COMP3260/6360 Data Security

Lecture 6

Prof Ljiljana Brankovic

Lecture Overview

1. Stream Ciphers
   a) Self-Synchronising Stream Ciphers
   b) Synchronous Stream Ciphers
   c) Design Principles for Stream Ciphers

2. Block Ciphers
   a) Feistel Block Cipher

3. Confusion and Diffusion

4. The Data Encryption Standard (DES)
   a) DES Encryption/Decryption
   b) Key Generation
   c) Avalanche Effect
   d) Completeness Effect

Data Encryption Standard (DES)

- Chapter 4. Block Ciphers and the Data Encryption Standard
- Chapter 8. Random Bit Generation and Stream Ciphers
- These lecture notes (based on the text and "Cryptography and Data Security" by D. Denning [1])

Note that in-text references and quotes are omitted for clarity of the slides. When you write an essay or report it is very important that you use both in-text references and quotes where appropriate.

Stream and Block Ciphers

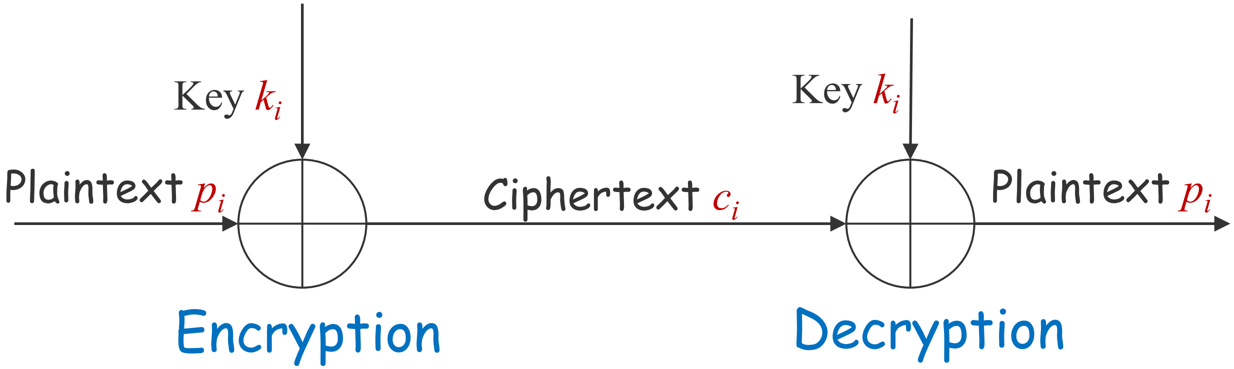
- ***Stream ciphers*** convert plaintext into ciphertext one bit (or one byte) at a time; in the same plaintext message, the same plaintext bit (or byte) is encrypted with a different key every time it appears (eventually the key will repeat in the periodic ciphers). Example: Vigenere, Vernam, rotor machine.
  - ***Advantage***: stream ciphers are faster and easier to implement than block ciphers.

- ***Block ciphers*** convert plaintext into ciphertext one block (typically 128 bit) at a time; in the same plaintext message, the same block is encrypted with the same key every time it appears (thus to the same ciphertext). Example: Playfair, transposition with period d, monoalphabetic substitution (blocksize 1)
  - ***Advantage***: block ciphers can reuse keys, and provide both ***confusion*** and ***diffusion***.

Stream Ciphers

In a typical stream cipher, a stream of plaintext bits is XORed with a stream of key bits to produce the stream of ciphertext bits: $c_i = p_i \oplus k_i$

To decrypt, ciphertext bits are XORed with the identical stream of key bits to produce the plaintext: $c_i \oplus k_i = p_i \oplus k_i \oplus k_i = p_i$



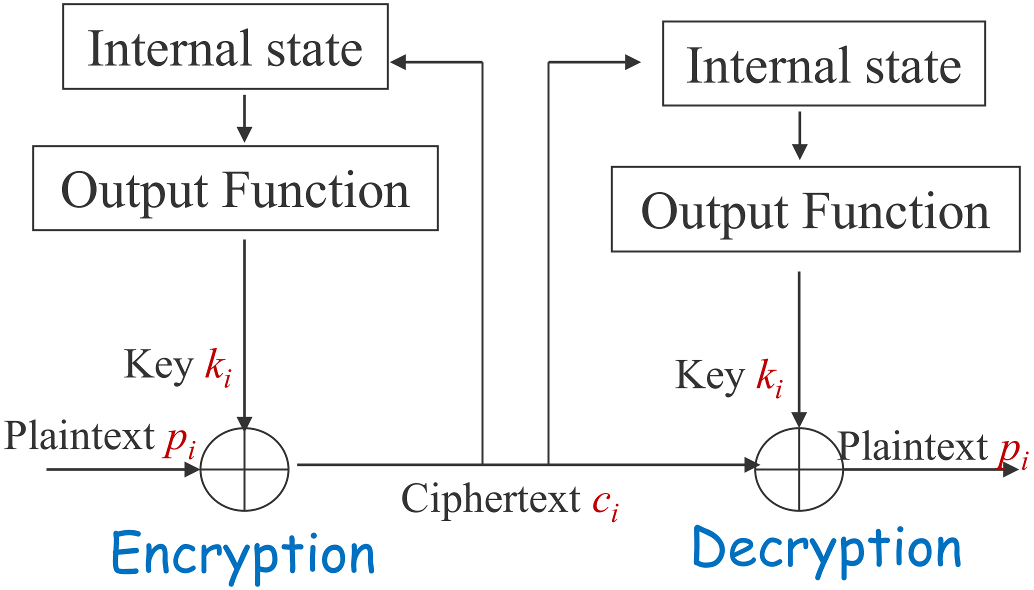
If the key stream is non-repeating and random, this is a one-time pad – it is perfectly secure. In practice, one-time pad is rarely used because of the need for secure exchange of long keys.

Keys used in practice look random, but are deterministically generated and can be reproduced at the decryption end.

Stream ciphers differ in ways the key is generated and fall into 2 categories: ***self-synchronising*** and ***synchronous***.

Self-Synchronising Stream Ciphers

In a self-synchronising stream cipher each bit in the key stream is a function of a fixed number (say n) of previous ciphertext bits.



The internal state depends only on n previous ciphertext bits.

The output function takes as its input the internal state and generates the key as its output.

The output function must be cryptographically strong, otherwise an intruder can intercept the ciphertext stream, generate the key stream and obtain plaintext.

Self-synchronising stream ciphers are:

- error propagating: each garbled ciphertext bit will result in n incorrect plaintext bits before it re-synchronizes.
- vulnerable to play-back attack: an intruder can replay some old ciphertext (that, for example, credits his bank account); after re-synchronising, the ciphertext will decrypt as normal.

Synchronous Stream Ciphers

The synchronous stream ciphers have a key generated independently of ciphertext. The same key stream must be generated at the decryption end, so must be deterministic.

Advantage: Synchronous ciphers are not error propagating - each garbled ciphertext bit will result in only one garbled plaintext bit.

Disadvantage: If one ciphertext bit gets lost during the transmission, all the subsequent bits will be deciphered incorrectly - the sender and receiver must re-synchronise; moreover, they should not use the same key stream as before.

What can happen if the same key stream is used twice?

An intruder can intercept a ciphertext (he does not know the plaintext or the key).

Original plaintext: p_1 p_2 p_3 p_4 …

Original key: k_1 k_2 k_3 k_4 …

Original ciphertext: c_1 c_2 c_3 c_4 …

Synchronous Stream Ciphers

Then the intruder can insert a single bit p' in the same (resent) plaintext encrypted under the same key, and again intercepts the ciphertext:

New plaintext: $p_1 \; p' \; p_2 \; p_3 \; p_4 \ldots$

Original key: $k_1 \; k_2 \; k_3 \; k_4 \; k_5 \ldots$

New ciphertext: $c_1 \; c_2' \; c_3' \; c_4' c_5 \ldots$

The intruder can now determine the entire plaintext:

$k_2 = c_2' \oplus p'$ **and then** $p_2 = c_2 \oplus k_2$

$k_3 = c_3' \oplus p_2$ **and then** $p_3 = c_3 \oplus k_3$

$k_4 = c_4' \oplus p_3$ **and then** $p_4 = c_4 \oplus k_4$

and so on.

This attack is known as an ***insertion attack***. To protect against insertion attack, we must never use the same key stream.

Design Principles for Stream Ciphers

1. The key-stream should be as close to a random stream as possible, e.g.,
    - approximately equal number of 0s and 1s, for a stream of bits;
    - approximately equal number of all the possible 256 bytes, for byte stream.

2. The period of the key-stream should be as long as possible.

3. The input key that is passed to Pseudo Random Number Generator (PRNG) should be sufficiently long – at least 128bits.

Block Ciphers

Recall that a block cipher takes as input an n-bit block of plaintext and produces an n-bit block of ciphertext. In order to enable encryption, for a given key each plaintext block must encrypt into a *unique* ciphertext block. Such mapping is called reversible (one-to-one).

Be careful not to confuse this with the discussions about perfect secrecy where any ciphertext can come from any plaintext, but for different keys.

Irreversible mapping is also called one-way mapping (function).

Example 1:

Reversible mapping		Irreversible mapping	
Plaintext	*Ciphertext*	*Plaintext*	*Ciphertext*
00	11	00	11
01	10	01	10
10	00	10	**01**
11	01	11	**01**

Block Ciphers

The mapping between plaintext and ciphertext blocks should be arbitrary (equivalent to general monoalphabetic cipher).

How about the block size?

- If it is small this converts to monoalphabetic substitution cipher - not secure.
- If it is very large, and mapping is arbitrary, statistical properties of the plaintext are not preserved in the ciphertext and the system is very secure, but implementation is not feasible.

If, on the other hand, mapping is not arbitrary but given with a system of equations, the implementation is easy, but the system is not secure.

Confusion and Diffusion

Roughly speaking, confusion obscures the local structure of the plaintext, while diffusion obscures the global structure.

Confusion means that the cipher should hide local patterns in the plaintext. For example, Caesar cipher replaces every letter with another letter, but fails to hide double letters.

Diffusion refers to spreading out the plaintext over the ciphertext. Ideally, each plaintext letter should affect many ciphertext letters.

Stream ciphers rely on confusion alone. Block ciphers use both confusion and diffusion.

Feistel Block Cipher

Feistel block cipher (1973) illustrates the underlying principles of many modern block encryption algorithms.

Feistel cipher is a product cipher, which means that it uses a sequence of two or more basic ciphers, so that the final result is cryptographically stronger than any of the components. In particular, Feistel cipher uses a sequence of substitutions and permutations. Such ciphers are also called S-P ciphers.

Feistel cipher is based on the work by Shannon, who proposed a development of a product cipher that alternates ***confusion*** and ***diffusion*** functions.

The Feistel cipher takes as input a plaintext block of size $2w$ and a key K.

The plaintext block is divided into two halves L_0 and R_0 which are then passed though n rounds and finally combined together to produce the ciphertext.

Each round has the same structure but uses a different subkey - the subkeys K_1 K_2 K_3 … K_n are derived from K and are different from each other.

Each round first applies a round function F to the right half of the data, and takes the XOR of the result and the left half of the data. Then the two halves are interchanged.

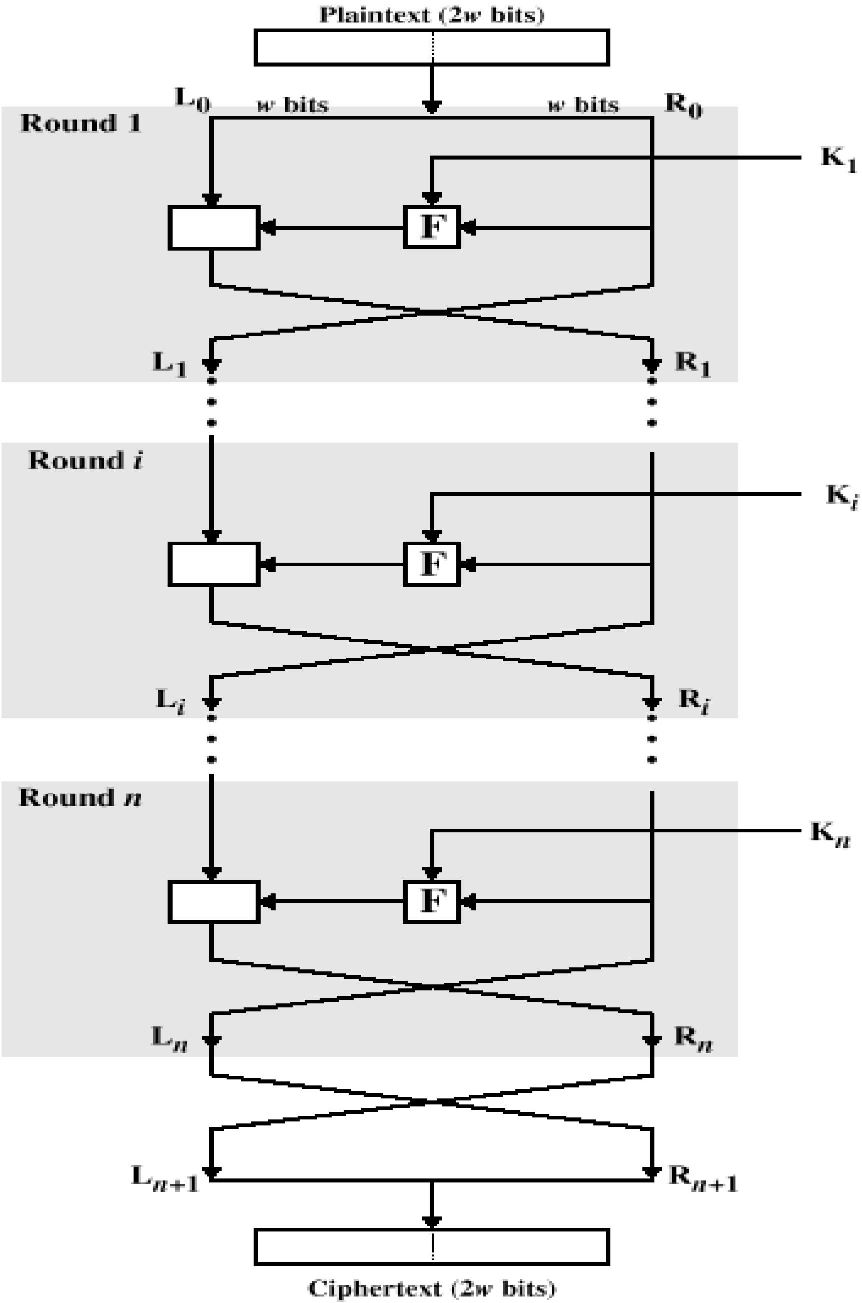


Figure 3.5 Classical Feistel Network

Parameters of Feistel Cipher

- ***Block size*** - the larger the block size, the greater security but slower encryption/decryption; typical block size is 128 bits (Data Encryption Standard (DES) uses 64 bit blocks).

- ***Key size*** - the larger the key size, the greater security but slower encryption/decryption; 128 bits is a common key size (Data Encryption Standard (DES) uses 56 bit key).

- ***Number of rounds*** - typically 16

- ***Subkey generation algorithm*** - the more complex the algorithm, the cipher harder to break, but also harder to analyse and discover weaknesses

- ***Round function*** - the more complex the function, the cipher harder to break, but also harder to analyse and discover weaknesses

Feistel Decryption

The Feistel decryption is essentially the same as encryption - the only difference being that the subkeys are used in reverse order.

We shall illustrate this by showing that

$LD_1 = RE_{15}$ and $RD_1 = LE_{15}$

At the end of the encryption we have:

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

At the beginning of decryption we have:

$$\begin{aligned}
LD_1 &= RD_0 = LE_{16} = RE_{15} \\
RD_1 &= LD_0 \otimes F(RD_0, K_{16}) = \\
&= RE_{16} \otimes F(RE_{15}, K_{16}) = \\
&= \big(LE_{15} \otimes F(RE_{15}, K_{16})\big) \otimes F(RE_{15}, K_{16}) \\
&= LE_{15}
\end{aligned}$$

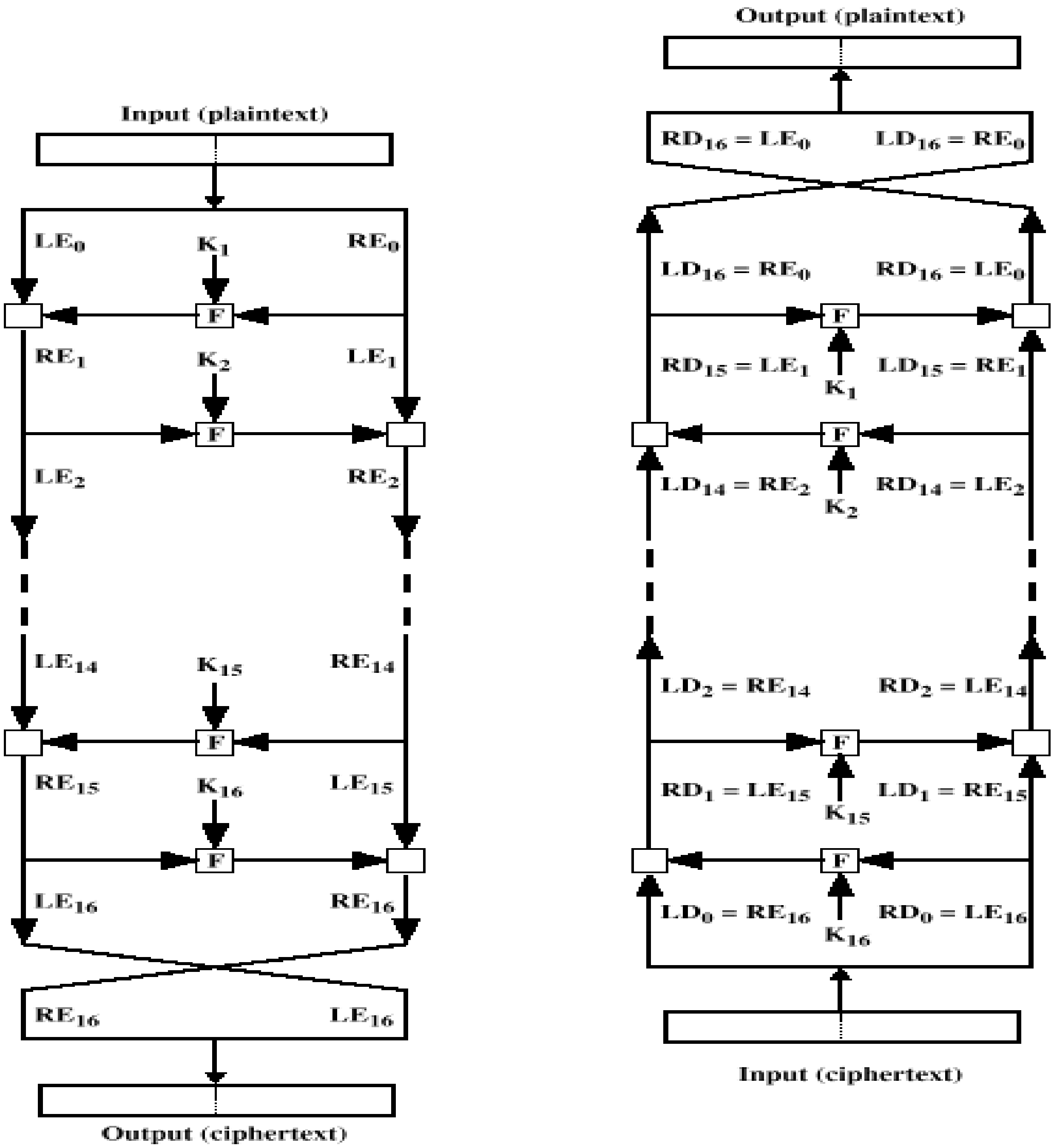


Figure 3.6 Feistel Encryption and Decryption

The Data Encryption Standard (DES)

In early 70's, the National Bureau of Standards (now called the National Institute of Standards and Technology - NIST) issued a request for a proposal for standard encryption algorithm. They had the following design criteria:

- The algorithm must provide a high level of security.
- The algorithm must be completely specified and easy to understand.
- The security of the algorithm must depend solely on the key and not on the secrecy of algorithm.
- The algorithm must be available to all users.
- The algorithm must be adaptable for use in diverse applications.
- The algorithm must be economically implementable.

None of the submissions came close to meeting the requirements, until they received what is now known as DES.

DES was developed by IBM, but with an input from National Security Agency (NSA). It was based on an earlier algorithm by IBM, called Lucifer.

In 1975 the details of DES were published and subjected to criticism from agencies and the general public.

The Data Encryption Standard (DES)

The two main points of criticism were:

- NSA reduced the key size from the original 128 bits in Lucifer to 56 bits. It was feared the key is too short to withstand the brute-force attack.
- NSA also modified some of DES's S-boxes; although the boxes themselves were public knowledge, the analysis behind them were classified; many feared that NSA has installed a trap-door that would enable them to decrypt a ciphertext without the key.

In 1977, DES was adopted by National Bureau of Standards as a national encryption standard.

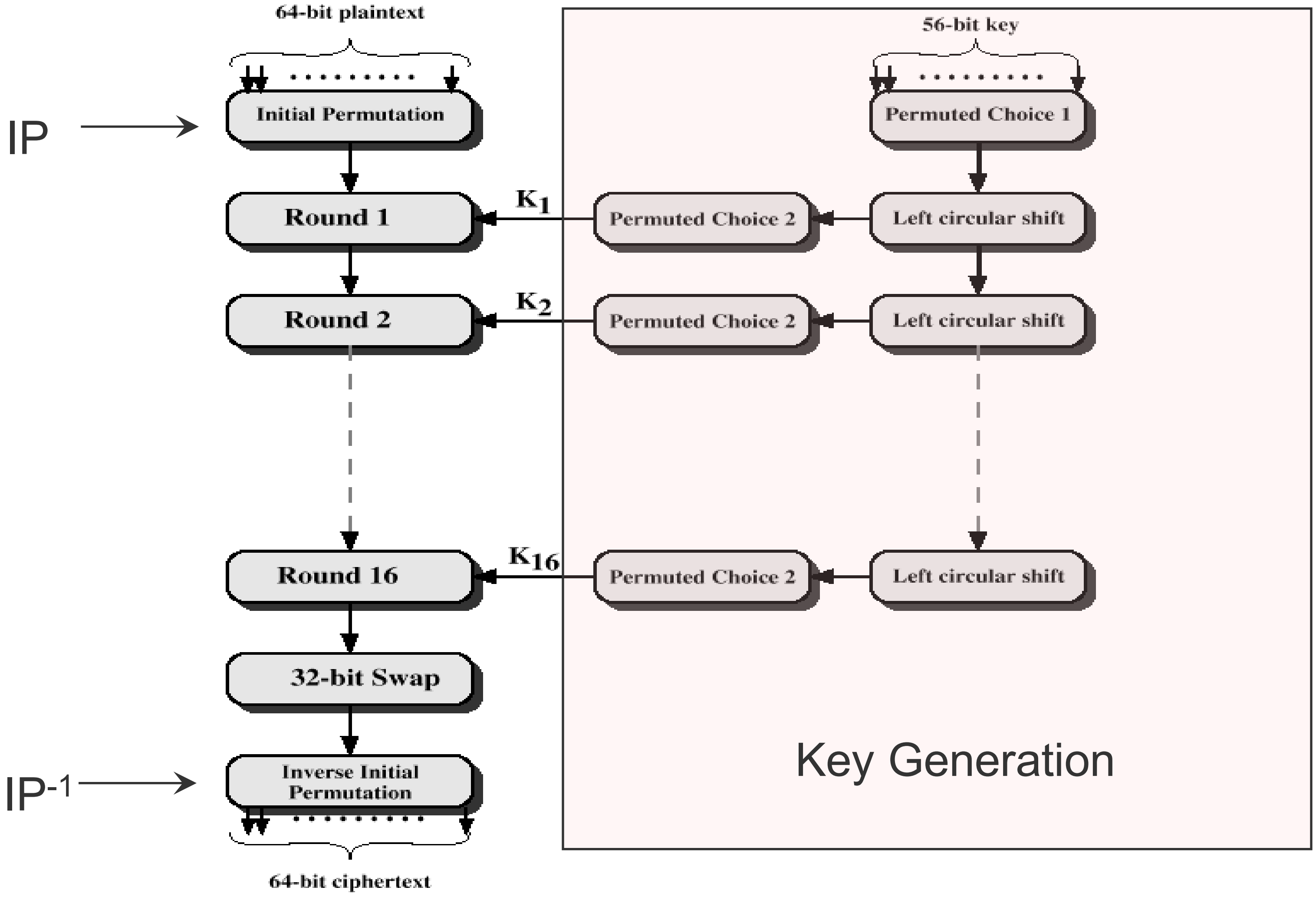
In 1998, the Electronic Frontier Foundation spent less than $250,000 to built a DES cracking machine that can get a key in three days.

Not surprisingly, DES is not considered secure anymore; nevertheless, it is still in use (as Triple-DES), as it is built into many applications (especially financial – for example EFTPOS).

Despite its main weakness (short key), DES is cryptographically strong; it is resistant to some attacks described in the open scientific literature in 90's.

DES Encryption

DES encrypts **64 bit block** of data with **56-bit key**. It has **16 rounds**, and each round incorporates both substitution and permutation.



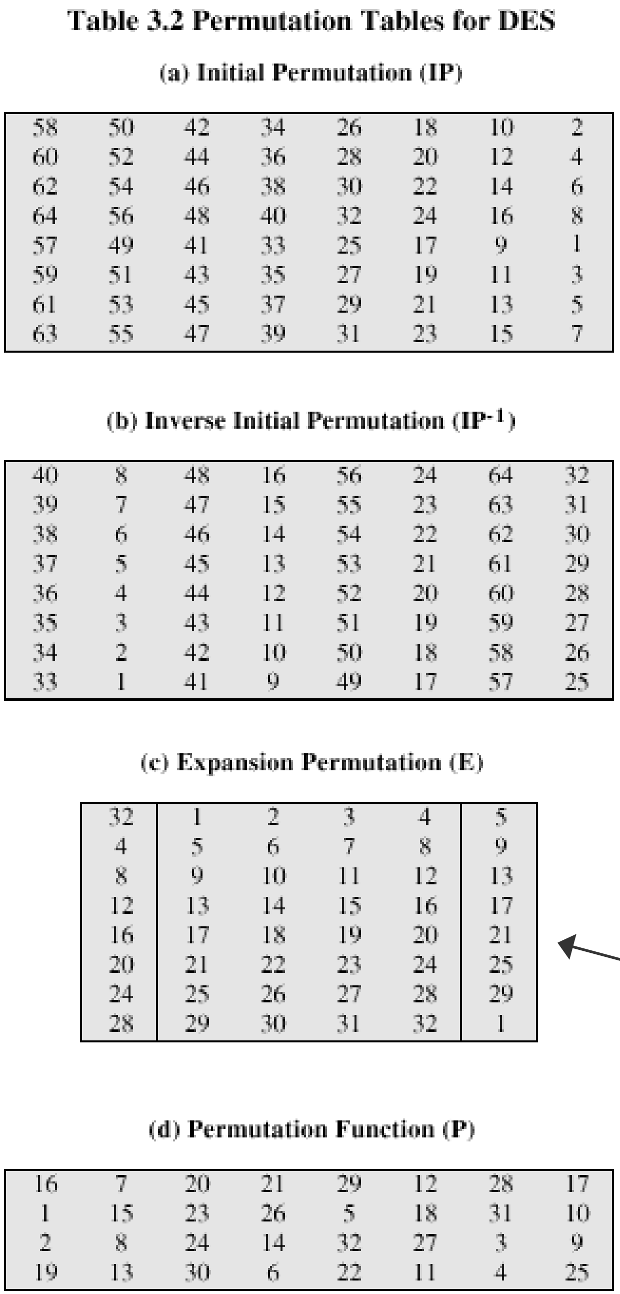


Table 3.2 Permutation Tables for DES

(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

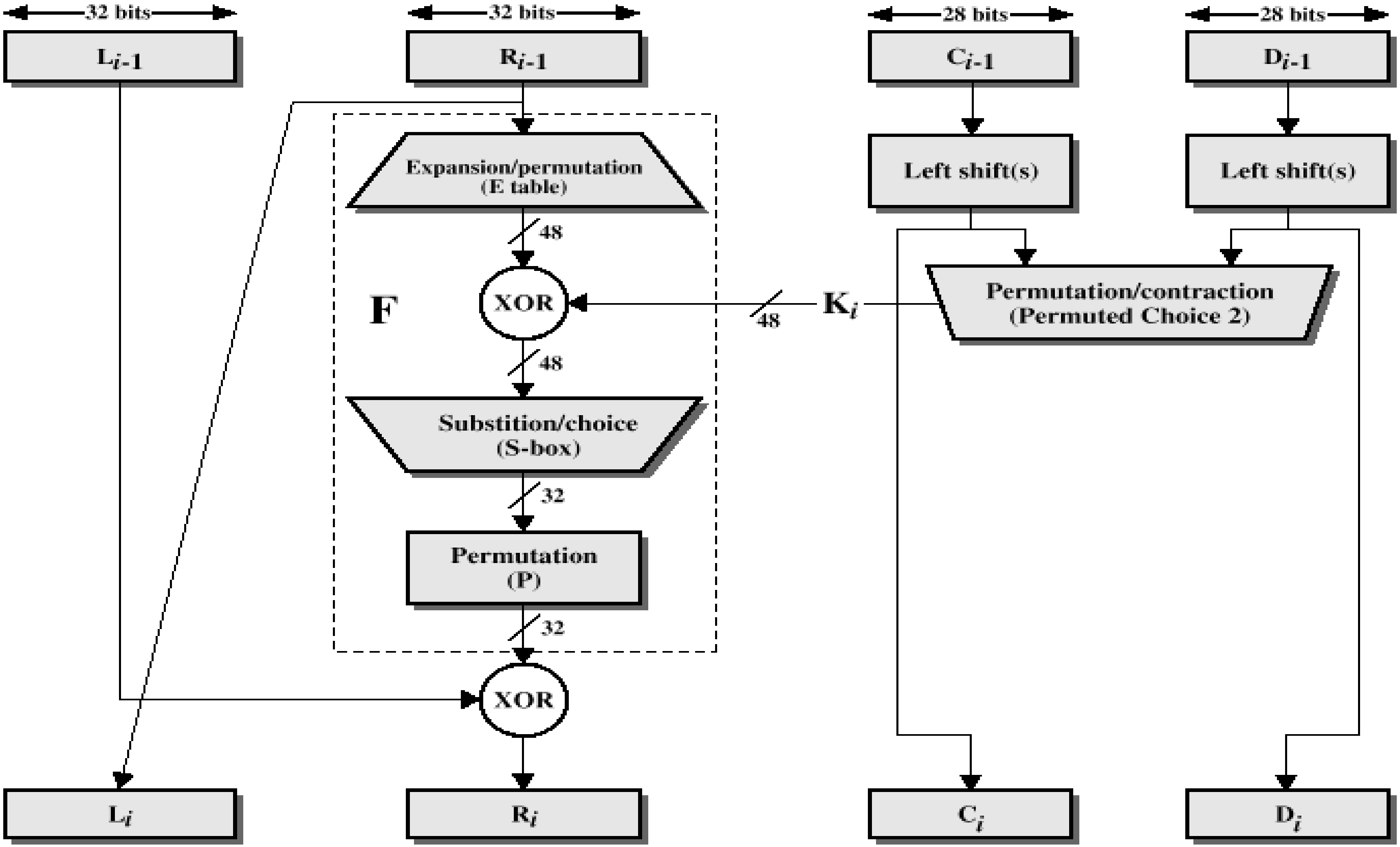
Used before and after the 16 rounds

Used at start and end of function (F)

Details of Single Round

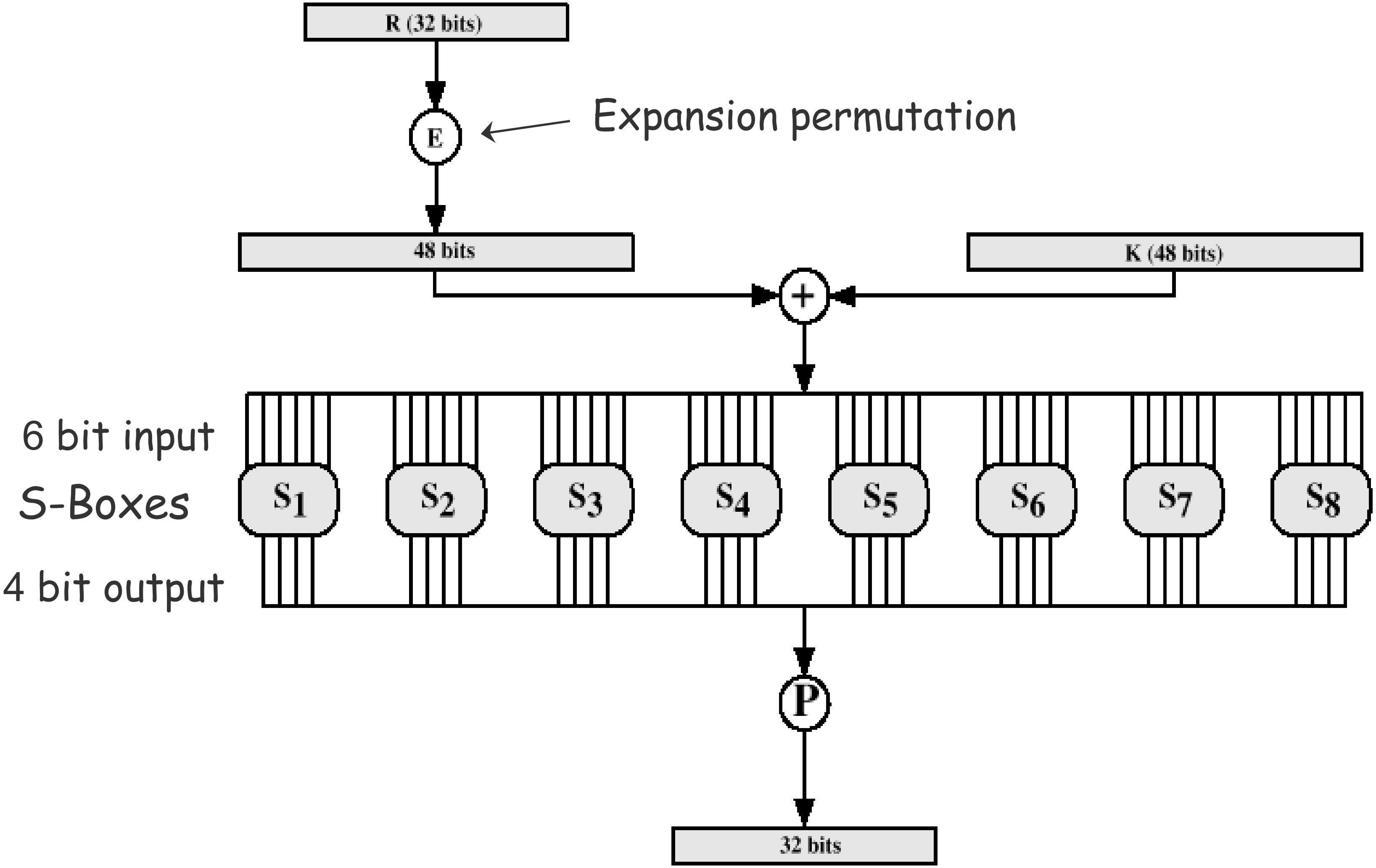
As in any classical Feistel cipher, the left and right half of each intermediate 64 bit value are treated separately and denoted L_i and R_i:

$$L_i = R_{i-1} \quad \text{and} \quad R_i = L_{i-1} \square \; F(R_{i-1}, Ki)$$



Round Function (F)

The round key K_i is 48 bits long. The 32 bit round input $R_{i-1}$is first expanded to 48 bits by using Expansion permutation E, and it is then XOR-ed with K_i. The 48 bit result is then passed through a substitution function (S-boxes), that also reduces the result to 32 bits, which is then permuted using permutation P.



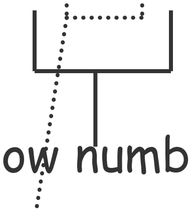
S-Boxes

Each row defines a general reversible substitution.

Note that column and row positioning starts at 0.

Input to S-Box 1 (S_1):

101100

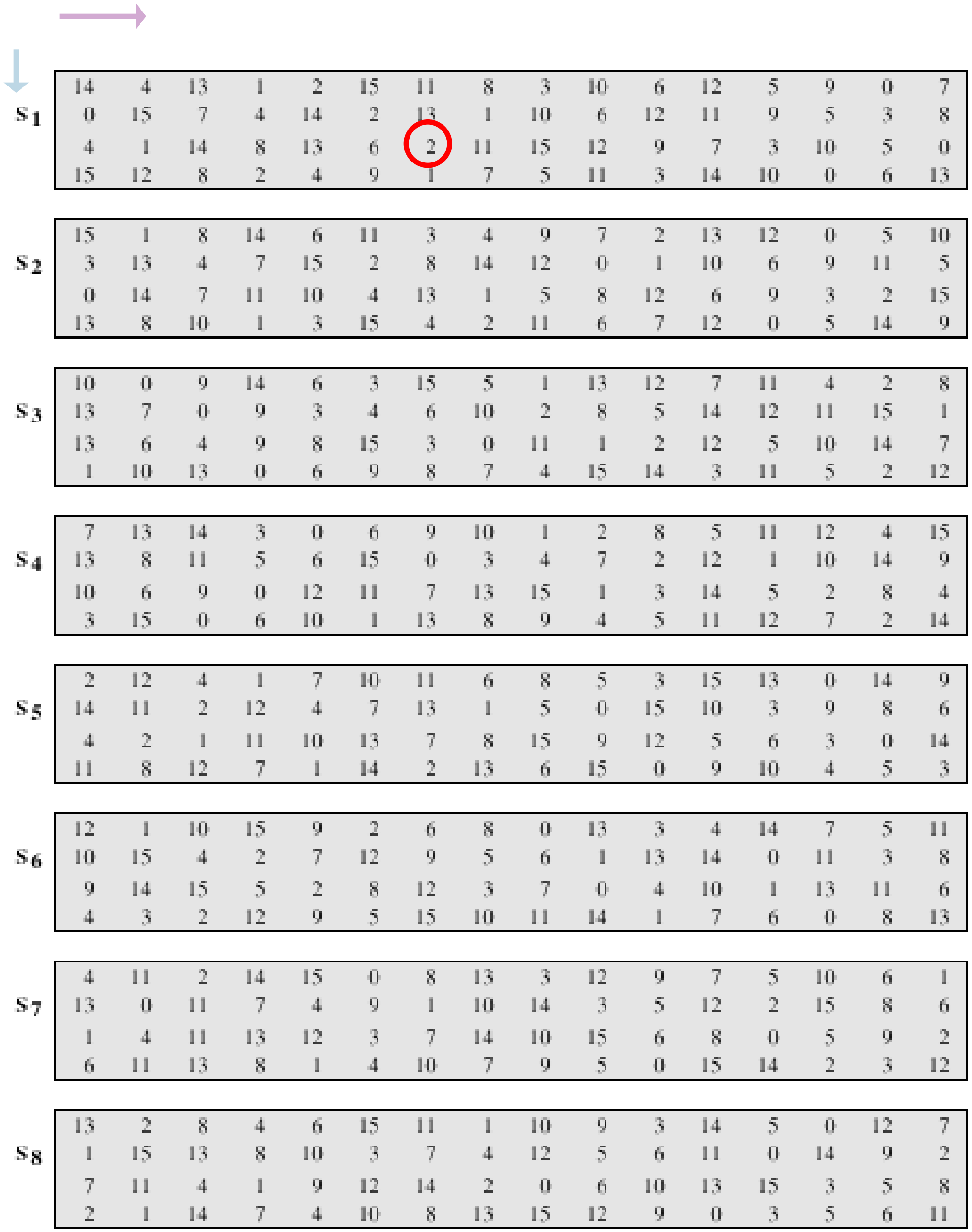


Row number: 10 (2)

Column number: 0110 (6)

Output: 0010 (2)

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

The input 56 bit key is first permuted according to permutation $PC-1$.

The result is then separated into two halves, denoted C_0 and D_0.

At each round C_{i-1} and D_{i-1} are separately circularly left shifted by 1 or 2 bits, according to the fixed schedule of left shifts.

Subsequently, the two halves are permuted according to $PC-2$ permutation table which produces 48 bit output that serves as the subkey K_i.

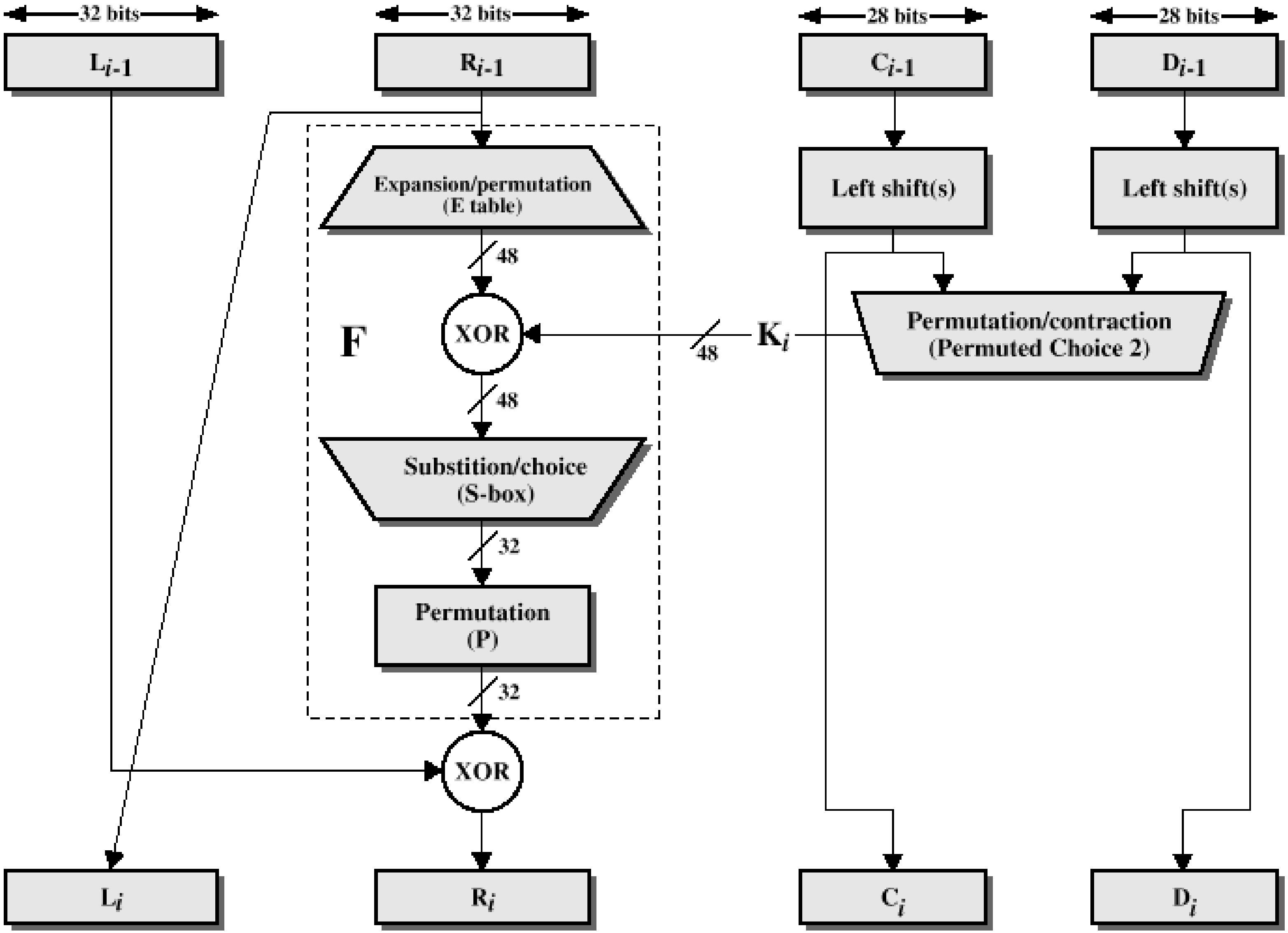


Figure 3.8 Single Round of DES Algorithm

Table 3.4 Tables Used for DES Key Schedule Calculation

(a) 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

(b) Permuted Choice Two (PC-2)

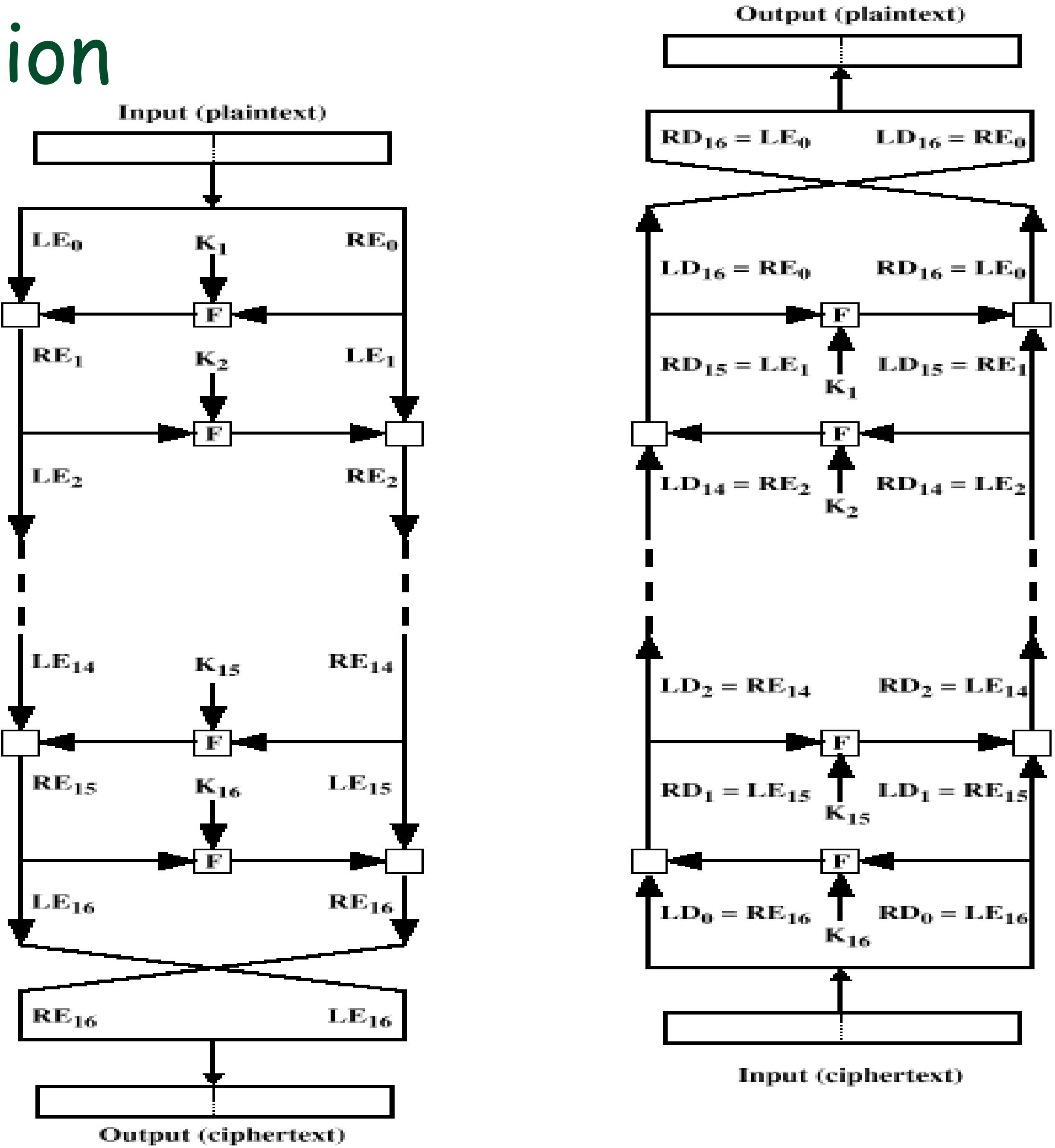
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

(c) 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

DES Decryption

As with any Feistel cipher, DES decryption is the same as encryption, the only difference being that the *subkeys are used in reverse order.*



The Avalanche Effect

It is desirable that a small change in either plaintext or key produces a significant change in the ciphertext; in particular, changing one input bit should result in changing approximately half of the output bits.

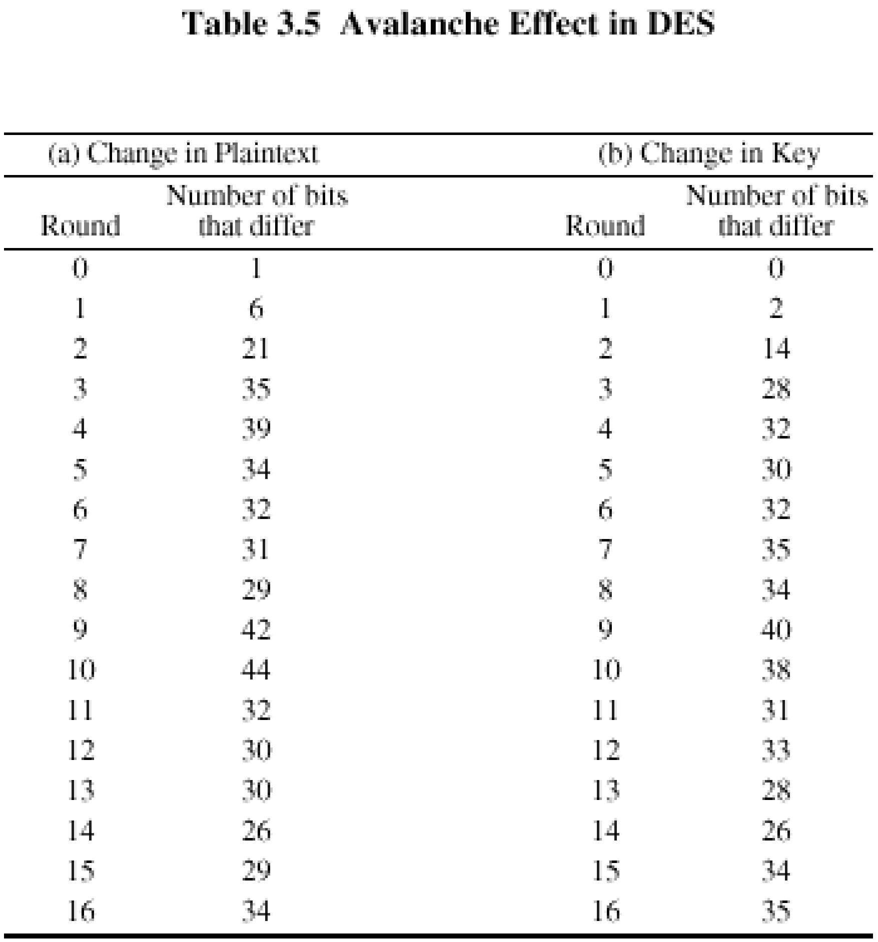


Table 3.5 Avalanche Effect in DES

(a) Change in Plaintext		(b) Change in Key	
Round	Number of bits that differ	Round	Number of bits that differ
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

The Completeness Effect

It is desirable that each output bit is a complex function of **all** the input bits.

Formally, a function f has a good **completeness** effect if

for each bit j, $0 \leq j < m$, in the ciphertext output vector,

there is at least one pair of plaintext vectors X and X_i which differ only in bit i, $0 \leq i < m$,

and for which $f(X)$ and $f(X_i)$ differ in bit j.

The Strength of DES

The main weakness of DES is its short key - it can be broken by brute-force attack on a specially designed machine in the matter of hours.

Another weak side are S-boxes and a possible trap-door built into them. The fact is, so far nobody discovered any such thing (actually, the fact is only that nobody publicly announced such a discovery).

However, DES proves to be cryptographically very strong, stronger than many systems developed afterwards.

It proves to be relatively resistant to differential cryptanalysis which was described in open literature in early 90s. Differential cryptanalysis looks at the difference of pairs of related plaintexts encrypted under the same key. With this attack DES can be broken with an effort 2^{47}, providing that 2^{47} chosen plaintexts are known.

An earlier version of DES, the so-called Lucifer that has 8 rounds requires only 256 chosen plaintexts, while 8 round DES requires 10^{14}.

The Strength of DES

As a comparison, earlier version of another cryptosystem, IDEA, proposed in 1990, uses 128 bit key, but only 2^{64} steps are required to break it with differential cryptanalysis attack.

Another attack, the so-called linear cryptanalysis can break DES given 2^{43} known plaintexts. Linear cryptanalysis tries to approximate the cipher by linear functions.

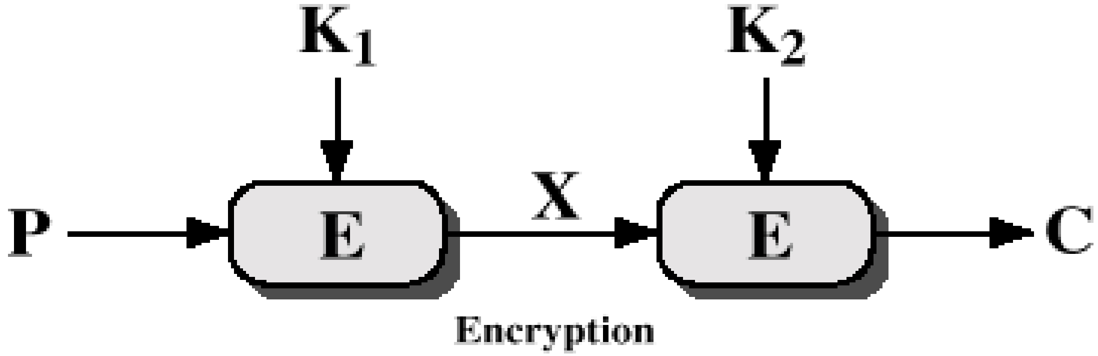
Although it may be easier to get this many known plaintexts than chosen plaintexts, this is only a minor improvement over differential cryptanalysis.

Double DES

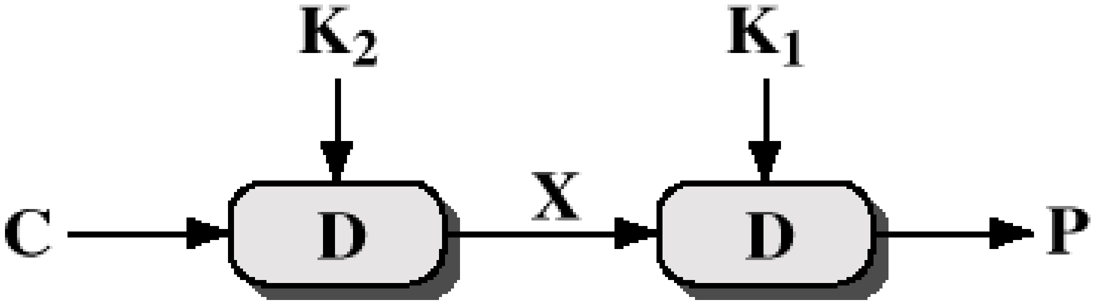
We have seen that, due to its short key, DES is vulnerable to a brute force attack. The simplest way to strengthen DES against this attack is to use multiple encryption with multiple keys.

Double DES works as follows: given a plaintext P and two encryption keys K_1 and K_2, ciphertext C is generated as $C = E_{K_2}[\,E_{K_1}[P]\,]$

For decryption, the keys are applied in reverse order: $P = D_{K_1}[D_{K_2}[C]\,]$



Encryption



Decryption

Double DES

It appears that double DES involves a key length of $56 \times 2 = 112$ bits and that it is thus cryptographically much stronger than ordinary DES.

Is that really the case? Is it possible to reduce double DES to a single stage, that is, is it possible to find a key K_3 such that

$$E_{K_2}[E_{K_1}[P]] = E_{K_3}[P] ?$$

If this is the case, double DES can be reduced to a single DES with 56 bit key.

It was proved in 1992 (Campbell, Wiener) that this is NOT the case.

Intuitive argument: DES encryption can be seen as a mapping of 64 bit input block (plaintext) into a 64 bit output block (ciphertext). There are $(2^{64})!$ different mappings, and there are only 2^{56} different mappings in DES (one for each key). It is, therefore, reasonable to assume that double DES corresponds to one of the many mappings that are not produced by single DES.

Double DES

Bad news is that double DES is not doing much better against the meet-in-the-middle attack than single DES against the brute-force attack.

Meet-in-the -middle attack was first described by Diffie and Hellman in 1977.

Consider the intermediate result X, between the two encryptions in double DES and observe that

$$X = E_{K_1}[P] = D_{K_2}[C]$$

Given a known plaintext-ciphertext pair (P, C), encrypt P for all 2^{56} possible values of key K_1 and store results in a table sorted by the value of X.

Decrypt C using all 2^{56} possible values of K_2 and after each decryption check the result against the table. If there is a match, test the two keys against a new plaintext-ciphertext pair.

Thus the effort for a successful known plaintext attack against double DES is 2^{56}, which is not much better than 2^{55} required for the single DES.

Triple DES

Meet-in-the-middle attack can be prevented by using a triple DES - three stages of encryption with three different keys.

Here the cost of meet-in-the-middle attack is 2^{112}, which is still not practical.

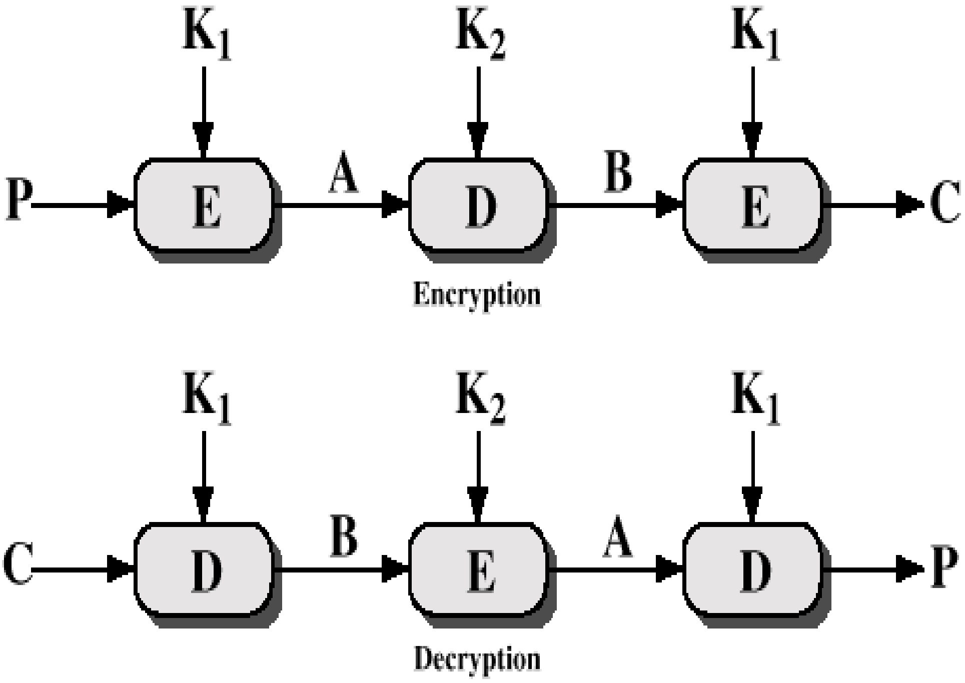
The problem with this scheme is that it requires a very long key: $56 \times 3 = 168$ bits.

Tuchman (1979) proposed the following triple encryption that uses only two keys:

$$C = E_{K_1}[D_{K_2}[E_{K_1}[P]\,]\,]$$

The only significance of using decryption as a second stage is that it allows triple DES to decrypt data encrypted by single DES:

$$C = E_{K_1}[D_{K_1}[E_{K_1}[P]\,]\,] = E_{K_1}[P]$$

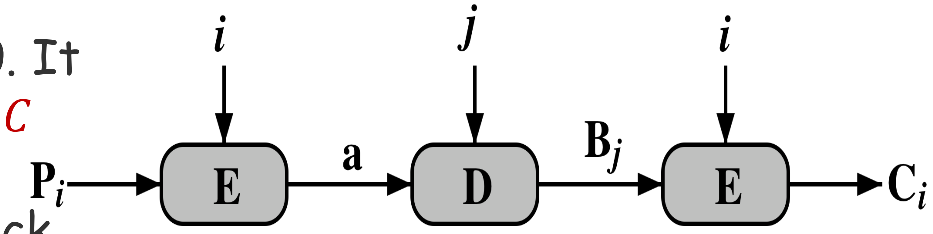


(b) Triple Encryption

Triple DES

There is no known practical attack on triple DES.

The following is a known-plaintext attack described by Oorschot and Wiener in 1990. It is based on the fact that if we know a and C from the figure on the right, then the attack reduces to meet-in-the-middle attack on double DES. We shall choose a potential value for a and try to find the corresponding pair (P, C).

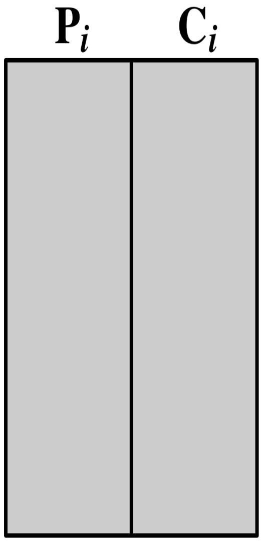


(a) Two-key Triple Encryption with Candidate Pair of Keys

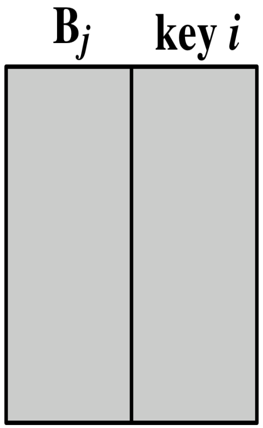
Obtain n known plaintext-ciphertext pairs (P, C). Place them in a table sorted on the values of P.

Pick an arbitrary value a. For each of the possible 2^{56} keys $K_1 = i$, calculate the plaintext value P_i that produces a: $P_i = D_i\ [a]$.

For each P_i that matches an entry in the first table create an entry in the second table consisting of the K_1 and $B = D_i[C]$



(b) Table of n known plaintext-ciphertext pairs, sorted on P



(c) Table of intermediate values and candidate keys

Triple DES

For each of the possible 2^{56} possible keys $K_2 = j$, calculate $B_j = D_j[a]$ and check if there is a match in table 2, in which case (i, j) is a candidate pair of keys.

Test each candidate pair of keys on a few plaintext-ciphertext pairs; if no pair succeeds, repeat from step 1 with a new value of a.

The expected running time of this attack is $2^{120-\log n}$

The following is a version of triple DES with three keys:

$$C = E_{K_3}[D_{K_2}[E_{K_1}[P]]]$$

It is still possible to use this algorithm to decrypt ciphertext encrypted by single DES by putting $K_3 = K_2$ or $K_1 = K_2$.

Triple DES with three keys is adopted by many internet applications.

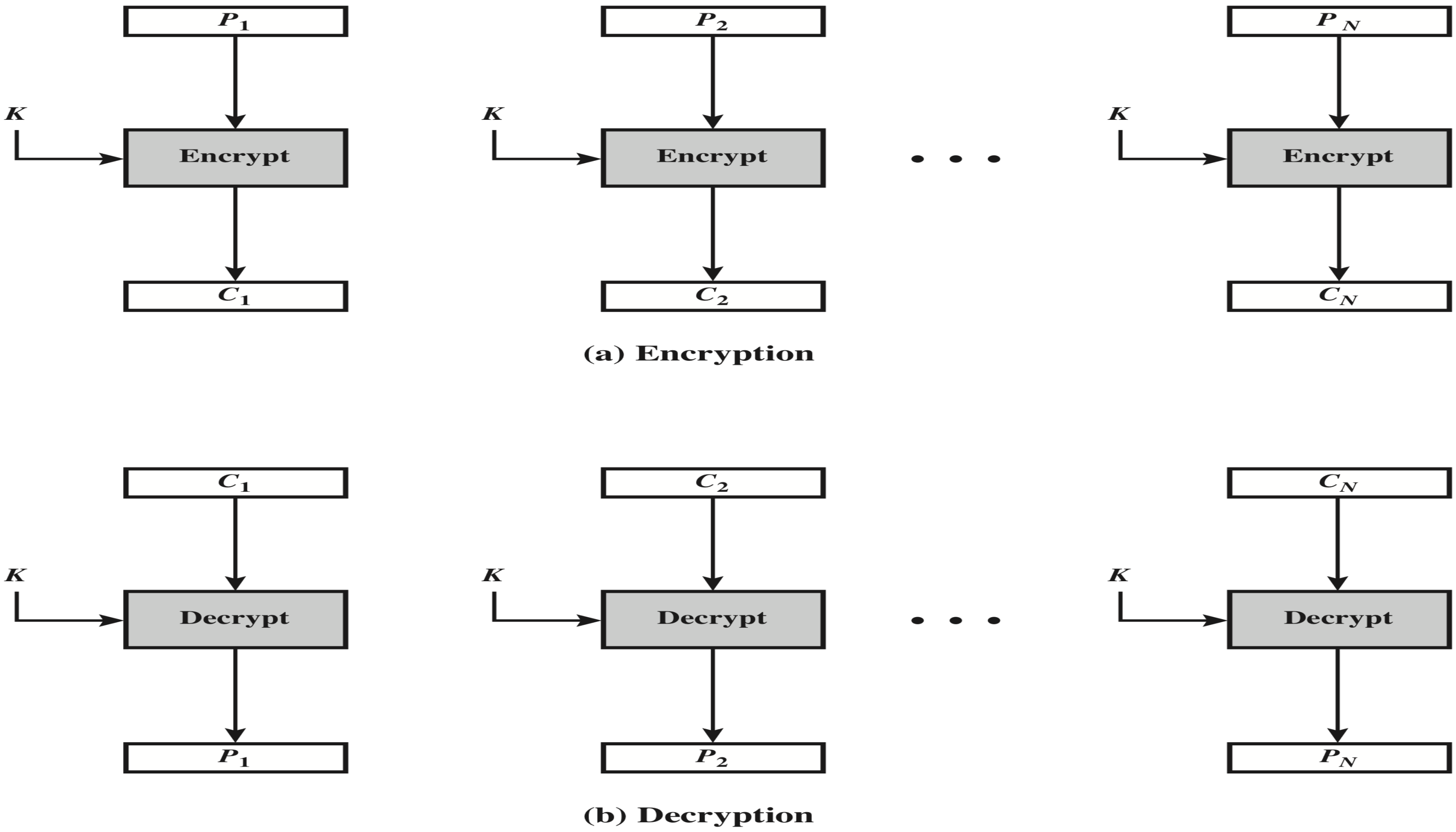
Block Ciphers Modes of Operation

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

Electronic Codebook Mode (ECB)

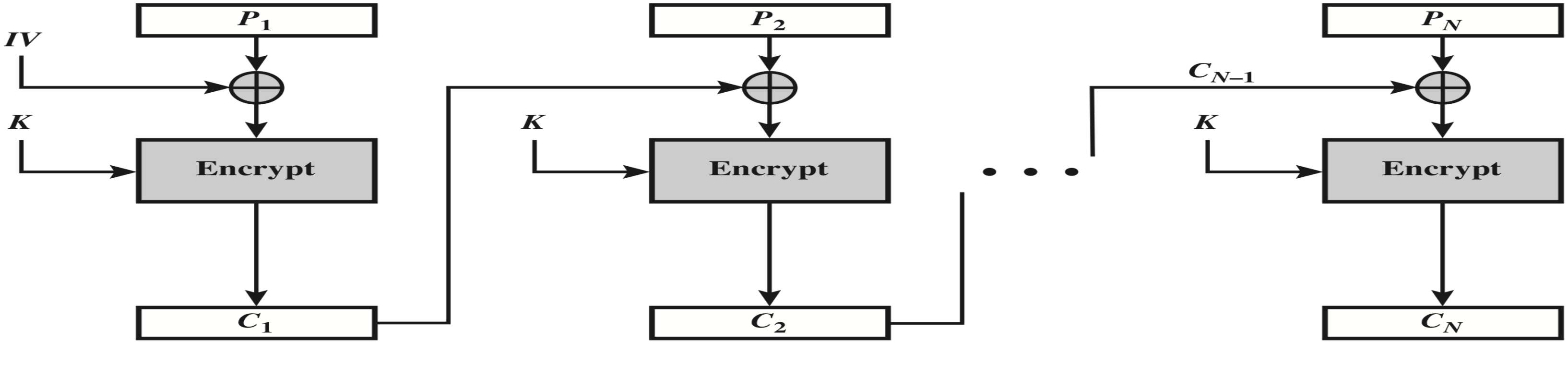
Each block of the plaintext is encrypted using the same key.

ECB mode is good for short messages, but it should be avoided for lengthy messages. Therefore, it can be used to send encryption keys.

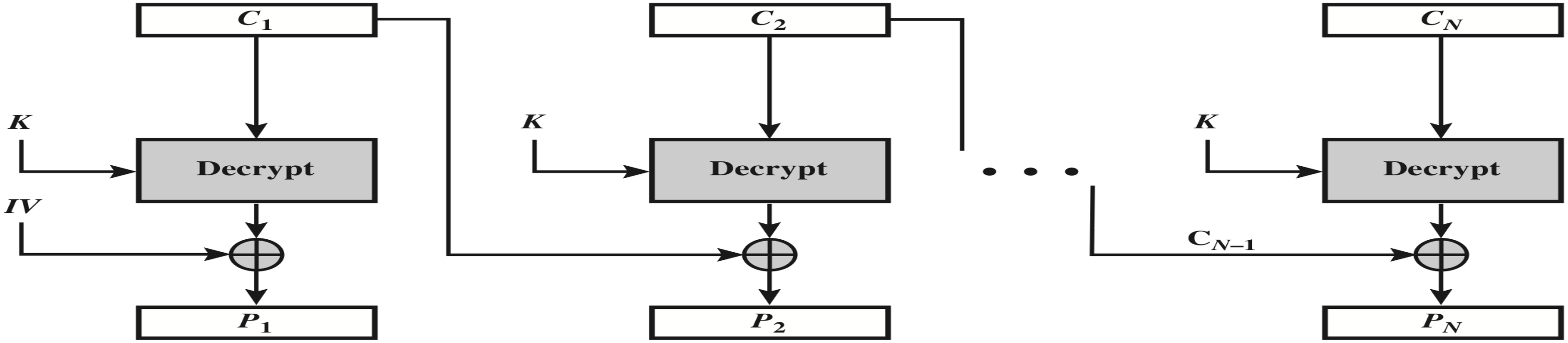


Cipher Block Chaining Mode (CBC)

The input is XOR of the current plaintext block and the preceding ciphertext block; the first plaintext block is XORed with the initialisation vector (IV). The IV should be know to both sender and receiver but not to an intruder. (Note that ECB is good enough for sending IV).



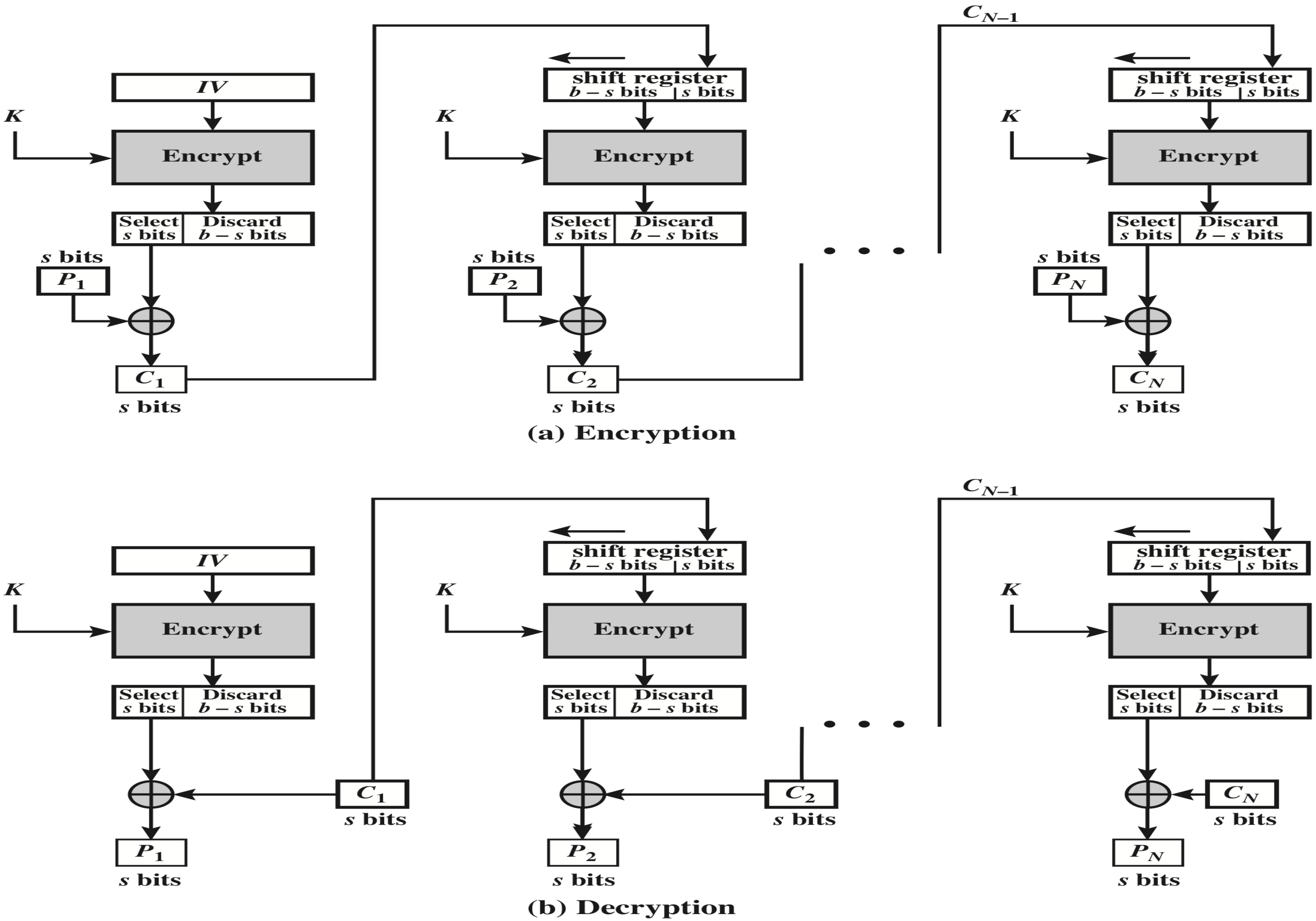
(a) Encryption



(b) Decryption

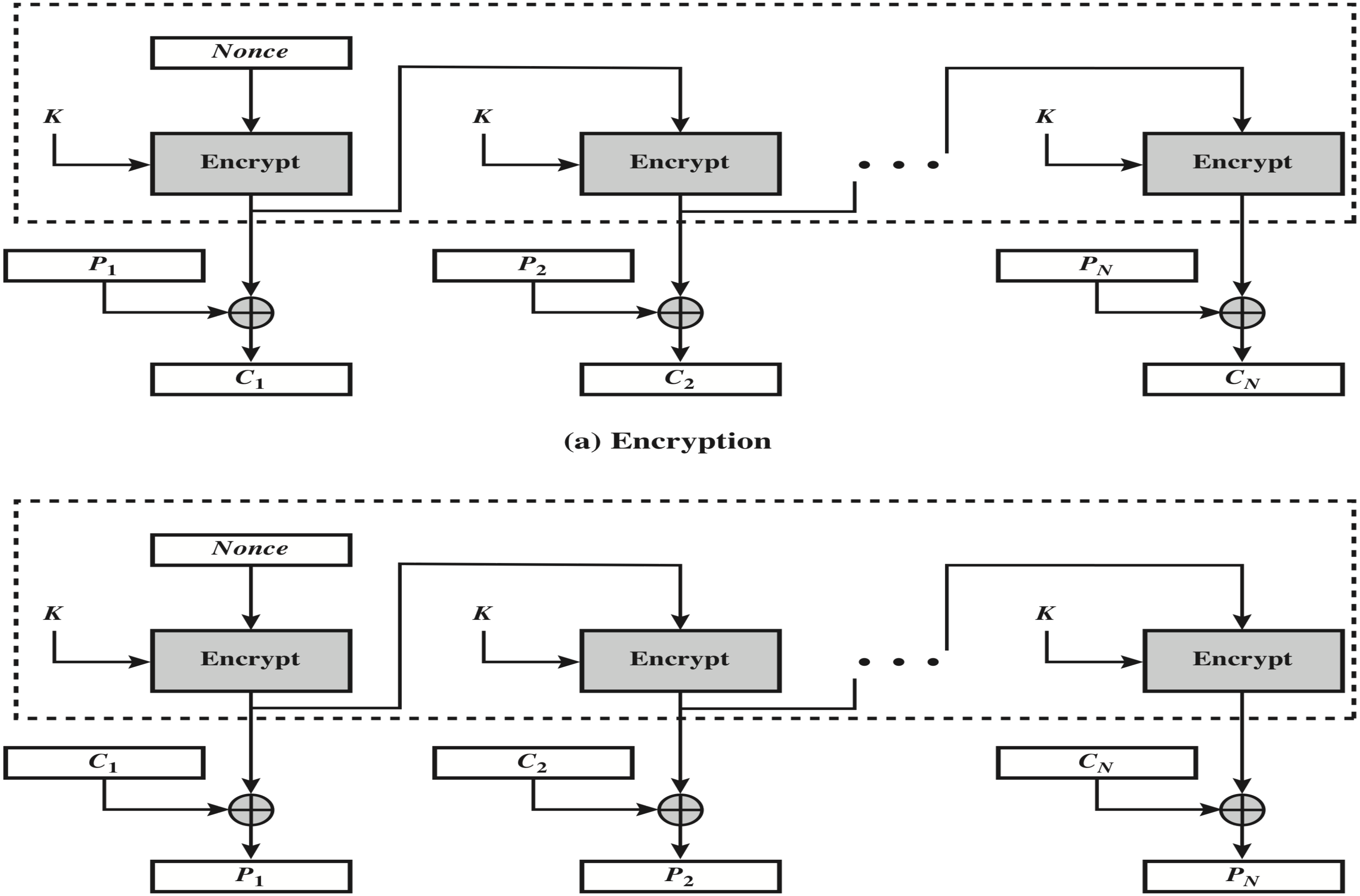
Cipher Feedback Mode (CFB)

CFB effectively converts DES into a self-synchronising stream cipher; here data is encrypted in units smaller that the block size.



Output Feedback Mode (OFB)

OFB effectively converts DES into a synchronous stream cipher.

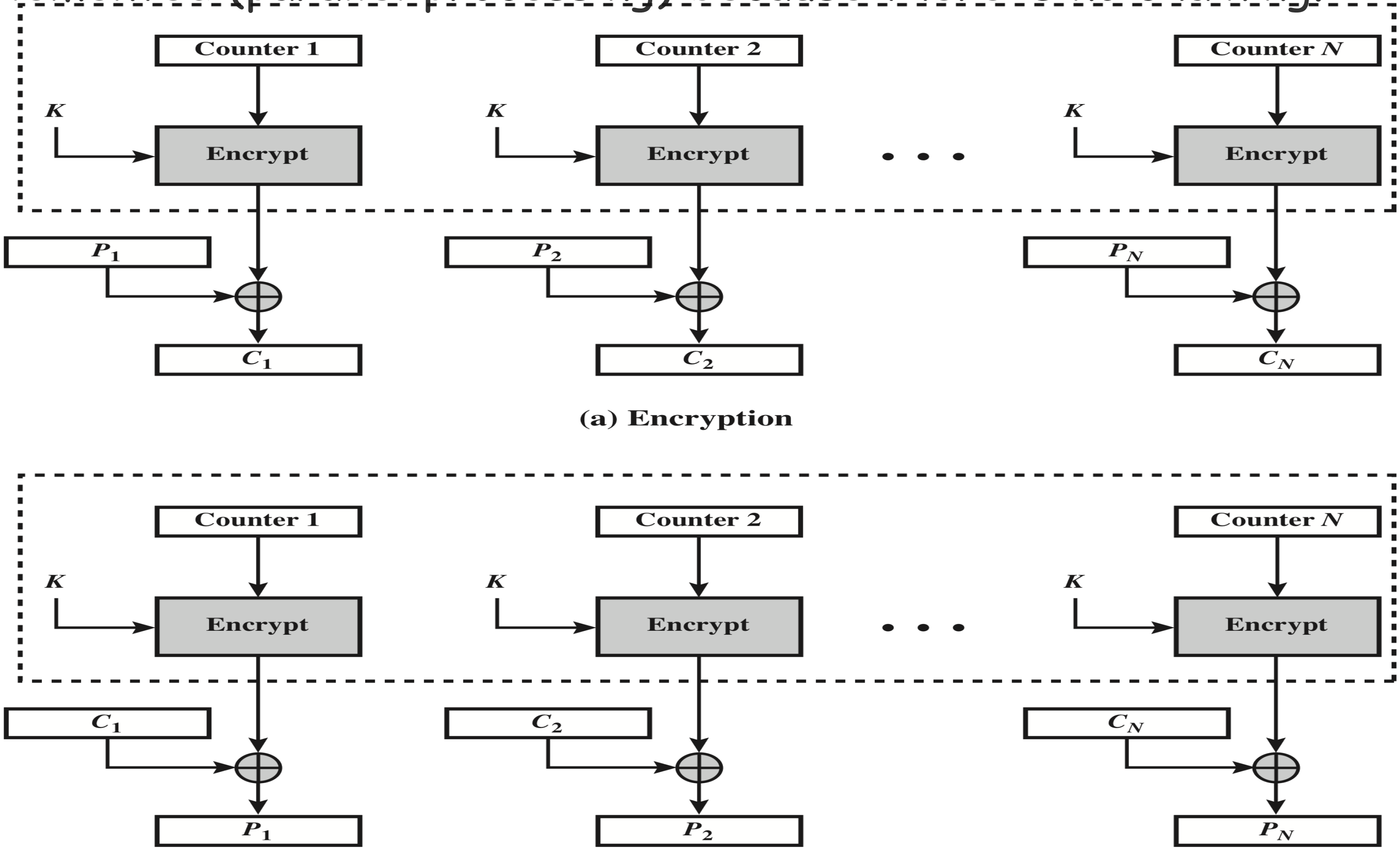


(a) Encryption

(b) Decryption

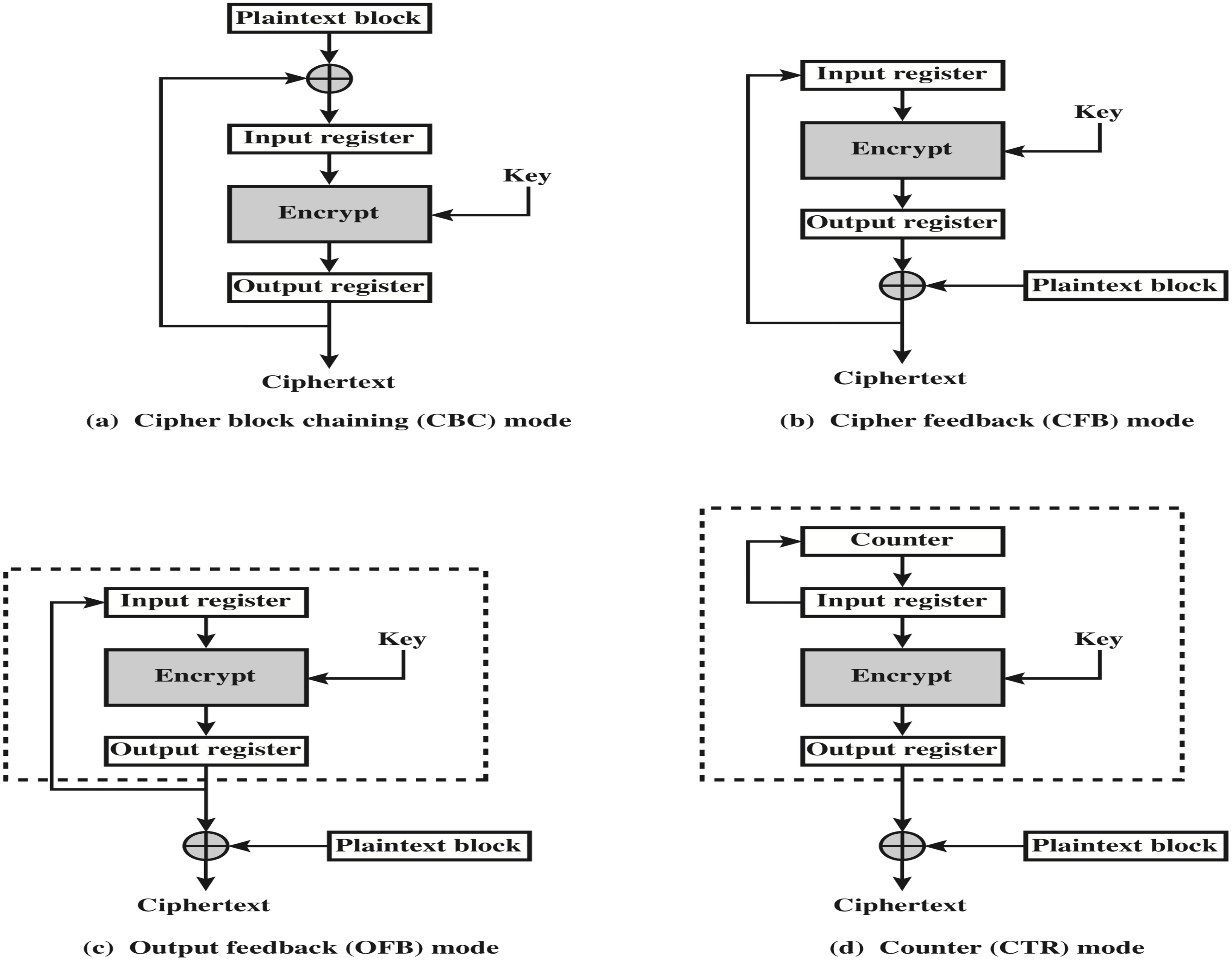
Counter (CTR) Mode

Proposed early on, but not as popular until applied to ATM network security and IP security. It uses a counter, equal in size to the plaintext. This is encrypted with key, and then XORed with plaintext. For subsequent blocks, the counter is incremented. It can be efficiently implemented (parallel processing) because there is no chaining.



(a) Encryption

(b) Decryption



(a) Cipher block chaining (CBC) mode

(b) Cipher feedback (CFB) mode

(c) Output feedback (OFB) mode

(d) Counter (CTR) mode

Figure 6.8 Feedback Characteristic of Modes of Operation

Next week:

1. Origins of AES
    a) AES Requirements
    b) AES Evaluation Criteria
    c) AES Shortlist
    d) Final NIST Evaluation of Rijndael
2. Rijndael
    a) Overall Structure
    b) Details of a single round
    c) Key Expansion
    d) AES Decryption
    e) Implementation Issues
3. AES as polynomial arithmetic with coefficients in GF(2^8)

Text Chapter 6, Advanced Encryption Standard

References

1. D. Denning. “Cryptography and Data Security”, Addison Wesley, 1982.

2. W. Stallings. “Cryptography and Network Security”, 6th Edition, Pearson Education, 2014.