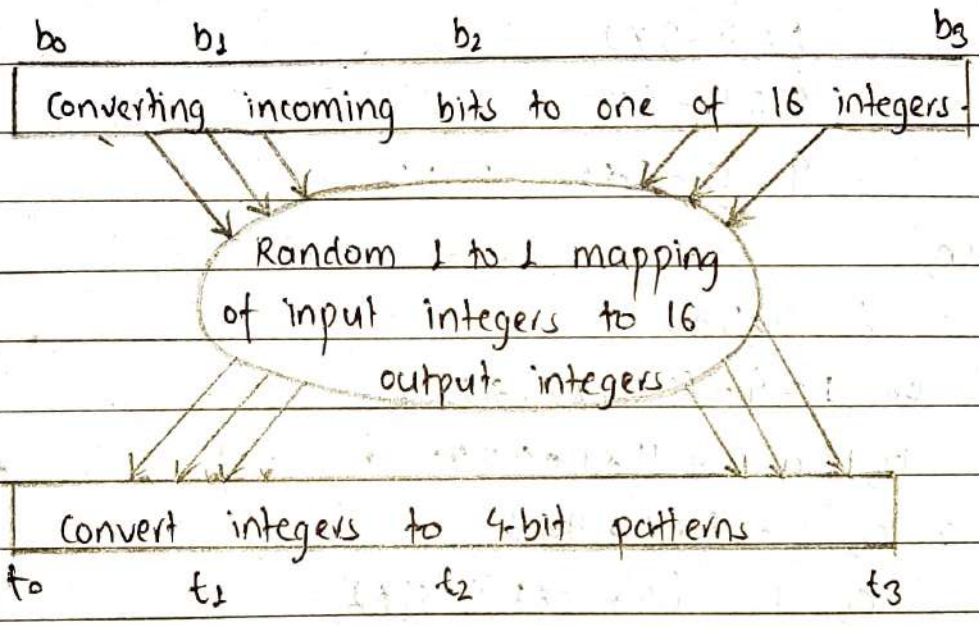
4 Modern Symmetric Ciphers

Binary Block Substitution

In a modern block cipher, we replace a block of N-bits from plaintext with a block of N-bits from a ciphertext. This general idea is illustrated in the figure below: [(for n=4 although N is set to 64 or its multiplier)].



→ Firstly, the 4 bit patterns are converted into $2^4 = 16$ different possible patterns, we can represent these pattern by an integer between 0 and 15. So, bit pattern will be,

0000 → 0
0001 → 1
⋮ ⋮
1111 → 15

↳ In ideal block cipher, relationship between input and output block is completely random. But it must be convertible for decryption work. Therefore, it has to be 1 to 1 mapping. (meaning that at each i/p block, it is mapped to unique output block).

↳ The mapping between input and output block can also be constructed as a mapping from integers corresponding to input block to the integers corresponding to output blocks.

↳ The encryption key for the ideal block cipher is the code block itself meaning the table shows relationship between the i/p and output blocks.

↳ In the above figure, an ideal block cipher is shown that uses block of size 4. Each block of 4-bits in the plaintext is converted into 4 ciphertext bits.

Size of Encryption Key for Binary Block substitution

Consider a 64-bits block encryption with a 64-bit block, we can consider each input block as 2^{64} integers and for each such integers, we can specify output block of 64-bits. We can construct the code-book by displaying just the output blocks in the order of corresponding input blocks. Such a code-book will be of size $64 \times 2^{64} \approx 10^{21}$. This implies that the encryption key for an ideal block cipher using 64 bit-blocks will be of size 10^{21}. This size of the encryption key makes the ideal block cipher an imprachical idea, Thinking of logical issue related to transmission, storage and processing such large keys.

Shannon's theory of Confusion and Diffusion

↳ Block cipher looks extremely large substitution as we need table of 2^{64} entries for 64-bit blocks.

↳ Using the idea of product cipher in 1949, Claude Shannon introduced idea of substitution permutation, (S-P) network called modern substitution transpose product cipher. These form basics for modern block cipher.

↳ S-P network based on two primitive cryptographic operation we've seen:

a) Substitution (S-box)

b) Permutation (P-box)

↳ Provides confusion and diffusion in messages.

Diffusion:

↳ Dissipates the statistical structure of plaintext over block of ciphertext.

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

Confusion:

↳ Makes the relationship between ciphertext and key as complex as possible to make the relationship between the statistics of ciphertext and the value of encryption key as complex as possible, against thwart attempts to discover the keys.

↳ substitution algorithm, stream cipher & block cipher,

Feistel Cipher

→ It implements Shannon's Diffusion and Confusion theory (S-P network concept) based on invertible product cipher.

→ In early 1970, IBM developer Horst Feistel introduced Feistel cipher and it was first implemented by Feistel Horst and Don Coppersmith in Lucifer cipher.

→ The Feistel cipher is a system based on same basic algorithm for encryption and decryption.

→ The Feistel cipher structure consists of multiple rounds of processing of the plaintext with each round consisting of substitution round followed by a permutation round as shown in the figure below:

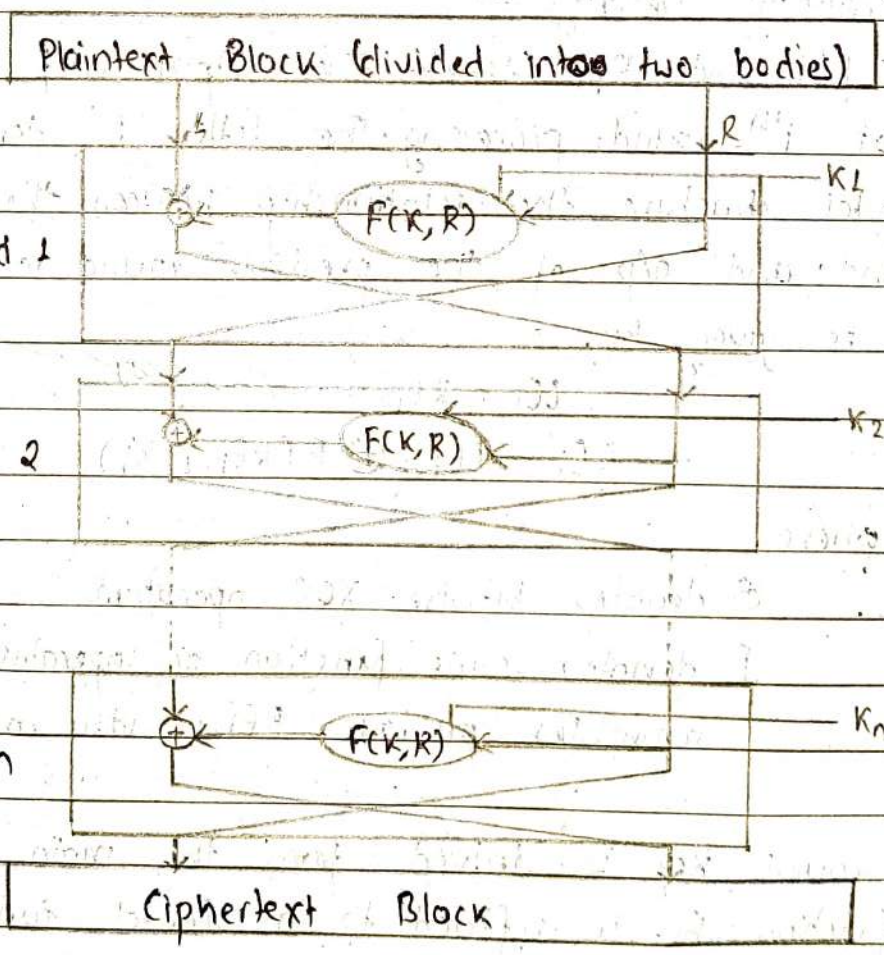


Fig: Feistel Structure of Symmetric Key Cryptography

The plaintext input block is divided into two halves at each round and we denote these halves with L and R for left block and right block respectively. In each round, the right hand of the block ie. R goes through an operation that depends upon R and the encryption key K.

The permutation step at the end of each round consist of swapping the modified L and R. Hence, the current L for the next round would be R and the current R for the next round would be L. The process continues for the next n rounds.

Mathematical Description:

Let LE_i and RE_i denote the output half block at the end of i^{th} round processing. The letter 'E' denotes encryption. In feistel structure, the relationship between the i/p of i^{th} round and o/p of the previous round ie. $(i-1)^{th}$ round is given by,

$$LE_i = RE_{i-1}$$

$$RE_i = LE_{i-1} \oplus F(RE_{i-1}, K_i)$$

where,

$\oplus$ denotes bitwise XOR operation

F denotes some function or operation that scrambles previous RE_{i-1} with round key K_i

The round key is derived from the main encryption key. The function F is referred to as Feistel function after

Horst Feistel (the inventor). If we assume 16 round of processing which is typical, the outputs of the last round of the processing is given by,

$$LE_{16} = RE_{15}$$

$$RE_{16} = LE_{15} \oplus F(RE_{15}, K_{16}).$$

Decryption of Ciphertext based on Feistel Structure:

The decryption algorithm is exactly the same as encryption algorithm with the only difference that the round key are used in reverse order. The output of each round during decryption is the input to the corresponding round during encryption except for left-right switch between the two halves. This property holds true regardless of the choice of Feistel function.

Data Encryption Standard (DES)

↳ IBM in 1974 created DES which was based on their Lucifer Algorithm.

↳ Symmetric key block cipher.

↳ Remarkably well engineered algorithm which has a powerful influence in cryptography.

↳ DES has a key length of 56 bits (+8 parity bits) and a block length of n=64 bits. It consists of 16 rounds of what is called "Feistel Network."

↳ Electronic Frontier Foundation broke DES in 22 hrs and 15 minutes

↳ NIST then issued a new directive which required the organization to be triple DES (3-DES) ie. three consecutive

application of DES.

↳ The round key are generated from the main key by permutation.

↳ The key length is 48 bits.

Plaintext Block (divided into two blocks)

Figure:

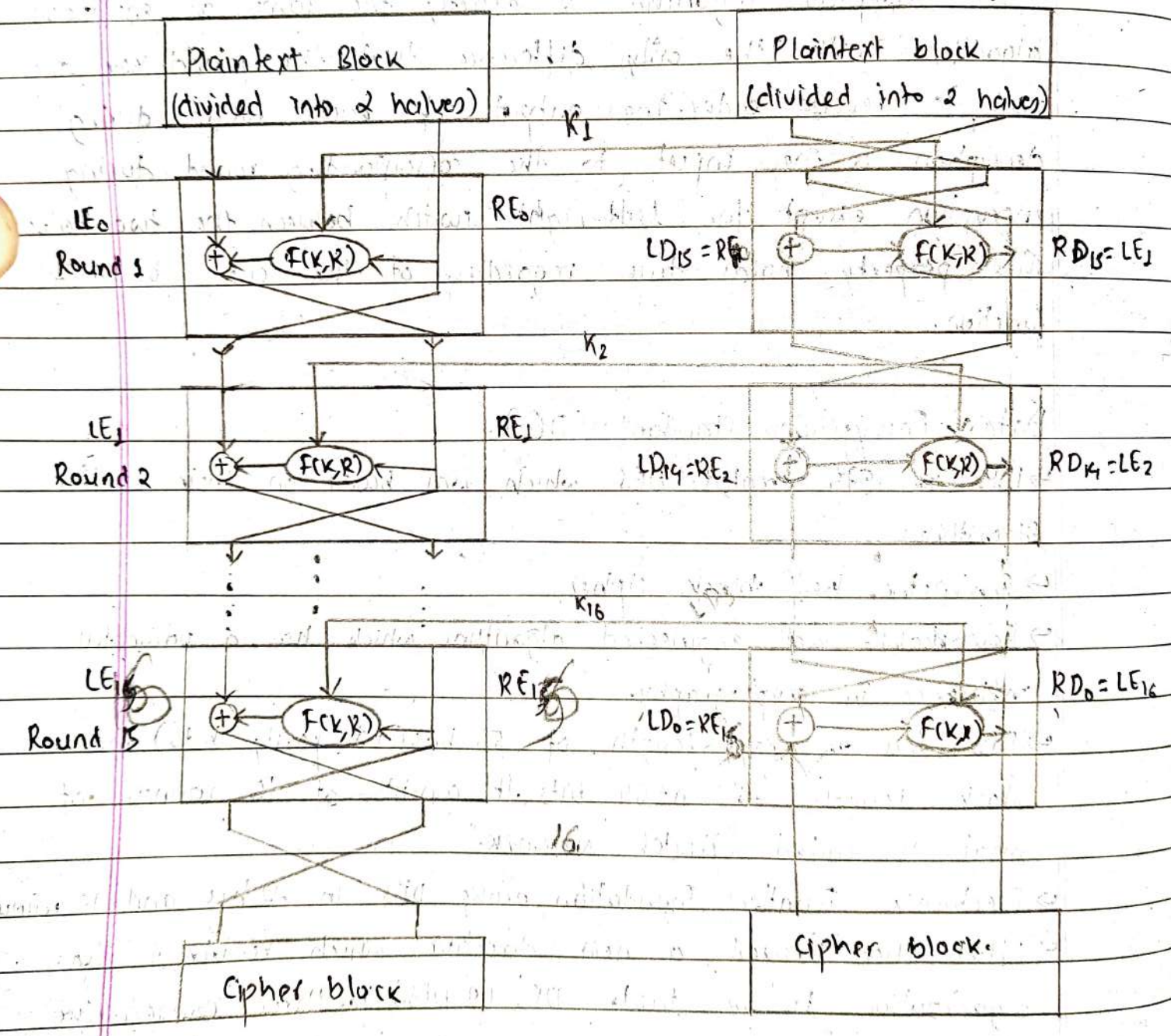


Fig: Encryption and Decryption in Feistel structure with round of 16.

To prove the claim of encryption and decryption, let RD_i and LD_i denote the left half and the right half i^{th} round. Now, we know that LD_1 and RD_1 occured in the first decryption round. The relationship between two halves that are input to the first decryption and the output of decryption algorithm is given by,

$$LD_0 = RE_{16} \quad \text{and} \quad RE_0 = LD_{16}$$

Now, we can write the following equation for the output of the first decryption round,

$$LD_1 = RD_0$$
$$\therefore LD_1 = LE_{16}$$
$$= RE_{15}$$

Again,

$$RD_1 = LD_0 \oplus F(RD_0, K_{16})$$
$$= RE_{16} \oplus F(RE_{15}, K_{16})$$
$$= (LE_{15} \oplus F(RE_{15}, K_{16})) \oplus F(RE_{15}, K_{16})$$
$$= LE_{15}$$

This means that except for the left-right switch, the o/p of the first round of decryption decryption is same as input to the last stage of encryption round.

$LD_0 = RE_{16}$	$RD_1 = LE_{15}$
$RD_0 = LE_{16}$	$RD_1 = RE_{15}$

The following identities have been used to derive the above expression.

- $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- $A \oplus 0 = A$
- $A \oplus A = 0$.

Algorithm Implementation of DES

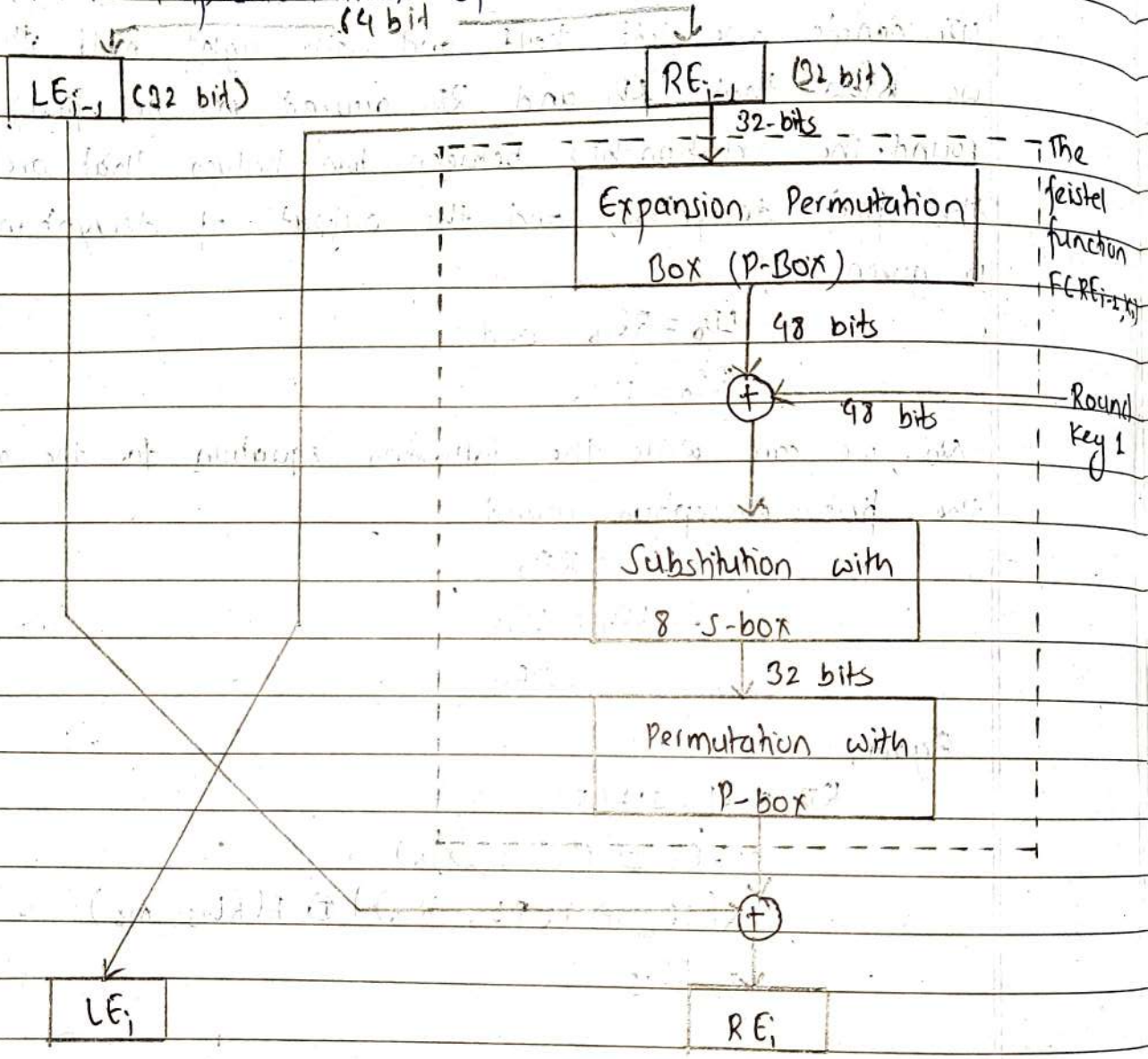
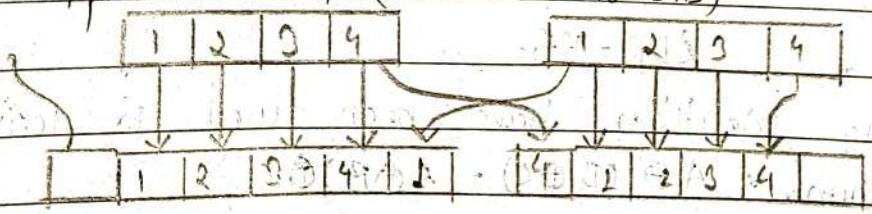


Fig: Single Round Processing of DES

1. Expansion P-Box (32 bits → 48 bits)



1→2, 2→3, 3→4, 4→5, 6→ 1st bit of next block.

2 nibble = 1 byte

The algorithm implementation of DES is called DEA or Data Encryption Algorithm. The figure given shows a single round processing of DES or DEA. The dotted rectangle constitute the Feistel function, $F(RE_{i-1}, K_i)$

The 32 bit right half of the 64-bit input block is expanded into 48 bit block using expansion P-box. This is refferred to as expansion-permutation step or E-step. The E-step consists of the following:

1. The 32-bit block is divided into eight 4-bit nibbles. Then, an addition bit of on the left of each bit is added. which is the last bit of the previous nibble. An additional bit to the right of the each nibble is attached at the end which is the beginning of next 4-bit nibble.

The 56-bit key is divided into 2 halves, each half is separately shifted and combined to yield a 48-bit key. The 48 bit of expanded output produced by that E-step is XORed with the round key. The output produced by previous step is broken into eight 6-bit blocks each of which goes through substitution step to yield a 4-bit block. The substitution is carried out using 8 S-block ie. 8 S-blocks.

After all the substitution, we encode 4 permutation boxes on the 32-bit output. For this purpose, the 32 bit block goes through ~~8-block~~ P-box permutation. The output of P-box is then

XORed with the left half of 64-bit block that we started with, and this output will act as the right half of next round.

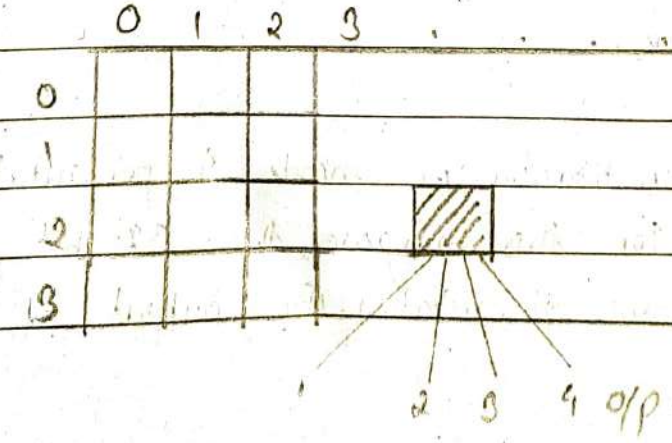
The main goal of substitution step implemented by s-box is to introduce diffusion. ie. each plaintext bit must be acting to affect as many ciphertext bit as possible.

The strategy of creating many round key is form the main key meant to introduce confusion into the encryption process. ie. The key must affect as many bit as possible of the output of the ciphertext.

2. Whitener (XOR operation)
   ↳ XOR between 48 bits key and 48 bits o/p from expansion box.

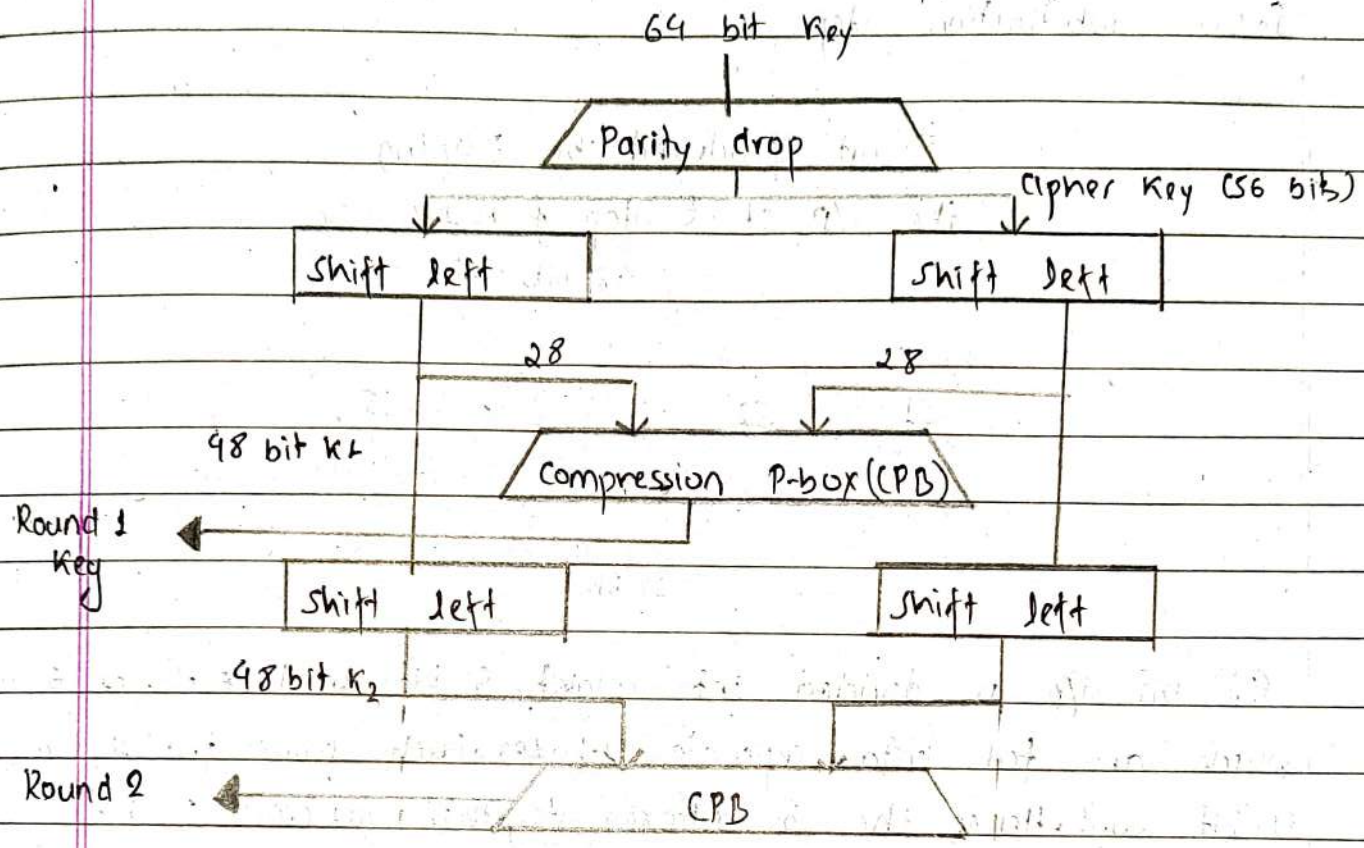
3. Substitution (S-Box)
   ↳ 6 bit input —Transform→ 4 bit o/p
   ↳ 8 bit are used.

Each ~~Subto~~ s-box has different table.



Key Generation:

The round key generator generates sixteen 48-bit keys from the 56 bit cipher key. Actually, the key length is 64 bit out of which 8 parity bits have been dropped.



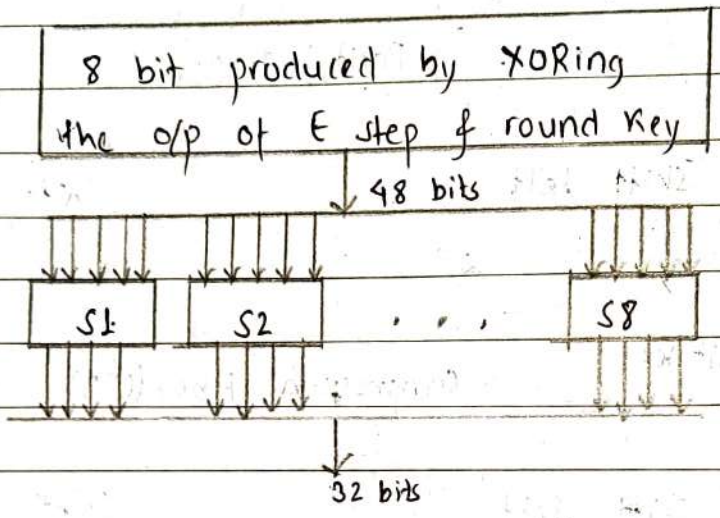
The left shift operation occurs according to the following scheme:

Rounds	Bit shift
1, 2, 9, 16	1 bit
Others	2 bit

So,

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Key shift	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1
Total shift	1	2	4	6	8	10	12	14	15	17	19	21	23	25	27	28

S-Box substitution step



48 bit i/p is divided into eight 6 bit words, each of 6 bits which are fed into separate S-boxes. Each S-box produces a 4 bit word. Hence, the 8 S-boxes together generate 32 bit o/p word.

Each S-box consists of 4x16 lookup table for an o/p 4 bit word. The first and last bit of 6 bit o/p is decoded into one of 4 rows and middle 4 bit decoded into 16 columns of the lookup table.

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

Fig: Table of S-box S1

Permutation in the Feistel Function

The last step of feistel function is permutation with P-box. The permutation is shown below:

P-box Permutation							
15	6	19	20	28	11	27	16
0	14	22	25	4	17	30	9
1	7	23	13	31	26	2	8
18	12	29	5	21	10	3	24

The permutation table says, 0^{th} o/p bit will be the 15^{th} i/p bit and 1^{st} o/p bit be the 6^{th} i/p bit and so on, for all 32 bit o/p that is obtained from the i/p.

The basic process of enciphering a 64 bit data block using DES consists of:

- an initial permutation
- 16 rounds of complex independent calculation 'f'
- a final permutation being inverse of IP.

This can be described as,

$$L(i) = R(i-1)$$

$$R(i) = L(i-1) \oplus P(SCE(R(i-1) \oplus K(i)))$$

and forms one round in the S-P network.

- An initial permutation of the key P(i) which selects 56 bit into 2 halves of 28 bits.

- The 16 round consist of:
  - selecting 24 bit of each halves and permutating them by P(2) for use in 'f'.
  - Rotating each half either 1 or 2 places depending upon the key rotation schedule (KS)
  - This can be described functionally as:

    $K(i) = P2(KS(PC1(K,i)))$

48 bit i/p is divided into eight 4 bit words each of 6 bit which are fed into separate S-boxes. Each S-box produces a 4 bit word. Hence, 8 S-boxes together generate 32 bit o/p word.

Modes of Block Cipher / Stream Cipher

The way we use a block cipher is called mode of use and four have been defined for DES by standard.

Block modes

→ splits the message into box (ECB, CBC)

1. Electronic Code Book

→ where the message (broken into independent 64 bits) blocks are encrypted.

$C(i) = DES(Ki)(P\_(i))$

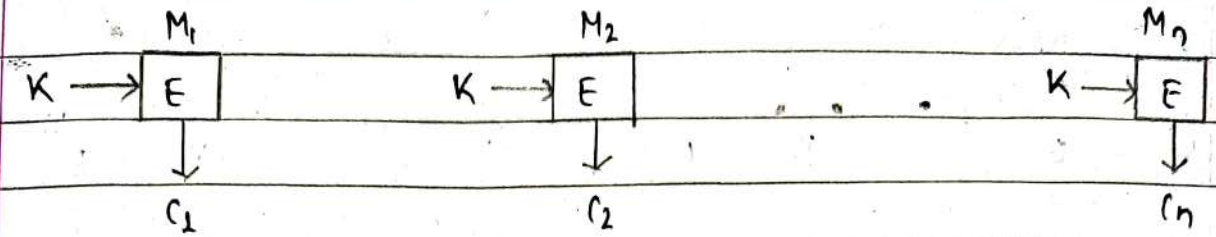


fig: ECB.

2. Cipher Block Chaining

↳ Again, the message is broken into 64 bit blocks, but they are linked together in the encryption operation with an initial vector (IV).

$C(i) = DES_{(K_i)}(P(i) \oplus C_{(i-1)})$

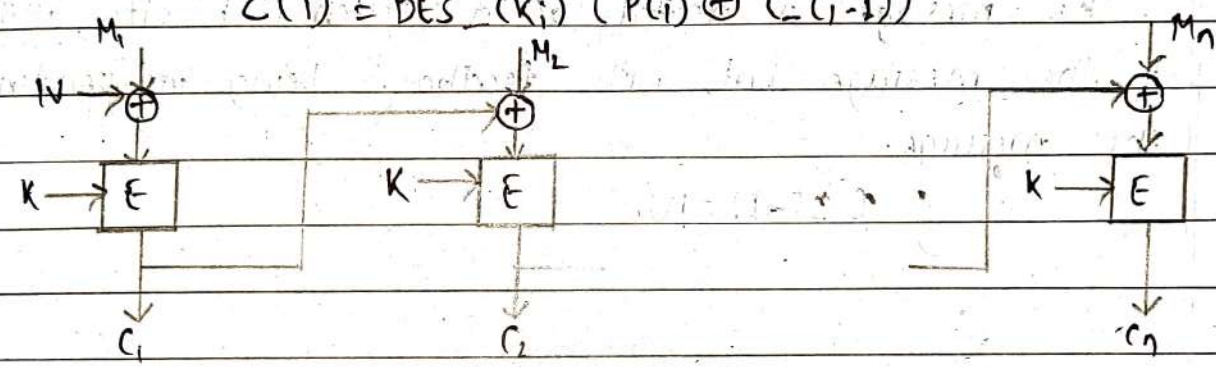


fig: CBC

Stream modes

↳ This is one bit stream mode (CFB, OFB)

1. Cipher Feedback Mode

The message is treated as a stream of bits, added to the o/p of DES, with the result being feedback for next stage.

$C_{(i)} = P_{(1)} \oplus DES_{(K_i)}(C_{(i-1)})$

$C_{(-1)} = IV.$

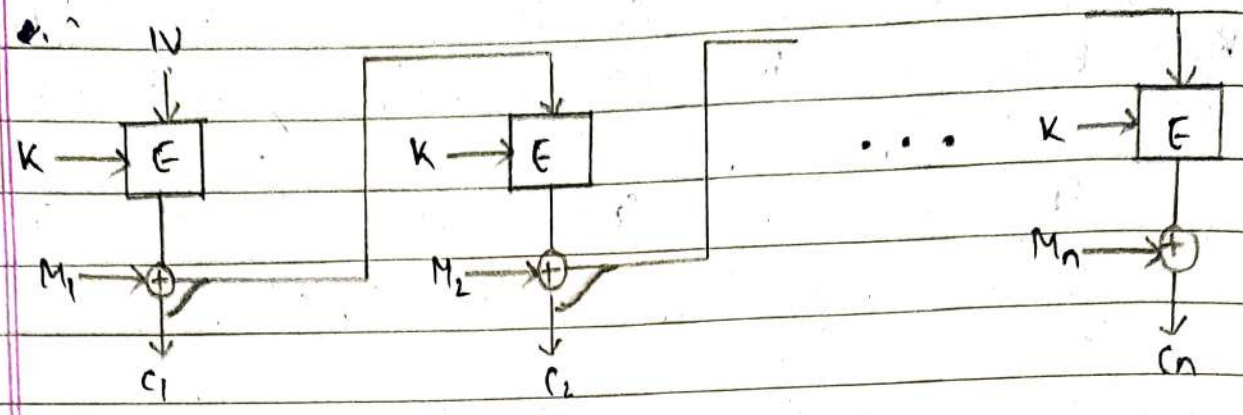


fig: Cipher Feed Back Mode

2. Output feedback mode.

The message is treated as a system of bits, added to the message but with feedback being independent of the message.

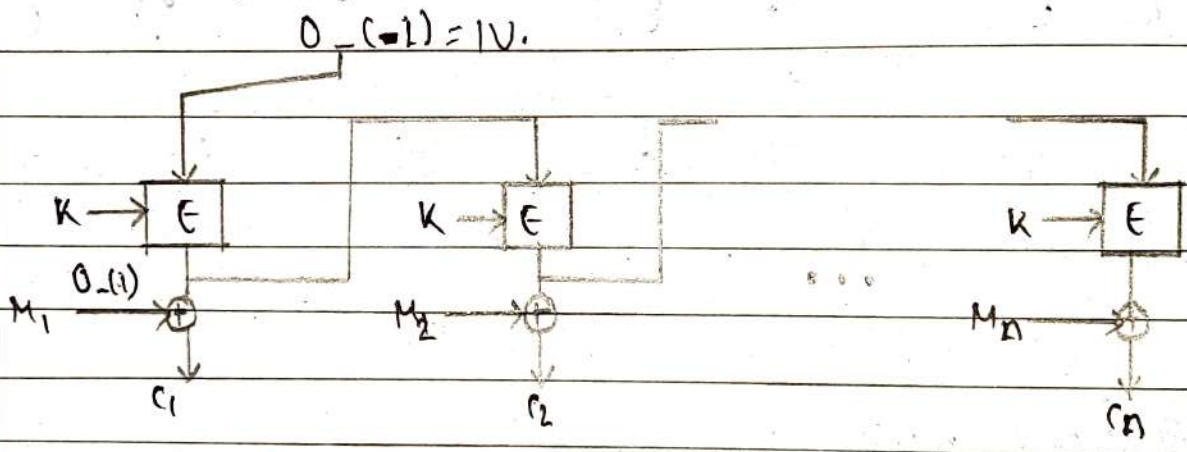


fig: Output Feedback Mode

$C_{(i)} = P_{(i)} \oplus O_{(i)}$

$O_{(i)} = DES_{(Ki)}(O_{(i-1)})$

$O_{(-1)} = IV$

Triple DES or 3DES can be used to make DEA algorithm more ~~rat~~ robust and secure against unauthorized and unauthorized approaches.

International Data Encryption Algorithm (IDEA)

IDEA was developed by James Massey and Xeujia Lai in 1990 at ETH Zurich. It was originally called IPES's. In 1992, the name was changed to IDEA. IDEA encrypts 64 bit message blocks using 128 days bits keys to obtain 64 bit cipher block. It is based on mixing operation from different algebraic groups (XOR, Addition mod 2^{16}, Multiplication mod $2^{16}+1$)

An operation is performed on 16 bit block with no permutation. Hence, it is very efficient in software implementation. IDEA is patented in Europe and US. However, non-commercial use is freely permitted.

It is used in public domain PGP secure email system. Currently, there is no attack against IDEA that are known to us as the key is too long for exhaustive search.

Encryption process

It encrypts 64 bit plain text into 64 bit cipher text block using a 128 bit key input key (say K). It is based on novel generalization of Feistel structure. It consists of 8 computational identical rounds followed by an output transformation. A round r consists of six 16 bits sub keys ie. K_i^r $(1 \leq i \leq 6)$, To transform the 64 bit input x into an output of four 16 bit blocks which are the input to the next round.

Round 8 output enters to the output transformation employing for additional sub keys K_i^9 for $1 \le i \le 4$ to produce the final ciphertext. $y = \{y_1, y_2, y_3, y_4\}$.

→ The dominant design concept of IDEA is mixing operation from three different algebraic groups of 2^n elements.

↳ The corresponding group operations of sub-blocks a & b of bit length $n=16$ are:

1. bitwise XOR : $a \oplus b$
2. addition mod 2^n : $(a+b)$ AND 0XFFFF (16 bit binary operation) denoted by $\boxplus$.
3. Modified multiplication : mod 2^n+1 with $0 \in Z_{2^n}$ associated with $2^n \in Z_{2^n+1}$; denoted by $a \odot b$.

IDEA : sub-key gener

The encryption key is obtained by splitting 128 key bit into 8 eight 16 bit sub keys. once this key is used, the key is rotated 25 bits and broken up again. The decryption is little more complex since inverse of sub-blocks need to be calculated.

IDEA Algorithm

The input is 64 bit plaintext : $m = m_1, m_2, m_3, m_4$.
The output is 64 bit ciphertext $y = y_1, y_2, y_3, y_4$.
The key length is 128 bits.

1. Key Schedule

Compute 16 bit sub keys for each of the eight rounds.

$K_1^{(r)}$ $K_2^{(r)}$ ... $K_{16}^{(r)}$.

2. Divide the 64 bit message block into four 16 bit blocks.

m_1 m_2 m_3 ... m_{64} $\Rightarrow$ x_1 x_2 x_3 x_4

16 bit 16 bit 16 bit 16 bit

3. For round r, from 1 to 8, do

a) $x_1 \leftarrow x_1 \odot K_1^{(r)}$, $x_4 \leftarrow x_4 \odot K_4^{(r)}$,

$x_2 \leftarrow x_2 \boxplus K_2^{(r)}$, $x_3 \leftarrow x_3 \boxplus K_3^{(r)}$.

b) $t_0 \leftarrow K_5^{(r)} \cdot (x_1 \oplus x_3)$, $t_1 \leftarrow K_6^{(r)} (t_0 \boxplus (x_2 \odot x_4))$,

$t_2 \leftarrow t_0 \boxplus t_1$

c) $x_1 \leftarrow x_1 \oplus t_1$, $x_4 \leftarrow x_4 \oplus t_2$, $a \leftarrow x_2 \oplus t_2$, $x_3 \leftarrow x_3 \oplus x$,

$x_3 \leftarrow a$

4. Output Transformation

$y_1 \leftarrow x_1 \odot K_1^{(9)}$, $y_4 \leftarrow x_4 \odot K_4^{(9)}$

$y_2 \leftarrow x_3 \boxplus K_2^{(9)}$, $y_3 \leftarrow x_2 \boxplus K_3^{(9)}$

The input to the output transformation is 128 bit key and the output is 52 bit.

⊙ - mod 2^n+1 multiplication
⊕ - addition mod 2^n

128 — $K_1, K_2, \ldots K_{128}$

52-bit subkeys

$K_1^1, K_2^1, \ldots K_6^1$

$K_1^2, K_2^2, \ldots K_6^2$

$K_1^8, K_2^8, \ldots K_6^8$

OR

3. step 3 of above algorithm can also be expressed as:

① $X_1 \times K_1$

② $X_2 + K_2$

③ $X_3 + K_3$

④ $X_4 \times K_4$

⑤ step ① + step ③

⑥ step ② + step ④

⑦ step ⑤ $\times K_5$

⑧ step ⑥ + step ⑦

⑨ step ⑧ $\times K_6$

⑩ step ⑦ + step ⑨

⑪ step ① ⊕ step ⑨ $\longrightarrow R_1$

⑫ step ③ ⊕ step ⑨ $\longrightarrow R_2$

⑬ step ② ⊕ step ⑩ $\longrightarrow R_3$

⑭ step ④ ⊕ step ⑩ $\longrightarrow R_4$.

Single Round of IDEA.

⊙ → Multiplicative mod $2^n + 1$

⊕ → bitwise XOR

⊞ → Addition mod 2^n

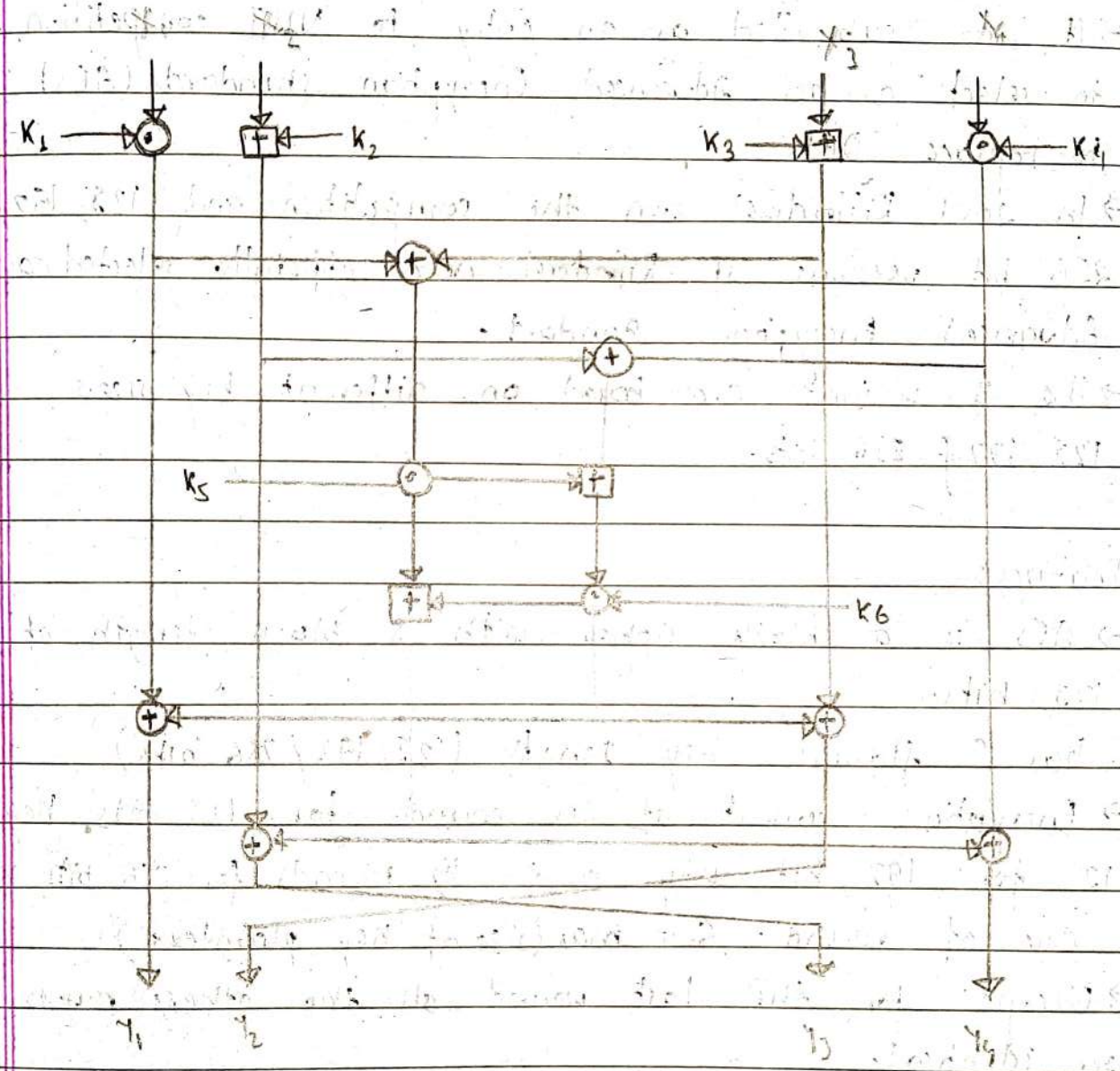


Fig: Single Round of IDEA

Advanced Encryption Standard (AES)

→ Rijndael is a family of block cipher developed by Belgian cryptographers Vincent Rijmen and Joen Daemen.

→ It was submitted as an entry to NIST competition to select as an Advanced Encryption standard (AES) to replace DES.

→ In 2001, Rijindael won the competition and 128, 192, 256 bit versions of Rijindael were officially selected as Advanced Encryption Standard.

→ The 3 variants are based on different key sizes 128, 192 & 256 bits.

Features:

→ AES is a block cipher with a block length of 128 bits.

→ has 3 flavours key length (128/192/256 bits)

→ Encryption consists of 10 rounds for 128 bits key 12 for 192 bits key and 14 rounds for 256 bit key

No. of round = 6 + max (size of key, plaintext)

→ Except for the last round all the others rounds are identical.

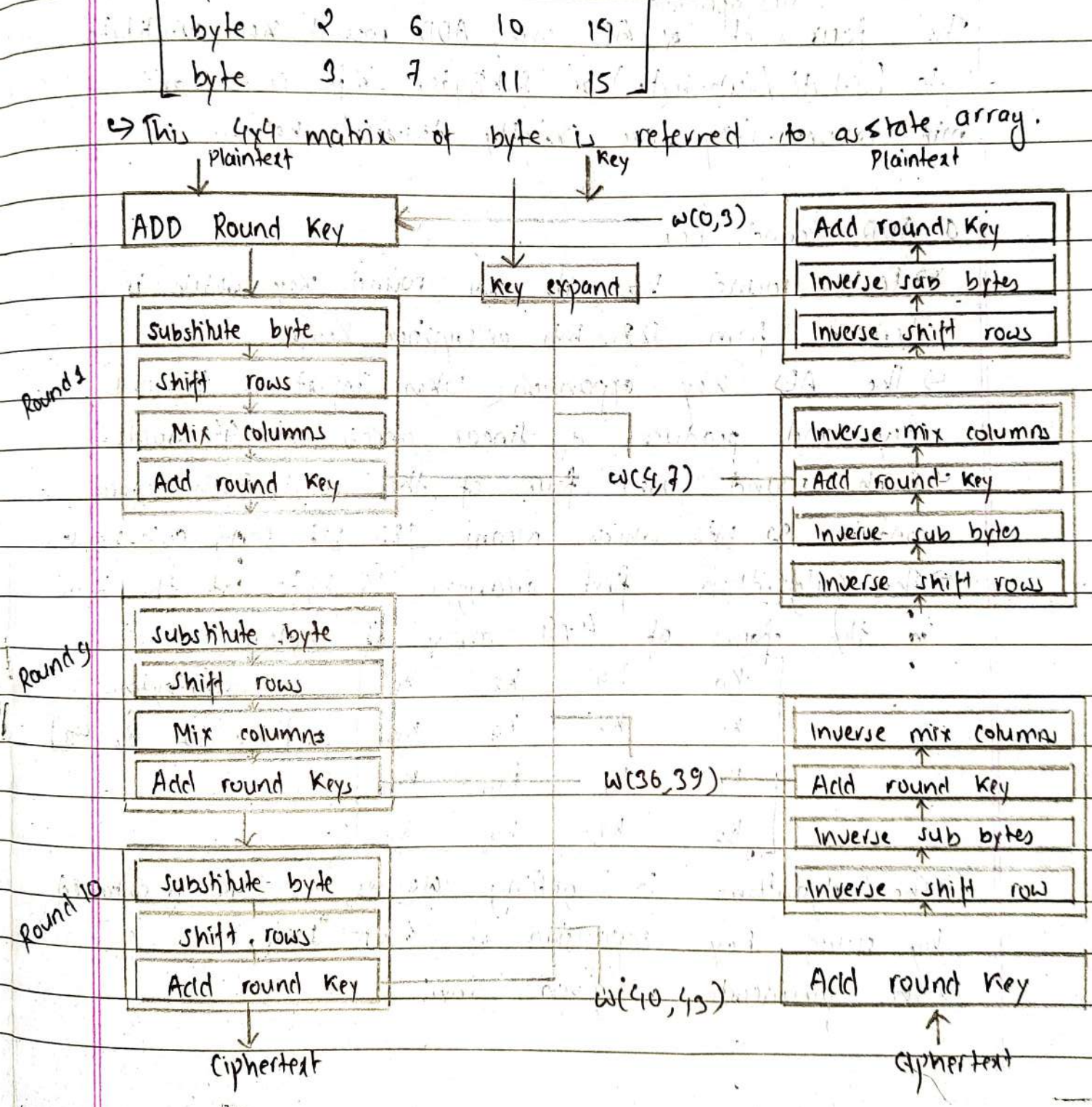
→ Each round processing includes one single based substitution a row-wise permutation step, a columnwise mixing step and addition of the round key. The order in which these four steps are executed is different for encryption and decryption.

→ Security strength of AES is equal or better than 3DES and significantly improved efficiency.

→ The 128 bit block represented as a 4x4 matrix of bytes as shown below:

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

→ This 4x4 matrix of byte is referred to as state array.



The encryption phase of AES can be divided into three phases.

1. Initial round
2. Main round.
3. Final round

The four sub-operation of AES are ADD round key, substitute byte (s-byte / sub-byte) or subBytes, shift rows and mix columns. They are briefly discussed below.

a. ADD Round Key

↳ Each round has its own round key which is derived from 128 bits encryption key.

↳ The AES key expansion takes input of 4 word key and produces a linear array of 44 words.

↳ Each round uses four of the words. Each word contain 32 bits which means 128 bit long sub-key.

→ The algorithm first arranges 16 bytes of the key in the form of 4×4 array of bytes.

$$\begin{bmatrix} K_0 & K_4 & K_8 & K_{12} \\ K_1 & K_5 & K_9 & K_{13} \\ K_2 & K_6 & K_{10} & K_{14} \\ K_3 & K_7 & K_{11} & K_{15} \end{bmatrix} \text{ sub-bytes.} = [w_0 \; w_1 \; w_2 \; w_3]$$

↳ The algorithm for getting $w_0, w_1, \ldots w_{44}$ is obtained by using key expansion of 4×1 2×4.

↳ The pseudocode is given below:

```
key_expansion (byte key[16], word w[44])
{
    word temp;
    for (i=0; i<4; i++)
    {
        w[i] = k[4*i], k[(4*i)+1], k[(4*i+2), k[(4*i),3]);
    }
for (i=4; i<43; i++)
    {
        temp = w[i-1];
        if (imod4 ==0)
        {
            temp = subword (Rotword (temp)) ⊕ R
            w[i] = w[i-4] ⊕ temp;
        }
    }
}
```

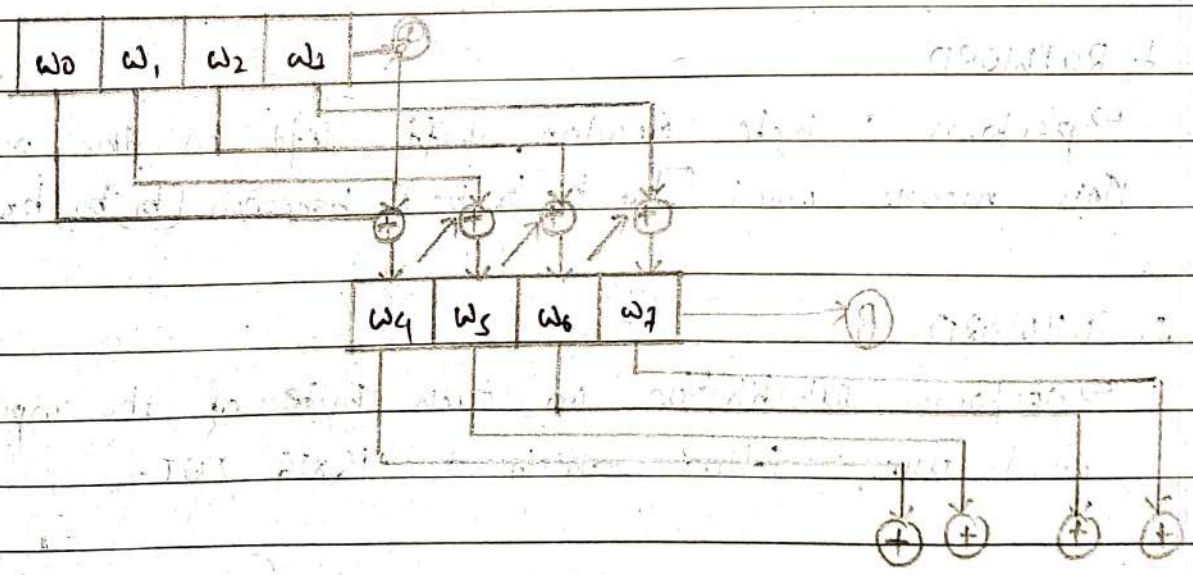


Fig: Key Expansion of DES

Let us consider, we have four words of round of ith round w_i, w_{i+1}, w_{i+2}, w_{i+3}

When (i mod 4 ≠ 0)

Then,

w[i] is found by immediately proceeding word w[i-1] XORed with four positive back word w[i-4]. i.e.

$$w[i] = w[i-4] \oplus w[i-1]$$

$$w[i+5] = w[i+1] \oplus w[i+4] \quad \text{---(1)}$$

$$w[i+6] = w[i+2] \oplus w[i+5] \quad \text{---(2)}$$

$$w[i+7] = w[i+3] \oplus w[i+6] \quad \text{---(3)}$$

For a word, those position in w array is multiple of 4, a more complex function is used.

→ The beginning of each round key is obtained by

$$w_{i+4} = w_i \oplus g(w_{i+3}) \quad \text{---(4)}$$

where g consists of following substitution:

1. ROTWORD

→ perform 1 byte circular shift left on the word. This means word $[b_0, b_1, b_2, b_3]$ becomes $[b_1, b_2, b_3, b_0]$

2. SUBWORD

→ perform substitution on each byte of its input word using S-box which is 16×16 LUT.

3. The result of step 2 is XORed with Round constant

GF = Galois Field.

Rcon[j].

The round constant is a word in which the right most 3 bytes is always zero. Thus, it has only effect on the left byte of the word.

$$Rcon(j) = [RC[j], 0x00, 0x00, 0x00]$$

with

$$RC[j] = 1; \quad RC[j] = 2 \cdot RC[j-1]$$

and the multiplication is defined over the field of $GF(2^8)$

↳ The addition of round constant is for the purpose of destroying any symmetry that may have been introduced in other steps in key expansion algorithm.

b. Sub Bytes.

↳ This round uses byte-by-byte substitution during the encryption forward process. The corresponding substitution for decryption process is called inverse sub-bytes.

↳ this step consists of a 16 to 16×16 lookup table (LUT) to find a replacement for a byte in the input state array. The entries in this LUT are created using the notion of multiplicative inverse of $GF(2^8)$.

↳ For a particular round, each bite is mapped to a new byte in the following way:

1. The leftmost nibble from the left half determines the row and the rightmost nibble determines the column.

The s-box transformation of 35 or Hex 23 (0x23) can be found in the cell at the intersection of the row labeled 20 and the column labeled 03. Therefore, the decimal 35 becomes Hex 26 (0x26) or decimal 38.

c. Shift Rows

This step is called shift rows for shifting the rows of state array during the forward process. The operation is called inverse shift rows during the transformation in decryption state. In this operation, the first row is unaltered. Second and third row and fourth (The remaining rows) are shifted to the right left in increasing order.

a_{00}	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

→

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

d. Mix Columns

The mix column operation is basically a substitution that uses GF(2^8) in which each column is operated individually. Each byte of the column is mapped into a new column by using a function of all four bytes in the column. The transformation is usually shown in the figure below:

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

X

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

=

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

Advantages of AES

↳ AES is ubiquity. Standard used by US government.

↳ Relatively fast in both software and hardware.

↳ The execution time of both encryption and decryption is increased.

↳ High speed and more security.

↳ Uses a single s-box for all bytes in all rounds.

Disadvantages

↳ Simple key schedule and simple encryption operation. Many AES are attacked based upon the simplicity of key schedule.

↳ so, attack is possible one day.

Characteristics	DES	IDEA	AES
1. Basic	Data block divided into 2 halves	same as AES	Entire data block is processed as a single unit matrix.
2. Principle	Feistel cipher	Works on substitution & Permutation	
3. Plaintext	64 bits	64 bits	128 bits
4. Key size	56 bits	128 bits	128, 192, 256 bits
5. Round	16 rounds	8 rounds	10 - 128 bits 12 - 192 bits 14 - 256 bits
6. Speed	slower	Faster than DES slower than AES	Faster
7. Security	less secure	More secure	More secure than IDEA
8. Round Name	Expansion, Permutation, X-OR, P-Box, S-Box, Swap	GF (2^{16}) GF $(2^{16}+1)$ X-OR	Sub Bytes Shift Rows Mix Columns Add Round keys