Conventional Encryption Principles

- An encryption scheme has five ingredients:
  - Plaintext
  - Encryption algorithm
  - Secret Key
  - Ciphertext
  - Decryption algorithm
- Security depends on the secrecy of the key, not the secrecy of the algorithm

Conventional Encryption Principles

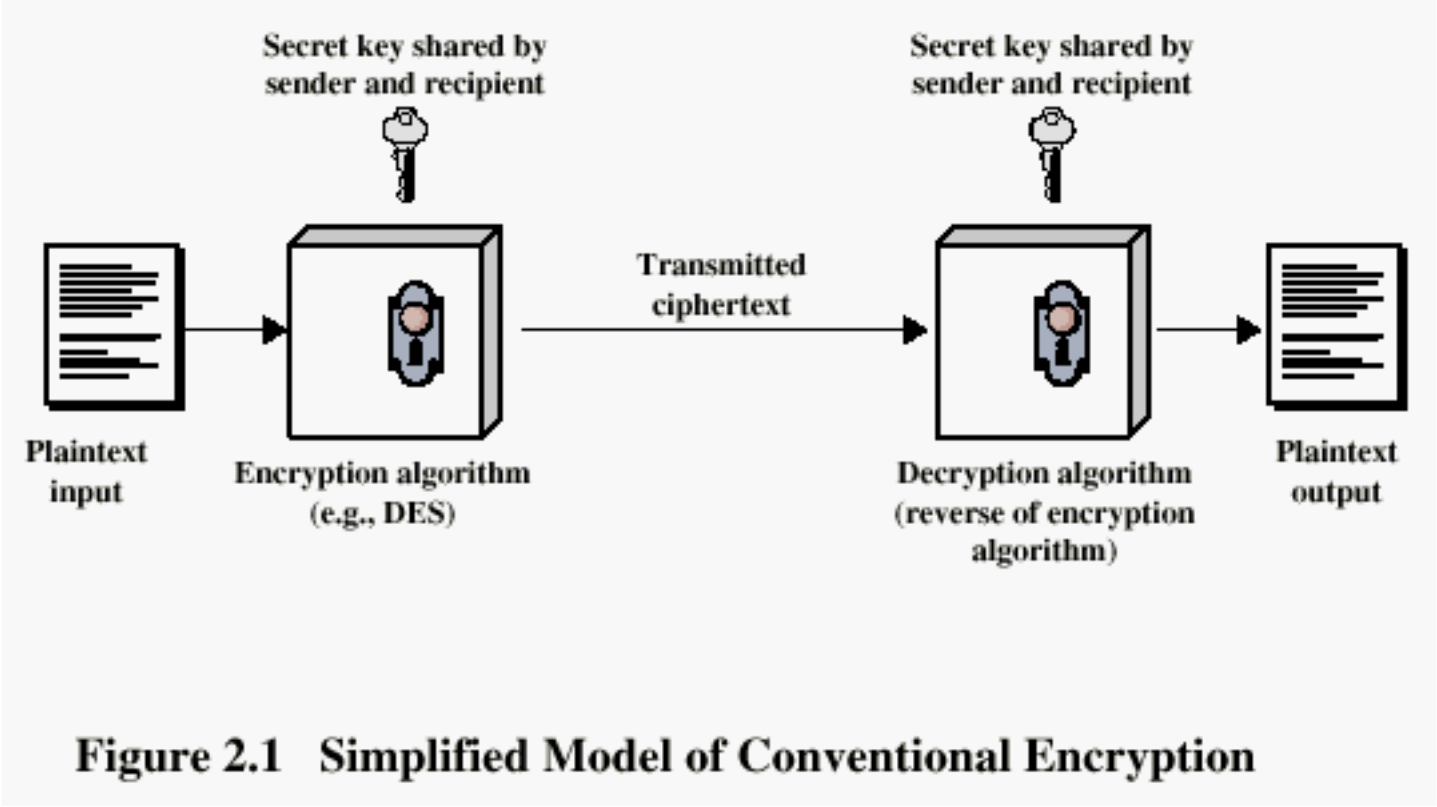


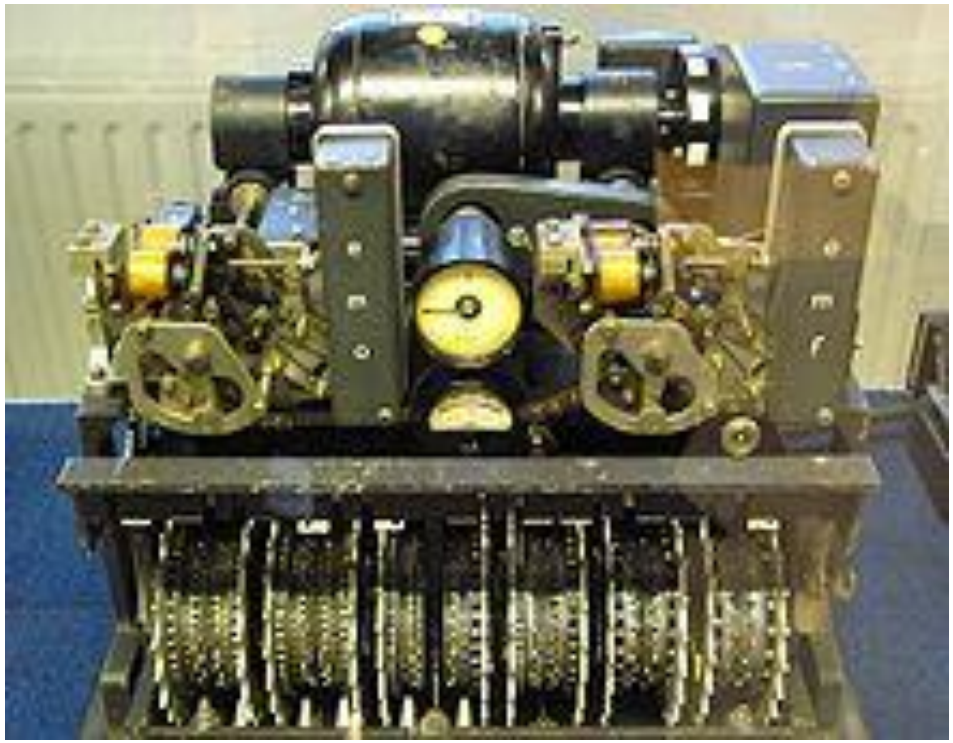
Figure 2.1 Simplified Model of Conventional Encryption

The 3 CRYPTs

- Cryptology= Cryptography + Cryptanalysis
- Cryptography= Deals with design of algorithms for encryption and decryption, intended to ensure the secrecy and/or authenticity of messages.
- Cryptanalysis= Deals with breaking of a cipher (algorithm) to recover information, or forging encrypted information that will be accepted as authentic.

Cryptography

- Classified along three independent dimensions:
  1. The type of operations used for transforming plaintext to ciphertext
  2. The number of keys used
     a) symmetric (single key)
     b) asymmetric (two-keys, or public-key encryption)
  3. The way in which the plaintext is processed



German Lorenz cipher machine, used in World War II to encrypt very-high-level general staff messages

Cryptanalysis

- The process of attempting to discover the plaintext or key is known as cryptanalysis.
- The strategy used by the cryptanalyst depends on nature of the encryption scheme and the information available to the cryptanalyst.

Table 2.1 Types of Attacks on Encrypted Messages

Type of Attack	Known to Cryptanalyst
Ciphertext only	•Encryption algorithm •Ciphertext to be decoded
Known plaintext	•Encryption algorithm •Ciphertext to be decoded •One or more plaintext-ciphertext pairs formed with the secret key
Chosen plaintext	•Encryption algorithm •Ciphertext to be decoded •Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	•Encryption algorithm •Ciphertext to be decoded •Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	•Encryption algorithm •Ciphertext to be decoded •Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key •Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

Average time required for exhaustive key search

Key Size (bits)	Number of Alternative Keys	Time required at 10^6 Decryption/μs
32	$2^{32} = 4.3 \times 10^9$	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	10 hours
128	$2^{128} = 3.4 \times 10^{38}$	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	5.9×10^{30} years

Feistel Cipher Structure

- Virtually all conventional block encryption algorithms, including DES have a structure first described by Horst Feistel of IBM in 1973
- The realization of a Fesitel Network depends on the choice of the following parameters and design features (see next slide):

Feistel Cipher Structure

- **Block size:** larger block sizes mean greater security
- **Key Size:** larger key size means greater security
- **Number of rounds:** multiple rounds offer increasing security
- **Subkey generation algorithm:** greater complexity will lead to greater difficulty of cryptanalysis.
- **Fast software encryption/decryption:** the speed of execution of the algorithm becomes a concern

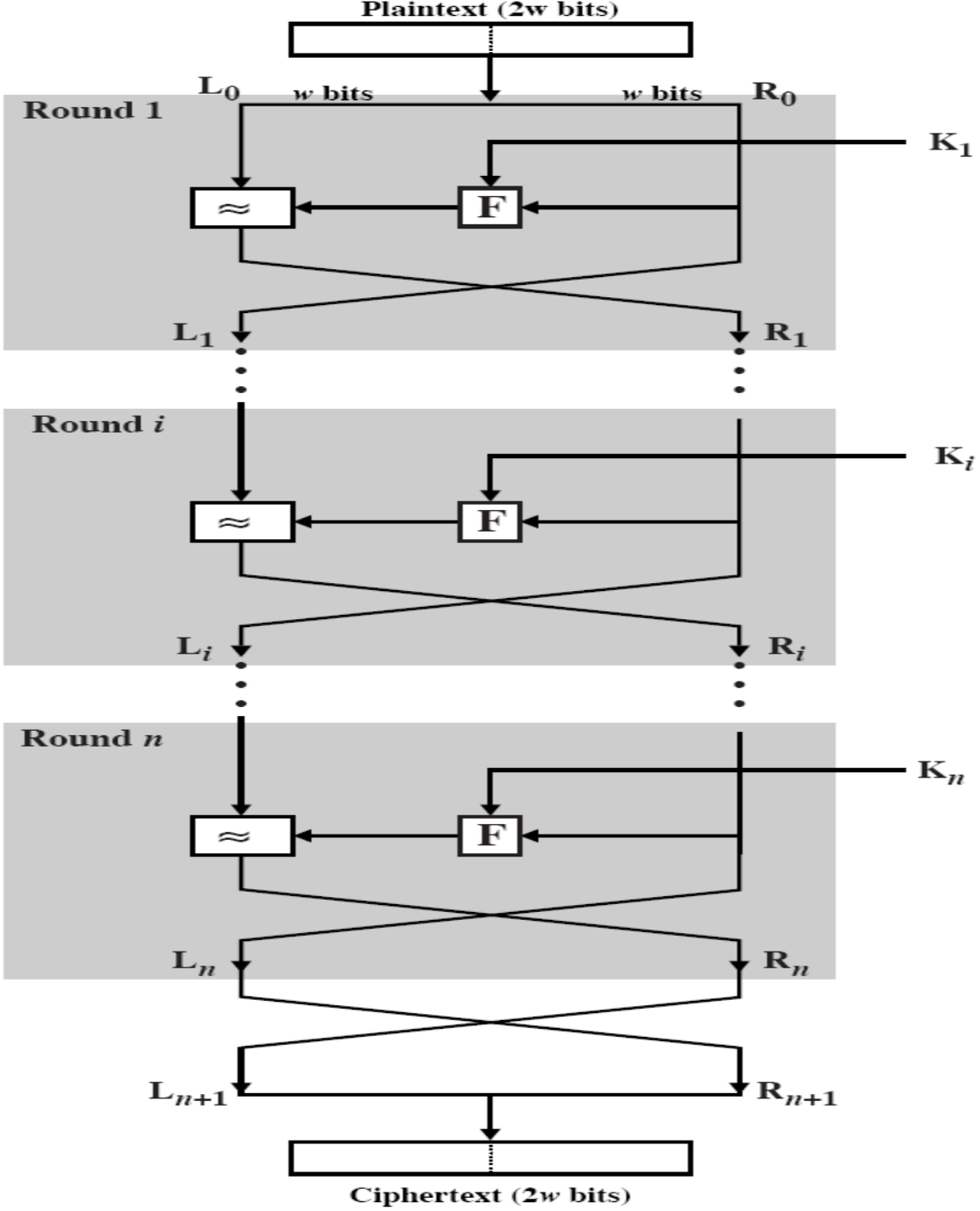


Figure 3.5 Classical Feistel Network

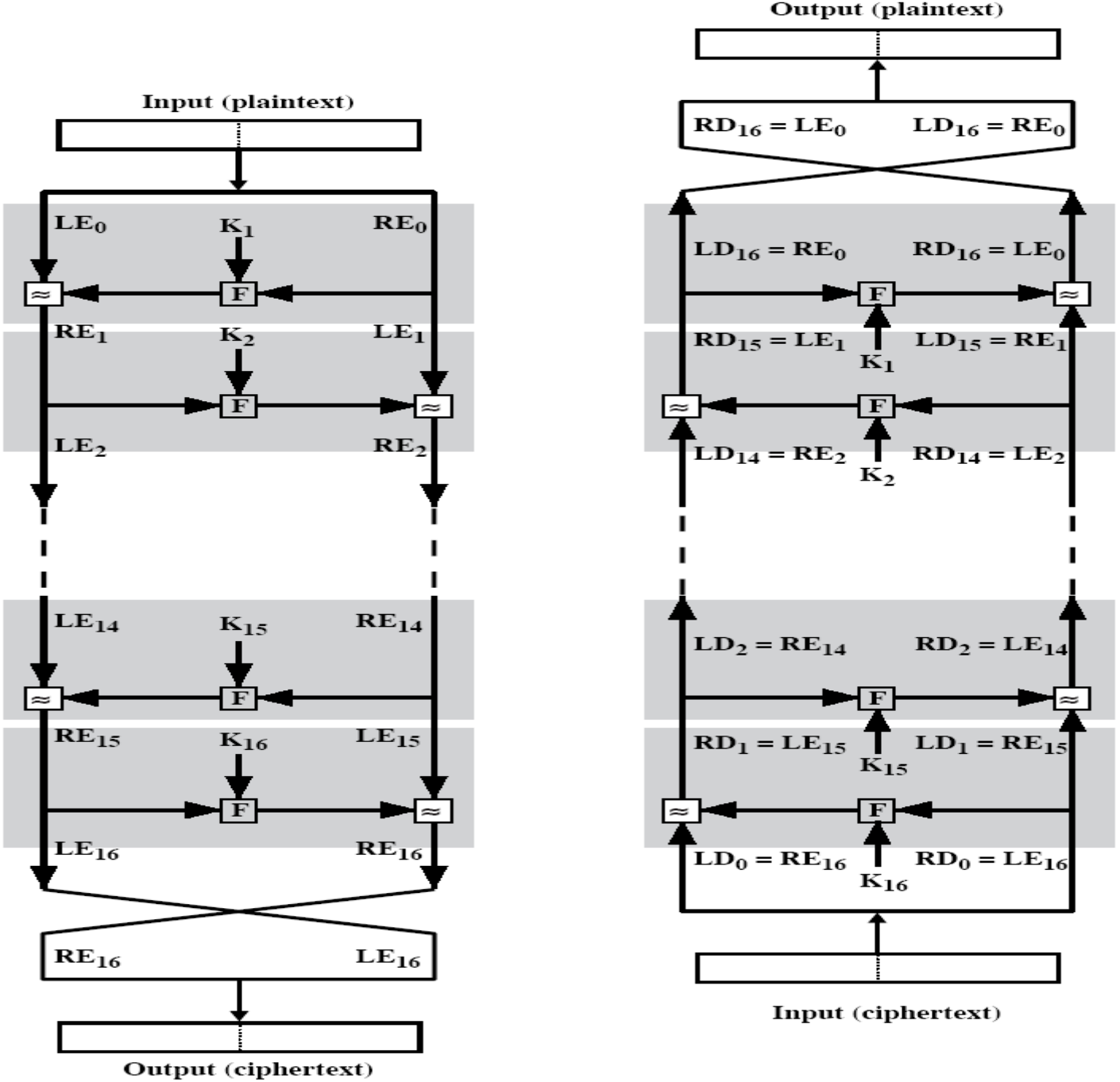


Figure 3.6 Feistel Encryption and Decryption

Simplified DES(S-DES)

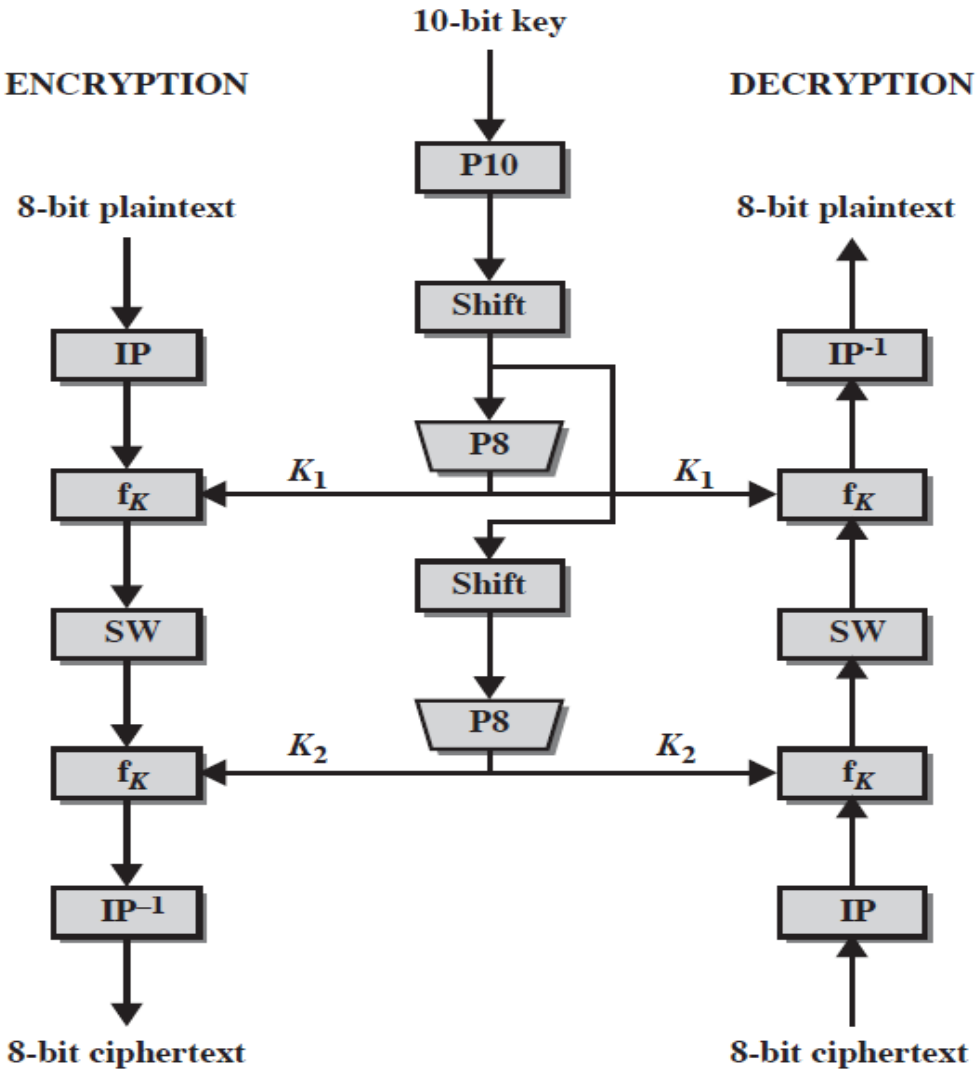


Figure G.1 Simplified DES Scheme

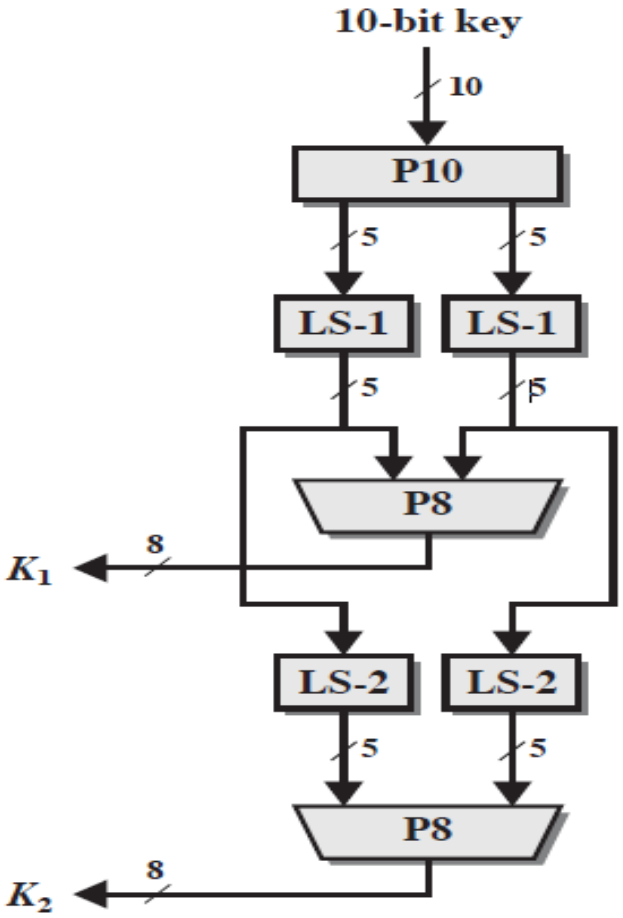


Figure G.2 Key Generation for Simplified DES

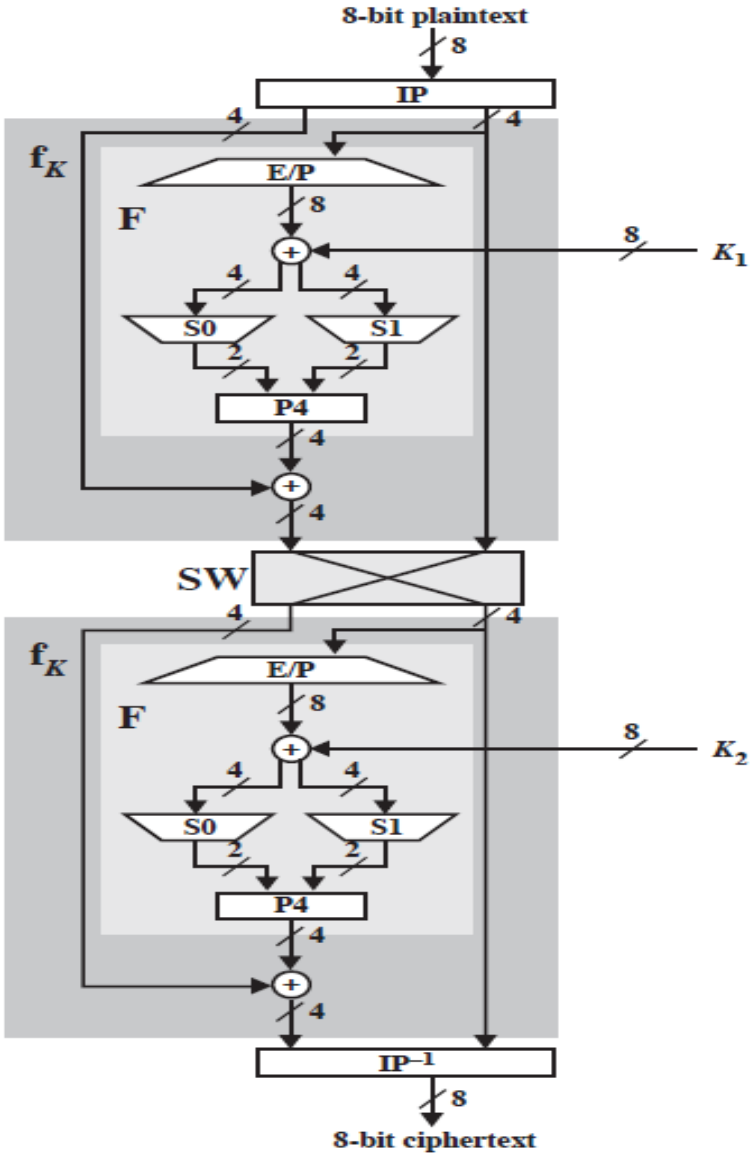
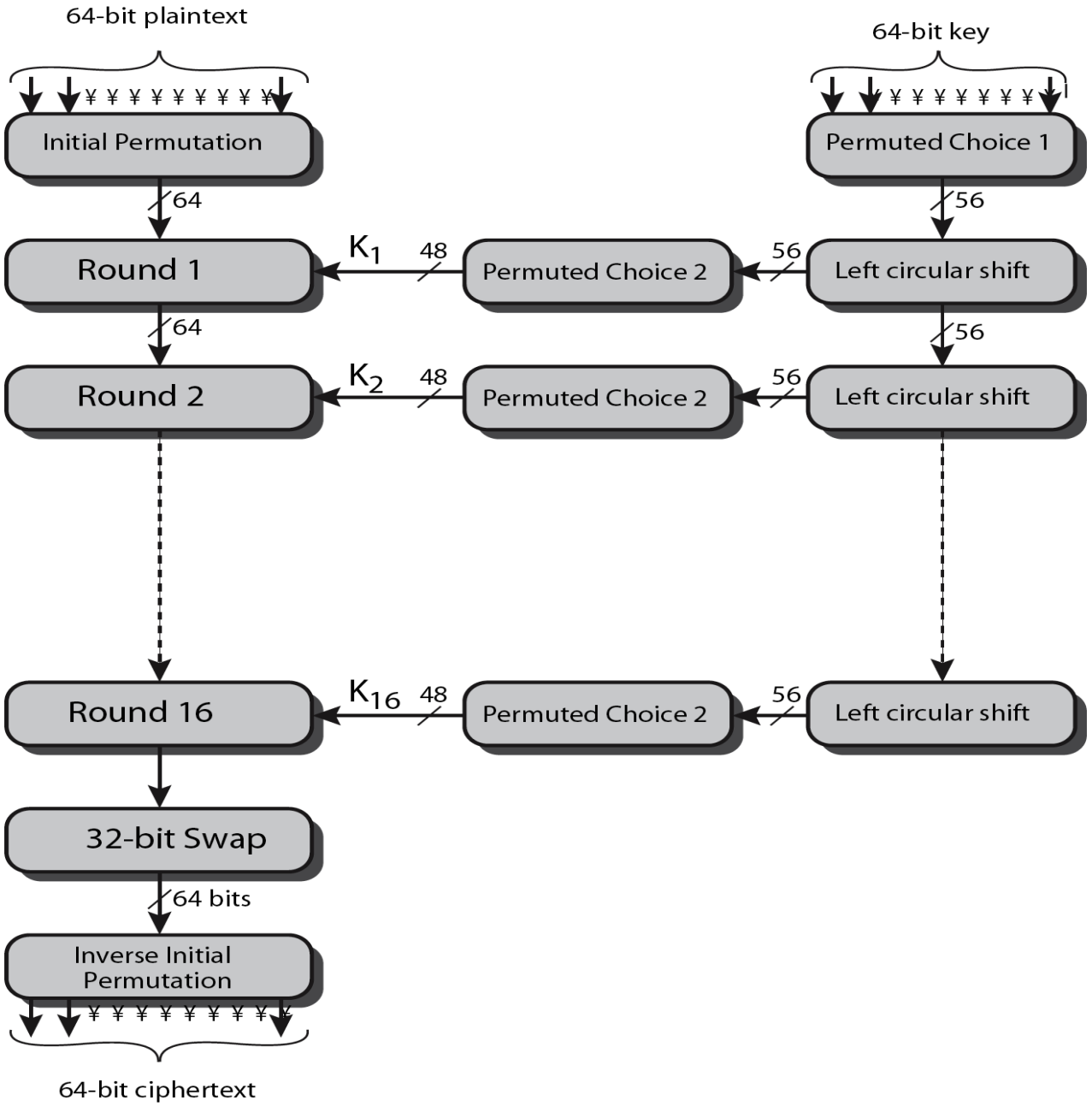


Figure G.3 Simplified DES Encryption Detail

Conventional Encryption Algorithms

- Data Encryption Standard (DES)
  - The most widely used encryption scheme
  - The algorithm is reffered to the Data Encryption Algorithm (DEA)
  - DES is a block cipher
  - The plaintext is processed in 64-bit blocks
  - The key is 56-bits in length

DES Encryption Overview



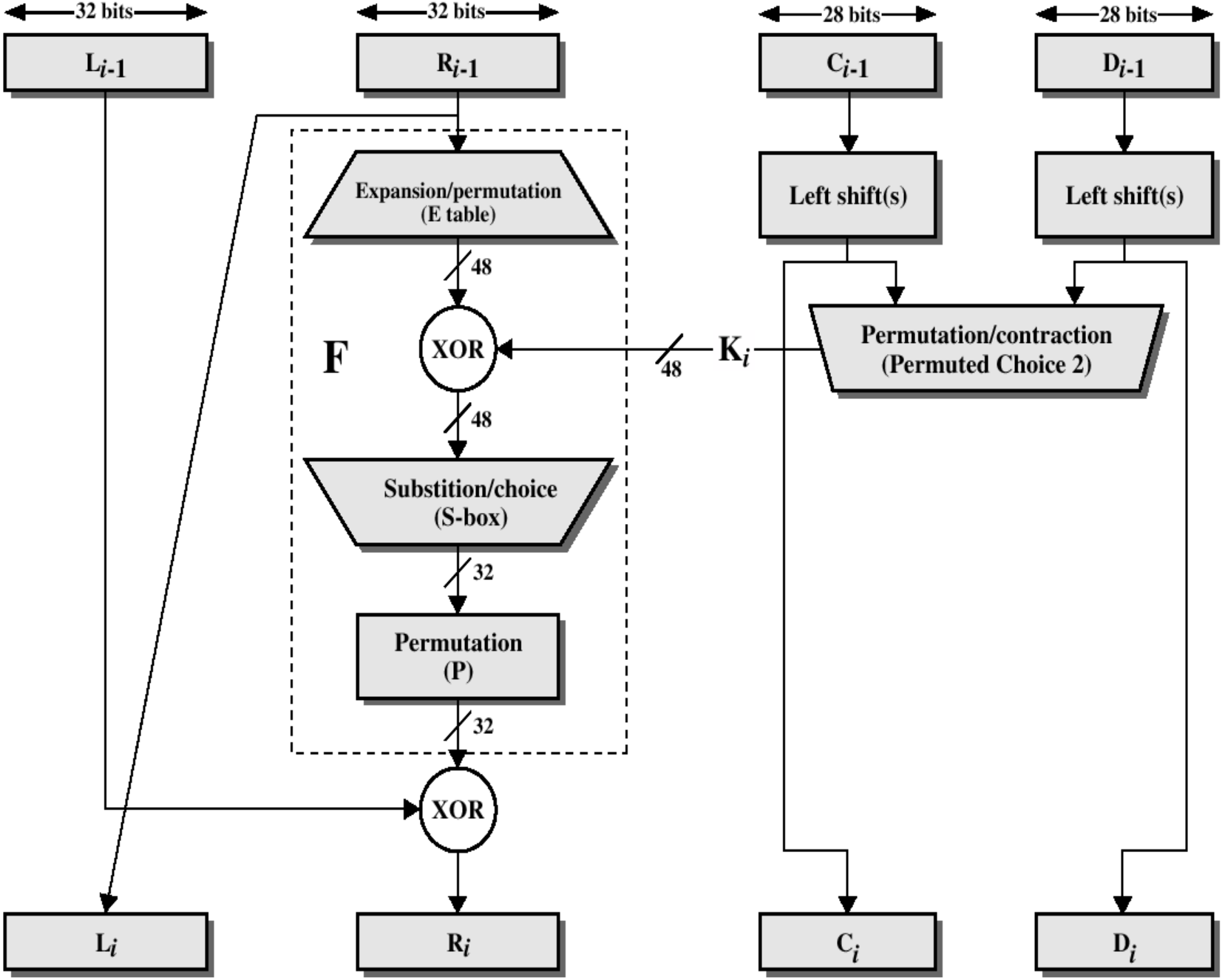


Figure 2.4 Single Round of DES Algorithm

DES Round Structure

- uses two 32-bit L & R halves
- as for any Feistel cipher can describe as:

  $L_i = R_{i-1}$

  $R_i = L_{i-1} \oplus \mathrm{F}(R_{i-1}, K_i)$
- F takes 32-bit R half and 48-bit subkey:
  - expands R to 48-bits using perm E
  - adds to subkey using XOR
  - passes through 8 S-boxes to get 32-bit result
  - finally permutes using 32-bit perm P

Encryption (IP, IP^{-1})

- IP

Bit	0	1	2	3	4	5	6	7
1	58	50	42	34	26	18	10	2
9	60	52	44	36	28	20	12	4
17	62	54	46	38	30	22	14	6
25	64	56	48	40	32	24	16	8
33	57	49	41	33	25	17	9	1
41	59	51	43	35	27	19	11	3
49	61	53	45	37	29	21	13	5
57	63	55	47	39	31	23	15	7

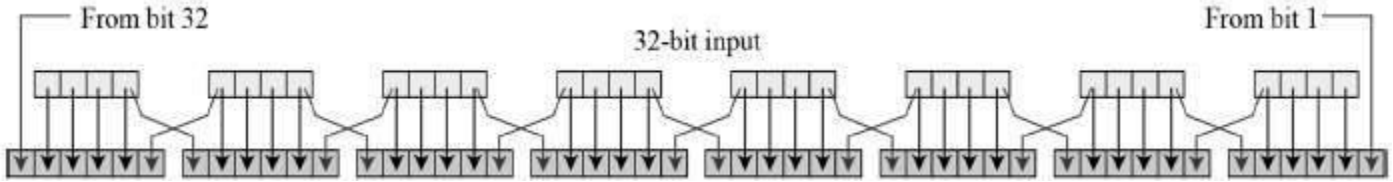
IP-1

Bit	0	1	2	3	4	5	6	7
1	40	8	48	16	56	24	64	32
9	39	7	47	15	55	23	63	31
17	38	6	46	14	54	22	62	30
25	37	5	45	13	53	21	61	29
33	36	4	44	12	52	20	60	28
41	35	3	43	11	51	19	59	27
49	34	2	42	10	50	18	58	26
57	33	1	41	9	49	17	57	25

Note: IP(IP-1) = IP-1(IP) = I

Expansion /Permutation box

Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits.
Permutation logic is graphically depicted in the following illustration −



The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown −

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

Encryption (Round) (cont.)

- E

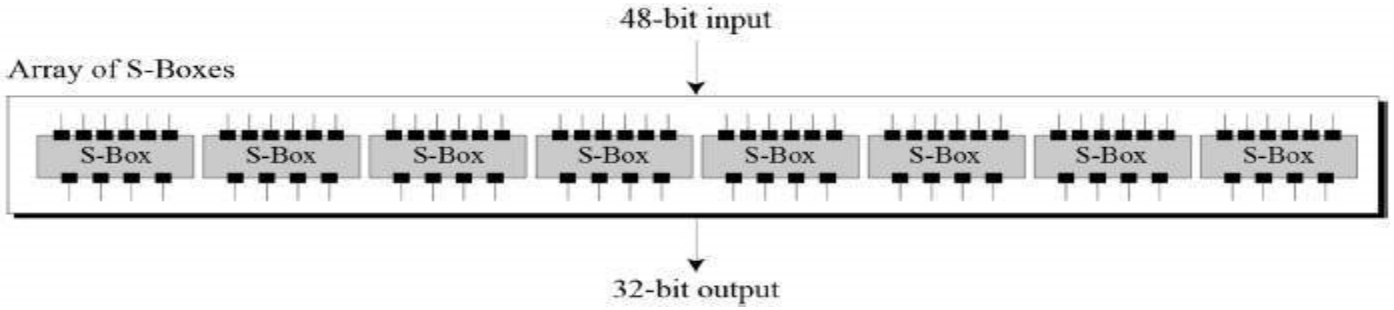
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	45	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

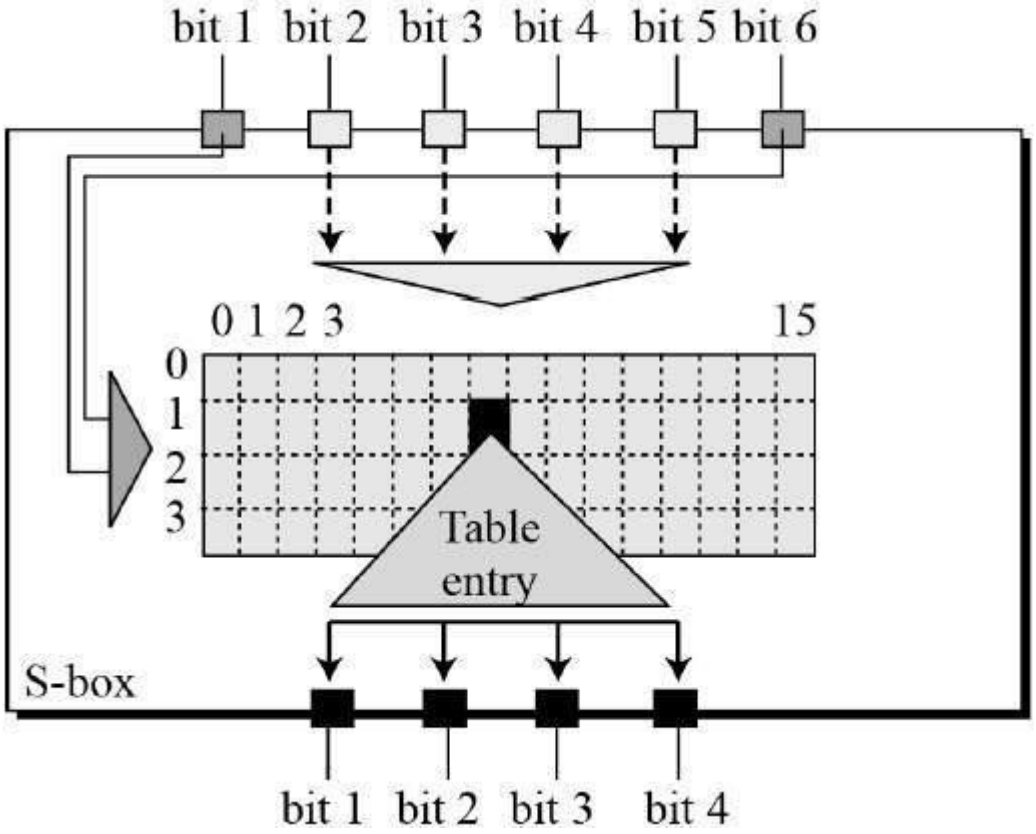
Expansion

- 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
9	13	30	6	22	11	4	25

Expansion


48-bit input
Array of S-Boxes
S-Box
S-Box
S-Box
S-Box
S-Box
S-Box
S-Box
S-Box
32-bit output


bit 1 bit 2 bit 3 bit 4 bit 5 bit 6
0 1 2 3
15
0
1
2
3
Table entry
S-box
bit 1 bit 2 bit 3 bit 4

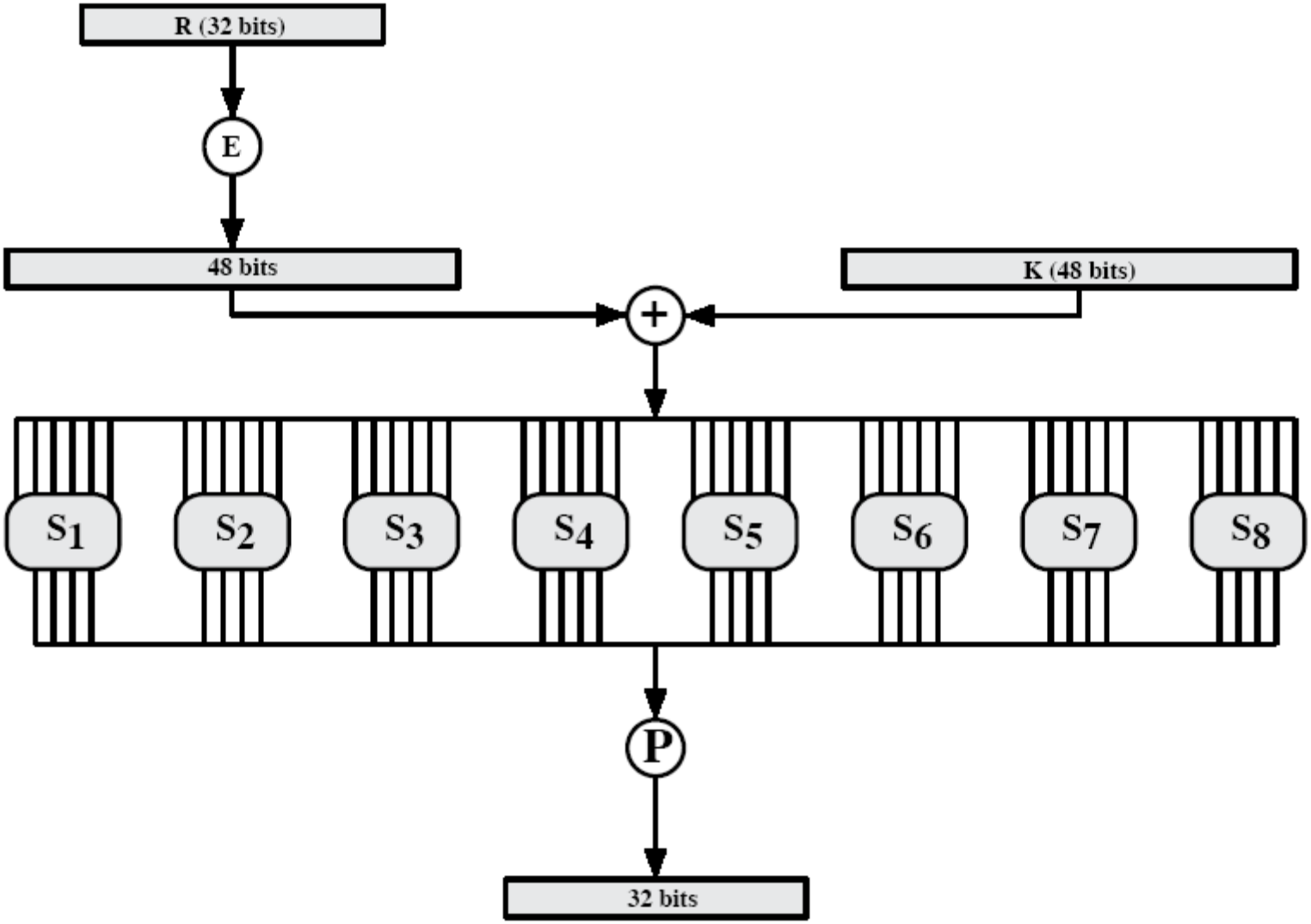


Figure 3.9 Calculation of F(R, K)

Encryption (Round) (cont.)

- S-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

[1]

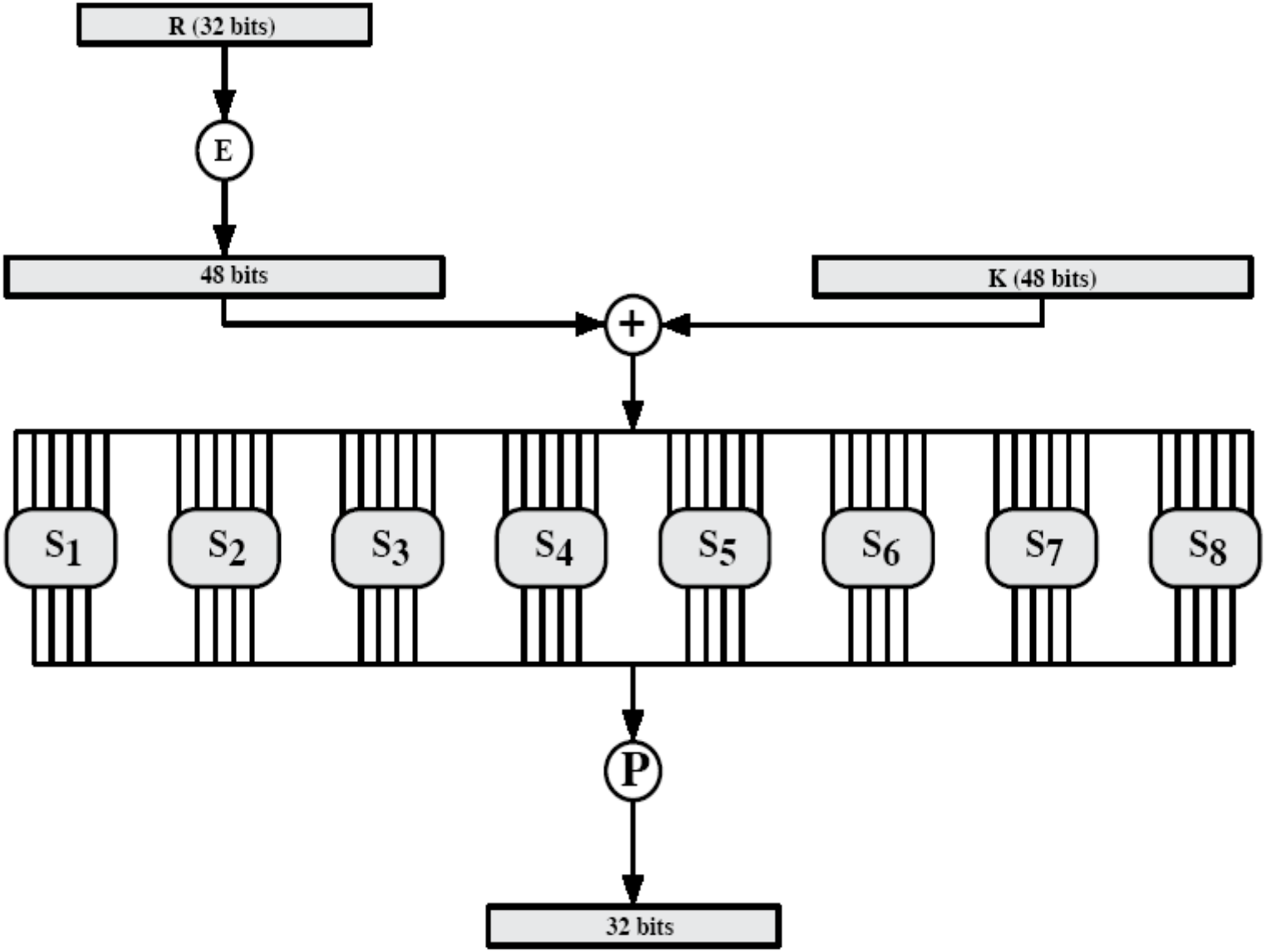


Figure 3.9 Calculation of F(R, K)

DES Key Schedule

- forms subkeys used in each round
  - initial permutation of the key (PC1) which selects 56-bits in 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 each half & permuting them by PC2 for use in round function F
- Concerns about:
  - The algorithm and the key length (56-bits)

DES Analysis

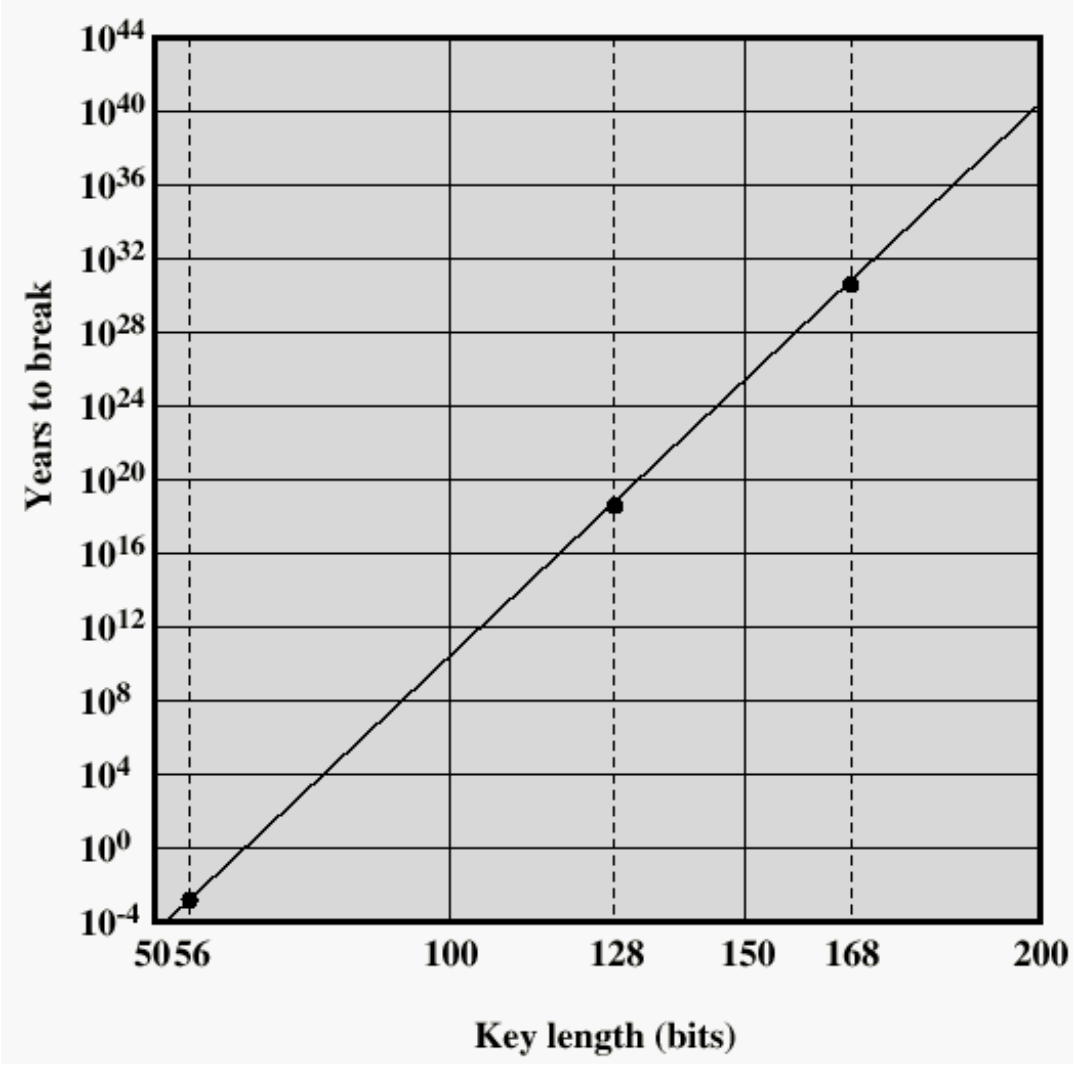
The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- **Avalanche effect** − A small change in plaintext results in the very grate change in the ciphertext.
- **Completeness** − Each bit of ciphertext depends on many bits of plaintext.

During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

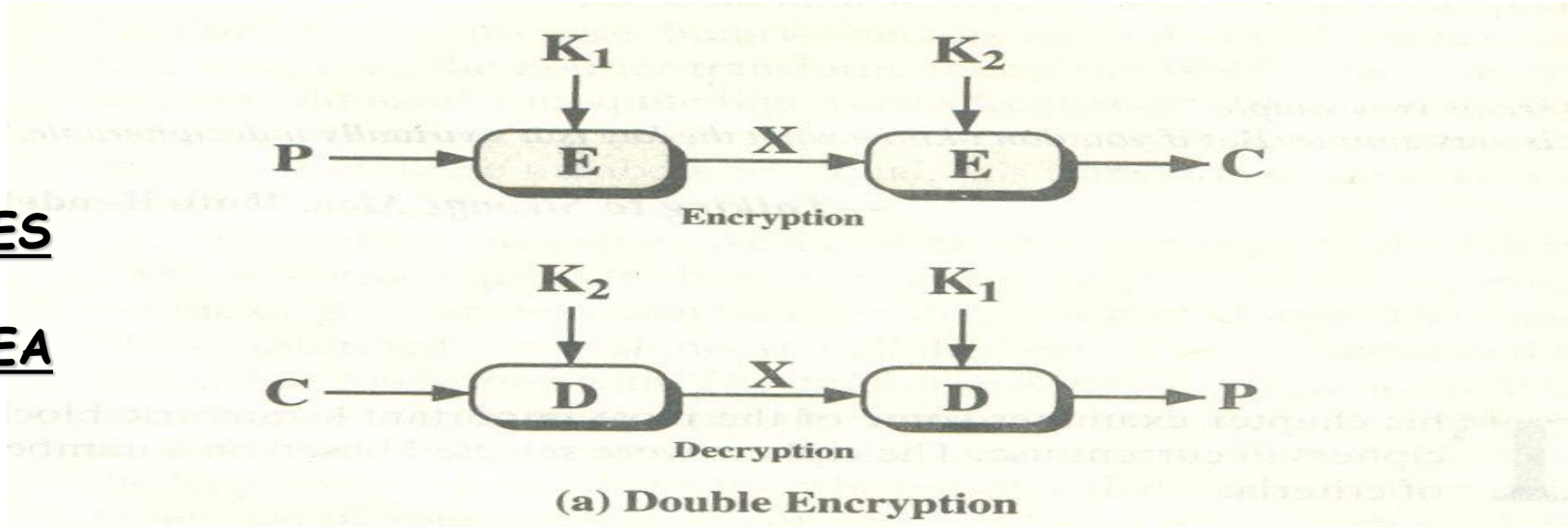
DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

Time to break a code (10^6 decryptions/μs)



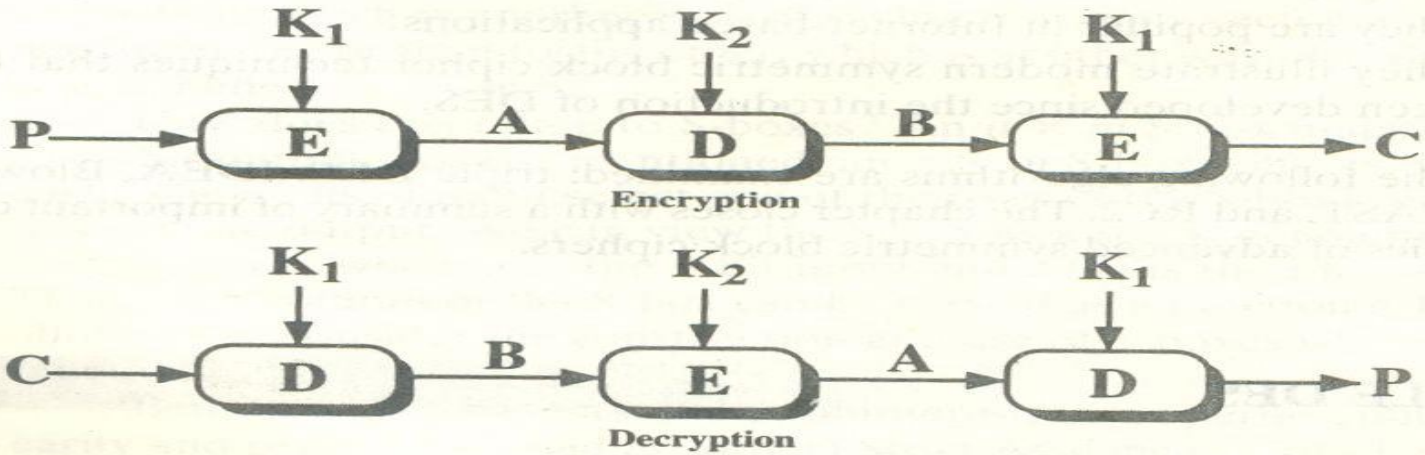
Double DES and Triple DES

Double DES (or) Double DEA



(a) Double Encryption

Triple DES (or) Triple DEA With two keys



(b) Triple Encryption

Triple DEA
with 3 keys

- Use three keys and three executions of the DES algorithm (encrypt-decrypt-encrypt)

$$C = E_{K3}[D_{K2}[E_{K1}[P]]]$$

  - C = ciphertext
  - P = Plaintext
  - EK[X] = encryption of X using key K
  - DK[Y] = decryption of Y using key K
- Effective key length of 168 bits

Triple DEA

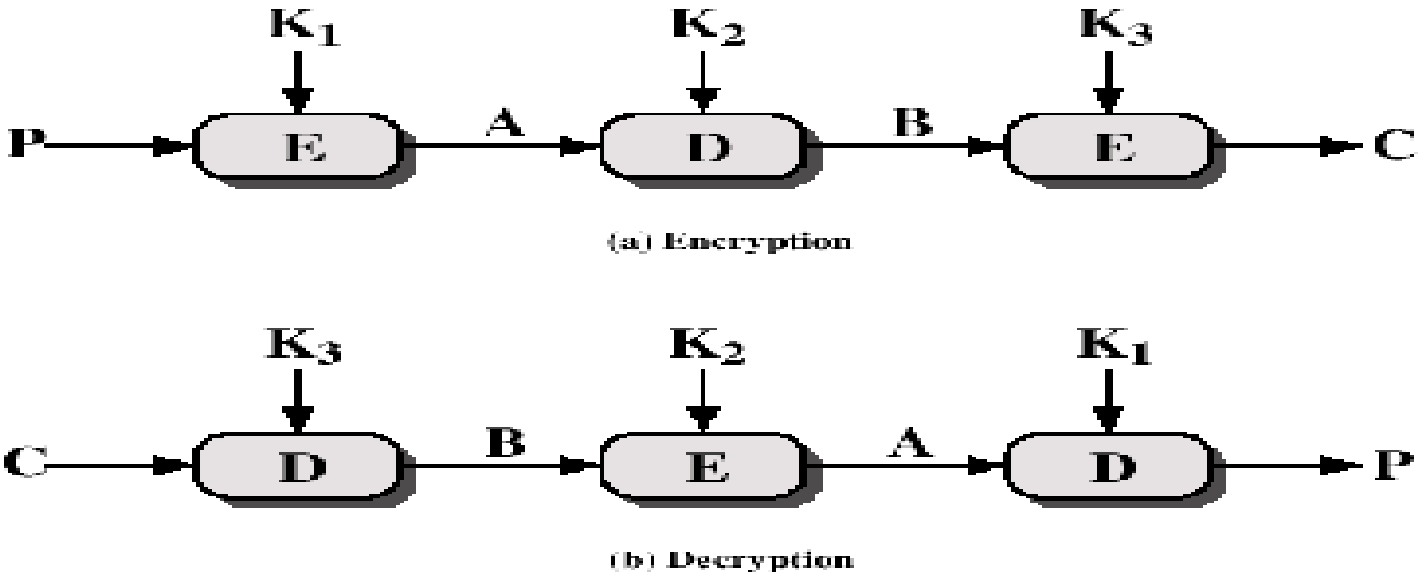


Figure 2.6 Triple DEA

Advanced Encryption Standard (AES) Origins

- clear a replacement for DES was needed
  - have theoretical attacks that can break it
  - have demonstrated exhaustive key search attacks
- can use Triple-DES – but slow, has small blocks
- US NIST issued call for ciphers in 1997
- 15 candidates accepted in Jun 98
- 5 were shortlisted in Aug-99
- Rijndael was selected as the AES in Oct-2000
- issued as FIPS PUB 197 standard in Nov-2001

The AES Cipher - Rijndael

- designed by Rijmen-Daemen in Belgium
- Key: 128/192/256 bit keys,
- Plaintext: 128 bit data

- an **iterative** rather than **Feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round

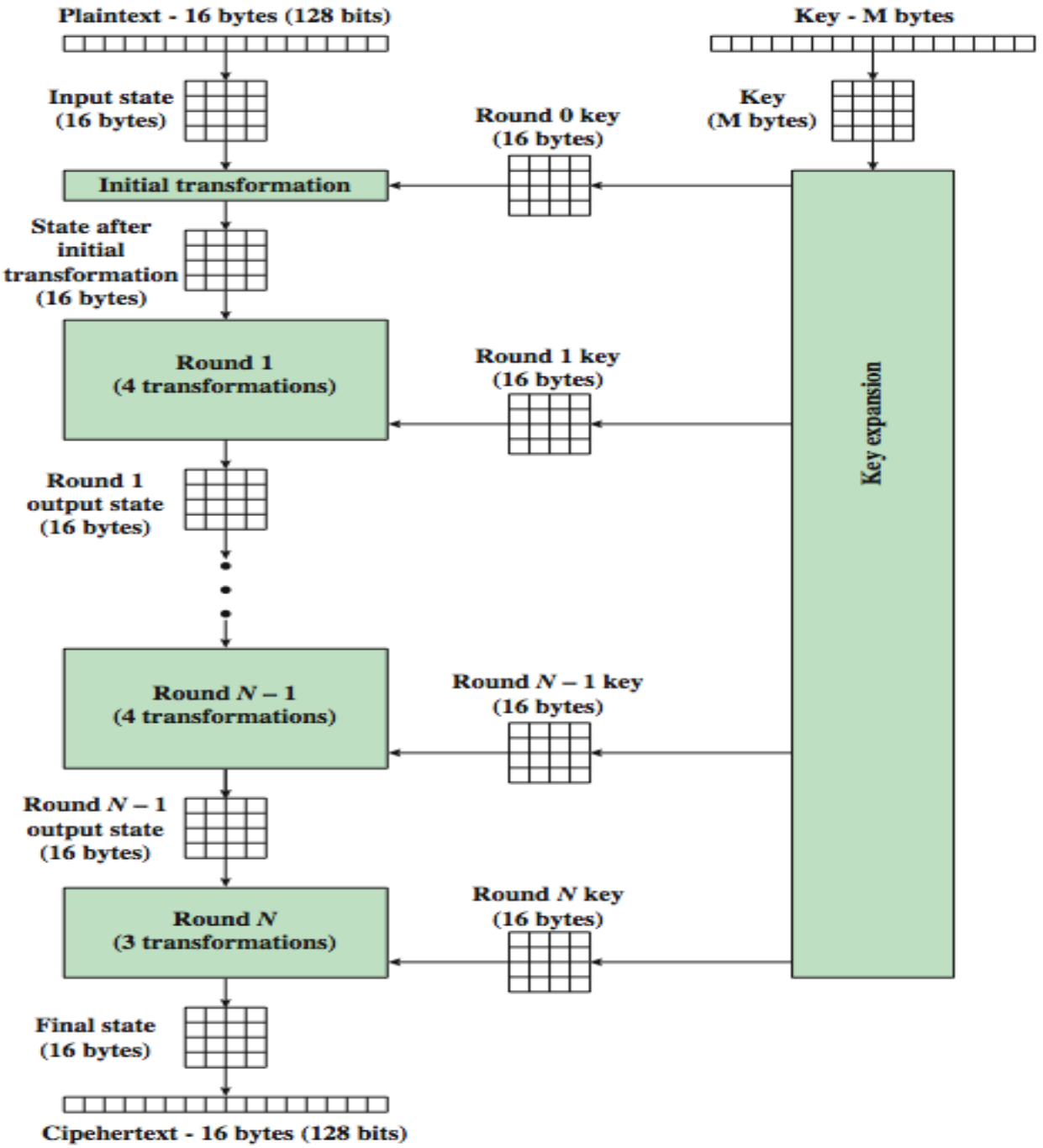
- designed to have:
  - resistance against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

AES Structure

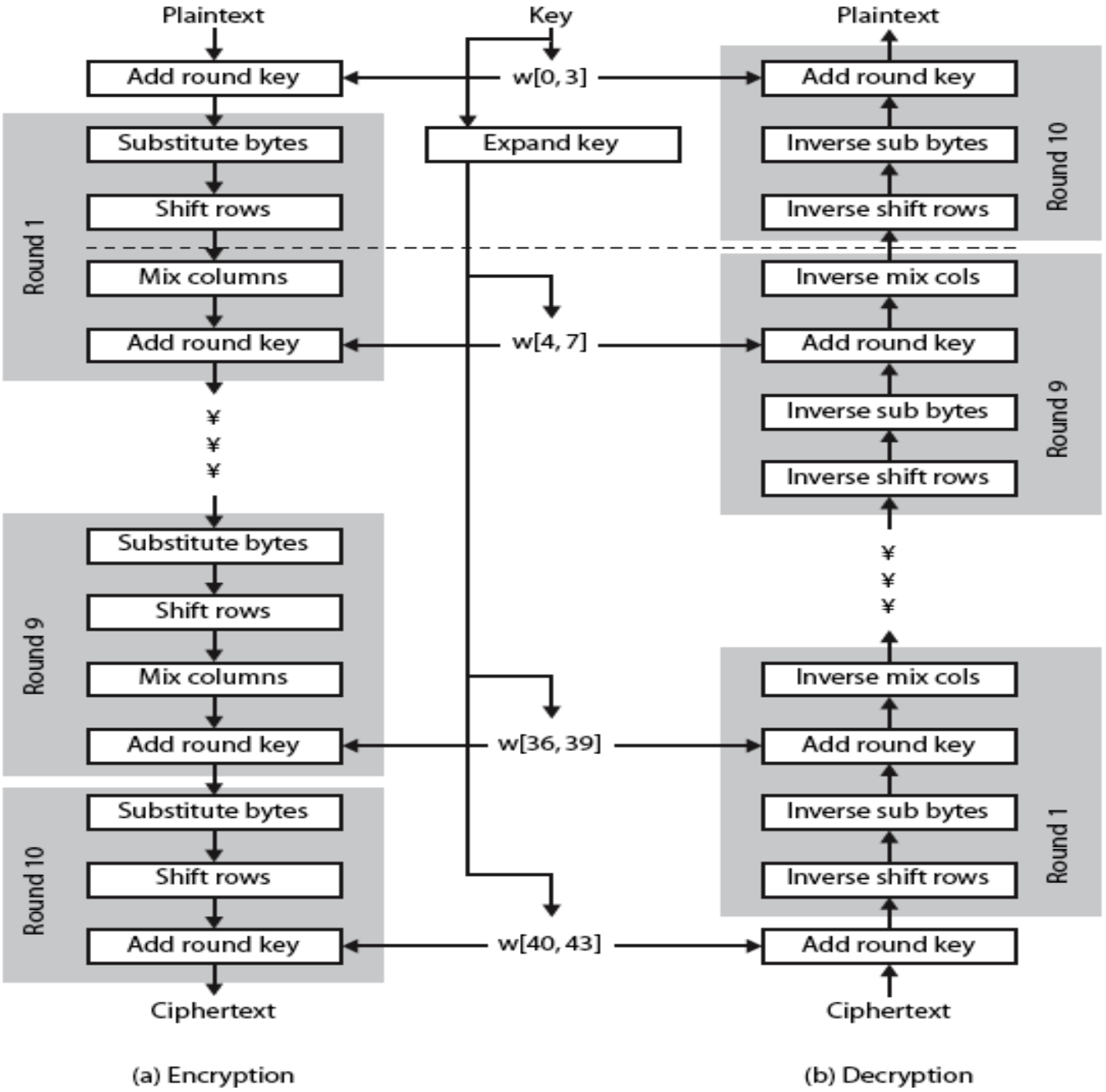
- data block of 4 columns of 4 bytes is state
- key is expanded to array of words
- has 9/11/13 rounds in which state undergoes:
  1. **byte substitution** (1 S-box used on every byte)
  2. **shift rows** (permute bytes between groups/columns)
  3. **mix columns** (subs using matrix multiply of groups)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

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

AES Encryption Process



AES Structure



Some Comments on AES

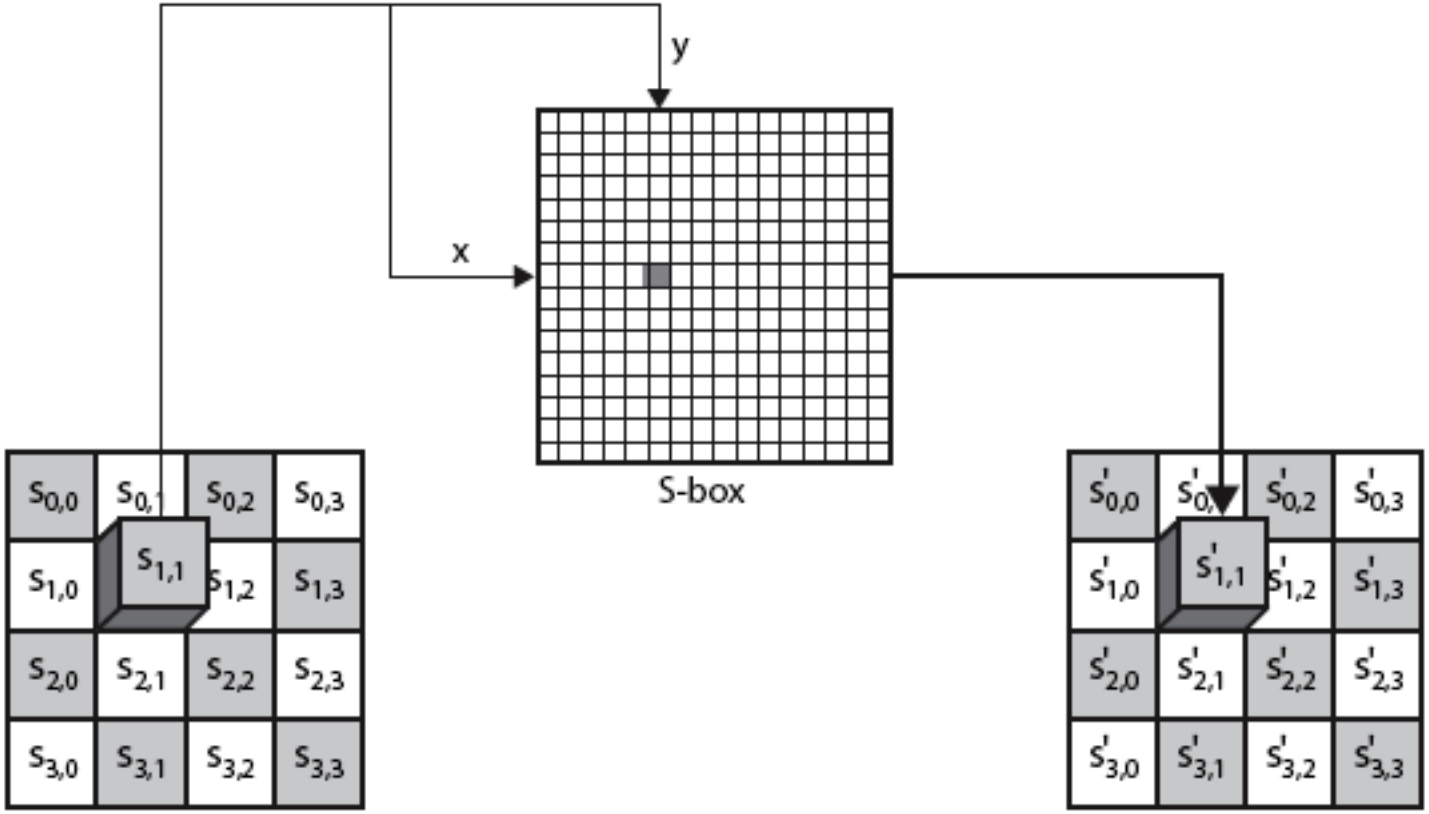
1. an iterative rather than Feistel cipher
2. key expanded into array of 32-bit words
    1. four words form round key in each round
3. 4 different stages are used as shown
4. has a simple structure
5. only AddRoundKey uses key
6. AddRoundKey a form of Vernam cipher
7. each stage is easily reversible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

Substitute Bytes

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

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

Substitute Bytes



Substitute Bytes Example →

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

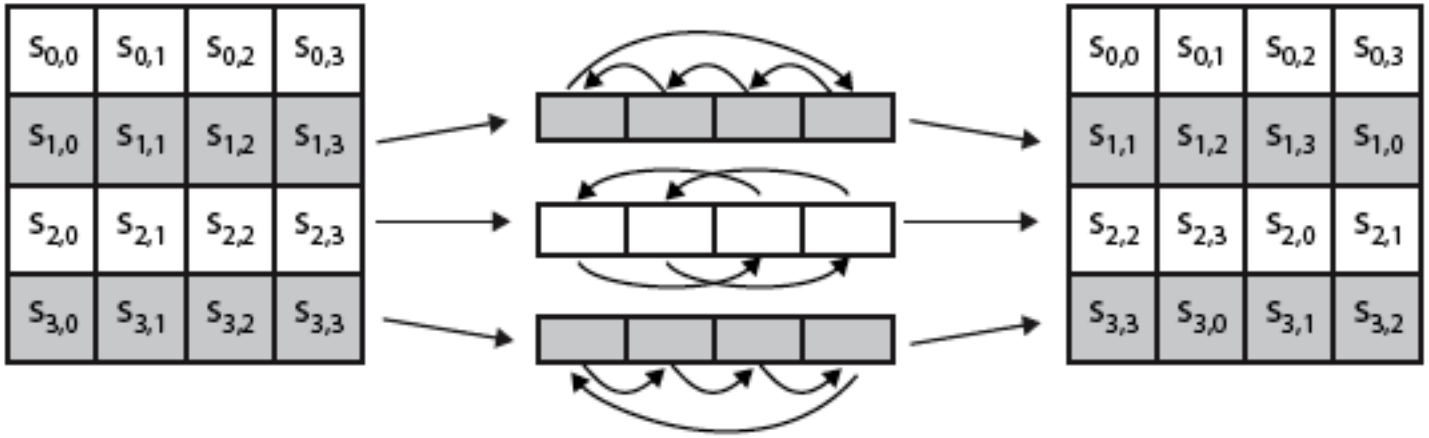
→

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

Shift Rows

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

Shift Rows



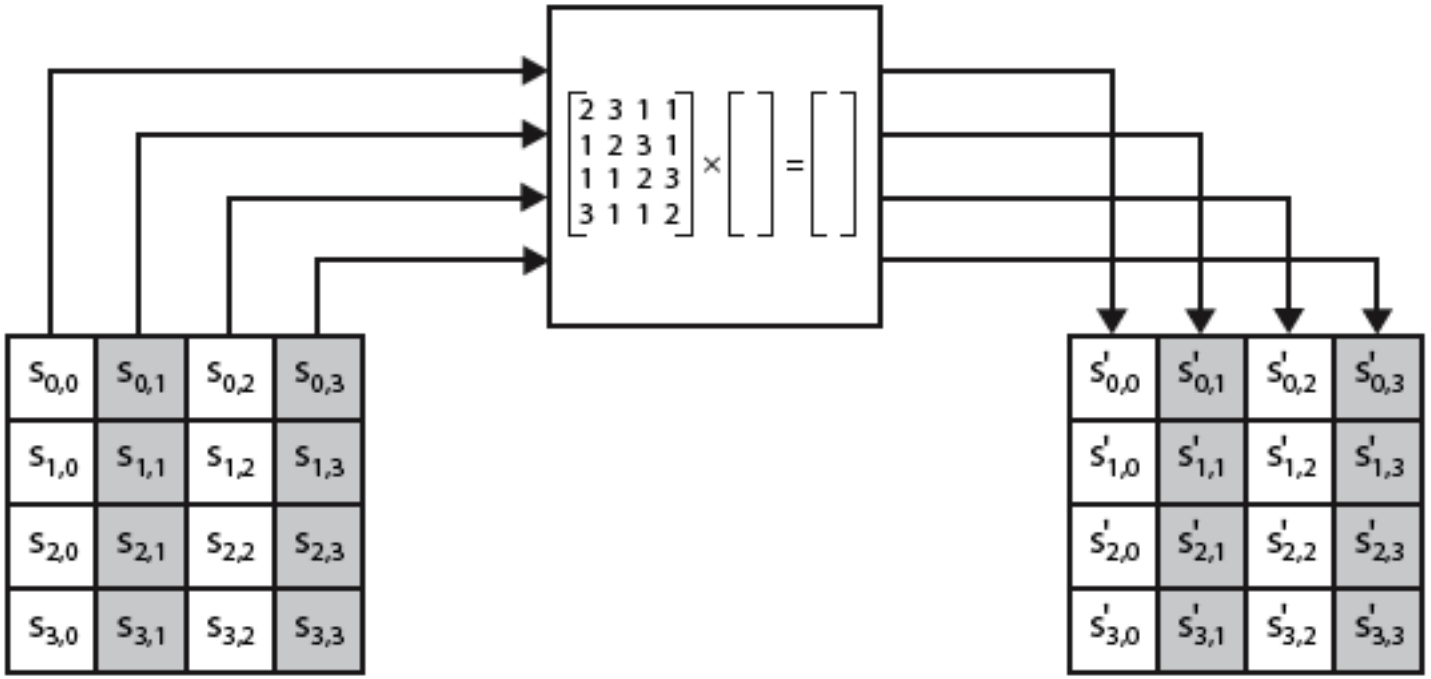
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

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

Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column



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

Mix Columns Example

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

→

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

$$(\{02\} \bullet \{87\}) \oplus (\{03\} \bullet \{6E\}) \oplus \{46\} \oplus \{A6\} = \{47\}$$

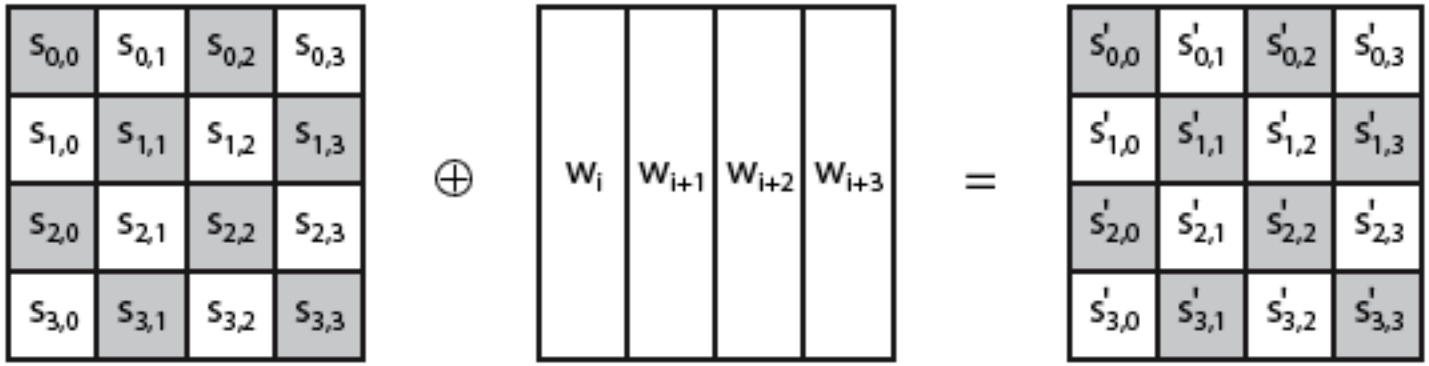
$$\{87\} \oplus (\{02\} \bullet \{6E\}) \oplus (\{03\} \bullet \{46\}) \oplus \{A6\} = \{37\}$$

$$\{87\} \oplus \{6E\} \oplus (\{02\} \bullet \{46\}) \oplus (\{03\} \bullet \{A6\}) = \{94\}$$

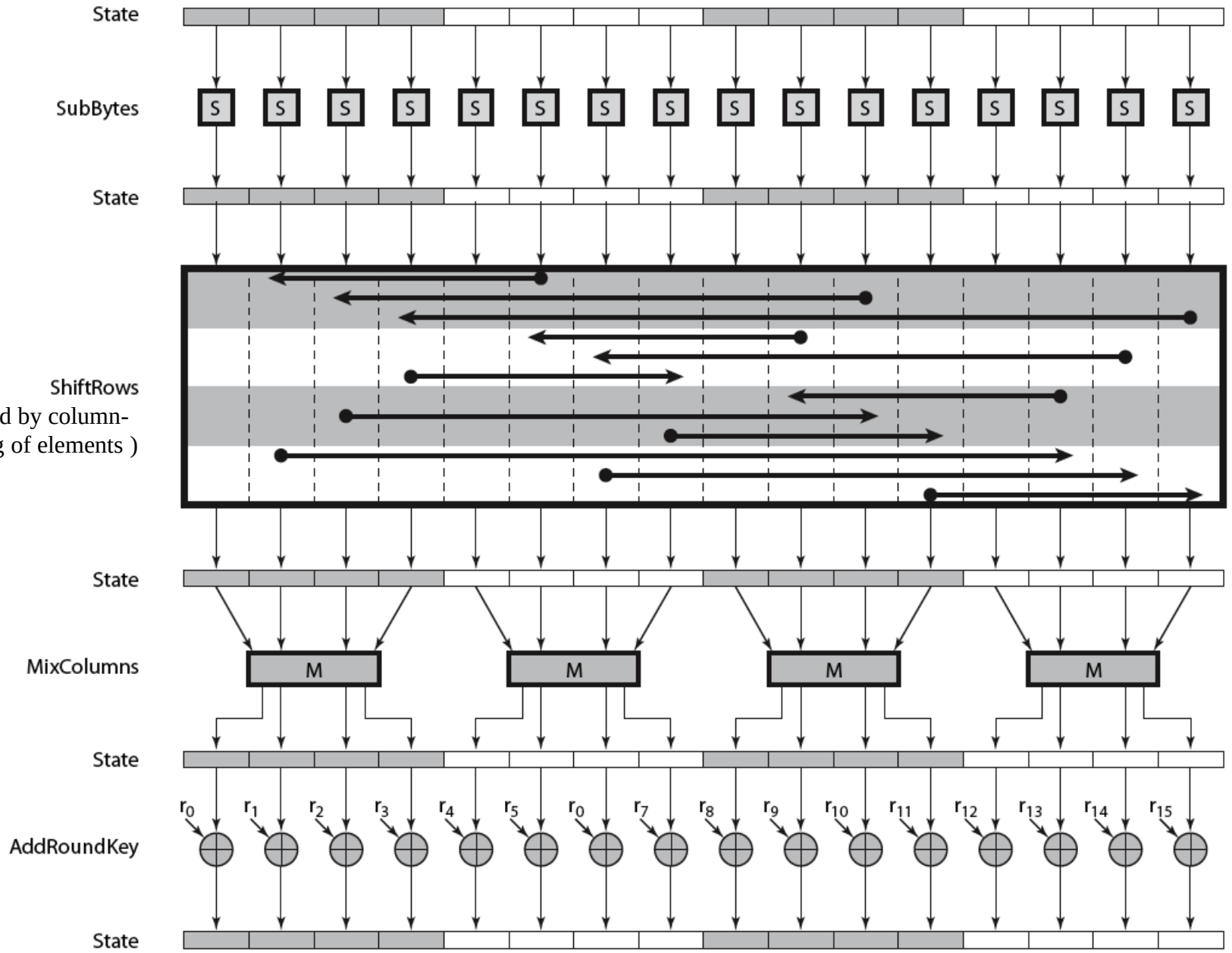
$$(\{03\} \bullet \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \bullet \{A6\}) = \{ED\}$$

Add Round Key

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



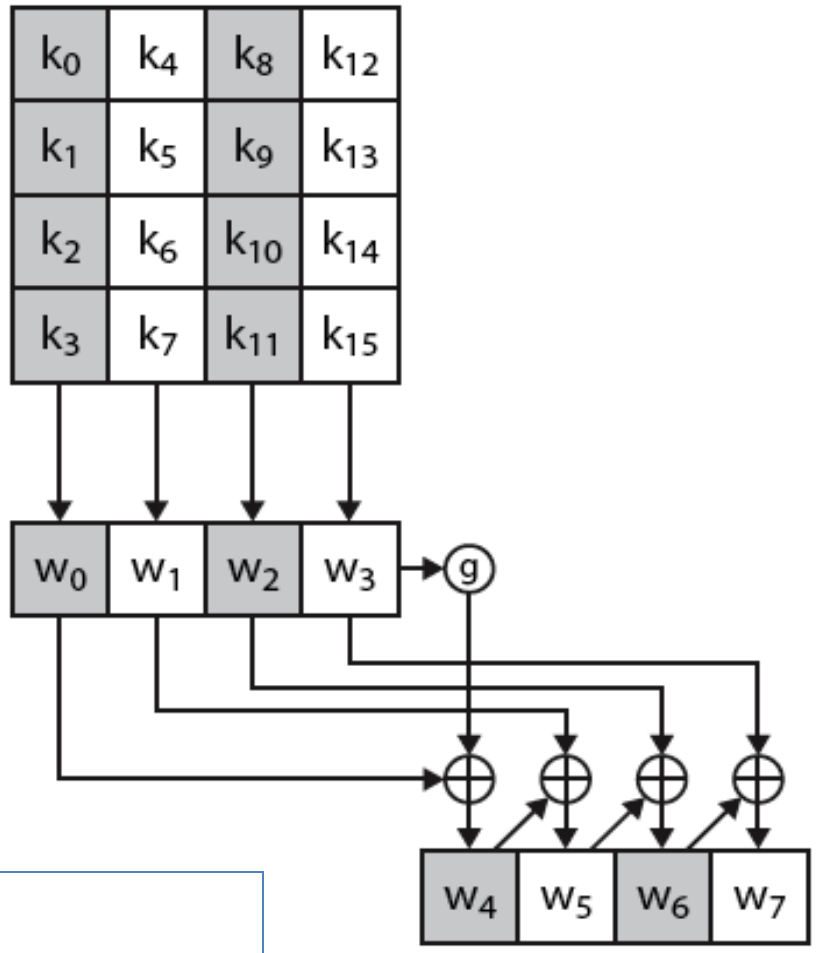
AES Round


State
SubBytes
S
State
ShiftRows
(state formed by column-wise placing of elements)
State
MixColumns
M
State
AddRoundKey
r0 r1 r2 r3 r4 r5 r0 r7 r8 r9 r10 r11 r12 r13 r14 r15
State

AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words

1. start by copying key into first 4 words
2. then loop creating words that depend on values in previous & 4 places back
   - ✓ in 3 of 4 cases just XOR these together
   - ✓ 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back



$w_i = w_{i-1}$ xor w_{i-4} | if i≠ multiple of 4

g = S-Box (LCS(w_3)) xor $RoundConstant_i$

Key Expansion Rationale

- designed to resist known attacks
- design criteria included
  - knowing part key insufficient to find many more
  - invertible transformation
  - fast on wide range of CPU's
  - use round constants to break symmetry
  - diffuse key bits into round keys
  - enough non-linearity to hinder analysis
  - simplicity of description

Other Symmetric Block Ciphers

- **International Data Encryption Algorithm (IDEA)**
  - 128-bit key
  - Used in PGP
- **Blowfish** **(A variant of DES with S-Box having values that are both random and key dependant)**
  - Easy to implement
  - High execution speed
  - Run in less than 5K of memory

Other Symmetric Block Ciphers

- **RC5**
  - Variable number of rounds
  - Data-dependent rotations
  - Variable-length key
  - Suitable for hardware and software
  - Fast, simple
  - Adaptable to processors of different word lengths
  - Low memory requirement
  - High security
- **Cast-128**
  - Key size from 40 to 128 bits
  - The round function differs from round to round

Table 2.3 Conventional Encryption Algorithms

Algorithm	Key Size (bits)	Block Size (bits)	Number of Rounds	Applications
DES	56	64	16	SET, Kerberos
Triple DES	112 or 168	64	48	Financial key management, PGP, S/MIME
AES	128, 192, or 256	128	10, 12, or 14	Intended to replace DES and 3DES
IDEA	128	64	8	PGP
Blowfish	variable to 448	64	16	Various software packages
RC5	variable to 2048	64	variable to 255	Various software packages

Cipher Block Modes of Operation

Encrypting a Large Massage

1. Electronic Code Book (ECB)
2. Cipher Block Chaining (CBC)
3. Output Feedback Mode (OFB)
4. Cipher Feedback Mode (CFB)
5. Counter Mode

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: 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.

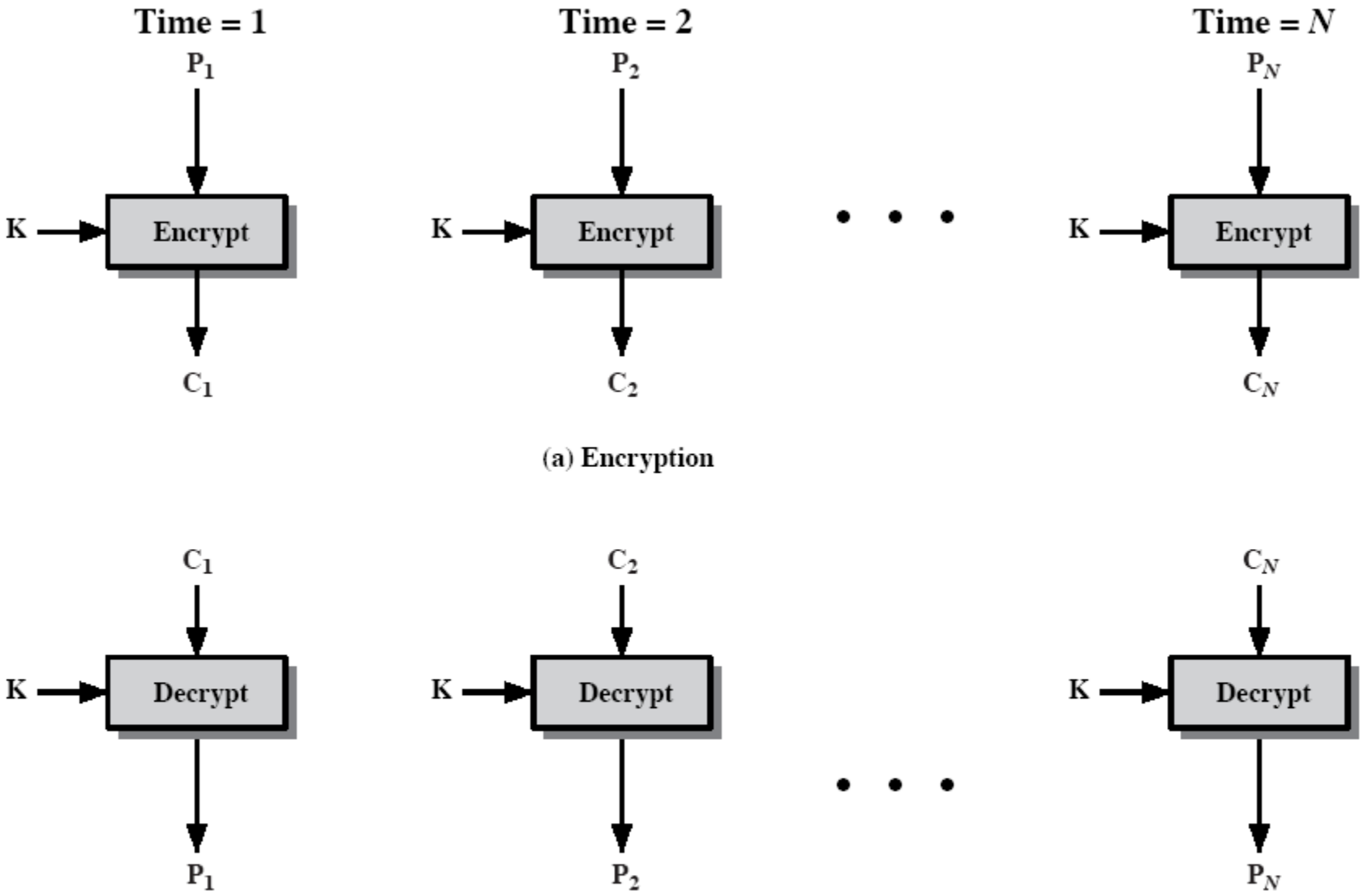


Figure 3.11 Electronic Codebook (ECB) Mode

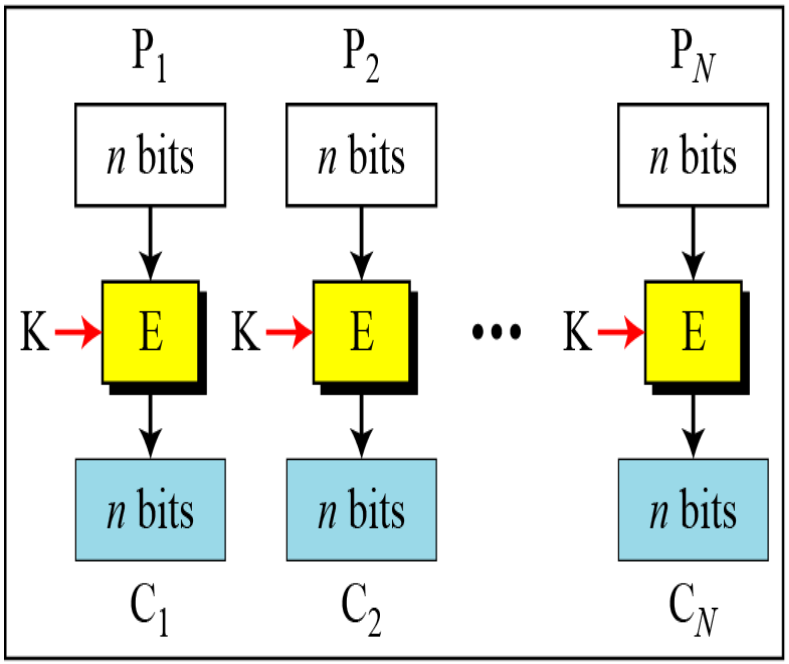
ECB Scheme

Encryption: $C_i = E_K (P_i)$ Decryption: $P_i = D_K (C_i)$

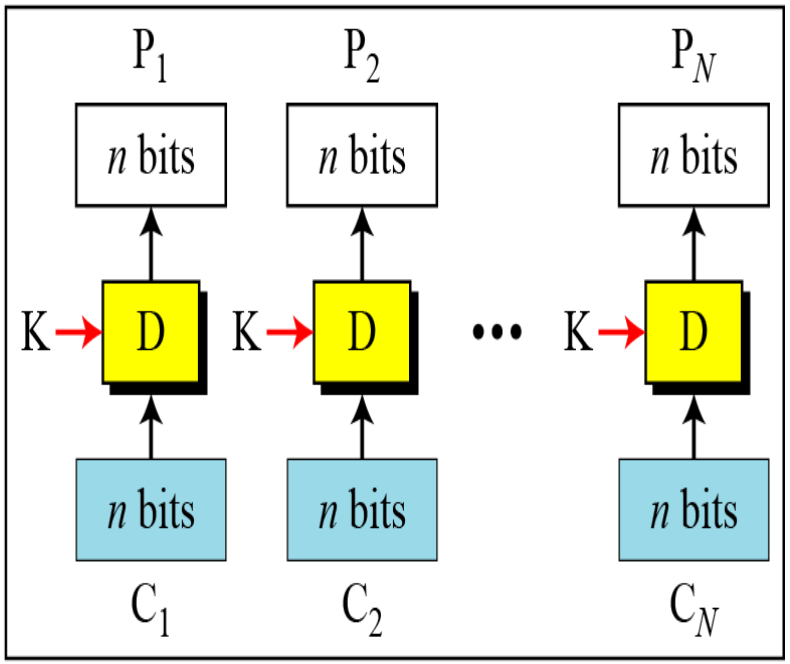
E: Encryption D: Decryption

P_i: Plaintext block i C_i: Ciphertext block i

K: Secret key



Encryption



Decryption

Remarks on ECB

Advantages

1. As we can process each block of plaintext independently of each other, we can process multiple blocks simultaneously.
2. If any plaintext or ciphertext blocks lost, it does not affect on the output of other blocks.
3. Parallel processing during encryption as well as decryption helps to increase the speed and the performance of the algorithm.

Disadvantages

If two plaintext blocks are identical, then the ciphertext block generated are also same. Therefore, known plaintext attack is possible.

- Typical application:
  - secure transmission of short pieces of information (e.g. a temporary encryption key)

Cipher Block Chaining (CBC)

- Solve security deficiencies in ECB
  - Repeated same plaintext block result different ciphertext block

- Each previous cipher blocks is chained to be input with current plaintext block, hence name

- Use Initialization (or Initial) Vector (IV) to start process

$$C_i = E_K (P_i \text{ XOR } C_{i-1})$$
$$C_0 = IV$$

- Uses: bulk data encryption, authentication

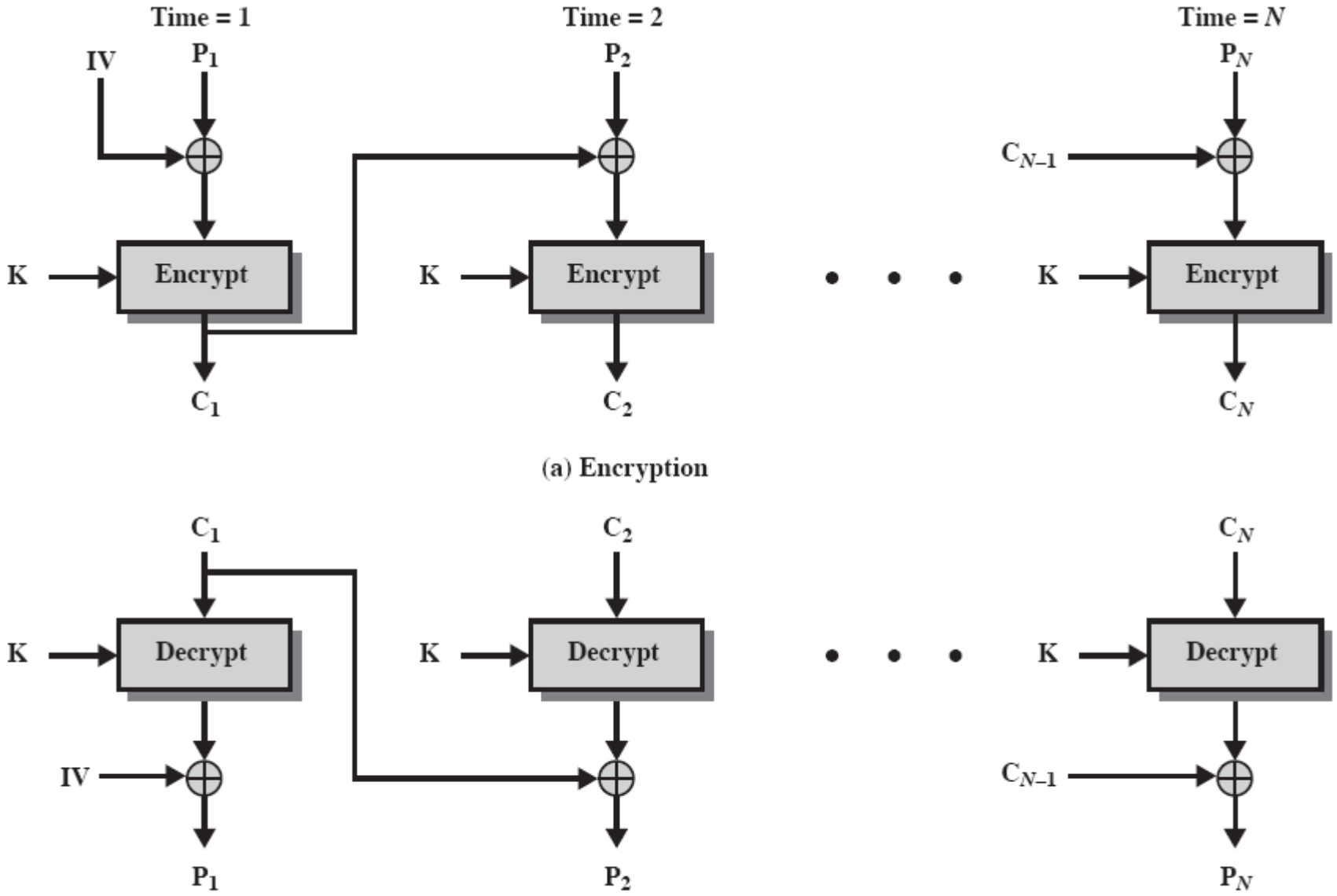


Figure 3.12 Cipher Block Chaining (CBC) Mode

Remarks on CBC

Advantages

1. For identical blocks of plaintext, different ciphertext blocks are generated. So, CBC is more secure as compared to ECB mode.
2. Hash value, i.e., last ciphertext block, helps to identify if the message is original or modified.

Disadvantages

1. Parallel operation cannot be performed. So, it is slower as compared to ECB.
2. Lost/missing of any block of ciphertext stops the decryption process of the remaining blocks.

- Initialization Vector (IV)
  - May sent encrypted in ECB mode before the rest of ciphertext

Cipher FeedBack (CFB)

- Use Initial Vector to start process
- Encrypt previous ciphertext , then combined with the plaintext block using X-OR to produce the current ciphertext
- Cipher is fed back (hence name) to concatenate with the rest of IV
- Plaintext is treated as a stream of bits
  - Any number of bit (1, 8 or 64 or whatever) to be feed back (denoted CFB-1, CFB-8, CFB-64)
- Relation between plaintext and ciphertext

$$C_i = P_i \text{ XOR } \text{SelectLeft}(E_K(\text{ShiftLeft}(C_{i-1})))$$
$$C_0 = IV$$

- Uses: stream data encryption, authentication

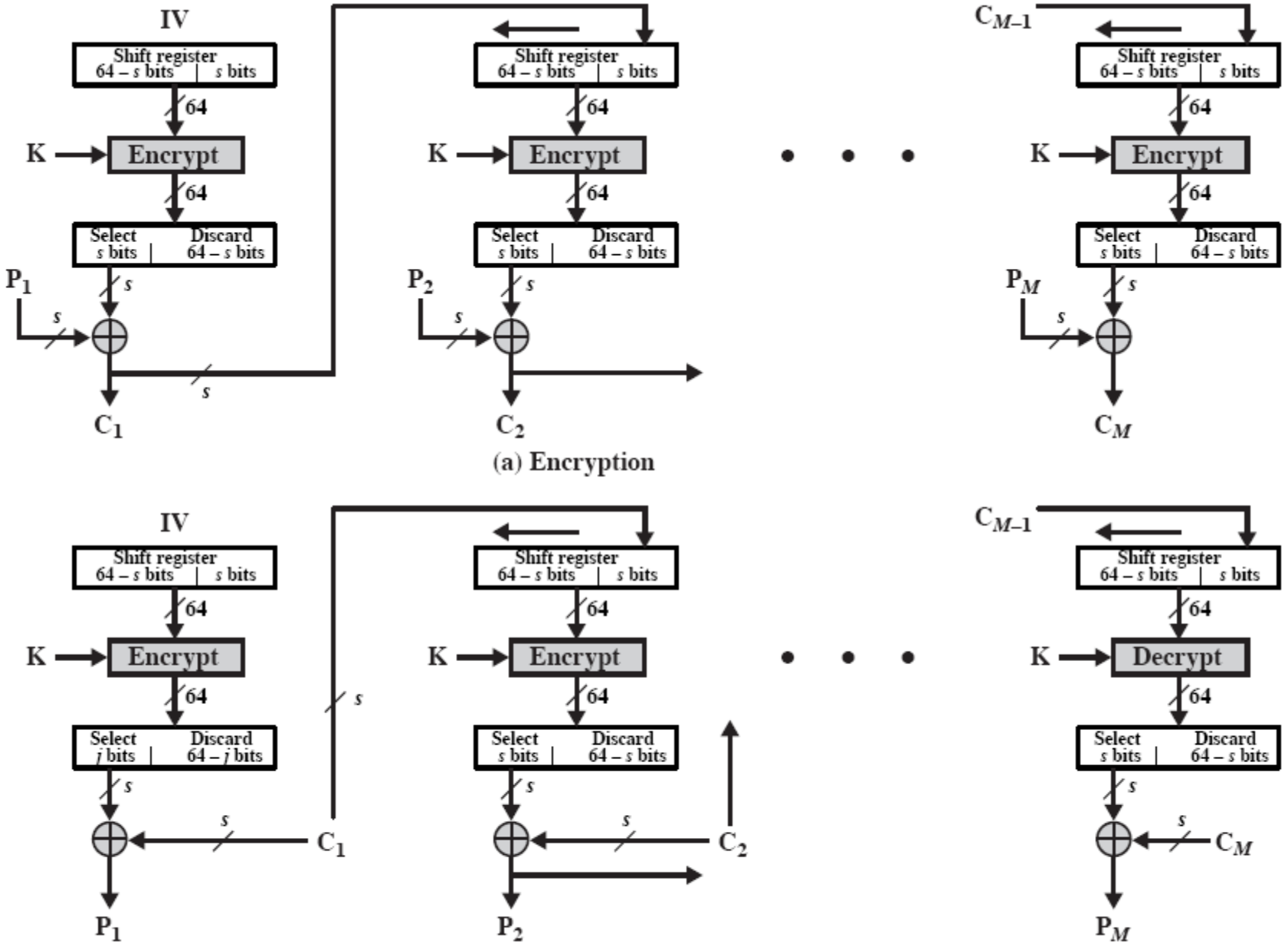


Figure 3.13 *s*-bit Cipher Feedback (CFB) Mode

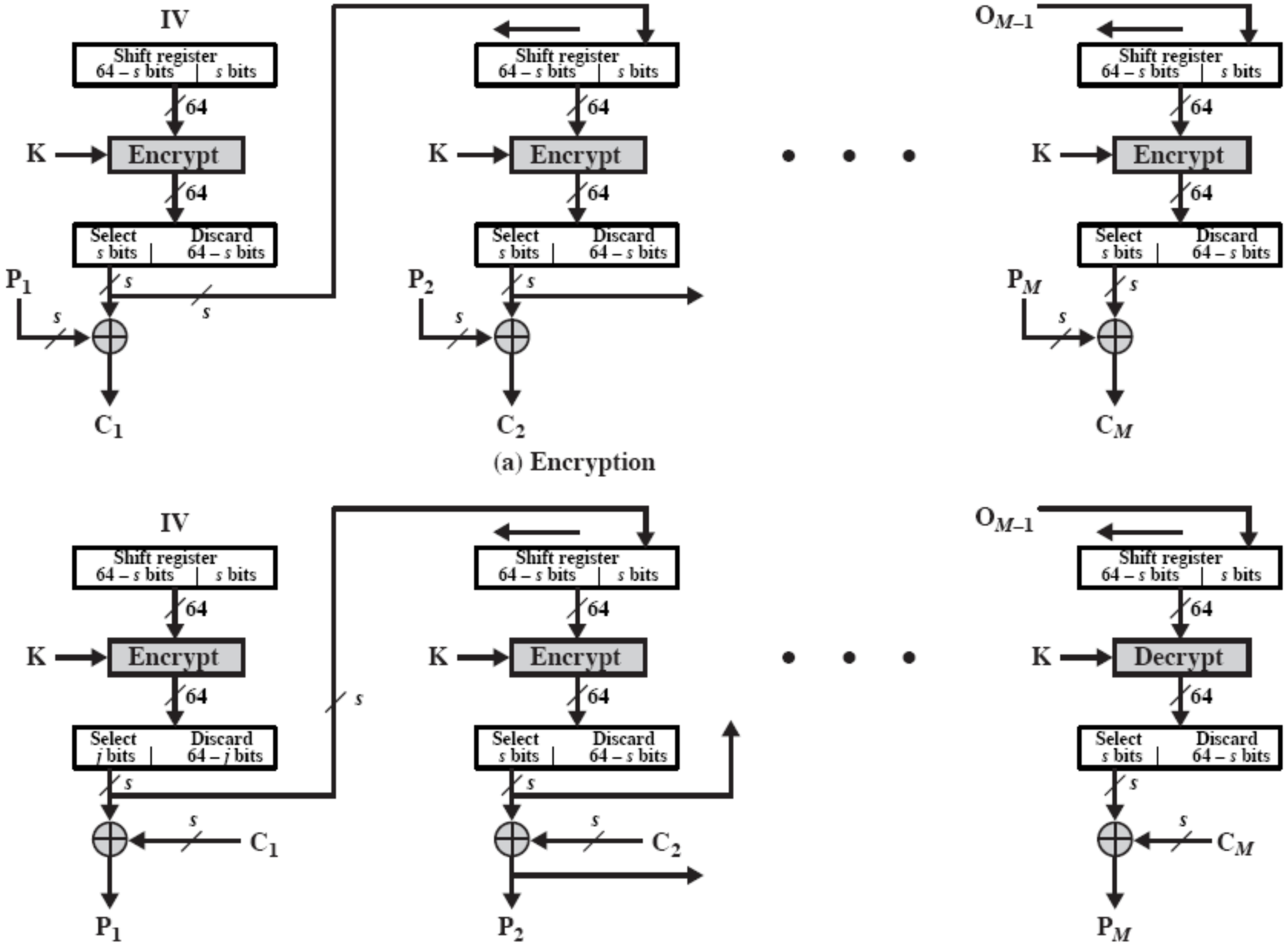


Figure 3.14 *s*-bit Output Feedback (OFB) Mode

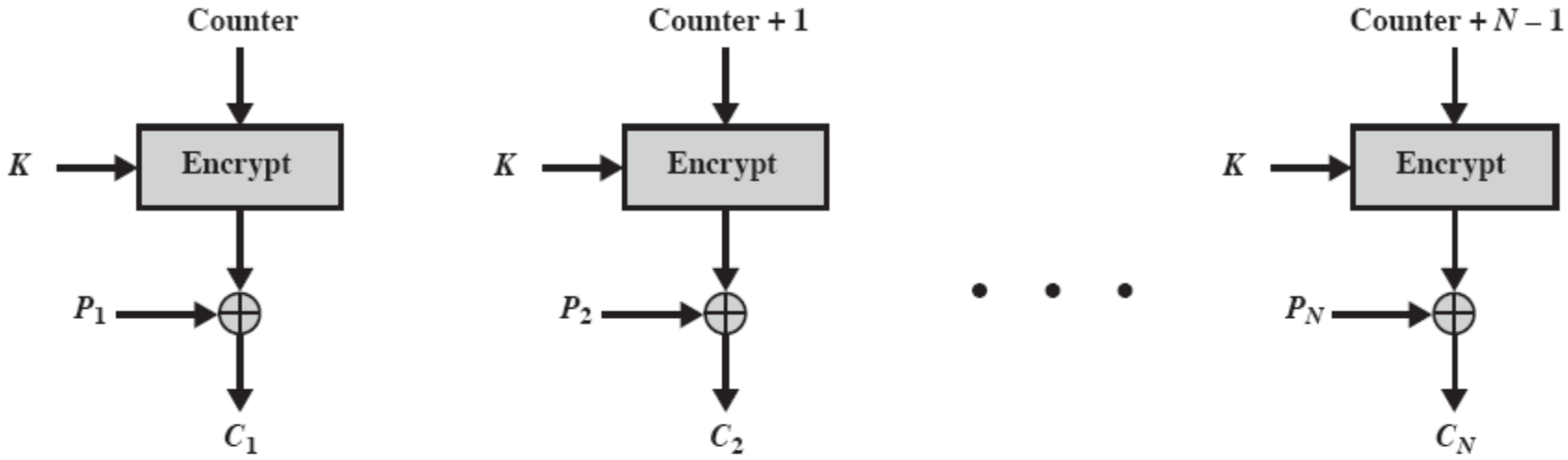
Remarks on Feedback Mode

Advantage

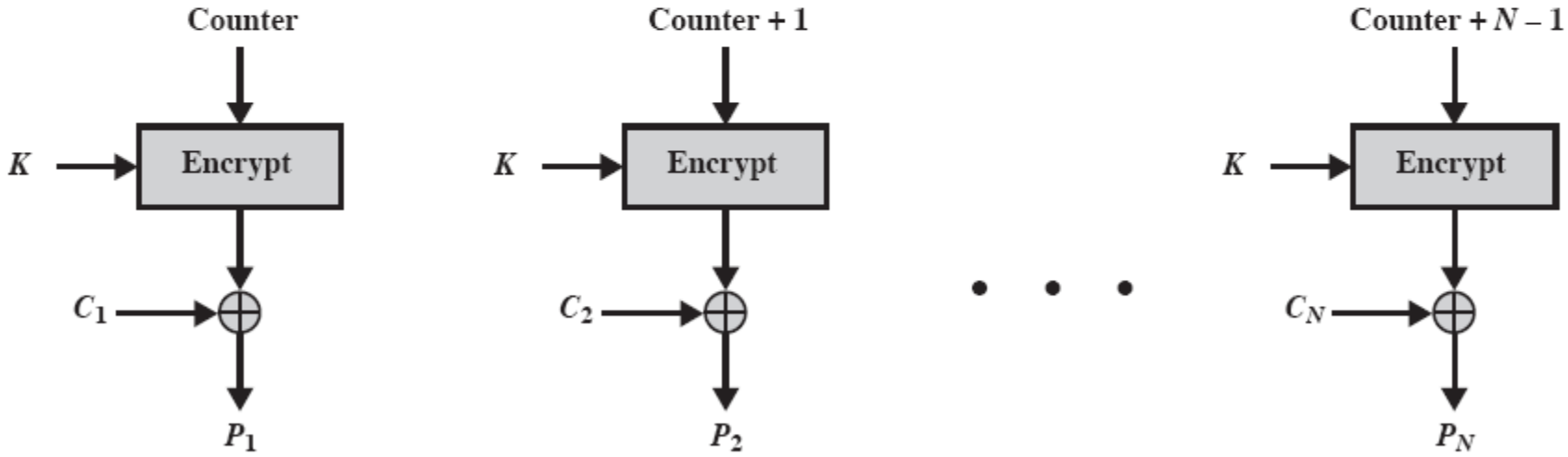
- Free from bit error rate.

Disadvantage

- Vulnerable to a stream modification attack.



(a) Encryption



(b) Decryption

Figure 3.15 Counter (CTR) Mode

Location of Encryption Devices

- **Link encryption:**
  - A lot of encryption devices
  - High level of security
  - Decrypt each packet at every switch
- **End-to-end encryption**
  - The source encrypt and the receiver decrypts
  - Payload encrypted
  - Header in the clear
- **High Security:** Both link and end-to-end encryption are needed (see Figure 2.9)

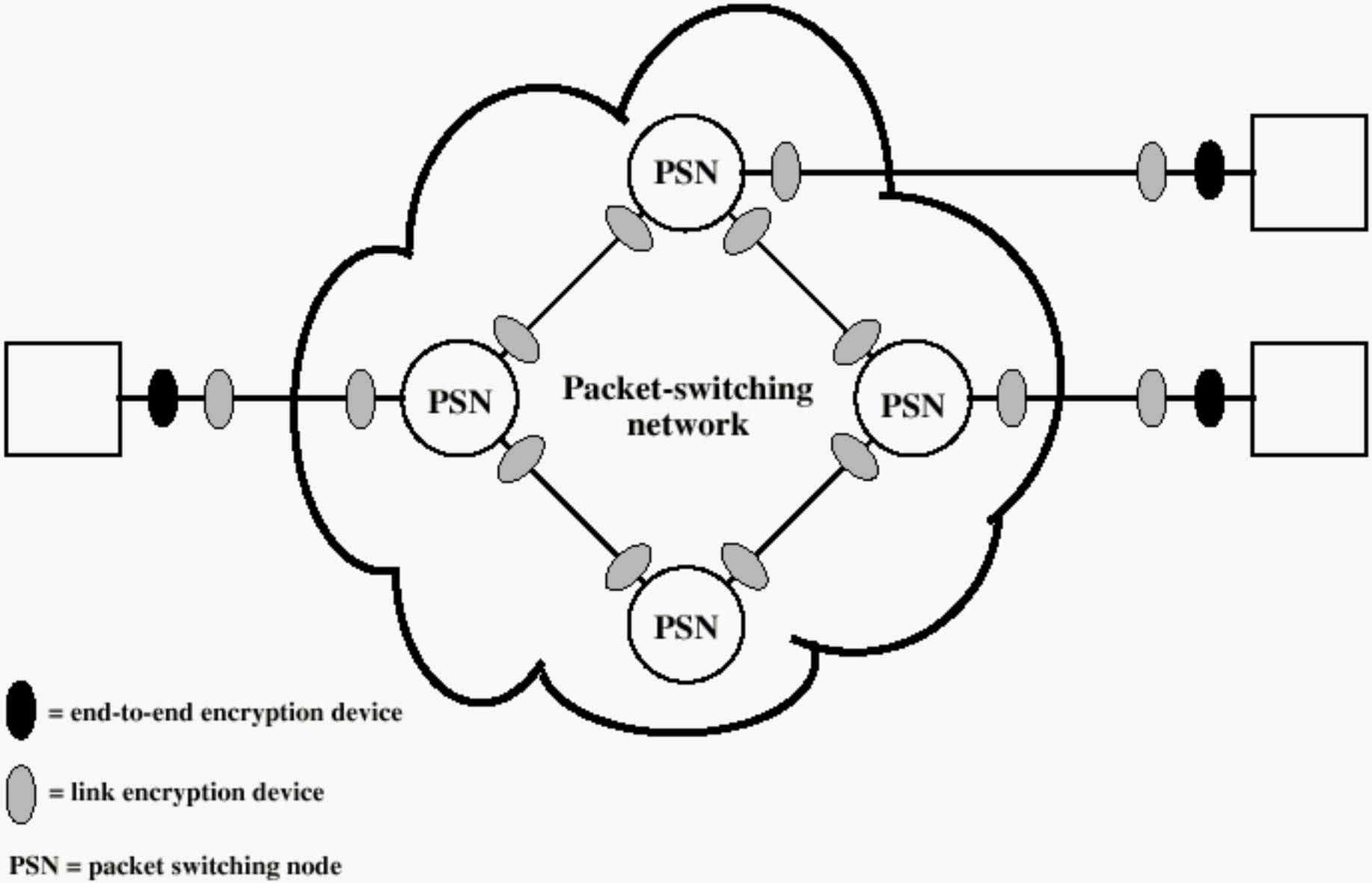


Figure 2.9 Encryption Across a Packet-Switching Network

Key Distribution

1. A key could be selected by A and physically delivered to B.
2. A third party could select the key and physically deliver it to A and B.
3. If A and B have previously used a key, one party could transmit the new key to the other, encrypted using the old key.
4. If A and B each have an encrypted connection to a third party C, C could deliver a key on the encrypted links to A and B.

Key Distribution (See Figure 2.10)

- **Session key:**
  - Data encrypted with a one-time session key.At the conclusion of the session the key is destroyed
- **Permanent key:**
  - Used between entities for the purpose of distributing session keys

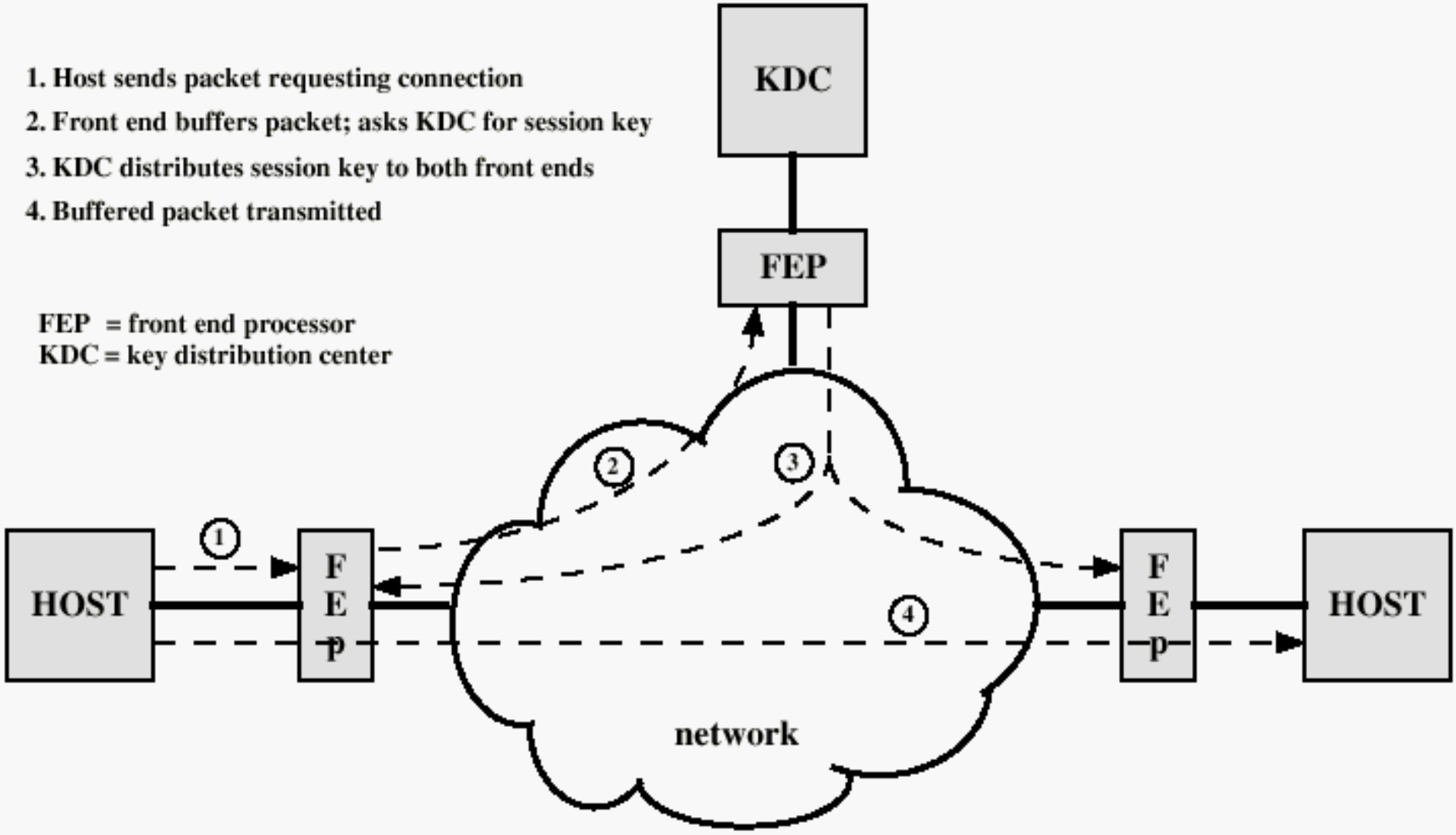


Figure 2.10 Automatic Key Distribution for Connection-Oriented Protocol

Authentication

- **Requirements** - must be able to verify that:
    1. Message came from apparent source or author,
    2. Contents have not been altered,
    3. Sometimes, it was sent at a certain time or sequence.

- Protection against active attack (falsification of data and transactions)

Approaches to Message Authentication

1. Authentication Using Conventional Encryption

   Only the sender and receiver should share a key.

   Message with an error-detection code and a sequence no, timestamp.

2. Message Authentication without Message Encryption
   - -Message Authentication Code
   - -One-Way hash function

2. Message Authentication without Message Encryption - Message Authentication Code

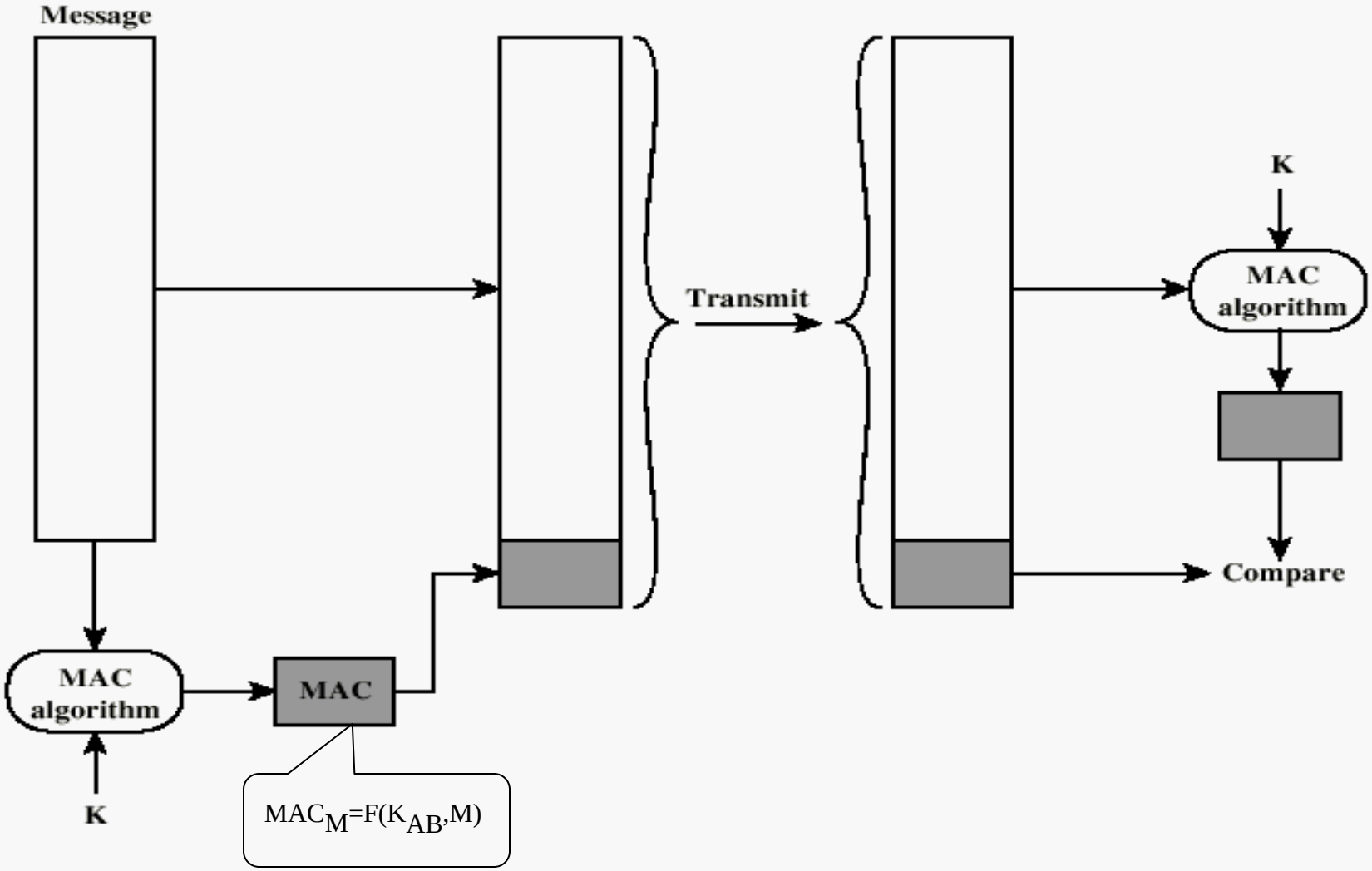


Figure 3.1 Message Authentication Using a Message Authentication Code (MAC)

If received code matches the calculated code, then:

- The receiver is assured that the message has not altered.

- The receiver is assured that the message is from the alleged sender.

- If the message includes a sequence number, then the receiver can be assured of the proper sequence, because an attacker cannot successfully alter the sequence number.

2. Message Authentication without Message Encryption

- One-Way hash function

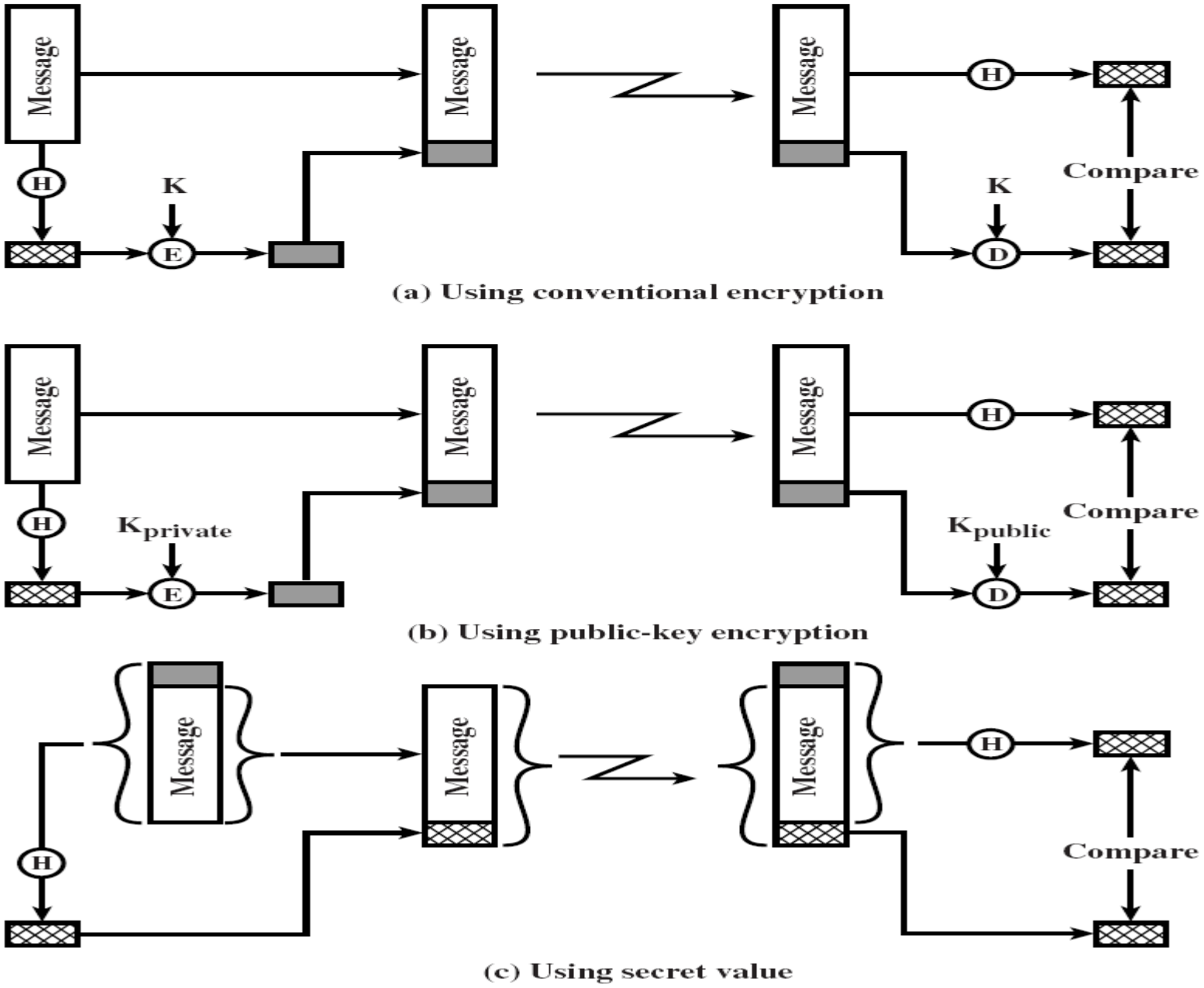


Figure 3.2 Message Authentication Using a One-Way Hash Func

Secure HASH Functions

Purpose of the HASH function is to produce a "fingerprint.

Properties of a HASH function H :

1. H can be applied to a block of data of any size
2. H produces a fixed length output
3. H(x) is easy to compute for any given x.
4. For any given code h, it is computationally infeasible to find x such that H(x) = h
5. For any given block x, it is computationally infeasible to find $y \neq x$ with H(y) = H(x).
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y)

Simple Hash Function

	bit 1	bit 2	• • •	bit *n*
block 1	b_{11}	b_{21}		b_{n1}
block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
block *m*	b_{1m}	b_{2m}		b_{nm}
hash code	C_1	C_2		C_n

b_{11} xor b_{12} xor …..xor b_{1m}

Figure 3.3 Simple Hash Function Using Bitwise XOR

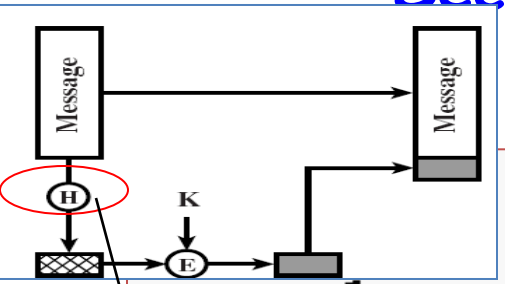
• SHA-1

- ✓ The algorithm takes as input a message with a maximum length of less than 2^64 bits and produces as output a 160-bit message digest.
- ✓ The input is processed in 512-bit blocks.

Steps

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 5-word (160-bit) buffer (A,B,C,D,E) to (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. process message in 16-word (512-bit) blocks:
   expand 16 words into 80 words by mixing & shifting
   use 4 rounds of 20 bit operations on message block & buffer
   add output to input to form new buffer value
5. output hash value is the final buffer value

Message Digest Generation Using Secure Hash Algorithm – 1 (SHA-1)



Padding (1 to 512 bits)

Message length (K mod 2^{64})

$L \times 512$ bits $= N \times 32$ bits

K bits

Message | 100...0

512 bits | 512 bits | 512 bits | 512 bits

Y_0 | Y_1 | Y_q | Y_{L-1}

512 | 512 | 512 | 512

IV (160) → H_{SHA} → CV_1 (160) → H_{SHA} → CV_q (160) → H_{SHA} → CV_{L-1} (160) → H_{SHA} → 160-bit digest

SHA-1 Processing of single 512-Bit Block

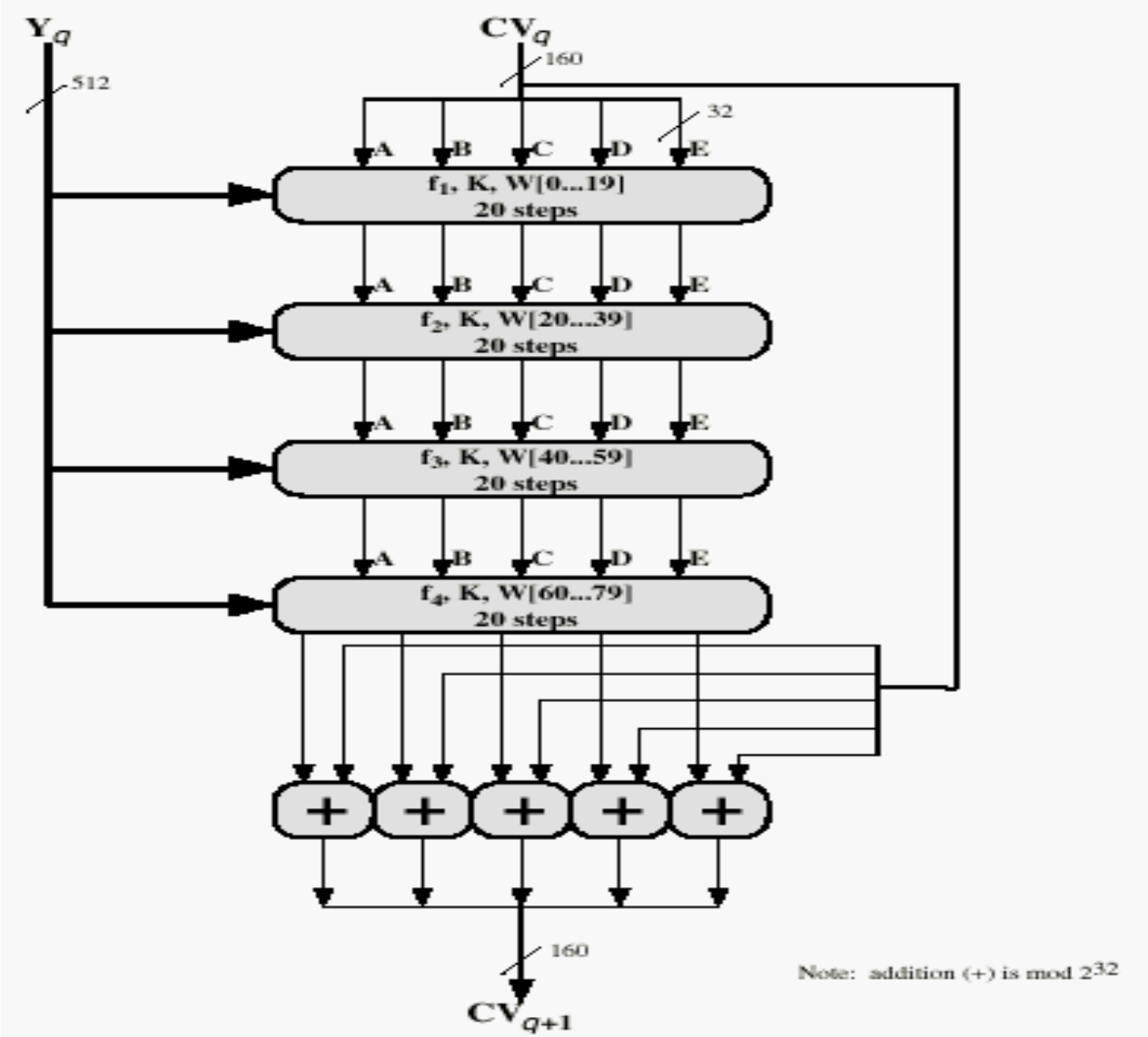


Figure 3.5 SHA-1 Processing of a Single 512-bit Block

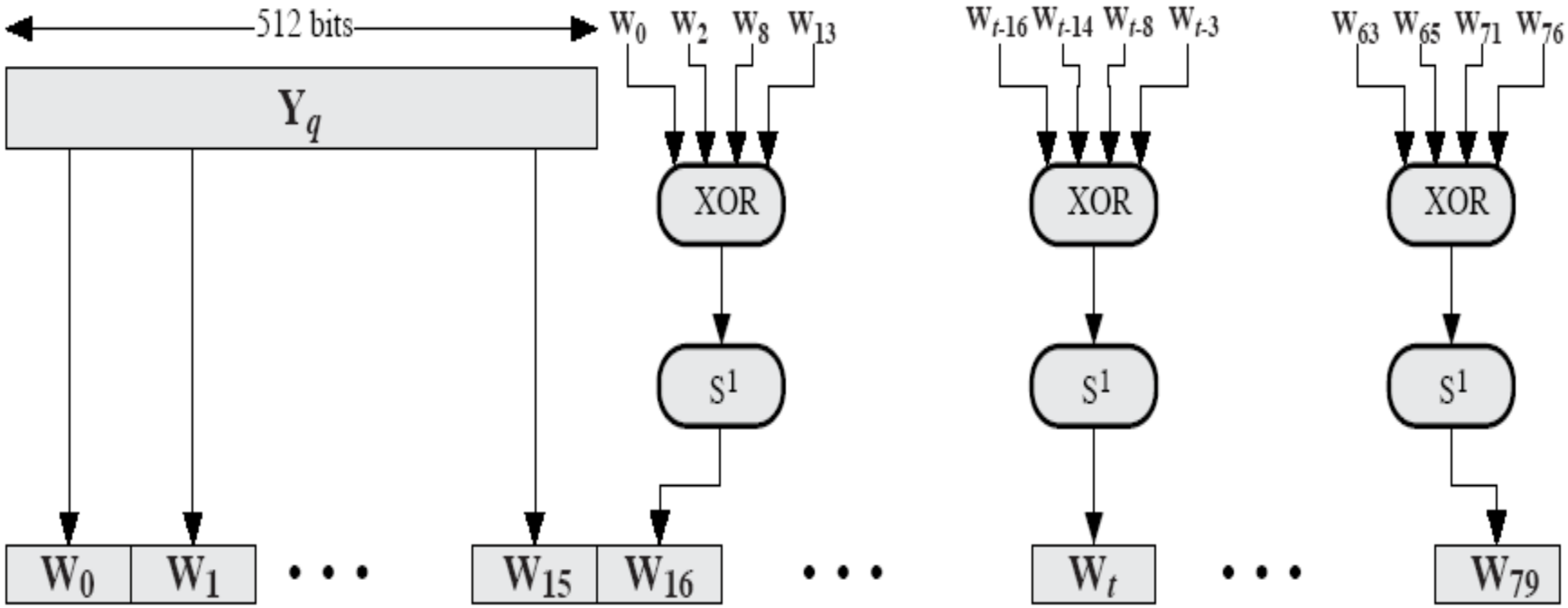
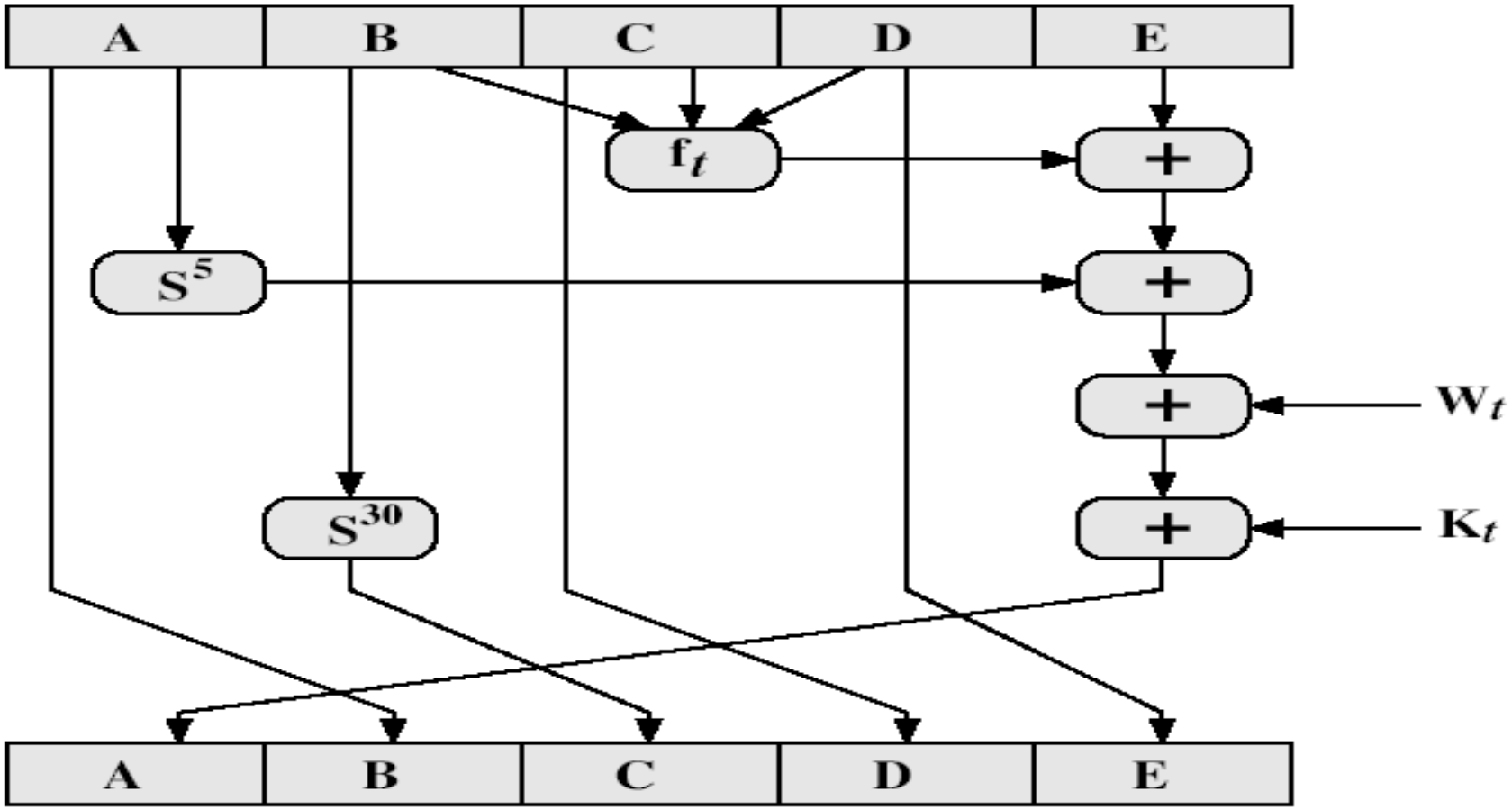


Figure 12.7 Creation of 80-word Input Sequence for SHA-1 Processing of Single Block

SHA-1 Compression Function



SHA-1 Compression Function

- each round has 20 steps which replaces the 5 buffer words thus:

  $(A,B,C,D,E) \leftarrow (E+f(t,B,C,D)+S^5(A)+W_t+K_t),A,S^{30}(B),C,D$

- a,b,c,d refer to the 4 words of the buffer
- t is the step number
- f(t,B,C,D) is nonlinear function for round
- W_t is derived from the message block
- K_t is an additive constant
- + addition modulo 2^{32}

Other Secure HASH functions

Table 3.1 A Comparison of Secure Hash Functions

	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	∞	$2^{64} - 1$ bits	∞
Primitive logical functions	4	4	5
Additive constants used	64	4	9

HMAC

- Use a MAC derived from a cryptographic hash code, such as SHA-1.
- **Motivations:**
  - Cryptographic hash functions executes faster in software than encryptoin algorithms such as DES
  - Library code for cryptographic hash functions is widely available
  - No export restrictions from the US

A **cryptographic hash function:** **hash function** which takes an input (or 'message') and returns a fixed-size alphanumeric string, which is called the **hash** value (sometimes called a message digest, a digital fingerprint, a digest or a checksum).

HMAC Design Objectives

- To use without modifications, available hash functions-in particular, hash functions that perform well in software, and for which code is freely and widely available.

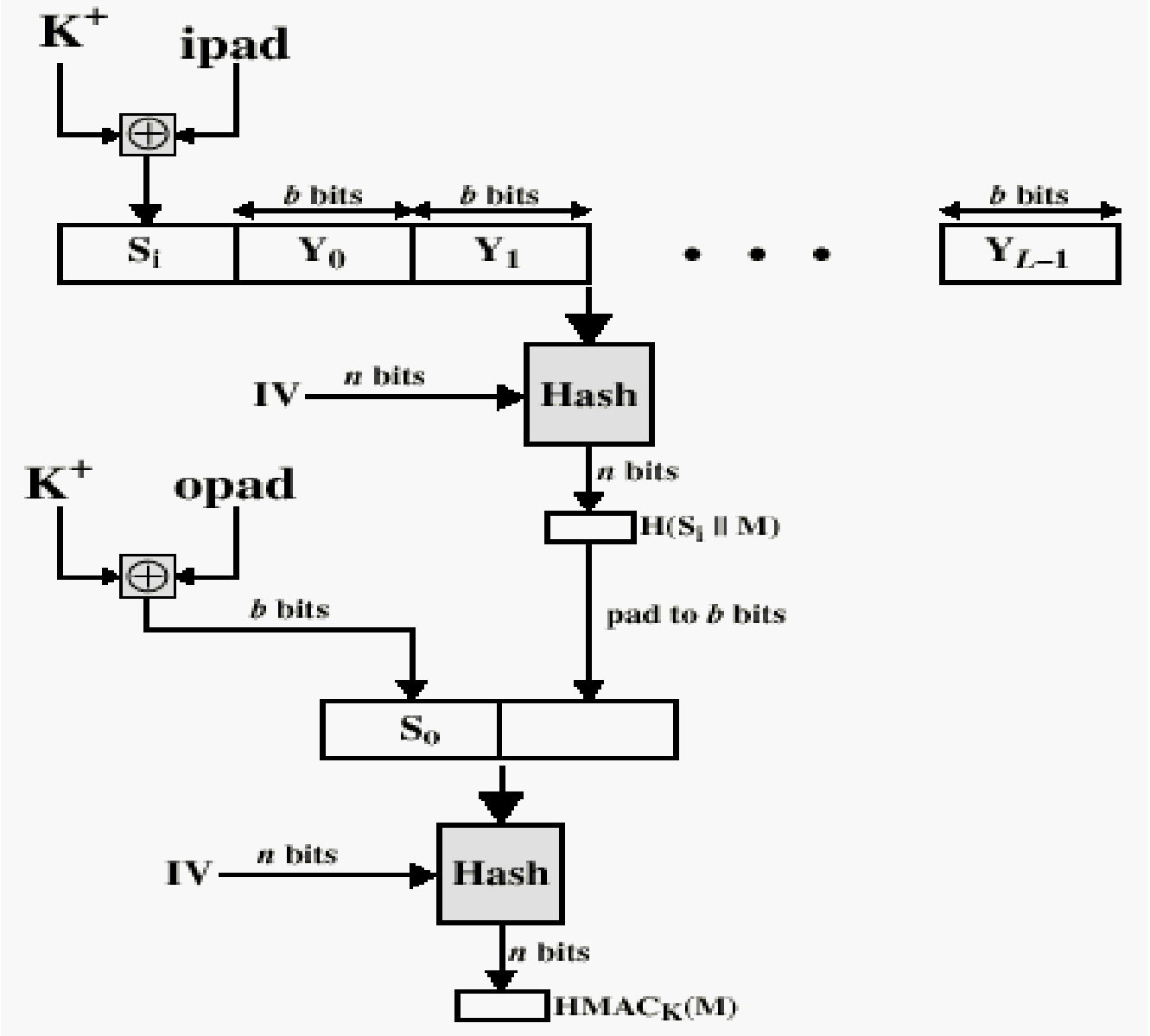
- To allow for easy replaceability of the embedded hash functions in case faster or more secure hash functions are found or required.

- To preserve the original performance of the hash functions without incurring a significant degradation.

- To use and handle keys in a simple way.

- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about embedded hash function.

HMAC Structure



H = embedded hash function (e.g., SHA-1)

IV= initial value input to hash function

M = message input to HMAC (including the padding specified in the embedded hash function)

Y_i = i th block of M, $0 \leq i \leq (L-1)$

L = number of blocks in M

b = number of bits in a block

n = length of hash code produced by embedded hash function

K = secret key; if key>b;the key is input to hash fun to produce an n-bit key:recommended length is >=n.

K^+ = K padded with zeros on the left so that the result is b bits in length

ipad = 00110110 (36 in hexadecimal) repeated b/8 times

opad = 01011100 (5C in hexadecimal) repeated b/8 times

Then HMAC can be expressed as follows:

- $HMAC_k = H[(K^+ \oplus opad) \,||\, H[(K^+ \oplus ipad) \,||\, M]]$

- **HMAC Algorithm**

1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and b=512, then K will be appended with 44 zero bytes 0x00).
2. XOR K^+ with ipad to produce the b-bit block s_i
3. Append M to s_i
4. Apply H to the stream generated in step3
5. XOR K^+ with opad to produce the b-bit block s_0
6. Append the hash result from step4 to s_0
7. Apply H to the stream generated in step6 and output the result.