Unit2

Symmetric Ciphers

Contents of Unit 2

- Fiestel Cipher Structure, Substitution Permutation Network (SPN)
- Data Encryption Standards (DES), Double DES, Triple DES
- Finite Fields: Groups Rings, Fields, Modular Arithmetic, Euclidean Algorithm, Galois Fields (GF(p) & GF(2n)), Polynomial Arithmetic
- International Data Encryption Standard (IDEA)
- Advanced Encryption Standards (AES) Cipher
- Modes of Block Cipher Encryptions (Electronic Code Book, Cipher Block Chaining, Cipher Feedback Mode, Output Feedback Mode, Counter Mode)

Substitution Permutation Network (SPN)

- Claude Shannon introduced idea of substitution-permutation (S-P) networks in 1949 paper
- form basis of modern block ciphers
- S-P nets are based on the two primitive

➢ cryptographic operations seen before:

  - *substitution* (S-box)
  - *permutation* (P-box)

➢ provide *confusion* & *diffusion* of message & key

SPN

- **Substitution:** Each plaintext element or group of elements is uniquely replaced by a corresponding cipher text element or group of elements.
- **Permutation:** A sequence of plaintext elements is replaced by a permutation of that sequence.

That is, no elements are added or deleted or replaced in the sequence, rather the order in which the elements appear in the sequence is changed.

Confusion and Diffusion

- Claude Shannon, in one of the fundamental papers on the theoretical foundations of cryptography ["Communication theory of secrecy systems," Bell Systems Technical Journal 28 (1949), 656 – 715], gave two properties that a good cryptosystem should have to hinder statistical analysis:
  - diffusion and confusion

Confusion

- Confusion means that each binary digit (bit) of the ciphertext should depend on several parts of the key, obscuring the connections between the two.
- The property of confusion hides the relationship between the ciphertext and the key.
- This property makes it difficult to find the key from the ciphertext and if a single bit in a key is changed, the calculation of most or all of the bits in the ciphertext will be affected.

Confusion contd..

- In substitution–permutation networks, confusion is provided by substitution boxes.

Diffusion

- Diffusion means that if we change a single bit of the plaintext, then about half of the bits in the ciphertext should change, and similarly, if we change one bit of the ciphertext, then about half of the plaintext bits should change.
- This is equivalent to the expectation that encryption schemes exhibit an avalanche effect.
- The purpose of diffusion is to hide the statistical relationship between the ciphertext and the plain text.
- In substitution–permutation networks, diffusion is provided by permutation boxes.

DIFFUSION AND CONFUSION

- In **diffusion**, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext. This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally, this is equivalent to having each ciphertext digit be affected by many plaintext digits.
- An example of diffusion is to encrypt a message
- M

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

adding successive letters to get a ciphertext letter yn .

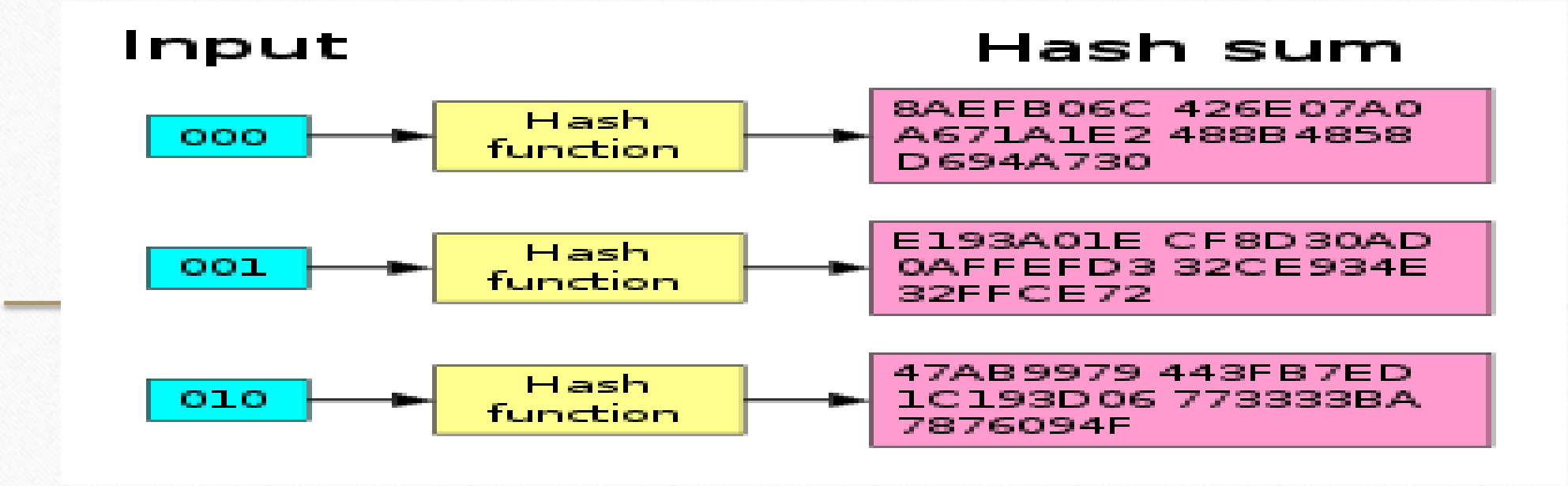
DIFFUSION AND CONFUSION contd..

- The mechanism of diffusion seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible in order to thwart attempts to deduce the key.

On the other hand, **confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, again to thwart attempts to discover the key. Thus, even if the attacker can get some handle on the statistics of the ciphertext, the way in which the key was used to produce that ciphertext is so complex as to make it difficult to deduce the key. This is achieved by the use of a complex substitution algorithm.

Avalanche effect

- The **avalanche effect** is the desirable property of cryptographic algorithms, typically block ciphers and cryptographic hash functions, wherein if an input is changed slightly (for example, flipping a single bit), the output changes significantly (e.g., half the output bits flip).
- In the case of high-quality block ciphers, such a small change in either the key or the plaintext should cause a drastic change in the ciphertext.
- The actual term was first used by Horst Feistel,although the concept dates back to at least Shannon's *diffusion*.



Example : Avalanche effect

The SHA-1 hash function exhibits good avalanche effect.

Reversible and irreversible block ciphers

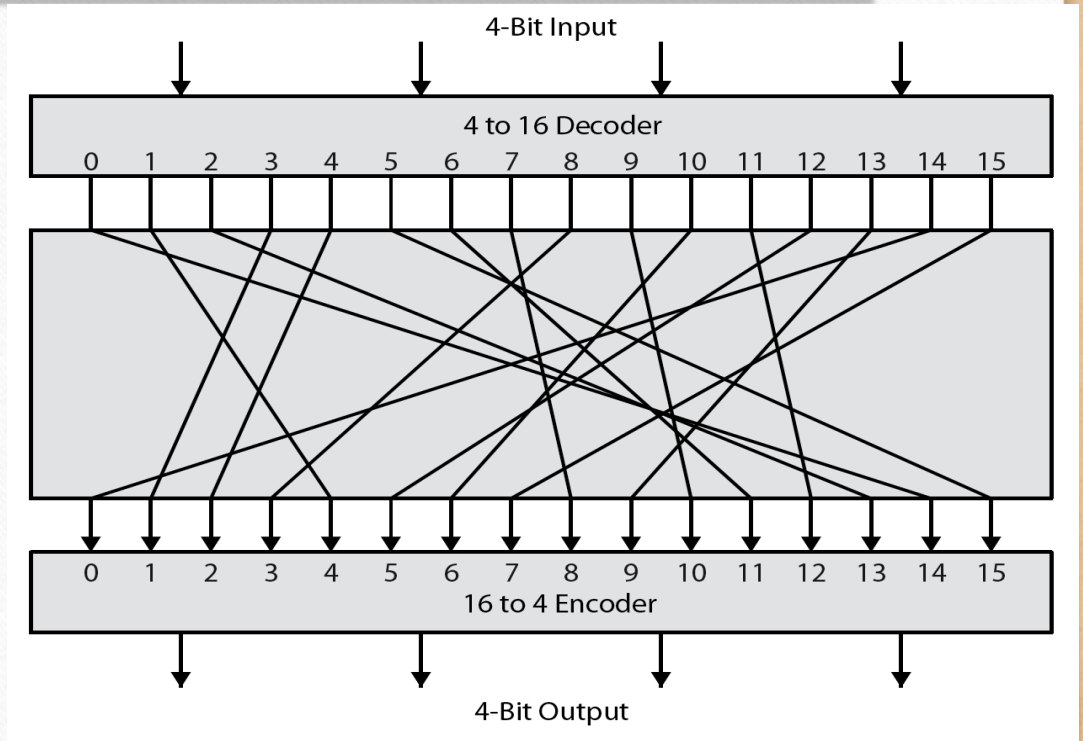
- A block cipher operates on a plaintext block of *n* bits to produce a cipher text block of *n* bits. There are possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique cipher text block. Such transformation is called reversible, or nonsingular.
- In the latter case, a cipher text of 01 could have been produced by one of two plaintext blocks. So if we limit ourselves to reversible mappings, the number of different transformations is n!

Reversible and Irreversible Mapping

Reversible Mapping		Irreversible Mapping	
Plaintext	**Cipher text**	**Plaintext**	**Cipher text**
00	11	00	11
01	10	01	10
10	01	10	01
11	00	11	01

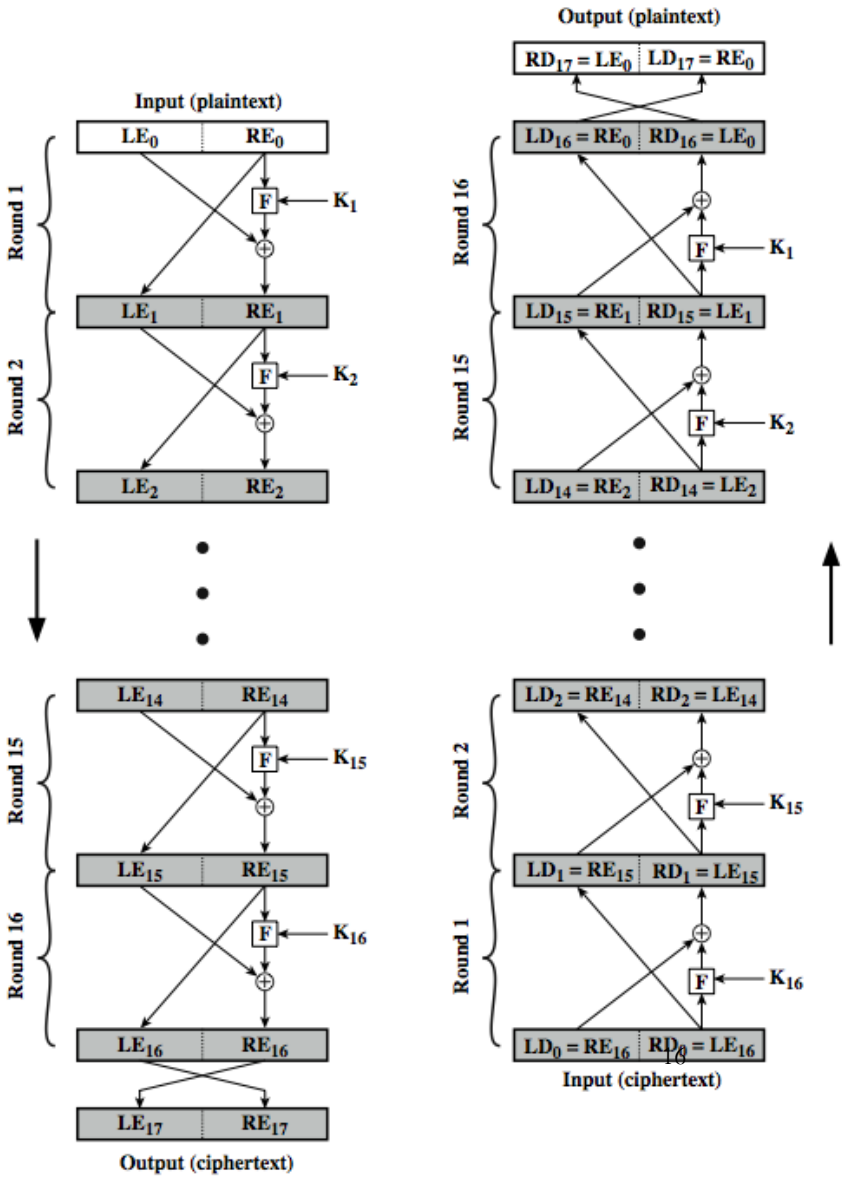
Ideal Block Cipher

- In an ideal block cipher, the relationship between the input blocks and the output block is completely random. But it must be invertible for decryption to work. Therefore, it has to be one-to- one, meaning that each input block is mapped to a unique output block.
- Therefore, it has to be one-to-one, meaning that each input block is mapped to a unique output block.
- Figure depicts an ideal block cipher that uses blocks of size 4. Each block of 4 bits in the plaintext is transformed into a block of 4 ciphertext bits.
- It illustrates a tiny 4-bit substitution to show that each possible input can be arbitrarily mapped to any output - which is why its complexity grows so rapidly.

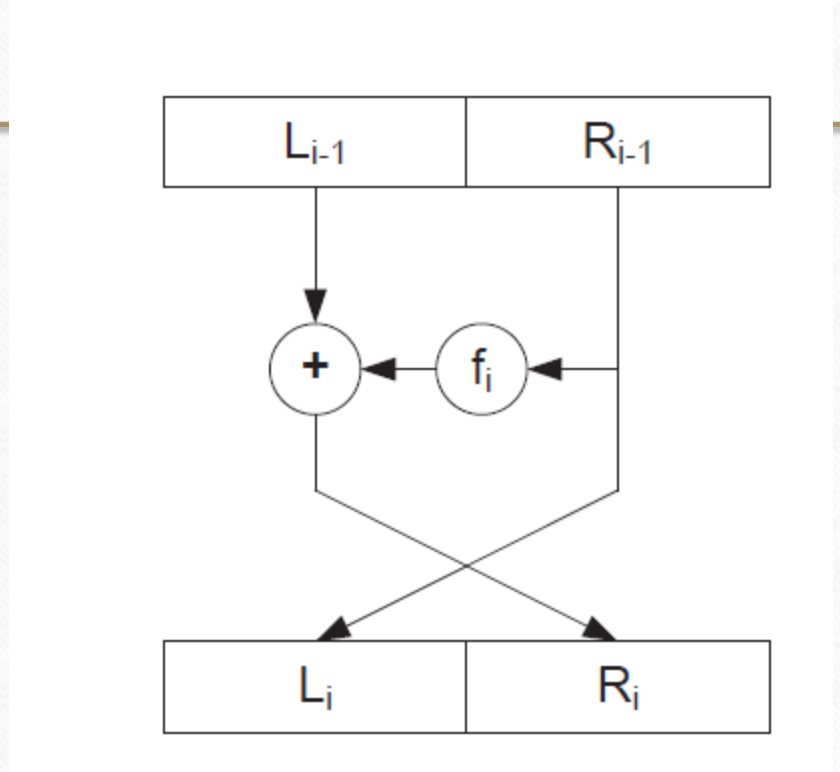


Feistel Cipher structure

- Horst Feistel devised the F**eistel cipher**
  - based on concept of invertible product cipher
- partitions input block into two halves
  - process through multiple rounds which
  - perform a substitution on left data half
  - based on round function of right half & subkey
  - then have permutation swapping halves
- implements Shannon's S-P net concept

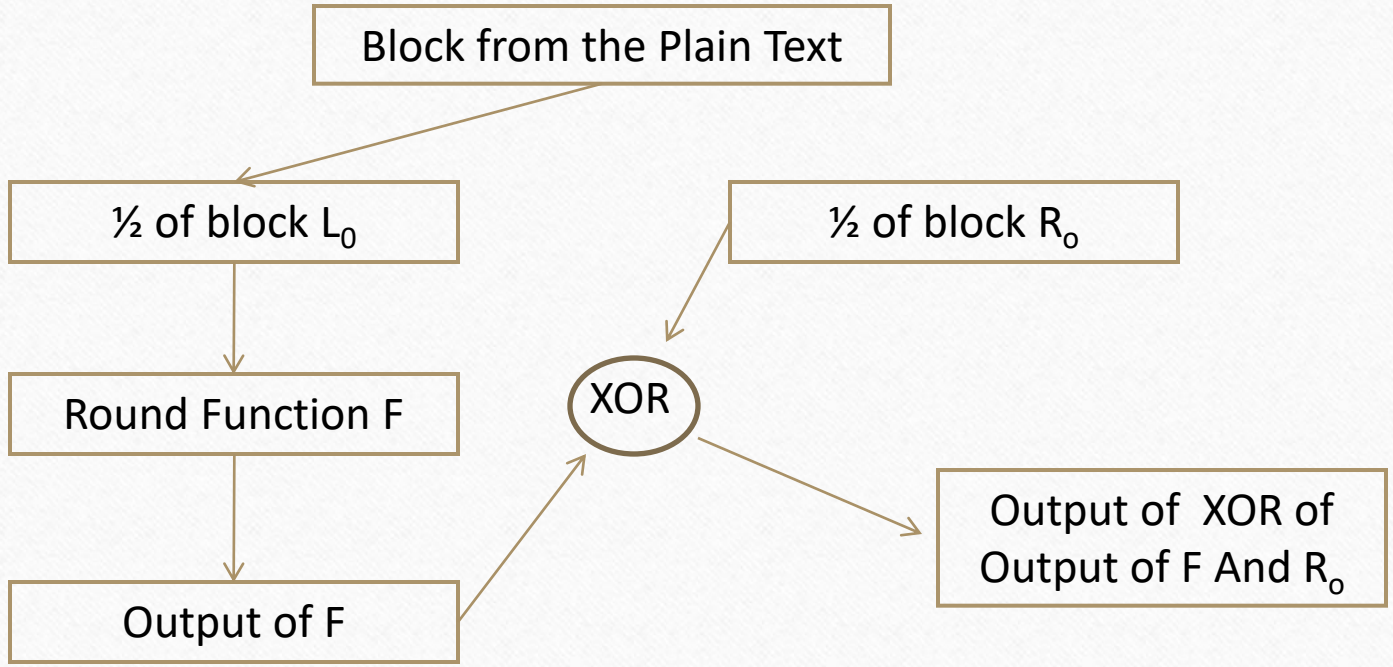


A simple view of a single Round



The Feistel Function - Continued

- Here is a general overview of a basic round of a Feistel cipher.



Feistel Cipher

Symbolically,

- n sequential rounds
- A substitution on the left half L_i
  - 1. Apply a round function F to the right half R_i and
  - 2. Take XOR of the output of (1) and L_i
- The round function is parameterized by the subkey K_i
  - K_i are derived from the overall key K

The Feistel Function-Continued

This function starts by splitting the bock of plain text data (often 64 bits) into two parts (traditionally termed L_0 and R_0)

The round function F is applied to 1 of the halves. The term 'round function' simply means a function performed with each iteration, or round, of the Feistel cipher. The details of the round function F can vary with different implementations. Usually these are relatively simple functions, to allow for increased speed of the algorithm.

The output of each round function F is then xor'd with the other half. What this means is that, for example, you take $L_{0,}$ pass it through the round function F, then take the result and xor it with $R_{0.}$

Then the halves are transposed. So L_0 gets moved to the right and R_0 gets moved to the left.

This process is repeated a given number of times. The main difference between cryptography algorithms is the exact nature of the round function F, and the number of iterations.

Mathematical Description of Each Round in the Feistel Structure Cipher Decryption

- Let LEi and REi denote the output half-blocks at the end of the i th round of processing. The letter 'E' denotes encryption.
- In the Feistel structure, the relationship between the output of the i th round and the output of the previous round, that is, the (i − 1)th round, is given by

$$
\begin{aligned}
LE_i &= RE_{i-1} \\
RE_i &= LE_{i-1} \oplus F(RE_{i-1}, K_i)
\end{aligned}
$$

Parameters and design features of Feistel network

- **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 achieved by greater diffusion.
- **Key size:** Larger key size means greater security but may decrease encryption/ decryption speed. The greater security is achieved by greater resistance to brute-force attacks.
- **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 16 rounds.

- **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
- **Round function F:** Again, greater complexity generally means greater resistance to cryptanalysis.

Data Encryption Standards (DES)

- The Data Encryption Standard is a classic in that annals of cryptography.
- It was selected by the National Bureau of Standards as an official Federal Information Processing Standard (FIPS) for the United States in 1976.
- While it is now considered outdated and is not recommended for use, it was the premier block cipher for many years and bears study.
- Many cryptography textbooks and university courses use this as the basic Processing Standard (FIPS) for the United States in 1976

DES..

- DES uses a 56-bit key applied to a 64 bit block. (note there is actually a 64 bit key generated but 8 bits are just for error correction.)
- DES is a Feistel cipher with 16 rounds and a 48-bit round key for each round.
- So its general functionality follows the Feistel method of dividing the 64 bit block into two halves (32 bits each, this is NOT an unbalanced Feistel cipher) , applying the round function to one half, then xor'ing that output with the other half.

- DES encryption was broken in 1999 by Electronics Frontiers Foundation (EFF, www.eff.org).
- This resulted in NIST issuing a new directive that year that required organiza- tions to use Triple DES, that is, three consecutive applications of DES.

DES encryption overview

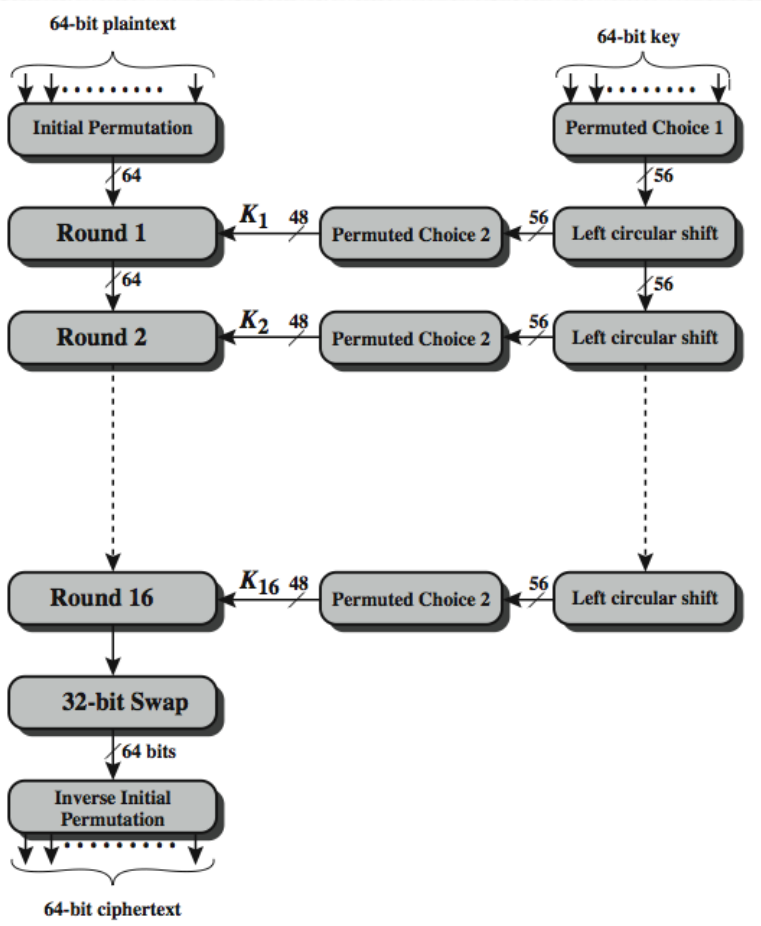
The overall scheme for DES encryption is illustrated Figure, which takes as input 64-bits of data and of key.

The left side shows the basic process for enciphering a 64-bit data block which consists of:

- an initial permutation (IP) which shuffles the 64-bit input block
- 16 rounds of a complex key dependent round function involving substitutions & permutations
- a final permutation, being the inverse of IP

The right side shows the handling of the 56-bit key and consists of:

- an initial permutation of the key (PC1) which selects 56-bits out of the 64-bits input, in two 28-bit halves
- 16 stages to generate the 48-bit subkeys using a left circular shift and a permutation of the two 28-bit halves



Initial Permutation IP

- The initial permutation and its inverse are defined by tables.
- The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64.The 64 entries in the 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 also consists of 64 bits.

Table 3.2 Permutation Tables for DES

(a) Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(b) Inverse Initial Permutation (IP^{-1})

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

(c) Expansion Permutation (E)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	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	1

(d) Permutation Function (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

For example: consider the following 64-bit input and its permutation :

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

where M_i is a binary digit. Then the permutation $X = (\mathrm{IP}(M)$ is as follows:

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

DETAILS OF SINGLE ROUND

As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

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

- The round key is 48 bits. The input is 32 bits.
- This input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the bits (Table 3.2c Expansion Permutation E Table).
- The resulting 48 bits are XORed with Ki.
- This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by Table 3.2d Permutation function P.

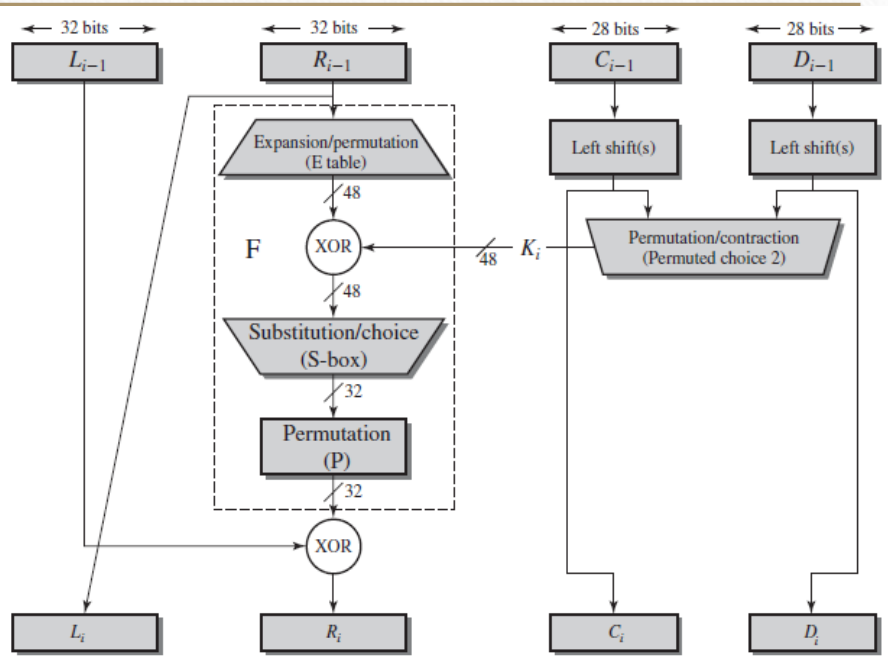
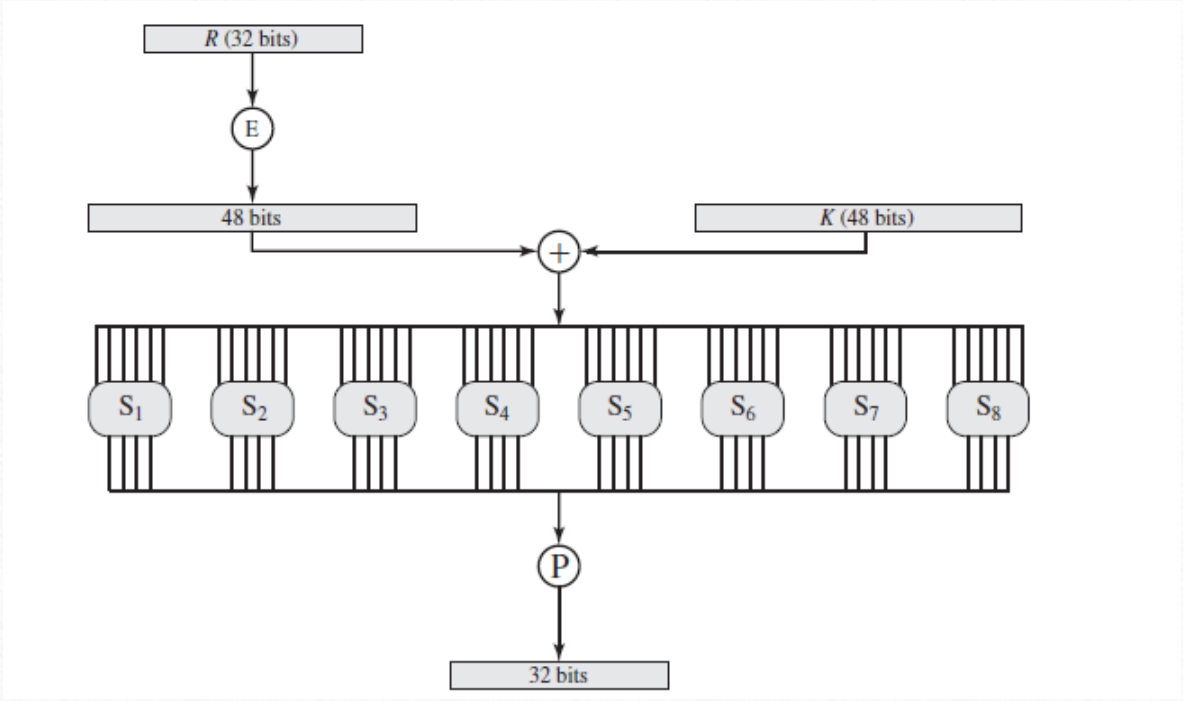


Figure 3.6 Single Round of DES Algorithm

Calculation of F(R, K)



The role of the S-boxes in the function F

- The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output.
- The first and last bits of the input to box form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for Si.
- The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output.

The role of the S-boxes in the function F

For example, in S_1, 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.

The 32-bit output from the eight S-boxes is then permuted, so that on the next round, the output from each S-box immediately affects as many others as possible.

Table 3.3 Definition of DES S-Boxes

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

KEY GENERATION

- **a 64-bit key is used as input to the algorithm.**
- The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in Table 3.4a.
- The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 3.4b).
- The resulting 56-bit key is then treated as two 28-bit quantities, labeled C0 and D0 and . At each round, Ci-1 and Di-1 are separately subjected to a circular left shift or (rotation) of 1 or 2 bits, as governed by Table 3.4d
- These shifted values serve as input to the next round. They also serve as input to the part labeled Permuted Choice Two (Table 3.4c), which produces a 48-bit output that serves as input to the function F(Ri-1,Ki)

Table 3.4 DES Key Schedule Calculation

(a) Input Key

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

(b) Permuted Choice One (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

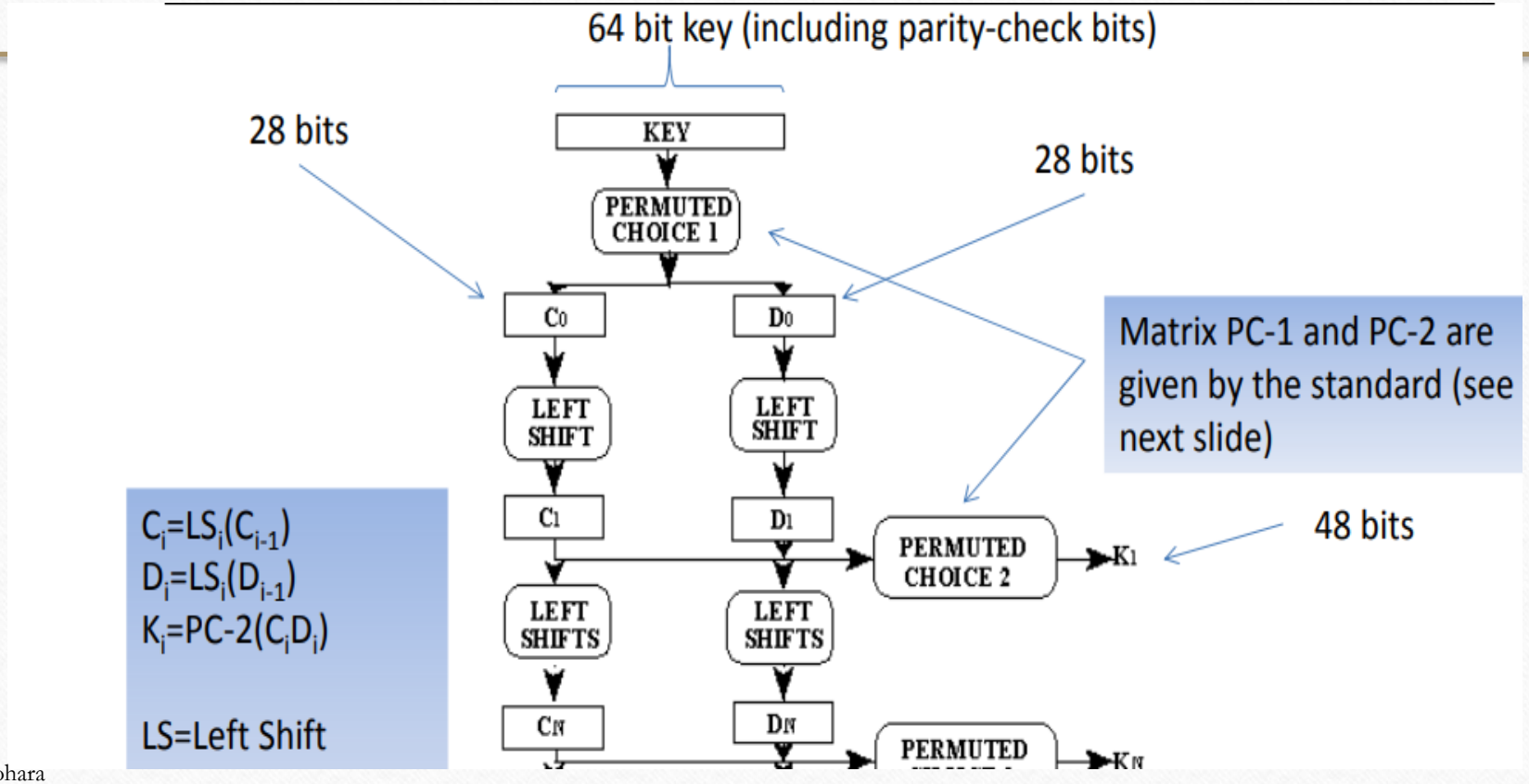
(c) Permuted Choice Two (PC-2)

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

(d) Schedule of Left Shifts

Round Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits Rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Sub Key generation



The Avalanche Effect

- A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext. This is referred to as the avalanche effect.

DES Example

- **Example:** Let **M** be the plain text message **M** = 0123456789ABCDEF, where **M** is in hexadecimal (base 16) format. Rewriting **M** in binary format, we get the 64-bit block of text:
- **M** = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
- Find IP
- Then divide IP into two halves
- Divide into 2 equal halves.
  L = 0000 0001 0010 0011 0100 0101 0110 0111
  R = 1000 1001 1010 1011 1100 1101 1110 1111

Subkey generation

K = 133457799BBCDFF1

- The DES algorithm uses the following steps:

1. **Step 1: Create 16 subkeys, each of which is 48-bits long**

The 64-bit key is permuted according to the following table, **PC-1**. Since the first entry in the table is "57", this means that the 57th bit of the original key **K** becomes the first bit of the permuted key **K**+. The 49th bit of the original key becomes the second bit of the permuted key. The 4th bit of the original key is the last bit of the permuted key. Note only 56 bits of the original key appear in the permuted key.

PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

- From the original 64-bit key

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

we get the 56-bit permutation

- **K**+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

From the permuted key **K**+, divide into two halves.

we get

C_0 = 1111000 0110011 0010101 0101111
D_0 = 0101010 1011001 1001111 0001111

With C_0 and D_0 defined, we now create sixteen blocks C_n and D_n, 1<=n<=16. Each pair of blocks C_n and D_n is formed from the previous pair C_{n-1} and D_{n-1}, respectively, for n = 1, 2, ..., 16, using the following schedule of "left shifts" of the previous block. To do a left shift, move each bit one place to the left, except for the first bit, which is cycled to the end of the block.

This means, for example, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1.

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

From original pair C_0 and D_0 we obtain:

$C_0 = 11110000110011001010101111$
$D_0 = 01010101011001100111100011111$

$C_1 = 11100001100110010101011111$
$D_1 = 10101010110011001111000111110$

$C_2 = 11000011001100101010111111$
$D_2 = 01010101100110011111000111101$

$C_3 = 00001100110010101010111111111$
$D_3 = 01010110011001111000111110101$

$C_4 = 00110011001010101011111111100$
$D_4 = 01011001100111110001111010101$

$C_5 = 11001100101010101111111110000$
$D_5 = 01100110011110001111010101011$

$C_6 = 00110010101010111111111000011$
$D_6 = 10011001111000111101010101011$

$C_7 = 11001010101011111111100001100$
$D_7 = 01100111100011111010101010110$

$C_8 = 00101010101111111110000110011$
$D_8 = 10011110001111010101011001$

$C_9 = 01010101011111111100001100110$
$D_9 = 00111100011110101010110011$

$C_{10} = 01010101111111110000110011001$
$D_{10} = 11110001111010101011001100$

$C_{11} = 01010111111110000110011001011$
$D_{11} = 11000111101010101100110011$

$C_{12} = 01011111111000011001100101011$
$D_{12} = 00011110101010101100110011111$

$C_{13} = 01111111100001100110010101011$
$D_{13} = 01111010101010110011001111100$

$C_{14} = 11111110000110011001010101011$
$D_{14} = 11101010101011001100111100011$

$C_{15} = 11111000011001100101010101111$
$D_{15} = 10101010101100110011110000111$

$C_{16} = 11110000110011001010101011111$
$D_{16} = 01010101011001100111100011111$

We now form the keys K_n, for $1<=n<=16$, by applying the following permutation table to each of the concatenated pairs C_nD_n. Each pair has 56 bits, but **PC-2** only uses 48 of these.

For the first key we have C_1D_1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

which, after we apply the permutation **PC-2**, becomes
K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

For the other keys we have

$K_2 = 011110\ 011010\ 111011\ 011001\ 110110\ 111100\ 100111\ 100101$
$K_3 = 010101\ 011111\ 110010\ 001010\ 010000\ 101100\ 111110\ 011001$
$K_4 = 011100\ 101010\ 110111\ 010110\ 110110\ 110011\ 010100\ 011101$
$K_5 = 011111\ 001110\ 110000\ 000111\ 111010\ 110101\ 001110\ 101000$
$K_6 = 011000\ 111010\ 010100\ 111110\ 010100\ 000111\ 101100\ 101111$
$K_7 = 111011\ 001000\ 010010\ 110111\ 111101\ 100001\ 100010\ 111100$
$K_8 = 111101\ 111000\ 101000\ 111010\ 110000\ 010011\ 101111\ 111011$
$K_9 = 111000\ 001101\ 101111\ 101011\ 111011\ 011110\ 011110\ 000001$
$K_{10} = 101100\ 011111\ 001101\ 000111\ 101110\ 100100\ 011001\ 001111$
$K_{11} = 001000\ 010101\ 111111\ 010011\ 110111\ 101101\ 001110\ 000110$
$K_{12} = 011101\ 010111\ 000111\ 110101\ 100101\ 000110\ 011111\ 101001$
$K_{13} = 100101\ 111100\ 010111\ 010001\ 111110\ 101011\ 101001\ 000001$
$K_{14} = 010111\ 110100\ 001110\ 110111\ 111100\ 101110\ 011100\ 111010$
$K_{15} = 101111\ 111001\ 000110\ 001101\ 001111\ 010011\ 111100\ 001010$
$K_{16} = 110010\ 110011\ 110110\ 001011\ 000011\ 100001\ 011111\ 110101$

Step 2: Encode each 64-bit block of data.

- There is an *initial permutation* **IP** of the 64 bits of the message data **M**. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of **M** becomes the first bit of **IP**. The 50th bit of **M** becomes the second bit of **IP**. The 7th bit of **M** is the last bit of **IP**.

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

- **M** = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
  IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

Here the 58th bit of **M** is "1", which becomes the first bit of **IP**. The 50th bit of **M** is "1", which becomes the second bit of **IP**. The 7th bit of **M** is "0", which becomes the last bit of **IP**.

Next divide the permuted block **IP** into a left half L_0 of 32 bits, and a right half R_0 of 32 bits.

- From **IP**, we get L_0 and R_0

L_0 = 1100 1100 0000 0000 1100 1100 1111 1111
R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

We now proceed through 16 iterations, for $1 <= n <= 16$, using function f which operates on two blocks--a data block of 32 bits and a key K_n of 48 bits--to produce a block of 32 bits. **Let + denote XOR addition.** Then for **n** going from 1 to 16 we calculate

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

This results in a final block, for $n = 16$, of $L_{16}R_{16}$. That is, in each iteration, we take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation f.

- For $n = 1$, we have
- $K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$
  $L_1 = R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$
  $R_1 = L_0 + f(R_0, K_1)$
- It remains to explain how the function f works.
- To calculate f, we first expand each block R_{n-1} from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in R_{n-1}. We'll call the use of this selection table the function **E**. Thus $\mathbf{E}(R_{n-1})$ has a 32 bit input block, and a 48 bit output block.

Let **E** be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

E BIT-SELECTION TABLE

32	1	2	3	4	5
4	5	6	7	8	9
8	9	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	1

Thus the first three bits of $\mathbf{E}(R_{n-1})$ are the bits in positions 32, 1 and 2 of R_{n-1} while the last 2 bits of $\mathbf{E}(R_{n-1})$ are the bits in positions 32 and 1.

Example: We calculate $\mathbf{E}(R_0)$ from R_0 as follows:

R_0 = 1111 0000 1010 1010 1111 0000 1010 1010
$\mathbf{E}(R_0)$ = 011110 100001 010101 010101 011110 100001 010101 010101

(Note that each block of 4 original bits has been expanded to a block of 6 output bits.)

Next in the f calculation, we XOR the output $\mathbf{E}(R_{n-1})$ with the key K_n:

$$K_n + \mathbf{E}(R_{n-1}).$$

- For K_1, $\mathbf{E}(R_0)$, we have
- K_1 = 000110 110000 001011 101111 111111 000111 000001 110010
  $\mathbf{E}(R_0)$ = 011110 100001 010101 010101 011110 100001 010101 010101
  $K_1 + \mathbf{E}(R_0)$ = 011000 010001 011110 111010 100001 100110 010100 100111.
- We now have 48 bits, or eight groups of six bits. We now do something strange with each group of six bits: we use them as addresses in tables called "**S boxes**". Each group of six bits will give us an address in a different **S** box. Located at that address will be a 4 bit number. This 4 bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the **S** boxes) for 32 bits total.

Write the previous result, which is 48 bits, in the form:

$$K_n + E(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8,$$

where each ***B*** is a group of six bits. We now calculate

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$$

where $S_i(B_i)$ referres to the output of the ***i***th **S**-box.

To repeat, each of the functions ***S1, S2,..., S8***, takes a 6-bit block as input and yields a 4-bit block as output. The table to determine S_1 is shown and explained below:

S1

Row No.	Column Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

$S_1(B)$ is determined as follows: The first and last bits of B represent in base 2 a number in the decimal range 0 to 3 (or binary 00 to 11). Let that number be i. The middle 4 bits of B represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be j. Look up in the table the number in the i-th row and j-th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B. For example, for input block B = 011011 the first bit is "0" and the last bit "1" giving 01 as the row. This is row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence S_1(011011) = 0101.

Other S-boxes are

S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

- For the first round, we obtain as the output of the eight **S** boxes:
- $\boldsymbol{K_1} + \mathbf{E}(\boldsymbol{R_0})$ = 011000 010001 011110 111010 100001 100110 010100 100111.
- $\boldsymbol{S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)}$ = 0101 1100 1000 0010 1011 0101 1001 0111

- The final stage in the calculation of ***f*** is to do a permutation **P** of the **S**-box output to obtain the final value of ***f***:

$$f = \mathbf{P}(S_1(B_1)S_2(B_2)...S_8(B_8))$$

The permutation **P** is defined in the following table. **P** yields a 32-bit output from a 32-bit input by permuting the bits of the input block

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

From the output of the eight **S** boxes:

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ = 0101 1100 1000 0010 1011 0101 1001 0111

we get

f = 0010 0011 0100 1010 1010 1001 1011 101

$$R_1 = L_0 + f(R_0 , K_1)$$

= 1100 1100 0000 0000 1100 1100 1111 1111
+ 0010 0011 0100 1010 1010 1001 1011 1011
= 1110 1111 0100 1010 0110 0101 0100 0100

- In the next round, we will have $L_2 = R_1$, which is the block we just calculated, and then we must calculate $R_2 = L_1 + f(R_1, K_2)$, and so on for 16 rounds.
- At the end of the sixteenth round we have the blocks L_{16} and R_{16}. We then ***reverse*** the order of the two blocks into the 64-bit block.

$$R_{16}L_{16}$$

and apply a final permutation $\mathbf{IP^{-1}}$ as defined by the following tabl

$\mathbf{IP^{-1}}$

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

- If we process all 16 blocks using the method defined previously, we get, on the 16th round,
- L_{16} = 0100 0011 0100 0010 0011 0010 0011 0100
  R_{16} = 0000 1010 0100 1100 1101 1001 1001 0101
- We reverse the order of these two blocks and apply the final permutation to
- $R_{16}L_{16}$ = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100
- IP^{-1} = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101
- which in hexadecimal format is 85E813540F0AB405.
- So the encrypted form of **M** = 0123456789ABCDEF:

Is **C** = 85E813540F0AB405.

Multiple Encryption and Triple DES

- Multiple encryption is a technique in which an encryption algorithm is used multiple times. In the first instance, plaintext is converted to ciphertext using the encryption algorithm. This ciphertext is then used as input and the algorithm is applied again. This process may be repeated through any number of stages.
- Triple DES makes use of three stages of the DES algorithm, using a total of two or three distinct keys.

- Given the potential vulnerability of DES to a brute-force attack, there has been considerable interest in finding an alternative.
- One approach is to design a completely new algorithm, of which AES is a prime example.
- Another alternative, which would preserve the existing investment in software and equipment, is to use multiple encryption with DES and multiple keys.

Double DES

- The simplest form of multiple encryption has two encryption stages and two keys
- Given a plaintext P and two encryption keys and , ciphertext is generated as

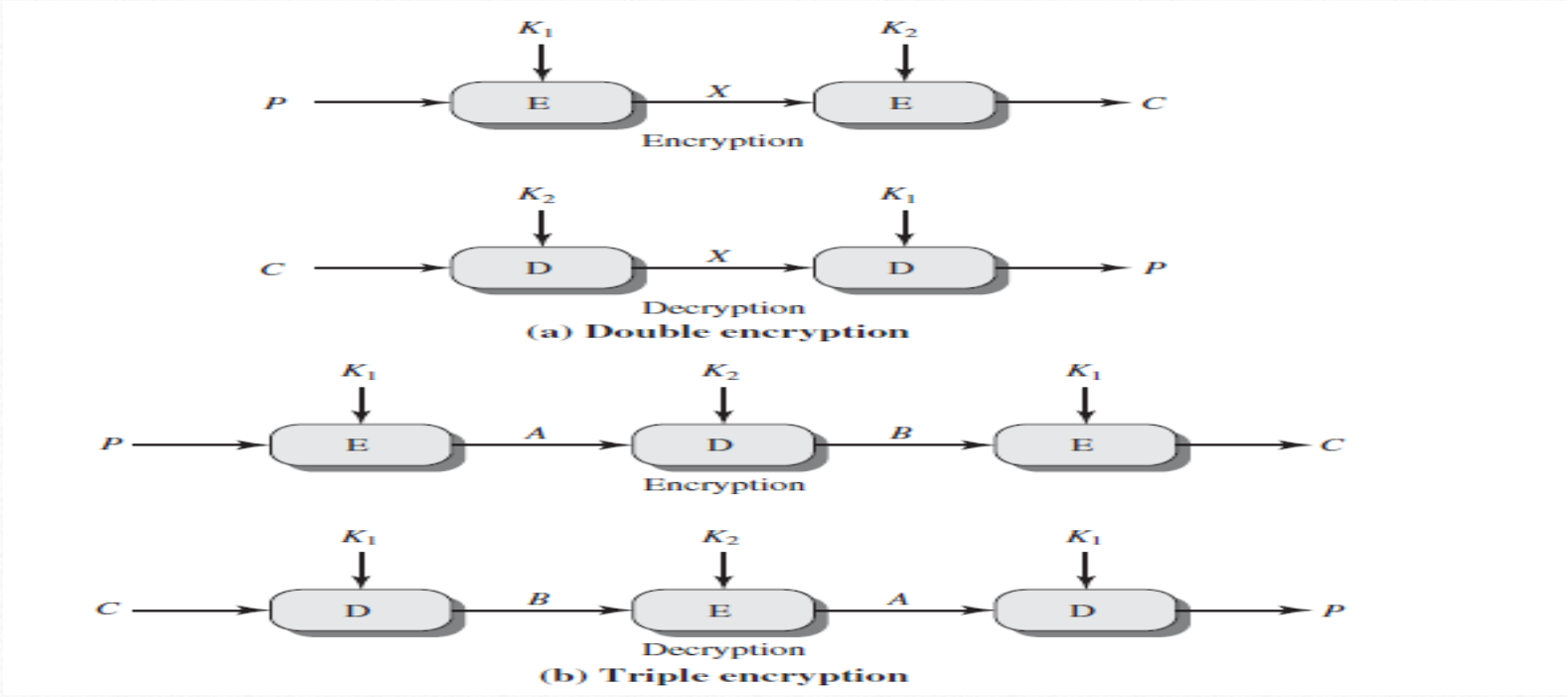
$$C = \mathrm{E}(K2, \mathrm{E}(K1, P))$$

- Decryption requires that the keys be applied in reverse order:

$$P = \mathrm{D}(K_1, \mathrm{D}(K_2, C))$$

- For DES, this scheme apparently involves a key length of 56 * 2 = 112 bits, resulting in a dramatic increase in cryptographic strength. But we need to examine the algorithm more closely.

Multiple Encryption



Triple DES with Two Keys

- Tuchman proposed a triple encryption method that uses only two keys [TUCH79]. The function follows an encrypt-decrypt-encrypt (EDE) sequence (Figure b):

$$C = \mathrm{E}(K1, \mathrm{D}(K2, \mathrm{E}(K1, P)))$$

$$P = \mathrm{D}(K1, \mathrm{E}(K2, \mathrm{D}(K1, C)))$$

Triple DES with Three Keys

- Three-key 3DES has an effective key length of 168 bits and is defined as

$$C = E(K3, D(K2, E(K1, P)))$$

Backward compatibility with DES is provided by putting

$$K3 = K2 \text{ or } K1 = K2$$

- A number of Internet-based applications have adopted three-key 3DES, including PGP and S/MIME.

Advanced Encryption Standard

- **AES Origins**
  - designed by Rijmen-Daemen in Belgium
  - has 128/192/256 bit keys, 128 bit data
  - an **iterative** rather than **Feistel** cipher
    - processes data as block of 4 columns of 4 bytes
    - operates on entire data block in every round

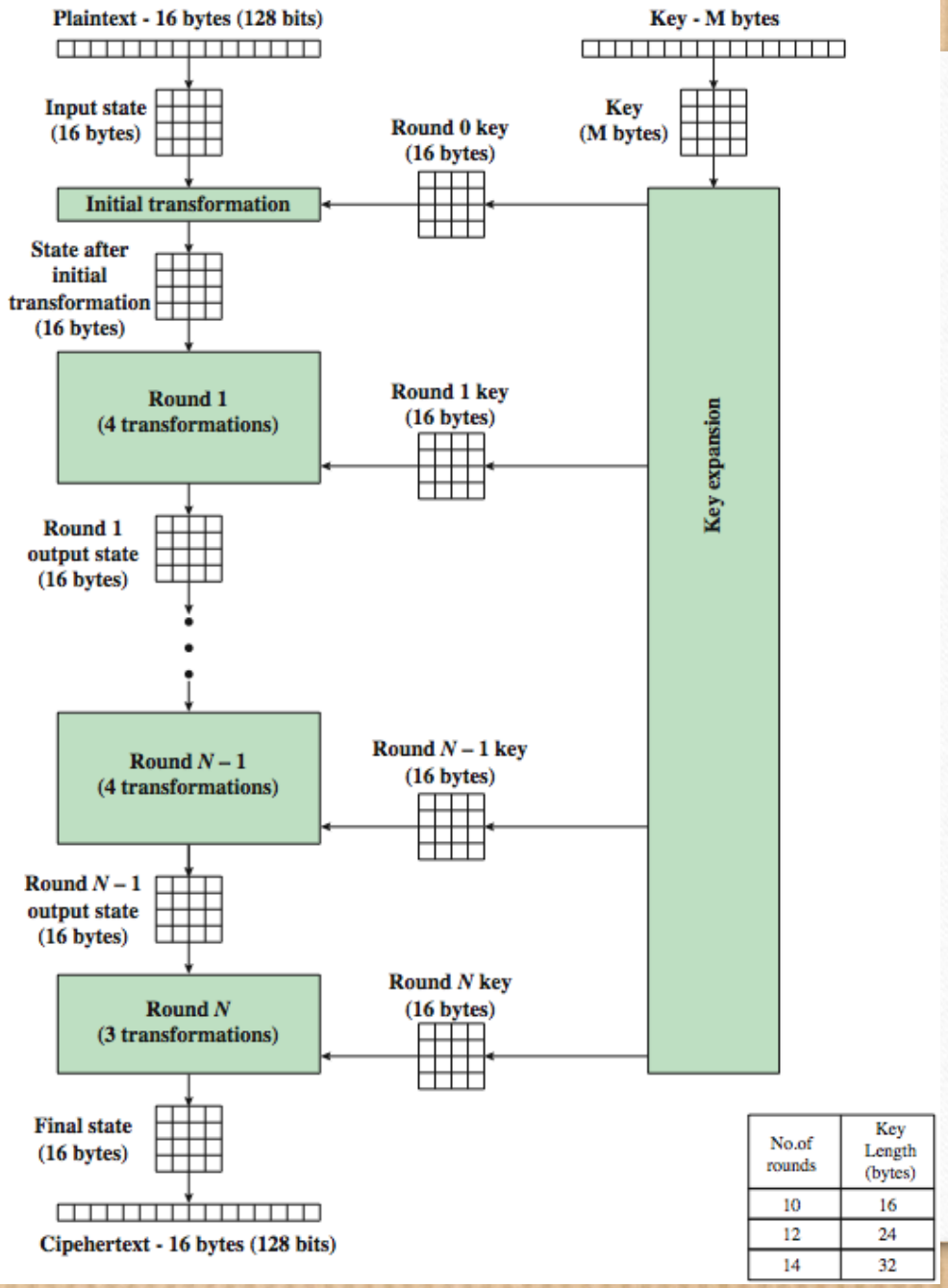
- AES uses 128 bits of plain text.
- AES consists of multiple rounds of processing different key bits like,
  - 128 bits key – perform 10 rounds of encryption
  - 192 bits key – perform 12 rounds of encryption.
  - 256 bits key – perform 14 rounds of encryption.
- Overall structure of AES encryption process shown in figure.
- For encryption, each round consists of the following four steps:

1) Sub Bytes – 10 Rounds
2) Shift Rows – 10 Rounds
3) Mix Columns – 9 Rounds (Not present in last round)
4) Add Round Key – 10 Rounds

Table 5.3. AES Parameters

Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

AES Encryption Process

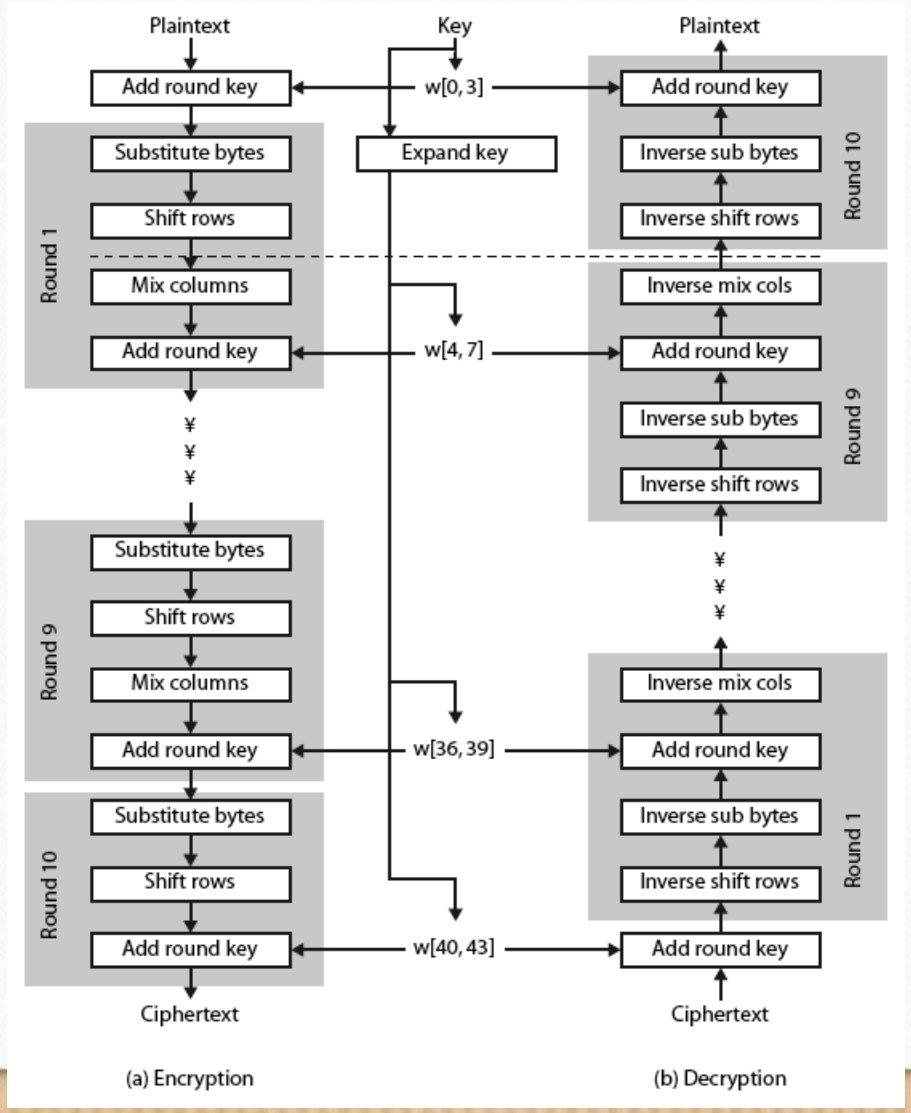


No.of rounds	Key Length (bytes)
10	16
12	24
14	32

AES Structure

- data block of 4 columns of 4 bytes is state
- key is expanded to array of words
- has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (subs using matrix multiply of groups)
  - add round key (XOR state with key material)

AES Structure



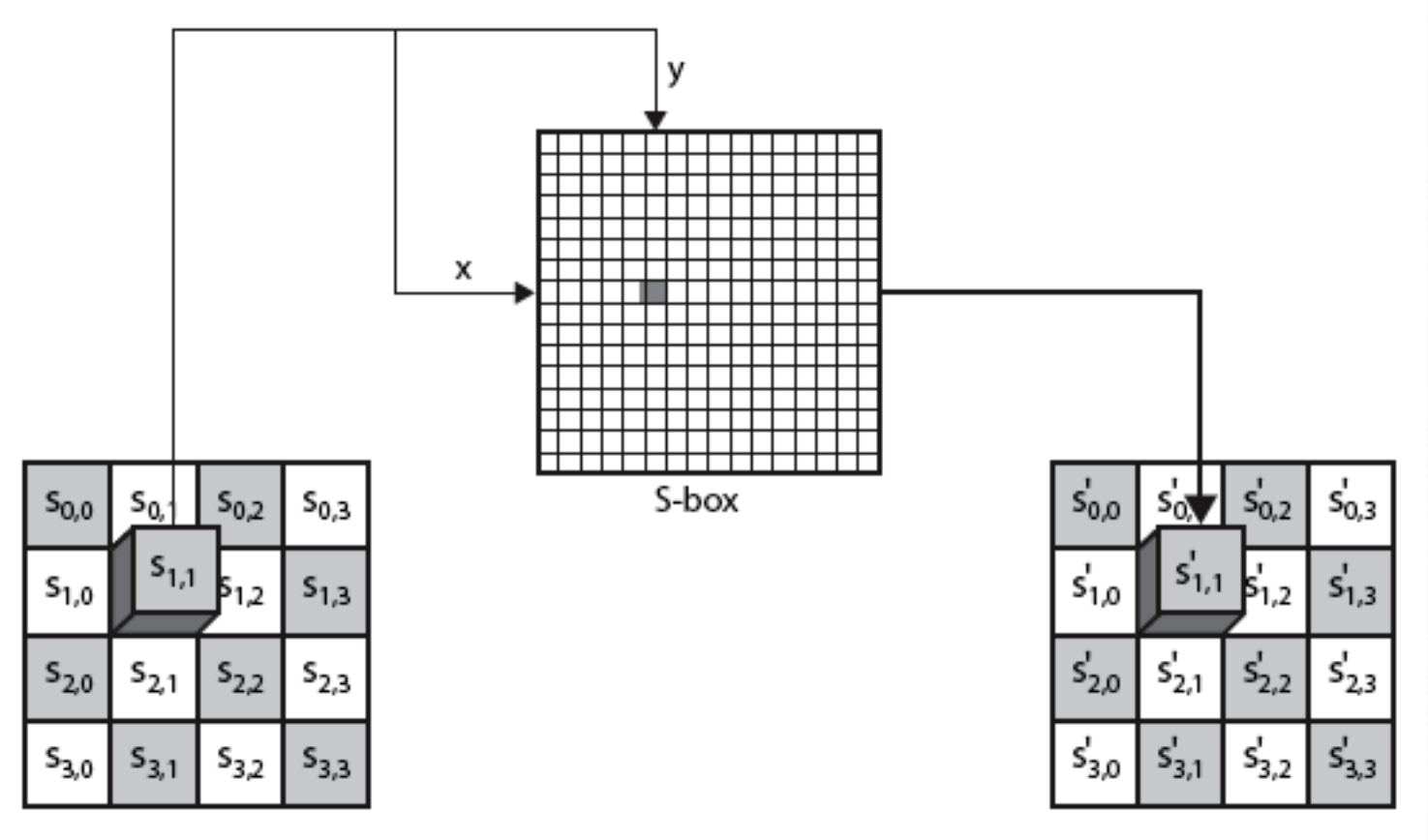
Some Comments on AES

1. an **iterative** rather than **Feistel** cipher
2. key expanded into array of 32-bit words
   four words form round key in each round
3. 4 different stages are used as shown
4. has a simple structure
5. only AddRoundKey uses key
6. AddRoundKey a form of Vernam cipher
7. each stage is easily reversible
8. decryption uses keys in reverse order
9. decryption does recover plaintext
10. final round has only 3 stages

Substitute Bytes

- a simple substitution of each byte
- uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by byte in row 9 column 5
  - which has value {2A}
- S-box constructed using defined transformation of values in $GF(2^8)$
- designed to be resistant to all known attacks

Substitute Bytes



SubBytes Table

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

InvSubBytes Table

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

Substitute Bytes Example

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

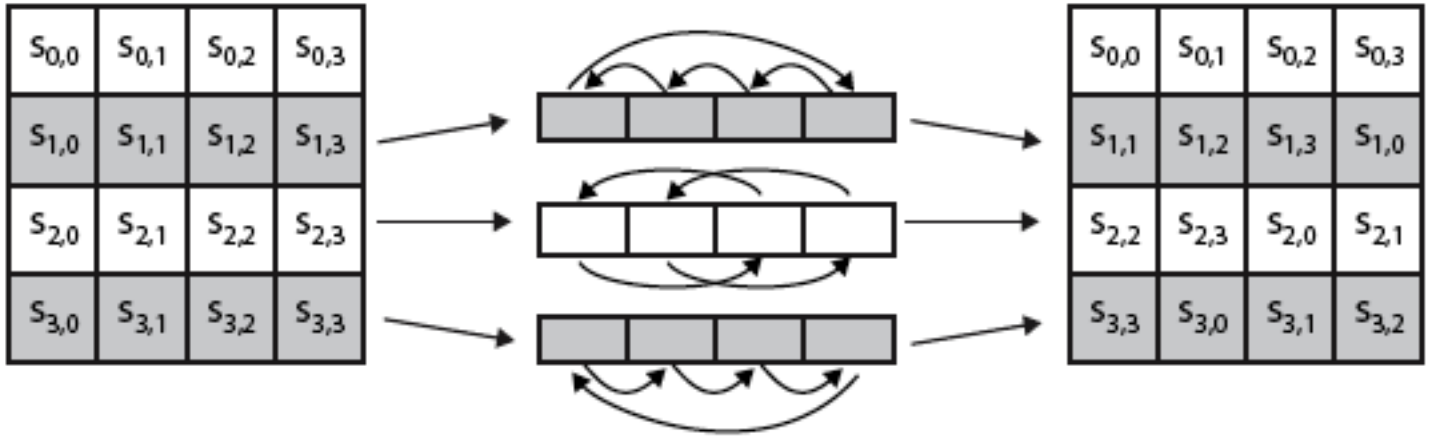
→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Shift Rows

- a circular byte shift in each
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- decrypt inverts using shifts to right
- since state is processed by columns, this step permutes bytes between the columns

Shift Rows



87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

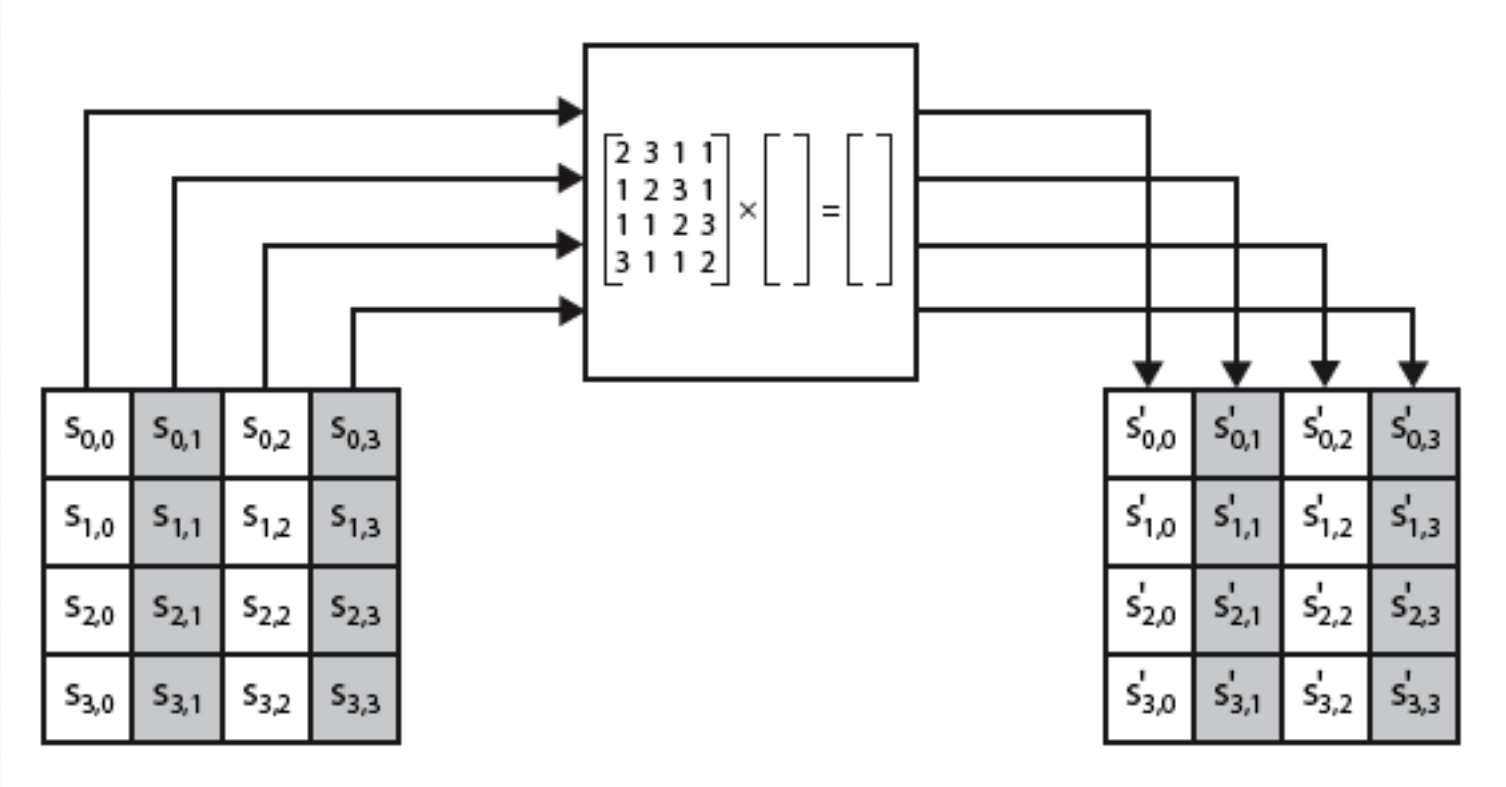
87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(2^8) using prime poly $m(x) = x^8+x^4+x^3+x+1$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Mix Columns


2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
×
=
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

- Third transformation function of AES is called as Mix Columns, operates on each column individually.
- State Array = Output of Shift Rows function.
- Each byte of a column is mapped into a new value that is a function of all four bytes in that column.

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

Predefine Matrix

*

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

State Array

=

$S'_{0,0}$	$S'_{0,1}$	$S'_{0,2}$	$S'_{0,3}$
$S'_{1,0}$	$S'_{1,1}$	$S'_{1,2}$	$S'_{1,3}$
$S'_{2,0}$	$S'_{2,1}$	$S'_{2,2}$	$S'_{2,3}$
$S'_{3,0}$	$S'_{3,1}$	$S'_{3,2}$	$S'_{3,3}$

New State Array

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

*

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

=

?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

$\{02\} * \{87\} \oplus \{03\} * \{6E\} \oplus \{01\} * \{46\} \oplus \{01\} * \{A6\}$

$02 = 0000\ 0010 = X$

$87 = 1000\ 0111 = X^7 + X^2 + X + 1$

$02 * 87 = X * (X^7 + X^2 + X + 1)$

$= X^8 + X^3 + X^2 + X$

$= X^4 + X^3 + X + 1 + X^3 + X^2 + X$

$= X^4 + X^2 + 1$

$= 0001\ 0101$

$03 = 0000\ 0011 = X + 1$

$6E = 0110\ 1110 = X^6 + X^5 + X^3 + X^2 + X$

$03 * 6E = (X+1) * (X^6 + X^5 + X^3 + X^2 + X)$

$= X^7 + X^6 + X^4 + X^3 + X^2 + X^6 + X^5 + X^3 + X^2 + X$

$= X^7 + X^5 + X^4 + X$

$= 1011\ 0010$

$01 * 46 = 46 = 0100\ 0110$

Similarly , **01.A6=A6**

In Binary, 1010 0110

Note: The Advanced Encryption Standard (AES) uses arithmetic in the finite field GF(2^8), with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. While manipulating algebraic expressions involving Powers 8 or higher, divide the expresión by $m(x) = x^8 + x^4 + x^3 + x + 1$.

$\{02\} * \{87\} \oplus \{03\} * \{6E\} \oplus \{01\} * \{46\} \oplus \{01\} * \{A6\} = \{47\}$

$02 * 87 = 0\ 0\ 0\ 1\ \ 0\ 1\ 0\ 1$
$03 * 6E = 1\ 0\ 1\ 1\ \ 0\ 0\ 1\ 0$
$01 * 46 = 0\ 1\ 0\ 0\ \ 0\ 1\ 1\ 0$
$01 * A6 = 1\ 0\ 1\ 0\ \ 0\ 1\ 1\ 0$

$= 0\ 1\ 0\ 0\ \ 0\ 1\ 1\ 1 = \{47\}$

4 7

$01 * A6 = A6 = 1010\ 0110$

$\{01\} * \{87\} \oplus \{02\} * \{6E\} \oplus \{03\} * \{46\} \oplus \{01\} * \{A6\}$

$01 * 87 = 87 = 1000\ 0111$

$02 = 0000\ 0010 = X$

$6E = 0110\ 1110 = X^6 + X^5 + X^3 + X^2 + X$

$02 * 6E = X * (X^6 + X^5 + X^3 + X^2 + X)$
$= X^7 + X^6 + X^4 + X^3 + X^2$
$= 1101\ 1100$

$03 = 0000\ 0110 = X + 1$

$46 = 0100\ 0110 = X^6 + X^2 + X$

$03 * 46 = (X + 1) * (X^6 + X^2 + X)$
$= X^7 + X^3 + X^2 + X^6 + X^2 + X$
$= X^7 + X^6 + X^3 + X$
$= 1100\ 1010$

$01 * 87 = 1\ 0\ 0\ 0\ \ 0\ 1\ 1\ 1$
$02 * 6E = 1\ 1\ 0\ 1\ \ 1\ 1\ 0\ 0$
$03 * 46 = 1\ 1\ 0\ 0\ \ 1\ 0\ 1\ 0$
$01 * A6 = 1\ 0\ 1\ 0\ \ 0\ 1\ 1\ 0$

$= 0\ 0\ 1\ 1\ \ 0\ 1\ 1\ 1 = \{37\}$

3 7

AES Mix Column

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

*

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

=

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

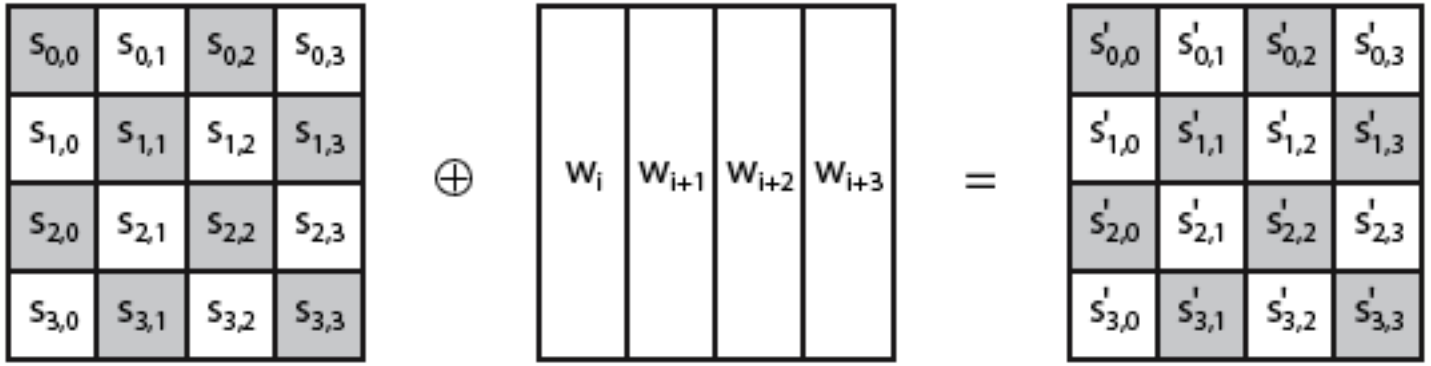
Add Round Key

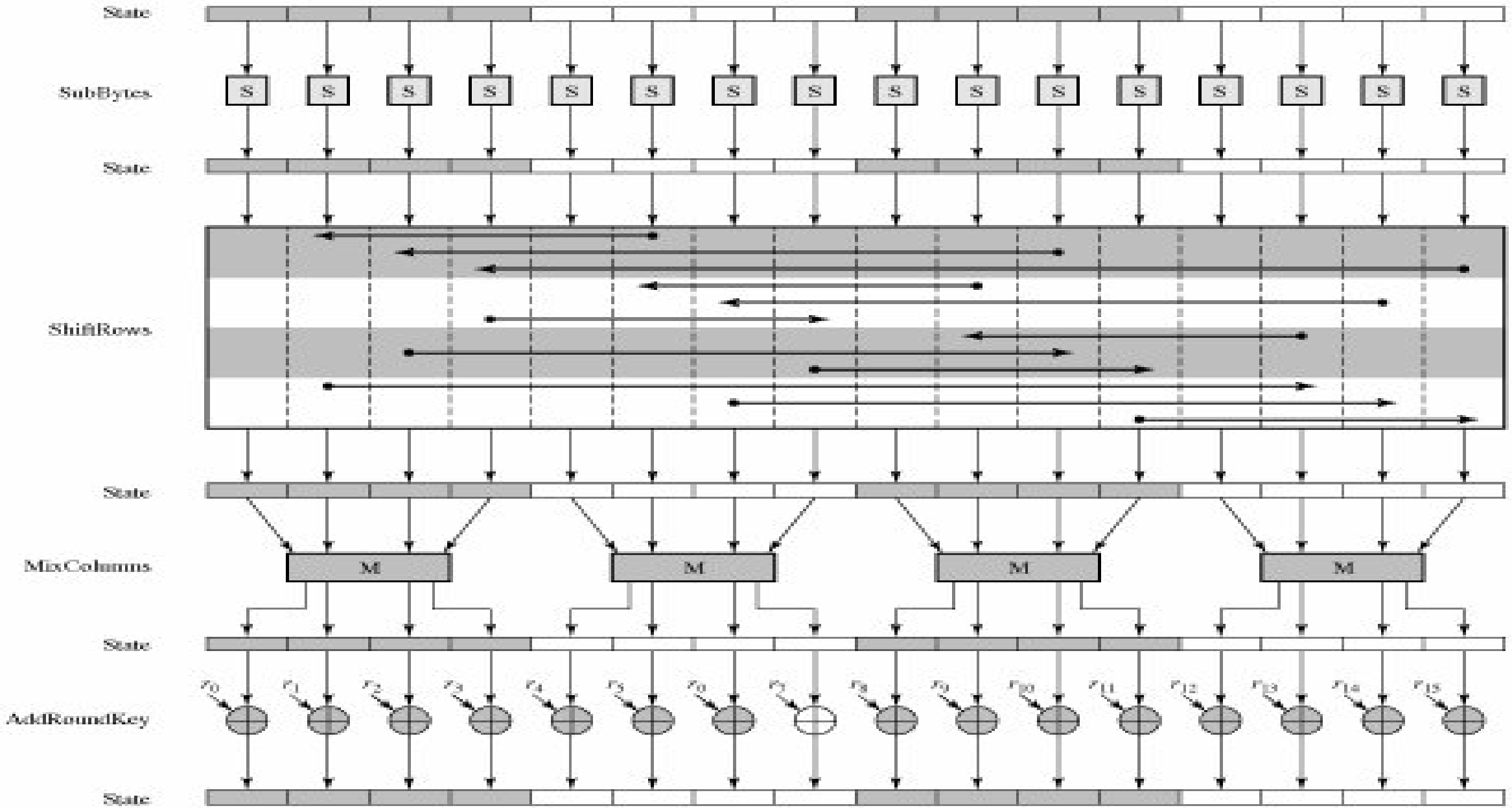
- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption identical
  - since XOR own inverse, with reversed keys
- designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security

AddRoundKey

- XOR state with 128-bits of the round key
- AddRoundKey proceeds one column at a time.
  - adds a round key word with each state column matrix
  - the operation is matrix addition
- Inverse for decryption identical
  - since XOR own inverse, with reversed keys
- Designed to be as simple as possible

AddRoundKey Scheme




State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
State

AES Key Scheduling

- takes 128-bits (16-bytes) key and expands into array of 44 32-bit words

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	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

AES Key Expansion

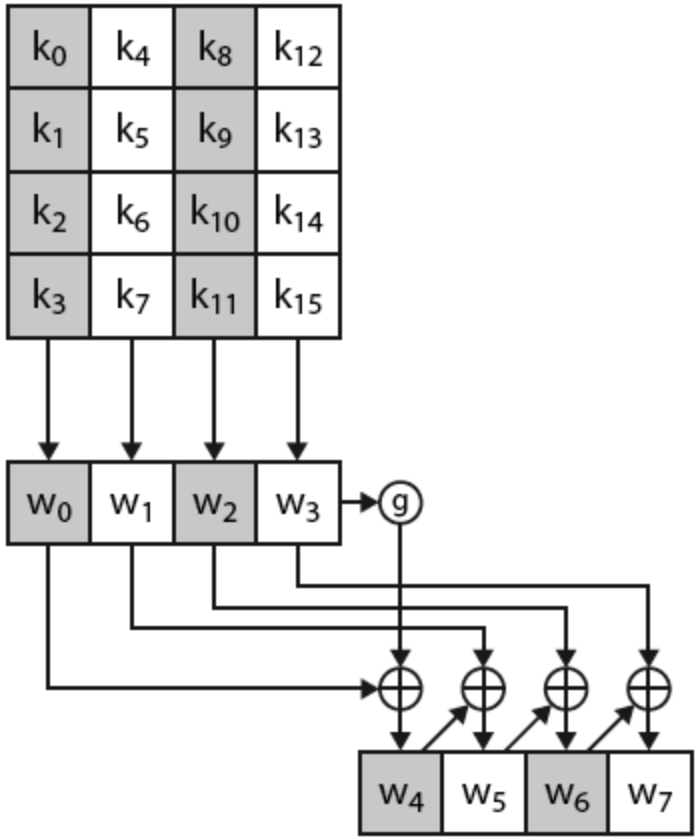
- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- start by copying key into first 4 words
- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back

Algorithm 7.5 *Pseudocode for key expansion in AES-128*

```
KeyExpansion ([key0 to key15], [w0 to w43])
{
    for (i = 0 to 3)
        wi ← key4i + key4i+1 + key4i+2 + key4i+3

    for (i = 4 to 43)
    {
        if (i mod 4 ≠ 0)    wi ← wi–1 + wi–4
        else
        {
            t ← SubWord (RotWord (wi–1)) ⊕ RCon i/4        // t is a temporary word
            wi ← t + wi–4
        }
    }
}
```

AES Key Expansion



Key Expansion submodule

- **RotWord** performs a one byte circular left shift on a word
  For example:

  RotWord[b0,b1,b2,b3] = [b1,b2,b3,b0]

- **SubWord** performs a byte substitution on each byte of input word using the S-box

- **SubWord(RotWord(temp))** is XORed with RCon[j] – the round constant

Round Constant (RCon)

- Round Constant ,RCON [j] ,is a word in which the three rightmost bytes are always zero.
- Thus the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word.
- It is different for each round and defined as:

  RCon[j] = (RC[j],0,0,0)

  where RC[1] =1 , RC[j] = 2 * RC[j-1]
- Multiplication is defined over GF(2^8)

The values of RCon[j] in hexadecimal are

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	$(\underline{\mathbf{36}}\ 00\ 00\ 00)_{16}$

RC[j)

The key-expansion routine can either use the above table when calculating the words or use the $GF(2^8)$ field to calculate the leftmost byte dynamically, as shown below (prime is the irreducible polynomial):

RC_1	$\rightarrow x^{1-1}$	$=x^0$	mod *prime*	$=1$	$\rightarrow 00000001$	$\rightarrow 01_{16}$
RC_2	$\rightarrow x^{2-1}$	$=x^1$	mod *prime*	$=x$	$\rightarrow 00000010$	$\rightarrow 02_{16}$
RC_3	$\rightarrow x^{3-1}$	$=x^2$	mod *prime*	$=x^2$	$\rightarrow 00000100$	$\rightarrow 04_{16}$
RC_4	$\rightarrow x^{4-1}$	$=x^3$	mod *prime*	$=x^3$	$\rightarrow 00001000$	$\rightarrow 08_{16}$
RC_5	$\rightarrow x^{5-1}$	$=x^4$	mod *prime*	$=x^4$	$\rightarrow 00010000$	$\rightarrow 10_{16}$
RC_6	$\rightarrow x^{6-1}$	$=x^5$	mod *prime*	$=x^5$	$\rightarrow 00100000$	$\rightarrow 20_{16}$
RC_7	$\rightarrow x^{7-1}$	$=x^6$	mod *prime*	$=x^6$	$\rightarrow 01000000$	$\rightarrow 40_{16}$
RC_8	$\rightarrow x^{8-1}$	$=x^7$	mod *prime*	$=x^7$	$\rightarrow 10000000$	$\rightarrow 80_{16}$
RC_9	$\rightarrow x^{9-1}$	$=x^8$	mod *prime*	$=x^4+x^3+x+1$	$\rightarrow 00011011$	$\rightarrow 1B_{16}$
RC_{10}	$\rightarrow x^{10-1}$	$=x^9$	mod *prime*	$=x^5+x^4+x^2+x$	$\rightarrow 00110110$	$\rightarrow 36_{16}$

For example, suppose that the round key for round 8 is

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

As we know, 8^{th} round key consists of words w[32-35] and 9^{th} round consists of words w[36-39].

So, To find the the 9^{th} round key, find w[36-39].

Now , $w_i = w_{36}$ i.e. i IS MULTIPLE OF 4.

TEMP=W[i-1]=W35= 7F 8D 29 2F

AFTER ROTATE WORD, 8D 29 2F 7F

AFTER SUB WORD, 5D A5 15 D2

RCON(j)=RCON(i\4)=RECON(j)=RCON(36/4)=RCON(9)= 1B000000

After XOR with Rcon i.e. XOR **5D A5 15 D2 with** 1B000000
(sub(rotate(word)) + RCON(i/4)

46A515D2, which is g or t or temp.

w[i] =temp/t/g+ w[i-4]
W[36]=temp/t/g+w[32]
= **46A515D2+ EA D2 73 21**
= AC7766F3
Similary , find w[37-39]

AES Example Key Expansion

Key Words	Auxiliary Function
w0 = 0f 15 71 c9	RotWord(w3)= 7f 67 98 af = x1
w1 = 47 d9 e8 59	SubWord(x1)= d2 85 46 79 = y1
w2 = 0c b7 ad	Rcon(1)= 01 00 00 00
w3 = af 7f 67 98	y1 ⊕ Rcon(1)= d3 85 46 79 = z1
w4 = w0 ⊕ z1 = dc 90 37 b0	RotWord(w7)= 81 15 a7 38 = x2
w5 = w4 ⊕ w1 = 9b 49 df e9	SubWord(x4)= 0c 59 5c 07 = y2
w6 = w5 ⊕ w2 = 97 fe 72 3f	Rcon(2)= 02 00 00 00
w7 = w6 ⊕ w3 = 38 81 15 a7	y2 ⊕ Rcon(2)= 0e 59 5c 07 = z2
w8 = w4 ⊕ z2 = d2 c9 6b b7	RotWord(w11)= ff d3 c6 e6 = x3
w9 = w8 ⊕ w5 = 49 80 b4 5e	SubWord(x2)= 16 66 b4 8e = y3
w10 = w9 ⊕ w6 = de 7e c6 61	Rcon(3)= 04 00 00 00
w11 = w10 ⊕ w7 = e6 ff d3 c6	y3 ⊕ Rcon(3)= 12 66 b4 8e = z3
w12 = w8 ⊕ z3 = c0 af df 39	RotWord(w15)= ae 7e c0 b1 = x4
w13 = w12 ⊕ w9 = 89 2f 6b 67	SubWord(x3)= e4 f3 ba c8 = y4
w14 = w13 ⊕ w10 = 57 51 ad 06	Rcon(4)= 08 00 00 00
w15 = w14 ⊕ w11 = b1 ae 7e c0	y4 ⊕ Rcon(4)= ec f3 ba c8 = 4
w16 = w12 ⊕ z4 = 2c 5c 65 f1	RotWord(w19)= 8c dd 50 43 = x5
w17 = w16 ⊕ w13 = a5 73 0e 96	SubWord(x4)= 64 c1 53 1a = y5
w18 = w17 ⊕ w14 = f2 22 a3 90	Rcon(5)= 10 00 00 00
w19 = w18 ⊕ w15 = 43 8c dd 50	y5 ⊕ Rcon(5)= 74 c1 53 1a = z5
w20 = w16 ⊕ z5 = 58 9d 36 eb	RotWord(w23)= 40 46 bd 4c = x6
w21 = w20 ⊕ w17 = fd ee 38 7d	SubWord(x5)= 09 5a 7a 29 = y6
w22 = w21 ⊕ w18 = 0f cc 9b ed	Rcon(6)= 20 00 00 00
w23 = w22 ⊕ w19 = 4c 40 46 bd	y6 ⊕ Rcon(6)= 29 5a 7a 29 = z6
w24 = w20 ⊕ z6 = 71 c7 4c c2	RotWord(w27)= a5 a9 ef cf = x7
w25 = w24 ⊕ w21 = 8c 29 74 bf	SubWord(x6)= 06 d3 df 8a = y7
w26 = w25 ⊕ w22 = 83 e5 ef 52	Rcon(7)= 40 00 00 00
w27 = w26 ⊕ w23 = cf a5 a9 ef	y7 ⊕ Rcon(7)= 46 d3 df 8a = z7
w28 = w24 ⊕ z7 = 37 14 93 48	RotWord(w31)= 7d a1 4a f7 = x8
w29 = w28 ⊕ w25 = bb 3d e7 f7	SubWord(x7)= ff 32 d6 68 = y8
w30 = w29 ⊕ w26 = 38 d8 08 a5	Rcon(8)= 80 00 00 00
w31 = w30 ⊕ w27 = f7 7d a1 4a	y8 ⊕ Rcon(8)= 7f 32 d6 68 = z8
w32 = w28 ⊕ z8 = 48 26 45 20	RotWord(w35)= be 0b 38 3c = x9
w33 = w32 ⊕ w29 = f3 1b a2 d7	SubWord(x8)= ae 2b 07 eb = y9
w34 = w33 ⊕ w30 = cb c3 aa 72	Rcon(9)= 1B 00 00 00
w35 = w34 ⊕ w32 = 3c be 0b 38	y9 ⊕ Rcon(9)= b5 2b 07 eb = z9
w36 = w32 ⊕ z9 = fd 0d 42 cb	RotWord(w39)= 6b 41 56 f9 = x10
w37 = w36 ⊕ w33 = 0e 16 e0 1c	SubWord(x9)= 7f 83 b1 99 = y10
w38 = w37 ⊕ w34 = c5 d5 4a 6e	Rcon(10)= 36 00 00 00
w39 = w38 ⊕ w35 = f9 6b 41 56	y10 ⊕ Rcon(10)= 49 83 b1 99 = z10
w40 = w36 ⊕ z10 = b4 8e f3 52	
w41 = w40 ⊕ w37 = ba 98 13 4e	
w42 = w41 ⊕ w38 = 7f 4d 59 20	
w43 = w42 ⊕ w39 = 86 26 18 76	

AES Example Encryption

Start of round	After SubBytes	After ShiftRows	After MixColumns	Round Key
01 89 fe 76 23 ab dc 54 45 cd ba 32 67 ef 98 10				0f 47 0c af 15 d9 b7 7f 71 e8 ad 67 c9 59 d6 98
0e ce f2 d9 36 72 6b 2b 34 25 17 55 ae b6 4e 88	ab 8b 89 35 05 40 7f f1 18 3f f0 fc e4 4e 2f c4	ab 8b 89 35 40 7f f1 05 f0 fc 18 3f c4 e4 4e 2f	b9 94 57 75 e4 8e 16 51 47 20 9a 3f c5 d6 f5 3b	dc 9b 97 38 90 49 fe 81 37 df 72 15 b0 e9 3f a7
65 0f c0 4d 74 c7 e8 d0 70 ff e8 2a 75 3f ca 9c	4d 76 ba e3 92 c6 9b 70 51 16 9b e5 9d 75 74 de	4d 76 ba e3 c6 9b 70 92 9b e5 51 16 de 9d 75 74	8e 22 db 12 b2 f2 dc 92 df 80 f7 c1 2d c5 1e 52	d2 49 de e6 c9 80 7e ff 6b b4 c6 d3 b7 5e 61 c6
5c 6b 05 f4 7b 72 a2 6d b4 34 31 12 9a 9b 7f 94	4a 7f 6b bf 21 40 3a 3c 8d 18 c7 c9 b8 14 d2 22	4a 7f 6b bf 40 3a 3c 21 c7 c9 8d 18 22 b8 14 d2	b1 c1 0b cc ba f3 8b 07 f9 1f 6a c3 1d 19 24 5c	c0 89 57 b1 af 2f 51 ae df 6b ad 7e 39 67 06 c0
71 48 5c 7d 15 dc da a9 26 74 c7 bd 24 7e 22 9c	a3 52 4a ff 59 86 57 d3 f7 92 c6 7a 36 f3 93 de	a3 52 4a ff 86 57 d3 59 c6 7a f7 92 de 36 f3 93	d4 11 fe 0f 3b 44 06 73 cb ab 62 37 19 b7 07 ec	2c a5 f2 43 5c 73 22 8c 65 0e a3 dd f1 96 90 50
f8 b4 0c 4c 67 37 24 ff ae a5 c1 ea e8 21 97 bc	41 8d fe 29 85 9a 36 16 e4 06 78 87 9b fd 88 65	41 8d fe 29 9a 36 16 85 78 87 e4 06 65 9b fd 88	2a 47 c4 48 83 e8 18 ba 84 18 27 23 eb 10 0a f3	58 fd 0f 4c 9d ee cc 40 36 38 9b 46 eb 7d ed bd
72 ba cb 04 1e 06 d4 fa b2 20 bc 65 00 6d e7 4e	40 f4 1f f2 72 6f 48 2d 37 b7 65 4d 63 3c 94 2f	40 f4 1f f2 6f 48 2d 72 65 4d 37 b7 2f 63 3c 94	7b 05 42 4a 1e d0 20 40 94 83 18 52 94 c4 43 fb	71 8c 83 cf c7 29 e5 a5 4c 74 ef a9 c2 bf 52 ef
0a 89 c1 85 d9 f9 c5 e5 d8 f7 f7 fb 56 7b 11 14	67 a7 78 97 35 99 a6 d9 61 68 68 0f b1 21 82 fa	67 a7 78 97 99 a6 d9 35 68 0f 61 68 fa b1 21 82	ec 1a c0 80 0c 50 53 c7 3b d7 00 ef b7 22 72 e0	37 bb 38 f7 14 3d d8 7d 93 e7 08 a1 48 f7 a5 4a
db a1 f8 77 18 6d 8b ba a8 30 08 4e ff d5 d7 aa	b9 32 41 f5 ad 3c 3d f4 c2 04 30 2f 16 03 0e ac	b9 32 41 f5 3c 3d f4 ad 30 2f c2 04 ac 16 03 0e	b1 1a 44 17 3d 2f ec b6 0a 6b 2f 42 9f 68 f3 b1	48 f3 cb 3c 26 1b c3 be 45 a2 aa 0b 20 d7 72 38
f9 e9 8f 2b 1b 34 2f 08 4f c9 85 49 bf bf 81 89	99 1e 73 f1 af 18 15 30 84 dd 97 3b 08 08 0c a7	99 1e 73 f1 18 15 30 af 97 3b 84 dd a7 08 08 0c	31 30 3a c2 ac 71 8c c4 46 65 48 eb 6a 1c 31 62	fd 0e c5 f9 0d 16 d5 6b 42 e0 4a 41 cb 1c 6e 56
cc 3e ff 3b a1 67 59 af 04 85 02 aa a1 00 5f 34	4b b2 16 e2 32 85 cb 79 f2 97 77 ac 32 63 cf 18	4b b2 16 e2 85 cb 79 32 77 ac f2 97 18 32 63 cf	4b 86 8a 36 b1 cb 27 5a fb f2 f2 af cc 5a 5b cf	b4 8e f3 52 ba 98 13 4e 7f 4d 59 20 86 26 18 76
ff 08 69 64 0b 53 34 14 84 bf ab 8f 4a 7c 43 b9				

AES Example Avalanche

Round		Number of bits that differ
	0123456789abcdeffedcba9876543210 0023456789abcdeffedcba9876543210	1
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c4a9ad090fc7ff3fc0e8e8ca4dd02a9c	20
2	5c7bb49a6b72349b05a2317ff46d1294 fe2ae569f7ee8bb8c1f5a2bb37ef53d5	58
3	7115262448dc747e5cdac7227da9bd9c ec093dfb7c45343d689017507d485e62	59
4	f867aee8b437a5210c24c1974cffeabc 43efdb697244df808e8d9364ee0ae6f5	61
5	721eb200ba06206dcbd4bce704fa654e 7b28a5d5ed643287e006c099bb375302	68
6	0ad9d85689f9f77bc1c5f71185e5fb14 3bc2d8b6798d8ac4fe36a1d891ac181a	64
7	db18a8ffa16d30d5f88b08d777ba4eaa 9fb8b5452023c70280e5c4bb9e555a4b	67
8	f91b4fbfe934c9bf8f2f85812b084989 20264e1126b219aef7feb3f9b2d6de40	65
9	cca104a13e678500ff59025f3bafaa34 b56a0341b2290ba7dfdfbddcd8578205	61
10	ff0b844a0853bf7c6934ab4364148fb9 612b89398d0600cde116227ce72433f0	58

International Data Encryption Algorithm (Idea)

- IDEA is a symmetric block cipher designed by Xuejia Lai and James L. Massey in 1991
- It is a minor revision of an earlier cipher, PES (Proposed Encryption Standard)
- IDEA was originally called IPES (Improved PES) and was developed to replace DES.

IDEA contd..

- It entirely avoids the use of any lookup tables or S-boxes
- IDEA was used as the symmetric cipher in early versions of the Pretty Good Privacy cryptosystem

International Data Encryption Algorithm (Idea)

- IDEA operates on 64-bit blocks (In each round 64 bit block is partitioned into 4 partitions each of 16 bits)
- Using a 128- bit key (from which 52 subkeys are generated), and
- consists of a series of eight identical transformations (a round, in each round 6 subkeys are used -16 bits each) and an output transformation (the half-round, which uses 4 subkeys -16 bits each).

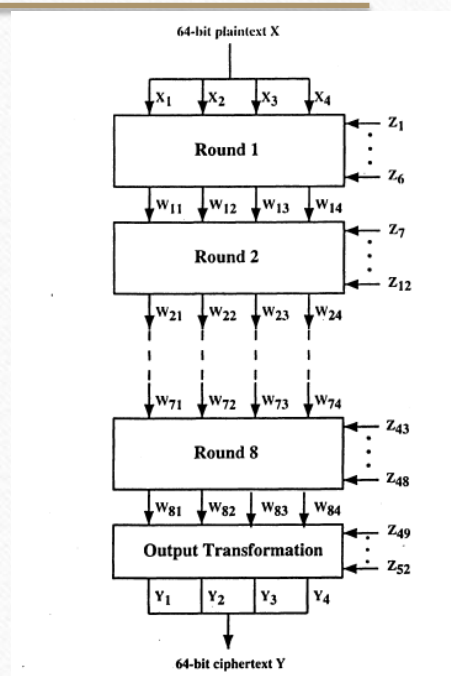


Figure 4.4 Overall IDEA Structure.

- The 64-bit plaintext block is partitioned into four 16-bit sub-blocks
- six 16-bit key are generated from the 128-bit key. Since a further four 16-bit key-sub-blocks are required for the subsequent output transformation, a total of 52 (= 8 x 6 + 4) different 16-bit sub-blocks have to be generated from the 128-bit key.

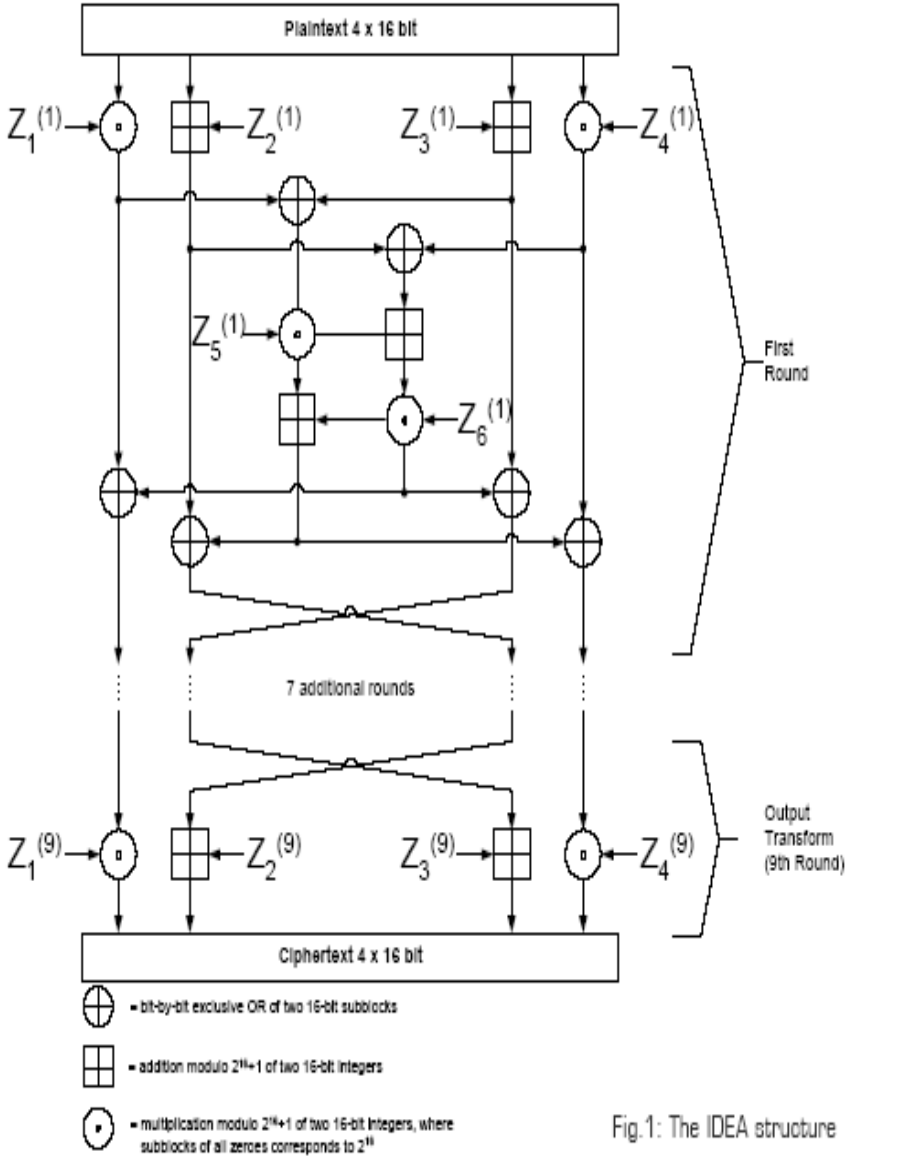


Fig.1: The IDEA structure

Key generation process

- First, the 128-bit key is partitioned into eight 16-bit sub-blocks which are then directly used as the first eight key sub-blocks
- The 128-bit key is then cyclically shifted to the left by 25 positions, after which the resulting 128-bit block is again partitioned into eight 16-bit sub-blocks to be directly used as the next eight key sub-blocks
- The cyclic shift procedure described above is repeated until all of the required 52 16-bit key sub-blocks have been generated

Encryption of the key sub-blocks

- The key sub-blocks used for the encryption and the decryption in the individual rounds are shown in Table 1

Encryption of the key sub-blocks

Round 1	$Z_1^{(1)}\ Z_2^{(1)}\ Z_3^{(1)}\ Z_4^{(1)}\ Z_5^{(1)}\ Z_6^{(1)}$
Round 2	$Z_1^{(2)}\ Z_2^{(2)}\ Z_3^{(2)}\ Z_4^{(2)}\ Z_5^{(2)}\ Z_6^{(2)}$
Round 3	$Z_1^{(3)}\ Z_2^{(3)}\ Z_3^{(3)}\ Z_4^{(3)}\ Z_5^{(3)}\ Z_6^{(3)}$
Round 4	$Z_1^{(4)}\ Z_2^{(4)}\ Z_3^{(4)}\ Z_4^{(4)}\ Z_5^{(4)}\ Z_6^{(4)}$
Round 5	$Z_1^{(5)}\ Z_2^{(5)}\ Z_3^{(5)}\ Z_4^{(5)}\ Z_5^{(5)}\ Z_6^{(5)}$
Round 6	$Z_1^{(6)}\ Z_2^{(6)}\ Z_3^{(6)}\ Z_4^{(6)}\ Z_5^{(6)}\ Z_6^{(6)}$
Round 7	$Z_1^{(7)}\ Z_2^{(7)}\ Z_3^{(7)}\ Z_4^{(7)}\ Z_5^{(7)}\ Z_6^{(7)}$
Round 8	$Z_1^{(8)}\ Z_2^{(8)}\ Z_3^{(8)}\ Z_4^{(8)}\ Z_5^{(8)}\ Z_6^{(8)}$
Output Transform	$Z_1^{(9)}\ Z_2^{(9)}\ Z_3^{(9)}\ Z_4^{(9)}$

Table 1

Short cut steps of IDEA

1. Multiply X_1 and the first subkey Z_1.
2. Add X_2 and the second subkey Z_2.
3. Add X_3 and the third subkey Z_3.
4. Multiply X_4 and the fourth subkey Z_4.
5. Bitwise XOR the results of steps 1 and 3.
6. Bitwise XOR the results of steps 2 and 4.
7. Multiply the result of step 5 and the fifth subkey Z_5.
8. Add the results of steps 6 and 7.
9. Multiply the result of step 8 and the sixth subkey Z_6.
10. Add the results of steps 7 and 9.
11. Bitwise XOR the results of steps 1 and 9.
12. Bitwise XOR the results of steps 3 and 9.
13. Bitwise XOR the results of steps 2 and 10.
14. Bitwise XOR the results of steps 4 and 10.

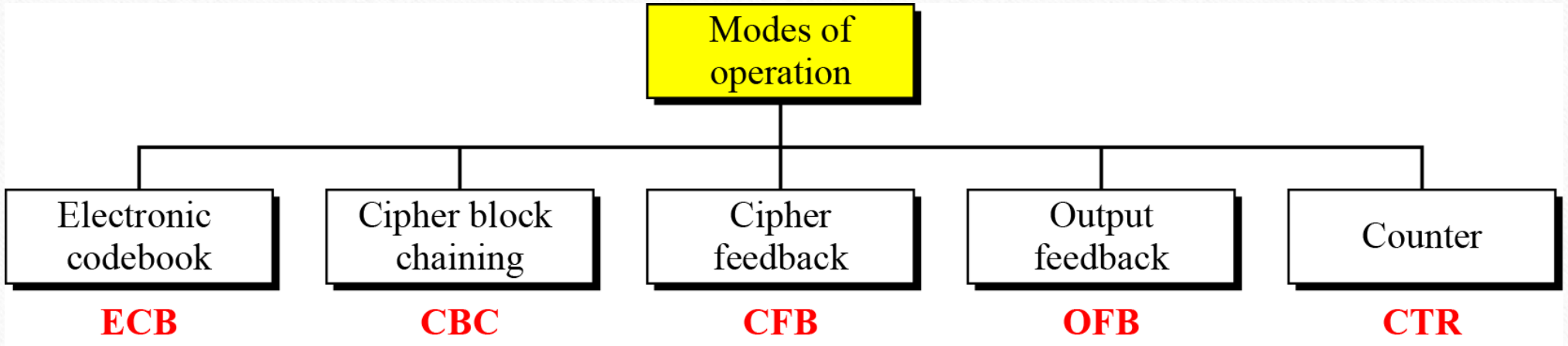
For every round except the final transformation, a swap occurs, and the input to the next round is: result of step 11 $\|$ result of step 13 $\|$ result of step 12 $\|$ result of step 14, which becomes $X_1 \| X_2 \| X_3 \| X_4$, the input for the next round.

After round 8, a ninth "half round" final transformation occurs:

Modes of Operation(confidentiality modes)

- When multiple blocks of plaintext are encrypted using the same key, a number of security issues arise.
- a mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream.
- To apply a block cipher in a variety of applications, five *modes of operation* have been defined by NIST

Different Operation Modes



More Terms

- Initialize Vector (IV)
  - a block of bits to randomize the encryption and hence to produce distinct ciphertext.
- Nonce : Number (used) Once
  - Random of psuedorandom number to ensure that past communications can not be reused in replay attacks
  - Some also refer to initialize vector as nonce
- Padding
  - final block may require a padding to fit a block size
  - Method
    - Add null Bytes
    - Add 0x80 and many 0x00
    - Add the *n* bytes with value *n*

Electronic Codebook (ECB)

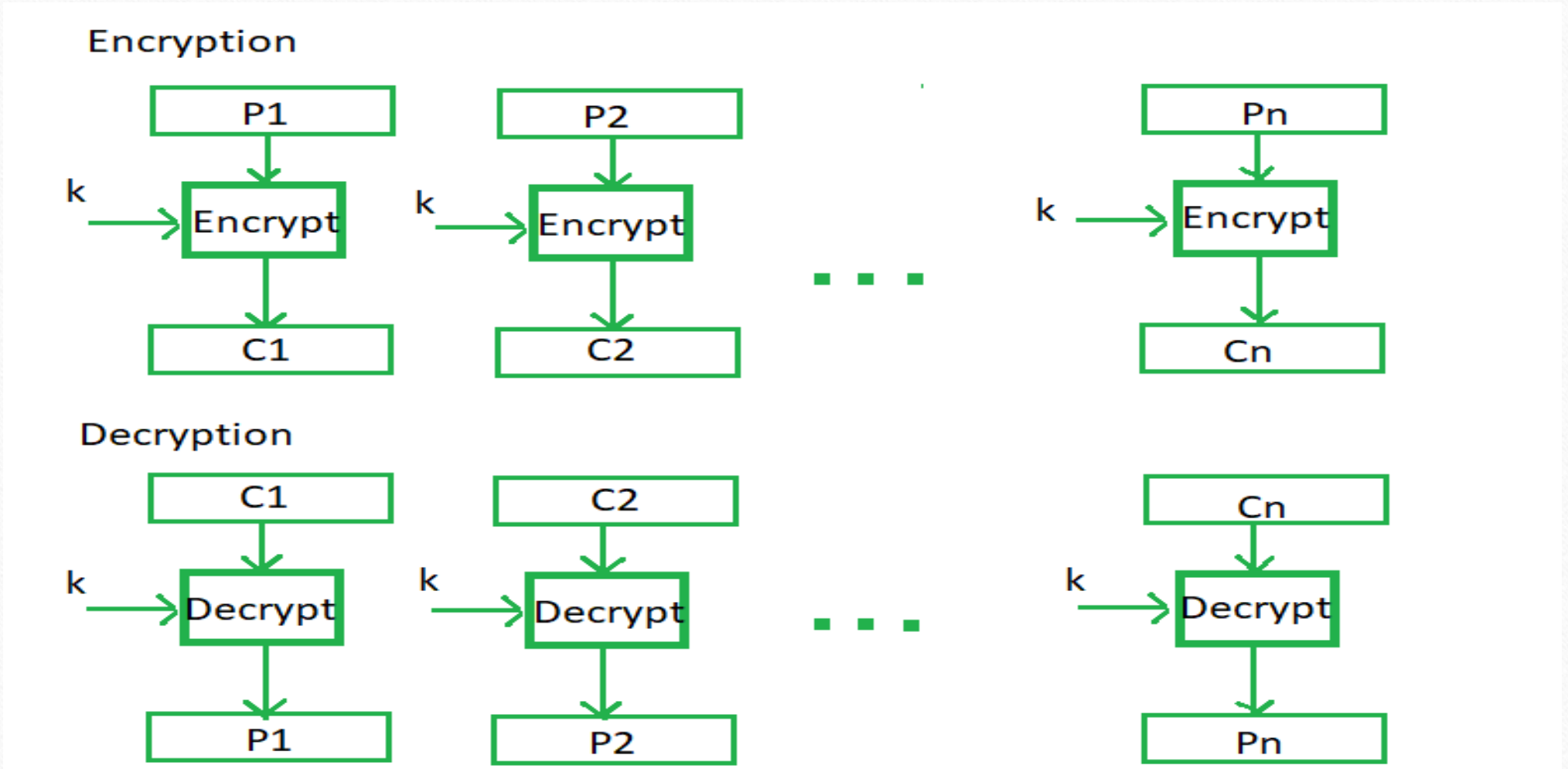
- The simplest mode is the **electronic codebook** (**ECB**) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key .
- The term *codebook* is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.
- For a message longer than 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.

ECB contd..

In Figure , the plaintext
(padded as necessary) consists of a sequence of b-bit blocks, $P1$, $P2$,… , PN; the corresponding sequence of ciphertext blocks is $C1$, $C2$, … , ..CN . We can define ECB mode as follows.

ECB	$C_j = E(K, P_j)$	$j = 1, \ldots, N$	$P_j = D(K, C_j)$	$j = 1, \ldots, N$

The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES or AES key securely, ECB is the appropriate mode to use.
The most significant characteristic of ECB is that if the same -bit block of plaintext appears more than once in the message, it always produces the same ciphertext.
For lengthy messages, the ECB mode may not be secure.


Encryption
P1
k
Encrypt
C1
P2
k
Encrypt
C2
Pn
k
Encrypt
Cn
Decryption
C1
k
Decrypt
P1
C2
k
Decrypt
P2
Cn
k
Decrypt
Pn

Remarks on ECB

- **Strength:** it's simple.
- **Weakness:**
  - If the same message is encrypted (with the same key) and sent twice, their ciphertext are the same.
- **Typical application:**
  - secure transmission of short pieces of information (e.g. a temporary encryption key)

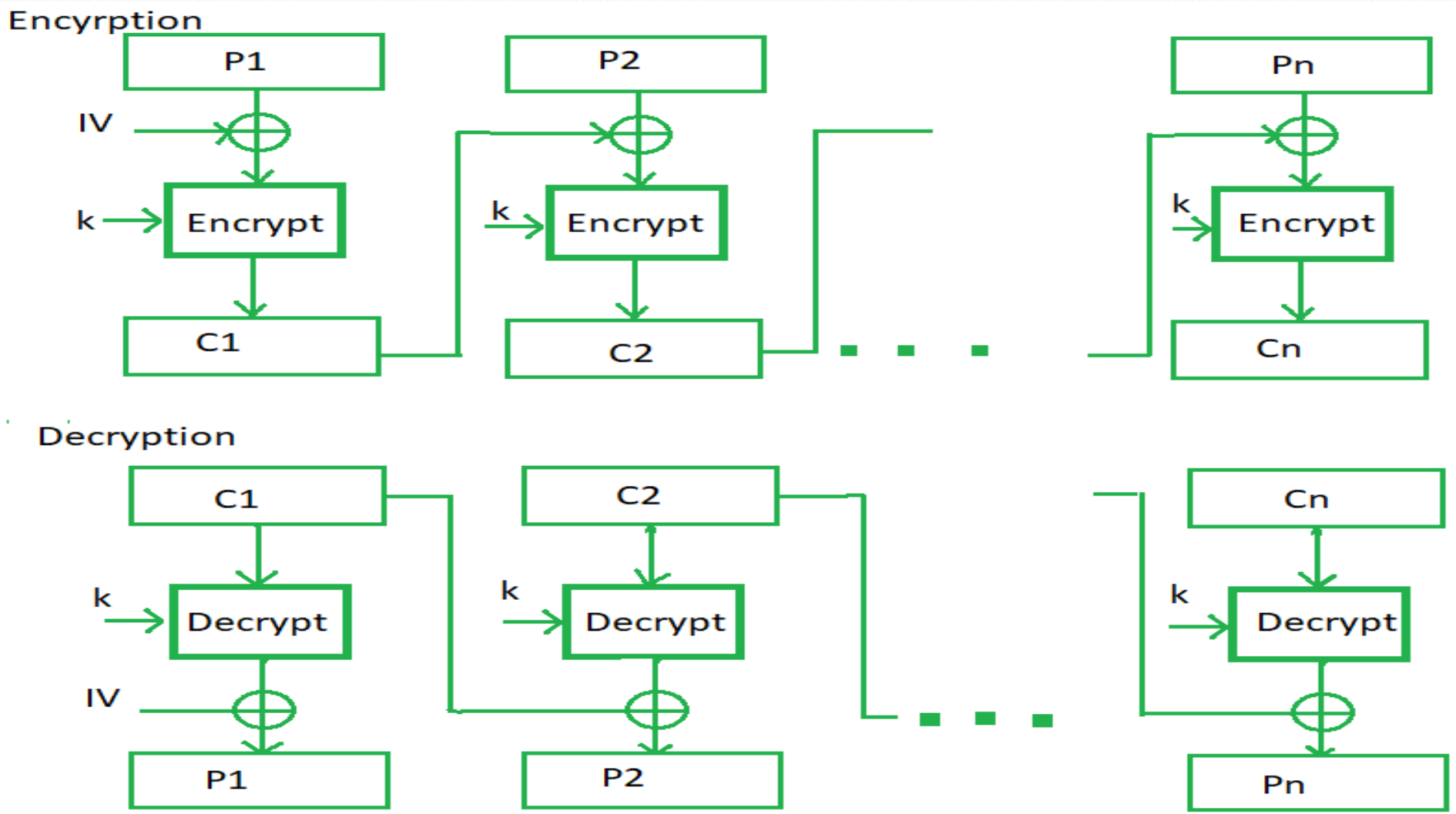
CIPHER BLOCK CHAINING MODE

- To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext 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 block and the preceding ciphertext block; the same key is used for each block.

CIPHER BLOCK CHAINING MODE

- In effect, we have chained together the processing of the sequence of plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of bits are not exposed. As with the ECB mode, the CBC mode requires that the last block be padded to a full bits if it is a partial block

- For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block. To see that this works, we can write


Encyrption
P1
P2
Pn
IV
k
Encrypt
k
Encrypt
k
Encrypt
C1
C2
Cn
Decryption
C1
C2
Cn
k
Decrypt
k
Decrypt
k
Decrypt
IV
P1
P2
Pn

- To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext.
- On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext. The IV is a data block that is that same size as the cipher block. We can define CBC mode as

CBC	$C_1 = \mathrm{E}(K, [P_1 \oplus \mathrm{IV}])$ $C_j = \mathrm{E}(K, [P_j \oplus C_{j-1}])\ \ j = 2, \ldots, N$	$P_1 = \mathrm{D}(K, C_1) \oplus \mathrm{IV}$ $P_j = \mathrm{D}(K, C_j) \oplus C_{j-1}\ \ \ j = 2, \ldots, N$

- The IV must be known to both the sender and receiver but be unpredictable by a third party.

CIPHER FEEDBACK MODE

- For AES, DES, or any block cipher, encryption is performed on a block of bits. In the case of DES b=64, and in the case of AES,b=128.
- However, it is possible to convert a block cipher into a stream cipher, using one of the three modes
  - **cipher feedback** (CFB) mode,
  - **output feedback** (OFB) mode, and
  - **counter** (CTR) mode

CIPHER FEEDBACK MODE

- A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.
- One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be encrypted to produce a ciphertext output of 8 bits.
- If more than 8 bits are produced, transmission capacity is wasted.

CIPHER FEEDBACK MODE

- In this mode the cipher is given as feedback to the next block of encryption with some new specifications: first, an initial vector IV is used for first encryption and output bits are divided as a set of *s* and *b*-*s* bits.
- The left-hand side *s* bits are selected along with plaintext bits to which an XOR operation is applied.
- The result (C1 of s bits) is given as input to a shift register where contents of IV are left shifted by ‘s’ bits and put s-bits of c1 into shift register (i.e. b-s bits to lhs,s bits to rhs and the process continues.

CIPHER FEEDBACK MODE

- Or It can be also expressed as "The input to the encryption function is a –b-bit shift register that is initially set to some initialization vector (IV).The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment of plaintext P1 to produce the first unit of ciphertext C1, which is then transmitted. In addition, the contents of the shift register are shifted left by s bits, and C1 is placed in the rightmost (least significant) bits of the shift register. This process continues until all plaintext units have been encrypted."

CIPHER FEEDBACK MODE

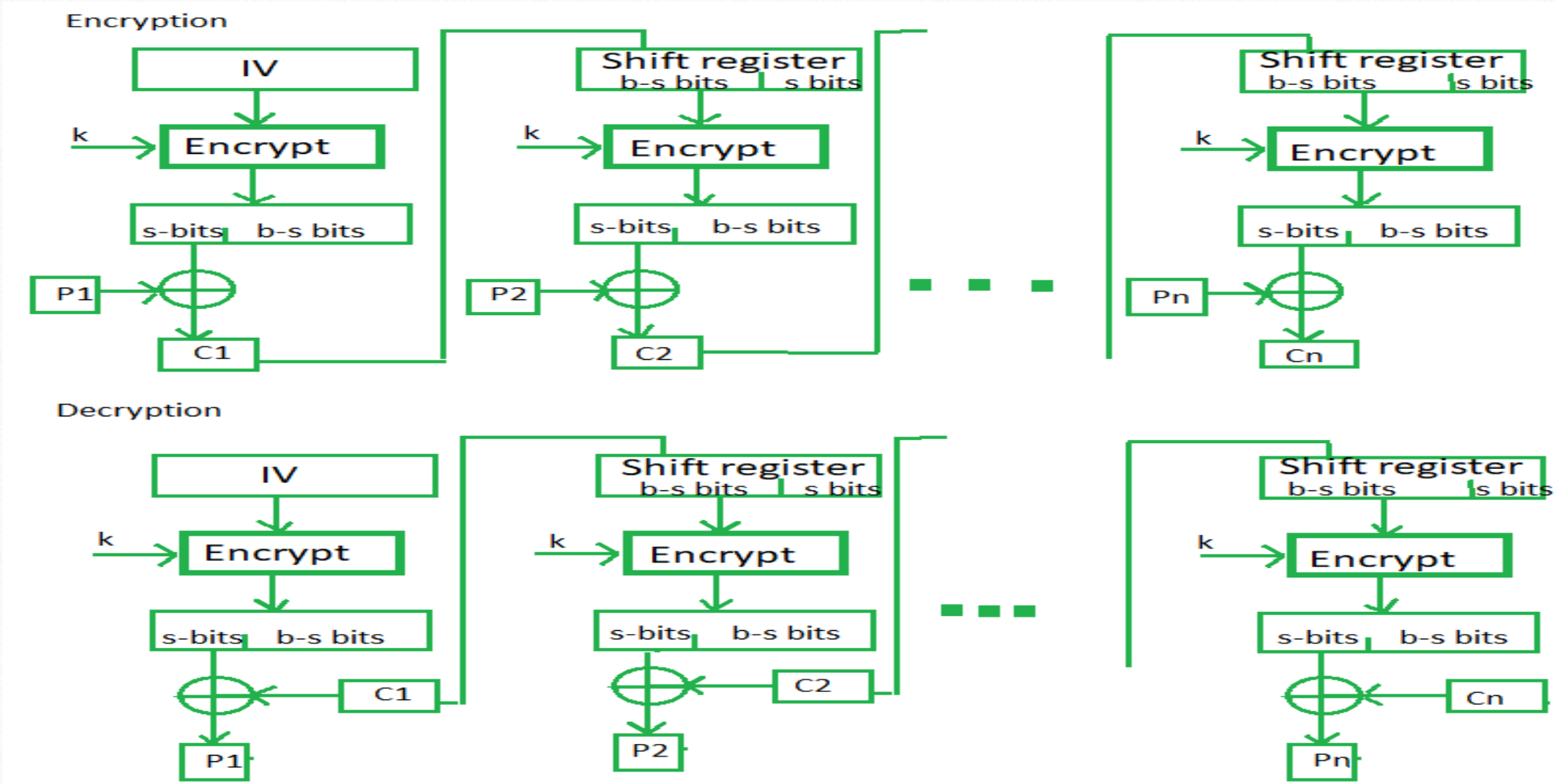
- For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit.
- Let MSB$_s(X)$ be defined as the most significant s bits of X.

$$C_1 = P_1 \oplus \text{MSB}_s[\text{E}(K, \text{IV})]$$

Therefore, by rearranging terms:

$$P_1 = C_1 \oplus \text{MSB}_s[\text{E}(K, \text{IV})]$$

The same reasoning holds for subsequent steps in the process.


Encryption
IV
k
Encrypt
s-bits
b-s bits
P1
C1
Shift register
b-s bits
s bits
k
Encrypt
s-bits
b-s bits
P2
C2
Shift register
b-s bits
s bits
k
Encrypt
s-bits
b-s bits
Pn
Cn
Decryption
IV
k
Encrypt
s-bits
b-s bits
C1
P1
Shift register
b-s bits
s bits
k
Encrypt
s-bits
b-s bits
C2
P2
Shift register
b-s bits
s bits
k
Encrypt
s-bits
b-s bits
Cn
Pn

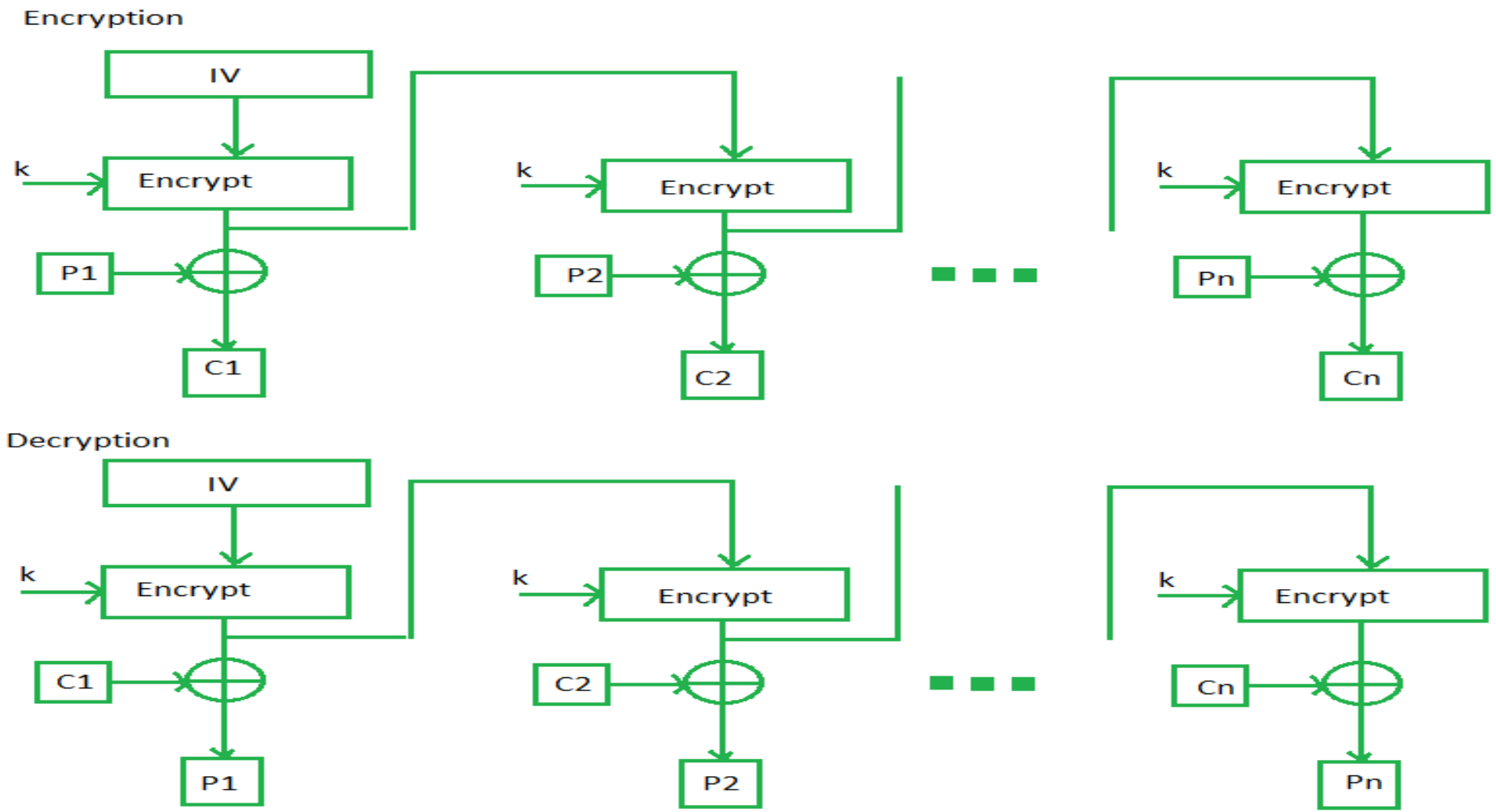
OUTPUT FEEDBACK MODE

- This scheme operates on full blocks of plaintext and ciphertext where the output of the encryption function is fed back to become the input for encrypting the next block of plaintext.
- The other difference is that the OFB mode operates on full blocks of plaintext and ciphertext, not on an s-bit subset.
- Encryption can be expressed as

$$C_j = P_j \oplus \mathrm{E}(K, [C_{j-i} \oplus P_{j-1}])$$

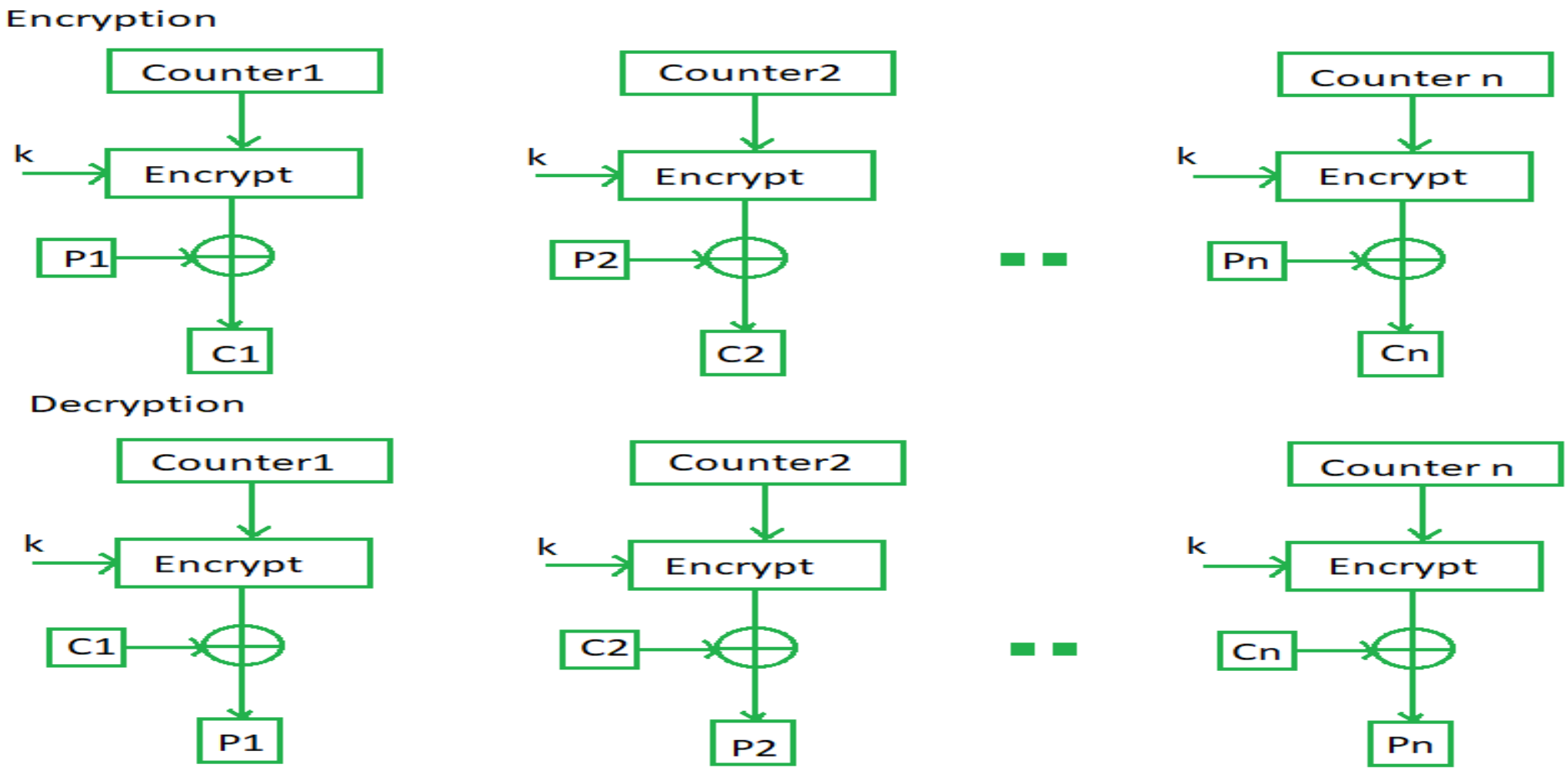
By rearranging terms, we can demonstrate that decryption works.

$$P_j = C_j \oplus \mathrm{E}(K, [C_{j-1} \oplus P_{j-1}])$$


Encryption
IV
k
Encrypt
P1
C1
k
Encrypt
P2
C2
k
Encrypt
Pn
Cn
Decryption
IV
k
Encrypt
C1
P1
k
Encrypt
C2
P2
k
Encrypt
Cn
Pn

Counter (CTR) Mode

- A counter equal to the plaintext block size is used. The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block.
- For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining.
- For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.


Encryption
Counter1
k
Encrypt
P1
C1
Counter2
k
Encrypt
P2
C2
Counter n
k
Encrypt
Pn
Cn
Decryption
Counter1
k
Encrypt
C1
P1
Counter2
k
Encrypt
C2
P2
Counter n
k
Encrypt
Cn
Pn

Block Cipher Modes of Operation

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 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 block of plaintext and the preceding block of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed *s* 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 encryption output, and full blocks are used.	• 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