UNIT-2

BLOCK CIPHER PRINCIPLES

Virtually, all symmetric block encryption algorithms in current use are based on a structure referred to as Fiestel block cipher. For that reason, it is important to examine the design principles of the Fiestel cipher. We begin with a comparison of stream cipher with block cipher.

- A stream cipher is one that encrypts a digital data stream one bit or one byte at a time. E.g, vigenere cipher.
- A block cipher is one in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Typically a block size of 64 or 128 bitsis used.

Block cipher principles

most symmetric block ciphers are based on a Feistel Cipher Structure needed since must be able to decrypt ciphertext to recover messages efficiently. block ciphers look like an extremely large substitution

- would need table of 264 entries for a 64-bit block • Instead create from smaller building blocks
- using idea of a product cipher in 1949 Claude Shannon introduced idea of substitution- permutation (S-P) networks called modern substitution-transposition product cipher these form the basis of modern block ciphers
- S-P networks are based on the two primitive cryptographic operations we have seen before:
  - *substitution* (S-box)
  - *permutation* (P-box)
- provide *confusion* and *diffusion* of message
- diffusion – dissipates statistical structure of plaintext over bulk of ciphertext
- confusion – makes relationship between ciphertext and key as complex as possible

The Feistel Cipher

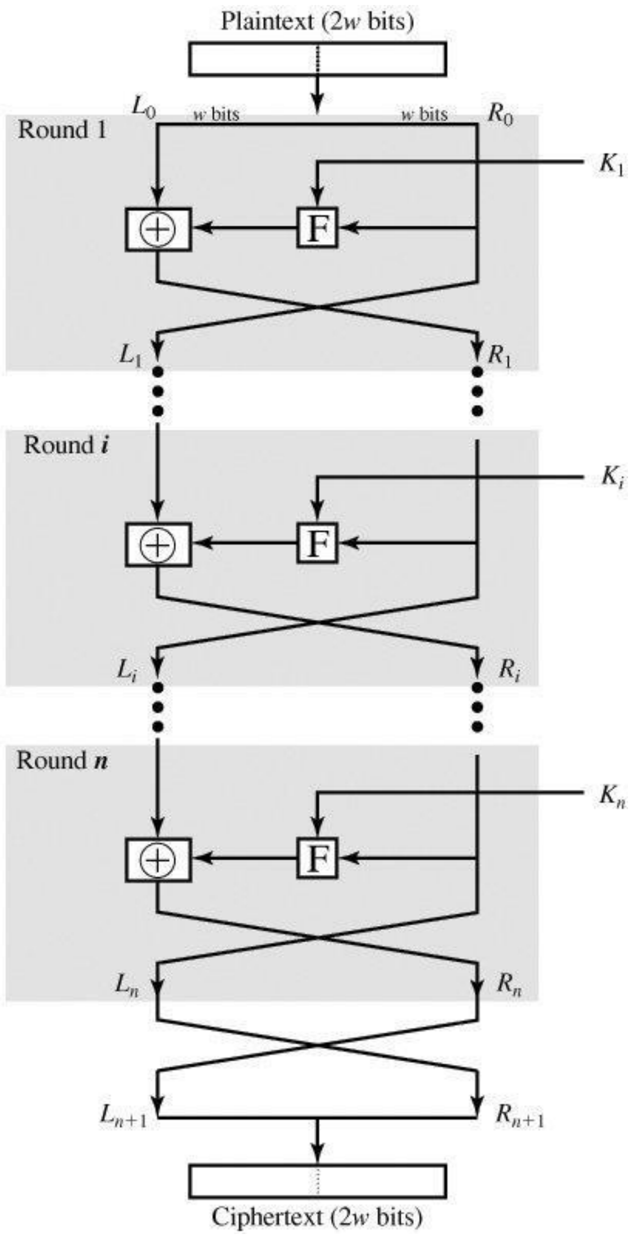
Feistel proposed [FEIS73] that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers in sequencein such a way that the final result or product is cryptographically stronger than any of the component ciphers

Feistel Cipher Structure

- The inputs to the encryption algorithm are a plaintext block of length 2*w* bits and a key *K*.
- The plaintext block is divided into two halves, L_0 and R_0. T

- he two halves of the data pass through *n* rounds of processing and then combine toproduce the ciphertext block.

- Each round *i* has as inputs L_{i-1} and R_{i-1}, derived from the previous round, as well as asubkey K_i, derived from the overall *K*.
- In general, the subkeys K_i are different from *K* and from each other.



Description:

- All rounds have the same structure.

- A **substitution** is performed on the left half ofthe data.

- This is done by applying a *round function* F to the right half of the data and then taking the exclusive-OR of the output of that function andthe left half of the data.

- The round function has the same general structure for each round but is parameterizedby the round subkey K_i.
- Following this substitution, a **permutation** isperformed that consists of the interchange ofthe two halves of the data

The exact realization of a Feistel network depends on the choic and design features:

● **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achievedby greater diffusion Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design.

● **Key size:** Larger key size means greater security but may decrease encryption/decryptionspeed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.

● **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16rounds.

The Data Encryption Standard

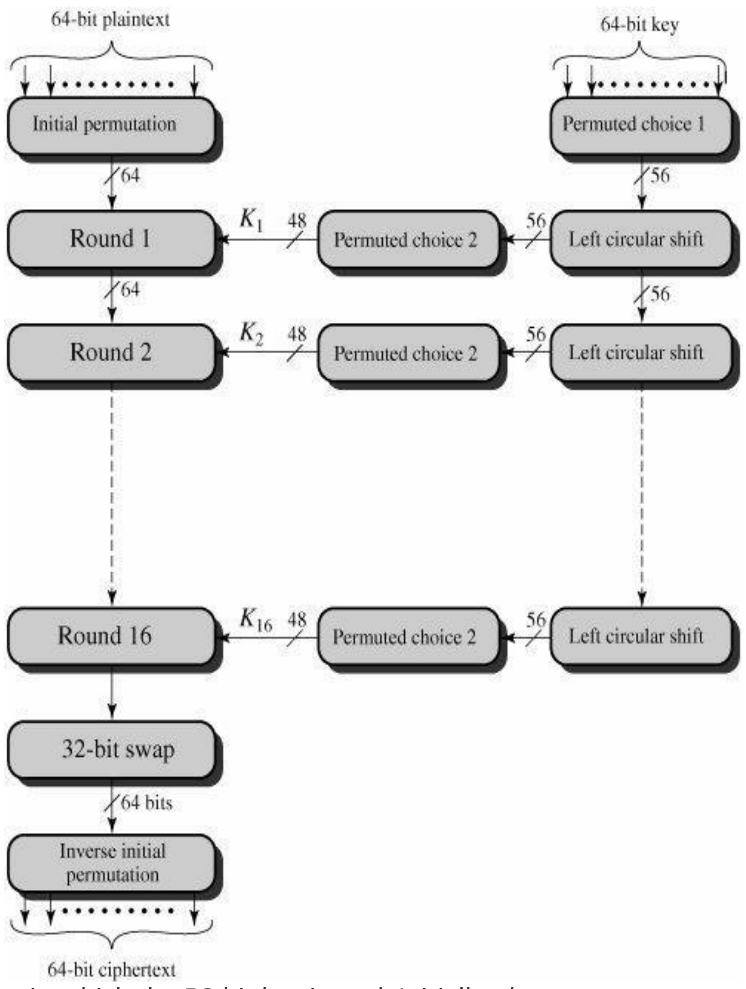
The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standardsand Technology (NIST). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).

.

DES Encryption

DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bitinput in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption

1. Looking at the **left-hand side** of the figure, we can see that the processing of the plaintext proceeds in three phases.
2. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the*permuted input*.
3. This is followed by a phase consisting of 16 rounds of the same function, which involves both permutation and substitution functions.
4. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key.
5. The left and right halves of the output are swapped to produce the **preoutput**.
6. Finally, the preoutput is passed through a permutation (IP-1) that is the inverse ofthe initial permutation function, to produce the 64-bit ciphertext.



1. **The right-hand** portion of Figure 3.4 shows the way in which the 56-bit key isused. Initially, the key is passed through a permutation function.

2. Then, for each of the 16 rounds, a *subkey* (*Ki*) is produced by
3. the combination of a left circular shift and a permutation.

4. The permutation function is the same for each round, but a different subkey isproduced because of the repeated shifts of the key bits.

Initial Permutation

permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which alsoconsists of 64 bits.
To see that these two permutation functions are indeed the inverse of each other, considerthe following 64-bit input *M*:

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}
M_{17}	M_{18}	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}
M_{25}	M_{26}	M_{27}	M_{28}	M_{29}	M_{30}	M_{31}	M_{32}
M_{33}	M_{34}	M_{35}	M_{36}	M_{37}	M_{38}	M_{39}	M_{40}
M_{41}	M_{42}	M_{43}	M_{44}	M_{45}	M_{46}	M_{47}	M_{48}
M_{49}	M_{50}	M_{51}	M_{52}	M_{53}	M_{54}	M_{55}	M_{56}
M_{57}	M_{58}	M_{59}	M_{60}	M_{61}	M_{62}	M_{63}	M_{64}

M_{58}	M_{50}	M_{42}	M_{34}	M_{26}	M_{18}	M_{10}	M_2
M_{60}	M_{52}	M_{44}	M_{36}	M_{28}	M_{20}	M_{12}	M_4
M_{62}	M_{54}	M_{46}	M_{38}	M_{30}	M_{22}	M_{14}	M_6
M_{64}	M_{56}	M_{48}	M_{40}	M_{32}	M_{24}	M_{16}	M_8
M_{57}	M_{49}	M_{41}	M_{33}	M_{25}	M_{17}	M_9	M_1
M_{59}	M_{51}	M_{43}	M_{35}	M_{27}	M_{19}	M_{11}	M_3
M_{61}	M_{53}	M_{45}	M_{37}	M_{29}	M_{21}	M_{13}	M_5
M_{63}	M_{55}	M_{47}	M_{39}	M_{31}	M_{23}	M_{15}	M_7

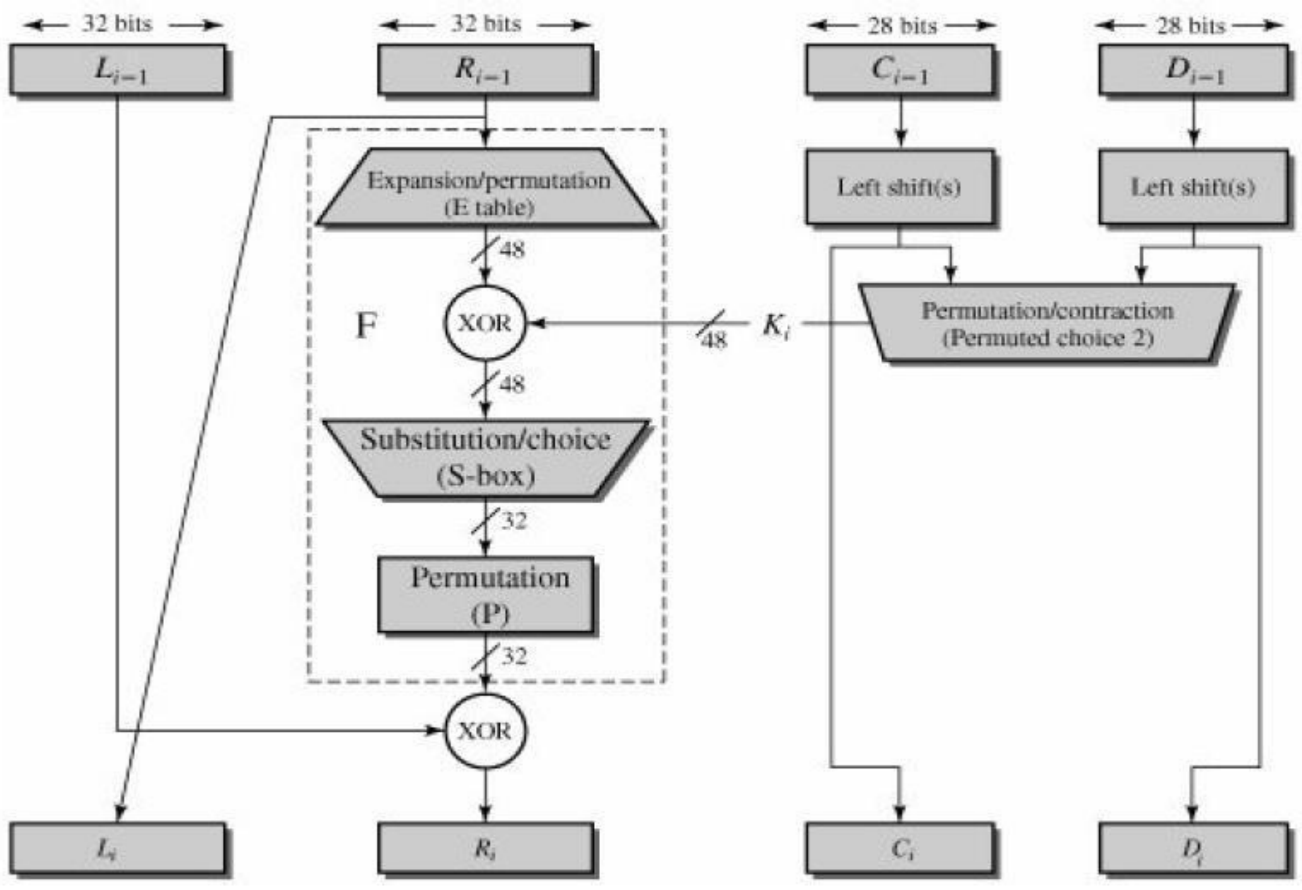
If we then take the inverse permutation **$Y = IP^{-1}(X) = IP^{-1}(IP(M))$,** it can be seen that theoriginal ordering of the bits is restored

Details of Single Round

1. Figure 3.5 shows the internal structure of a single round.
2. Again, begin by focusing on the **left-hand side** of the diagram.
3. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right).
4. As in any classic Feistel cipher, the overall processing at each round can besummarized in the following formulas

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \times F(R_{i-1}, K_i)$$

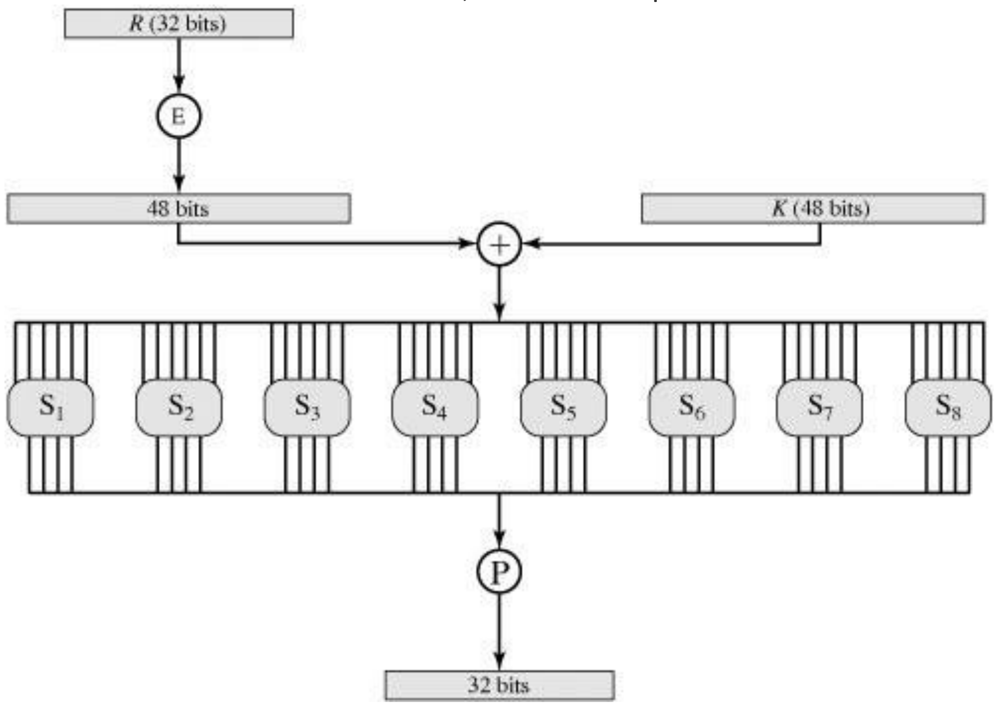
5. The round key K_i is 48 bits
6. The *R* input is 32 bits.
7. This *R* input is first expanded to 48 bits by using a table that defines a permutationplus an expansion that involves duplication of 16 of the *R* bits .
8. The resulting 48 bits are XORed with K_i.



This 48-bit result passes through a substitution function that produces a 32-bit output.

9. The role of the S-boxes in the function F is illustrated in Figure 3.6.
10. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits asinput and produces 4 bits as output.

These transformations are defined, which is interpreted as follows:



1. The first and last bits of the input to box S*i* form a 2-bit binary number to select one offour substitutions defined by the four rows in the table for S*i*.
2. The middle four bits select one of the sixteen columns.
3. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in S1 for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

****

Differential and Linear Cryptanalysis

Differential Cryptanalysis:

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis.

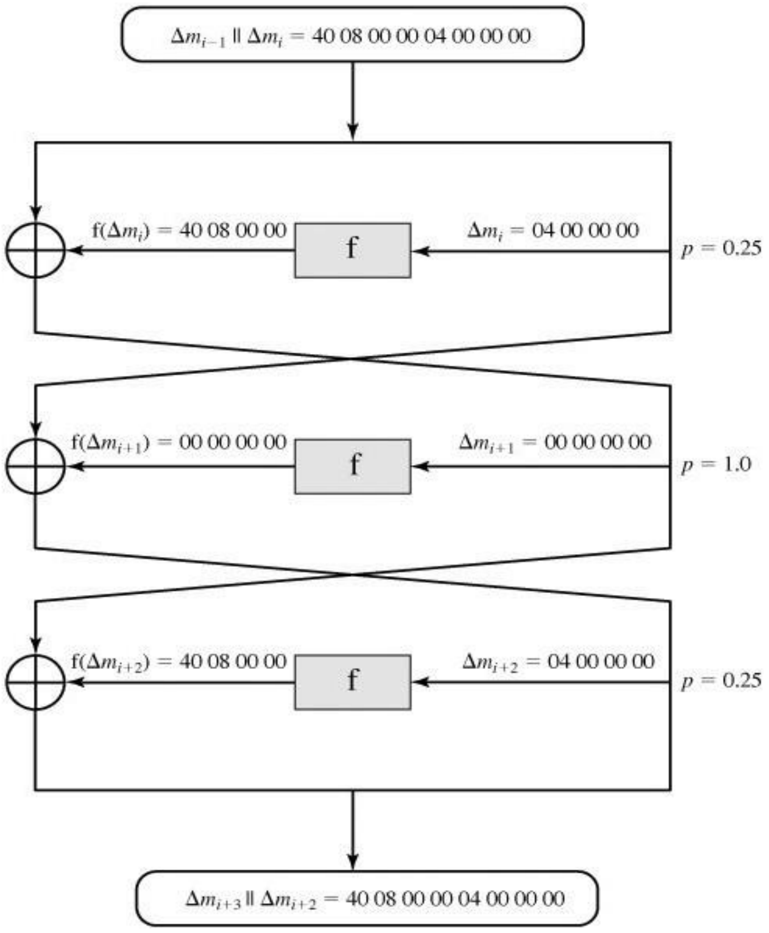
The differential cryptanalysis attack is complex;provides a complete description. The rationalebehind differential cryptanalysis is to observe the behavior of pairs of text blocks evolving along each round of the cipher, instead of observing the evolution of a single text block

We begin with a change in notation for DES. Consider the original plaintext block *m* to consist of two halves m_0, m_1. Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be a function of the left-hand input and the subkey for this round. So, at each round, only one new 32-bit block is created. If we label each new block $m_1(2 \leq i \leq 17)$, then the intermediate message halves are related as follows:

$$m_{i+1} = m_{i-1} \oplus f(m_i, K_i),\ i = 1, 2, \ldots, 16$$

In differential cryptanalysis, we start with two messages, *m* and *m*', with a known XOR difference $\Delta m = m \oplus m'$, and consider the difference between the intermediate message halves: $m_i = m_i \oplus m_i'$ Then we have:

$$\begin{aligned}\Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus f(m_i, K_i)] \oplus [m'_{i-1} \oplus f(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [f(m_i, K_i) \oplus f(m'_i, K_i)]\end{aligned}$$



$\Delta m_{i-1} \| \Delta m_i$ = 40 08 00 00 04 00 00 00
$f(\Delta m_i)$ = 40 08 00 00
f
Δm_i = 04 00 00 00
$p = 0.25$
$f(\Delta m_{i+1})$ = 00 00 00 00
f
Δm_{i+1} = 00 00 00 00
$p = 1.0$
$f(\Delta m_{i+2})$ = 40 08 00 00
f
Δm_{i+2} = 04 00 00 00
$p = 0.25$
$\Delta m_{i+3} \| \Delta m_{i+2}$ = 40 08 00 00 04 00 00 00

Linear Cryptanalysis

We now give a brief summary of the principle on which linear cryptanalysis is based. For acipher with nbit plaintext and ciphertext blocks and an *m*-bit key, let the plaintext block belabeled P[1], ... P[*n*], the cipher text block C[1], ... C[*n*], and the key K[1], ... K[*m*]. Thendefine

$A[i, j, \ldots, k] = A[i] \oplus A[j] \oplus \ldots \oplus A[k]$

The objective of linear cryptanalysis is to find an effective *linear* equation of the form:

$P[\alpha_1, \alpha_2, \ldots, \alpha_a] \oplus C[\beta_1, \beta_2, \ldots, \beta_b] = K[\gamma_1, \gamma_2, \ldots, \gamma_c]$

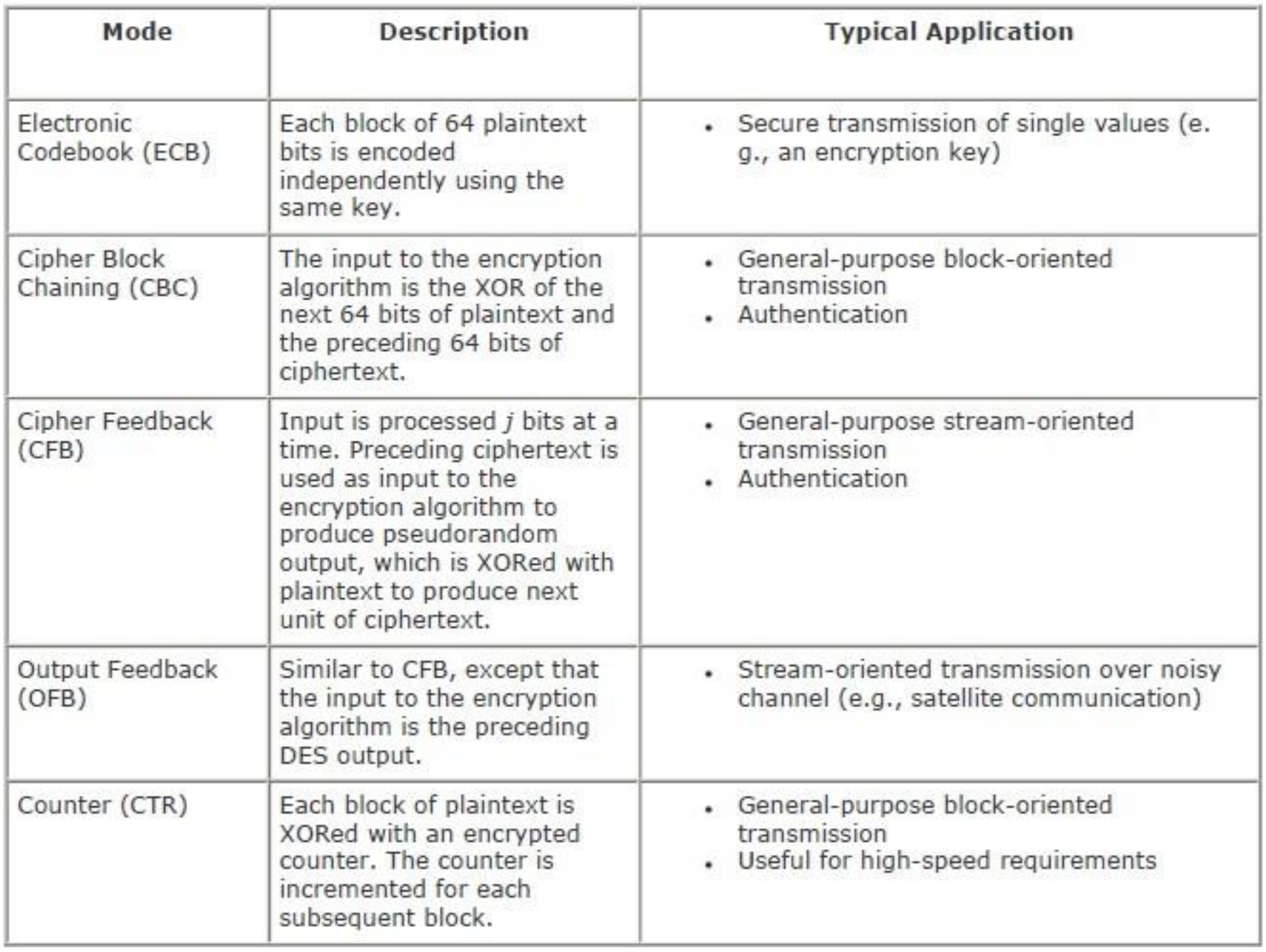
(where x = 0 or 1; $1 \leq a, b \leq n, 1 \leq c \leq m$, and where the α, β and γ terms represent fixed, unique bit locations) that holds with probability $p \neq 0.5$. The further p is from 0.5, the more effective the equation. Once a proposed relation is determined, the procedure is to compute the results of the left-hand side of the preceding equation for a large number of plaintext-ciphertext pairs. If the result is 0 more than half the time, assume $K[\gamma_1, \gamma_2, \ldots, \gamma_c] = 0$. If it is 1 most of the time, assume $K[\gamma_1, \gamma_2, \ldots, \gamma_c] = 1$. This gives us a linear equation on the key bits. Try to get more such relations so that we can solve for the key bits. Because we are dealing with linear equations, the problem can be approached one round of the cipher at a time, with the results combined.

Block Cipher Modes of Operation

A block cipher algorithm is a basic building block for providing data securityThe modes

are summarized in Table 6.1 and described briefly as follows:

Table 6.1. Block Cipher Modes of Operation



Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e. g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed *j* bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements

---

Electronic Codebook Mode:

1. The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the samekey (Figure 6.3).

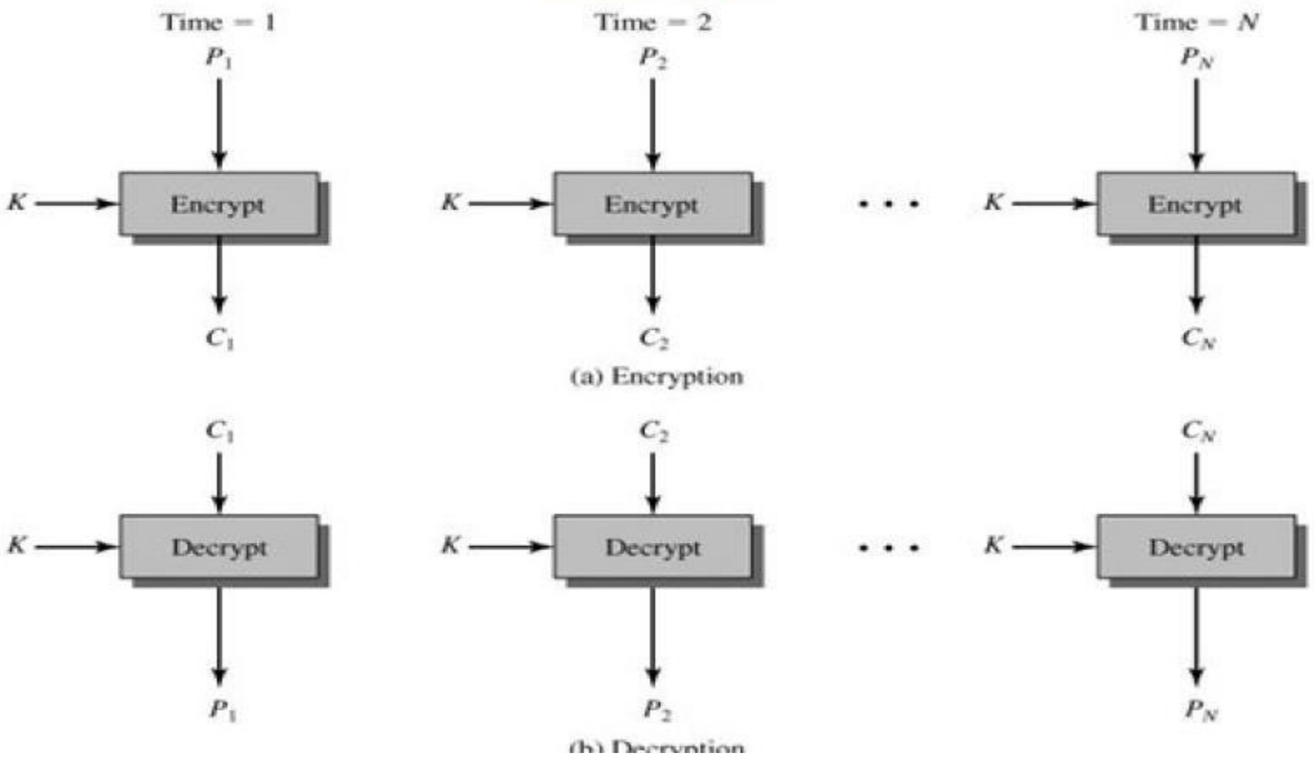
2. The term *codebook* is used because, for a given key, there is a unique ciphertext forevery *b*-bit block of plaintext.

3. Therefore, we can imagine a gigantic codebook in which there is an entry for everypossible *b*-bit plaintext pattern showing its corresponding ciphertext

4. For a message longer than *b* bits, the procedure is simply to break the message into *b*-bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key.

Figure 6.3. Electronic Codebook (ECB) Mode

[View full size image]



5. The ECB method is ideal for a short amount of data, such as an encryption key. Thus,if you want to transmit a DES key securely, ECB is the appropriate mode to use.

6. The most significant characteristic of ECB is that the same *b*-bit block of plaintext, if itappears more than once in the message, always produces the same ciphertext.

7. For lengthy messages, the ECB mode may not be secure

Cipher Block Chaining Mode:

To overcome the security deficiencies of ECB, we would like a technique in which the sameplaintext block, if repeated, produces different ciphertext blocks.

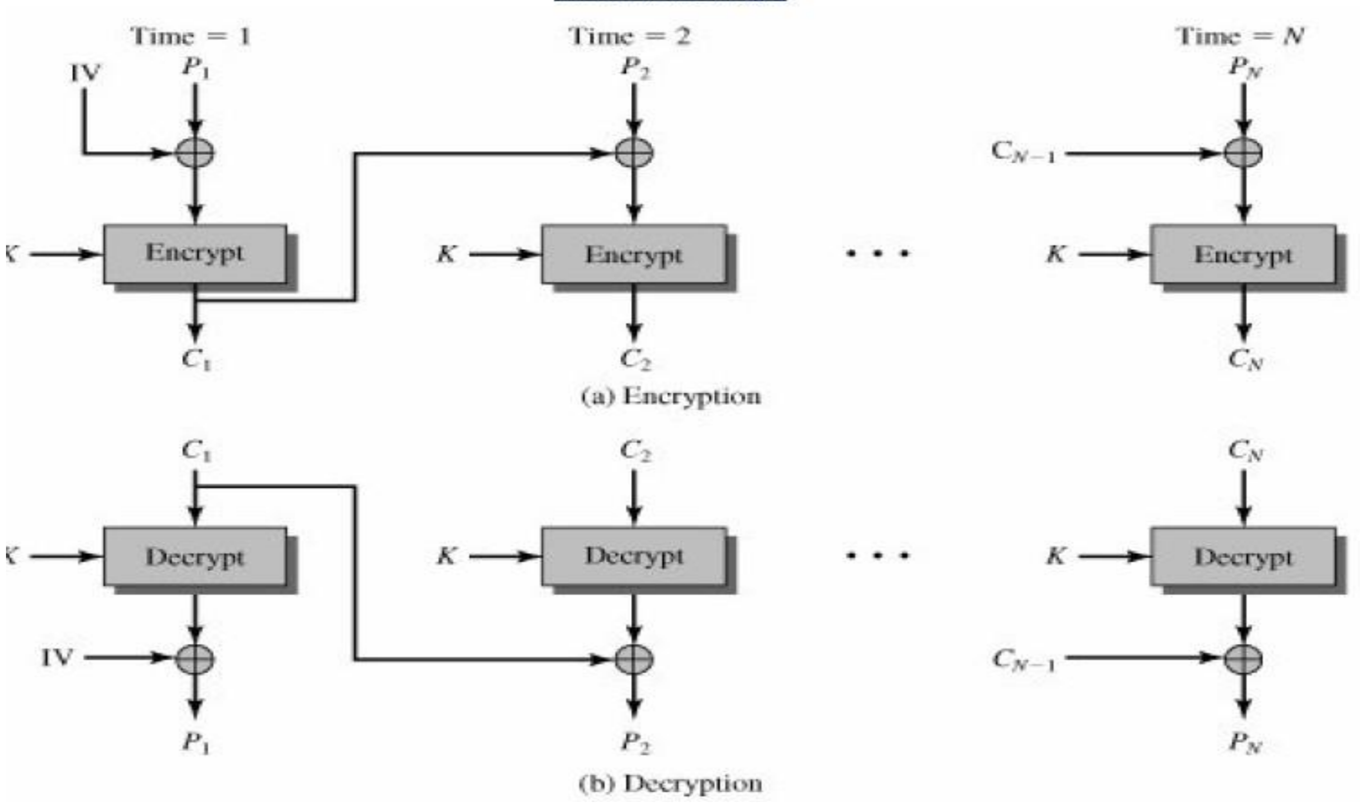
A simple way to satisfy this requirement is the cipher block chaining (CBC) mode

In this scheme, the input to the encryption algorithm is the XOR of the current plaintext blockand the preceding ciphertext block; the same key is used for eachblock.

In effect, we have chained together the processing of the sequence of plaintext blocks. Theinput to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of *b* bits are not exposed.

Figure 6.4. Cipher Block Chaining (CBC) Mode

[View full size image]



$$C_1 = E(K, [IV \oplus P_1])$$

$$P_1 = IV \oplus D(K, C_1)$$

Cipher Feedback Mode :

Figure 6.5 depicts the CFB scheme. In the figure, it is assumed that the unit of transmissionis s bits; a common value is s = 8. As with CBC, the units of plaintext are chained together,so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. In thiscase, rather than units of b bits, the plaintext is divided into ***segments*** of s bits.

Encryption.

1. The input to the encryption function is a b-bit shift register that is initially set to someinitialization vector (IV).

2. The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment of plaintext P_1 to produce the first unit of ciphertext C_1, which is then transmitted. In addition, the contents of the shift register are shifted leftby s bits and C_1 is placed in the rightmost (least significant) s bits of the shift register.

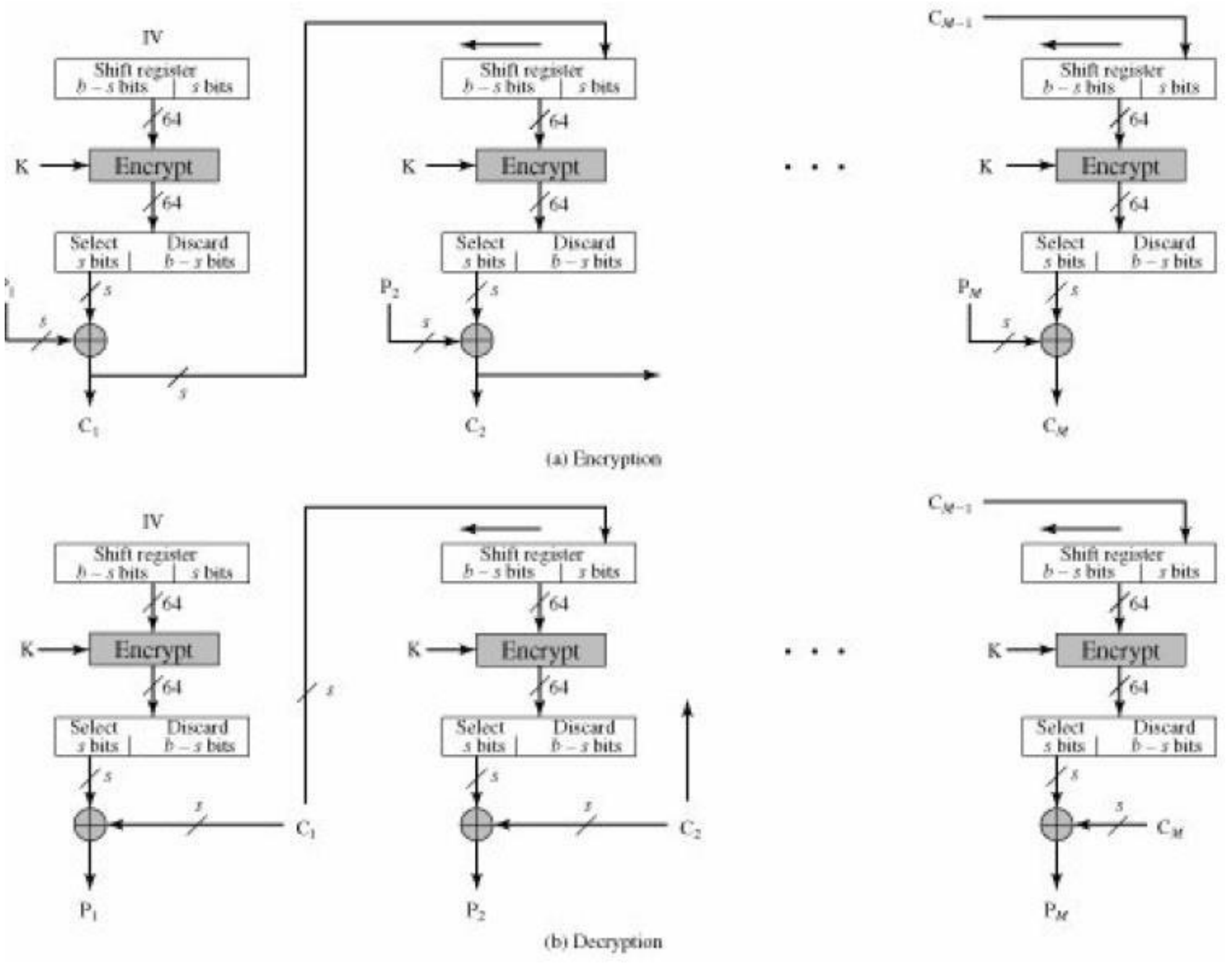
3. This process continues until all plaintext units have been encrypted.

$$C_1 = P_1 \oplus S_s[E(K, IV)]$$

Decryption

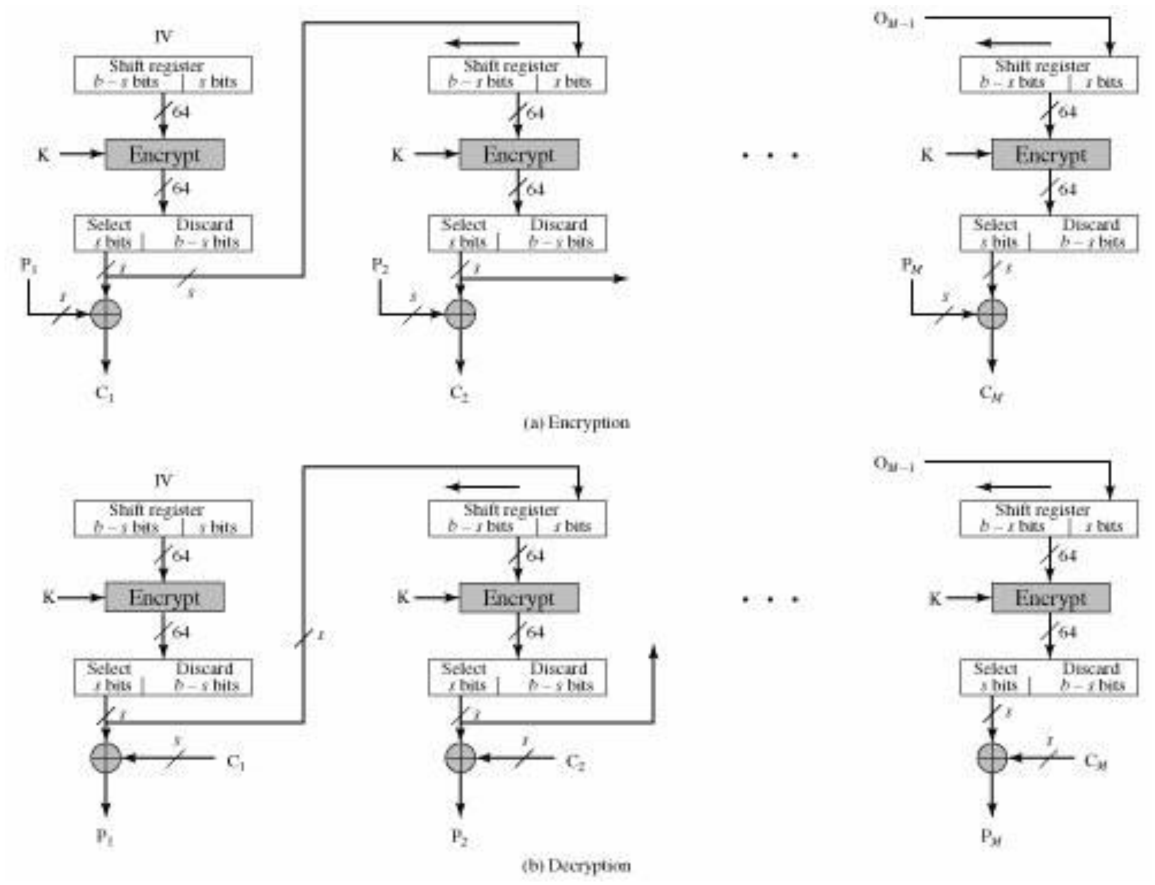
The same scheme is used, except that the received ciphertext unit is XORed with the outputof the encryption function to produce the plaintext unit.

$$P_1 = C_1 \oplus S_s[E(K, IV)]$$



Output Feedback Mode:

The output feedback (OFB) mode is similar in structure to that of CFB, as illustrated in Figure
As can be seen, it is the output of the encryption function that is fed back to the shiftregister in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.



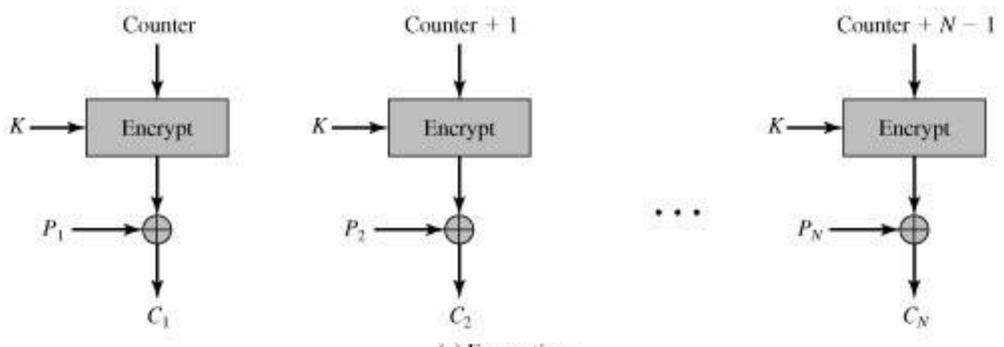
One **advantage** of the OFB method is that bit errors in transmission do not propagate.

The **disadvantage** of OFB is that it is more vulnerable to a message stream modificationattack than is CFB.

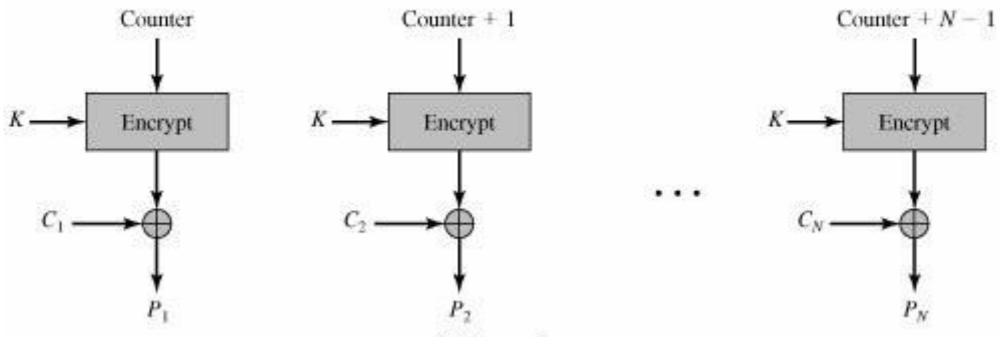
Counter Mode:

Although interest in the counter mode (CTR) has increased recently, with applications to ATM (asynchronous transfer mode) network security and IPSec (IP security).

1. A counter, equal to the plaintext block size is used.
2. Typically, the counter is initialized to some value and then incremented by 1 for eachsubsequent block (modulo 2*b* where *b* is the block size).
3. For **encryption**, the counter is encrypted and then XORed with the plaintext block toproduce the ciphertext block; there is no chaining.
4. For **decryption**, the same sequence of counter values is used, with each encryptedcounter XORed with a ciphertext block to recover the corresponding plaintext block.



(a) Encryption



(b) Decryption

Advantages of CTR mode:

Hardware efficiency :
in CTR mode can be done in parallel on multiple blocks of plaintext or ciphertext.

Software efficiency :
Because of the opportunities for parallel execution.

Preprocessing:
The execution of the underlying encryption algorithm does not depend on input of theplaintext or ciphertext

Random access:
The *i*th block of plaintext or ciphertext can be processed in random-accessFashion

Simplicity:

Unlike ECB and CBC modes, CTR mode requires only the implementation of theencryption algorithm and not the decryption algorithm

Advanced Encryption Standard (AES).

The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six time faster than triple DES.

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

The features of AES are as follows −

- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C and Java

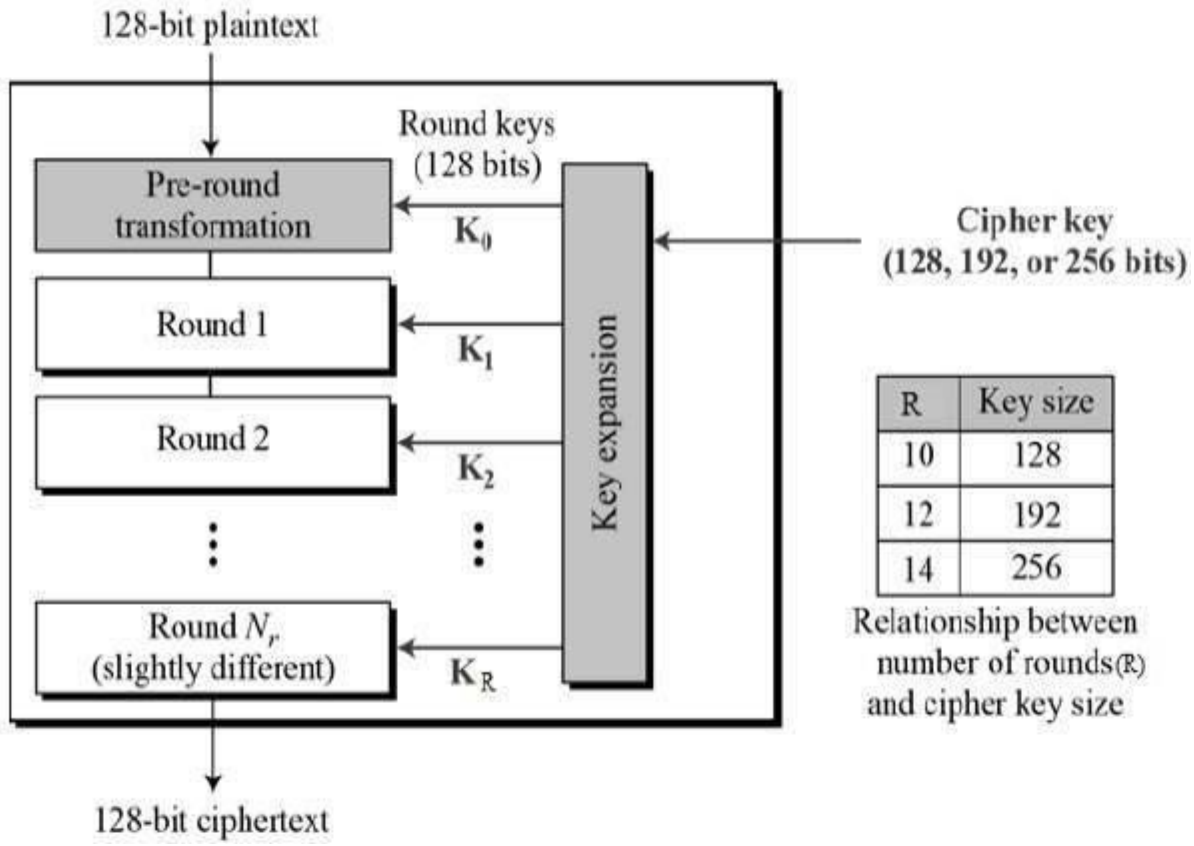
Operation of AES

AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutationnetwork'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arrangedin four columns and four rows for processing as a matrix −

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration −

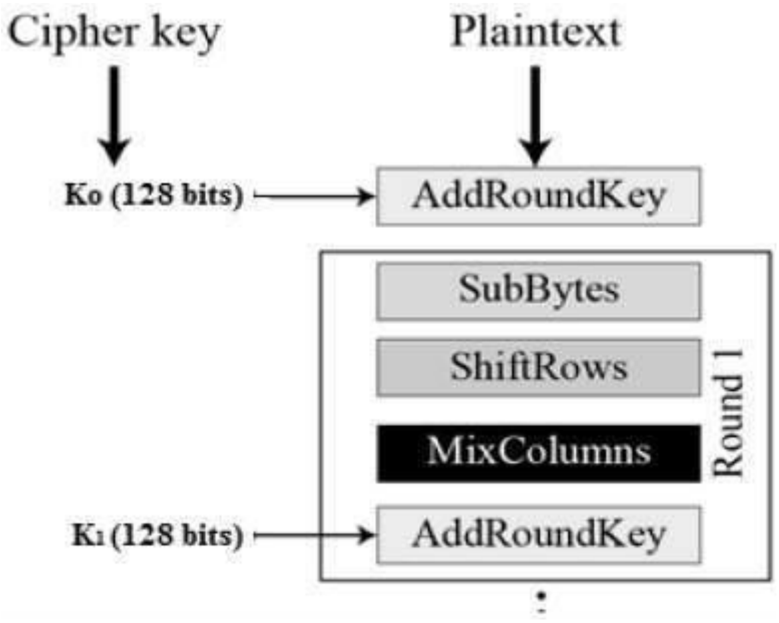


R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each roundcomprise of four sub-processes. The first round process is depicted below –



Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given indesign. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of row. Shift is carried out as follows −

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respectto each other.

MixColumns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

- Add round key
- Mix columns
- Shift rows
- Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher,the encryption and decryption algorithms needs to be separately implemented,although they are very closely related.

AES Analysis

In present day cryptography, AES is widely adopted and supported in both hardware and software. Till date, no practical cryptanalytic attacks against AES has been discovered. Additionally, AES has built-in flexibility of key length, which allows a degree of 'future-proofing' against progress in the ability to perform exhaustive key searches.

However, just as for DES, the AES security is assured only if it is correctly implemented and good key management is employed.

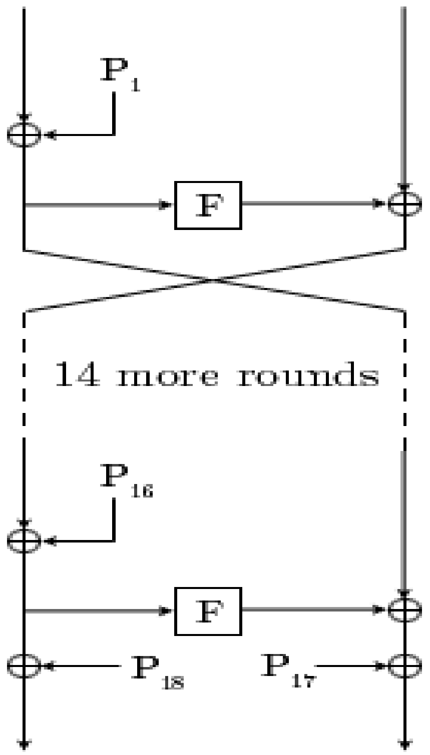
Blow Fish

- Blowfish is a keyed, symmetric block cipher, designed in 1993 by Bruce Schneier and included in a large number of cipher suites and encryption products.
- Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDE**A.**
- Blowfish was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms.
- It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use.
- Blowfish is unpatented and license-free, and is available free for all uses.

- While no effective cryptanalysis of Blowfish has been found to date, more attention is now given to block ciphers with a larger block size, such as AES or Twofish.

The Blowfish Algorithm

- **There are two parts to this algorithm;**
    - A part that handles the expansion of the key.
    - A part that handles the encryption of the data.
- **The expansion of the key:** break the original key into a set of subkeys. Specifically, a key of no more than 448 bits is separated into 4168 bytes. There is a P-array and four 32-bit S-boxes. The P-array contains 18 32-bit subkeys, while each S-box contains 256 entries.
- **The encryption of the data:** 64-bit input is denoted with an x, while the P-array is denoted with a Pi (where i is the iteration).
- Blowfish has a 64-bit block size and a key length of anywhere from 32 bits to 448 bits (32-448 bits in steps of 8 bits; default 128 bits).
- It is a 16-round Feistel cipher and uses large key-dependent S-boxes. It is similar in structure to CAST-128, which uses fixed S-boxes.



- From the above diagram each line represents 32 bits. The algorithm keeps two subkey arrays: the 18-entry P-array and four 256-entry S-boxes.
- The S-boxes accept 8-bit input and produce 32-bit output. One entry of the P-array is used every round, and after the final round, each half of the data block is XORed with one of the two remaining unused P-entries.

Working Principle

- Initialize the P-array and S-boxes
- XOR P-array with the key bits. For example, P1 XOR (first 32 bits of key), P2 XOR (second 32 bits of key), ...
- Use the above method to encrypt the all-zero string
- This new output is now P1 and P2
- Encrypt the new P1 and P2 with the modified subkeys
- This new output is now P3 and P4
- Repeat 521 times in order to calculate new subkeys for the P-array and the four S-boxes

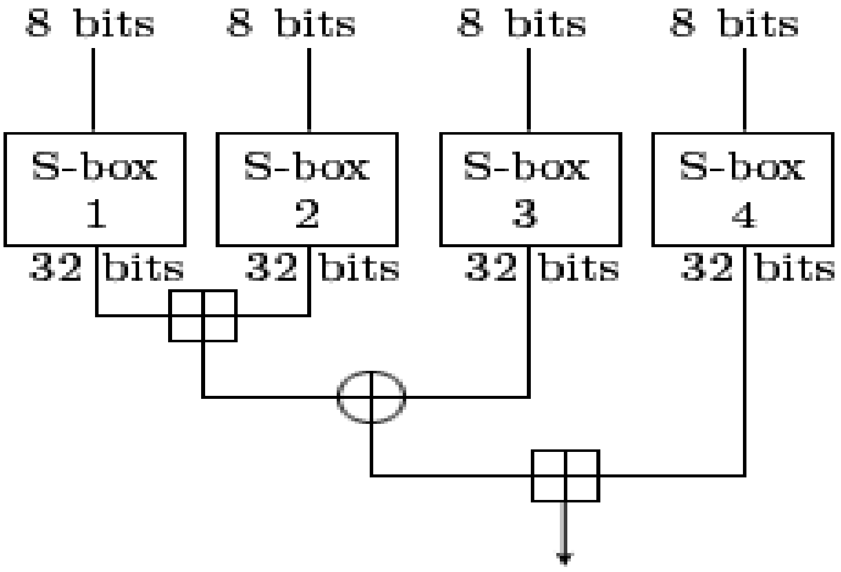


Fig *Blowfish's F function*

- The above diagram shows Blowfish's F-function. The function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The outputs are added modulo 2^{32} and XORed to produce the final 32-bit output.
- Since Blowfish is a Feistel network, it can be inverted simply by XORing P17 and P18 to the ciphertext block, then using the P-entries in reverse order.
- Blowfish's key schedule starts by initializing the P-array and S-boxes with values derived from the hexadecimal digits of pi, which contain no obvious pattern.
- The secret key is then XORed with the P-entries in order (cycling the key if necessary). A 64-bit all-zero block is then encrypted with the algorithm as it stands.
- The resultant ciphertext replaces P1 and P2. The ciphertext is then encrypted again with the new subkeys, and P3 and P4 are replaced by the new ciphertext. This continues, replacing the entire P-array and all the S-box entries.
- In all, the Blowfish encryption algorithm will run 521 times to generate all the subkeys - about 4KB of data is processed.

RC4:

RC4 is an encryption algorithm that was created by Ronald Rivest of RSA Security. Itis used in WEP and WPA, which are encryption protocols commonly used on wireless routers. The workings of RC4 used to be a secret, but its code was leaked onto the internet in 1994. RC4 was originally very widely used due to its simplicity and speed. Typically 16 byte keys are used for strong encryption, but shorter key lengths are also widely used due to export restrictions. Over time this code was shown to producebiased outputs towards certain sequences, mostly in first few bytes of the keystream generated.

To begin the process of RC4 encryption, you need a key, which is often user-defined and between 40-bits

and 256-bits. A 40-bit key represents a five character ASCII code that gets translated into its 40 character binary equivalent (for example, the ASCII key "pwd12" is equivalent to 0111000001110111011001000011000100110010
in binary). The next part of RC4 is the key-scheduling algorithm (KSA), listed below.

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap(S[i],S[j])
endfor
```

KSA creates an array S that contains 256 entries with the digits 0 through 255, as inthe table below.

0	1	2	...	i	i+1	...	253	254	255

Each of the 256 entries in *S* are then swapped with the *j*-th entry in *S*, which is computed to be

$$j = [(j + S(i) + key[i \bmod keylength]) \bmod 256],$$

where j is the previous j value (which is initially zero). S[i] is the value of the current entry in
S. key[i mod keylength] is either a zero or a one.

For example, if we are at the 52th entry in S and the keylength was 40-bit, then 52 mod 40
= 12. The 13th element (because numbering for arrays begins at zero) in the binary versionof "pwd12" is 0. For example, consider the first iteration of KSA with key "pwd12".

Then, since i = 0, i mod 256 = 0. So, the element at the index 0 of the key is p, and its asciivalue is 112. So, the new j is computed as

j = [(0 + 0 + 112) mod 256] = 112.

So, swapping the i-th and the j-th elements, we obtain the following array after the firstiteration:

112	1	2	...	111	0	113	114	...	255

The next part of RC4 is the pseudo-random generation algorithm (PRGA). The PRGA is below:

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap(S[i],S[j])
    output S[(S[i] + S[j]) mod 256]
endwhile
```

In PRGA, we begin with the array S that was swapped in the KSA. In PRGA, an element in S (at index i) is swapped with another element in S (at index j). Then, the next element in the encrypted text is the element of S at the index calculated by (S[i]

+ S[j] mod 256). At each iteration, i is recalculated as (i + 1) mod 256, and j is recalculated as (j + S[i]) mod 256. The number of iterations performed is the length of the key, and every value of S is swapped at least once beyond 256 iterations (due to the fact that i and j are calculated by some number n mod 256).

PKC-Public-Key Cryptosystems (Asymmetric Cryptosystem)

Principles of Public-Key Cryptosystems

Key Points

●Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the different keys one a public key and one a private key. It is also known as public-key encryption.

●Asymmetric encryption transforms plaintext into ciphertext using a one of two keys and an encryption algorithm. Using the paired key and a decryption algorithm, the plaintext is recovered from the ciphertext.

●Asymmetric encryption can be used for confidentiality, authentication, or both.

●The most widely used public-key cryptosystem is RSA. The difficulty of attacking RSA is based on the difficulty of finding the prime factors of a composite number.

Asymmetric algorithms have the following important characteristic:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

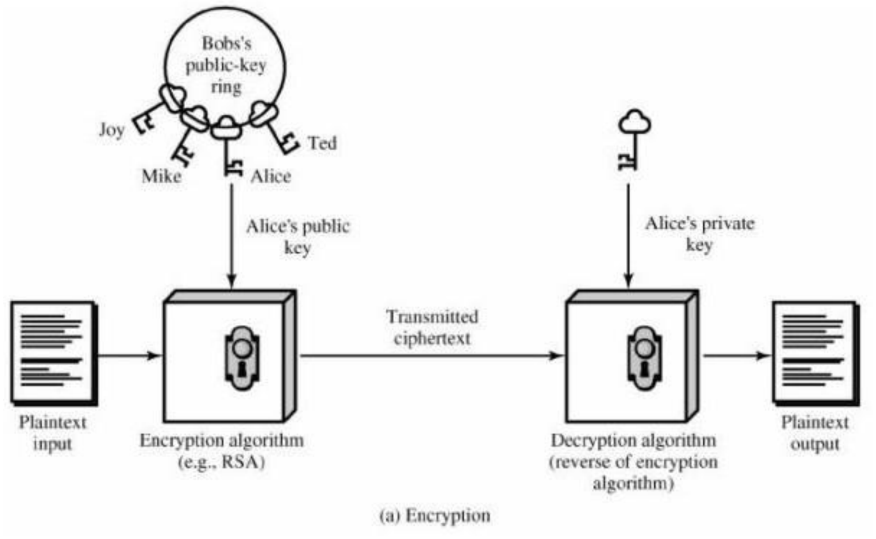
A public-key encryption scheme has six ingredients:

- **Plaintext**: This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private keys:** 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.
- **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.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

The essential steps of PKC:

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 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.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a user's private key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

Difference between Conventional and Public-Key Encryption

Conventional Encryption	Public-Key Encryption
1. The same algorithm with the same key is used for encryption and decryption.	**1.** One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.
2. The sender and receiver must share the	**2.** The sender and receiver must each have

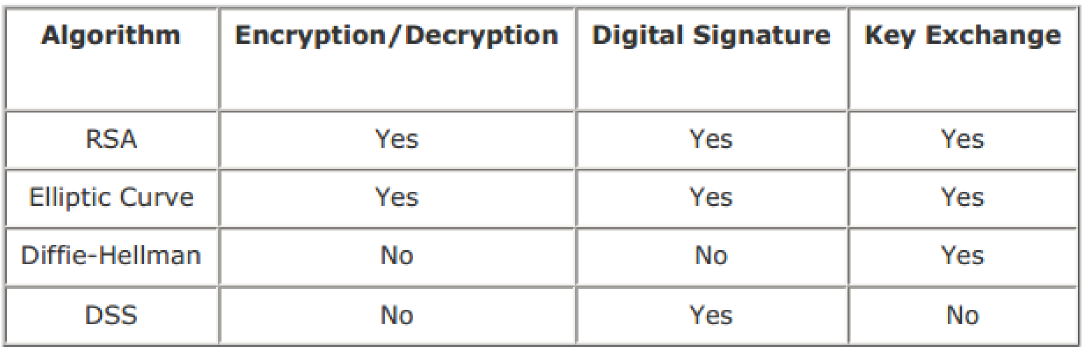
algorithm and the key.	one of the matched pair of keys (not the same one).
3. The key must be kept secret.	**3.** One of the two keys must be kept secret.
4. It must be impossible or at least impractical to decipher a message if no other information is available.	**4.** It must be impossible or at least impractical to decipher a message if no other information is available.
5. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	**5.** Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Applications for Public-Key Cryptosystems

In broad terms, we can classify 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

Requirements for Public-Key Cryptography

1. It is computationally easy for a party B to generate a pair (public key *PUb*, private key *PRb*).
2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, *M*, to generate the corresponding ciphertext:

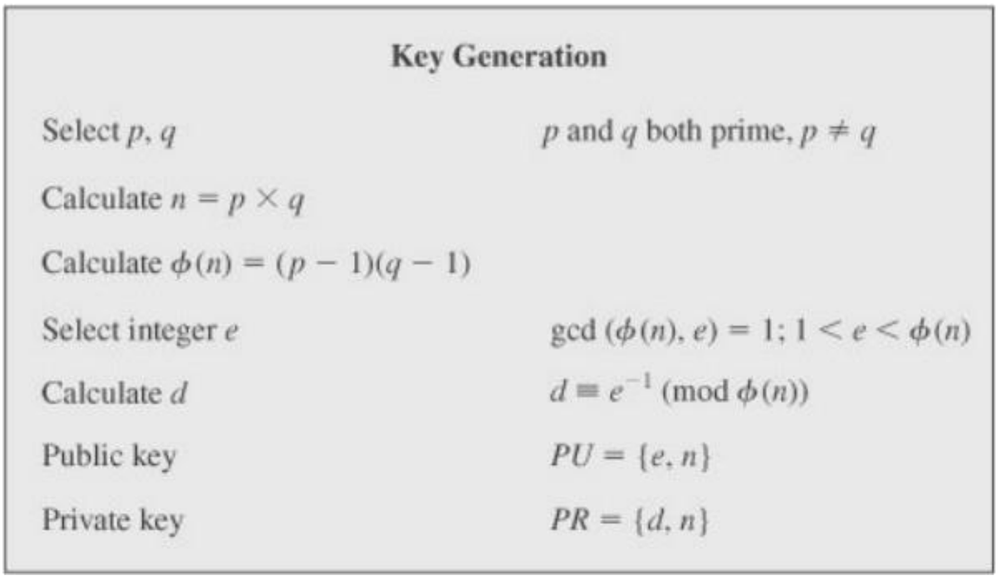
   $C = \mathrm{E}(PUb, M)$

3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:
   M = D(PRb, C) = D[PRb, E(PUb, M)]
4. It is computationally infeasible for an adversary, knowing the public key, *PUb*, to determine the private key, *PRb*.
5. It is computationally infeasible for an adversary, knowing the public key, *PUb*, and a ciphertext, *C*, to recover the original message, *M*.
   We can add a sixth requirement that, although useful, is not necessary for all public-key applications:
6. The two keys can be applied in either order:
   M = D[PUb, E(PRb, M)] = D[PRb, E(PUb, M)]

The RSA Algorithm

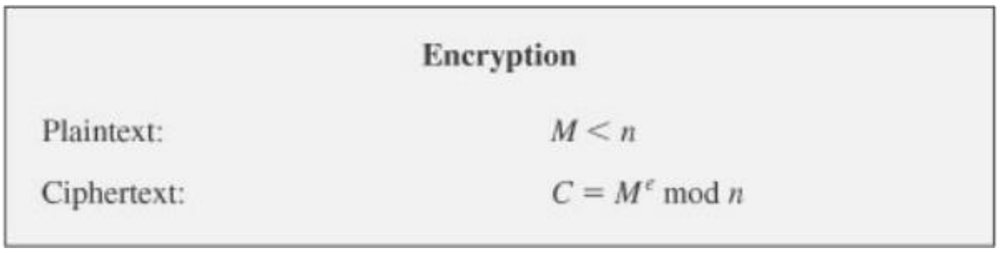
RSA was invented by three scholars **Ron Rivest, Adi Shamir,** and **Len Adleman** and hence, it is termed as RSA cryptosystem.We will see two aspects of the RSA cryptosystem,

- firstly generation of key pair and
- Secondly encryption-decryption algorithms.



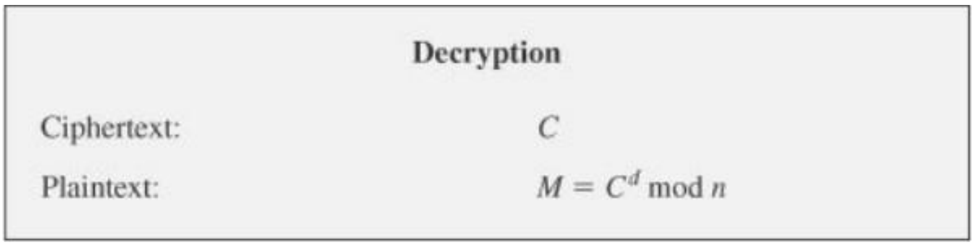
Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$



Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$



Decryption

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Example

An example of generating RSA Key pair is given below. (For ease of understanding, the primes p & q taken here are small values. Practically, these values are very high).

- Let two primes be p = 7 and q = 13. Thus, modulus n = pq = 7 x 13 = 91.
- Select e = 5, which is a valid choice since there is no number that is common factor of 5 and (p − 1)(q − 1) = 6 × 12 = 72, except for 1.
- The pair of numbers (n, e) = (91, 5) forms the public key and can be made available to anyone whom we wish to be able to send us encrypted messages.
- Input p = 7, q = 13, and e = 5 to the Extended Euclidean Algorithm. The output will be d = 29.
- Check that the d calculated is correct by computing –

```
de = 29 × 5 = 145 = 1 mod 72
```

- Hence, public key is (91, 5) and private keys is (91, 29).

RSA Security Analysis

The security of RSA depends on the strengths of two separate functions. The RSA cryptosystem is most popular public-key cryptosystem strength of which is based on the practical difficulty of factoring the very large numbers.

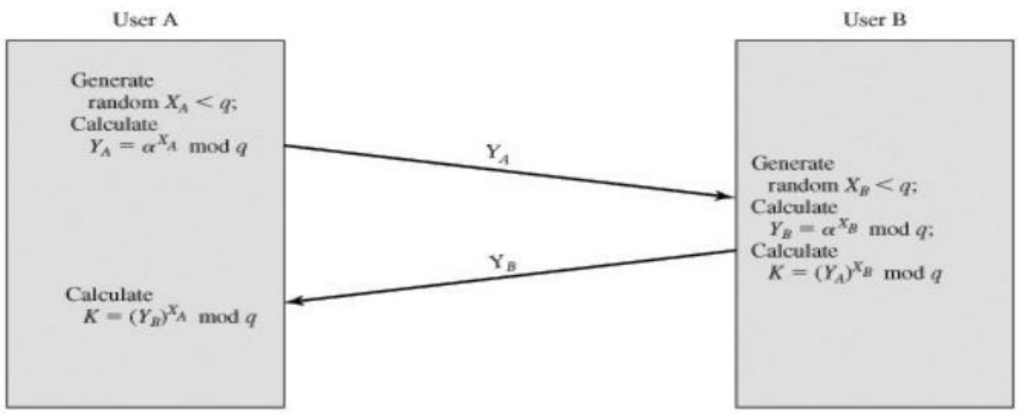
- **Encryption Function** − It is considered as a one-way function of converting plaintext into ciphertext and it can be reversed only with the knowledge of private key d.
- **Key Generation** − The difficulty of determining a private key from an RSA public key is equivalent to factoring the modulus n. An attacker thus cannot use knowledge of an RSA public key to determine an RSA private key unless he can factor n. It is also a one way function, going from p & q values to modulus n is easy but reverse is not possible.

If either of these two functions are proved non one-way, then RSA will be broken. In fact, if a technique for factoring efficiently is developed then RSA will no longer be safe.

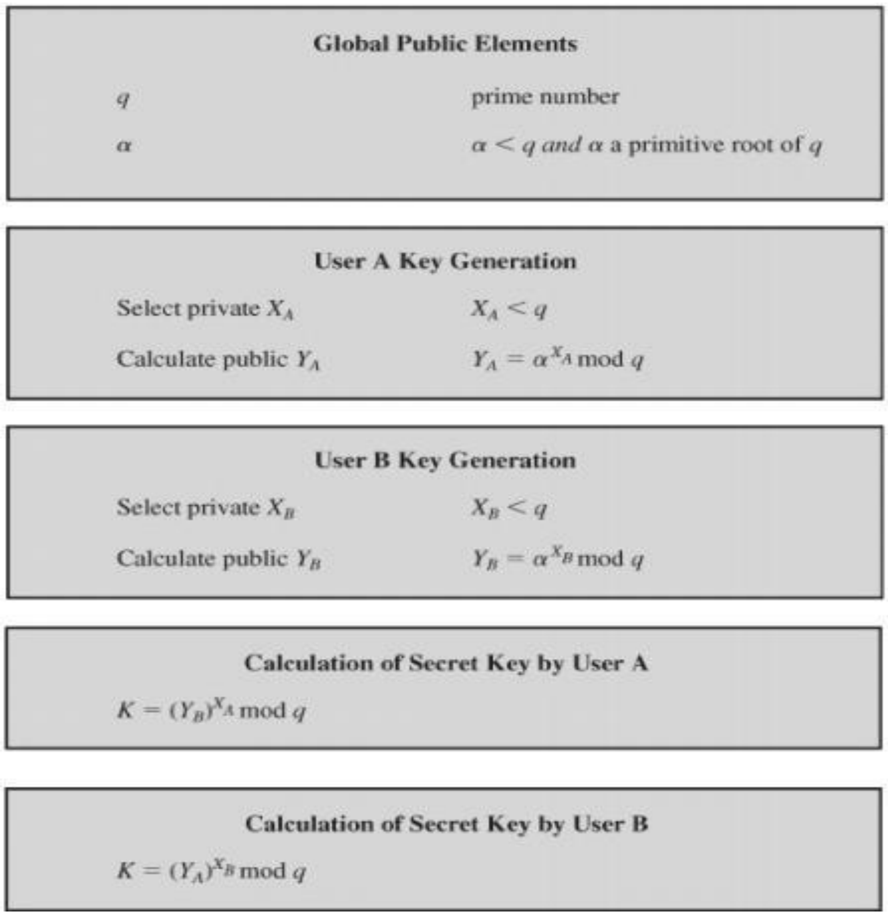
The strength of RSA encryption drastically goes down against attacks if the number p and q are not large primes and/ or chosen public key e is a small number.

Diffie-Hellman Key Exchange

- The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.
- The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms


User A
User B
Generate random $X_A < q$;
Calculate $Y_A = \alpha^{X_A} \bmod q$
Y_A
Generate random $X_B < q$;
Calculate $Y_B = \alpha^{X_B} \bmod q$;
Calculate $K = (Y_A)^{X_B} \bmod q$
Y_B
Calculate $K = (Y_B)^{X_A} \bmod q$

The Algorithm



Global Public Elements

q	prime number
α	$\alpha < q$ *and* α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Calculation of Secret Key by User A

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

Calculation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Assume two communication Parties A for (Anu) and B for (Banu):

First, Anu and Banu agree on a large prime, ***n*** and ***g***, such that ***g*** is *primitive root* mod ***n***. These two integers don't have to be secret; Anu and Banu can agree to them over some insecure channel. They can even be common among a group of users. It doesn't matter.

Then, the algorithm goes as follows:

(1) Anu chooses a random large integer x and sends Banu

$$X = g^{x} \; mod \; n$$

(2) Banu chooses a random large integer *y* and sends to Anu

$$Y = g^{y} \; mod \; n$$

(3) Anu computes

$$k = Y^{x} \; mod \; n$$

(4) Banu computes

$$K^{I} = X^{y} \; mod \; n$$

Both k and k′ are equal to g^{xy} ***mod n***.

Example

Assume Anu generated the values as ***n*** =11 and ***g***= 7 and communicated to Banu over a channel.

Anu	Banu
n=11, g=7	n=11, g=7

b. Anu and Banu has the same n and g values.

Anu	Banu
x=3	y=9

c. Anu and Banu computed ***A*** and ***B*** values at their sides.

Anu	Banu
$A=g^{X}$ *mod n*	$B=g^{Y}$ *mod n*
$=7^{3}$ mod 11	$=7^{9}$ mod 11
=343 mod 11	= 40353607 mod 11
=2	**=8**

d. Anu and Banu computing their keys where K1 = K2.

Anu	Banu
$K1=B^{X}$ *mod n*	$K2=A^{Y}$ *mod n*
$=8^{3}$ mod 11	$=2^{9}$ mod 11
=512 mod 11	=512 mod 11
=6	=6

Man-In-The-Middle Attack (MIMA)

In cryptography and computer security, a man-in-the-middle attack is an attack where the attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other.

For example,

1. Anu wants to communicate with Banu privately. Then Anu chooses and sends the values of ***n*** and ***g*** to Banu. Let ***n=11 and g=7.***

2. Anu does not realize Cnu that Cnu is listening quietly to the conversation. Cnu simply note down the values of ***n*** and ***g***, and also forwards them to Banu as they originally were.

Anu	**Cnu**	**Banu**
n=11, g=7	**n=11, g=7**	**n=11, g=7**

3. Now Anu, Cnu and Banu select random numbers say ***x*** and ***y***.

Anu	**Cnu**	**Banu**
x=3	**x=8,y=6**	**y=9**

4. Anu calculates her **A** value and Banu calculates his **B** value whereas Cnu calculates both **A** and **B** values at his side to play the role of *man in middle*.

Anu	**Cnu**	**Banu**
$A=g^X$ mod n	$A=g^X$ mod n	$B=g^Y$ mod n
$=7^3$ mod 11	$=7^8$ mod 11	$=7^9$ mod 11
=343 mod 11	= 5764801 mod 11	= 40353607 mod 11
=2	= **9**	**=8**
	$B=g^Y$ mod n	
	$= 7^6$ mod 11	
	= 117649 mod 11	
	= **4**	

5. Anu sends her **A** value to Banu. Cnu intercepts it and send his **A** value 9 instead of Anu's **A** values to him. In response, Banu sends his B value 8 to Anu but Cnu intercepts it and sends his B value 4 to Anu.

Based on these values, all the three persons now calculate their keys.

Anu	**Cnu**	**Banu**
$K1=B^X$ mod n	$K1=B^Y$ mod n	$K2=A^Y$ mod n
$=4^3$ mod 11	$=8^8$ mod 11	$=9^9$ mod 11
=64 mod 11	= 16777216 mod 11	=387420489 mod 11
=9	=5	=5
	$K2=A^Y$ mod n	
	$=2^6$ mod 11	
	=64 mod 11	
	=9	

The ElGamal Public Key Encryption Algorithm

The ElGamal Algorithm provides an alternative to the RSA for public key encryption.

1) Security of the RSA depends on the (presumed) *difficulty of factoring large integers*.
2) Security of the ElGamal algorithm depends on the (presumed) *difficulty of computing discrete logs* in a large prime modulus.

ElGamal has the disadvantage that the ciphertext is twice as long as the plaintext.

It has the advantage the same plaintext gives a different ciphertext (with near certainty) each time it is encrypted.

Alice chooses

i) A large prime $\boldsymbol{p}_{\mathbf{A}}$ (say 200 to 300 digits),
ii) A primitive element $\boldsymbol{\alpha}_{\mathbf{A}}$ modulo p_A,
iii) A (possibly random) integer $\boldsymbol{d}_{\mathbf{A}}$ with $2 \le d_A \le p_A - 2$.

Alice computes

iv) $\boldsymbol{\beta}_{\mathbf{A}} \equiv \alpha_A^{d_A} \pmod{p_A}$.

Alice's *public key* is $(\boldsymbol{p}_{\mathbf{A}}, \boldsymbol{\alpha}_{\mathbf{A}}, \boldsymbol{\beta}_{\mathbf{A}})$. Her *private key* is $\boldsymbol{d}_{\mathbf{A}}$.

Bob encrypts a short message M ($M < p_A$) and sends it to Alice like this:

i) Bob chooses a random integer k (which he keeps secret).
ii) Bob computes $\boldsymbol{r} \equiv \boldsymbol{\alpha}_{\mathbf{A}}^{k} \pmod{\boldsymbol{p}_{\mathbf{A}}}$ and $\boldsymbol{t} \equiv \boldsymbol{\beta}_{\mathbf{A}}^{k}\,\mathbf{M} \pmod{p_A}$, and then discards k.

Bob sends his encrypted message $(\boldsymbol{r}, \boldsymbol{t})$ to Alice.

When Alice receives the encrypted message (r, t), she decrypts (using her private key d_A) by computing $\boldsymbol{t}\boldsymbol{r}^{-d_A}$.

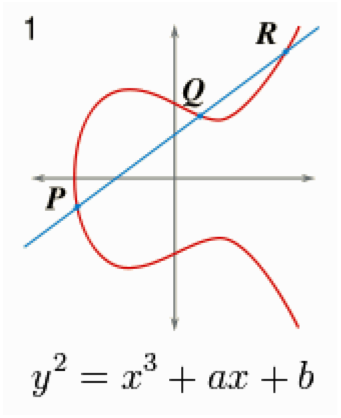
Note
$$\begin{aligned} tr^{-d_A} &\equiv \beta_A^{k} M (\alpha_A^{k})^{-d_A} && \pmod{p_A} \\ &\equiv (\alpha_A^{d_A})^{k} M (\alpha_A^{k})^{-d_A} && \pmod{p_A} \\ &\equiv M && \pmod{p_A} \end{aligned}$$

Elliptic Curve Cryptographic algorithm (ECC)

- Elliptic Curve Cryptography (ECC) was discovered in 1985 by Victor Miller (IBM) and Neil Koblitz (University of Washington) as an alternative mechanism for implementing public-key cryptography.
- Elliptical curve cryptography is a public key encryption technique which is based on the theory of elliptical curves.
- This encryption technique uses the properties of elliptic curve in order to generate keys instead of using the traditional methodology of generation of keys using the product of two very large prime numbers.
- ECC is a public key cryptosystem which is used to generate the public key and the private key in order to encrypt and decrypt the data.
- It is based on the mathematical complexity of solving the elliptic curve discrete logarithm problem which deals with the problem of calculating the number of steps or hops it takes to move from one point to another point on the elliptic curve.
- Elliptic curves are the binary curves and are symmetrical over x- axis. These are defined by the function:

$$y^2 = x^3 + ax + b$$

- Where x and y are the standard variables that define the function while as a and b are the constant coefficients that define the curve.
- As the values of a and b change, elliptical curve also alters.
- For elliptical curves, the discriminant is non zero.
- The operations used on elliptical curves in cryptography are point addition, point multiplication and point doubling.
- The important characteristic of elliptic curve is the finite field concept.
- This means that there is a way to limit the values on the curve.



<u>Few terms that will be used,</u>

E -> Elliptic Curve
P -> Point on the curve
n -> Maximum limit (This should be a prime number)

<u>Key Generation</u>

- Key generation is an important part where we have to generate both public key and private key. The sender will be encrypting the message with receiver's public key and the receiver will decrypt its private key.

- Now, we have to select a number**'d'** within the range of **'n'**.
- Using the following equation we can generate the public key

- **Q = d * P**
- **d** = The random number that we have selected within the range of (**1 to n-1**). **P** is the point on the curve.
- **‘Q’ is the public key** and **‘d’ is the private key.**

<u>Encryption</u>

- Let ‘M’ be the message that we are sending. We have to represent this message on the curve.
- Randomly select ‘k’ from [1 – (n-1)].
- Two cipher texts will be generated let it be **C1** and **C2**.

C1 = k*P

C2 = M + k*Q

C1 and C2 will be send.

<u>Decryption</u>

- We have to get back the message ‘m’ that was send to us,
    - **M = C2 – d * C1**
- M is the original message that we have send.

<u>Proof</u>

How do we get back the message,

- C2 – d * C1 = (M + k * Q) – d * (k * P) (C2 = M + k * Q and C1 = k * P)
- = M + k * **d * P** – d * k *P (Since Q =d * p)
- = M (Original Message)