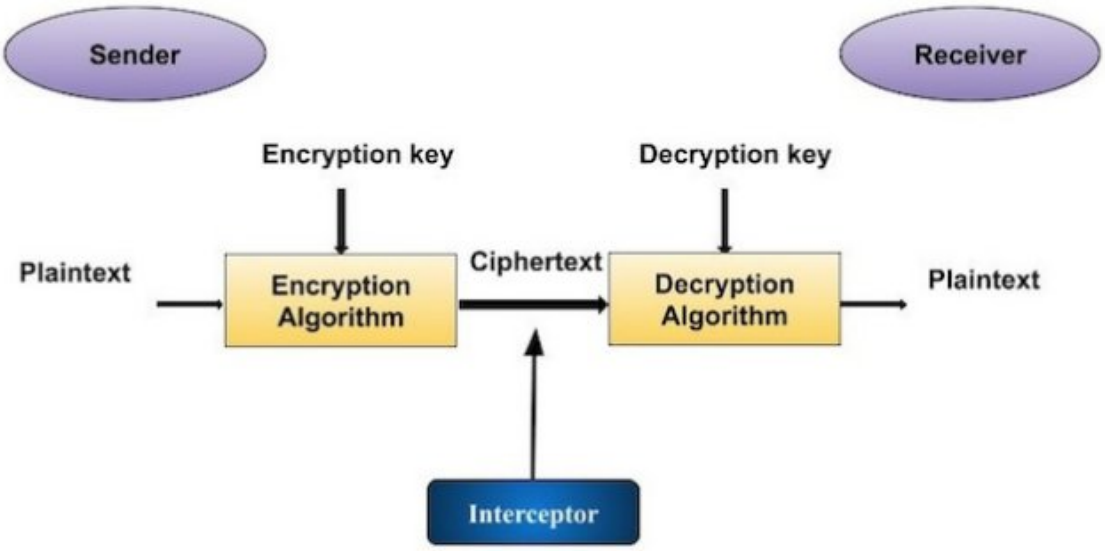
Cryptography:

Cryptography is the science and practice of securing communication and information by converting it into a form that is unintelligible to unauthorized parties.



Types of cryptography:

There are three types of cryptography techniques :
1. Secret key Cryptography
2. Public key cryptography
3. Hash Functions

1. Secret Key Cryptography or Symmetric-Key Encipherment:

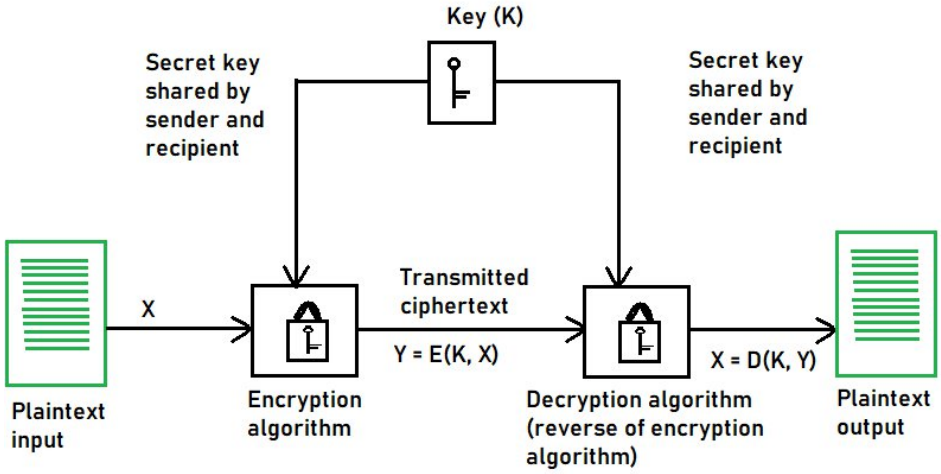
The encryption process where **same keys are used for encrypting and decrypting** the information is known as Symmetric Key Encryption.

Symmetric Key Cryptography, also known as **private key cryptography**,

- **Encryption**:
    - The sender uses a symmetric encryption algorithm and a secret key to transform plaintext into ciphertext.
    - Ciphertext is transmitted to the recipient.
- **Decryption**:
    - The recipient uses the same secret key and the decryption algorithm to convert the ciphertext back into plaintext.

Symmetric Encryption Algorithms:

- **DES (Data Encryption Standard)**:
- **3DES (Triple DES)**:
- **AES (Advanced Encryption Standard)**:
- **RC4, RC5, RC6**
- **Blowfish/Twofish**:



2. Public key cryptography or Asymmetric-Key Encipherment:

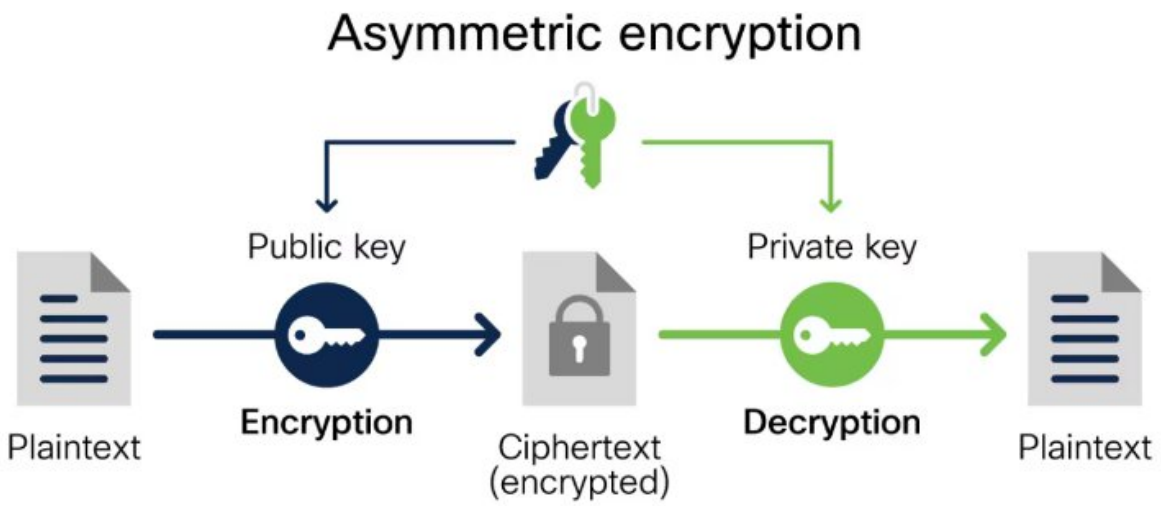
Asymmetric Key Cryptography, also known as **public key cryptography**, is uses a pair of keys: a **public key** and a **private key**.

Process:

1. **Encryption**:
   - The sender encrypts the plaintext using the recipient's public key.
   - The ciphertext is sent to the recipient.
2. **Decryption**:
   - The recipient uses their private key to decrypt the ciphertext, retrieving the original plaintext.

Asymmetric Encryption Algorithms:

- **RSA (Rivest-Shamir-Adleman)**
- **Elliptic Curve Cryptography (ECC)**
- **DSA (Digital Signature Algorithm):**
- **Diffie-Hellman**



Terminology:

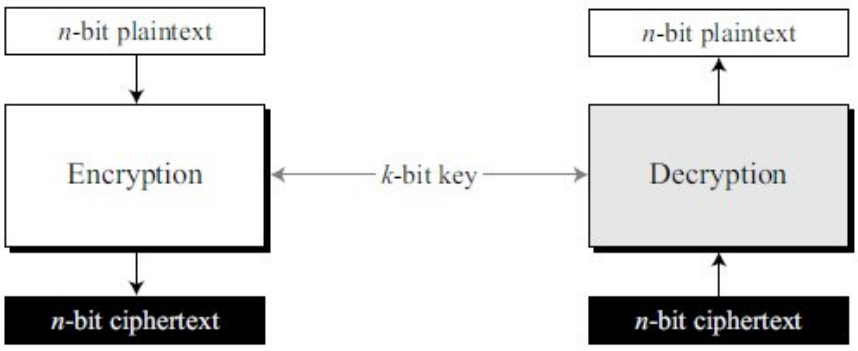
Basic Terms:

1. **Plaintext**: The original, readable data or message before encryption.
2. **Ciphertext**: The encrypted, unreadable version of the plaintext.
3. **Encryption**: The process of converting plaintext into ciphertext using an algorithm and a key.
4. **Decryption**: The process of converting ciphertext back into plaintext using an algorithm and a key.
5. **Key**: A string of characters used in encryption and decryption. Keys can be symmetric (shared) or asymmetric (public/private pair).

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.

Figure 5.1 *A modern block cipher*



- The common values for n are 64, 128, 256, or 512 bits.

Substitution or Transposition:

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.

If the cipher is designed as a transposition cipher, the bits are only reordered (transposed).

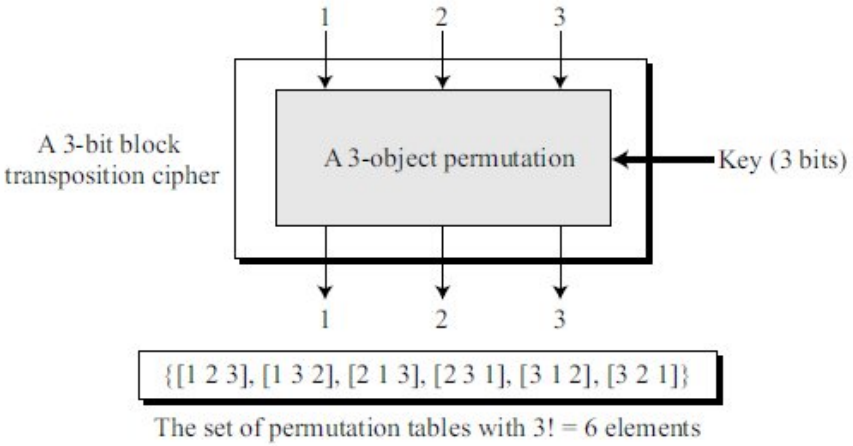
Block Ciphers as Permutation Groups:

Full-Size Key Transposition Block Ciphers:

- A full-size key transposition cipher only transposes bits without changing their values.
- an n-object permutation with a set of n! permutation tables
- 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 Figure 5.2. The key should be $\lceil \log_2 6 \rceil$ = 3 bits long

Figure 5.2 *A transposition block cipher modeled as a permutation*

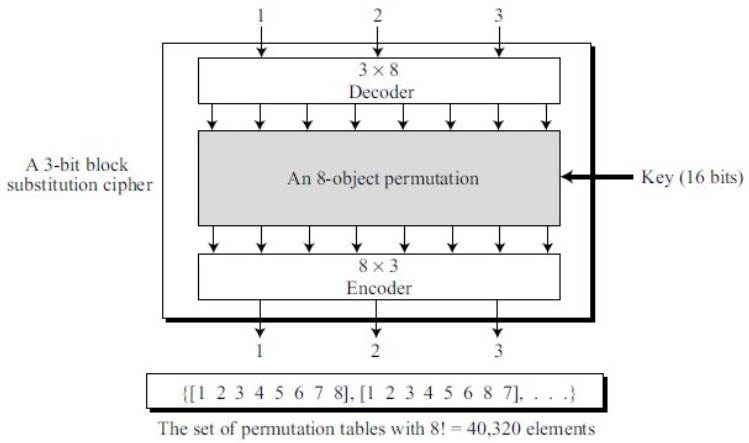


Full-Size Key Substitution Block Ciphers:

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.
Encoding is the reverse process.

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.

Figure 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.

Components of a Modern Block Cipher

A modern block cipher is made of a combination of transposition units (called P-boxes), substitution units (called S-boxes), and some other units.

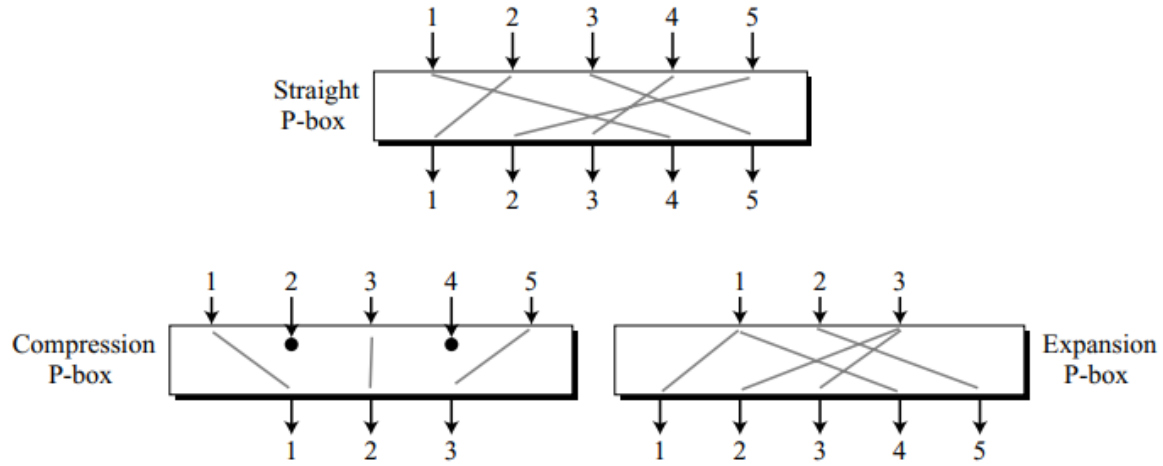
P-Boxes:

It transposes bits.

We can find three types of P-boxes in modern block ciphers:

- Straight P-boxes,
- Expansion P-boxes,
- Compression P-boxes,

Figure 5.4 *Three types of P-boxes*



Straight P-Boxes:

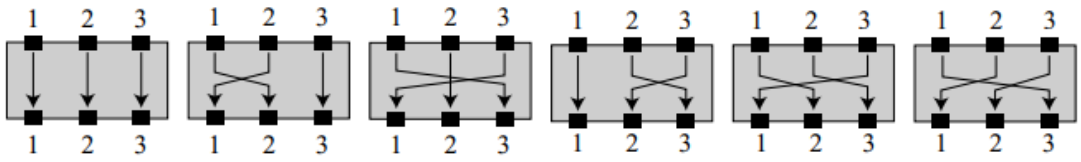
A straight P-Box with n inputs and n outputs is a permutation.

There are n! possible mappings.

Example 5.5

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

Figure 5.5 *The possible mappings of a 3×3 P-box*



Compression P-Boxes :

A compression P-box is a P-box with n inputs and m outputs where m < n.

Some of the inputs are blocked and do not reach the output.

Expansion P-Boxes:

An expansion P-box is a P-box with n inputs and m outputs where m > n.

Some of the inputs are connected to more than one output.

Invertibility:

A straight P-box is invertible. This means that we can use a straight P-box in the encryption cipher and its inverse in the decryption cipher.

Compression and expansion P-boxes have no inverses.

Example 5.7

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

Figure 5.6 *Inverting a permutation table*

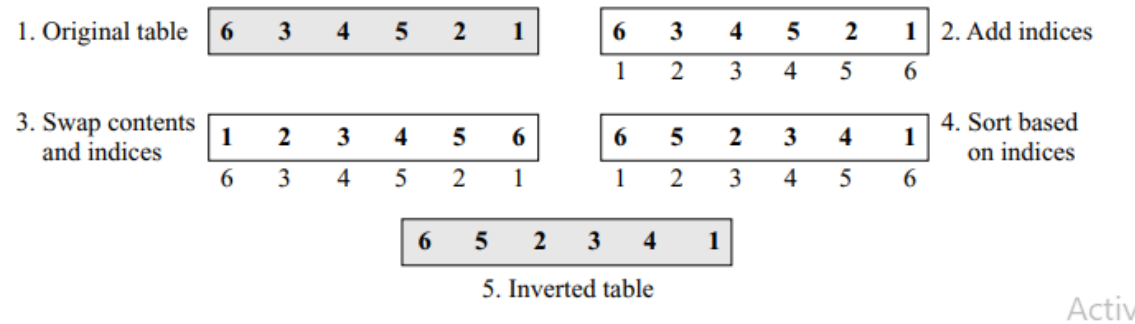
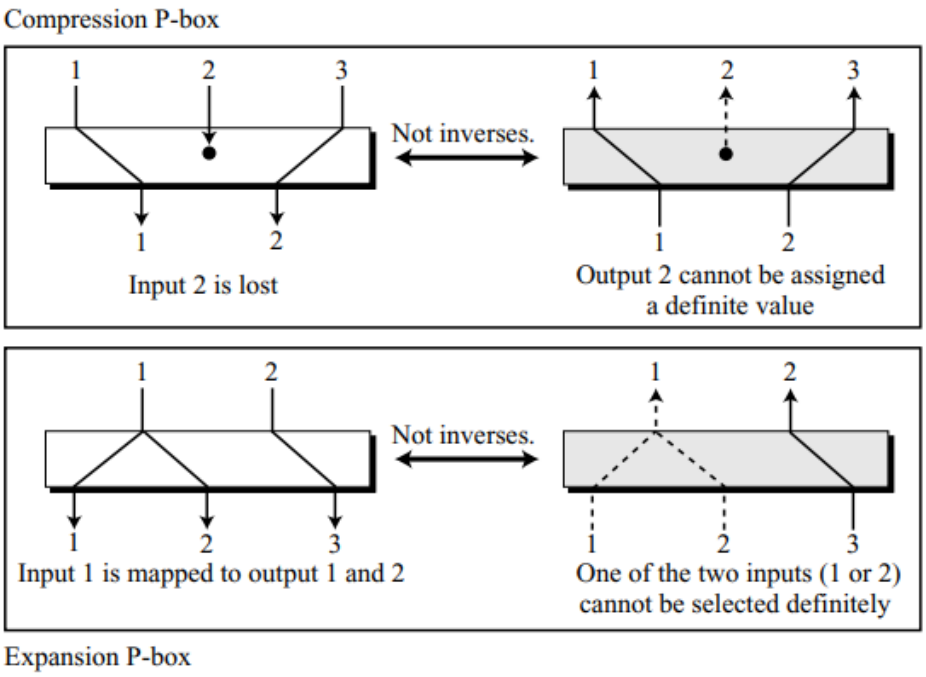


Figure 5.7 *Compression and expansion P-boxes as non-invertible components*



S-Boxes:

An S-box (substitution box).

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.

Invertibility. 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

Circular Shift:

Another component found in some modern block ciphers is the circular shift operation.

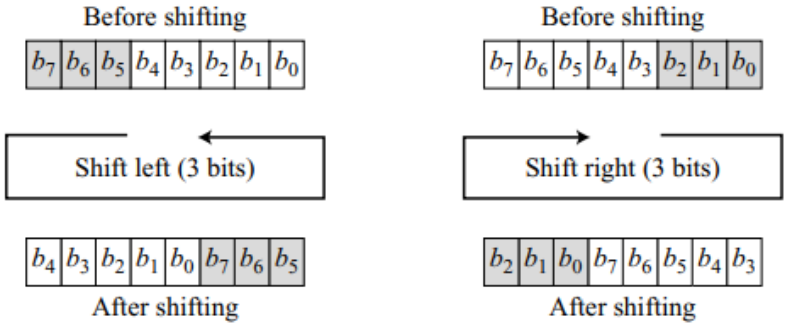
Shifting can be to the left or to the right.

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.

Figure 5.10 *Circular shifting an 8-bit word to the left or right*



Invertibility

A circular left-shift operation is the inverse of the circular right-shift operation.

Swap:

The swap operation is a special case of the circular shift operation where k = n/2.

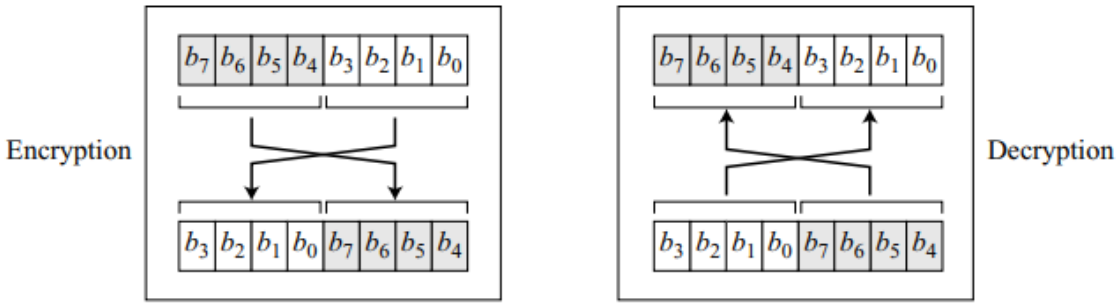
This means this operation is valid only if n is an even number.

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.

Figure 5.11 *Swap operation on an 8-bit word*



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.

Product Ciphers:

A product cipher is a complex cipher combining substitution, permutation, and other components.

Diffusion and Confusion:

The idea of diffusion is to hide the relationship between the ciphertext and the plaintext.

The idea of confusion is to hide 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, P-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.

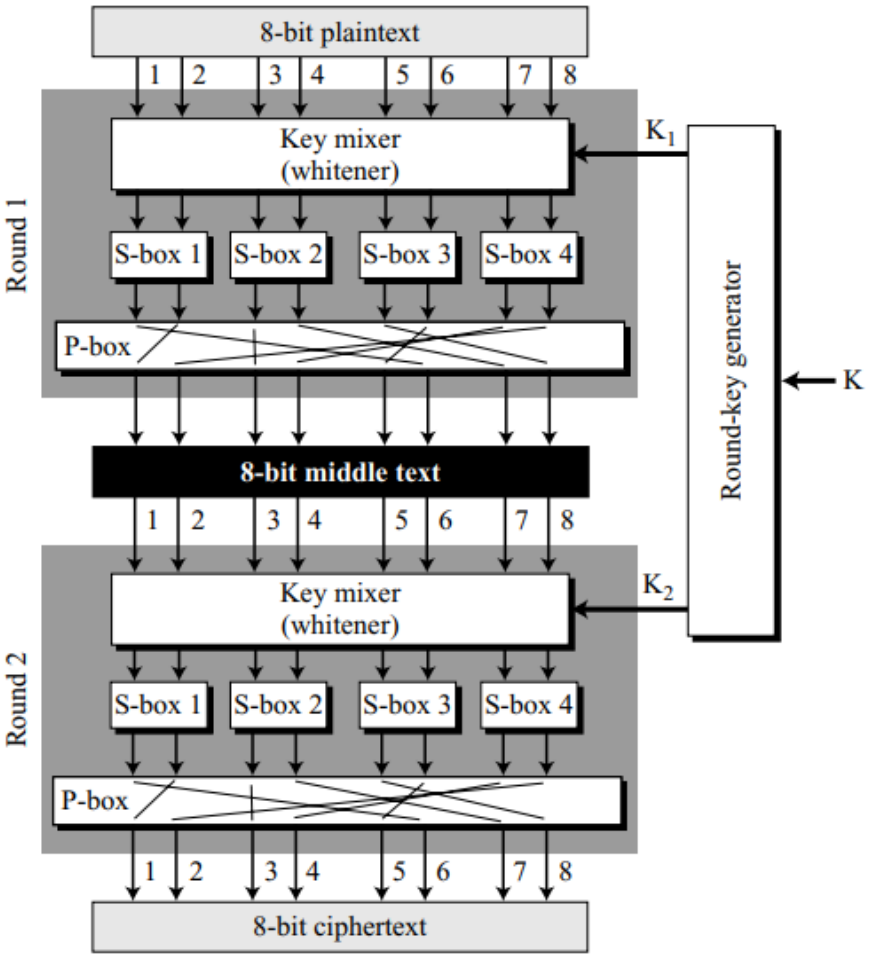
In an N-round cipher, the plaintext is encrypted N times to create the ciphertext; the ciphertext is decrypted N times to create the plaintext.

We refer to the text created at the intermediate levels (between two rounds) as the middle text.

In Figure 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).
b. The outputs of the whitener are organized into four 2-bit groups and are fed into four S-boxes.
c. The outputs of S-boxes are passed through a P-box to permute the bits so that in the next round each box receives different inputs

Figure 5.13 *A product cipher made of two rounds*



Two Classes of Product Ciphers:

Modern block ciphers are all product ciphers, they are divided into two classes.

- Feistel ciphers

The ciphers in the first class use both invertible and noninvertible components

- non-Feistel ciphers

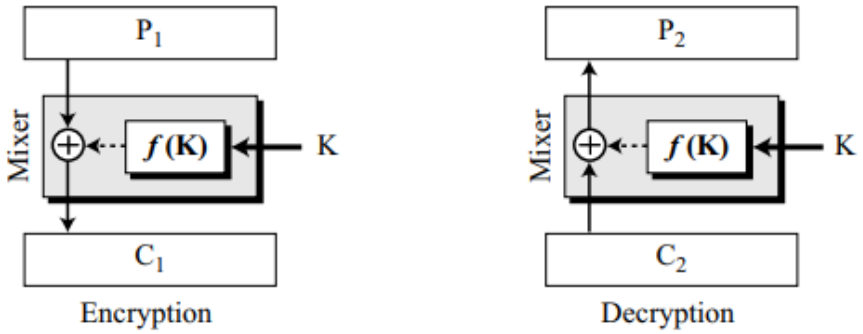
The ciphers in the second class use only invertible components.

Feistel Ciphers:

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.

Figure 5.15 *The first thought in Feistel cipher design*



In the encryption, a noninvertible function, ƒ(K), accepts the key as the input. The output of this component is exclusive-ored with the plaintext.

The result becomes the ciphertext.

combination of the function and the exclusive-or operation the mixer.

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

Decryption: $P_2 = C_2 \oplus f(K) = C_1 \oplus f(K) = P_1 \oplus f(K) \oplus f(K) = P_1 \oplus (00\ldots0) = P_1$

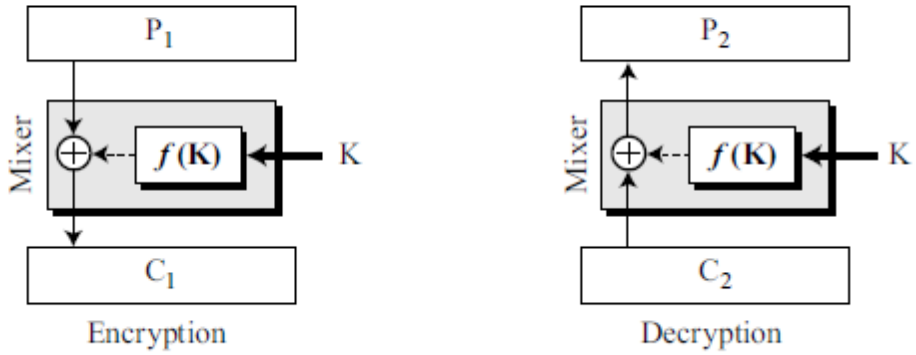
Feistel Ciphers

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.

First Thought:
The effects of a noninvertible component in the encryption algorithm can be canceled in the decryption algorithm if we use an exclusive-or operation.

Figure 5.15 *The first thought in Feistel cipher design*



- 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 in the Feistel design is self-invertible.

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

Decryption: $P_2 = C_2 \oplus f(K) = C_1 \oplus f(K) = P_1 \oplus f(K) \oplus f(K) = P_1 \oplus (00\ldots0) = P_1$

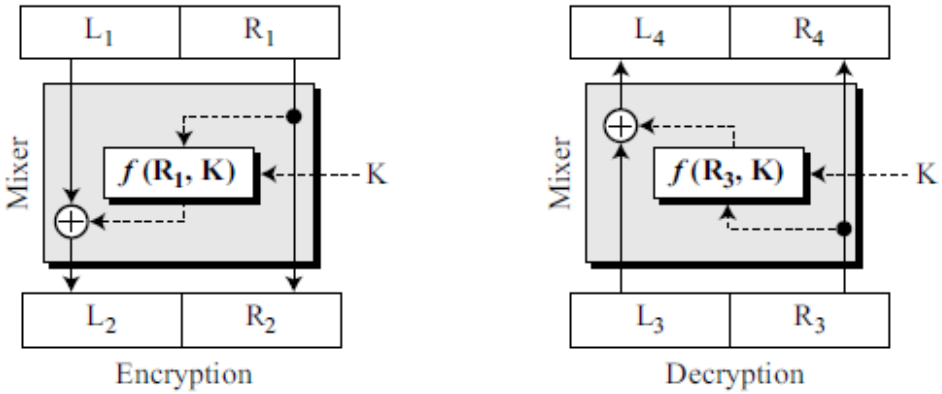
Encryption: $C = P \oplus f(K) = 0111 \oplus 1001 = 1110$

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

Improvement:

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.
the right section of plaintext in the encryption and the right section of the ciphertext in the decryption must be the same.

Figure 5.16 *Improvement of the previous Feistel design*

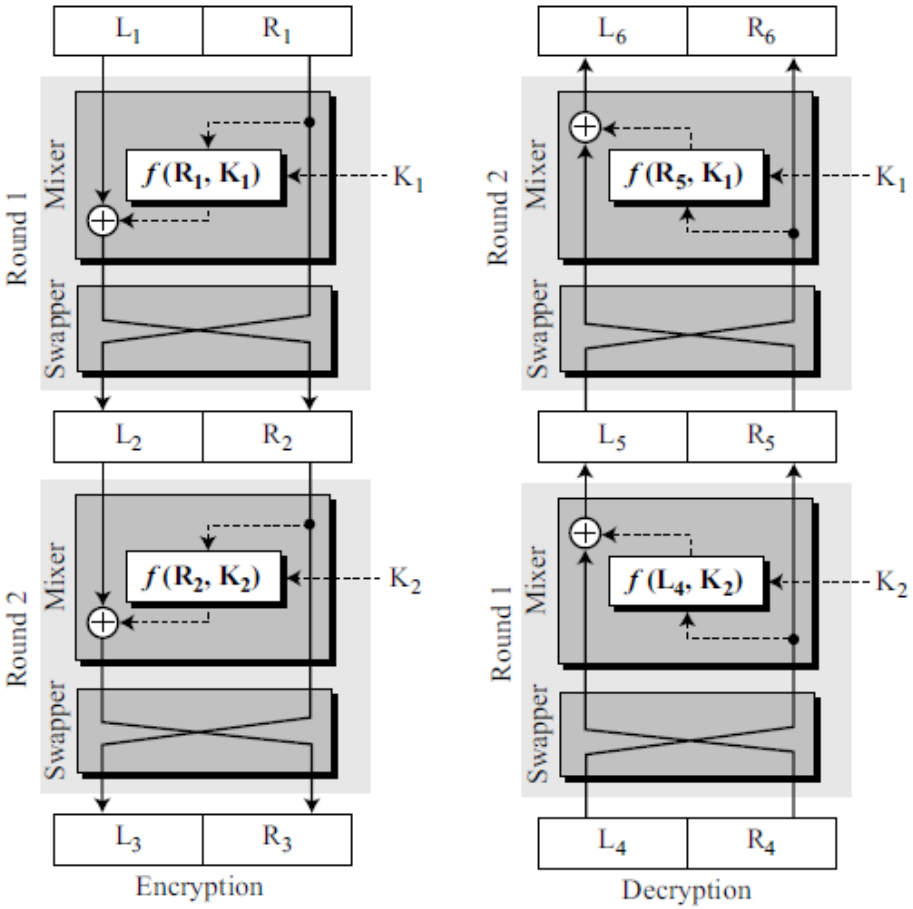


$$R_4 = R_3 = R_2 = R_1$$
$$L_4 = L_3 \oplus f(R_3, K) = L_2 \oplus f(R_2, K) = L_1 \oplus f(R_1, K) \oplus f(R_1, K) = L_1$$

Final Design:

- 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

Figure 5.17 *Final design of a Feistel cipher with two rounds*



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

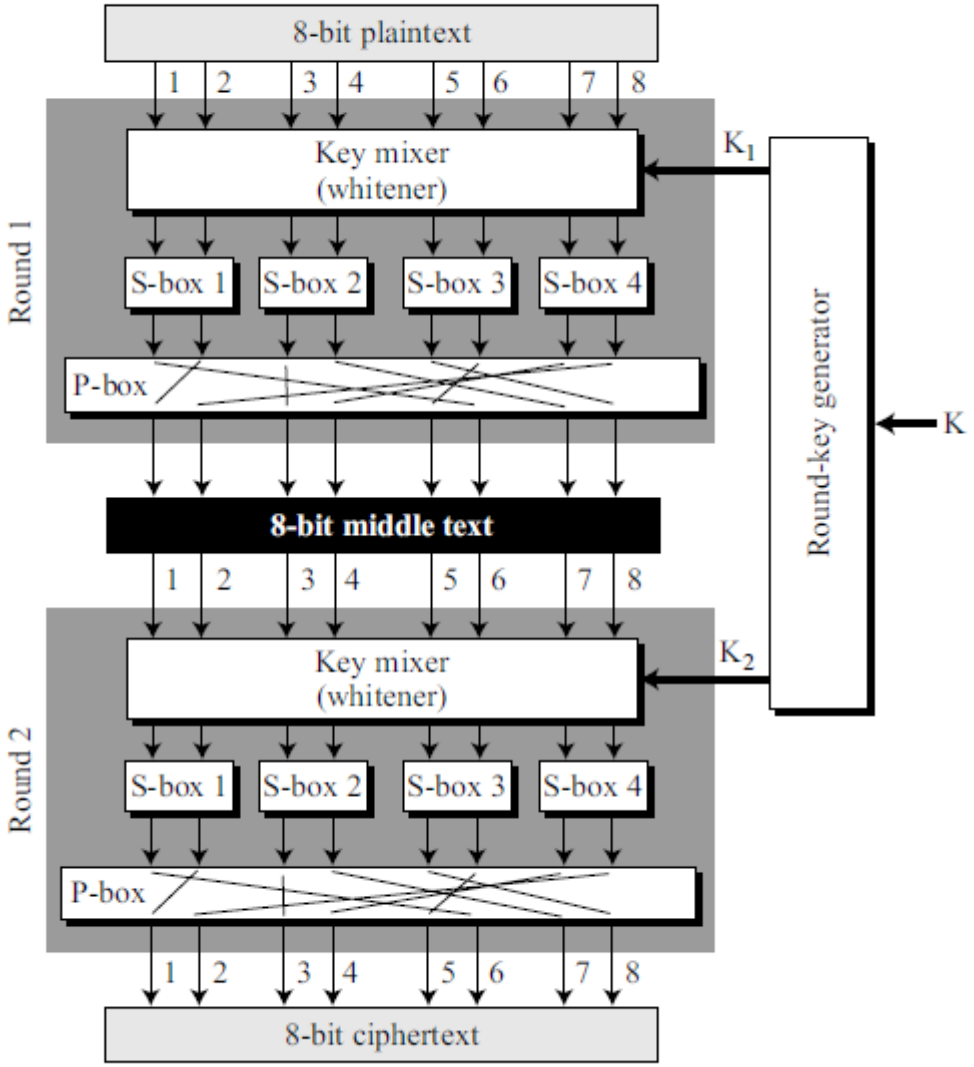
$$L_6 = R_5 \oplus f(L_5, K_1) = R_2 \oplus f(L_2, K_1) = L_1 \oplus f(R_1, K_1) \oplus f(R_1, K_1) = L_1$$
$$R_6 = L_5 = L_2 = R_1$$

Non-Feistel Ciphers:

- A non-Feistel cipher uses only invertible components.
- A component in the encryption cipher has the corresponding component in the decryption cipher.
- S-boxes
- need to have an equal number of inputs and outputs to be compatible.
- No compression or expansion P-boxes are allowed, because they are not invertible.
- In a non-Feistel cipher, there is no need to divide the plaintext into two halves.

example of Non - Feistel Cipher:

Figure 5.13 *A product cipher made of two rounds*

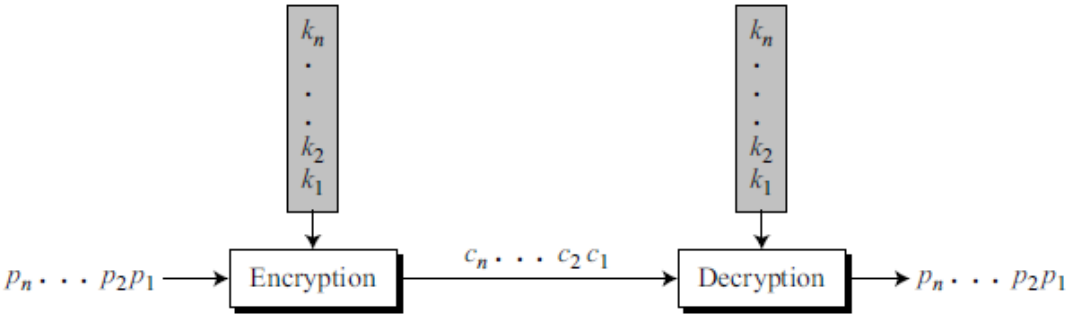


MODERN STREAM CIPHERS

Modern stream cipher, encryption and decryption are done r bits at a time.

We have a plaintext bit stream P = $p_n...p_2p_1$, a ciphertext bit stream C = $c_n...c_2c_1$, and a key bit stream K = $k_n...k_2k_1$, in which p_i , c_i , and k_i are r-bit words. Encryption is $c_i = E(k_i, p_i)$, and decryption is $p_i = D(k_i, c_i)$.

Figure 5.21 *Stream cipher*



Modern stream ciphers are divided into two broad categories:

1. Synchronous
2. Non-synchronous

Synchronous Stream Ciphers:

In a synchronous stream cipher, the key stream is independent of the plaintext or ciphertext stream.

One-Time Pad:

- The simplest and the most secure type of synchronous stream cipher is called the onetime
- pad.
- invented and patented by Gilbert Vernam.
- It is an unbreakable cipher.
- A one-time pad cipher uses a key stream that is randomly chosen for each encipherment.

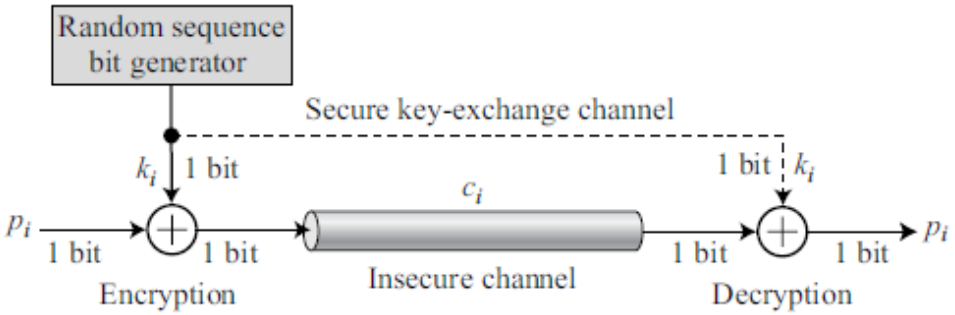
Key characteristics of a one-time pad:

- **Key Length**: The key must be at least as long as the message and completely random.
- **Key Use**: The key is used only once and never repeated.
- **Security**: If the key is truly random, kept secret, and used only once, the ciphertext (encrypted message) is completely unbreakable

How it works:

Each character in the plaintext is XORed with the corresponding character in the random key.

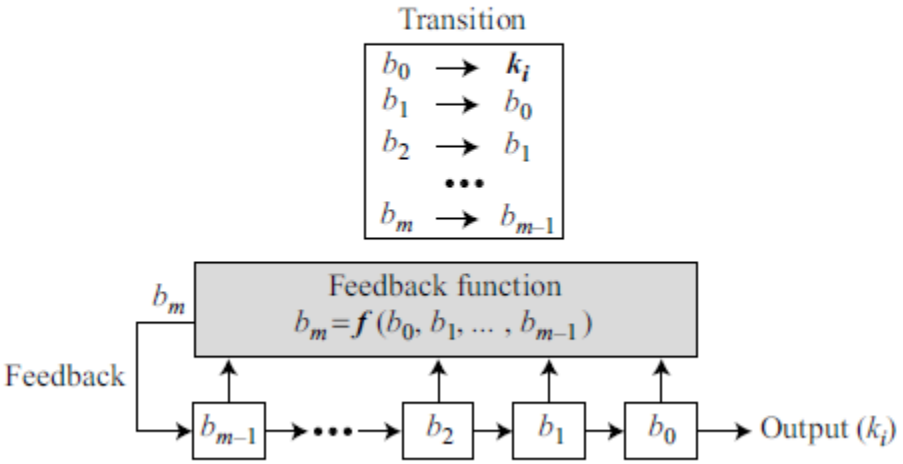
Figure 5.22 *One-time pad*



Feedback Shift Register:

- An FSR can be implemented in either software or hardware.
- A feedback shift register is made of a shift register and a feedback function.

Figure 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.
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.
A feedback shift register two types:

1. linear feedback shift register
2. nonlinear feedback shift register

Linear Feedback Shift Register:

In a linear feedback shift register (LFSR), b_m is a linear function of b_0, b_1, ..., b_{m-1}.

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

Nonlinear Feedback Shift Register:

An NLFSR has the same structure as an LFSR except that the b_m is the nonlinear function of b_0, b_1, ..., b_m.

$$b_4 = (b_3 \text{ AND } b_2) \text{ OR } (b_1 \text{ AND } \overline{b}_0)$$

where AND means bit-wise and operation, OR means bit-wise or operation.

Non-synchronous Stream Ciphers:

In a nonsynchronous stream cipher, each key in the key stream depends on previous plaintext or ciphertext.

Attacks on Block Ciphers:

Also called chosen-plaintext attack.
A **chosen-plaintext attack (CPA)** is a type of cryptanalysis where an attacker can choose arbitrary plaintexts and obtain their corresponding ciphertext.

How It Works

1. **Selection of Plaintext**: The attacker selects one or more plaintext messages.
2. **Encryption Access**: The attacker is able to encrypt these plaintexts using the target encryption algorithm.
3. **Analysis**: The attacker studies the relationship between the plaintexts and the resulting ciphertexts to infer information about the encryption key or algorithm.

Algorithm Analysis:

To find a relationship between the plaintext differences and ciphertext differences without knowing the key.

$$C_1 = P_1 \oplus K \quad C_2 = P_2 \oplus K \quad \rightarrow \quad C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2$$

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.

Guessing the Key Value

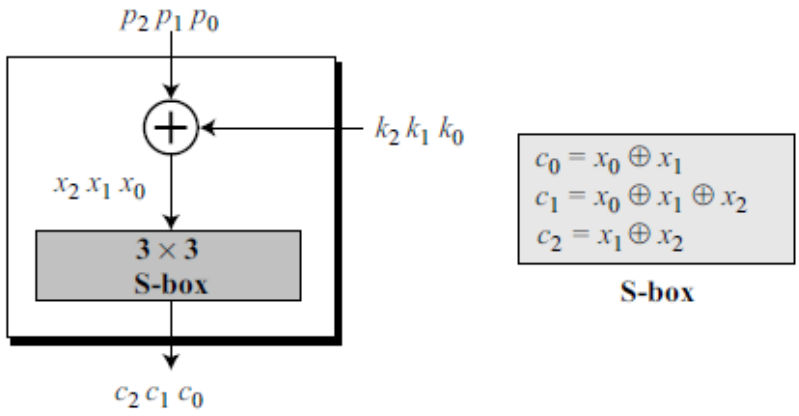
find some plaintext-ciphertext pairs that allow her to guess the value of the key.

$$C_1 = 00 \quad \rightarrow \quad X_1 = 001 \quad \text{or} \quad X_1 = 111$$
$$\text{If } X_1 = 001 \quad \rightarrow \quad K = X_1 \oplus P_1 = 011 \qquad \text{If } X_1 = 111 \quad \rightarrow \quad K = X_1 \oplus P_1 = 101$$

Linear Cryptanalysis:
The cipher is made of a single round, as shown in Figure 5.20,
where c_0, c_1, and c_2 represent the three bits in the output
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:

Figure 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

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

Linear Approximation:

In some modern block ciphers, it may happen that some S-boxes are not totally nonlinear; they can be approximated, probabilistically, by some linear functions. In general, given a cipher with plaintext and ciphertext of n bits and a key of m bits, we are looking for some equations of the form:

$$(k_0 \oplus k_1 \oplus \cdots \oplus k_x) = (p_0 \oplus p_1 \oplus \cdots \oplus p_y) \oplus (c_0 \oplus c_1 \oplus \cdots \oplus c_z)$$

where $1 \le x \le m$, $1 \le y \le n$, and $1 \le z \le n$.

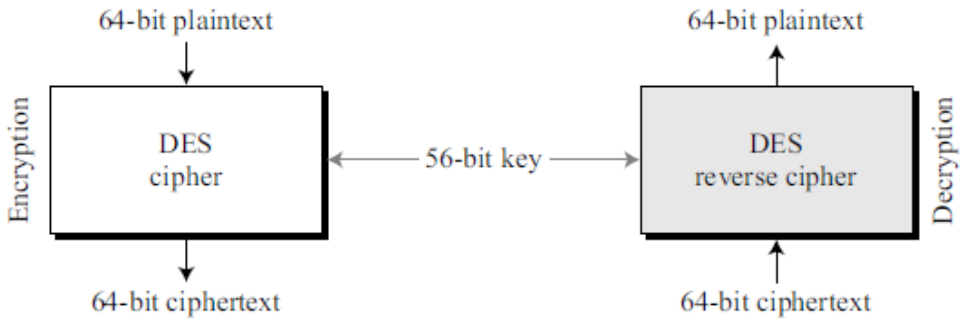
The bits in the intercepted plaintext and ciphertext can be used to find the key bits.

Data Encryption Standard (DES):

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

DES is a block cipher, as shown in Figure 6.1.

Figure 6.1 *Encryption and decryption with DES*

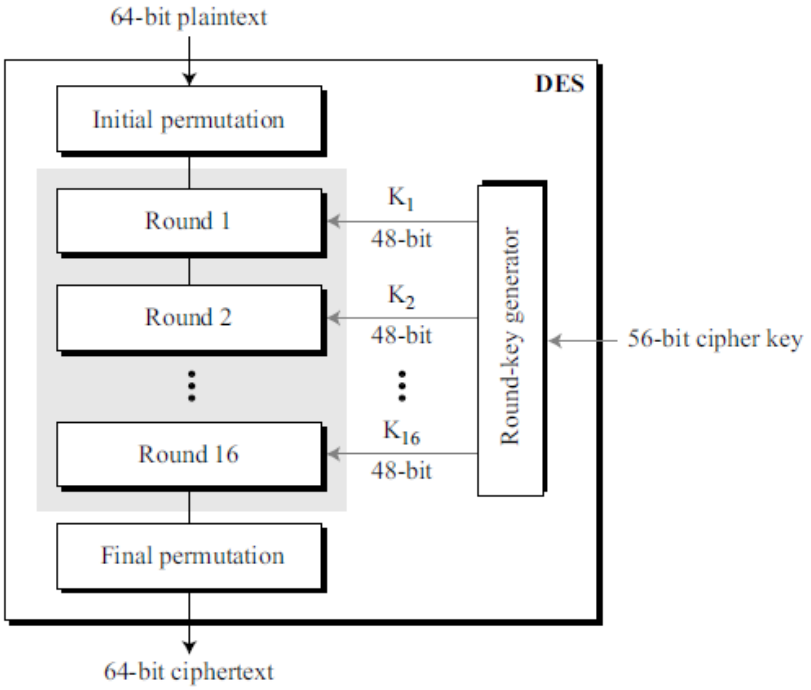


- At the encryption site, DES takes a 64-bit plaintext and creates a 64-bit ciphertext;
- At the decryption site, DES takes a 64-bit ciphertext and creates a 64-bit block of plaintext.

6.2 DES STRUCTURE:

- The encryption process is made of two permutations (P-boxes), which we call initial and final permutations,
- Sixteen Feistel rounds.
- Each round uses a different 48-bit round key generated from the cipher key

Figure 6.2 *General structure of DES*



Initial and Final Permutations:

- 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.

Figure 6.3 *Initial and final permutation steps in DES*

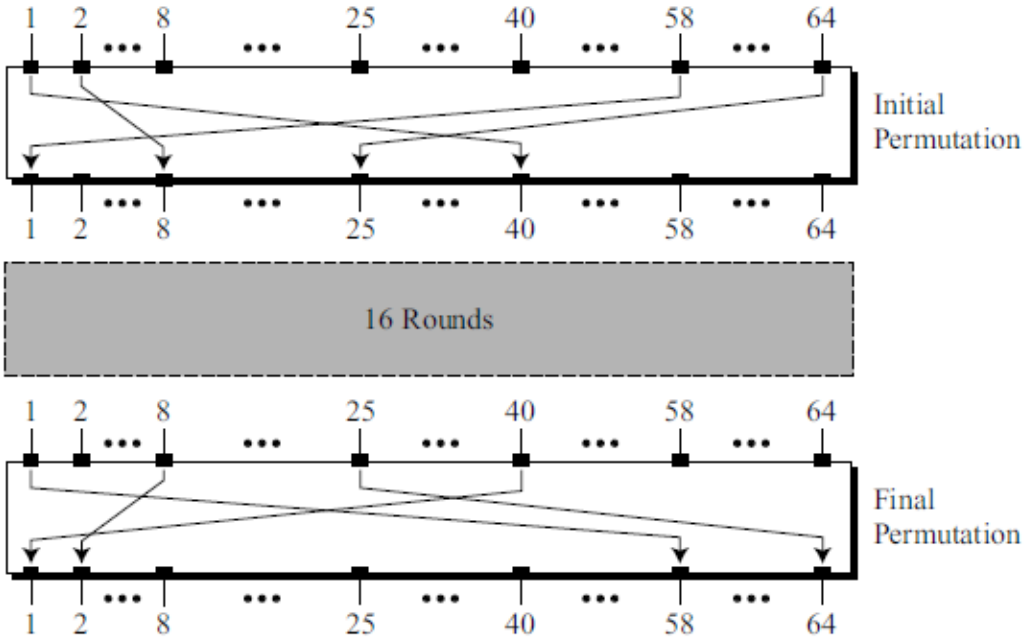


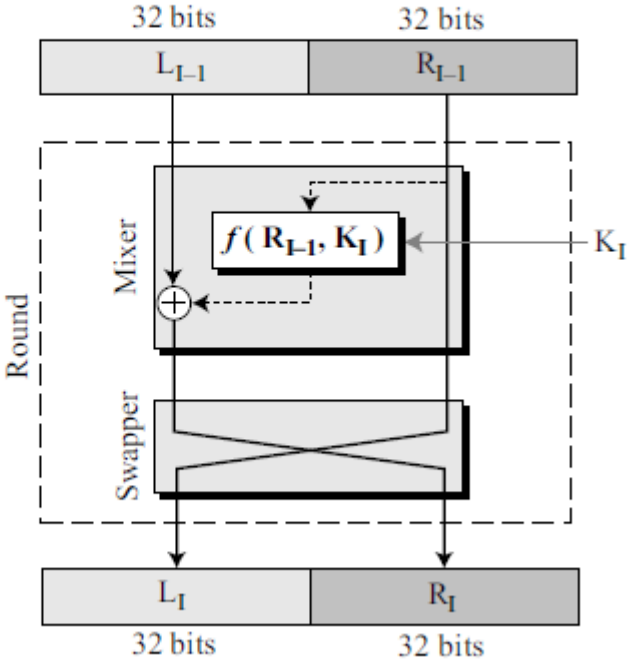
Table 6.1 *Initial and final permutation tables*

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

Rounds:

DES uses 16 rounds. Each round of DES is a Feistel cipher:

Figure 6.4 *A round in DES (encryption site)*



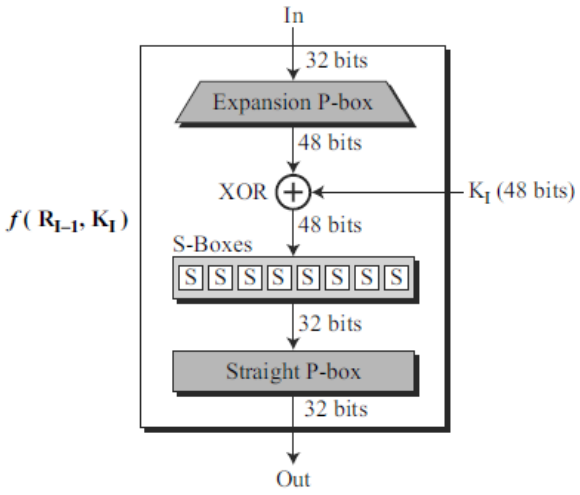
- The round takes L_{I-1} and R_{I-1} from previous round (or the initial permutation box)
- and creates L_I and R_I, which go to the next round (or final permutation box).
- each round has two cipher elements (mixer and swapper). Each of these elements is invertible.
- All noninvertible elements are collected inside the function 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 P-box,
- a whitener (that adds key),
- a group of S-boxes,
- a straightP-box

Figure 6.5 *DES function*



Expansion P-box:
Since R_{I-1} is a 32-bit input and K_I is a 48-bit key, we first need to expand R_{I-1} to 48 bits. R_{I-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.

Figure 6.6 *Expansion permutation*

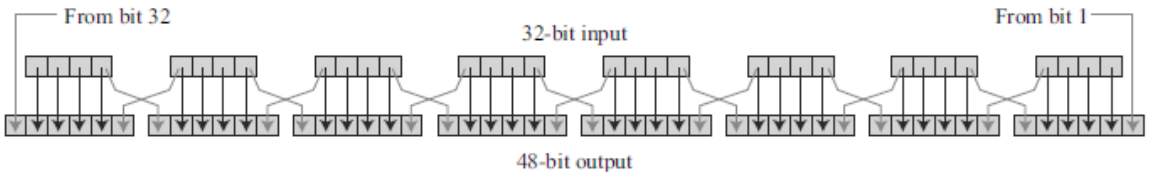
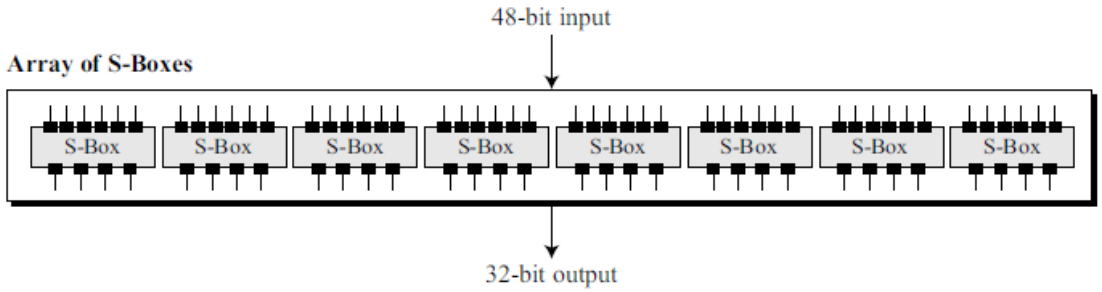


Table 6.2 *Expansion P-box table*

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

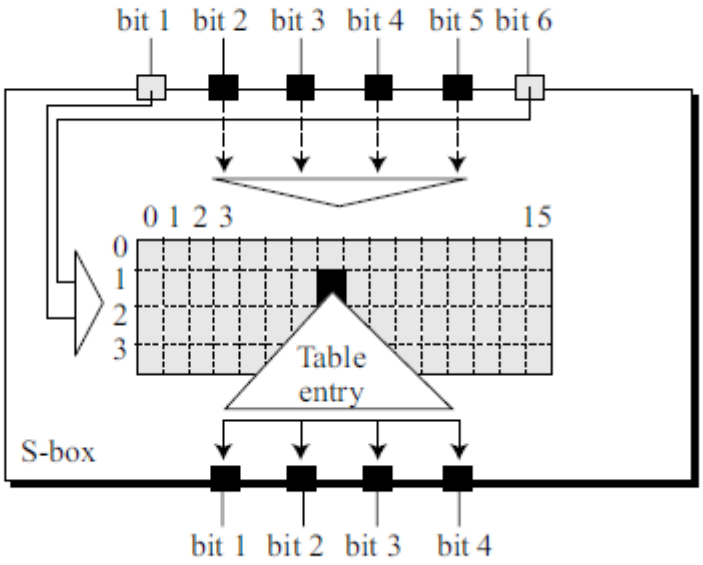
Whitener (XOR):
DES uses the XOR operation on the expanded right section and the round key.
S-Boxes:
The S-boxes do the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output.

Figure 6.7 *S-boxes*



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.

Figure 6.8 *S-box rule*



Because each S-box has its own table, we need eight tables.

Table 6.3 *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

Table 6.4 *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

Table 6.5 *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

Table 6.6 *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

Table 6.7 *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

Table 6.8 *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

Table 6.9 *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

Table 6.10 *S-box 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

Straight Permutation:

The last operation in the DES function is a straight permutation with a 32-bit input and a 32-bit output.

Table 6.11 *Straight permutation table*

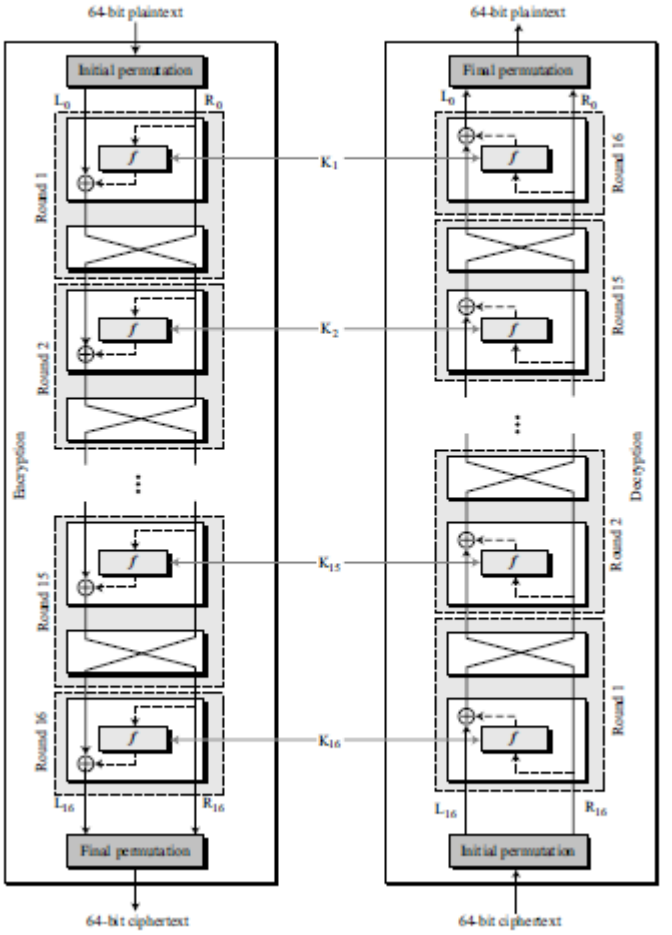
16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Cipher and Reverse Cipher

Using mixers and swappers, we can create the cipher and reverse cipher, each having 16 rounds. The cipher is used at the encryption site; the reverse cipher is used at the decryption site.

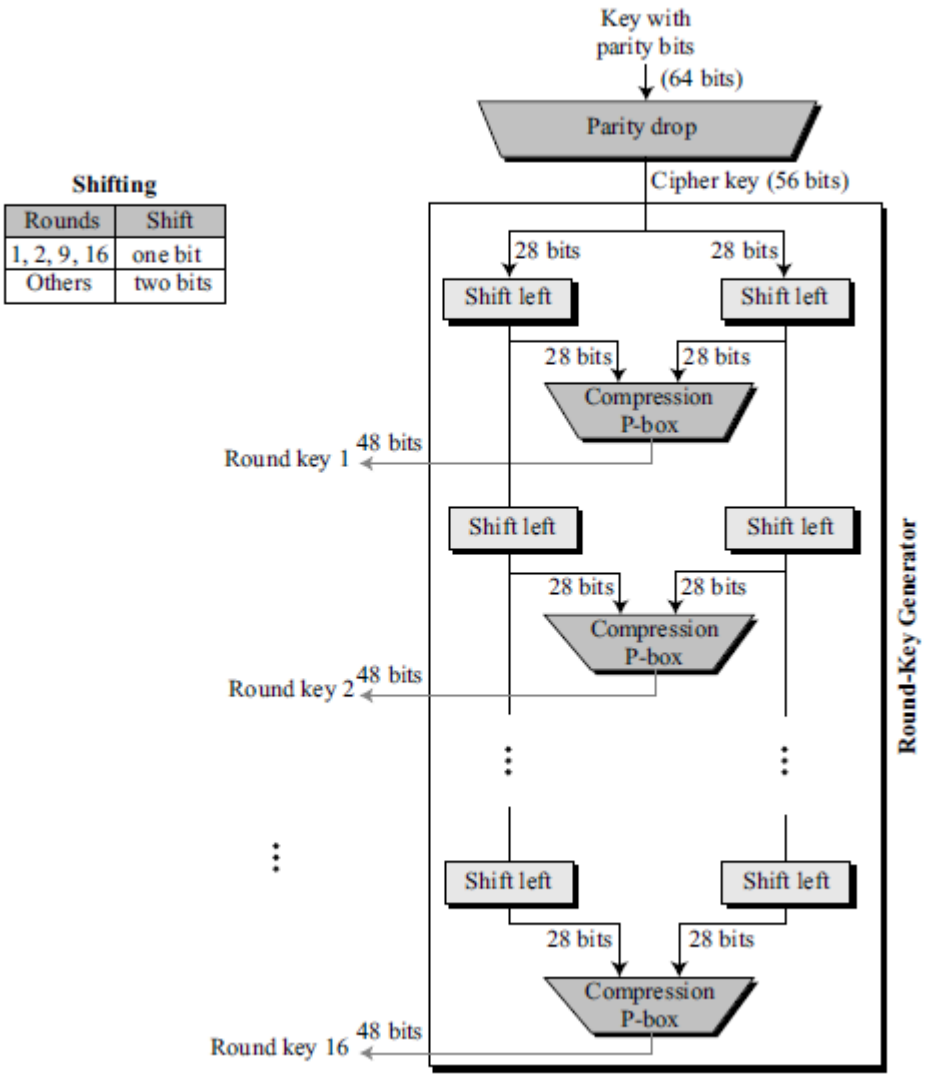
First Approach: one approach is to make the last round (round 16) different from the others; it has only a mixer and no swapper

Figure 6.9 *DES cipher and reverse cipher for the first approach*



Alternative Approach: This is needed to make the last mixer in the cipher and the first mixer in the reverse cipher aligned.

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.



Parity Drop:

The preprocess before key expansion is a compression permutation 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 6.12. The remaining 56-bit value is the actual cipher key which is used to generate round keys.

Table 6.12 *Parity-bit drop 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 *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 Permutation:
The compression permutation (P-box) changes the 58 bits to 48 bits, which are used as a key for a round.

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

MULTIPLE DES: (2DES, 3DES):

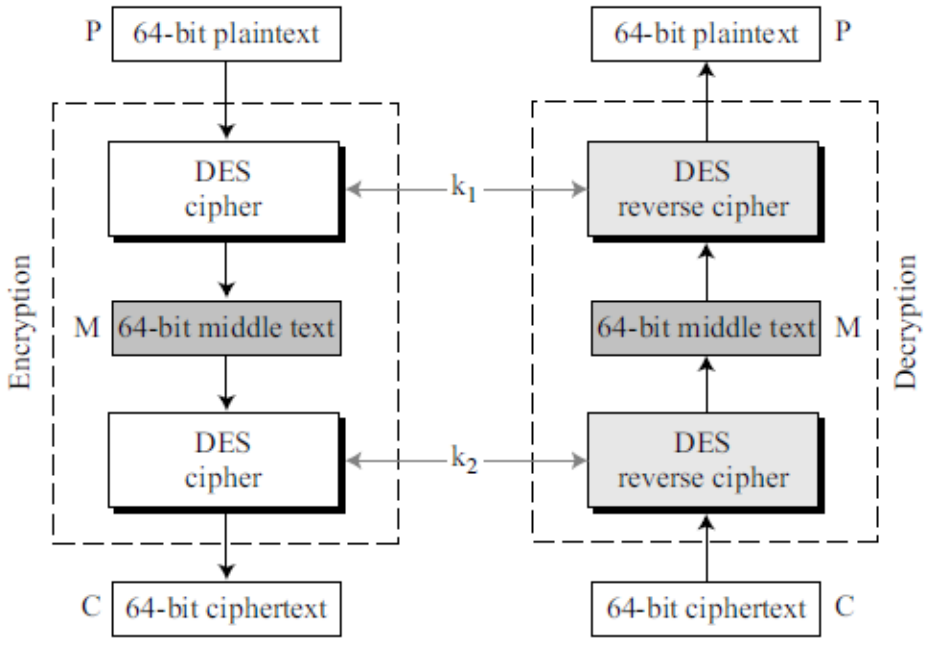
Multiple DES refers to using the **DES algorithm** multiple times with different keys to increase security.

1. Double DES (2DES):

- **Double DES (2DES)** involves applying **DES encryption twice** with **two different keys**.
- The basic process is as follows:

  - **Step 1**: Encrypt the plaintext using the **first key (K1)**.
  - **Step 2**: Encrypt the output of Step 1 using the **second key (K2)**.

Figure 6.14 *Meet-in-the-middle attack for double DES*



Double DES Process:

Let's denote the plaintext as `P` and the ciphertext as `C`.

1. **Encryption** with `K1`: C1=DES(P,K1)
2. **Encryption** with `K2`: C=DES(C1,K2)

So, the encryption process is:

C=DES(DES(P,K1),K2)

Weakness of 2DES:

2DES is vulnerable to a meet-in-the-middle attack.

2. Triple DES (3DES):

To overcome the weakness of 2DES, **Triple DES (3DES)** was introduced. It applies **DES encryption three times** with **three different keys**, and is much more secure than 2DES.

Triple DES Process:

The encryption process with 3DES involves three steps:

1. **Encrypt** the plaintext using the first key (`K1`).
2. **Decrypt** the intermediate result using the second key (`K2`).
3. **Encrypt** the result of the second step using the third key (`K3`).

So, the encryption process is:

C=DES(DES(DES(P,K1),K2),K3

3DES Variants:

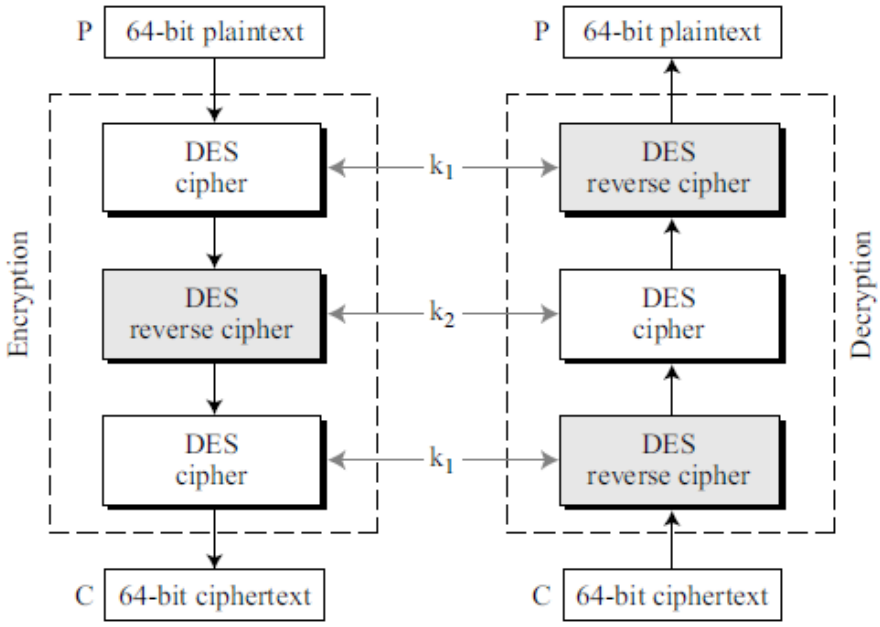
3DES with 2 keys: Sometimes, **K1 = K3**, meaning the first and third keys are the same. This is a **reduced version** of 3DES, but still provides more security than 2DES.

- In this case, the encryption process becomes:

C=DES(DES(DES(P,K1),K2),K1)

3DES with 3 distinct keys: This is the most secure form of 3DES and uses three **independent keys**.

Figure 6.16 *Triple DES with two keys*



Brute-Force Attacks:

A **brute-force attack** is a **cryptographic attack** where the attacker systematically tries every possible key until the correct one is found.

How Brute-Force Attacks Work

1. **Key Exhaustion**: In a brute-force attack, an attacker tries every **possible key** until the correct one is found.
2. **For Block Ciphers**: The attacker may try each key to **decrypt the ciphertext** and check if the result matches some known plaintext (this is known as **known-plaintext attack**).
3. **For Stream Ciphers**: The attacker tries each key to produce the ciphertext and checks for a **valid output**.

Block Ciphers:

- If a cipher uses a **56-bit key** (like DES), a brute-force attack would involve trying all 2562^{56}256 possible keys.
- If a cipher uses a **128-bit key**, there are 21282^{128}2128 possible keys,

Brute-Force Attack Example

Let's take the DES cipher with a 56-bit key:

- The key space consists of 2^{56} possible keys.
- If we use a powerful computer that can try 10^9 (1 billion) keys per second:
  - Time to complete brute-forcing DES:

$$\text{Time} = \frac{2^{56}}{10^9 \,\text{keys/second}} \approx 72 \,\text{quadrillion seconds}$$

Meet-in-the-Middle Attack:

The **Meet-in-the-Middle (MITM) attack** is a cryptographic attack that can be particularly effective against **Double DES (2DES).**

What is the Meet-in-the-Middle Attack?

In a standard **brute-force attack**, you would try every possible key for encryption and decryption, resulting in an exhaustive search of the key space. However, the **Meet-in-the-Middle attack** reduces the complexity by **splitting the encryption process into two stages** and performing a search from both ends simultaneously, meeting in the middle.

Brute-Force Attack on 2DES:

A brute-force attack on 2DES would involve trying every possible key for both `K1` and `K2`. If the key size is 56 bits, then there are 2^{56} possible values for each key. Thus, the brute-force complexity would be $2^{56} \times 2^{56} = 2^{112}$.

Meet-in-the-Middle Attack Steps

The Meet-in-the-Middle attack on Double DES works as follows:

1. Step 1:
   - Generate a list of all possible intermediate ciphertexts by encrypting the plaintext `P` with all possible values of `K1`.
     - For each `K1`, compute:

$$C_1 = DES(P, K1)$$

and store the pairs `(K1, C_1)` in a table.

2. Step 2:
    - For each ciphertext `C`, decrypt it using all possible `K2` values.
        - For each `K2`, compute:

$$P' = DES(C, K2)$$

and check if `P'` matches any `C_1` from the table created in Step 1.

3. Step 3:
    - When you find a match, say `(C_1, P')`, it means that for the key pair `K1` and `K2`, the encryption of `P` with `K1` and `K2` produced the ciphertext `C`.
    - Thus, you have found the correct key pair: `(K1, K2)`.

Known-Plaintext Attack (KPA):

A **Known-Plaintext Attack (KPA)** is a type of cryptographic attack in which the attacker has access to both the **plaintext** (original message) and the corresponding **ciphertext** (encrypted message) for one or more encrypted messages. The goal of the attacker is to use this information to **discover the encryption key** or to break the encryption scheme entirely.

The process generally involves:

1. **Access to Plaintext and Ciphertext**: The attacker has access to both the plaintext and its corresponding ciphertext.
2. **Analysis of the Encryption**: The attacker uses the plaintext-ciphertext pair to analyze the encryption scheme. This could involve trying to reverse-engineer the encryption key or deducing patterns that can help break the encryption.
3. **Key Recovery or Decryption**: If the attacker can find a way to derive the encryption key or the algorithm's internal structure, they can decrypt other ciphertexts encrypted with the same key.

Example of Known-Plaintext Attack on a Block Cipher

Let’s assume the attacker knows the following:

- **Plaintext**: "HELLO"
- **Ciphertext**: "JXNNQ"
- **Cipher**: A block cipher that uses a simple shift pattern (just for illustration).

Steps for a Known-Plaintext Attack:

1. The attacker knows that "HELLO" is encrypted to "JXNNQ".
2. They analyze the relationship between the plaintext and ciphertext. For example:
    - **H → J** (Shifted by 2 positions)
    - **E → X** (Shifted by 19 positions)
    - **L → N** (Shifted by 2 positions)

- **L → N** (Shifted by 2 positions)
- **O → Q** (Shifted by 2 positions)

3. Based on this analysis, the attacker may deduce the pattern of shifting and potentially reverse-engineer the encryption key or algorithm.

DES Weaknesses:

we have found some weaknesses in DES:

Weaknesses in Cipher Design:

S-boxes:

- we have weakness in S-box 4.
- Two specifically chosen inputs to an S-box array can create the same output.
- It is possible to obtain the same output in a single round by changing bits in only three neighbouring S-boxes

P-boxes:

- It is not clear why the designers of DES used the initial and final permutations
- In the expansion permutation (inside the function), the first and fourth bits of every 4-bit series are repeated.

Weakness in the Cipher Key:

1. Short Key Length (56-bit Key) → Vulnerability to Brute-Force Attacks.

2. Weak Keys

Weak Keys (4 total):

- If used, **encryption and decryption produce the same result.**

Keys before parities drop (64 bits)	*Actual key (56 bits)*
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 0E0E 0E0E	0000000 FFFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

Semi-weak Keys:

There are six key pairs that are called semi-weak keys.

Table 6.19 *Semi-weak keys*

First key in the pair	*Second key in the pair*
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E1 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

A semi-weak key creates only two different round keys and each of them is repeated eight times.

Possible Weak Keys:

There are also 48 keys that are called possible weak keys.

The sixteen round keys are divided into four groups and each group is made of four equal round keys.

Advanced Encryption Standard
(AES)

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

Criteria:

The criteria defined by NIST for selecting AES fall into three areas:

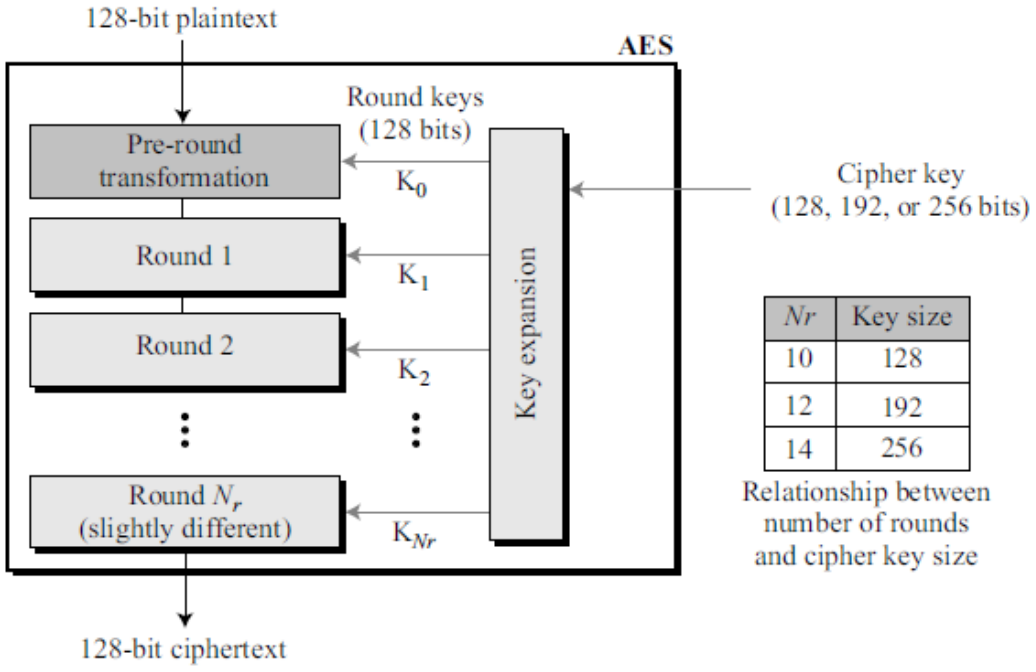
1. security,
2. cost,
3. implementation

Security:
This criterion focused on resistance to cryptanalysis attacks other than brute-force attack.
Cost:
This covers the computational efficiency and storage requirement for different implementations such as hardware, software, or smart cards.
Implementation:
the algorithm must have flexibility (be implementable on any platform) and simplicity.

Rounds:

AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits.
It uses 10, 12, or 14 rounds.
The key size, which can be 128, 192, or 256 bits, depends on the number of rounds.

Figure 7.1 *General design of AES encryption cipher*



Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

- N_r defines the number of rounds
- We can have three different AES versions; they are referred as AES-128, AES-192, and AES-256.
- The round keys, which are created by the key-expansion algorithm are always 128 bits,
- The number of round keys generated by the key-expansion algorithm is always one more than the number of rounds. In other words, we have

Number of round keys = $N_r + 1$

We refer to the round keys as $K_0, K_1, K_2, \ldots, K_{Nr}$

. Data Units:

AES uses five units of measurement to refer to data:

- bits,
- bytes,
- words,
- blocks,
- state

Bit:

- 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
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.
each element of a state is referred to as $s_{r,c}$, 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
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.

Figure 7.2 *Data units used in AES*

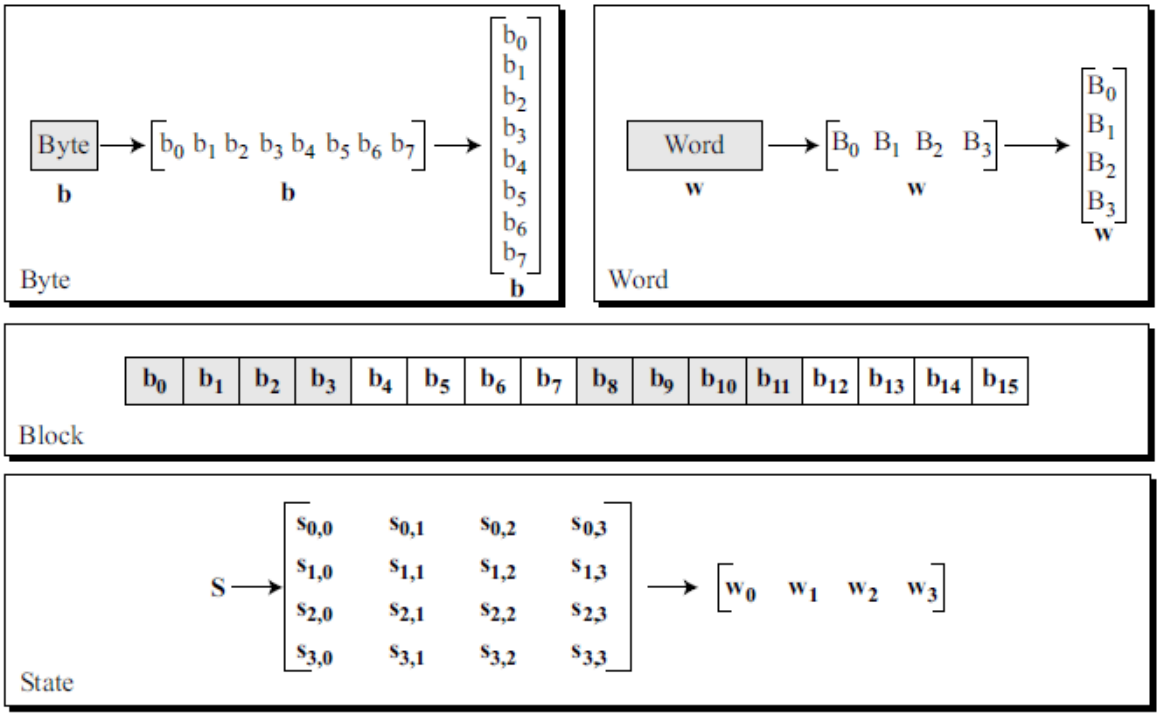


Figure 7.3 *Block-to-state and state-to-block transformation*

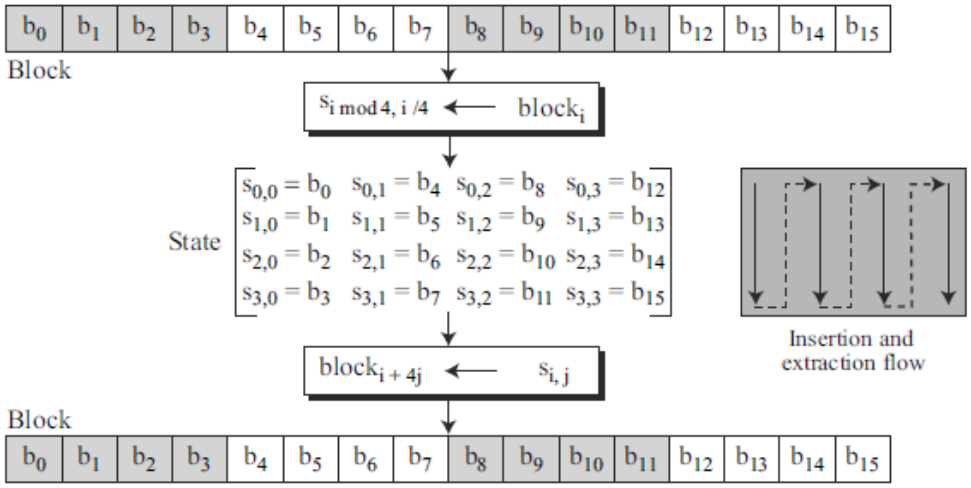
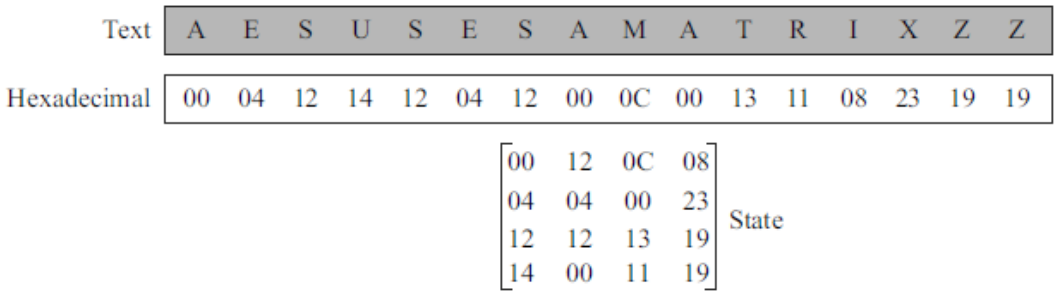


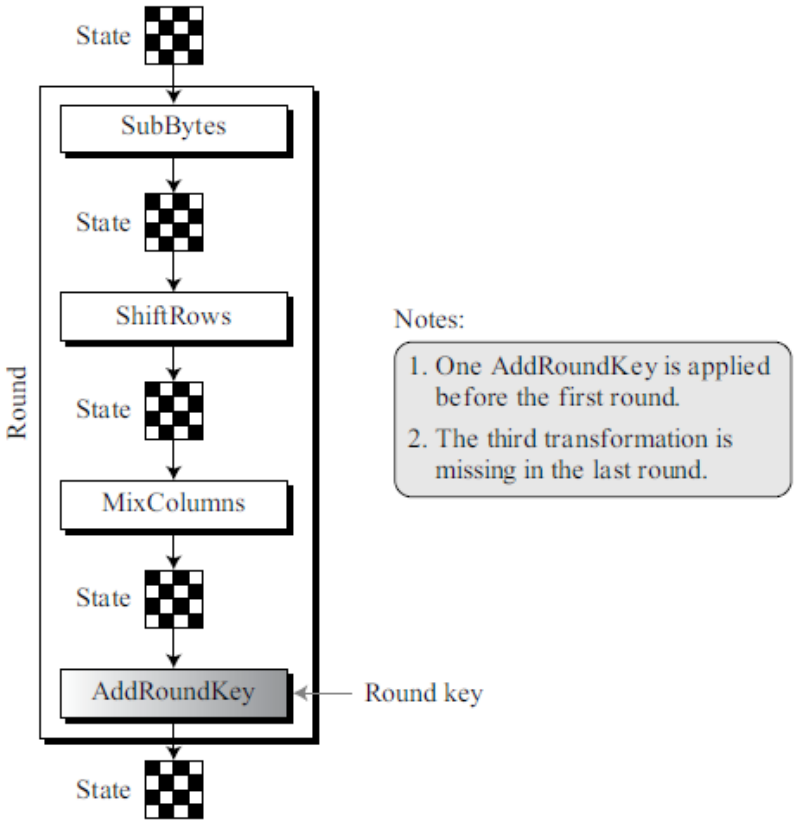
Figure 7.4 *Changing plaintext to state*



Structure of Each Round:

Figure 7.5 shows the structure of each round at the encryption side. Each round, except the last, uses four transformations that are invertible. The last round has only three transformations.
As Figure 7.5 shows, each transformation takes a state and creates another state to be used for the next transformation or the next round. The pre-round section uses only one transformation (AddRoundKey); the last round uses only three transformations (MixColumns transformation is missing).

Figure 7.5 *Structure of each round at the encryption site*



At the decryption site, the inverse transformations are used: InvSubByte, InvShiftRows, InvMixColumns, and AddRoundKey (this one is self-invertible).

TRANSFORMATIONS:

SubBytes:

the transformation is defined by either a table lookup process or mathematical calculation in the GF(2^8) field. AES uses two invertible transformations.

Table lookup process:

The first transformation, SubBytes, is used at the encryption site. To substitute a byte, we interpret the byte as two hexadecimal digits. The left digit defines the row and the right digit defines the column of the substitution table. The two hexadecimal digits at the junction of the row and the column are the new byte.

Figure 7.6 *SubBytes transformations*

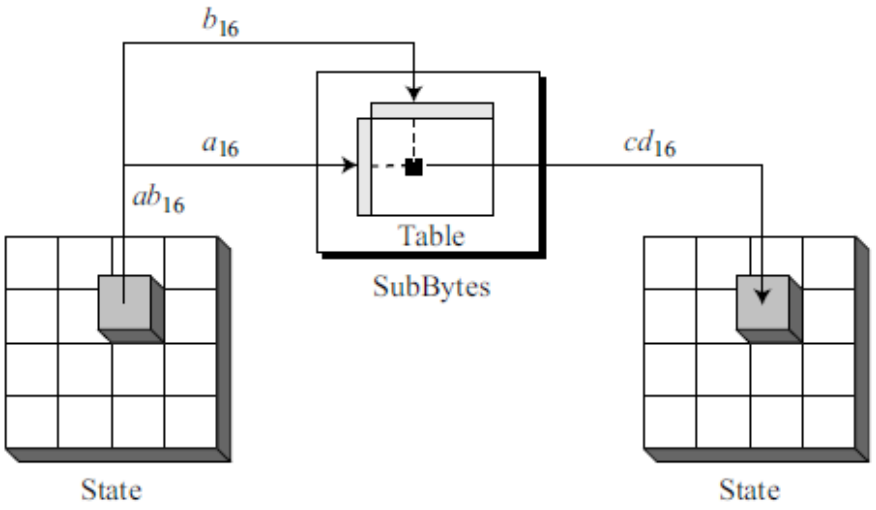


Table 7.1 *SubBytes transformation table*

	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

Table 7.1 *SubBytes transformation table (continued)*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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	CB	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

For example, two bytes, $5A_{16}$ and $5B_{16}$, which differ only in one bit (the rightmost bit) are transformed to BE_{16} and 39_{16}, which differ in four bits.

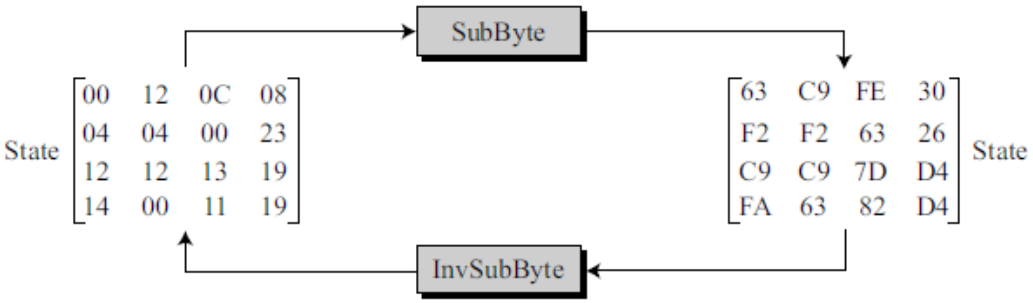
InvSubBytes:

InvSubBytes is the inverse of SubBytes.

Table 7.2 *InvSubBytes transformation table*

	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

Figure 7.7 *SubBytes transformation for Example 7.2*



Transformation Using the GF(2^8) Field:

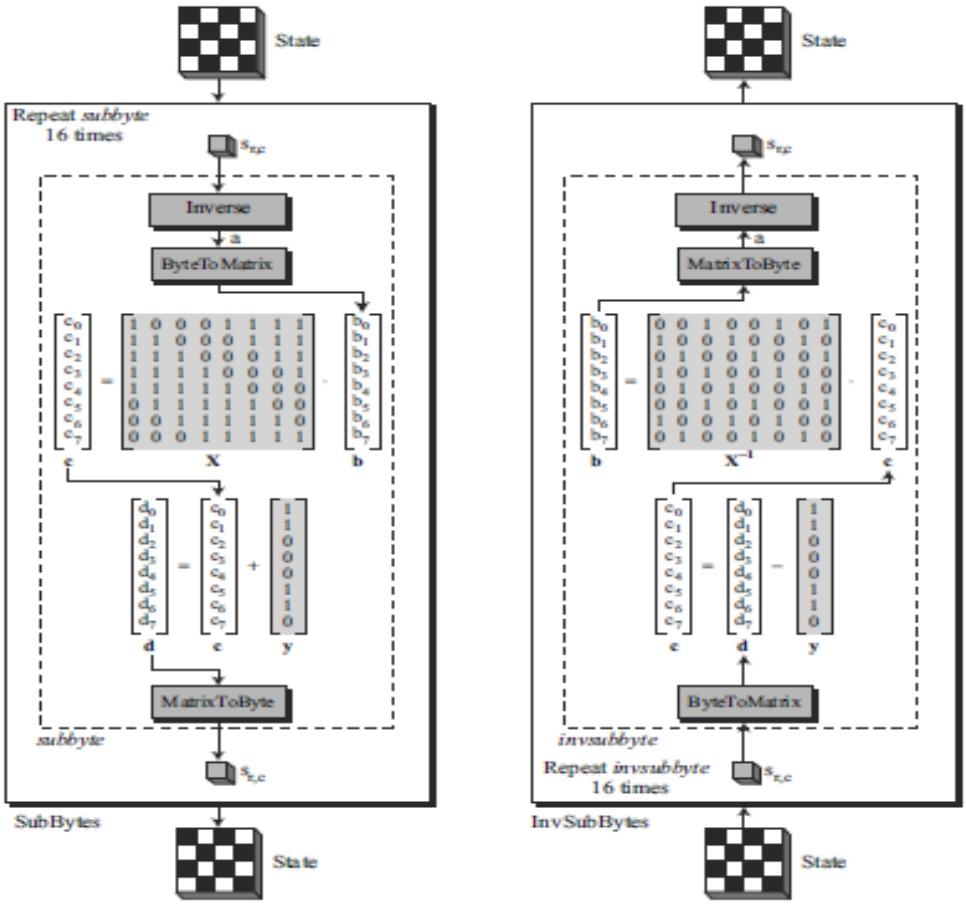
AES also defines the transformation algebraically using the GF(2^8) field with the irreducible polynomials $(x^8 + x^4 + x^3 + x + 1)$.

The SubBytes transformation repeats a routine, called subbyte, sixteen times. The InvSubBytes repeats a routine called invsubbyte. Each iteration transforms one byte.

$$\text{subbyte:} \quad \rightarrow \quad \mathbf{d} = \mathbf{X}\,(s_{r,c})^{-1} \oplus \mathbf{y}$$

$$\text{invsubbyte:} \quad \rightarrow \quad [\mathbf{X}^{-1}(\mathbf{d} \oplus \mathbf{y})]^{-1} = [\mathbf{X}^{-1}(\mathbf{X}\,(s_{r,c})^{-1} \oplus \mathbf{y} \oplus \mathbf{y})]^{-1} = [(s_{r,c})^{-1}]^{-1} = s_{r,c}$$

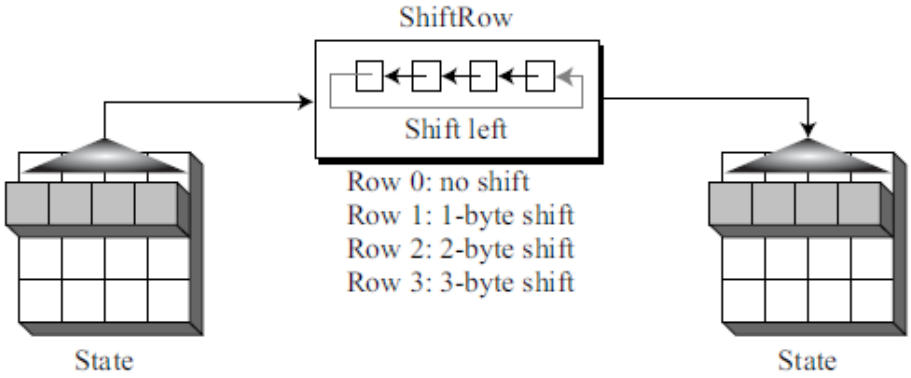
Figure 7.8 *SubBytes and InvSubBytes processes*



ShiftRows:

In the encryption, the transformation is called ShiftRows and the shifting is to the left.The number of shifts depends on the row number (0, 1, 2, or 3) of the state matrix. This means the row 0 is not shifted at all and the last row is shifted three bytes.

Figure 7.9 *ShiftRows transformation*

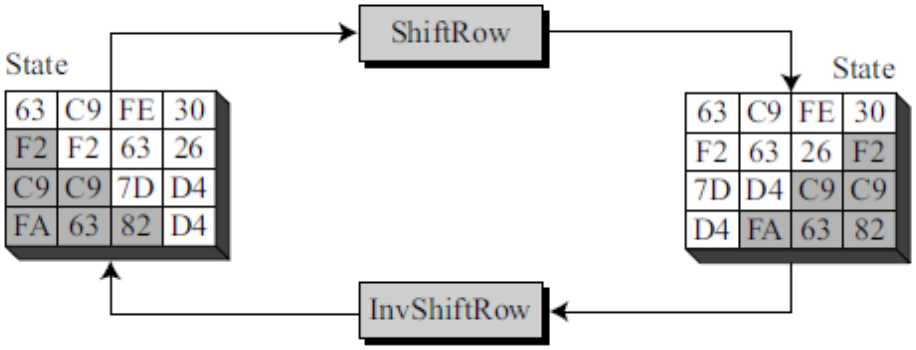


Note that the ShiftRows transformation operates one row at a time.

InvShiftRows:

In the decryption, the transformation is called InvShiftRows and the shifting is to the right. The number of shifts is the same as the row number (0, 1, 2, and 3) of the state matrix.

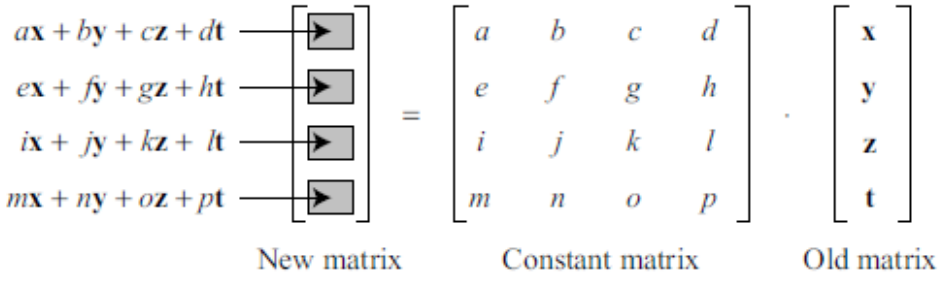
Figure 7.10 *ShiftRows transformation in Example 7.4*



Mixing:

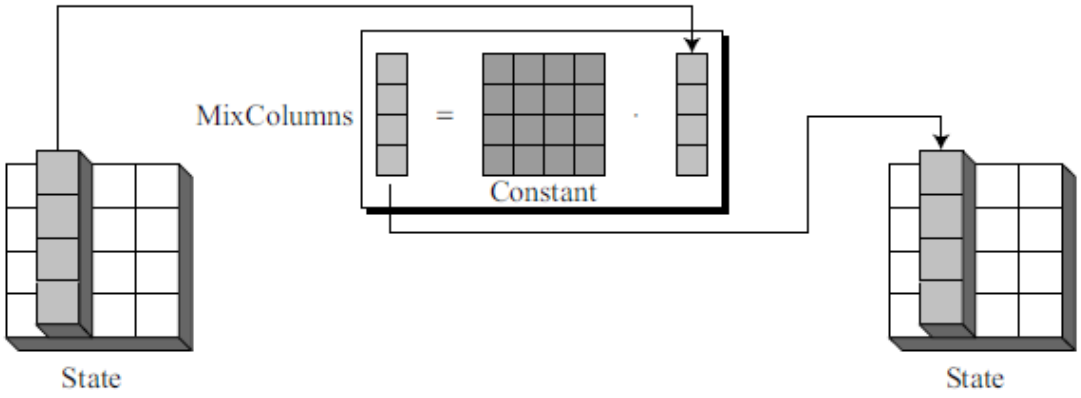
The mixing transformation changes the contents of each byte by taking four bytes at a time and combining them to recreate four new bytes.

Figure 7.11 *Mixing bytes using matrix multiplication*



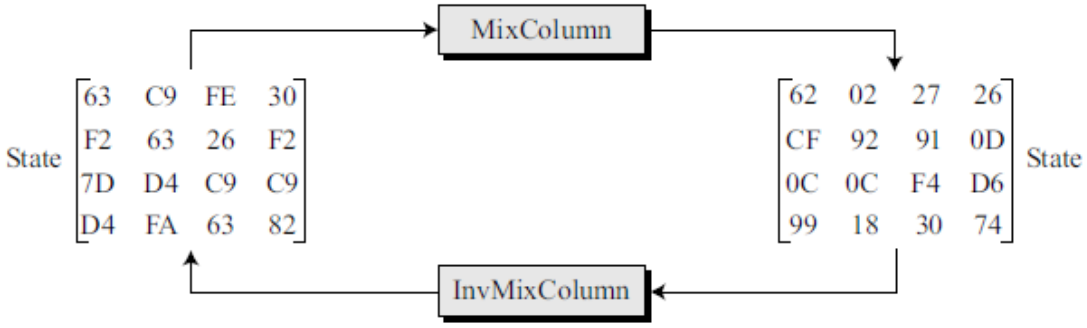
MixColumns:
The MixColumns transformation operates at the column level; it transforms each column of the state to a new column. The transformation is actually the matrix multiplication of a state column by a constant square matrix.

Figure 7.13 *MixColumns transformation*



InvMixColumns:
The InvMixColumns transformation is basically the same as the MixColumns transformation.
If the two constant matrices are inverses of each other,

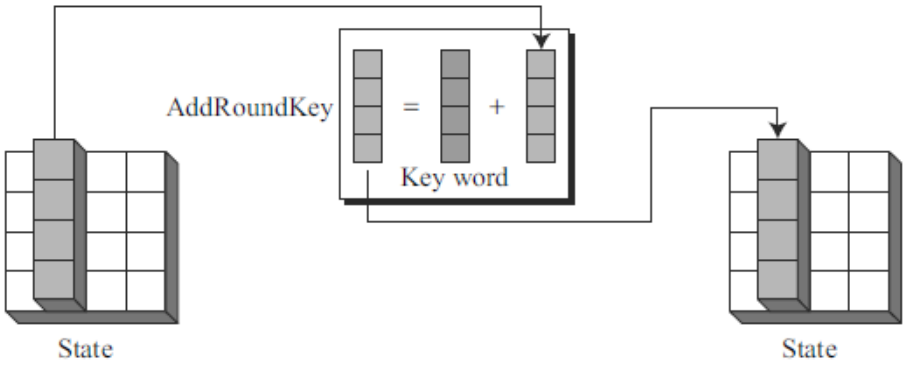
Figure 7.14 *The MixColumns transformation in Example 7.5*



AddRoundKey:

- AddRoundKey adds a round key word with each state column matrix.
- The operation in AddRoundKey is matrix addition.

Figure 7.15 *AddRoundKey transformation*



The AddRoundKey transformation is the inverse of itself.

KEY EXPANSION:

To create round key for each round, AES uses a key-expansion process.
If the number of rounds is N_r , the key-expansion routine creates N_r + 1 128-bit round keys from one single 128-bit cipher key.
The first round key is used for pre-round transformation (AddRoundKey); the remaining round keys are used for the last transformation (AddRoundKey) at the end of each round.
The key-expansion routine creates round keys word by word, where a word is an array of four bytes.
The routine creates 4 × (N_r +1) words that are called

$$\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{4(N_r+1)-1}$$

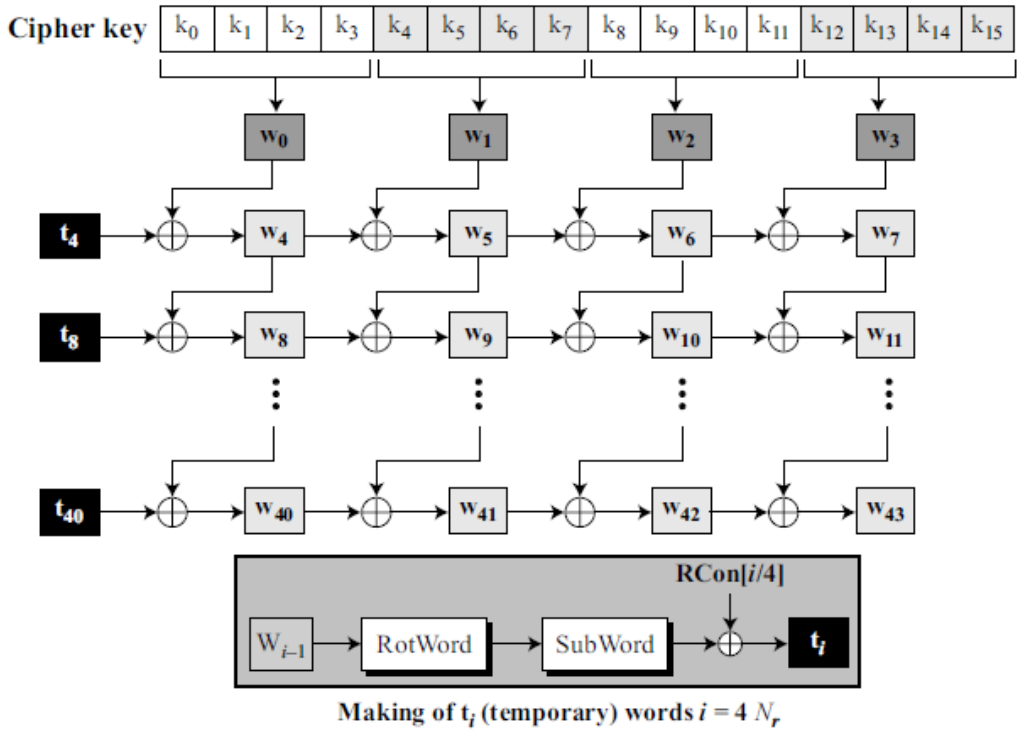
Table 7.3 *Words for each round*

Round	*Words*			
Pre-round	$\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
1	$\mathbf{w}_4$	$\mathbf{w}_5$	$\mathbf{w}_6$	$\mathbf{w}_7$
2	$\mathbf{w}_8$	$\mathbf{w}_9$	$\mathbf{w}_{10}$	$\mathbf{w}_{11}$
. . .	. . .			
N_r	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

Key Expansion in AES-128:

The AES-128 version (10 rounds), there are 44 words.

Figure 7.16 *Key expansion in AES*



The process is as follows:

1. The first four words (w_0, w_1, w_2, w_3) are made from the cipher key. The cipher key is thought of as an array of 16 bytes (k_0 to k_{15}). The first four bytes (k_0 to k_3) become w_0; the next four bytes (k_4 to k_7) become w_1; and so on.
2. The rest of the words (w_i for i = 4 to 43) are made as follows:
   - If (i mod 4) ≠ 0, $w_i = w_{i-1} \oplus w_{i-4}$. Referring to Figure 7.16, this means each word is made from the one at the left and the one at the top.
   - If (i mod 4) = 0, $w_i = t \oplus w_{i-4}$. Here t, a temporary word, is the result of applying two routines, SubWord and RotWord, on w_{i-1} and XORing the result with a round constants, RCon. In other words.

$$\mathbf{t} = \text{SubWord}\,(\text{RotWord}\,(\mathbf{w}_{i-1})) \oplus \text{RCon}_{i/4}$$

RotWord:
The RotWord (rotate word) routine is similar to the ShiftRows transformation, but it is applied to only one row.

SubWord:
The SubWord (substitute word) routine is similar to the SubBytes transformation, but it is applied only to four bytes.

Round Constants:
Each round constant, RCon, is a 4-byte value in which the rightmost three bytes are always zero.

Table 7.4 *RCon constants*

Round	*Constant (RCon)*	*Round*	*Constant (RCon)*
1	$(\underline{\mathbf{01}}\ 00\ 00\ 00)_{16}$	6	$(\underline{\mathbf{20}}\ 00\ 00\ 00)_{16}$
2	$(\underline{\mathbf{02}}\ 00\ 00\ 00)_{16}$	7	$(\underline{\mathbf{40}}\ 00\ 00\ 00)_{16}$
3	$(\underline{\mathbf{04}}\ 00\ 00\ 00)_{16}$	8	$(\underline{\mathbf{80}}\ 00\ 00\ 00)_{16}$
4	$(\underline{\mathbf{08}}\ 00\ 00\ 00)_{16}$	9	$(\underline{\mathbf{1B}}\ 00\ 00\ 00)_{16}$
5	$(\underline{\mathbf{10}}\ 00\ 00\ 00)_{16}$	10	$(\underline{\mathbf{36}}\ 00\ 00\ 00)_{16}$

Key Expansion in AES-192 and AES-256:

Key-expansion algorithms in the AES-192 and AES-256 versions are very similar to the key expansion algorithm in AES-128, with the following differences:

AES-192:

AES- 192 version (12 rounds), there are 52 words;

1. In AES-192, the words are generated in groups of six instead of four.
   a. The cipher key creates the first six words ($\mathbf{w}_0$ to $\mathbf{w}_5$).
   b. If $i \bmod 6 \neq 0$, $\mathbf{w}_i \leftarrow \mathbf{w}_{i-1} + \mathbf{w}_{i-6}$; otherwise, $\mathbf{w}_i \leftarrow \mathbf{t} + \mathbf{w}_{i-6}$.

AES-256:

the AES-256 version (with 14 rounds), there are 60 words.

2. In AES-256, the words are generated in groups of eight instead of four.
   a. The cipher key creates the first eight words ($\mathbf{w}_0$ to $\mathbf{w}_7$).
   b. If $i \bmod 8 \neq 0$, $\mathbf{w}_i \leftarrow \mathbf{w}_{i-1} + \mathbf{w}_{i-8}$; otherwise, $\mathbf{w}_i \leftarrow \mathbf{t} + \mathbf{w}_{i-8}$.
   c. If $i \bmod 4 = 0$, but $i \bmod 8 \neq 0$, then $\mathbf{w}_i = \text{SubWord}(\mathbf{w}_{i-1}) + \mathbf{w}_{i-8}$.

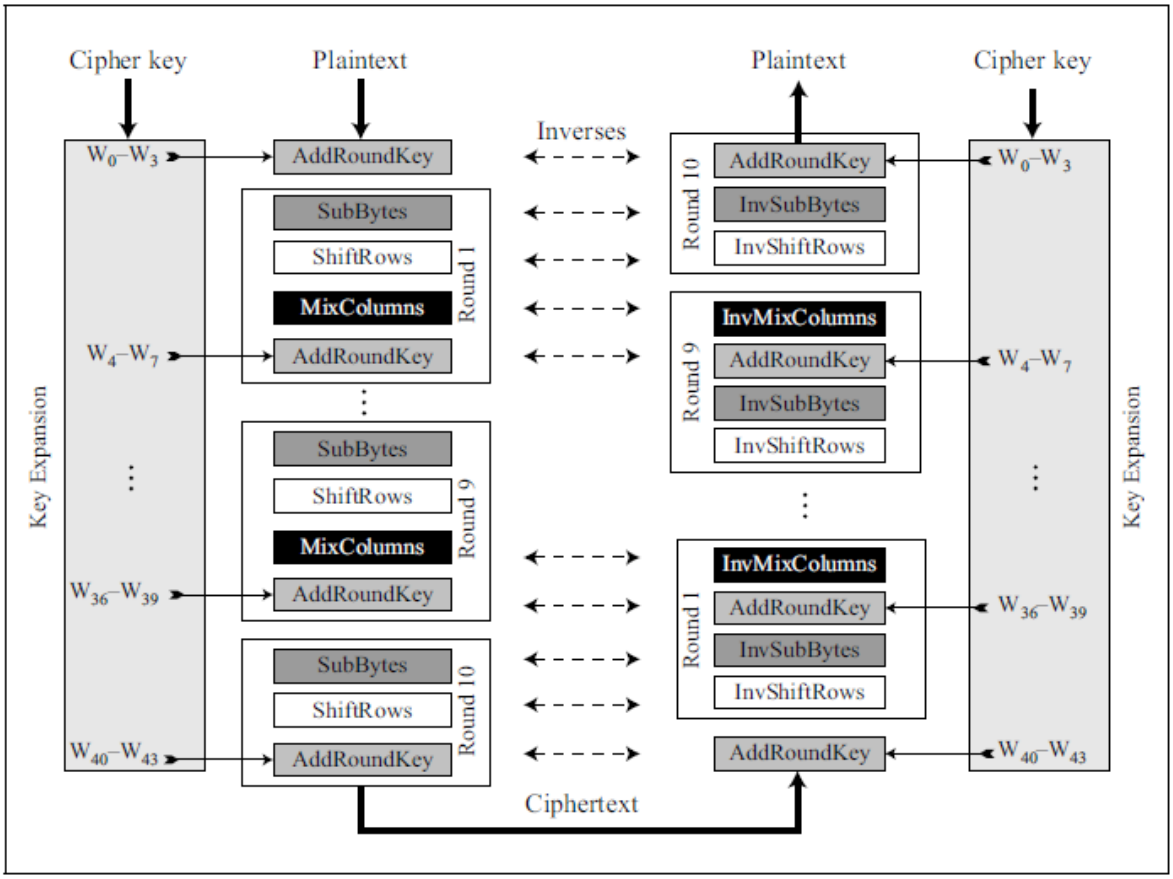
CIPHERS

AES uses four types of transformations for encryption and decryption.
In the standard, the encryption algorithm is referred to as the cipher and the decryption algorithm as the inverse cipher.
AES is a non-Feistel cipher, which means that each transformation or group of transformations must be invertible.

Original Design:

In the original design, the order of transformations in each round is not the same in the cipher and reverse cipher.

Figure 7.17 *Cipher and inverse cipher of the original design*

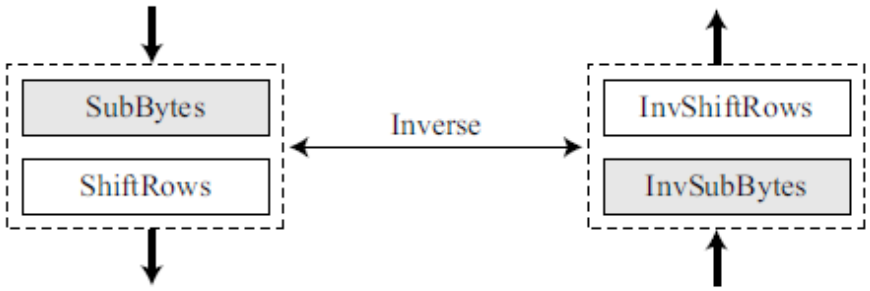


First, the order of SubBytes and ShiftRows is changed in the reverse cipher.
Second, the order of MixColumns and AddRoundKey is changed in the reverse cipher. This difference in ordering is needed to make each transformation in the cipher aligned with its inverse in the reverse cipher.

Alternative Design:

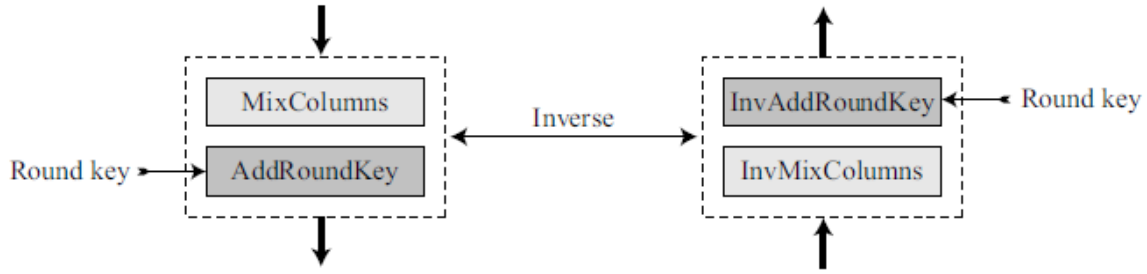
SubBytes/ShiftRows Pairs:
SubBytes change the contents of each byte without changing the order of the bytes in the state;
ShiftRows change the order of the bytes in the state without changing the contents of the bytes

Figure 7.18 *Invertibility of SubBytes and ShiftRows combinations*



MixColumns/AddRoundKey Pair:
the pairs can become inverses of each other if we multiply the key matrix by the inverse of the constant matrix used in MixColumns transformation. We call the new transformation InvAddRoundKey

Figure 7.19 *Invertibility of MixColumns and AddRoundKey combinations*



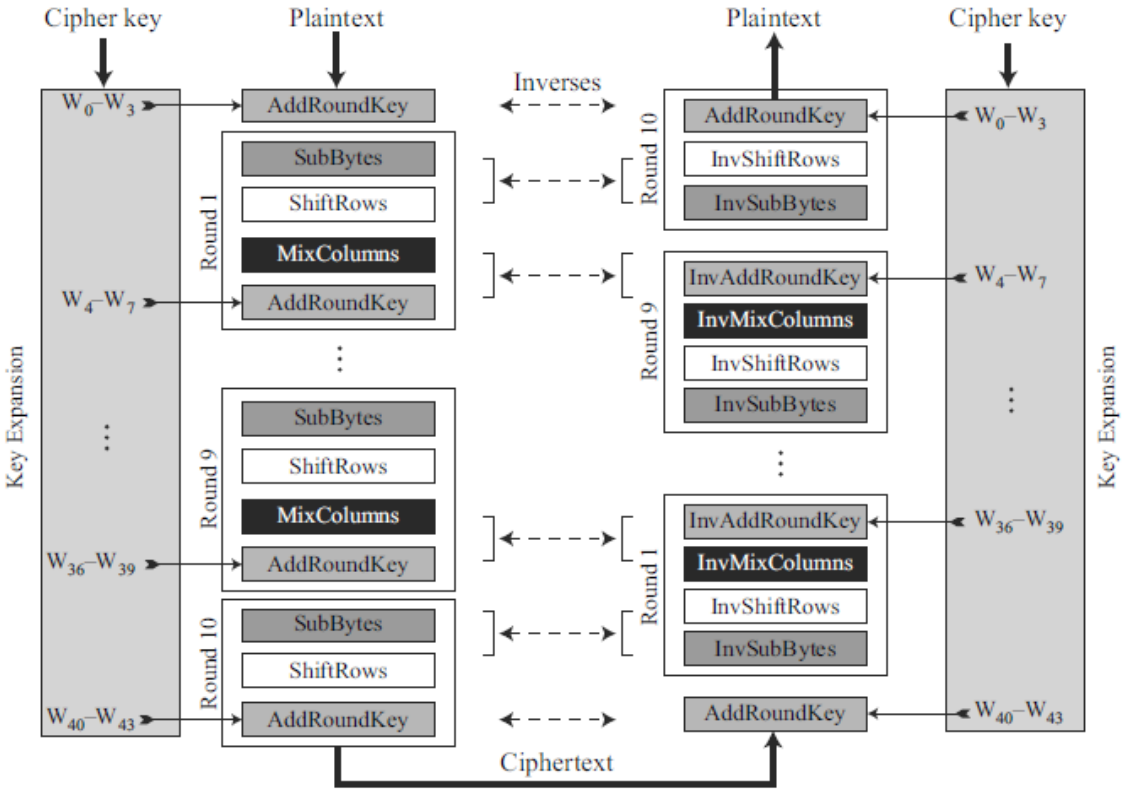
Cipher: $T = CS \oplus K$
Inverse Cipher: $C^{-1}T \oplus C^{-1}K = C^{-1}(CS \oplus K) \oplus C^{-1}K = C^{-1}CS \oplus C^{-1}K \oplus C^{-1}K = S$

the input state to the combination S and the output state T

Changing Key-Expansion Algorithm:

Instead of using InvRoundKey transformation in the reverse cipher, the key-expansion algorithm can be changed to create a different set of round keys for the inverse cipher.

Figure 7.20 *Cipher and reverse cipher in alternate design*



Attacks on AES:

- Brute-Force Attack
  Refer Brute Force Attack.
- Statistical Attacks
  The strong diffusion and confusion provided by the combination of the SubBytes, ShiftRows, and MixColumns transformations removes any frequency pattern in the

plaintext.

- Differential and Linear Attacks refer DES Attacks.