Secret Key Cryptography

General Block Encryption:

The general way of encrypting a **64-bit block** is to take each of the:
2^{64} input values and *map it to a unique* one of the 2^{64} output values.
This would take $(2^{64})*(64) = 2^{70}$ bits to store this map. NOT practical.

Secret key cryptographic systems take a reasonable length **key** (e.g., 64 bits) and **generate a one-one mapping** that looks, to someone who does not know the key, **completely random**.
I.e., any single bit change in the input result in a totally independent random number output.

Types of transformation for k-bit blocks:

- **Substitution:**

  For small values of ***k***, specify for each of the 2^k possible values of the input, the *k*-bit output.
  This takes $k*2^k$ bits. E.g., for ***k***=8 we need 2048 bits.

- **Permutation:**

  Specify for each of the ***I*** input bits, the output position to which it goes.
  This takes $I*\log_2 I$ bits. E.g., for I=64, we need 64*5=320 bits

The following figure (Fig. 3-1) shows a secret key algorithm based on **rounds** of substations and permutation. If we do only a single round, then a bit of input can only affect 8 bits of output. There is ***optimal*** number of rounds to achieve complete randomization.
The algorithm take the ***same effort*** to reverse (decrypt).

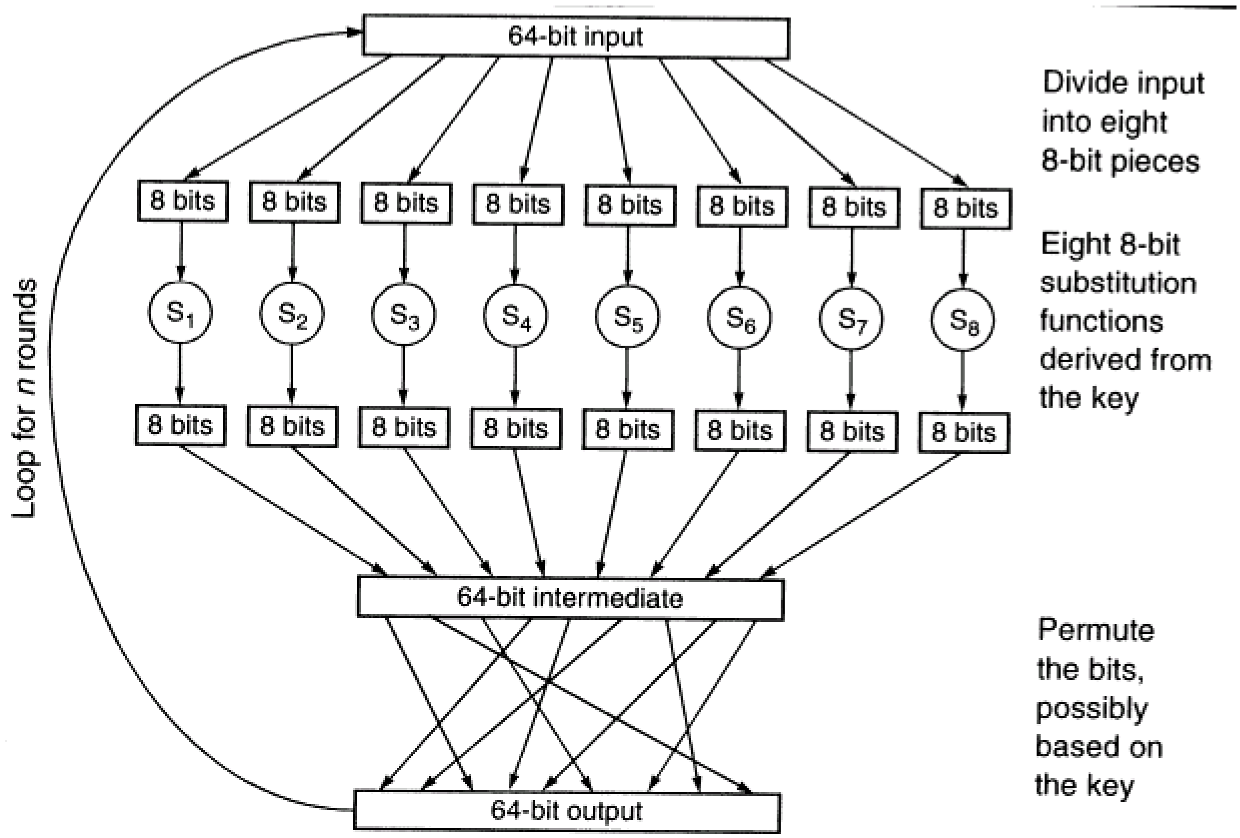


Figure 3-1. Example of Block Encryption

Data Encryption Standard (DES):

Key length: 64 bits

> 8 bits are used for parity check,
> why is that? to make it 265 times less secure!
> read **why 56 bits?** section in the textbook.

How secure is DES?

> In 1998, $150K machine can break the key in 5 days!
> For added security **triple DES** is 2^{56} more secure.

Basic Structure of DES: (Fig. 3-2)

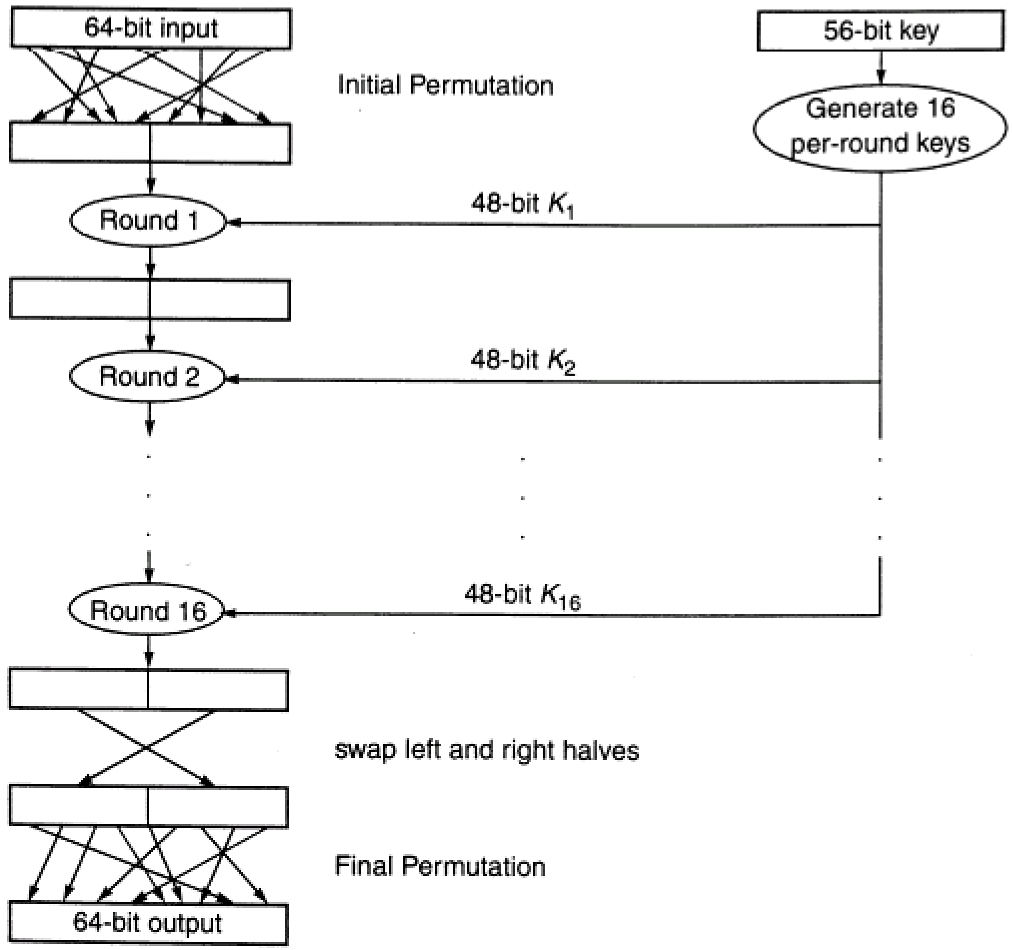


Figure 3-2. Basic Structure of DES

The decryption works by essentially running DES **backward** (with keys: K16 .. K1).

The Permutation of Data (Fig. 3-3)

This is not random, see Fig. 3-3 to get IP, and reverse the arrows to get IP^{-1}

In the IP table, bit 1 comes from bit 58, bit 2 comes from bit 50, etc.

The first octet of the input (ABC....H) is distributed over the 8 octets of the output

(A to 5th octet, B to 1st Octet, ... H to 4th octet).

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

Final Permutation (IP

40	8	48	16	56	24
39	7	47	15	55	23
38	6	46	14	54	22
37	5	45	13	53	21
36	4	44	12	52	20
35	3	43	11	51	19
34	2	42	10	50	18
33	1	41	9	49	17

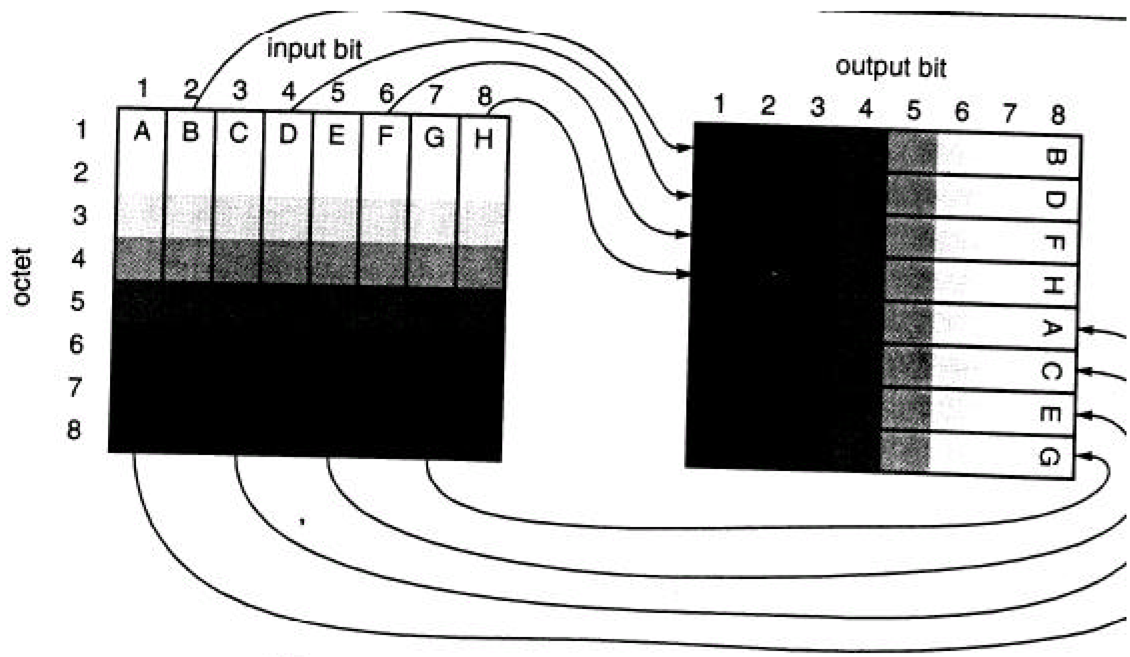


Figure 3-3. Initial Permutation of Data Block

In this Figure:

Bit 58 at position[8,2] --> bit 1 at position [1,1].
Bit 1 at position [1,1] --> bit 40 at position [5,8].

Generating the Per-Round Keys:

- *Key-Permutation: (Fig. 3-4) Produces C_0 and D_0*

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

⇒

57	49	41	33	25
1	58	50	42	34
10	2	59	51	43
19	11	3	60	52
63	55	47	39	31
7	62	54	46	38
14	6	61	53	45
21	13	5	28	20

Figure 3-4. Initial Permutation of Key

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

C0 **D0**

- *Key-Generation: (Fig. 3-5)*

8 bits are discarded: 9, 18, 22, 25 from C_i and 35, 38,43, 54 from D_i
so that each K_i is 48 bits.

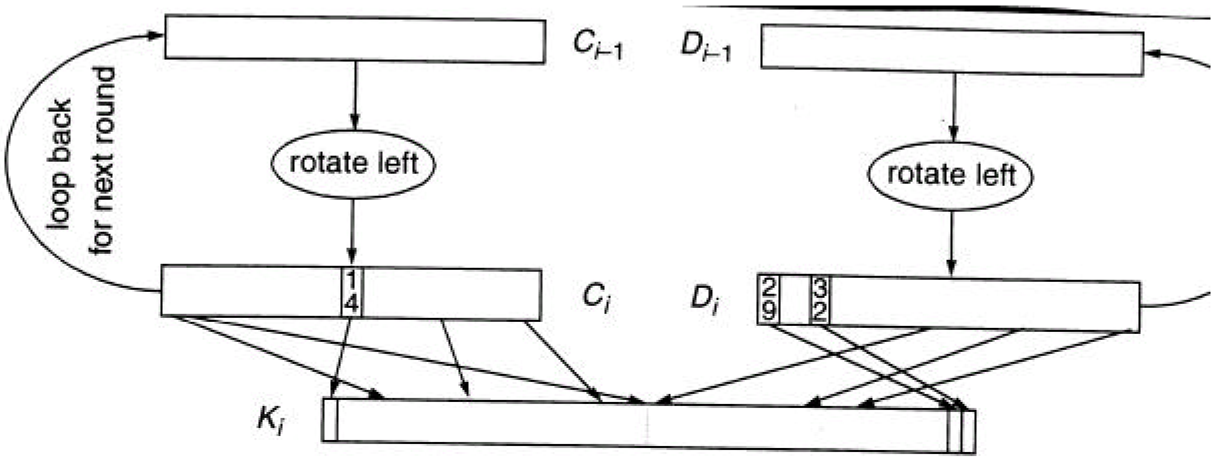


Figure 3-5. Round i for generating K_i

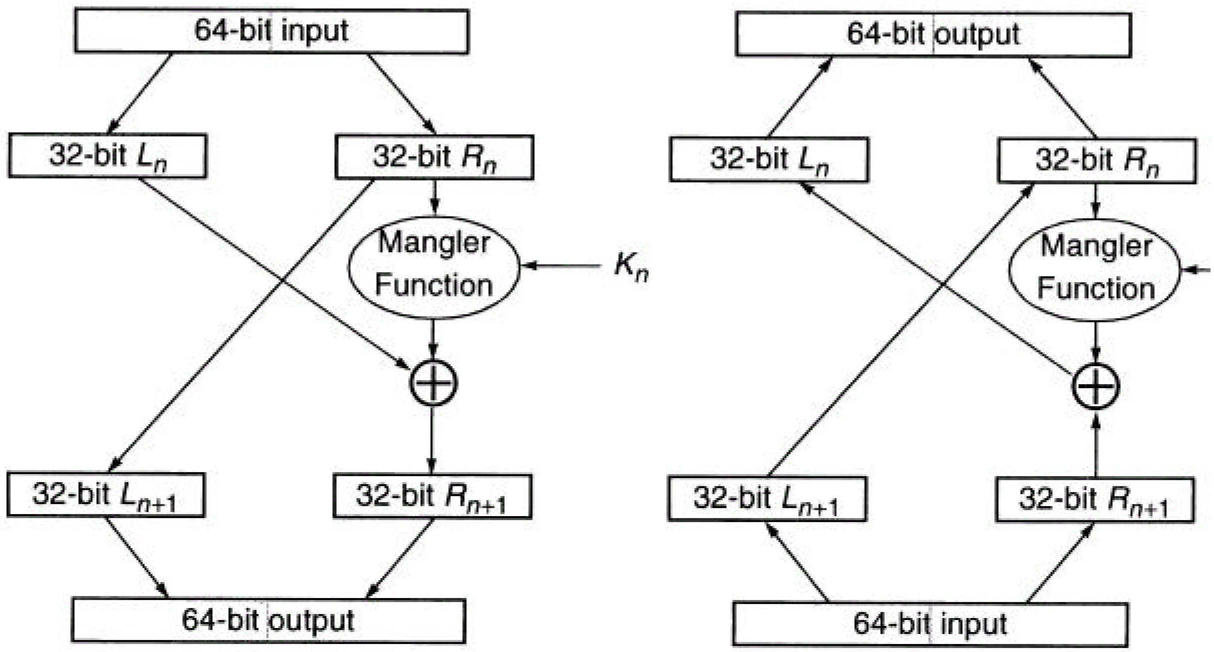
permutation to obtain the left half of K_i:

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2

permutation to obtain the right half of K_i:

41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

A DES Round: *(Fig. 3-6)*



Encryption Decryption

Figure 3-6. DES Round

Why decryption works?

- The output of the Mangler Function (M) is the same for both encryption and decryption.
- In encryption: M ® $L_n = R_{n+1}$
- In decryption: M ® $R_{n+1} = M$ ® $(M$ ® $L_n) = L_n$

The Mangler Function:

- Expands R from 32 bit to 48 bits as shown in Fig3-7:

  It breaks R into eight 4-bit chunks and expand each to 6-bit by concatenating the adjacent 2 bits. Let CR_i refer to chunk i of expanded R.

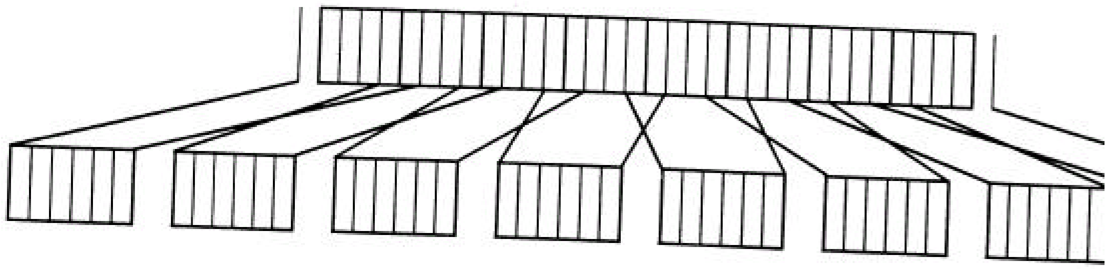


Figure 3-7. Expansion of *R* to 48 bits

- The 48-bit K is broken to eight 6-bit chunks. Let CK_i refer to chunk i of K.
- Let $S_i = CR_i ® CK_i$
- S_i is fed into an S-box, a substitution which produces a 4-bit output for each possible 6-bit input as shown in Figure 3-8 (i.e., 4 input *mapped to* 1 output).

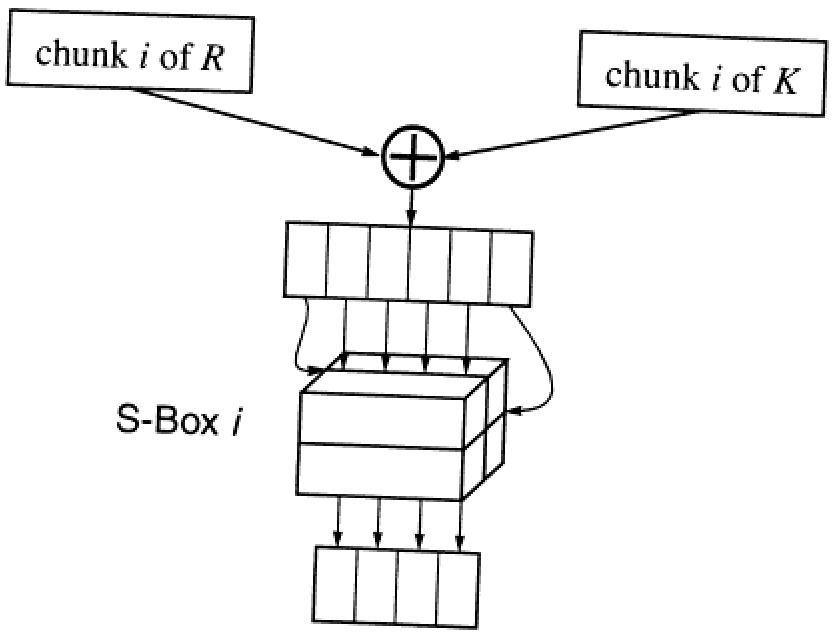


Figure 3-8. Chunk Transformation

- The 8 S-boxes specified in [Fig. 3-9 to 3-16](#):

Input bits 1 and 6 ↓ / Input bits 2 thru 5	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110

Figure 3-9. Table of 4-bit outputs of S-box 1 (bits 1 thru 4)

Input bits 7 and 12 ↓ / Input bits 8 thru 11	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110
00	1111	0001	1000	1110	0110	1011	0011	0100	1001	0111	0010	1101	1100	0000	0101
01	0011	1101	0100	0111	1111	0010	1000	1110	1100	0000	0001	1010	0110	1001	1011
10	0000	1110	0111	1011	1010	0100	1101	0001	0101	1000	1100	0110	1001	0011	0010
11	1101	1000	1010	0001	0011	1111	0100	0010	1011	0110	0111	1100	0000	0101	1110

Figure 3-10. Table of 4-bit outputs of S-box 2 (bits 5 thru 8)

- The 4-bit output of each of the eight S-boxes is permuted as shown in Fig. 3-17

  (to ensure that the output of an S-box in one round affects the input of multiple S-boxes on the next round):

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

Figure 3-17. Permutation of the 32 bits from the S-boxes

What's So Special about DES?

The S-boxes!
Are they random?. no one knows.
Playing around with the S-boxes can be dangerous!

International Data Encryption Algorithm (IDEA):

Encrypts 64-bit blocks using 128-bit key.
It is similar to DES since it:

- operates in **rounds**,
- the **mangler function** runs in the *same direction* for both encryption and decryption.

Fig. 3-18 shows the basic Structure of IDEA:

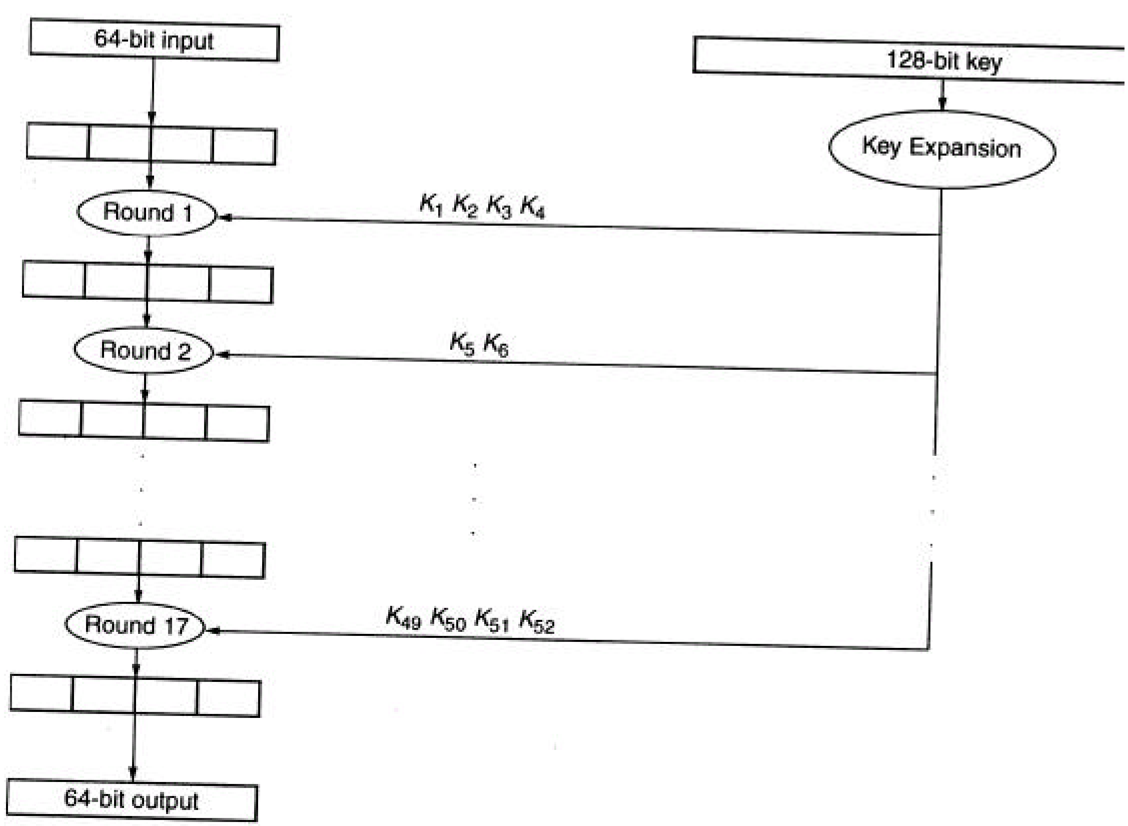


Figure 3-18. Basic Structure of IDEA

IDEA operations:

® exclusive OR
+ addition mod 2^{16} and
x multiplication mod 2^{16}

These operations are *reversible*:

a ® K = A » A ® K = a since (a ® K) ® K = a
a + K = A » A + (-K) = a since (a + K) + (-K) = a
a x K = A » A x (K^{-1}) = a since (a x K) x (K^{-1}) = a

Key Expansion:

The 128-bit key is expanded into 52 16-bit keys: K1, K2 , ...K52.
After generating the first 8 keys (Fig. 3-19),
shift 25 bits and continue the generation (Fig. 3-20).

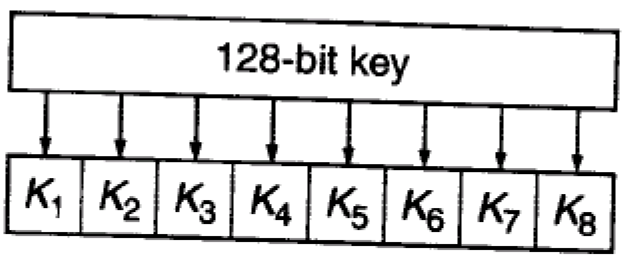


Figure 3-19. Generation of keys 1 through 8

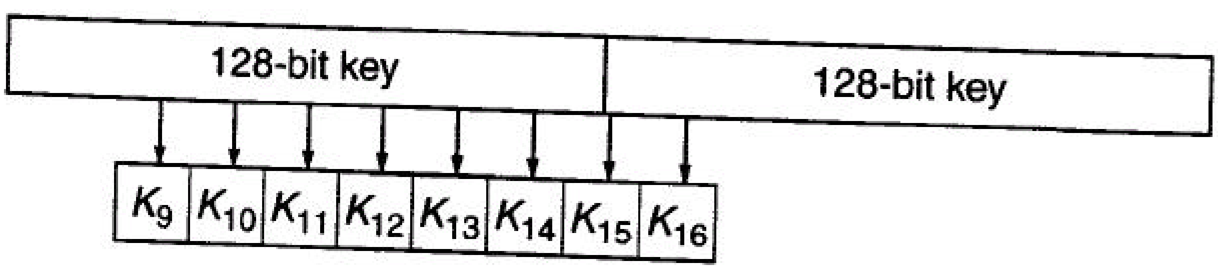


Figure 3-20

Rounds:

Total of 17 rounds, **odd**: 1, 3, ...17 & **even** 2, 4, .., 16

- **Odd Round:** (**Fig. 3-21**)

  This is reversible using the inverse keys.

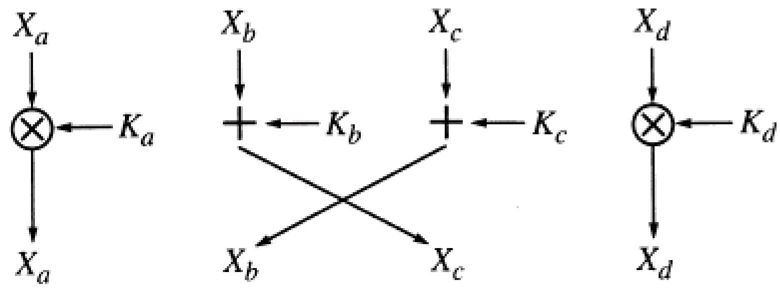


Figure 3-21. IDEA Odd Round

- **Even Round:** (*Fig. 3-22*)

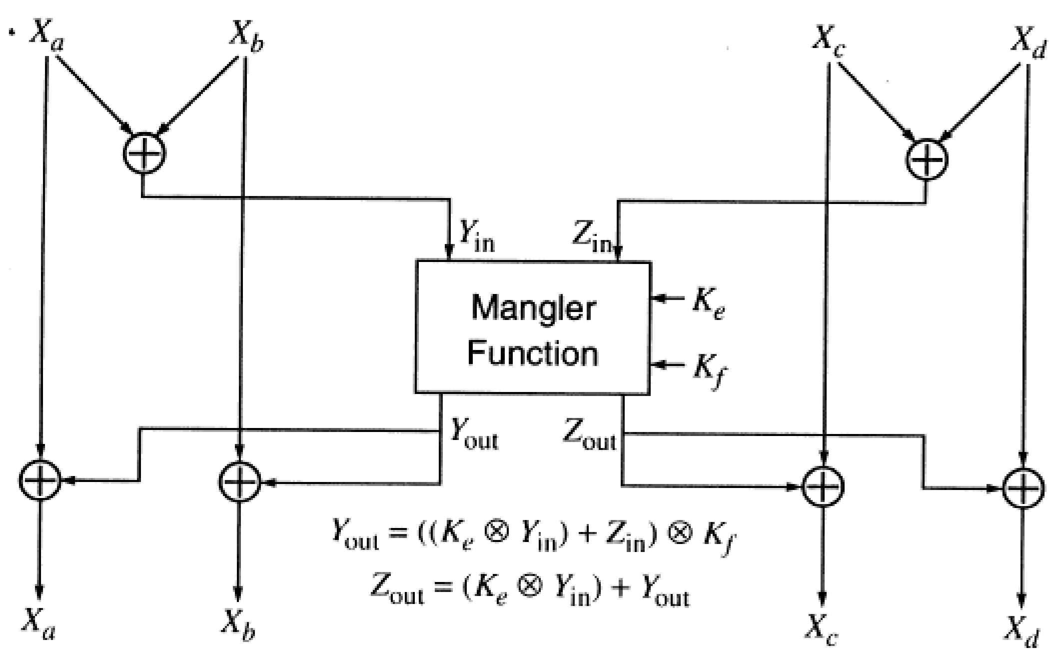


Figure 3-22. IDEA Even Round

How it is reversed?

Just apply it again, using the same keys (not the inverse as in odd rounds!).

Why?

From Figure 3-22 we have:

X'a = Xa ® Yout
X'b = Xb ® Yout
Yin = Xa ® Xb

Thus:
X'a ® X'b = (Xa ® Yout) ® (Xb ® Yout)
= Xa ® Xb
= Yin
I.e, Yin is the same if we use (Xa , Xb) or (X'a , X'b)
Similarly, Zin the the same if we use (Xc , Xd) or (X'c , X'd)
Thus Yout and Zout are the same in both encryption and decryption.

Therefore, since we know Yout and Zout we can get:

Xa = X'a ® Yout
Xb = X'b ® Yout
Xc = X'c ® Zout
Xd = X'd ® Zout

Inverse Keys for Decryption::

Encryption keys:
K1 K2 K3 K4 K5 K6 K7 K8 ......

Decryption Keys:
$(K49)^{-1}$ -(K50) -(K51) $(K52)^{-1}$ K47 K48 $(K43)^{-1}$ -(K44) ....

Advanced Encryption Standard (AES):

Developed with the help of **NIST** as an efficient, flexible, secure and unencumbered (free to implement) standard for protecting sensitive non classified, U.S. government information.

NIST selected an algorithm called **Rijndael** (named after two Belgium cryptographers).
It uses a variety of block and key sizes (mainly 128, 192 and 256)
and the standards are named: ***AES-128, AES-192, AES-256***!
(block sizes are fixed in all to 128 bits).
It is similar to DES and IDEA in that there is ***rounds*** and ***key expansion***.

RC4
A long random string is called a **one-time pad.**
A **stream cipher** generates a one-time and applis it to a stream of plain text with ®.
RC4 is a stream cipher designed by Ron Rivest.
Page 93 gives a C code for RC4 one-time pad generator.

Modes of Operation

Encrypting a Large Massage

Electronic Code Book (ECB):

Break the message into 64-bit blocks (padding the last one) and
encrypt each block with the secret key.

***Two problems*:**

1. two identical plain text block produce two identical cipher blocks
2. blocks can be rearranged or modified.

Example: See Fig. 4-3 where an eavesdropper:

1. can see which sets of employees have identical or similar salaries and
2. he can alter his own salary to match another employee with higher salary.

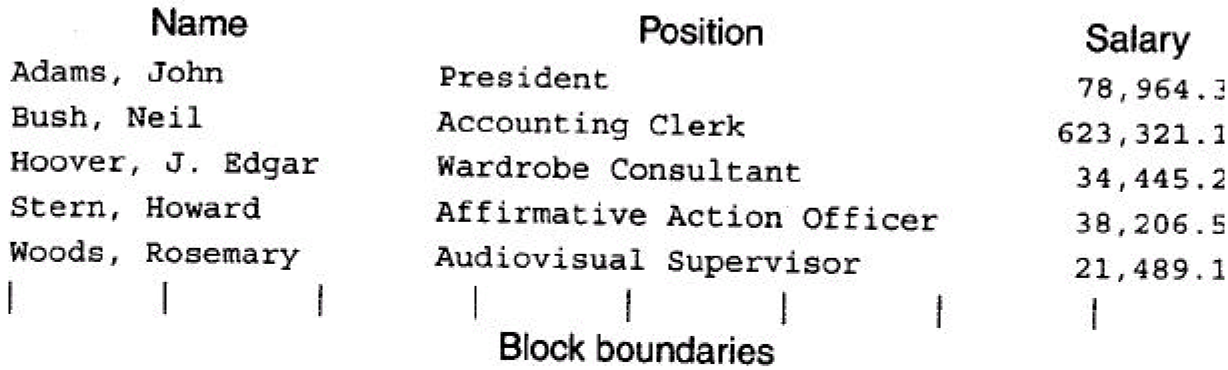


Figure 4-3. Payroll Data

Cipher Block Chaining (CBC):

See Figure Fig. 4-5 & Fig 4-6: The randomly chosen IV (Initialization Vector)
Two identical plain messages produces two different cipher messages.
(e.g., continue holding, continue holding, ..., start attach)
This prevents Chosen plain text attach

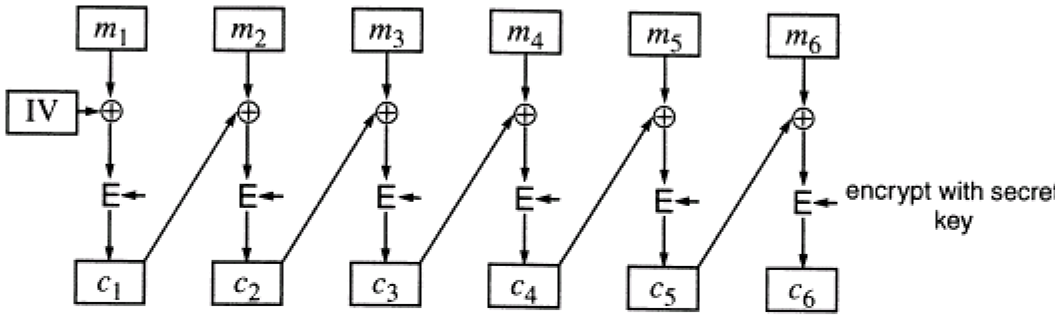


Figure 4-5. Cipher Block Chaining Encryption

Decryption is simple because ⊕ is its own inverse.

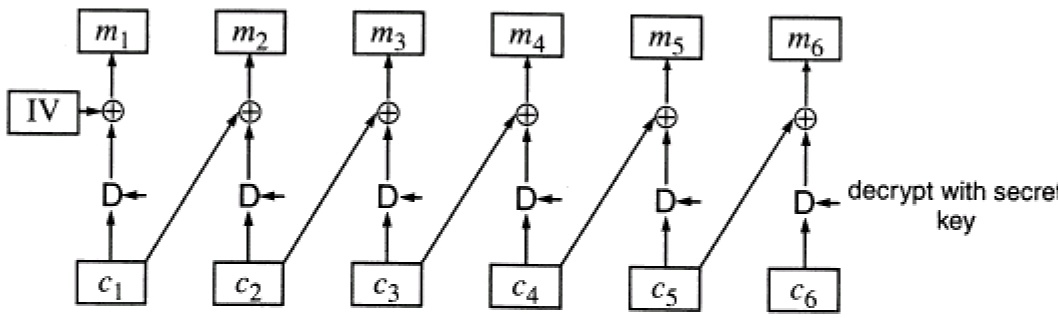
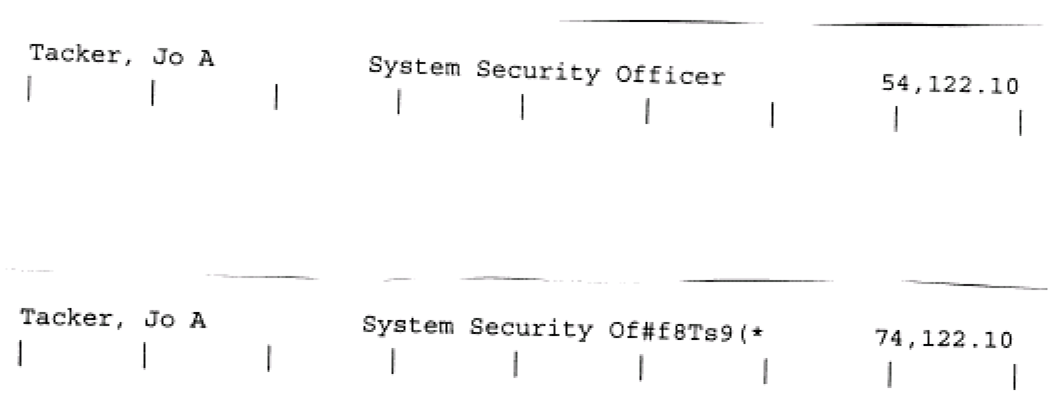


Figure 4-6. Cipher Block Chaining Decryption

CBC Threat- Modifying Cipher Blocks
You can modify the contents of one cipher block to make the plain text of next block as you wish, however the preceding plain text block will be garbled, as shown:



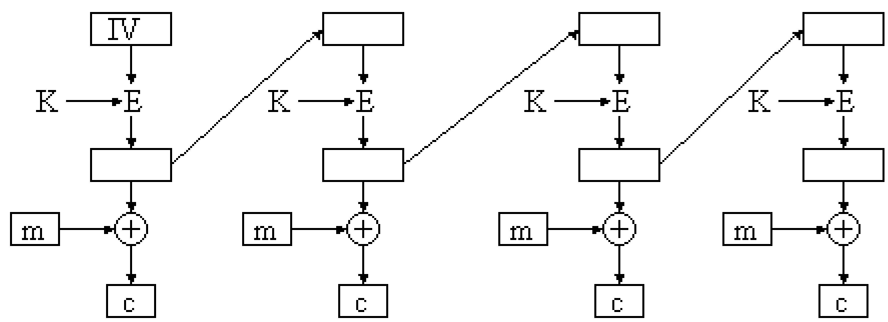
Thus if c_n is garbled then m_n will be completely garbed.
Only the same portion of m_{n+1} as what was changed in c_n will be changed.

This can be solved by attaching a CRC to the plain text before encryption.

Output Feedback Mode (OFB):

It is a stream cipher, encryption/decryption is performed by ®ing the message with one-time pad generated as follows:

1. A 64-bit random IV is generated (and is transmitted with the encrypted message).
2. b_1 is the DES encryption of IV with the secret key.
3. b_i , i > 1, is the DES encryption of b_{i-1} with secret key.
4. The resulting one-time pad is: *b1* | *b2* | *b3* | ....
5. $c_i = b_i$ ® m_i for i =1, 2, ...



Major advantages of OFB:

- the pad can be *generated in advance* and used when the message arrive.
- if some bits of cipher text get garbled,

only the corresponding bits in the plain text get garbled.

Major disadvantages of OFB:

- If the <plaintext m, ciphertext c=m®E) > are known by Trudy,

he can modify the plain text m into anything he wants (m') since he can make:

$c' = m' © E$

and thus

$c' © E = (m' © E) © E = m'$

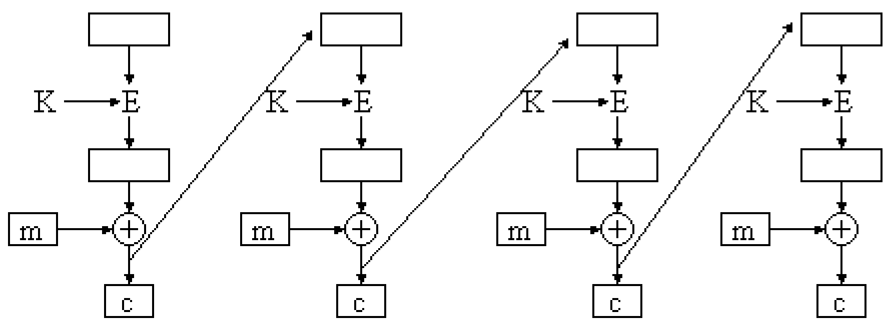
- If one block is lost, the rest of the blocks will be garbled.
- If data is stored on disk, you can not randomly read any block

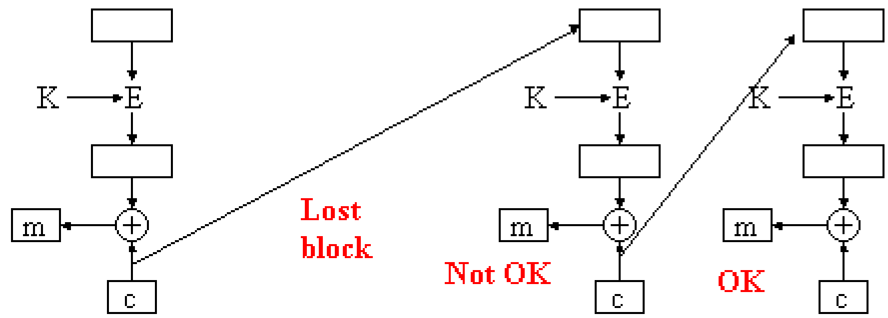
  unless you decrypt all the preceding blocks.

  To solve the last two problems, we use CFB below, where if one block is lost,
  only the next block is garbled and the rest of the blocks will decrypt properly.

Cipher Feedback Mode (CFB):

1. A 64-bit random IV is generated (and is transmitted with the encrypted message).
2. b_1 is the DES encryption of IV with the secret key.
3. b_i , $i > 1$, is the DES encryption of ***c_{i-1}*** with secret key.
   (Thus you can't generate a one-time pad in advance like OFB)
4. $c_i = b_i ® m_i$ for i =1, 2, ...





Counter Mode (CTR):

See Fig. 4-10 , CTR have the following advantages:

- You can generate the one-time pad in advance.
- You can randomly access any block without decrypting all the preceding blocks.

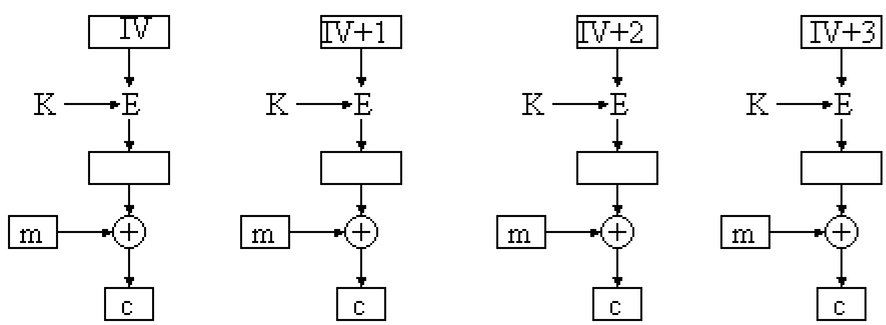


Figure 4-10

Generating MACs

A secret key system can be used to generate a cryptographic checksum MAC (message authentication code) or MIC (message integrity code).

Send Plain text + CBC residue: (see Fig. 4-11)
The receiver computes the CBC residue from the plain text and compare it with the received CBC residue.

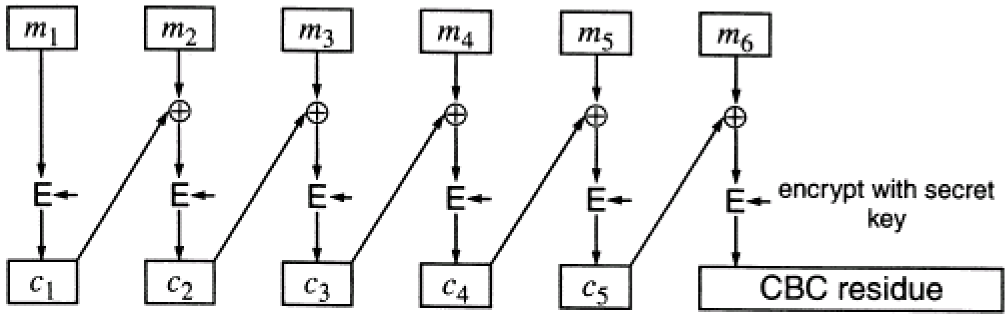


Figure 4-11. Cipher Block Chaining Residue

Multiple Encryption DES

It is called 3DES or EDE (encrypt-decrypt-encrypt):

```
m>>>> E >>>> D >>>> E >>>>c
      |      |      |
      K1     K2     K1
      |      |      |
c >>>> E >>>> D >>>> E >>>> m
```

CBC is used for stream encryption as shown is Fig. 4-15:

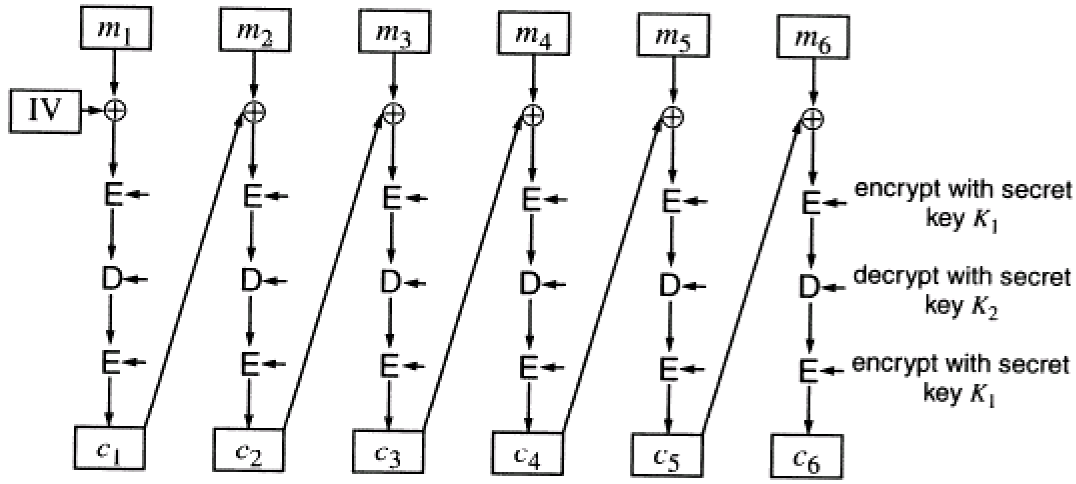


Figure 4-15. EDE with CBC on the Outside (3DES)