messages 2, 4, and 6 are combined to make the second message. Message 5 is sent as the third message. The idea is the same.

Preshared-Key Method

Figure 18.24 shows the preshared-key method in the aggressive mode. Note that after receiving the first message, the responder can calculate SKEYID and consequently, HASH-R. But the initiator cannot calculate SKEYID until it receives the second message. HASH-I in the third message can be encrypted.

Figure 18.24 *Aggressive mode, preshared-key method*



Original Public-Key Method

Figure 18.25 shows the exchange of messages using the original public-key method in the aggressive mode. Note that the responder can calculate the SKEYID and HASH-R after receiving the first message, but the initiator must wait until it receives the second message.

Revised Public-Key Method

Figure 18.26 shows the revised public-key method in the aggressive mode. The idea is the same as for the main mode, except that some messages are combined.

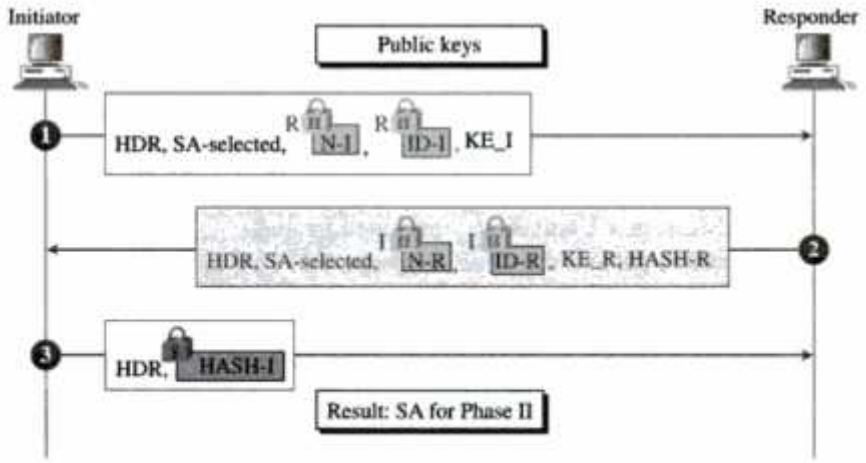
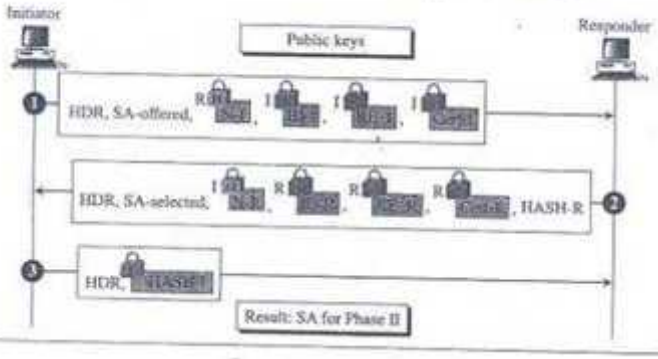


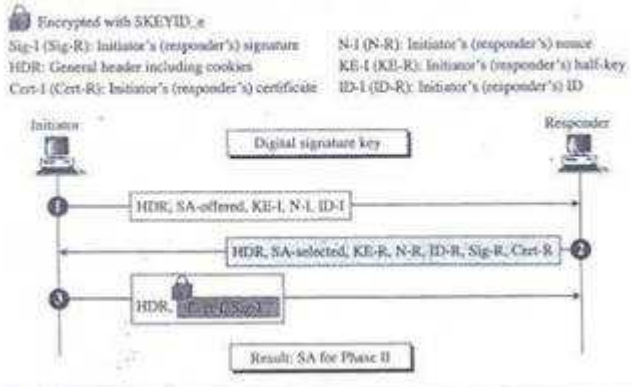
Fig. 18.25 *Aggressive mode, original public-key method*

Figure 18.26 *Aggressive mode, revised public-key method*



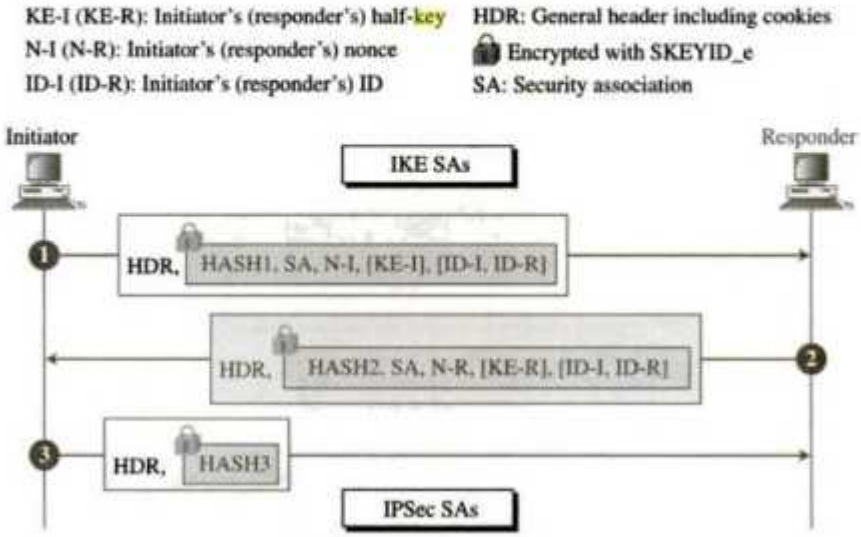
Digital signature method:

Figure 18.27 *Aggressive mode, digital signature method*



After SAs have been created in either the main mode or the aggressive mode, phase II can be started. There is only one mode defined for phase II so far, the *quick mode*. This mode is under the supervision of the IKE SAs created by phase I. However, each quick-mode method can follow any main or aggressive mode.

The quick mode uses IKE SAs to create IPSec SAs (or SAs for any other protocol). Figure 18.28 shows the messages exchanged during the quick mode.



In phase II, either party can be the initiator. That is, the initiator of phase II can be the initiator of phase I or the responder of phase I.

The initiator sends the first message, which includes the keyed-HMAC HASH1 (explained later), the entire SA created in phase I, a new nonce (N-I), an optional new Diffie-Hellman half-key (KE-I), and the optional IDs of both parties. The second message is similar, but carries the keyed-HMAC HASH2, the responder nonce (N-R), and, if present, the Diffie-Hellman half-key created by the responder. The third message contains only the keyed-HMAC HASH3.

The messages are authenticated using three keyed-HMACs: HASH1, HASH2, and HASH3. These are calculated as follows:

$$\text{HASH1} = \textit{prf}(\text{SKEYID\_d}, \text{MsgID} \mid \text{SA} \mid \text{N-I})$$

$$\text{HASH2} = \textit{prf}(\text{SKEYID\_d}, \text{MsgID} \mid \text{SA} \mid \text{N-R})$$

$$\text{HASH3} = \textit{prf}(\text{SKEYID\_d}, 0 \mid \text{MsgID} \mid \text{SA} \mid \text{N-I} \mid \text{N-R})$$

Each HMAC includes the message ID (MsgID) used in the header of ISAKMP headers. This allows multiplexing in phase II. The inclusion of MsgID prevents simultaneous creations of phase II from bumping into each other.

All three messages are encrypted for confidentiality using the SKEYID_e created during phase I.

Perfect Forward Security (PFS) After establishing an IKE SA and calculating SKEYID_d in phase I, all keys for the quick mode are derived from SKEYID_d. Since multiple phase IIs can be derived from a single phase I, phase II security is at risk if the intruder has access to SKEYID_d. To prevent this from happening, IKE allows **Perfect Forward Security (PFS)** as an option. In this option, an additional Diffie-Hellman half-key is exchanged and the resulting shared key (g^{ir}) is used in the calculation of key material (see the next section) for IPSec. PFS is effective if the Diffie-Hellman key is immediately deleted after the calculation of the key material for each quick mode.

Key Materials After the exchanges in phase II, an SA for IPSec is created including the key material, K, that can be used in IPSec. The value is derived as:

$$K = \textit{prf}(\text{SKEYID\_d}, \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R}) \qquad \text{(without PFS)}$$
$$K = \textit{prf}(\text{SKEYID\_d}, g^{ir} \mid \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R}) \qquad \text{(with PFS)}$$

If the length of K is too short for the particular cipher selected, a sequence of keys is created, each key is derived from the previous one, and the keys are concatenated to make a longer key. We show the case without PFS; we need to add g^{ir} for the case with PFS.

The key material created is unidirectional; each party creates different key material because the SPI used in each direction is different.

$$K_1 = \textit{prf}(\text{SKEYID\_d}, \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$$
$$K_2 = \textit{prf}(\text{SKEYID\_d}, K_1 \mid \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$$
$$K_3 = \textit{prf}(\text{SKEYID\_d}, K_2 \mid \text{protocol} \mid \text{SPI} \mid \text{N-I} \mid \text{N-R})$$
$$\ldots$$
$$K = K_1 \mid K_2 \mid K_3 \mid \ldots$$

The key material created after phase II is unidirectional; there is one key for each direction.

18.5.7 SA Algorithms

Before leaving this section, let us give the algorithms that are negotiated during the first two IKE exchanges.

Diffie-Hellman Groups The first negotiation involves the Diffie-Hellman group used for exchanging half-keys. Five groups have been defined, as shown in Table 18.3.

Table 18.3 Diffie-Hellman groups

Value	*Description*
1	Modular exponentiation group with a 768-bit modulus
2	Modular exponentiation group with a 1024-bit modulus
3	Elliptic curve group with a 155-bit field size
4	Elliptic curve group with a 185-bit field size
5	Modular exponentiation group with a 1680-bit modulus

Hash Algorithms The hash algorithms that are used for authentication are shown in Table 18.4.

Table 18.4 *Hash algorithms*

Value	*Description*
1	MD5
2	SHA
3	Tiger
4	SHA2-256
5	SHA2-384
6	SHA2-512

Encryption Algorithms The encryption algorithms that are used for confidentiality are shown in Table 18.5. All of these are normally used in CBC mode.

Value	*Description*
1	DES
2	IDEA
3	Blowfish
4	RC5
5	3DES
6	CAST
7	AES

- **ISAKMP:**

The ISAKMP protocol is designed to carry messages for the IKE exchange.

18.6.1 General Header

The format of the general header is shown in Fig. 18.29.

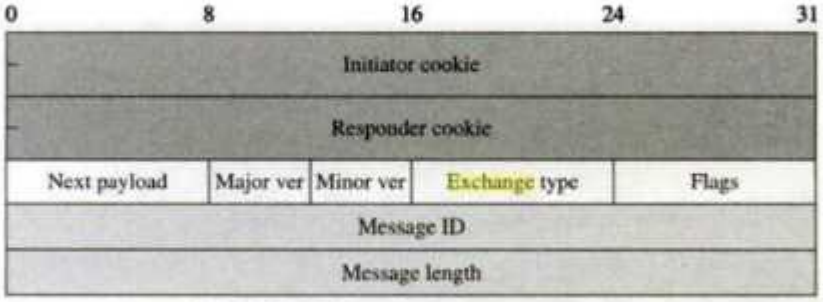


Fig. 18.29 *ISAKMP general header*

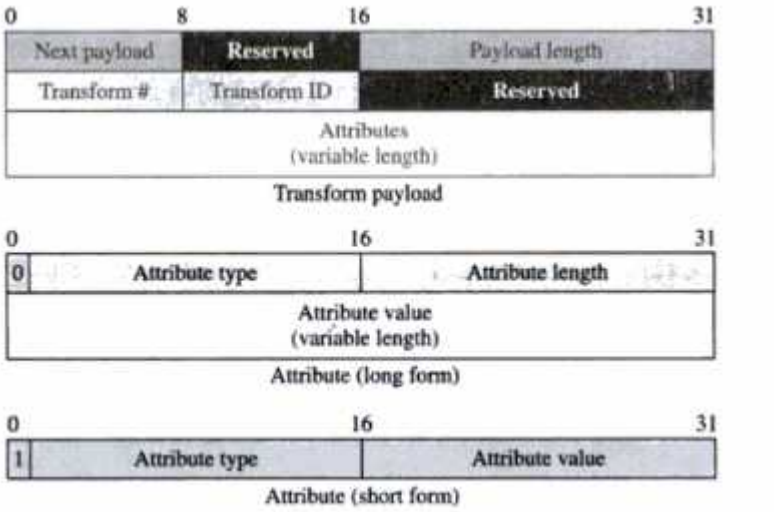
- **Initiator cookie.** This 32-bit field defines the cookie of the entity that initiates the SA establishment, SA notification, or SA deletion.
- **Responder cookie.** This 32-bit field defines the cookie of the responding entity. The value of this field is 0 when the initiator sends the first message.
- **Next payload.** This 8-bit field defines the type of payload that immediately follows the header. We discuss the different types of payload in the next section.

Payloads:

The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **Proposal #.** The initiator defines a number for the proposal so that the responder can refer to it. Note that an SA payload can include several proposal payloads. If all of the proposals belong to the same set of protocols, the proposal number must be the same for each protocol in the set. Otherwise, the proposals must have different numbers.
- **Protocol ID.** This 8-bit field defines the protocol for the negotiation. For example, IKE phase1 = 0, ESP = 1, AH = 2, etc.
- **SPI size.** This 8-bit field defines the size of the SPI in bytes.
- **Number of Transforms.** This 8-bit field defines the number of transform payloads that will follow this proposal payload.
- **SPI.** This variable-length field is the actual SPI. Note that if the SPI does not fill the 32-bit space, no padding is added.

Transform Payload The *transform payload* actually carries attributes of the SA negotiation. Figure 18.33 shows the format of the transform payload.

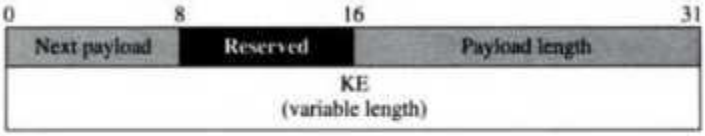


The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **Transform #.** This 8-bit field defines the transform number. If there is more than one transform payload in a proposal payload, then each must have its own number.
- **Transform ID.** This 8-bit field defines the identity of the payload.
- **Attributes.** Each transform payload can carry several attributes. Each attribute itself can have three or two subfields (see Fig. 18.33). The *attribute type* subfield defines the type of attribute as defined in the DOI. The *attribute length* subfield, if present, defines the length of the attribute value. The *attribute value* field is two bytes in the short form or of variable-length in the long form.

Key-Exchange Payload The *key exchange payload* is used in those exchanges that need to send preliminary keys that are used for creating session keys. For example, it can be used to send a Diffie-Hellman half-key. Figure 18.34 shows the format of the key-exchange payload.

Figure 18.34 *Key-exchange payload*



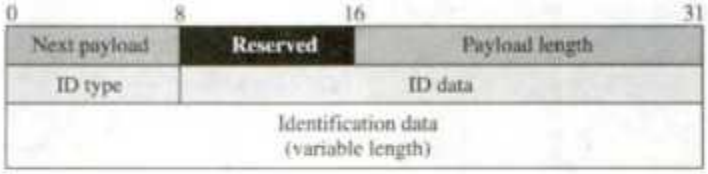
The fields in the generic header have been discussed. The description of the KE field follows:

- **KE.** This variable-length field carries the data needed for creating the session key.

Identification Payload

The *identification payload* allows entities to send their identifications to each other. Figure 18.35 shows the format of the identification payload.

Figure 18.35 *Identification payload*



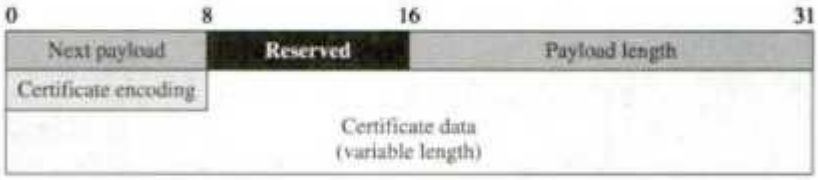
The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **ID type.** This 8-bit field is DOI specific and defines the type of ID being used.
- **ID data.** This 24-bit field is usually set to 0.
- **Identification data.** The actual identity of each entity is carried in this variable-length field.

Certification Payload:

Anytime during the exchange, an entity can send its certification (for public-encryption/decryption keys or signature keys). Although the inclusion of the *certification payload* in an exchange is normally optional, it needs to be included if there is no secure directory available to distribute the certificates. Figure 18.36 shows the format of the certification payload.

Figure 18.36 *Certification payload*



The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **Certificate encoding.** This 8-bit field defines the encoding (type) of the certificate. Table 18.7 shows the types defined so far.
- **Certificate data.** This variable-length field carries the actual value of the certificate. Note that the previous field implicitly defines the size of this field.

Table 18.7 *Certification types*

Value	*Type*
0	None
1	Wrapped X.509 Certificate
2	PGP Certificate
3	DNS Signed Key
4	X.509 Certificate—Signature
5	X.509 Certificate—Key Exchange
6	Kerberos Tokens
7	Certification Revocation List
8	Authority Revocation List
9	SPKI Certificate
10	X.509 Certificate—Attribute

Certificate Request Payload Each entity can explicitly request a certificate from the other entity using the *certificate request payload*. Figure 18.37 shows the format of this payload.

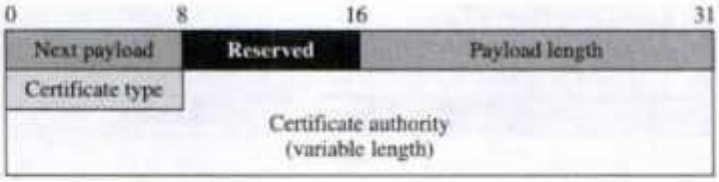


Fig. 18.37 *Certification request payload*

The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **Certificate type.** This 8-bit field defines the type of certificate as previously defined in the certificate payload.
- **Certificate authority.** This is a variable-length field that defines the authority for the type of certificate issued.

Hash Payload The *hash payload* contains data generated by the hash function as described in the IKE exchanges. The hash data guarantee the integrity of the message or part of the ISAKMP states. Figure 18.38 shows the format of the hash payload.

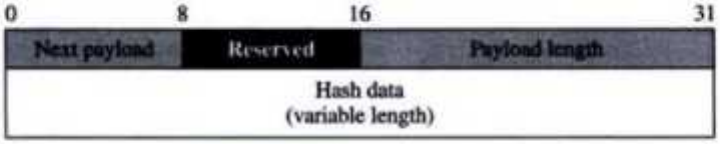


Fig. 18.38 *Hash payload*

The fields in the generic header have been discussed. The description of the last field follows:

- **Hash data.** This variable-length field carries the hash data generated by applying the hash function to the message or part of the ISAKMP states.

Signature Payload The *signature payload* contains data generated by applying the digital signature procedure over some part of the message or ISAKMP state. Figure 18.39 shows the format of the signature payload.

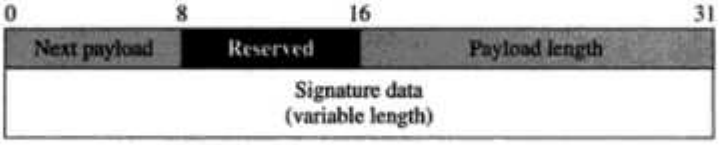


Fig. 18.39 *Signature payload*

The fields in the generic header have been discussed. The description of the last field follows:

- **Signature.** This variable-length field carries the digest resulting from applying the signature over part of the message or ISAKMP state.

Nonce Payload The *nonce payload* contains random data used as a nonce to assure liveliness of the message and to prevent a replay attack. Figure 18.40 shows the format of the nonce payload.

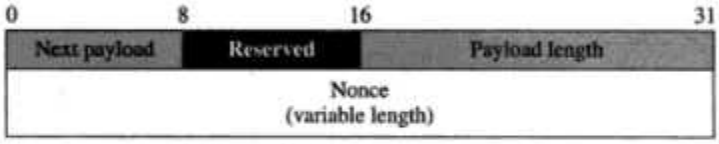


Fig. 18.40 *Nonce payload*

The fields in the generic header have been discussed. The description of the last field follows:

- **Nonce.** This is a variable-length field carrying the value of the nonce.

Notification Payload During the negotiation process, sometimes a party needs to inform the other party of the status or errors. The *notification payload* is designed for these two purposes. Figure 18.41 shows the format of the notification payload.

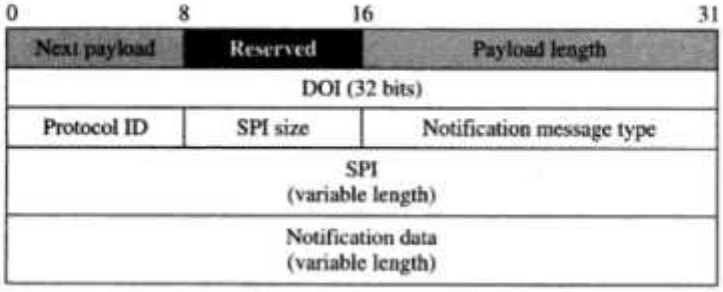


Fig. 18.41 *Notification payload*

The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **DOI.** This 32-bit field is the same as that defined for the Security Association payload.
- **Protocol ID.** This 8-bit field is the same as that defined for the proposal payload.
- **SPI size.** This 8-bit field is the same as that defined for the proposal payload.
- **Notification message type.** This 16-bit field specifies the status or the type of error that is to be reported. Table 18.8 gives a brief description of these types.
- **SPI.** This variable-length field is the same as that defined for the proposal payload.
- **Notification data.** This variable-length field can carry extra textual information about the status or errors. The types of errors are listed in Table 18.8. The values 31 to 8191 are for future use and the values 8192 to 16383 are for private use.

Table 18.8 *Notification types*

Value	*Description*	*Value*	*Description*
1	INVALID-PAYLOAD-TYPE	16	PAYLOAD-MALFORMED
2	DOI-NOT-SUPPORTED	17	INVALID-KEY-INFORMATION
3	SITUATION-NOT-SUPPORTED	18	INVALID-ID-INFORMATION
4	INVALID-COOKIE	19	INVALID-CERT-ENCODING
5	INVALID-MAJOR-VERSION	20	INVALID-CERTIFICATE
6	INVALID-MINOR-VERSION	21	CERT-TYPE-UNSUPPORTED
7	INVALID-EXCHANGE-TYPE	22	INVALID-CERT-AUTHORITY
8	INVALID-FLAGS	23	INVALID-HASH-INFORMATION
9	INVALID-MESSAGE-ID	24	AUTHENTICATION-FAILED
10	INVALID-PROTOCOL-ID	25	INVALID-SIGNATURE
11	INVALID-SPI	26	ADDRESS-NOTIFICATION
12	INVALID-TRANSFORM-ID	27	NOTIFY-SA-LIFETIME
13	ATTRIBUTE-NOT-SUPPORTED	28	CERTIFICATE-UNAVAILABLE
14	NO-PROPOSAL-CHOSEN	29	UNSUPPORTED EXCHANGE-TYPE
15	BAD-PROPOSAL-SYNTAX	30	UNEQUAL-PAYLOAD-LENGTHS

Table 18.9 is a list of status notifications. Values from 16385 to 24575 and 40960 to 65535 are reserved for future use. Values from 32768 to 40959 are for private use.

Table 18.9 *Status notification values*

Value	*Description*
16384	CONNECTED
24576-32767	DOI-specific codes

Delete Payload The delete payload is used by an entity that has deleted one or more SAs and needs to inform the peer that these SAs are no longer supported. Figure 18.42 shows the format of the delete payload.

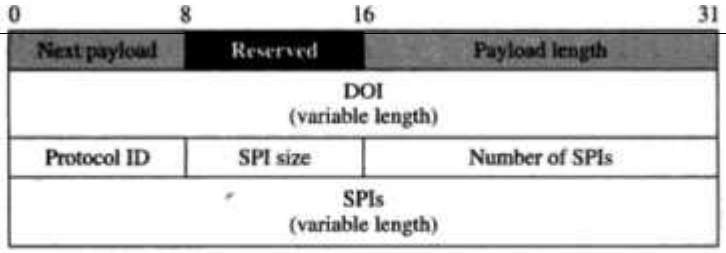


Fig. 18.42 *Delete payload*

The fields in the generic header have been discussed. The descriptions of the other fields follow:

- **DOI.** This 32-bit field is the same as that defined for the Security Association payload.
- **Protocol ID.** This 8-bit field is the same as that defined for the proposal payload.
- **SPI size.** This 8-bit field is the same as that defined for the proposal payload.
- **Number of SPIs.** This 16-bit field defines the number of SPIs. One delete payload can report the deletion of several SAs.
- **SPIs.** This variable-length field defines the SPIs of the deleted SAs.

Vendor Payload ISAKMP allows the exchange of information particular to a specific vendor. Figure 18.43 shows the format of the *vendor payload.*

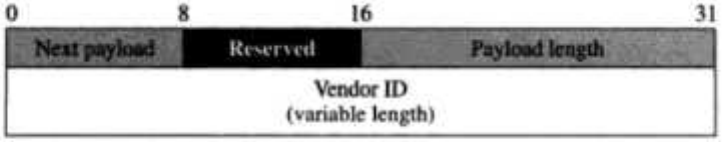


Fig. 18.43 *Vendor payload*

The fields in the generic header have been discussed. The description of the last field follows:

- **Vendor ID.** This variable-length field defines the constant used by the vendor.

Topic 2:

- **System security:**

Description of the system:

- A system is a vague entity that comprises the totality of the computing and the communication environment over which the developers have some control.
- A system boundary demarcates between the protected and unprotected components of a system.
- It defines the interface between the system and the outside world.
  The components inside the system can also be divided into two distinct categories
  1. **Security Relevant:** These components are crucial to the security.A malfunction or penetration in these components can lead to security violations. The Operating system (OS) and the computer hardware its examples.
  2. **Others:** These are the objects that the system controls and protects.

USERS,TRUST AND TRUSTED SYSTEMS:

- A user is a person whose information the system protects and whose access to information is controlled by the system.
- A user is in general trusted to keep his secret, often in the form of a password, confidential from other users who do not have access to the trusted user's documents.
- In order to protect against such a threat, a trusted user should be warned when he accidentally gives away valuable information.
- However, if a user betrays this trust intentionally, then system security cannot handle such a scenario.
- The system identifies a user through a unique identifier(ID) which is public information like name or account number.
- The identifier must be unique and unforgettable.
- The act of associating a user with a unique identifier is called authentication.
- The identification number is used by the system to associate a process (a running program) with a user.
- Trust in systems is built using the techniques of identification and authentication.
- Trusted system enforces a given security policy based on this trust.
- Trusted systems classify programs based on the level of trust on them.
- Trusted programs are responsible for the security of system.

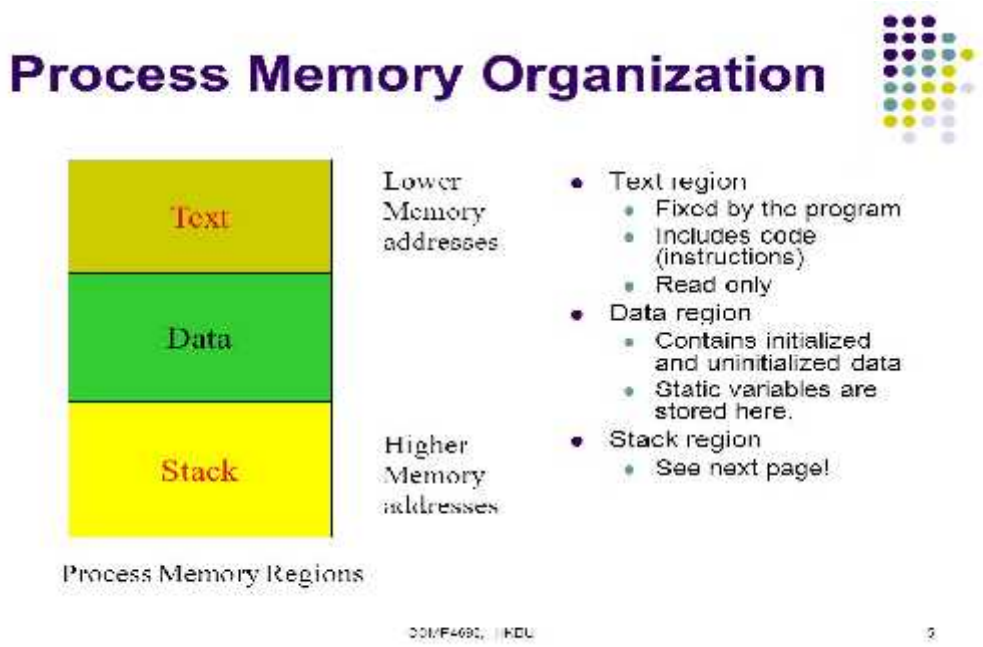
BUFFER OVERFLOW AND MALICIOUS SOFTWARE

- Buffer overflow is a commonly known mistake that exist in some C implementations.
- These classes of bugs are extremely dangerous, as they write past the end or array and hence corrupt the process stack.
- Often they change the return address of a process after a function call to a secret memory location where a malicious code is planted.
- A buffer, often visualized as an array is a contiguous space of related variables of the same data type.
- In C or C++ there are no automatic checks on the buffer, which means a user may write past a buffer. This phenomenon is known as **buffer overflow**.
- Arrays, like all variables in C, can be either static or dynamic. Static variables are allocated at load time on data segment.
- Dynamic variables are allocated at runtime on stack.
  For example, consider the following C code snippet as a simple example of buffer overflow.

```
int main()
{
int buffer[10];
buffer[20]=5;
}
```

  Compilers will compile the above program without any errors. This seemingly simple mistake can be used by the authors of malicious software to trigger their codes and thus compromise security.
    - A process, which is a program in execution, divided into three regions: text, data, stack.

- The text region is decided by the program and is made of binary instructions and read-only data.
- The initialized and uninitialized data region stores static variables.
- If the available memory is rendered unavailable , the process is blocked and rescheduled with a larger memory space between the text and stack segments.



- A stack is a contiguous block of memory containing data,which grows from higher memory address to lower ones.
- A stack of object has the property of LIFO (Last In First Out) which means that the last object placed on the stack is the first object to be removed.
- Two of the most important stack operations are the PUSH and POP.
- PUSH adds an element at the top of the stack, while POP removes the last element from the stack.
- This removes the last element from the stack.
- High level languages use stacks to implement procedure or function calls.
- After the call of a function, the program flow is changed towards the function being called.
- But after the function finishes its task , the control returns to the instruction immediately after the call function.
- The parameters of the function call, the return address and the local variables used in the function are dynamically located in the stack.
- A register called the stack pointer(sp) points to the top of the stack.
- The bottom of the stack is a fixed address, while the sp is updated by the kernel at run time to accommodate for extra space required by the local variables of the function.
- In general architectures like Intel,SPARC and MIPS processors, the stack grows down, that is from higher memory address to lower memory address.

Consider the following C code:

```
Void function(int a, int b, int c)  {
char buffer1[5];
char buffer2[10];
}
void main()
{
function(1,2,3);
}
```

Example 1:

```
void function(int a,int b,int c) {
char buffer1[5];
char buffer2[10];
}
void main(){
function(1,2,3);
}
```

The corresponding assembly code to call the function is described next.The parameters immediate values 1,2,3 are saved in the stack.In the assembly code, $ means immediate value and (%esp) indicates the address value of the register stack pointer, esp. The number before the parenthesis indicates the offset.

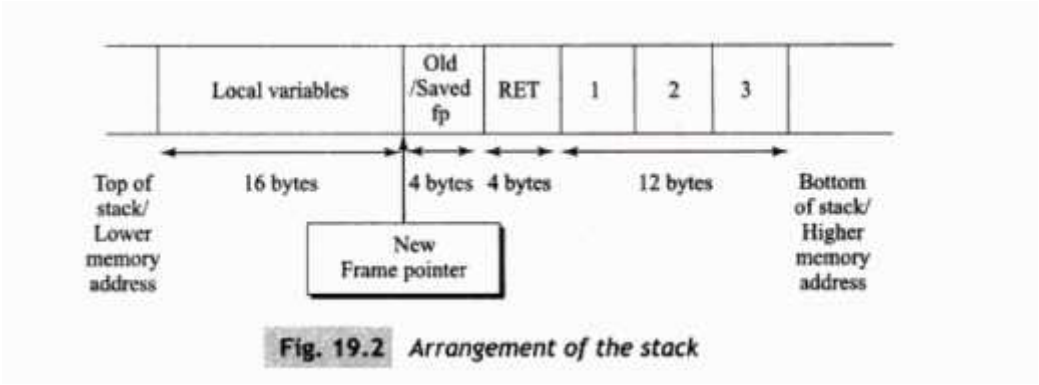


Fig. 19.2 *Arrangement of the stack*

The main passes the character string large_string to the function. The function calls the *strcpy*( ) function to overwrite the string buffer which is of size only 16 bytes. This causes an overflow in the buffer, and the ascii value of the letter 'A' is written onto the buffer, across the saved frame pointer, the return address and *str. The return address is thus overwritten by the value 0x41414141 (41 is the hexadecimal value of 65, which is the ascii value of 'A'). This being an illegal address, outside the process address space leads to a segmentation fault. The state of the stack after the function call is shown in Fig. 19.4.

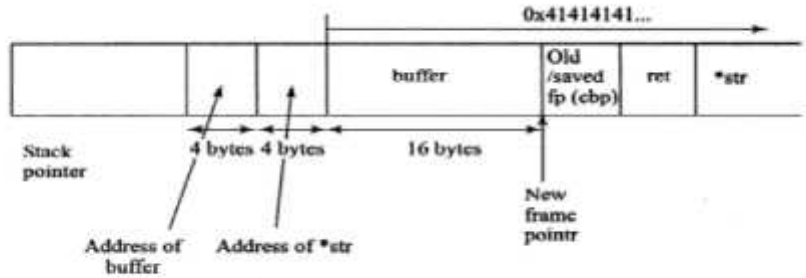


Fig. 19.4 *Stack's configuration by the function*

- **Malicious programs:**

- **Malicious programs** are those programs which try to subvert the expected operation of secured and benign codes.
- The two most commonly known categories of malicious programs are worms and viruses.
- Although, worms also referred to as viruses, it would be appropriate to mention at this point the difference between the two types.

Worms:

- These are programs that can run independently.
- It can propagate a full working version of itself to other machines.
- The term tries to draw an analogy of these programs with that of parasites which live inside a host and uses its resources for its existence.
- The concept of a worm a program that spreads itself anong machines was first mentioned in the classic science fiction, named "the shockwave Rider"by John Brunner in 1975.

Viruses:

- These programs on the other hand, cannot run independently.
- It requires the host program to run and host them.
- These are analogous to the biological viruses, which are not alive themselves,but invade and corrupt host cells.
- The first time when the word VIRUS was used to indicate a program that infects a computer was by David Gerroid in a science fiction, named "When Harlie was One".
- He defined computer virus to be a program that can infect other programs by modifying them to include a copy of itself.

Logic Bombs:

- A logical bomb is a malicious program which has typically two parts: payload and the trigger.
- The payload typically is a malicious piece of code, and the trigger is usually a Boolean logic unit triggers the malicious code when the condition is satisfied.
- A trigger is usually developed using local conditions like date.
- These are inserted stealthily into a big program.
- They often have an objective of causing financial harm.
- Thus, logical bomb creates violation of security when some external events occur.
- An example is , a person fired from his job, implanted logic bomb would delete all the files in the system.
- Such example of logic bombs which fires on a particular time or date, say April 1,2010 is known as time bombs.

Trojans:

- **Trojans** are malicious programs that perform some harmless activities in addition to some malicious activities.

- **A Trojan horse** is a program with some known or documented effect and some undocumented or unexpected effects.
- A classic example is a password grabbing programs.
- A fake login prompt asks the user to enter the password.
- The program then obtains the password and displays an error message showing incorrect password.
- Then the actual login prompt is displayed, thus making the user believe that he typed in the password incorrectly.
- Now he enters the system with correct password, but meanwhile he has given his password.

Spyware:

- Spyware is software that is used to collect information from a computer and transmit it to another computer.
- The information which the spyware exports to the another system could be of the same type as done by the viruses, but the difference is that spywares is that spywares do not replicate.
- Examples of information gathered include the following:
    1. **Passwords**
    2. **Credit card numbers and bank secrets**
    3. **Software license keys**

Worms:

Intrusion Detection System (IDS):

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for harmful activity or policy breaching. Any malicious venture or violation is normally reported either to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system integrates outputs from multiple sources and uses alarm filtering techniques to differentiate malicious activity from false alarms.

Although intrusion detection systems monitor networks for potentially malicious activity, they are also disposed to false alarms. Hence, organizations need to fine-tune their IDS products when they first install them. It means properly setting up the intrusion detection systems to recognize what normal traffic on the network looks like as compared to malicious activity.

Intrusion prevention systems also monitor network packets inbound the system to check the malicious activities involved in it and at once sends the warning notifications.

Classification of Intrusion Detection System:

IDS is basically classified into 2 types:

1. **Network Intrusion Detection System (NIDS):**
   Network intrusion detection systems (NIDS) are set up at a planned point within the

network to examine traffic from all devices on the network. It performs an observation of passing traffic on the entire subnet and matches the traffic that is passed on the subnetts to the collection of known attacks. Once an attack is identified or abnormal behavior is observed, the alert can be sent to the administrator. An example of an NIDS is installing it on the subnet where firewalls are located in order to see if someone is trying crack the firewall.

2. **Host Intrusion Detection System (HIDS):**
   Host intrusion detection systems (HIDS) run on independent hosts or devices on the network. A HIDS monitors the incoming and outgoing packets from the device only and will alert the administrator if suspicious or malicious activity is detected. It takes a snapshot of existing system files and compares it with the previous snapshot. If the analytical system files were edited or deleted, an alert is sent to the administrator to investigate. Anexample of HIDS usage can be seen on mission critical machines, which are not expected to change their layout.

Detection Method of IDS:

1. **Signature-based Method:**
   Signature-based IDS detects the attacks on the basis of the specific patterns such as number of bytes or number of 1's or number of 0's in the network traffic. It also detects on the basis of the already known malicious instruction sequence that is used by the malware. The detected patterns in the IDS are known as signatures.

Signature-based IDS can easily detect the attacks whose pattern (signature) already exists in system but it is quite difficult to detect the new malware attacks as their pattern (signature) is not known.

2. **Anomaly-based Method:**
   Anomaly-based IDS was introduced to detect the unknown malware attacks as new malware are developed rapidly. In anomaly-based IDS there is use of machine learning to create a trustful activity model and anything coming is compared with that model and it is declared suspicious if it is not found in model. Machine learning based method has a better generalized property in comparison to signature-based IDS as these models can be trained according to the applications and hardware configurations.

Comparison of IDS with Firewalls:

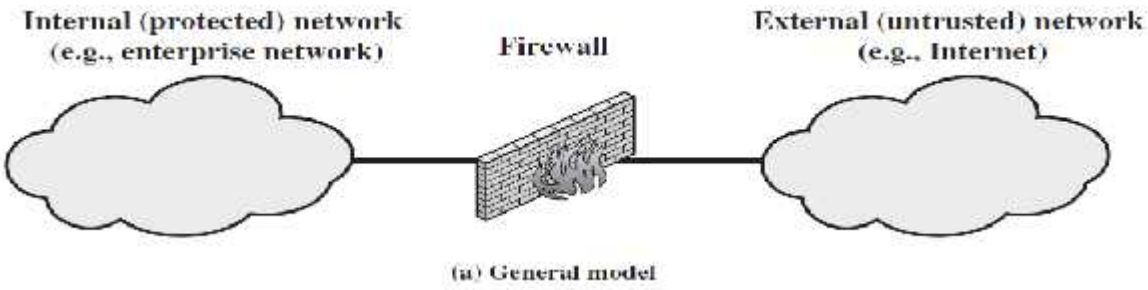
IDS and firewall both are related to the network security but an IDS differs from a firewall as a firewall looks outwardly for intrusions in order to stop them from happening. Firewalls restrict access between networks to prevent intrusion and if an attack is from inside the network it don't signal. An IDS describes a suspected intrusion once it has happened and then signals an alarm.

Firewalls:

Definition: Firewall is software or hardware based network security system that controls the incoming and outgoing network traffic, based on applied rule set.

A fire wall establishes a barrier between a trusted secure internal network and another network.

Firewalls can be an effective means of protecting a local system or network of systems from network-based security threats while at the same time affording access to the outside world via wide area networks and the Internet.



(a) General model

THE NEED FOR FIREWALLS

- ✓ Centralized data processing system, with a central mainframe supporting a number of directly connected terminals
- ✓ Local area networks (LANs) interconnecting PCs and terminals to each other and the mainframe
- ✓ Premises network, consisting of a number of LANs, interconnecting PCs, servers, and perhaps a mainframe or two
- ✓ Enterprise-wide network, consisting of multiple, geographically distributed premises networks interconnected by a private wide area network (WAN)
- ✓ Internet connectivity, in which the various premises networks all hook into the Internet and may or may not also be connected by a private WAN

FIREWALL CHARACTERISTICS

The following design goals for a firewall:

1. All traffic from inside to outside, and vice versa, must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible available in fire wall.
2. Only authorized traffic, as defined by the local security policy, will be allowed to pass. Various types of firewalls are used, which implement various types of security policies, as explained later in this chapter.
3. The firewall itself is immune to penetration. This implies the use of a hardened system with a secured operating system. Trusted computer systems are suitable for hosting a firewall and often required in government applications

Firewall controls:

Firewalls use to control access and enforce the site's security policy. Originally, firewalls focused primarily on service control, but they have since evolved to provide all four:

1. Service control: Determines the types of Internet services that can be accessed, inbound or outbound. The firewall may filter traffic on the basis of IP address, protocol, or port number; may provide proxy software that receives and interprets each service request before passing it on; or may host the server software itself, such as a Web or mail service.

2. Direction control: Determines the direction in which particular service requests may be initiated and allowed to flow through the firewall.

3. User control: Controls access to a service according to which user is attempting to access it. This feature is typically applied to users inside the firewall perimeter (local users). It may also be applied to incoming traffic from external users; the latter requires some form of secure authentication technology, such as is provided in IPsec.

4. Behavior control: Controls how particular services are used. For example, the firewall may filter e-mail to eliminate spam, or it may enable external access to only a portion of the information on a local Web server.

Capabilities of firewall:

The following capabilities are within the scope of a firewall:

1. A firewall defines a single choke point that keeps unauthorized users out of the protected network, prohibits potentially vulnerable services from entering or leaving the network, and provides protection from various kinds of IP spoofing and routing attacks. The use of a single choke point simplifies security management because security capabilities are consolidated on a single system or set of systems.

2. A firewall provides a location for monitoring security-related events. Audits and alarms can be implemented on the firewall system.

3. A firewall is a convenient platform for several Internet functions that are not security related. These include a network address translator, which maps local addresses to Internet addresses, and a network management function that audits or logs Internet usage.

4. A firewall can serve as the platform for IPsec.

Limitations of Firewalls:

Firewalls have their limitations

1. The firewall cannot protect against attacks that bypass the firewall. Internal systems may have dial-out capability to connect to an ISP. An internal LAN may support a modem pool that provides dial-in capability for travelling employees and telecommuters.
2. The firewall may not protect fully against internal threats, such as a disgruntled employee or an employee who unwittingly cooperates with an external attacker.
3. An improperly secured wireless LAN may be accessed from outside the organization. An internal firewall that separates portions of an enterprise network cannot guard against wireless communications between local systems on different sides of the internal firewall.
4. A laptop, PDA, or portable storage device may be used and infected outside the corporate network, and then attached and used internally.

TYPES OF FIREWALLS:

- Packet Filtering Firewall
- Application Level Gateway
- Circuit Level Gateway

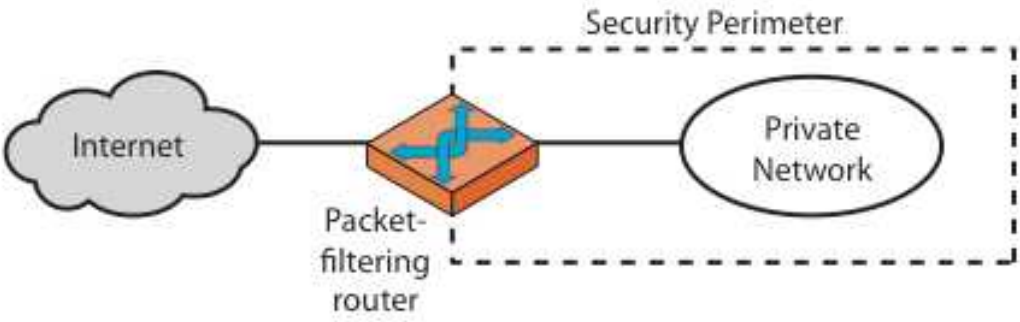
Packet Filtering Firewall:

Packet filtering firewall applies a set of rules to each incoming and outgoing IP packet and then forwards or discards the packet .The firewall is typically configured to filter packets going in both directions (from and to the internal network). Filtering rules are based on information contained in a network packet:

- Source IP address: The IP address of the system that originated the IP packet (e.g., 192.178.1.1)
- Destination IP address: The IP address of the system the IP packet is trying to reach (e.g., 192.168.1.2)
- Source and destination transport-level address: The transport-level (e.g., TCP or UDP) port number, which defines applications such as SNMP or TELNET
- IP protocol field: Defines the transport protocol
- Interface: For a firewall with three or more ports, which interface of the firewall the packet came from or which interface of the firewall the packet is destined for.
- possible default policies

Default = discard: That which is not expressly permitted is prohibited.

Default = forward: That which is not expressly prohibited is permitted.



(a) Packet-filtering router

Table 20.1 Packet-Filtering Examples

A

action	ourhost	port	theirhost	port	comment
block	*	*	SPIGOT	*	we don't trust these people
allow	OUR-GW	25	*	*	connection to our SMTP port

B

action	ourhost	port	theirhost	port	comment
block	*	*	*	*	default

C

action	ourhost	port	theirhost	port	comment
allow	*	*	*	25	connection to their SMTP port

D

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	25		our packets to their SMTP port
allow	*	25	*	*	ACK	their replies

E

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	*		our outgoing calls
allow	*	*	*	*	ACK	replies to our calls
allow	*	*	*	>1024		traffic to nonservers

Attacks on Packet Filters

- IP address spoofing: fake source address to be trusted add filters on router to block
- Source routing attacks: attacker sets a route other than default block source routed packets.
- Tiny fragment attacks: split header info over several tiny packets, either discard or reassemble before check

Application Level Gateway (or Proxy)

•have application specific gateway / proxy

•has full access to Protocol

User requests service from proxy

Proxy validates request as legal

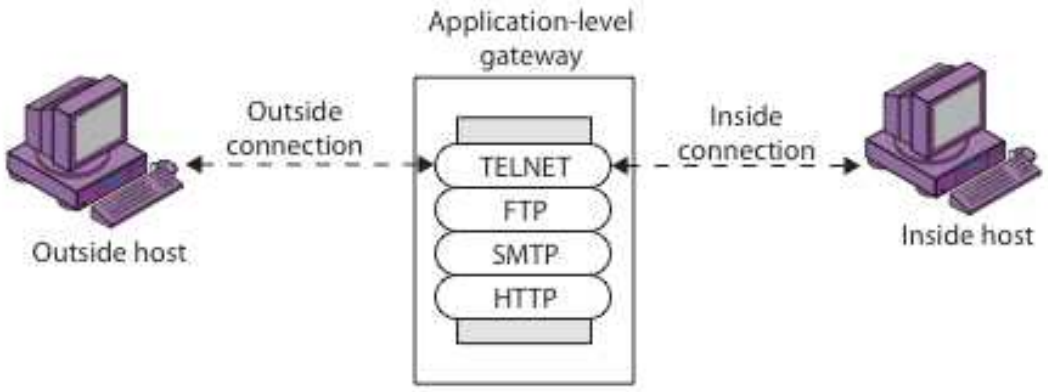
Then actions request and returns result to user

Can log / audit traffic at application level.

•Need separate proxies for each service

Some services naturally support proxying

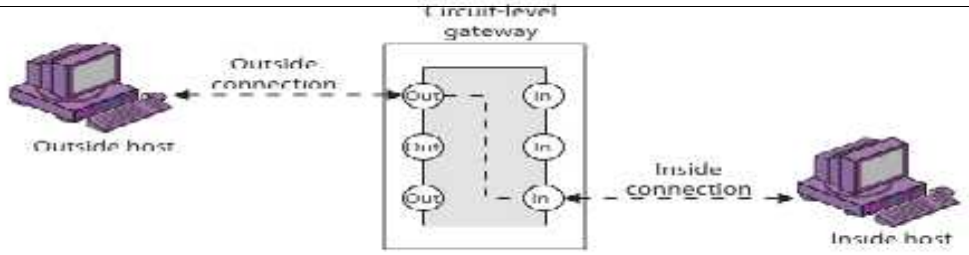
Others are more problematic.



(b) Application-level gateway

Circuit Level Gateway:

- It is a stand a-lone system or it can be a specialized function performed by an application level gate way for certain applications.
- It does not permit end to end TCP connection; this relays two TCP connections, one between itself and a TCP user on an inner host, and one between itself and TCP user on outside host.
- Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the content.



(c) Circuit level gateway

Example:

Circuit level gateway example is implementation of the SOCKS package.

SOCKS:

- This protocol designed to provide a framework for client-server application in both the TCP & UDP domains to conveniently and securely use the services of a Network firewall.
- SOCKS server, which runs on a UNIX based firewall.
- SOCKS client library, which runs on internal hosts protected by the firewall.
- The implementation of the SOCKS protocol typically involves the recompilation or re-linking of TCP-based client applications to use the appropriate encapsulation routines in the SOCKS library.
- The SOCKS service is located on TCP port 1080.