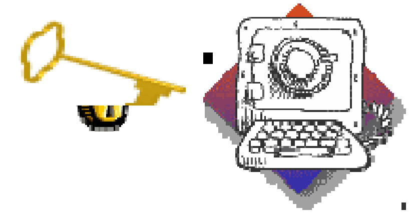


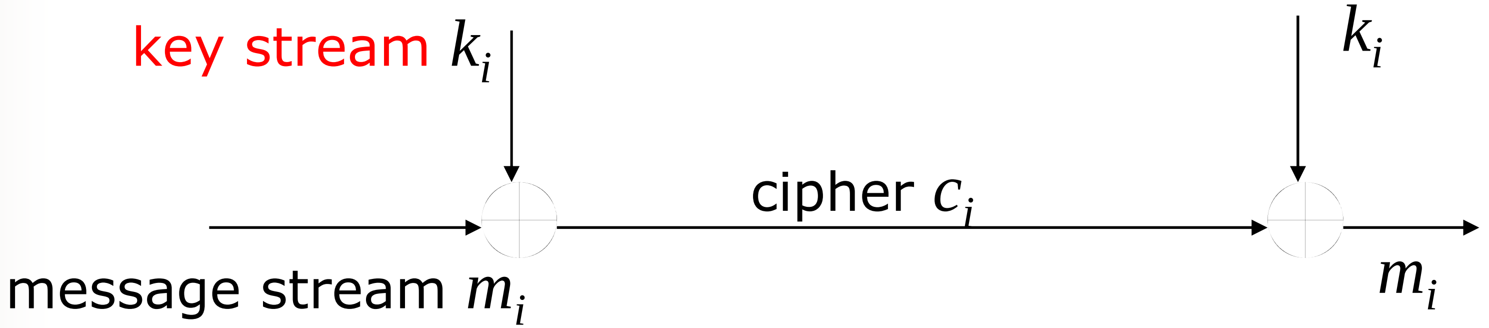
Chapter 2
Secret Key Cryptography

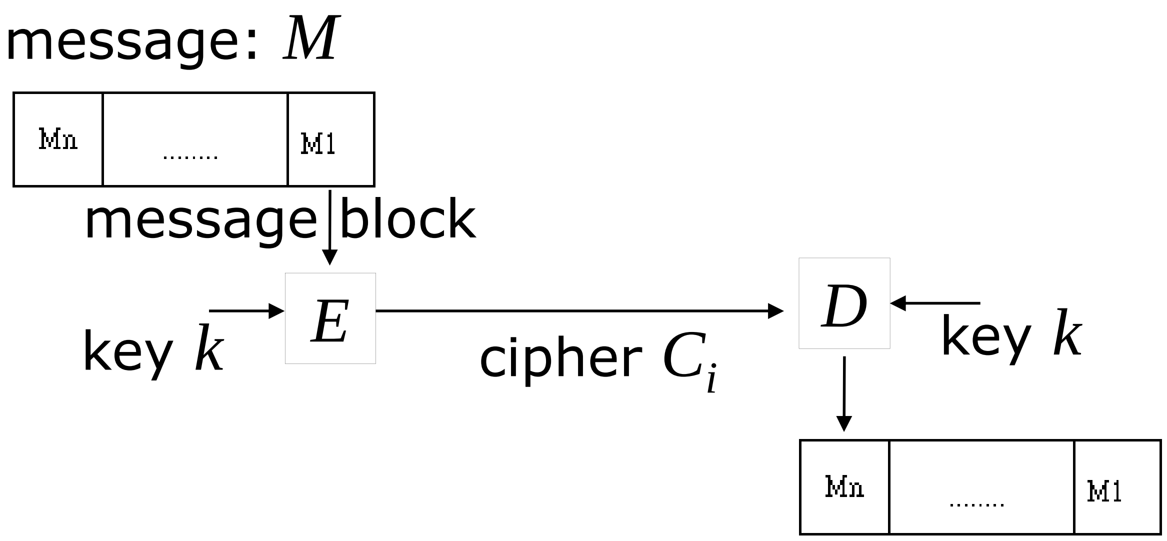
顏嵩銘 (Sung-Ming Yen)

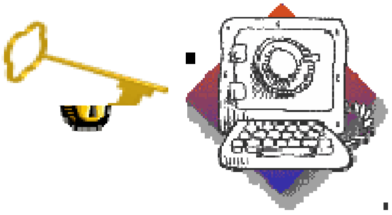
中央大學 資訊工程系所
密碼與資訊安全實驗室
Laboratory of Cryptography and Information Security LCIS
http://www.csie.ncu.edu.tw/~yensm/lcis.html

Tel： (03) 4227151 Ext- 35316
Fax： (03) 4222681
E-Mail：yensm@csie.ncu.edu.tw

Stream Cipher and Block Cipher







Stream Cipher

how to generate key stream
(pseudo random number)

Methods to generate key stream (Pseudo Random Number)

- Linear congruence method

$$x_i \equiv a x_{i-1} + b \bmod m$$

where (a, b, m, x_0) is the seed (secret)

Ex: Let a=5, b=3, m=16, x_0=1

We obtain

$$\{x_0, x_1, x_2, \ldots.. x_{15}, x_{16}\} =$$

$$\{1, 8, 11, 10, 5, 12, 15, 14, 9, 0, 3, 2, 13, 4, 7, 6, 1\}$$

- For some selection of (a, b, m), only odd or even integers can be generated.

- Linear congruence method is very weak!

  Given x_0, x_1, and x_2:

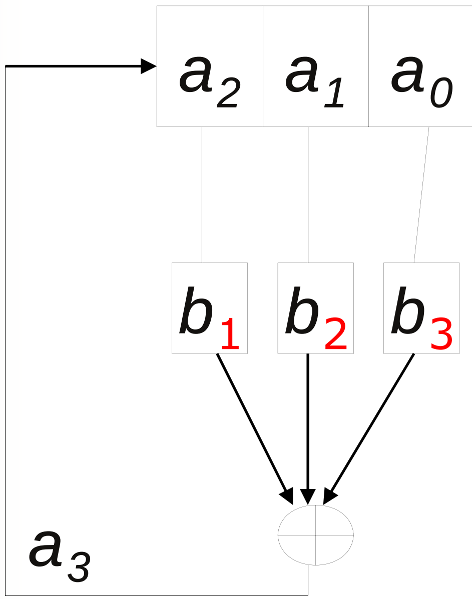
  $x_1 = a*x_0+b$ ...... (1)

  $x_2 = a*x_1+b$ ...... (2)

  (2)−(1) leads to $a = (x_2-x_1)/(x_1-x_0)$

  then, $b = x_1-a*x_0$

Linear feedback shift register (LFSR)



$$\{a_n, a_{n-1}, \ldots\ldots a_3, a_2, a_1, a_0\}$$

$$a_3 = a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

AND operation

$b_i = 1$: switch on

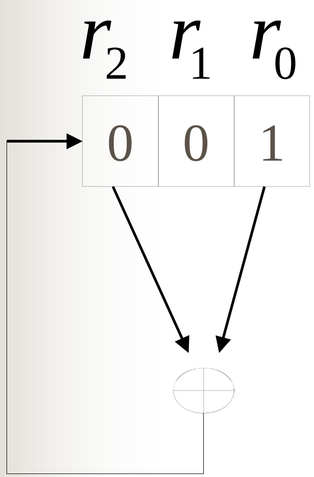
$b_i = 0$: switch off

where $\{a_2, a_1, a_0, b_1, b_2, b_3\}$

are the seed (secret key)

Ex: Let $\{b_1, b_2, b_3\} = \{1, 0, 1\}$ and

$\{a_2, a_1, a_0\} = \{0, 0, 1\}$



r_2	r_1	r_0
0	0	1
1	0	0
1	1	0
1	1	1
0	1	1
1	0	1
0	1	0

The period=7=2^3-1

* If $\{b_i\}$ are well selected, the max period of $\{a_i\}$ can be 2^n-1 where n is the number of stage of registers.

no (0, 0, 0) as state

- The max period of LFSR

Ex: Given $\{b_1, b_2, b_3\}$ and let $b(x) = b_3x^3 + b_2x^2 + b_1x + 1$ be the connection polynomial. If $b(x)$ is a primitive polynomial over $\mathbf{Z}_2$, then the LFSR can generate an **m-sequence**.

- Primitive polynomial

A primitive poly. over $\mathbf{Z}_2$ of degree n is an *irreducible* poly. that divides $x^{2^n-1} - 1$ but not $x^d - 1$ for any d that divides 2^n-1.

* The case in "*integers*"

- cryptanalysis (predictability)

For the same example, give$\{a_0,a_1,a_2,a_3,a_4,a_5\}$.

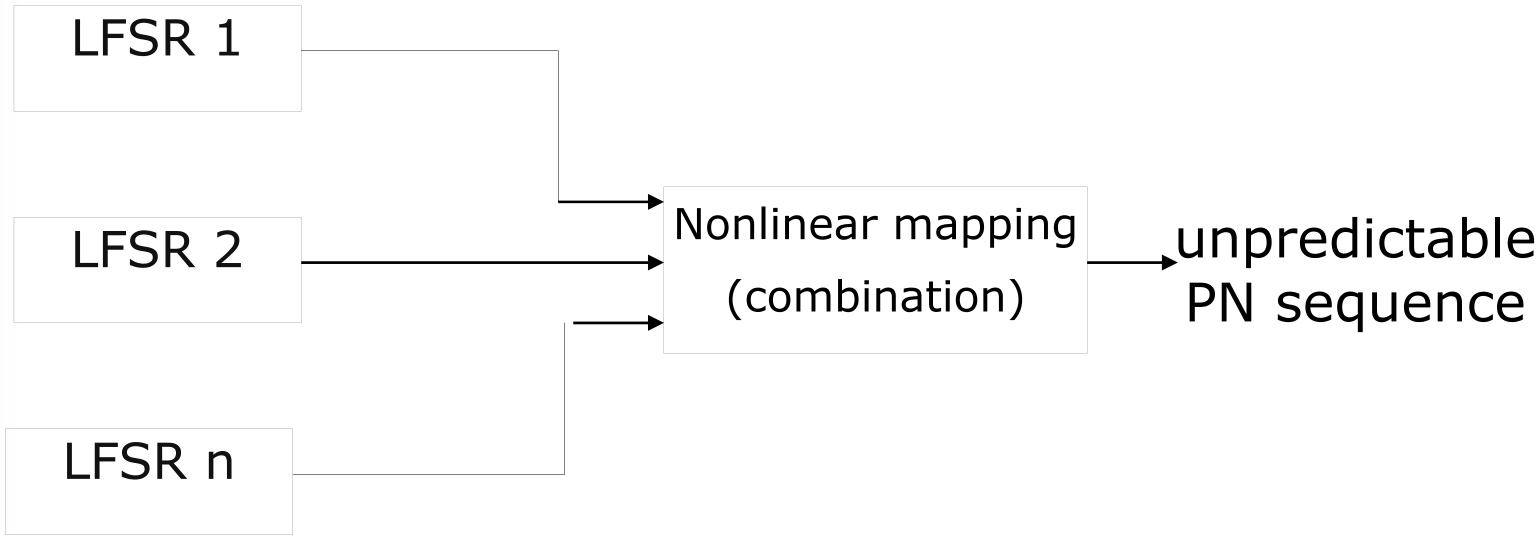
$$a_0=1$$
$$a_1=0$$
$$a_2=0$$
$$\begin{cases} a_3=1=a_2b_1+a_1b_2+a_0b_3 \mod 2 \\ a_4=1=a_3b_1+a_2b_2+a_1b_3 \mod 2 \\ a_5=1=a_4b_1+a_3b_2+a_2b_3 \mod 2 \end{cases}$$

$^*(a+b+c) \bmod 2 \Leftrightarrow a \oplus b \oplus c$

unknown (b_1, b_2, b_3)

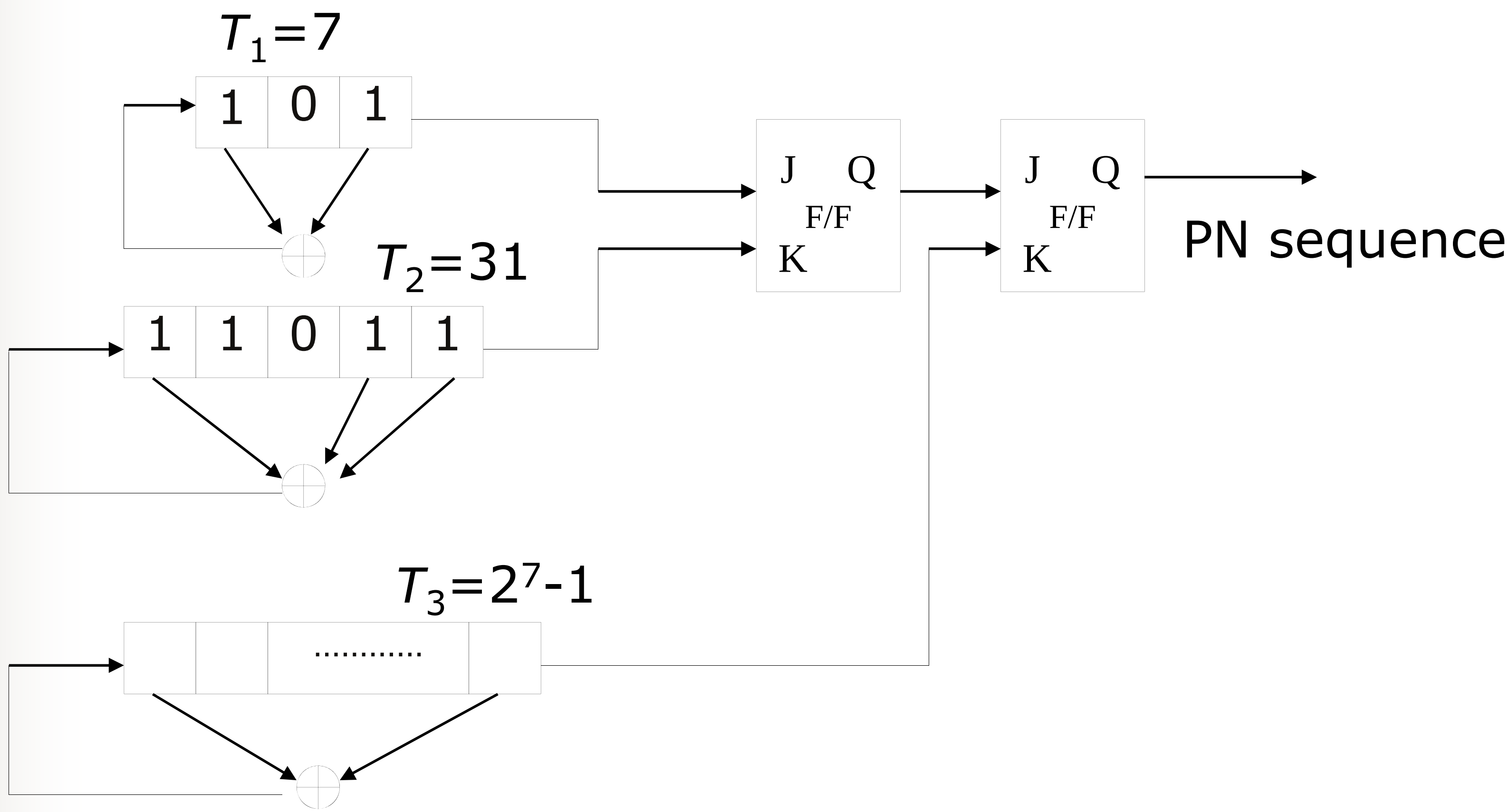
LFSR若長度為n, 則已知$2n$個連續output就可以得知後續所有(2^n-1-2n)個output

- Countermeasure against the predictability attack



Let m_i be the # of stage of LFSR *i* and all m_i's are pairwise relatively prime. The period *Z* of the combined PN generator is

$$Z = \prod_{i=1}^{n} T_i \quad \text{where } T_i = 2^{m_i} - 1$$


T_1=7
1 0 1
T_2=31
1 1 0 1 1
T_3=2^7-1
....
J Q
F/F
K
J Q
F/F
K
PN sequence

Basic requirement of PN sequence

- long period
- unpredictable
- balanced (“1”與”0”之個數只差一個,因為 (0 0 0) 不出現)
- low correlation

Ex:

```
1 0 0 1 1 1 0
0 1 0 0 1 1 1
```

∑ -1-1 +1-1+1+1-1=-1

low correlation: 低區域相似(重覆)性

1 0 → -1

0 1 → -1

1 1 → +1

0 0 → +1

balanced run

Ex:

1 0 0 1 1 1 0 : “1”有4個 (2^{n-1})

“0”有3個 ($2^{n-1}-1$)

(n=3) “11”有2個 (2^{n-2})

“10”有2個

“01”有2個

“00”有1個

Synchronous vs. Self-synchronous Stream Ciphers

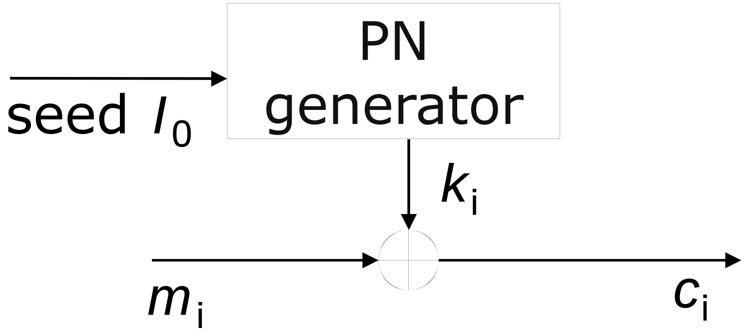
- Two types of stream cipher
  - Synchronous stream cipher

    Key stream is generated *independently* of the message (cipher).
  - Self-synchronous stream cipher

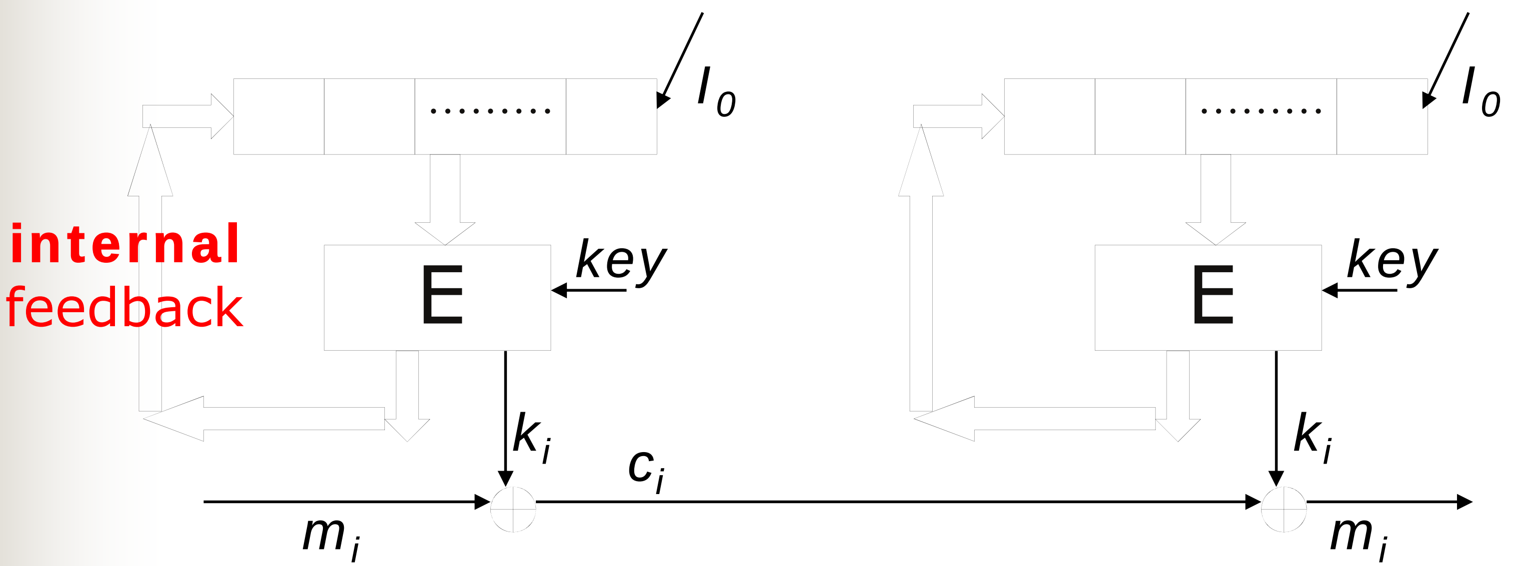
    Key stream is *derived* from some preceding cipher bits.

Synchronous Stream Ciphers

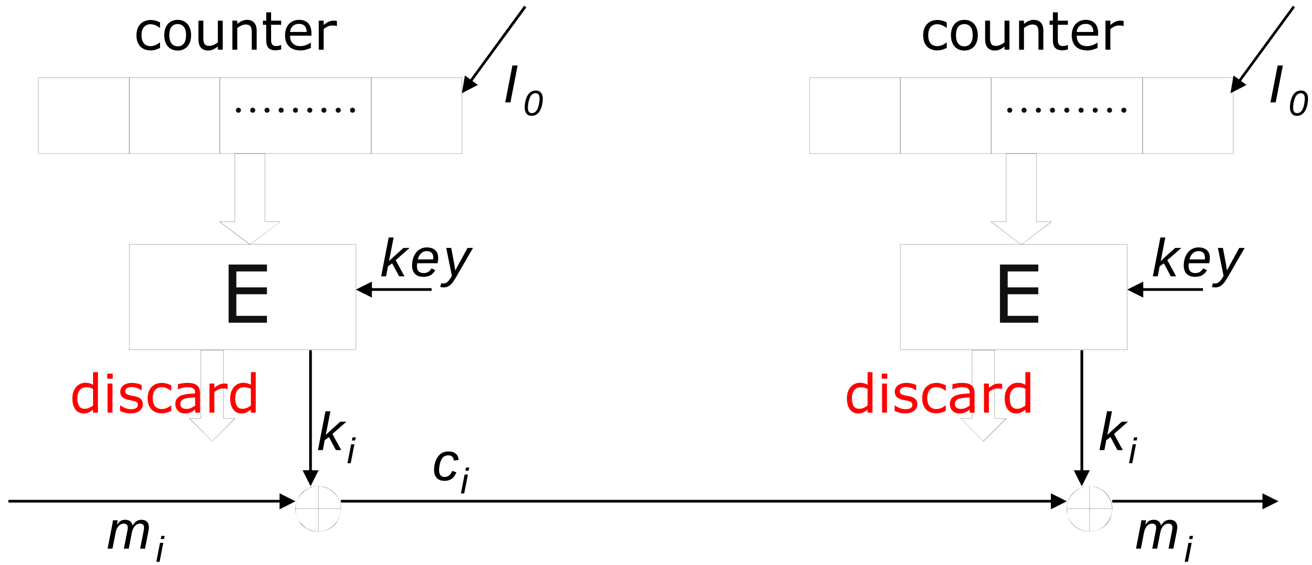
- **Synchronous** stream cipher (basic version)
  - no bit *error propagation* if error happened
  - however, any *bit loss* will cause loss of synchronization



- Synchronous **stream** cipher based on nonlinear **block** cipher to generate the required PN sequence (key stream)
  - Output Feedback Mode



- a modified version: Counter Mode

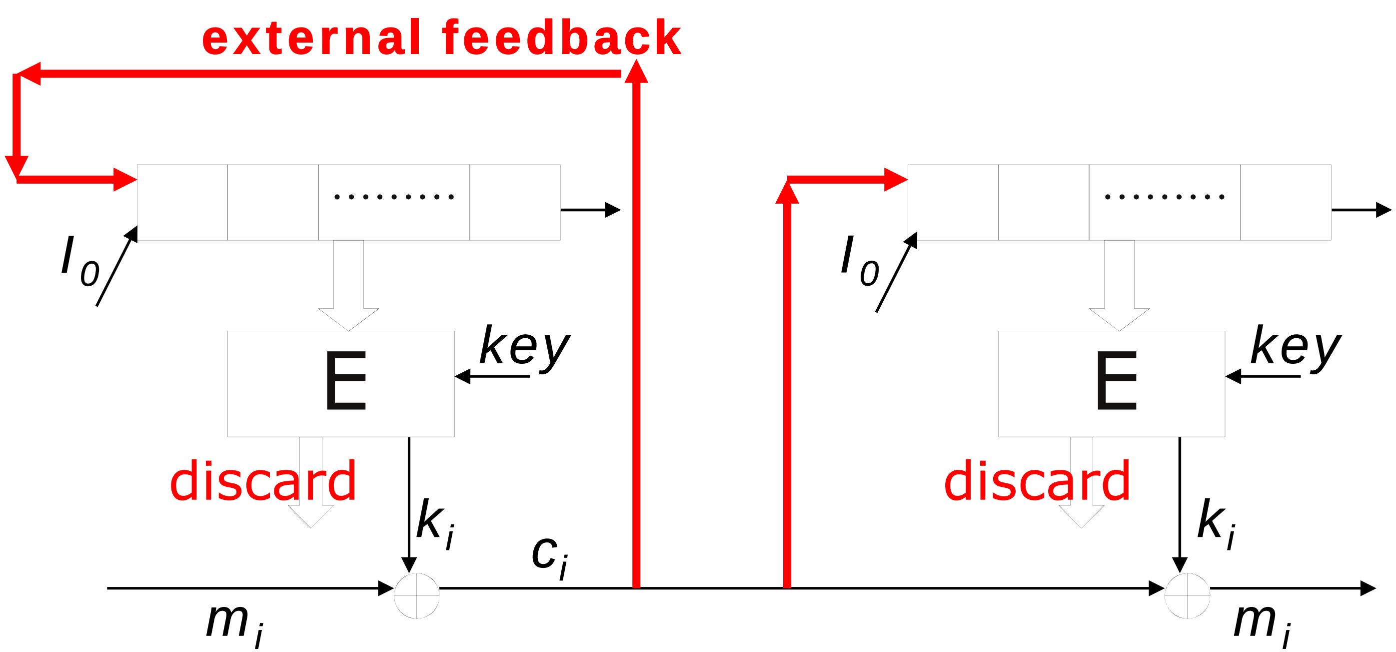


- With counter mode, it is possible to generate k_i without generating the first i-1 key bits by setting counter value to I_0+i-1.
- Why the mode secure? 1 bit modification (even on LSB) on cipher **input** will cause $n/2$ bits modification on cipher **output** under a nondeterministic way.

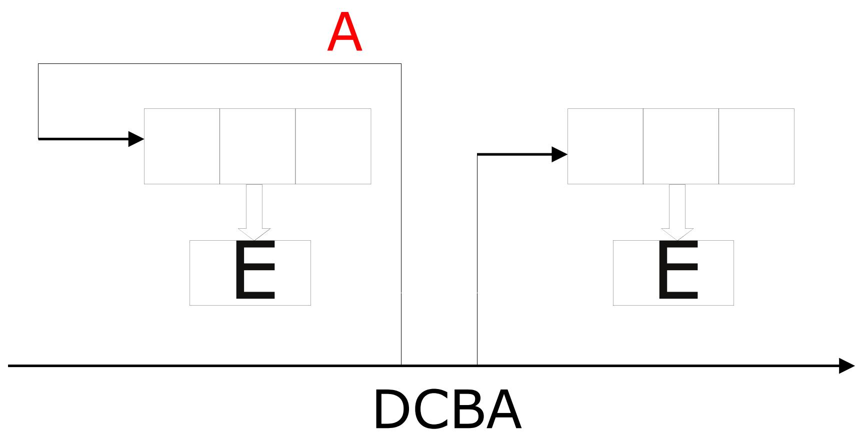
- a modified version: Counter Mode
  - **Random access** is possible by setting counter value to a necessary one.
    - original synchronous mode is not the case

Self-synchronous stream cipher

- Cipher Feedback Mode (CFB)
  - If a ciphertext bit is **lost** during transmission, the registers of both sides will be synchronized again after *n* cycles (*n* is the # of stages).



Ex: when cipher bit "A" lost and let $I_0=(X, Y, Z)$



Original:	X Y Z	X Y Z
	A X Y	X Y Z
	B A X	B X Y
	C B A	C B X
	D C B	D C B

the same, because "A" lost

get synchronized again

From now on, K_i on both sides are the same

- CFB suffers from *error propagation*
  - until the erroneous ciphertext has shifted “out of” the registers
- **Random access** is **effective** by loading the registers with the *n* preceding ciphertext bits ($c_{i-1}, c_{i-2}, \ldots, c_{i-(n-1)}, c_{i-n}$) to get k_i.
- **CFB** can be used to compute a checksum of message because
  - the final state of the registers depends on all message bits so as the checksum

Will *message* feedback mode be self-synchronous ?

A: $k_1 = E(XYZ)$

cipher=$A \oplus k_1$ ⟶ lost

B: $k_2 = E(AXY)$ | $k'_2 = E(XYZ) = k_1$

cipher=$B \oplus k_2$ ⟶ $m = B \oplus k_2 \oplus k'_2 =$ **B'** $\neq B$

C: $k_3 = E($**B**$AX)$ | $k'_3 = E($**B'**$XY)$

cipher=$C \oplus k_3$ ⟶ $m = C \oplus k_3 \oplus k'_3 = C' \neq C$

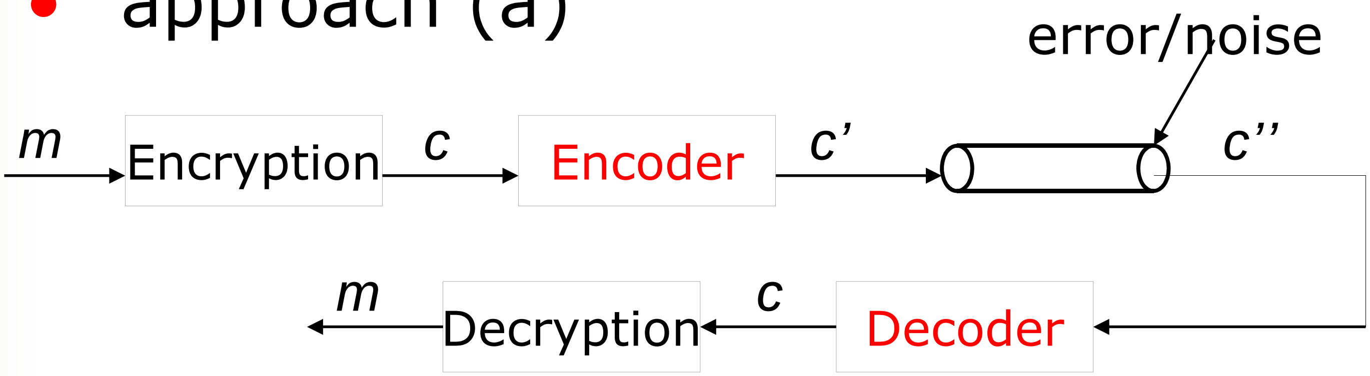
D: $k_4 = E(CBA)$ | $k'_4 = E(C'B'X)$

cipher=$D \oplus k4$ ⟶ $m = D \oplus k_4 \oplus k'_4 = D' \neq D$

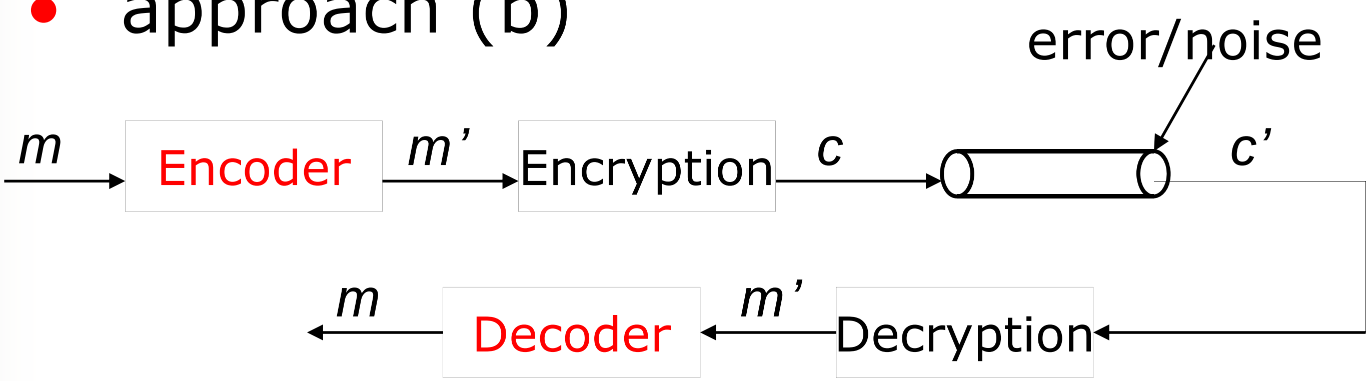
Theoretically, it will NOT get synchronized.

Communication System Problem

- When both *reliability* and *security* are required, which design is better?
  - approach (a)

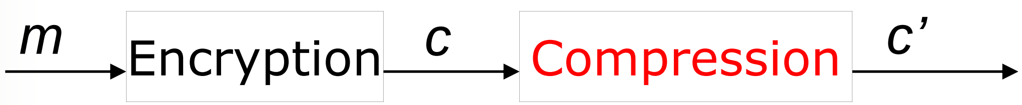


  - approach (b)

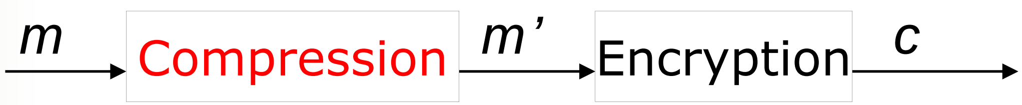


Communication System Problem

- When both *<u>compression</u>* and *security* are required, which design is better?
  - approach (a)



  - approach (b)



Block Cipher

Basics of Block Cipher

- Multiple-round S-box = S-box of same size
- Multiple-round P-box = P-box of same size
- S-box || P-box of same size?
  - example:
    - P permutes (x_1,x_2) to (x_2,x_1)
    - S||P is equivalent to S', so P is in vain

S_{in}	S_{out}	S'_{out} = S||P
00	01	10
01	11	11
10	00	00
11	10	01

- How to implement a large size S-box?
  - why a matter? memory size: $O(2^n)$
  - S-box || P-box of different sizes?
  - **multiple-round** S-P network with smaller size S-box & larger size P-box
  - key issues: (discussed later)
    - multiple rounds
    - permutation P of larger size is necessary

- Example: 4-bit S-box based on two 2-bit S-boxes & one 4-bit P-box
  - given 2-bit S1 & S2, 4-bit permutation P:
    - P permutes (x_1,x_2,y_1,y_2) to (y_1,x_1,y_2,x_2)
    - (S1,S2)||P is equivalent to 4-bit S''
    - without P, (S1,S2) is not a 4-bit S-box!
    - direct 4-bit S-box needs $2^4=16$ space while S-P network based on 2-bit S-box need $2*2^2=8$ space

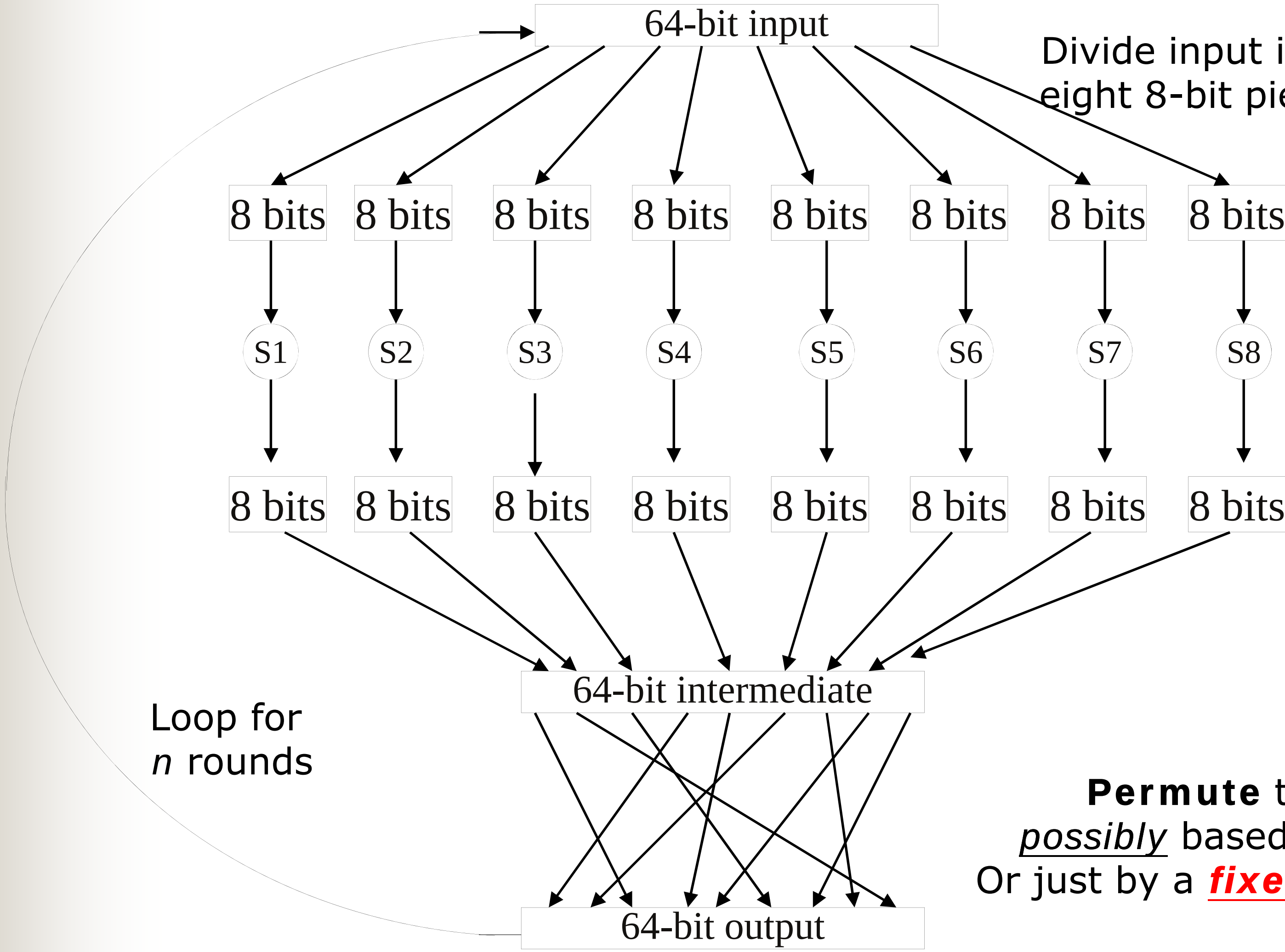
$S1_{in}$	$S1_{out}$
00	01
01	11
10	00
11	10

$S2_{in}$	$S2_{out}$
00	10
01	00
10	11
11	01

$S1_{in}$	$S2_{in}$	$S1_{out}$	$S2_{out}$	$S'_{out} = (S1_{out}, S2_{out}) \| \| P$
00	00	01	10	1 0 0 1
00	01	01	00	0 0 0 1
00	10	01	11	1 0 1 1
00	11	01	01	0 0 1 1
01	00	11	10	1 1 0 1
01	01	11	00	0 1 0 1
01	10	11	11	1 1 1 1
01	11	11	01	0 1 1 1
10	00	00	10	1 0 0 0
10	01	00	00	0 0 0 0
10	10	00	11	1 0 1 0
10	11	00	01	0 0 1 0
11	00	10	10	1 1 0 0
11	01	10	00	0 1 0 0
11	10	10	11	1 1 1 0
11	11	10	01	0 1 1 0

- Why permutation P is necessary?
  - combine S1,S2 (next page); avalanche effect

S-P Network for Block Cipher



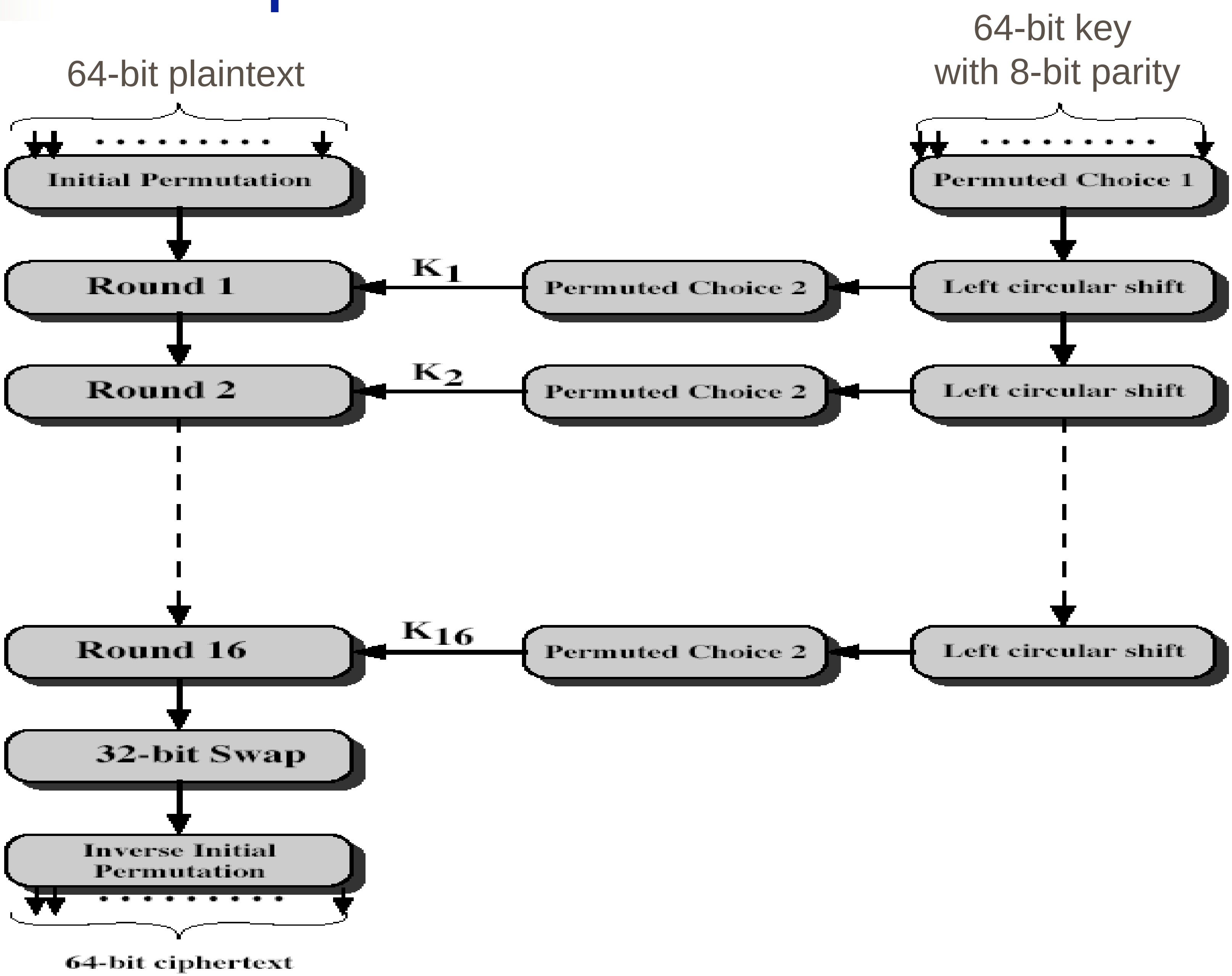
Divide input into eight 8-bit pieces

Substitution functions derived from the ***key***

Loop for *n* rounds

Permute the bits, *possibly* based on the **key**
Or just by a ***fixed*** permutation

DES Cipher

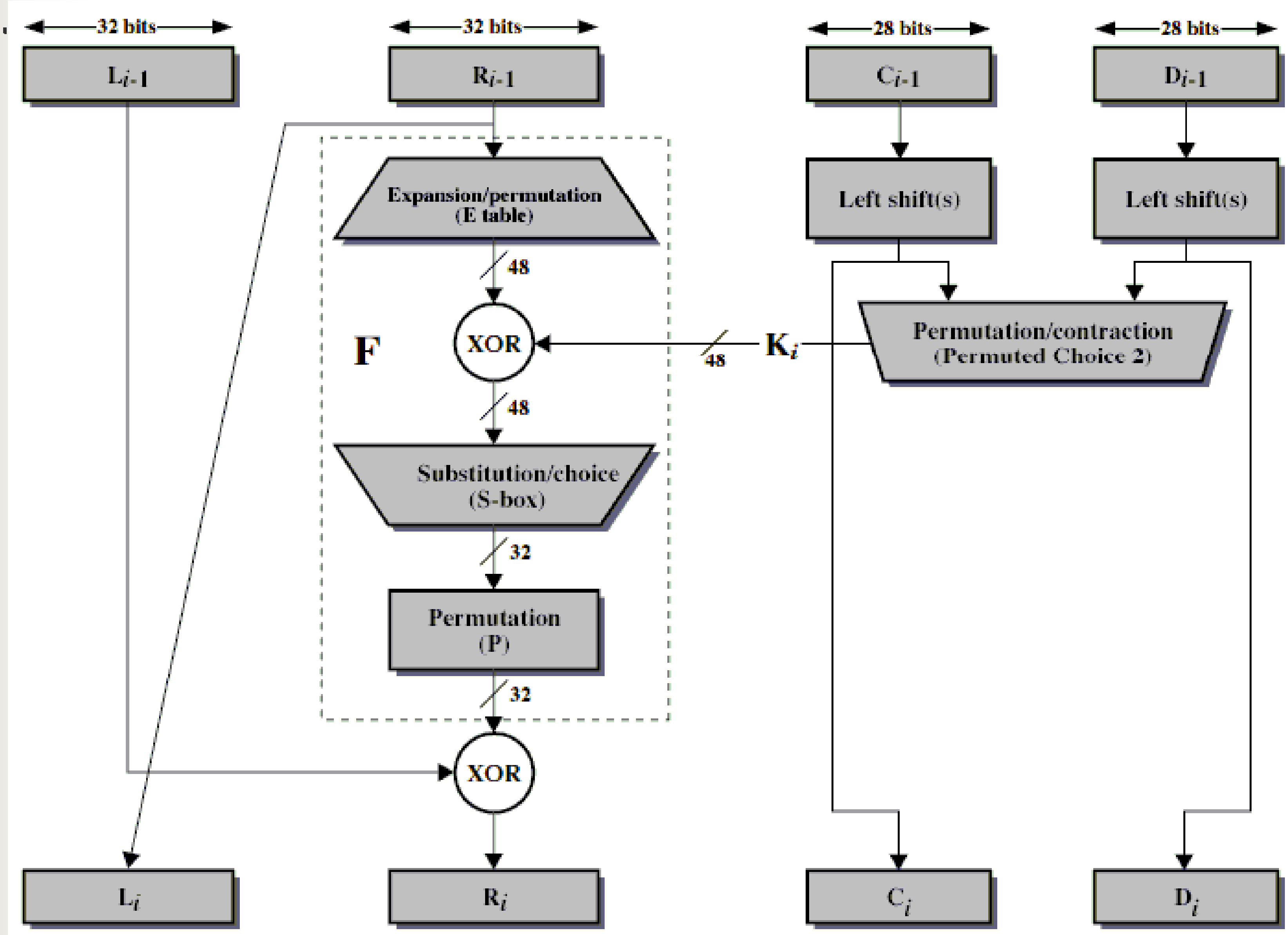

64-bit plaintext
64-bit key
with 8-bit parity
Initial Permutation
Permuted Choice 1
Round 1
K1
Permuted Choice 2
Left circular shift
Round 2
K2
Permuted Choice 2
Left circular shift
Round 16
K16
Permuted Choice 2
Left circular shift
32-bit Swap
Inverse Initial Permutation
64-bit ciphertext

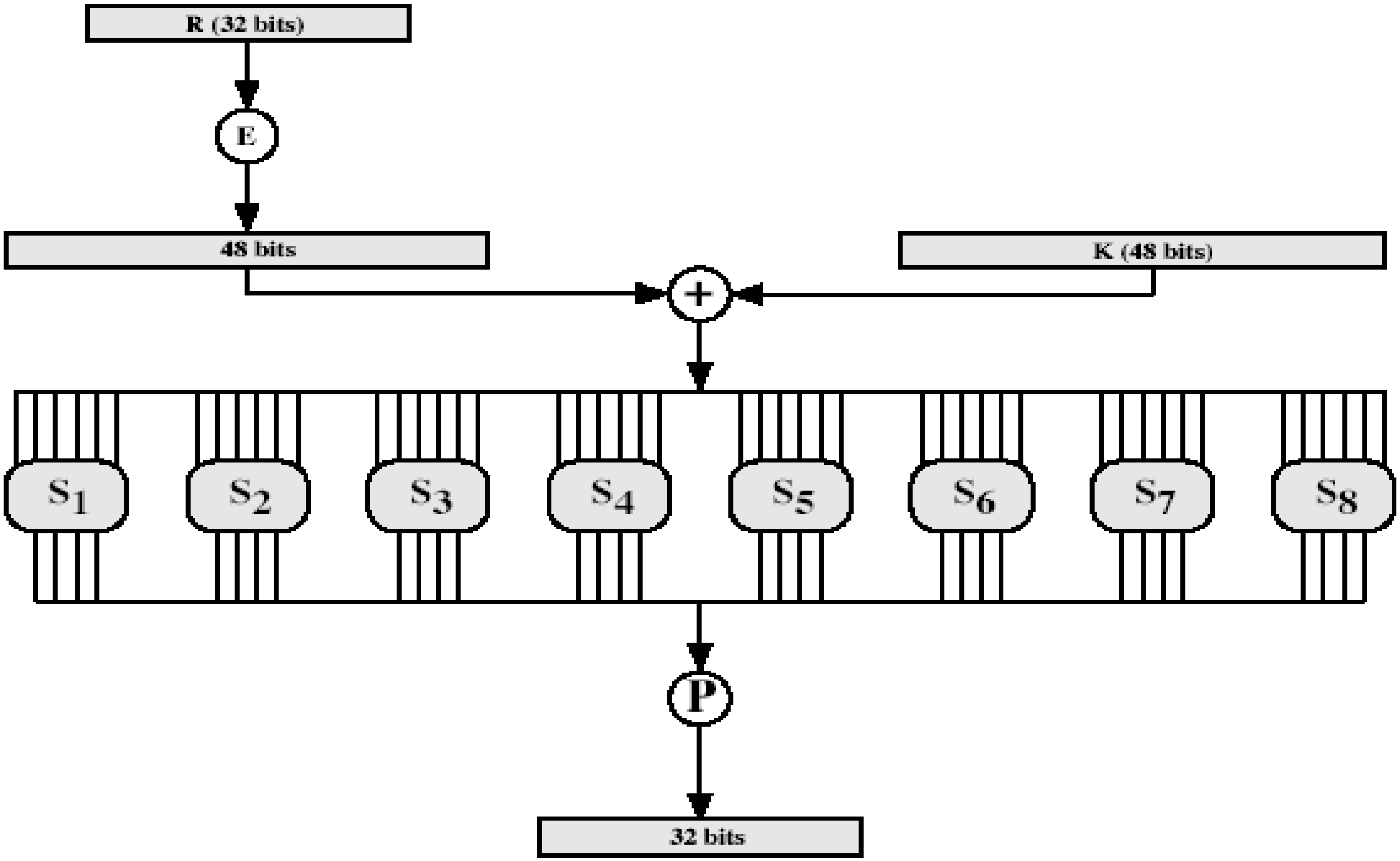
DES Round Structure

- uses two 32-bit L & R halves
- any Feistel cipher can be described as:

  $L_i = R_{i-1}$

  $R_i = L_{i-1}$ XOR $\mathbf{F}(R_{i-1}, K_i)$
- takes **32**-bit R half and 48-bit subkey and:
  - **expand**s R to **48** bits using perm E
  - **add**s to subkey
  - passes through 8 **S-boxe**s to get 32-bit result
  - finally **permute**s this using 32-bit perm P


32 bits
32 bits
28 bits
28 bits
L$_{i-1}$
R$_{i-1}$
C$_{i-1}$
D$_{i-1}$
Expansion/permutation (E table)
Left shift(s)
Left shift(s)
48
F
XOR
48
K$_i$
Permutation/contraction (Permuted Choice 2)
48
Substitution/choice (S-box)
32
Permutation (P)
32
XOR
L$_i$
R$_i$
C$_i$
D$_i$


R (32 bits)
E
48 bits
K (48 bits)
+
S1
S2
S3
S4
S5
S6
S7
S8
P
32 bits

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**

Permutation **P**

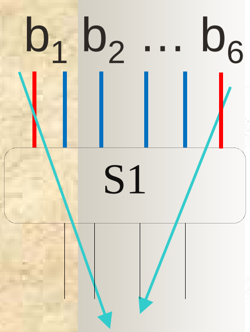
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

$R_{i-1} = r_1\ r_2 \ldots r_{32}$

$T = E(R_{i-1})$

$T = r_{32}\ r_1\ r_2 \ldots r_{32}\ r_1$

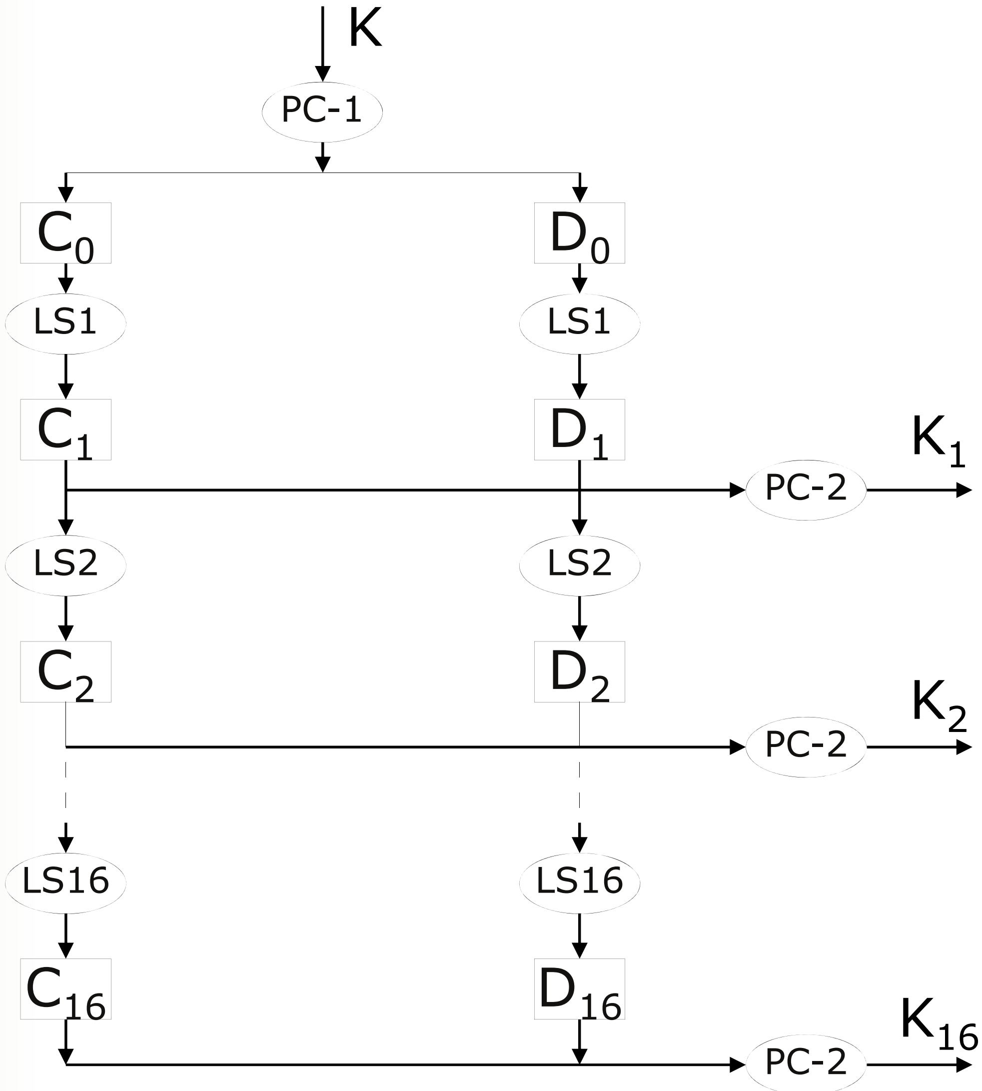
S-box (substitution box)



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

DES Key Schedule

- forms subkeys (**round keys**) used in each round
- consists of:
  - **initial permutation** of the key (PC1) which selects 56 bits as two 28-bit halves
  - 16 stages consisting of:
    - **rotating each half** separately either 1 or 2 places depending on the **key rotation schedule** K
    - **selecting** **24 bits** from **28 bits** of each half
    - **permuting** them by PC2 for use in function F



iteration	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

Permutation PC-1

(from 64 bits to 56 bits)

for C_0

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**

for D_0

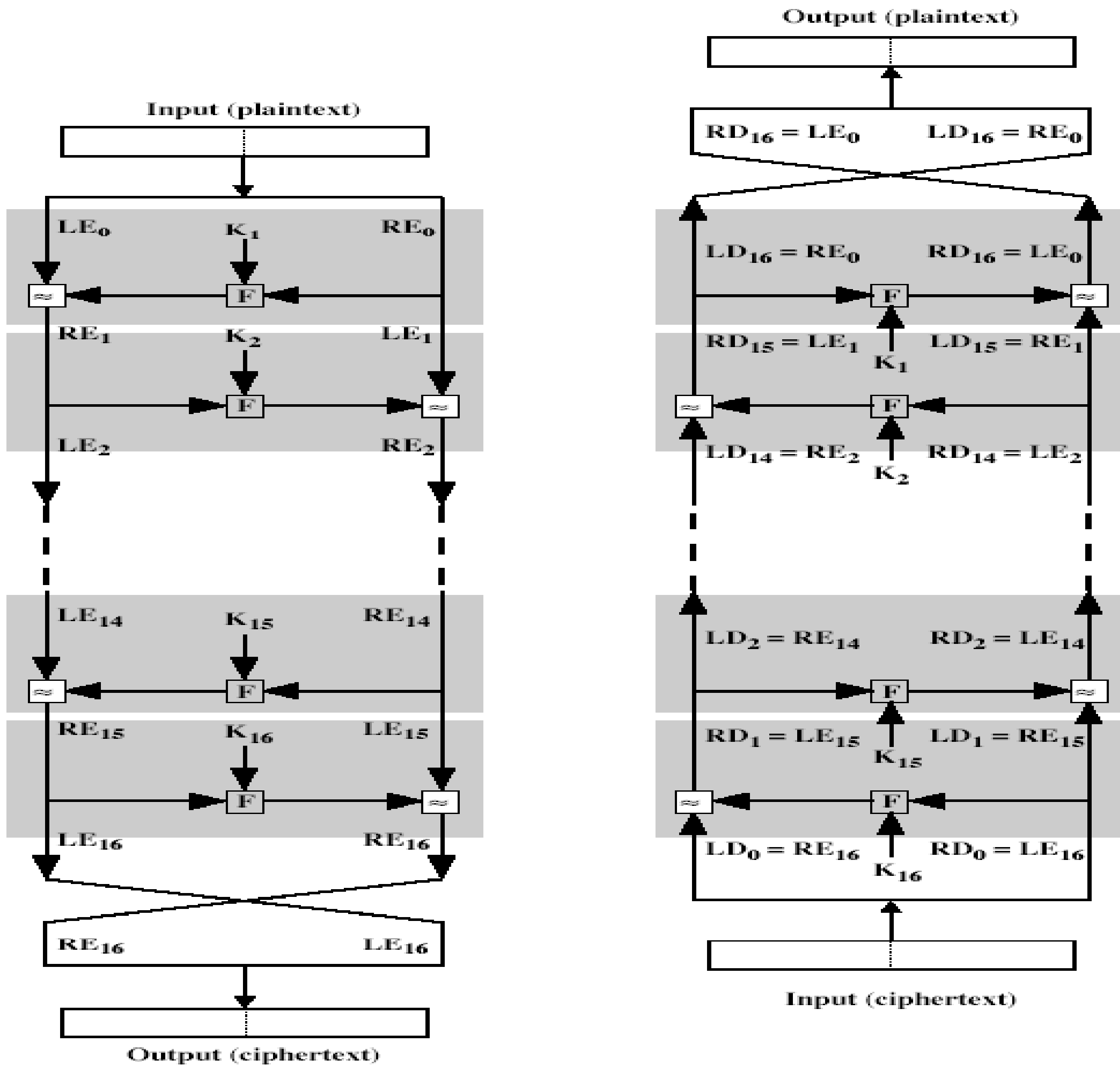
Permutation PC-2

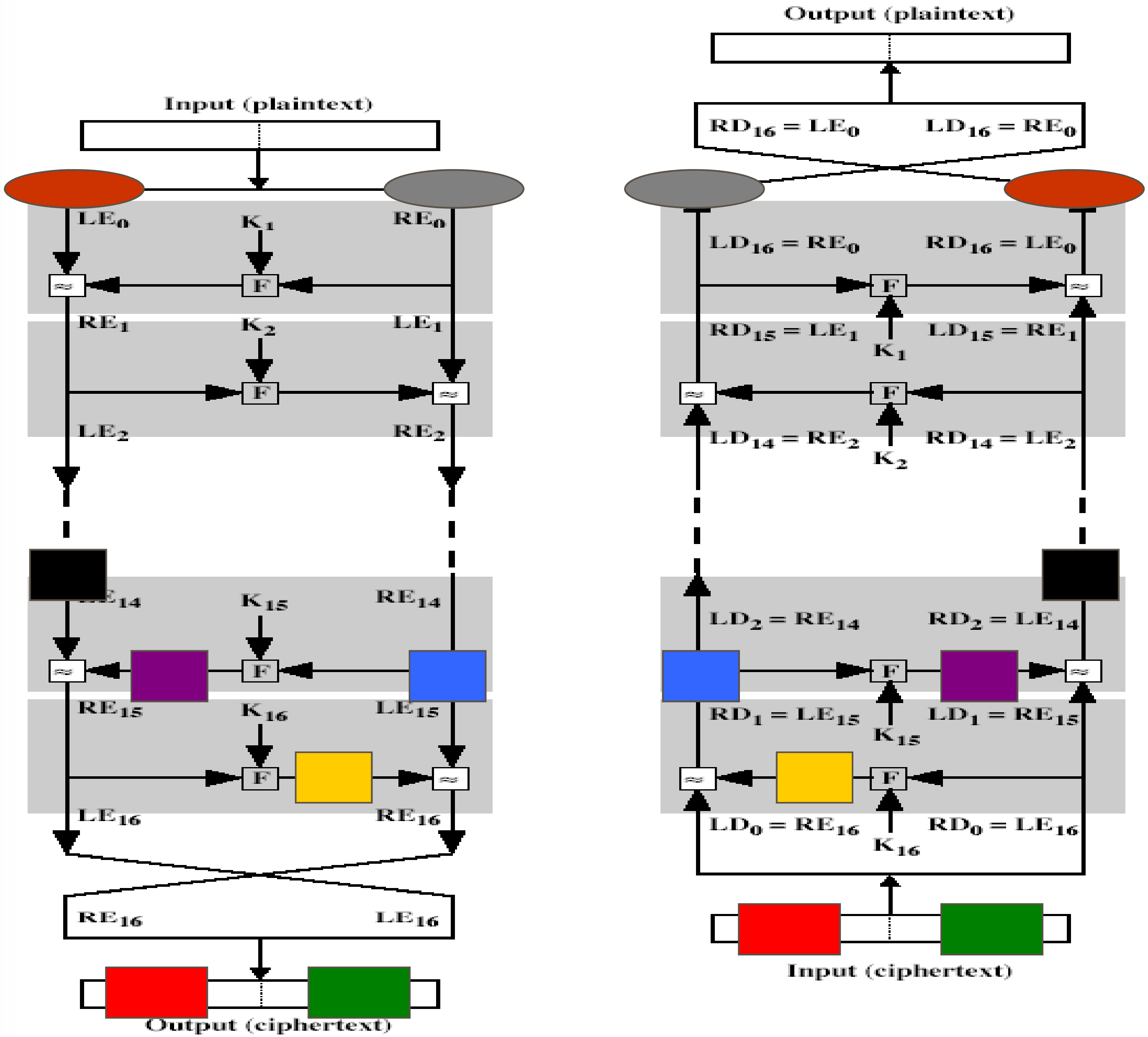
from C_i & always for S1 to S4 !

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

from D_i & always for S5 to S8 !

DES Decryption Process





Output (plaintext)
RD16 = LE0
LD16 = RE0
Input (plaintext)
LE0
K1
RE0
F
RE1
K2
LE1
LD16 = RE0
RD16 = LE0
RD15 = LE1
LD15 = RE1
K1
LE2
RE2
LD14 = RE2
RD14 = LE2
K2
[redacted]
K15
RE14
LD2 = RE14
RD2 = LE14
RE15
K16
LE15
RD1 = LE15
LD1 = RE15
K15
LE16
RE16
LD0 = RE16
RD0 = LE16
K16
RE16
LE16
Input (ciphertext)
Output (ciphertext)

Why DES a correct cipher ?

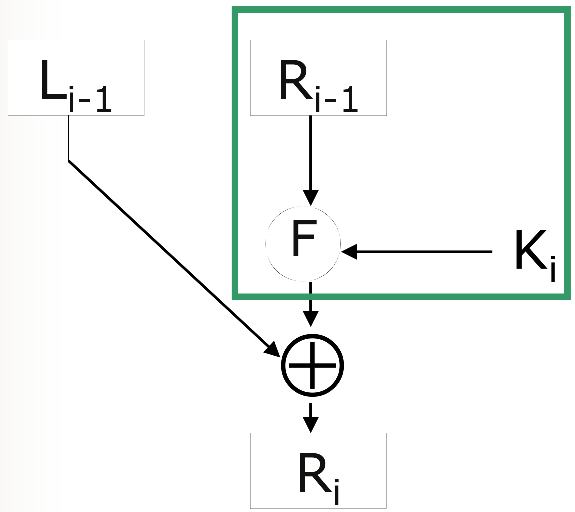
- a cipher should provide its decryption operation (the inverse function)
- so a cipher should not be a **multiple-to-one** mapping
- why DES always

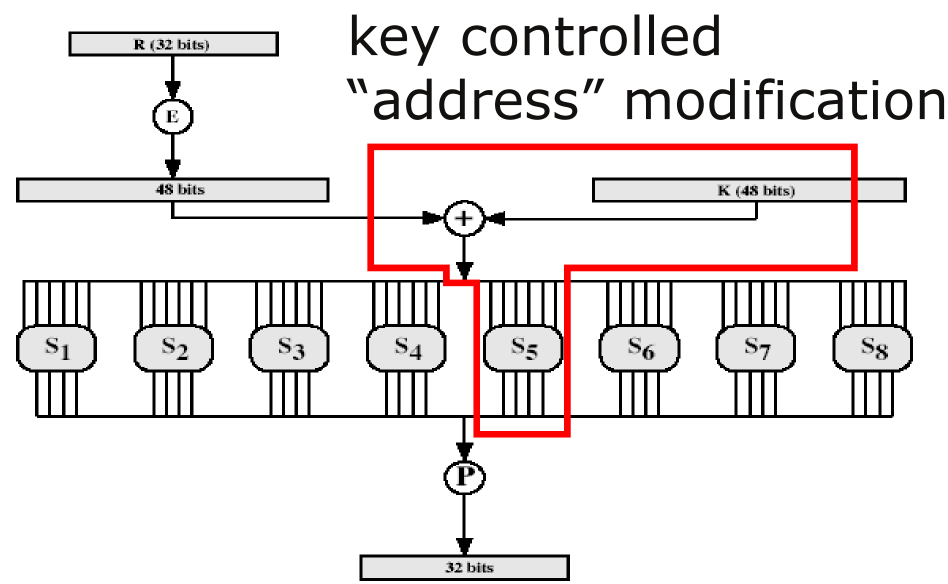
  $$DES_K(M_1) \neq DES_K(M_2) \text{ if } M_1 \neq M_2$$

  - why DES can decrypt correctly even if it has temporary internal **data expansion**, 32-to-48 bits then **data compression** 48-to-32 bits again?

Substitution and Permutation

- Key-controlled substitution is used in DES

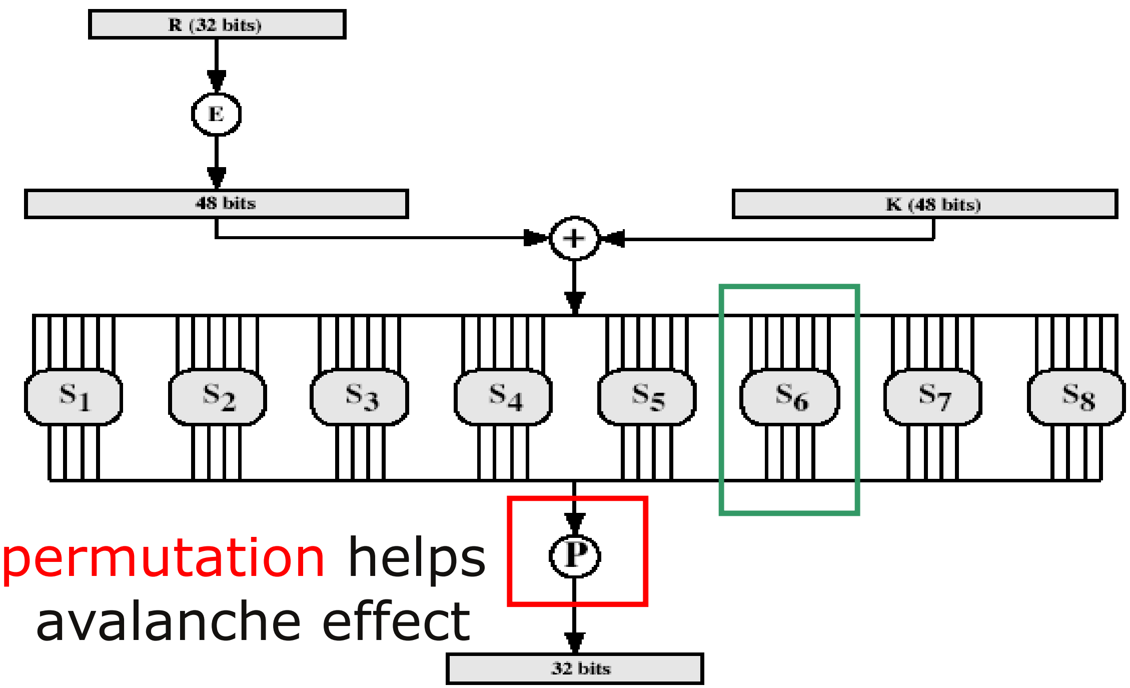




- **No** key-controlled permutation is used in DES, but **why** still need permutation?
  - Key controlled permutation of long size is not easy to implement

Avalanche Effect

- desirable property of encryption algorithm
- a change of **one** input bit or key bit results in changing approx. **half** of output bits
- DES exhibits strong avalanche



permutation helps avalanche effect

Strength of DES

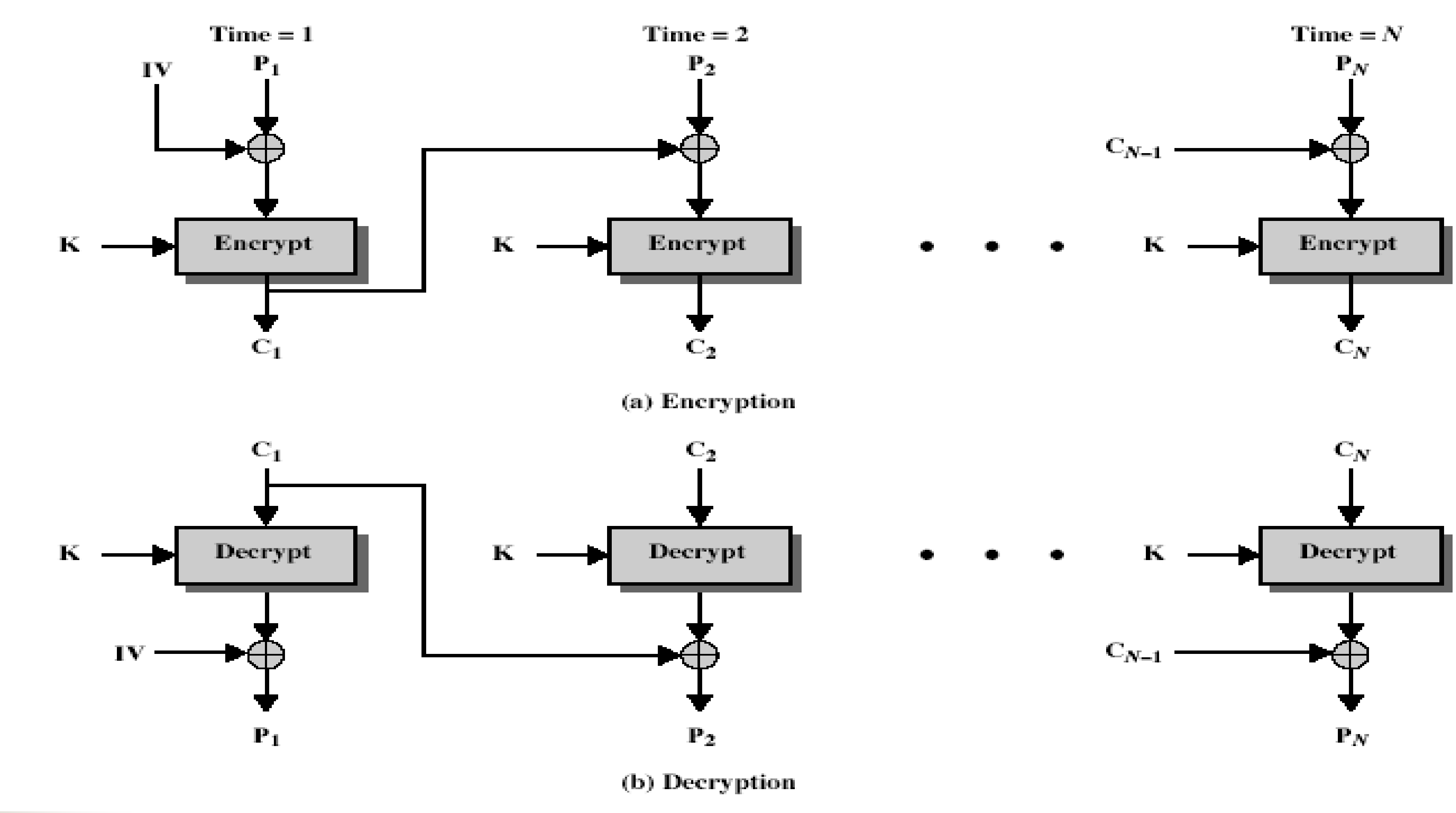
- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- brute force search looks hard
- recent advanced analytic attack & hardware physical characteristics exploiting have shown possible
  - differential cryptanalysis; linear cryptanalysis; related key attacks
  - implementation attacks
- now considering alternatives to DES – **Triple DES** and **AES** (Advanced Encryption Standard)

Stream cipher vs. Block cipher

- Stream cipher can protect against "ciphertext searching" attack because of randomized encryption.
  - naive use of block cipher however can not due to the same key used
  - "cipher block chaining" (CBC) can enhance security of block cipher
- But, in synchronous stream cipher it is more easier to modify a ciphertext character (or bit) without being detected than in the case of block cipher.
  - CFB (self-synchronous) can improve security

Cipher Block Chaining (CBC)

- each **previous cipher** block (as **random mask**) is chained with current plaintext block



Advantages & Limitations of CBC

- advantage: random mask
  - same plaintexts lead to different ciphertexts $C_i = E_K(M \oplus C_{i-1})$ & $C_{i+1} = E_K(M \oplus C_i)$ then $C_i \neq C_{i+1}$ if masks are different $C_{i-1} \neq C_i$
- disadvantage:
  - an error in C_i leads to incorrect P_i & P_{i+1}
    - fortunately, **no** error propagation
  - bitwise modification of P_1 is possible by changing Initial Vector (IV)
    - so, IV must be known to sender & receiver (or fixed) or encrypted in ECB mode

Message Authentication Code (MAC)

- generated by an algorithm $\mathtt{MAC}_K(M)$ that creates a small fixed-sized block
  - depending on message *M* and a “shared” key K

$$\mathtt{MAC} = \mathtt{MAC}_K(M)$$

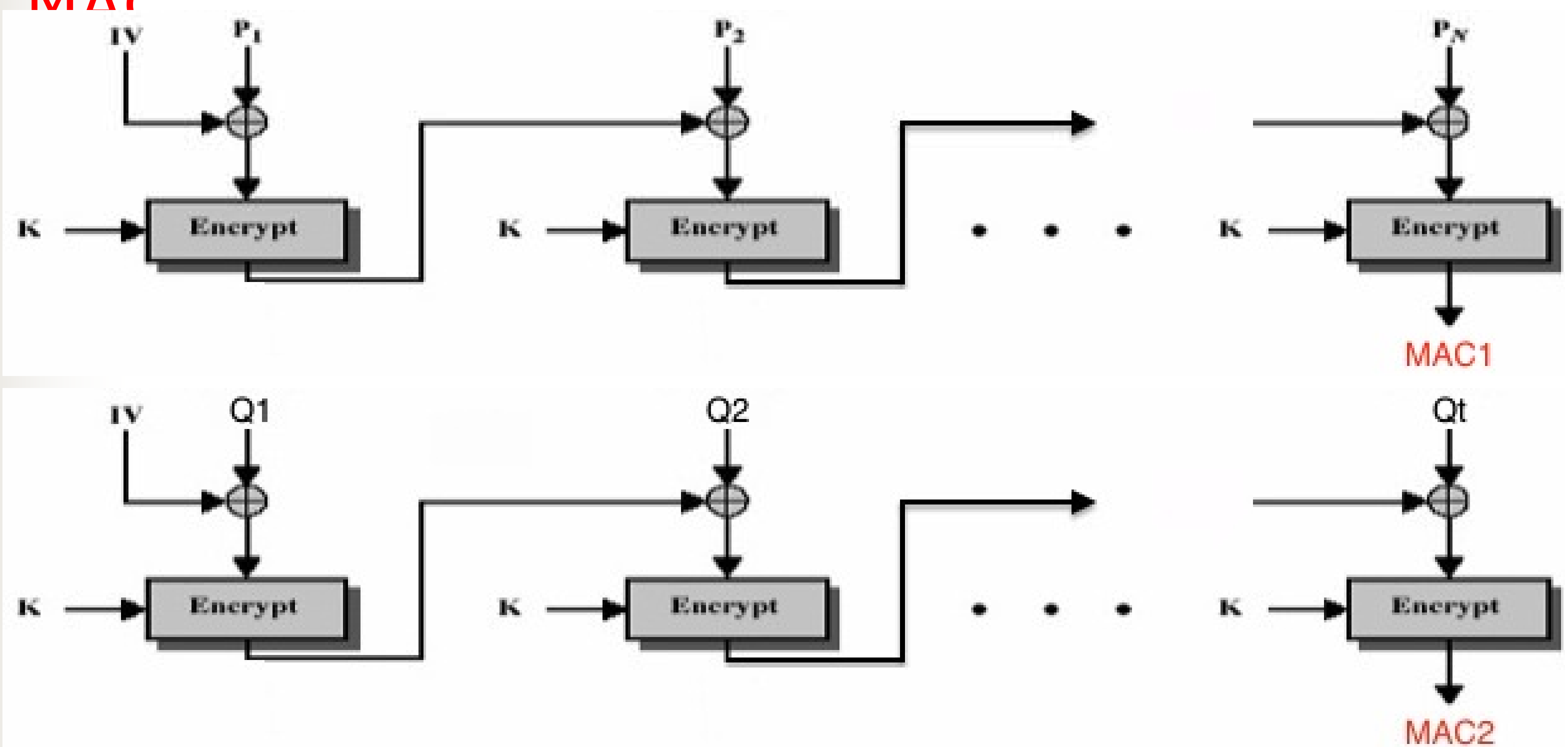
- **appended** to message as a **checksum**
- receiver performs same computation on message and checks whether it matches the received MAC
- provides assurance that message is **unaltered** and comes from claimed sender
  - giving *M* and its MAC but without K, it is infeasible to find *M***’** with the same MAC

Using Symmetric Ciphers for MAC

- can use the cipher block chaining mode (e.g., CBC) and use **final block** as the MAC
- but this CBC-based MAC is somewhat weak for security reason
- HMAC is usually used as secure MAC algorithm

CBC-based MAC -- Attack 1

- Attack-1: concatenation attack of two MACs
  - given MAC_1 of message P: $(P_1, P_2, \ldots, P_n)$ &
    MAC_2 of message Q: $(Q_1, Q_2, \ldots, Q_t)$
  - forgery of $MAC_K(P||(Q_1 \oplus IV \oplus MAC_1), Q_2, \ldots, Q_t)$ = MAC

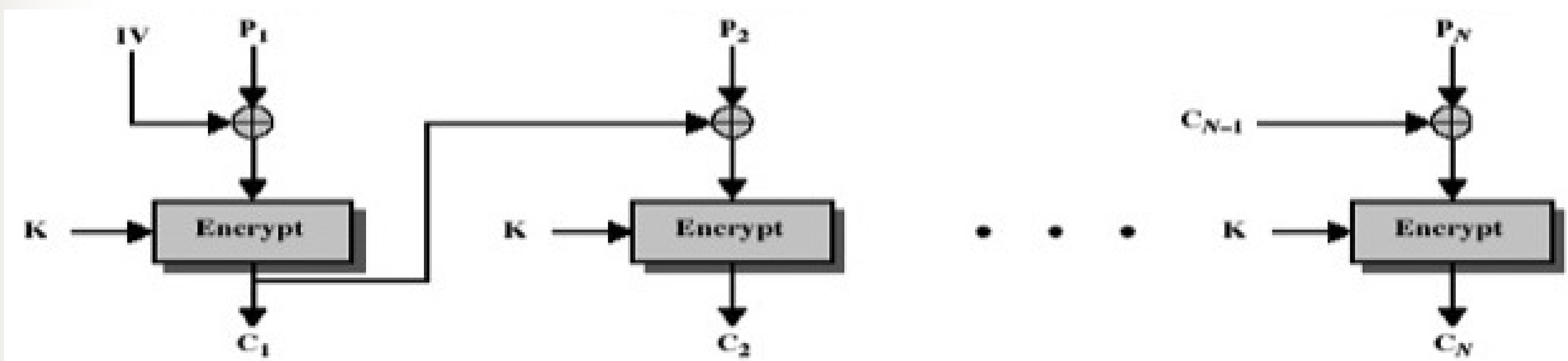


- Attack-1: concatenation attack of two MACs
  - **solution**: to protect MAC by sending $E_{K2}(MAC)$
    - disadvantage: but you need two keys, one for computing MAC & one for encrypting MAC
  - **Note**: if **one** key used: forgery is still possible! $MAC_K(P||\mathbf{0}||(Q_1 \oplus IV \oplus E_K(MAC_1)), Q_2, \ldots, Q_t) = MAC_2$ where "$||\mathbf{0}$" simulates $E_K(MAC_1)$

CBC-based MAC -- Attack 2

- Attack-2: when CBC used as both encryption & MAC with a same key "K"
  - C_N as a ciphertext block & as MAC (**kept safely**)

(P'_1, P'_2, … P'_{N-1}, P'_N)



(C'_1, C'_2, … C'_{N-1}, C_N)

  - if attacker modifies (C'_1, C'_2, … C'_{N-1}) but <u>**not**</u> C_N
    - for communication, can of course modify C_N
  - user/receiver decrypts (P'_1, P'_2,… P'_{N-1}, P'_N) then computes MAC=C_N so no detection is possible

- Attack-2: when CBC used as both encryption & MAC with a same key "K"
  - **solution**: use different keys for CBC encryption & CBC-MAC
    all ciphertext blocks = CBC_E_{K1}(message)
    MAC = CBC_MAC_{K2}(message)
    - disadvantage: but you need two keys & need twice effort of block cipher computation for each message block
  - **Note**: MAC=E_{K2}(**last** block of cipher) is insecure!