Course: Cryptography and Network Security
Code: CS-34310
Branch: M.C.A - 4th Semester

Lecture – 16 : Data Encryption Standard (DES) and Advanced Encryption Standard (AES)

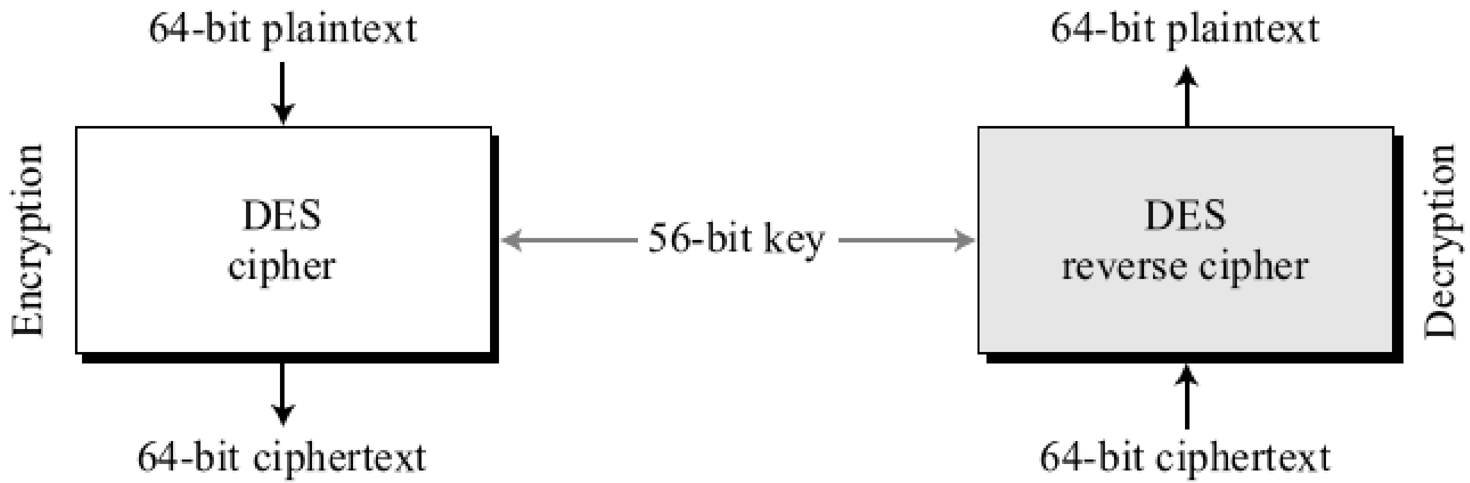
Faculty & Coordinator : Dr. J Sathish Kumar (JSK)

Department of Computer Science and Engineering

Motilal Nehru National Institute of Technology Allahabad,
Prayagraj-211004

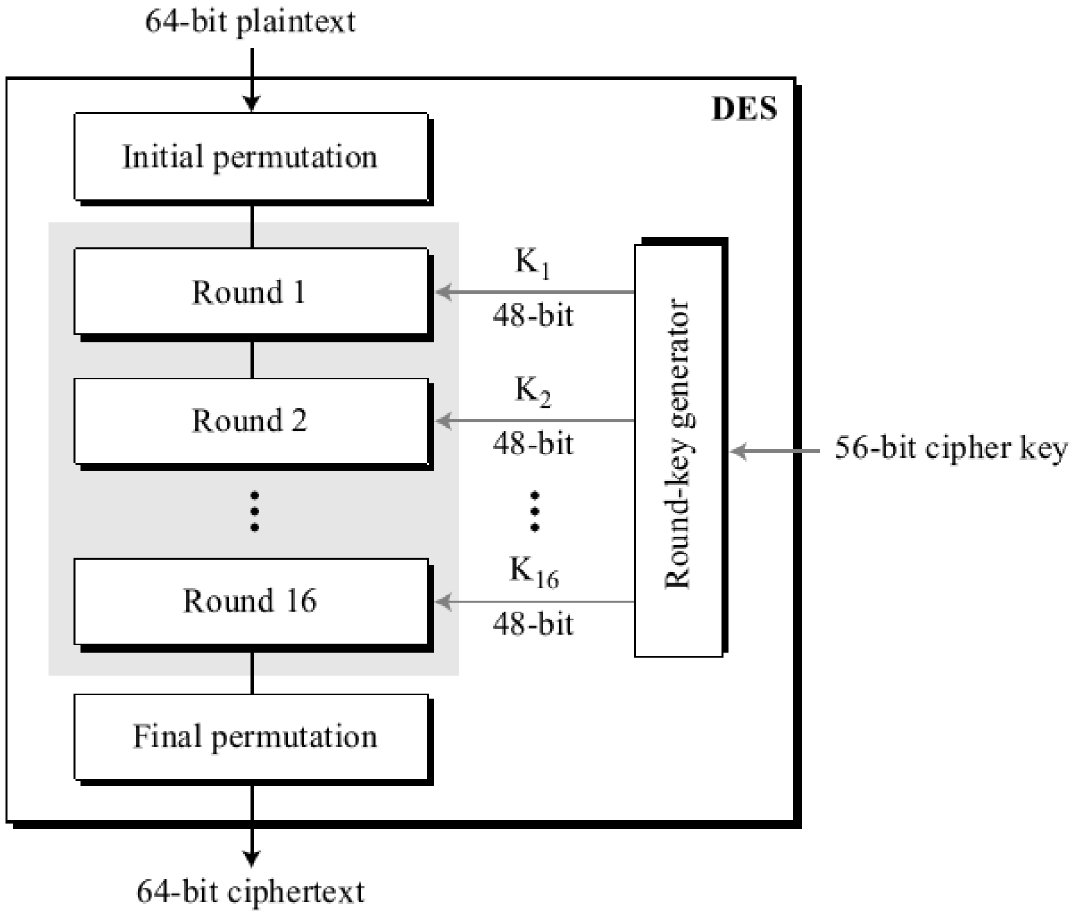
INTRODUCTION

- The **Data Encryption Standard (DES)** is a symmetric-key block cipher published by the **National Institute of Standards and Technology (NIST).**
- DES is a block cipher

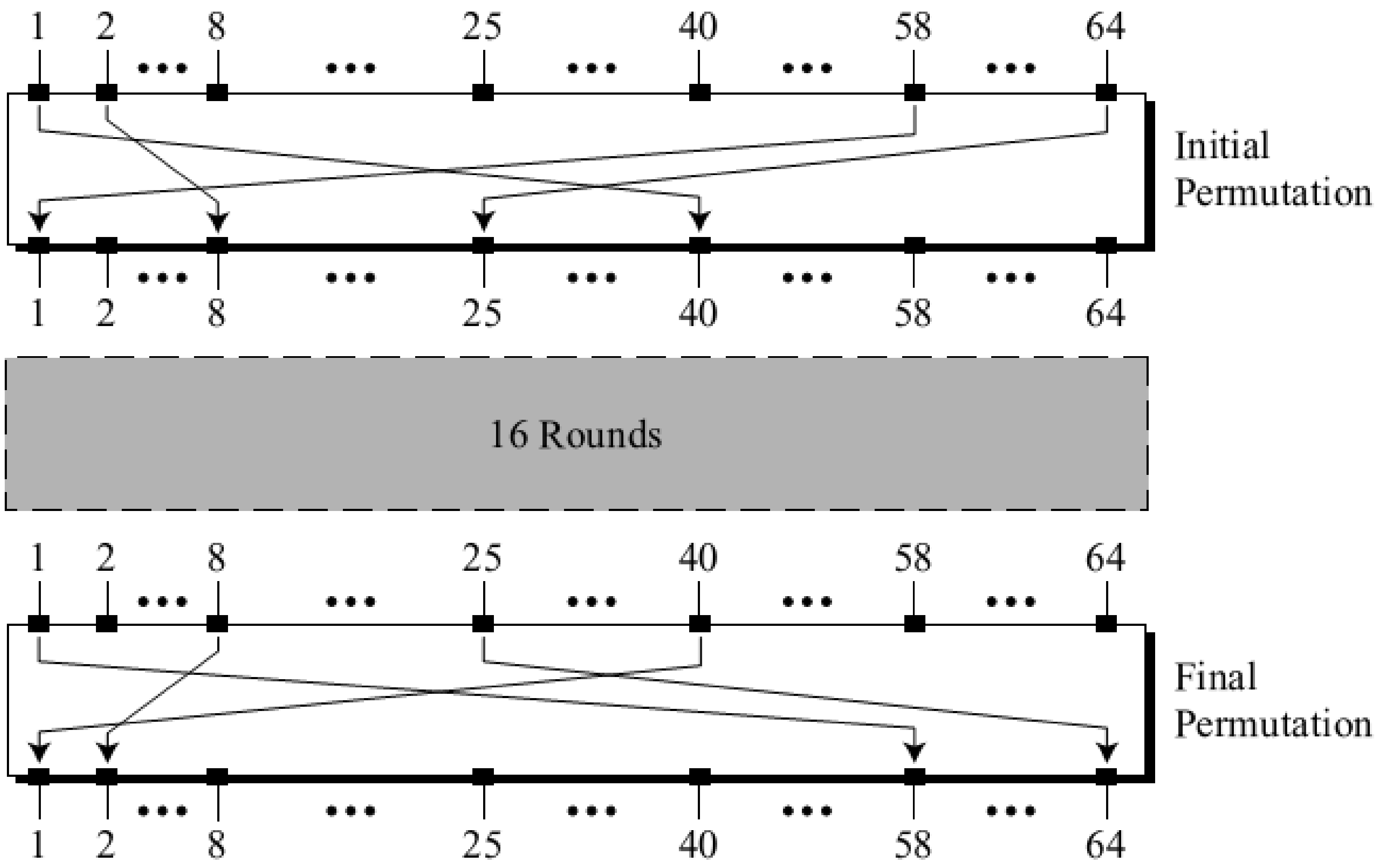


DES STRUCTURE

- The encryption process is made of two permutations (P-boxes), which we call initial and final permutations, and sixteen Feistel rounds.
- Each round uses a different 48-bit round key generated from the cipher key according to a predefined algorithm
- The initial and final permutations are straight P-boxes that are inverses of each other.
- They have no cryptography significance in DES.



Initial and Final Permutations

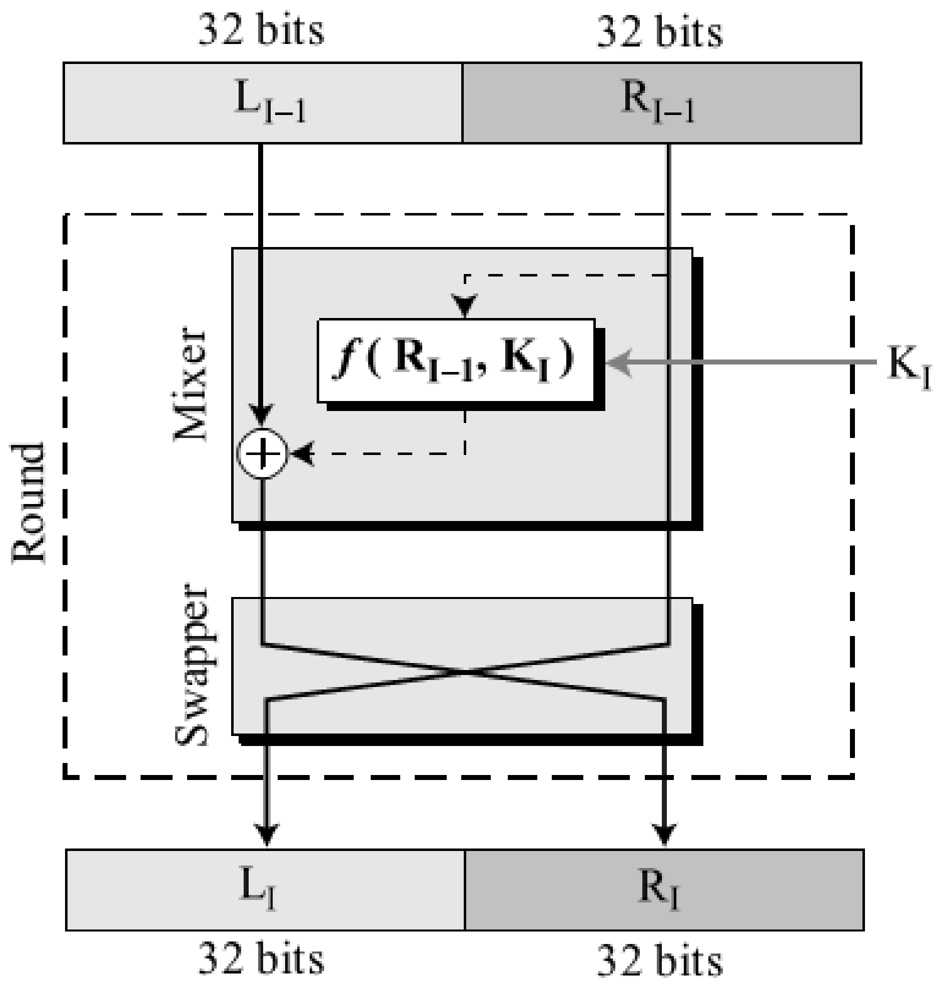


Initial and final permutation tables

Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

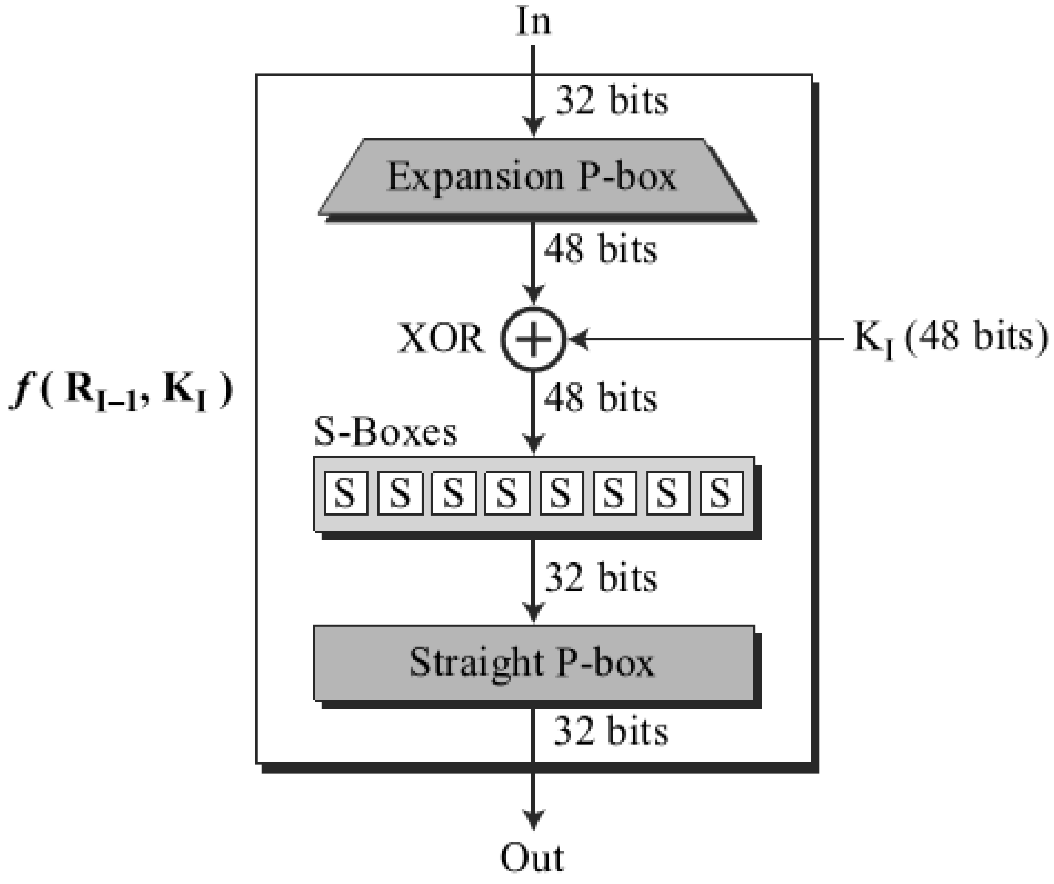
Rounds

- DES uses 16 rounds.
- Each round of DES is a Feistel cipher
- The round takes L_{I-1} and R_{I-1} from previous round (or the initial permutation box) and creates L_I and R_I, which go to the next round (or final permutation box).
- The mixer is invertible because of the XOR operation.



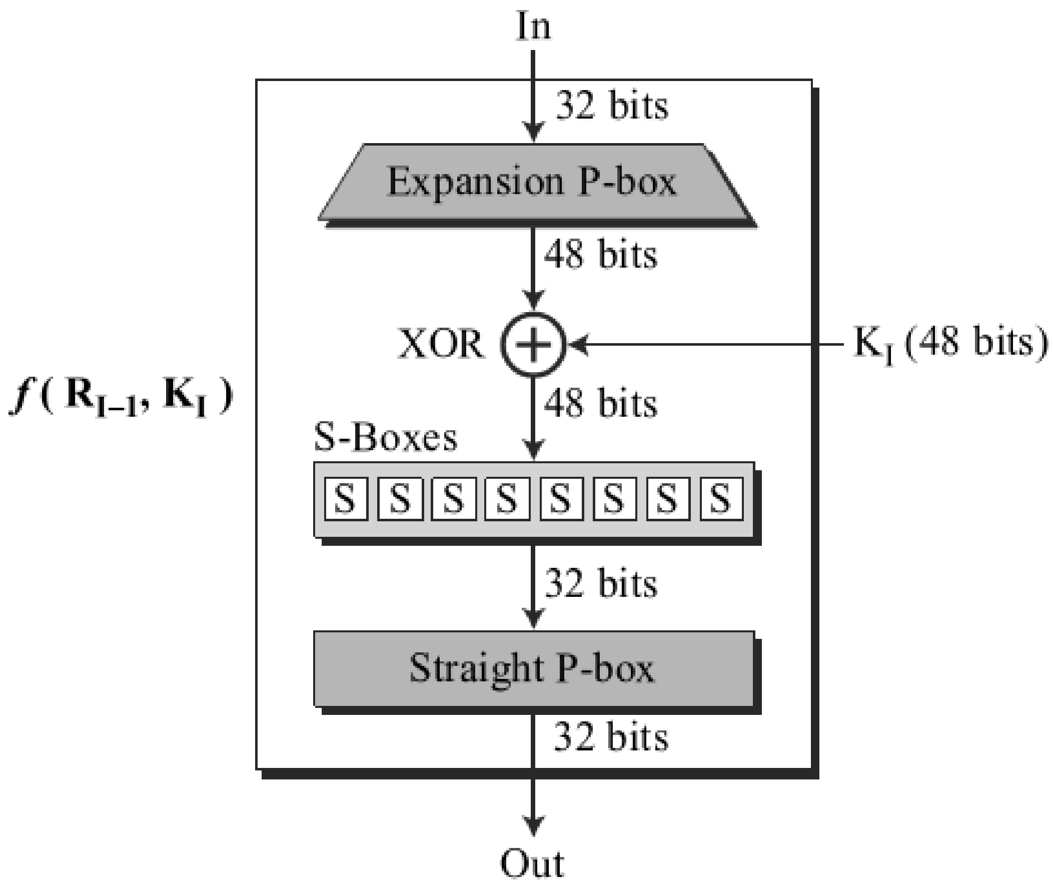
DES Function

- The heart of DES is the DES function.
- The DES function applies a 48-bit key to the rightmost 32 bits (R_{I-1}) to produce a 32-bit output.
- This function is made up of four sections:
  - an expansion P-box,
  - a whitener (that adds key),
  - a group of S-boxes, and
  - a straight P-box

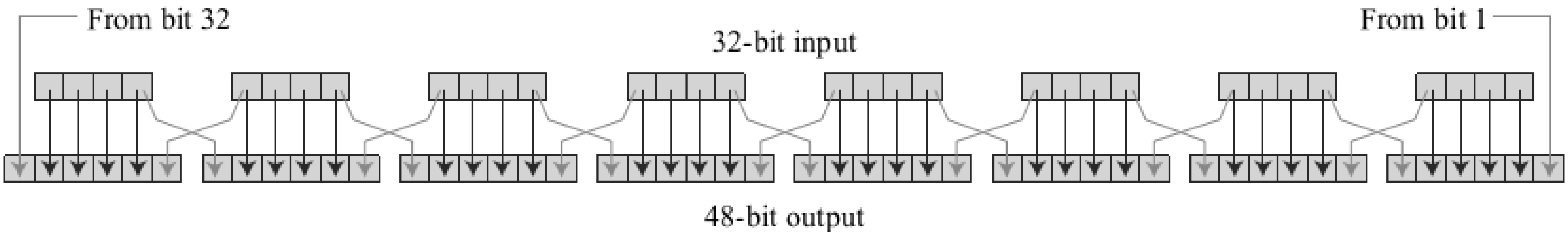


DES Function

- The heart of DES is the DES function.
- The DES function applies a 48-bit key to the rightmost 32 bits (R_{I-1}) to produce a 32-bit output.
- This function is made up of four sections:
  - an expansion P-box,
  - a whitener (that adds key),
  - a group of S-boxes, and
  - a straight P-box



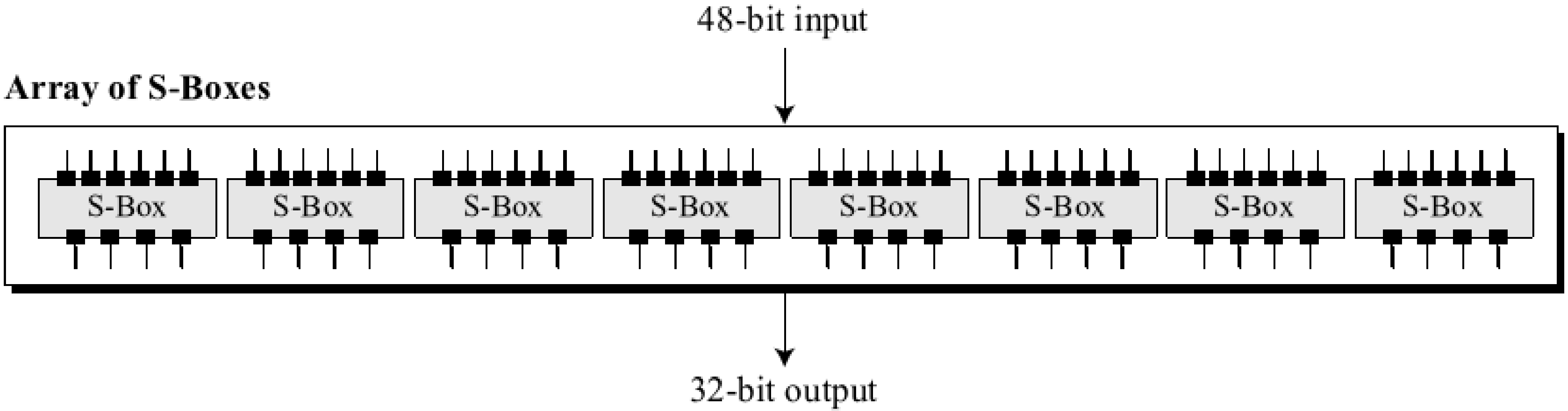
Expansion permutation and Expansion P-box table



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	30	31	32	01

S-Boxes

The S-boxes do the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output.



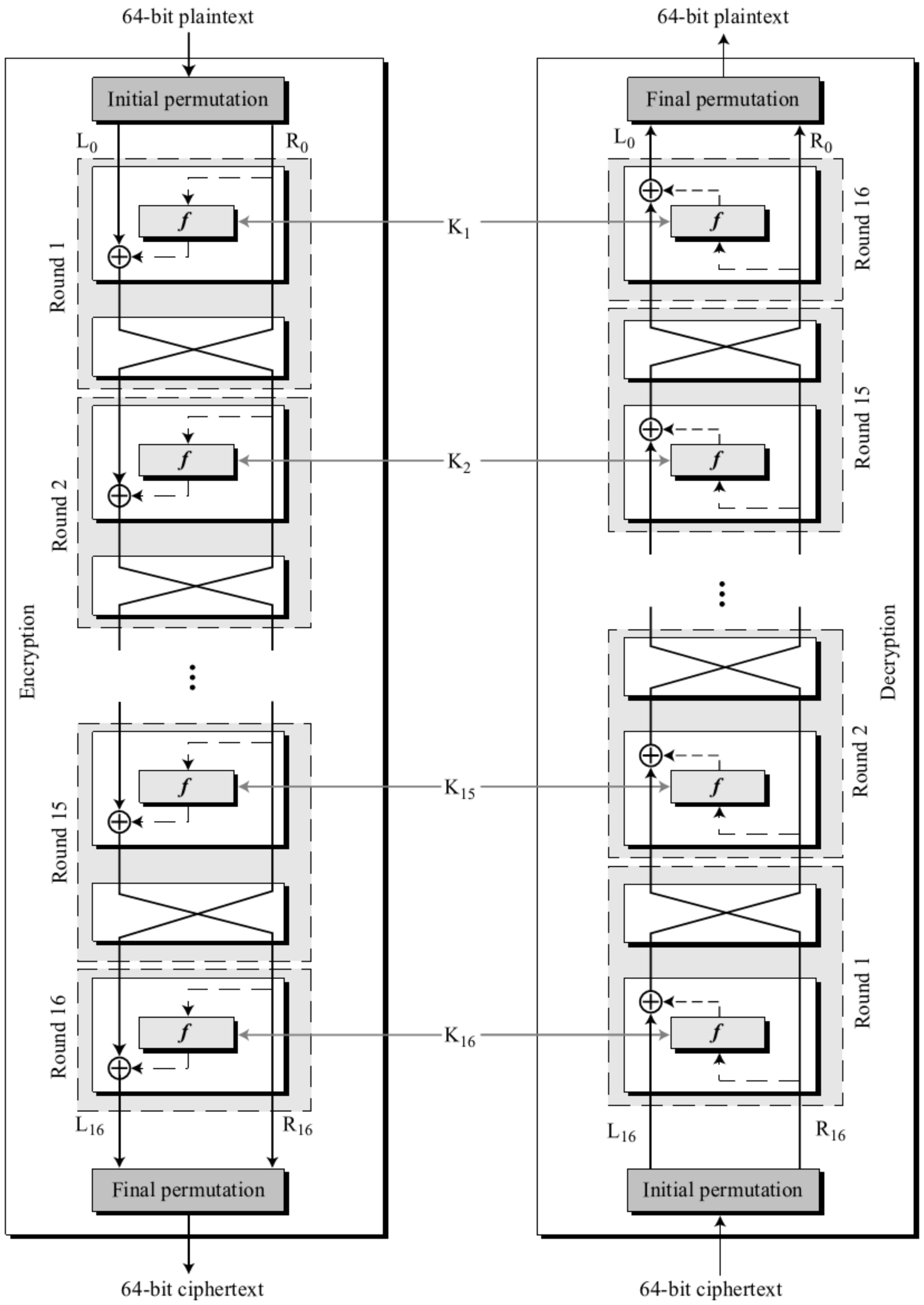
Straight Permutation

- The last operation in the DES function is a straight permutation with a 32-bit input and a 32-bit output
- The seventh bit of the input becomes the second bit of the output

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Cipher and Reverse Cipher

- Using mixers and swappers, we can create the cipher and reverse cipher, each having 16 rounds.
- The cipher is used at the encryption site; the reverse cipher is used at the decryption site.
- The whole idea is to make the cipher and the reverse cipher algorithms similar.
- Many Approaches to Implement DES



DES cipher and reverse cipher for the first approach

Pseudocode for DES cipher

```
Cipher (plainBlock[64], RoundKeys[16, 48], cipherBlock[64])
{
    permute (64, 64, plainBlock, inBlock, InitialPermutationTable)
    split (64, 32, inBlock, leftBlock, rightBlock)
    for (round = 1 to 16)
    {
        mixer (leftBlock, rightBlock, RoundKeys[round])
        if (round!=16) swapper (leftBlock, rightBlock)
    }
    combine (32, 64, leftBlock, rightBlock, outBlock)
    permute (64, 64, outBlock, cipherBlock, FinalPermutationTable)
}
```

Pseudocode for DES cipher

```
mixer (leftBlock[32], rightBlock[32], RoundKey[48])
{
    copy (32, rightBlock, T1)
    function (T1, RoundKey, T2)
    exclusiveOr (32, leftBlock, T2, T3)
    copy (32, T3, rightBlock)
}
```

```
swapper (leftBlock[32], rigthBlock[32])
{
    copy (32, leftBlock, T)
    copy (32, rightBlock, leftBlock)
    copy (32, T, rightBlock)
}
```

Pseudocode for DES cipher

```
function (inBlock[32], RoundKey[48], outBlock[32])
{
    permute (32, 48, inBlock, T1, ExpansionPermutationTable)
    exclusiveOr (48, T1, RoundKey, T2)
    substitute (T2, T3, SubstituteTables)
    permute (32, 32, T3, outBlock, StraightPermutationTable)
}
```

Pseudocode for DES cipher

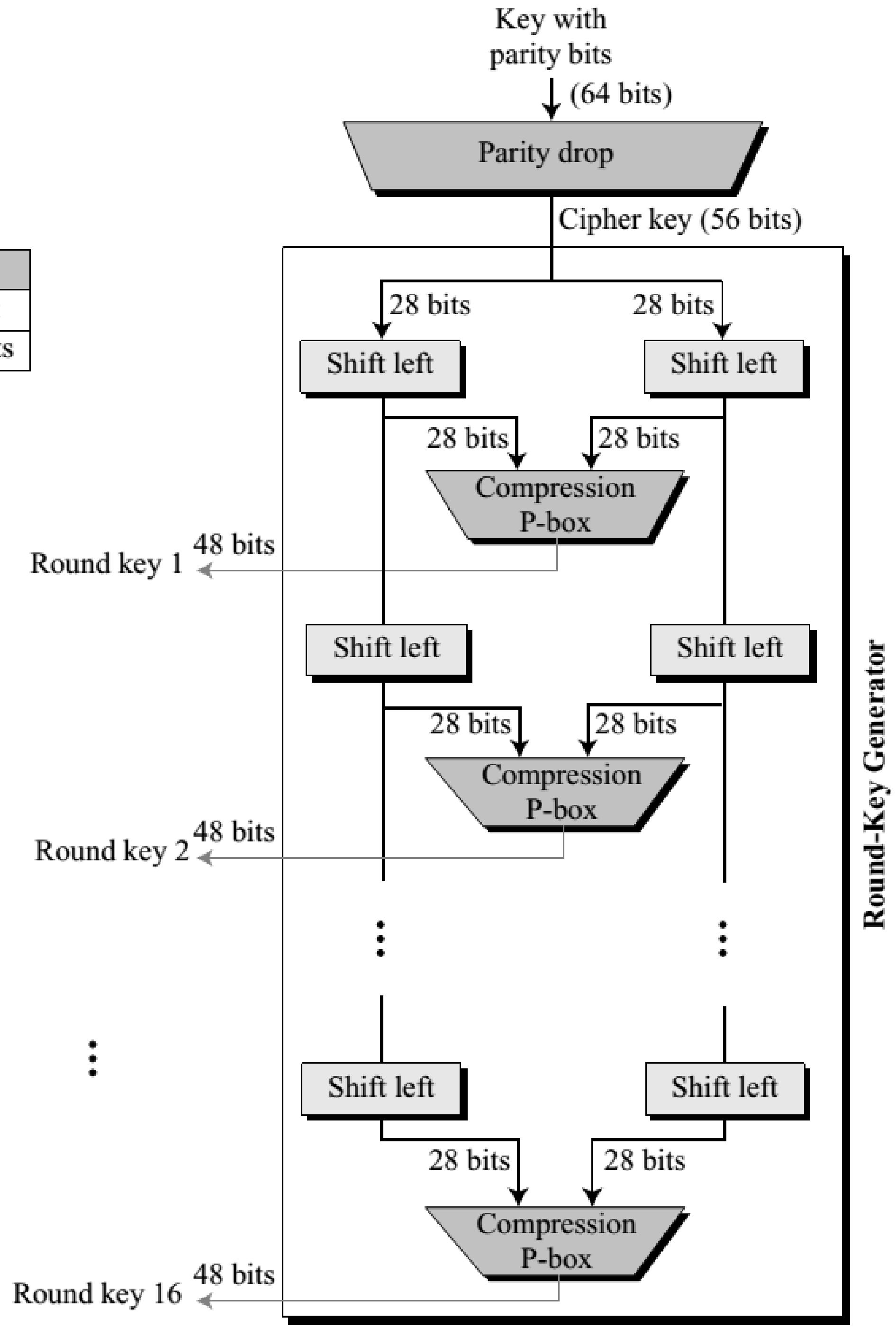
```
substitute (inBlock[32], outBlock[48], SubstitutionTables[8, 4, 16])
{
    for (i = 1 to 8)
    {
        row ← 2 × inBlock[i × 6 + 1] + inBlock [i × 6 + 6]
        col ← 8 × inBlock[i × 6 + 2] + 4 × inBlock[i × 6 + 3] +
                2 × inBlock[i × 6 + 4] + inBlock[i × 6 + 5]

        value = SubstitutionTables [i][row][col]


        outBlock[[i × 4 + 1] ← value / 8;        value ← value mod 8
        outBlock[[i × 4 + 2] ← value / 4;        value ← value mod 4
        outBlock[[i × 4 + 3] ← value / 2;        value ← value mod 2
        outBlock[[i × 4 + 4] ← value
    }
}
```

Key Generation

- The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key.
- However, the cipher key is normally given as a 64-bit key in which 8 extra bits are the parity bits, which are dropped before the actual key-generation process.

Shifting

Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits



Algorithm for round-keys generation

```
Key_Generator (keyWithParities[64], RoundKeys[16, 48], ShiftTable[16])
{
    permute (64, 56, keyWithParities, cipherKey, ParityDropTable)
    split (56, 28, cipherKey, leftKey, rightKey)
    for (round = 1 to 16)
    {
        shiftLeft (leftKey, ShiftTable[round])
        shiftLeft (rightKey, ShiftTable[round])
        combine (28, 56, leftKey, rightKey, preRoundKey)
        permute (56, 48, preRoundKey, RoundKeys[round], KeyCompressionTable)
    }
}
```

DES ANALYSIS

- Critics have used a strong magnifier to analyze DES.
- Tests have been done to measure the strength of some desired properties in a block cipher.
- Properties
  - Two desired properties of a block cipher are the avalanche effect and the completeness.
- Avalanche Effect
  - Avalanche effect means a small change in the plaintext (or key) should create a significant change in the ciphertext.
  - DES has been proved to be strong with regard to this property.
- Completeness effect
  - Completeness effect means that each bit of the ciphertext needs to depend on many bits on the plaintext.
  - The diffusion and confusion produced by P-boxes and S-boxes in DES, show a very strong completeness effect.

DES Weaknesses

- Weaknesses in Cipher Design
  - S-boxes
    - In S-box 4, the last three output bits can be derived in the same way as the first output bit by complementing some of the input bits.
    - Two specifically chosen inputs to an S-box array can create the same output.
    - It is possible to obtain the same output in a single round by changing bits in only three neighboring S-boxes.
  - P-boxes
    - It is not clear why the designers of DES used the initial and final permutations; these have no security benefits.
    - In the expansion permutation (inside the function), the first and fourth bits of every 4-bit series are repeated.

DES Weaknesses

- Weakness in the Cipher Key
  - DES is in its key size (56 bits). To do a brute-force attack on a given ciphertext block, the adversary needs to check 2^{56} keys.
  - With available technology, it is possible to check one million keys per second.
  - If we can make a computer with one million chips (parallel processing), then we can test the whole key domain in approximately 20 hours.
  - Computer networks can simulate parallel processing. In 1977 a team of researchers used 3500 computers attached to the Internet to find a key challenged by RSA Laboratories in 120 days. The key domain was divided among all of these computers, and each computer was responsible to check the part of the domain.
  - If 3500 networked computers can find the key in 120 days, a secret society with 42,000 members can find the key in 10 days

DES Weaknesses

- Weak Keys
  - Four out of 2^{56} possible keys are called weak keys
  - The round keys created from any of these weak keys are the same and have thesame pattern as the cipher key.

Keys before parities drop (64 bits)	*Actual key (56 bits)*
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 0E0E 0E0E	0000000 FFFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

DES Weaknesses

- Semi-weak Keys
  - There are six key pairs that are called semi-weak keys.
  - A semi-weak key creates only two different round keys and each of them is repeated eight times.

First key in the pair	*Second key in the pair*
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E1 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

DES Weaknesses

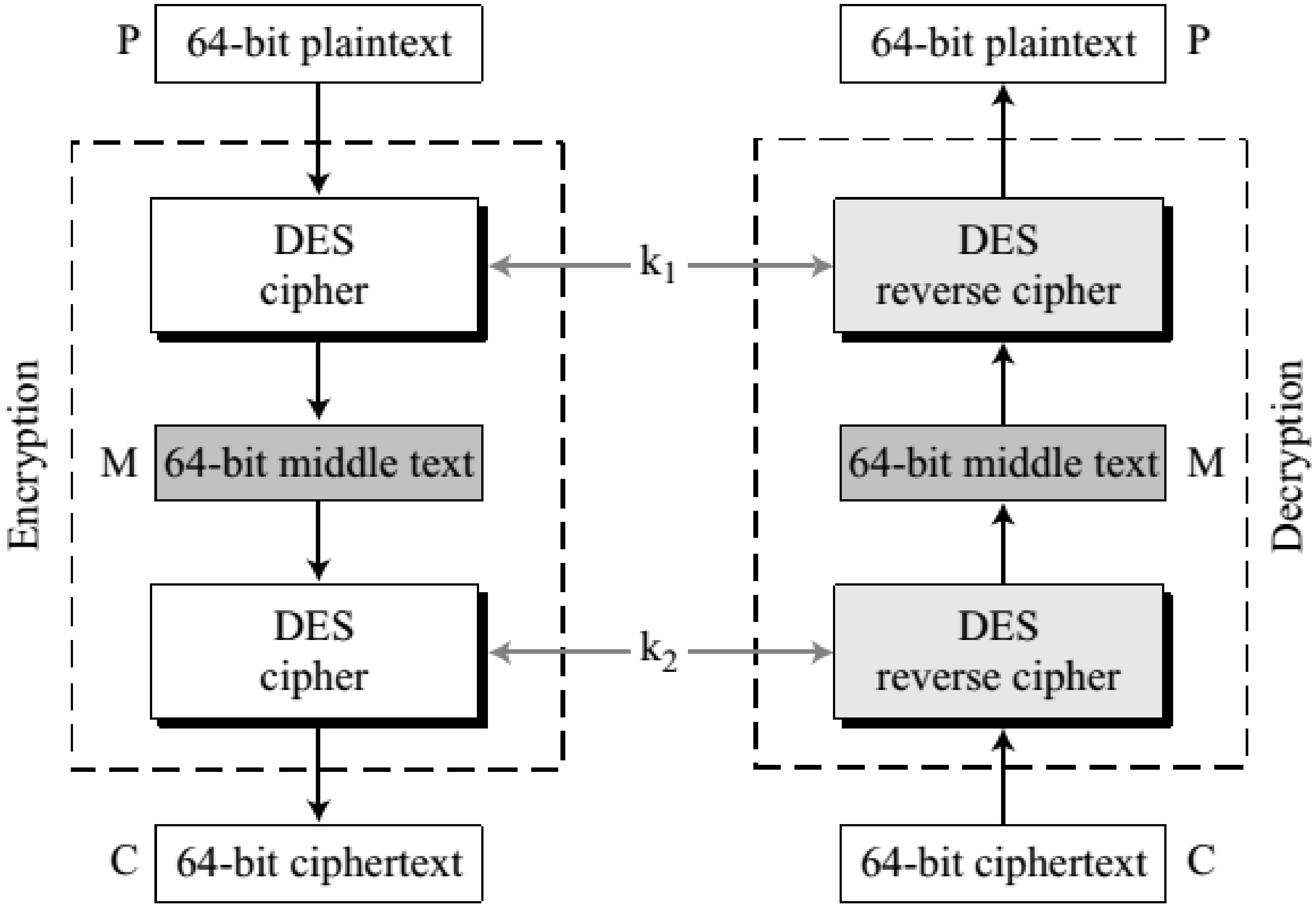
- Semi-weak Keys
  - There are six key pairs that are called semi-weak keys.
  - A semi-weak key creates only two different round keys and each of them is repeated eight times.

First key in the pair	*Second key in the pair*
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E1 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

MULTIPLE DES

- The major criticism of DES regards its key length.
- With available technology and the possibility of parallel processing, a brute-force attack on DES is feasible.
- One solution to improve the security of DES is to abandon DES and design a new cipher, with the advent of AES.
- The second solution is to use multiple (cascaded) instances of DES with multiple keys
    - Double DES
    - Triple DES

Double DES


P
64-bit plaintext
DES
cipher
M
64-bit middle text
DES
cipher
C
64-bit ciphertext
Encryption
k1
k2
64-bit plaintext
P
DES
reverse cipher
64-bit middle text
M
DES
reverse cipher
64-bit ciphertext
C
Decryption

Meet-in-the-Middle Attack

- Using a known-plaintext attack called meet-in-the-middle attack proves that double DES improves this vulnerability slightly (to 2^{57} tests), but not tremendously (to 2^{112}).

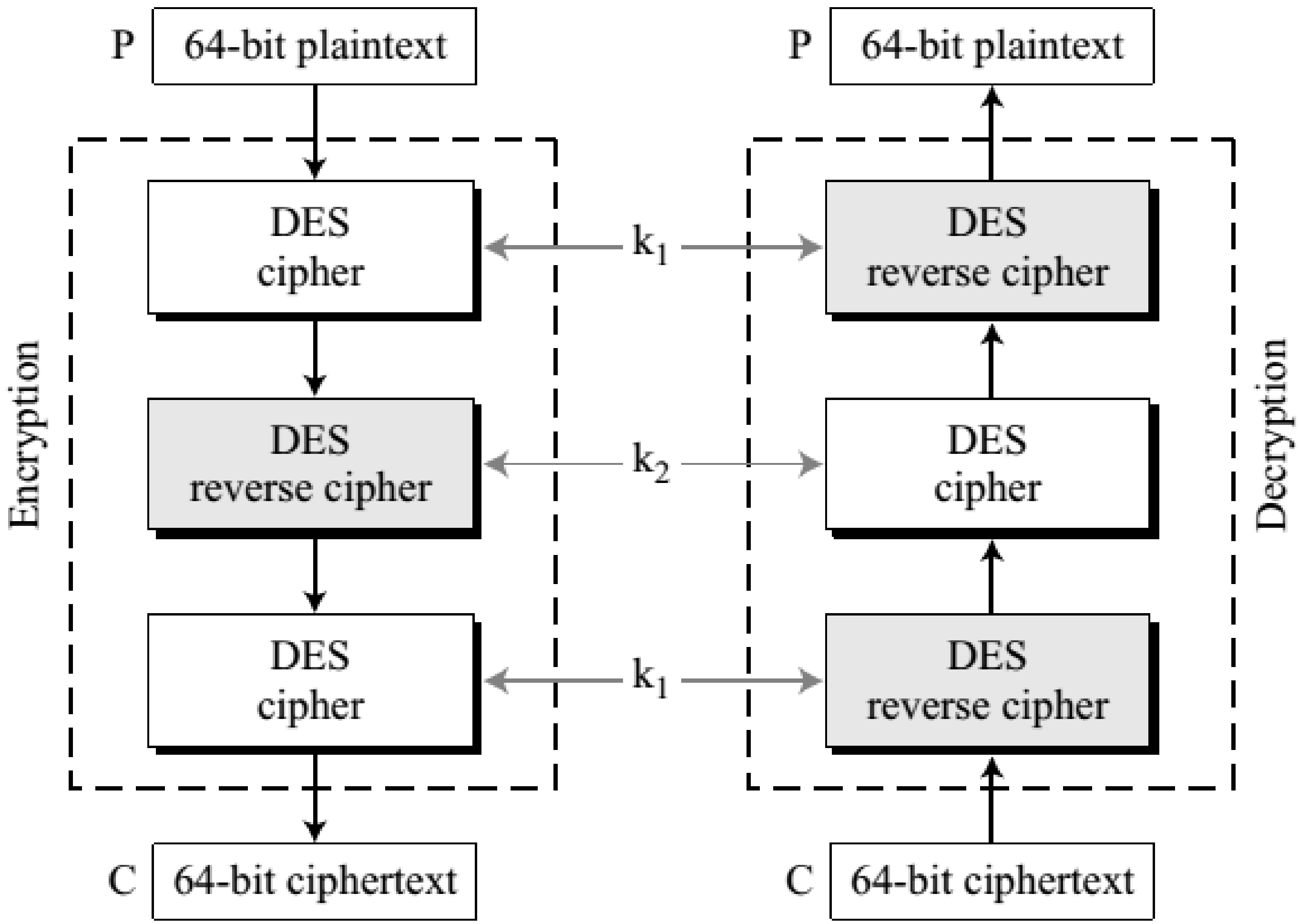
$$M = E_{k_1}(P) \qquad \text{and} \qquad M = D_{k_2}(C)$$

- Eve encrypts P using all possible values (2^{56}) of k1 and records all values obtained for M.
- Eve decrypts C using all possible values (2^{56}) of k2 and records all values obtained for M.
- This means that instead of using 2^{112} key-search tests, Eve uses 2^{56} key-search tests two times

Triple DES

- To improve the security of DES, triple DES (3DES) was proposed.
- This uses three stages of DES for encryption and decryption.
- Two versions of triple DES are in use today:
  - triple DES with two keys and
  - triple DES with three keys.

Triple DES with two keys

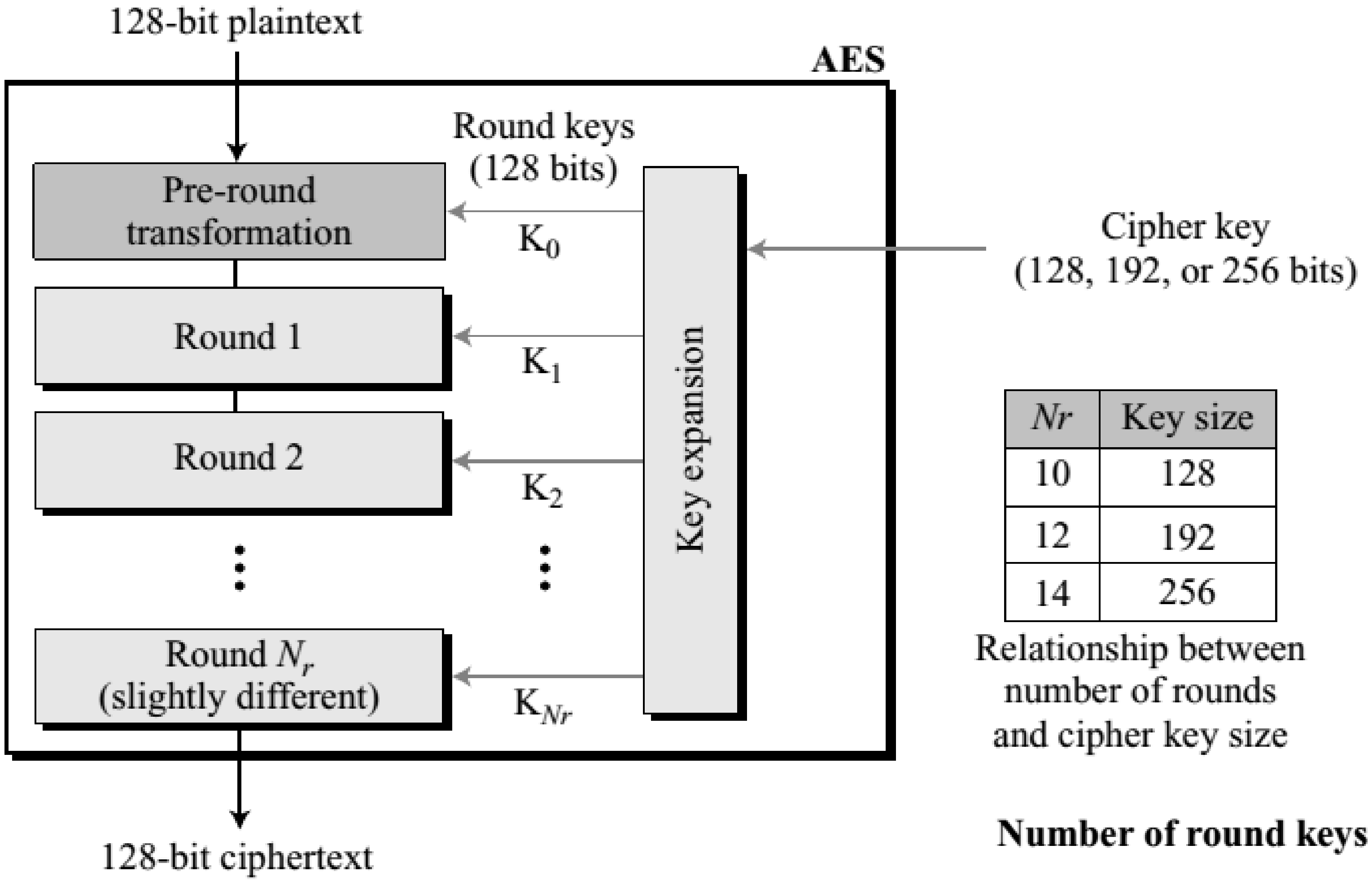


Advanced Encryption Standard (AES)

INTRODUCTION

- The Advanced Encryption Standard (AES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST) in December 2001.
- The criteria defined by NIST for selecting AES fall into three areas: security, cost, and implementation.
- AES has defined three versions, with 10, 12, and 14 rounds.
- Each version uses a different cipher key size (128, 192, or 256), but the round keys are always 128 bits.

General design of AES encryption cipher



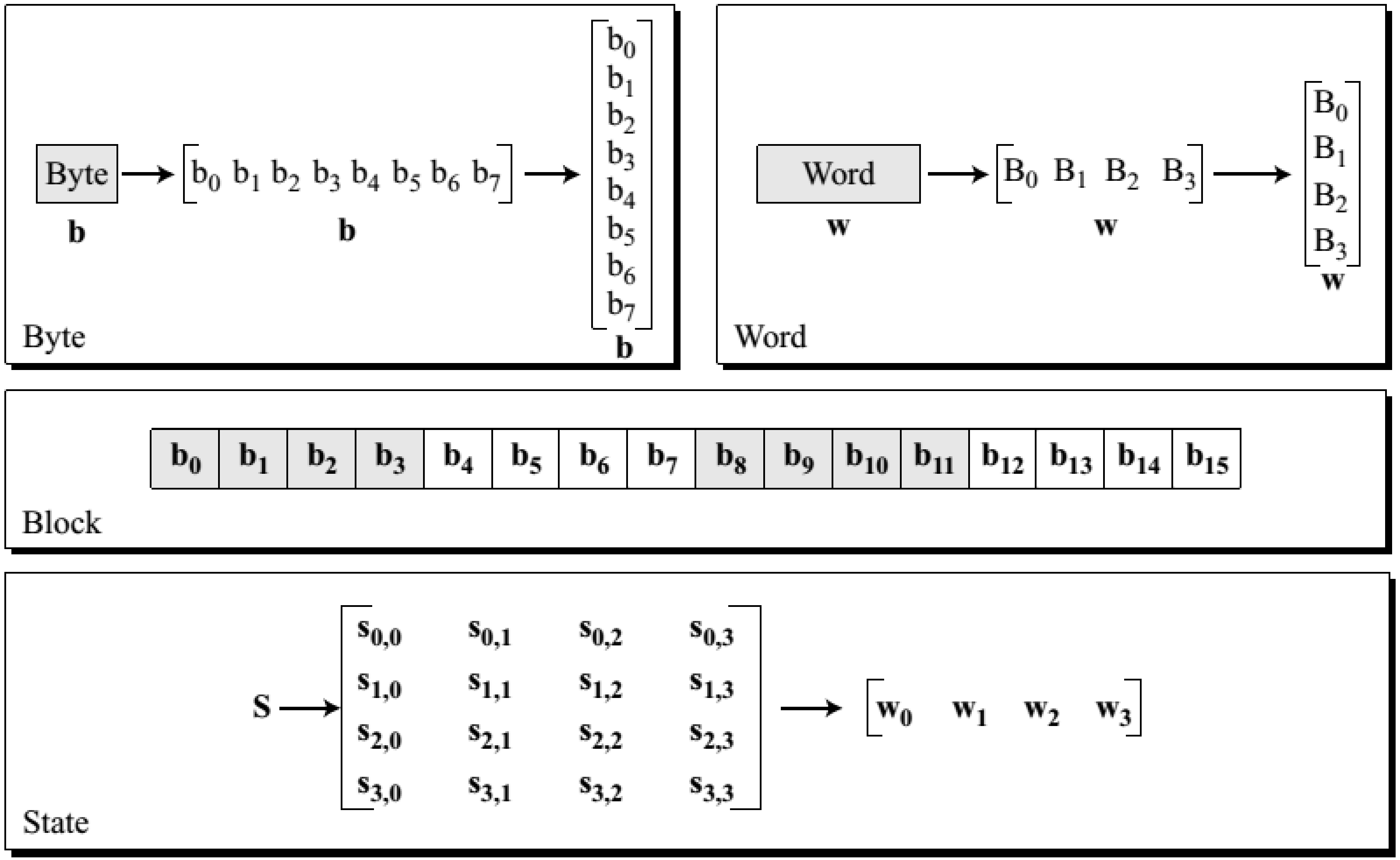
Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

Number of round keys = $N_r + 1$

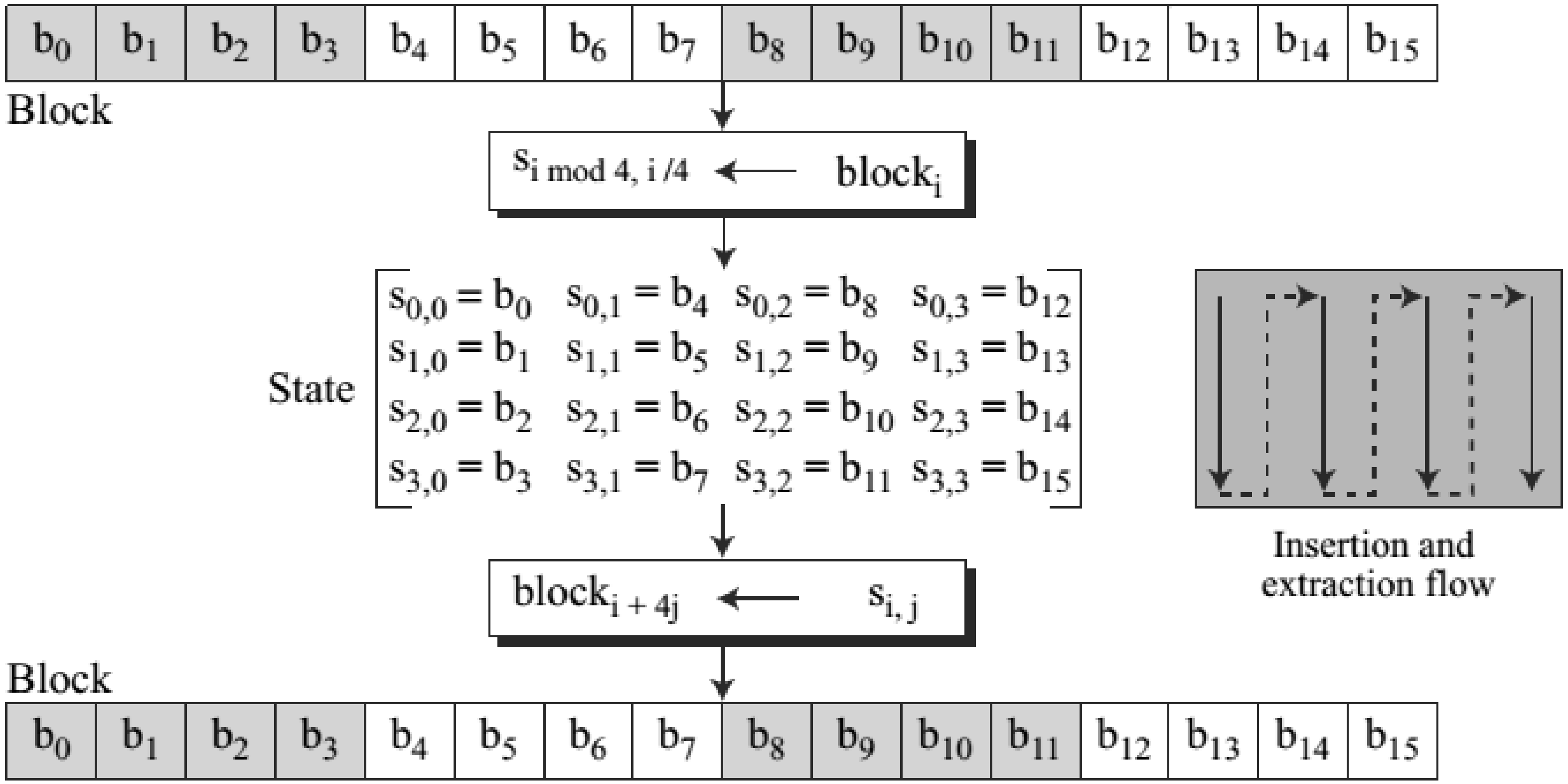
Data Units

- AES uses five units of measurement to refer to data: bits, bytes, words, blocks, and state.
- The bit is the smallest and atomic unit; other units can be expressed in terms of smaller ones.



Block-to-state and state-to-block transformation

- Although the states in different stages are normally called S, we occasionally use the letter T to refer to a temporary state.



Example #1

- Assume that the text block is “AES uses a matrix”.
- We add two bogus characters at the end to get “AESUSESAMATRIXZZ”

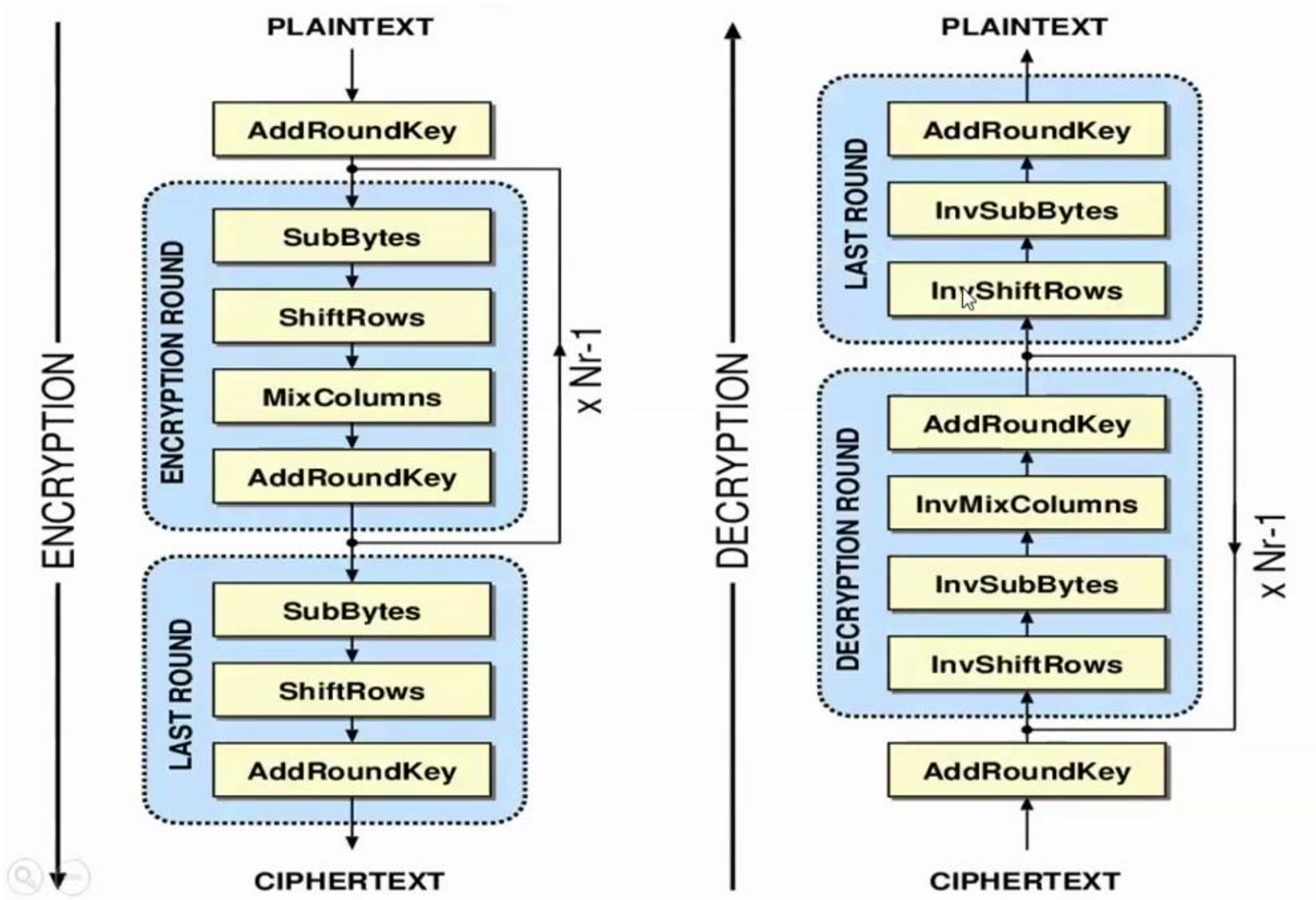
Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix} \text{State}$$

e

	DEC	HEX		DEC	HEX
A	00	00	N	13	0D
B	01	01	O	14	0E
C	02	02	P	15	0F
D	03	03	Q	16	10
E	04	04	R	17	11
F	05	05	S	18	12
G	06	06	T	19	13
H	07	07	U	20	14
I	08	08	V	21	15
J	09	09	W	22	16
K	10	0A	X	23	17
L	11	0B	Y	24	18
M	12	0C	Z	25	19

AES



ENCRYPTION
PLAINTEXT
AddRoundKey
ENCRYPTION ROUND
SubBytes
ShiftRows
MixColumns
AddRoundKey
x Nr-1
LAST ROUND
SubBytes
ShiftRows
AddRoundKey
CIPHERTEXT
DECRYPTION
PLAINTEXT
LAST ROUND
AddRoundKey
InvSubBytes
InvShiftRows
DECRYPTION ROUND
AddRoundKey
InvMixColumns
InvSubBytes
InvShiftRows
x Nr-1
AddRoundKey
CIPHERTEXT

TRANSFORMATIONS

- To provide security, AES uses four steps for each round:
  - substitution,
  - permutation,
  - mixing, and
  - key-adding.
- The above steps are also called AES Transformation function

Substitution

- First, the substitution is done for each byte.
- Second, only one table is used for transformation of every byte, which means that if two bytes are the same, the transformation is also the same.
- Third, the transformation is defined by either a table lookup process or mathematical calculation in the $GF(2^8)$ field.
- AES uses two invertible transformations.
  - SubBytes
  - InvSubBytes

Substitution :SubBytes

- The SubBytes operation involves 16 independent byte-to-byte transformations.
- Substitution table (S-box) for SubBytes transformation
- For example, two bytes, $5A_{16}$ and $5B_{16}$, which differ only in one bit (the rightmost bit) are transformed to BE_{16} and 39_{16}, which differ in four bits.

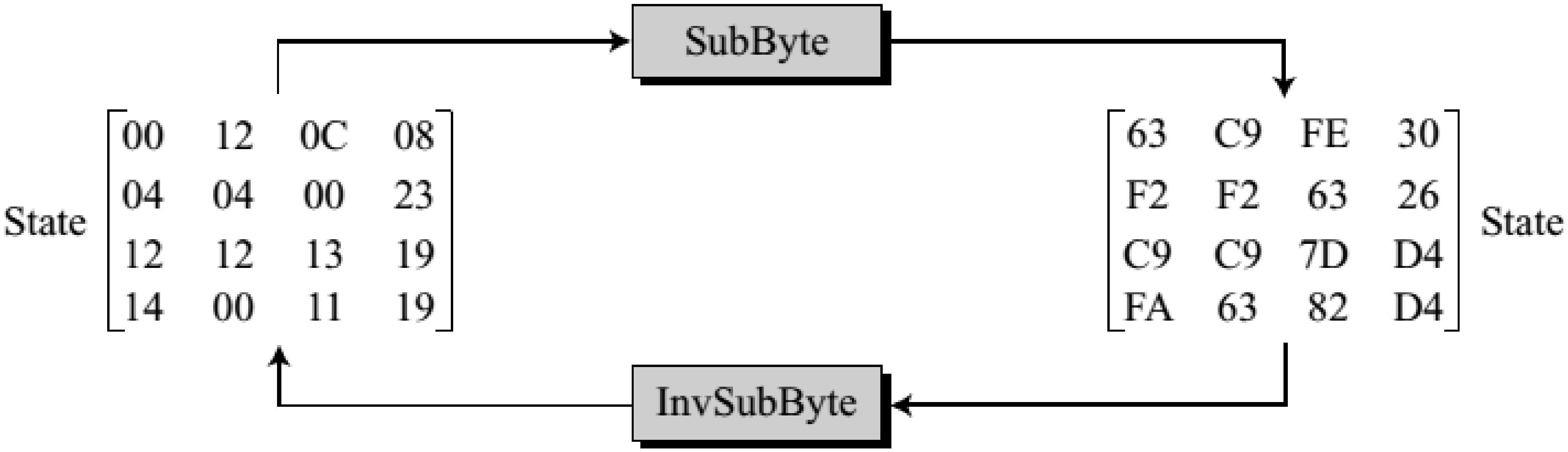
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	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	CB	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

Substitution :InvSubBytes

- InvSubBytes is the inverse of SubBytes

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

SubBytes transformation for Example #2



If the two bytes have the same values, their transformation is also the same.
For example, the two bytes 04_{16} and 04_{16} in the left state are transformed to $F2_{16}$ and $F2_{16}$ in the right state and vice versa.

Pseudocode for SubBytes transformation

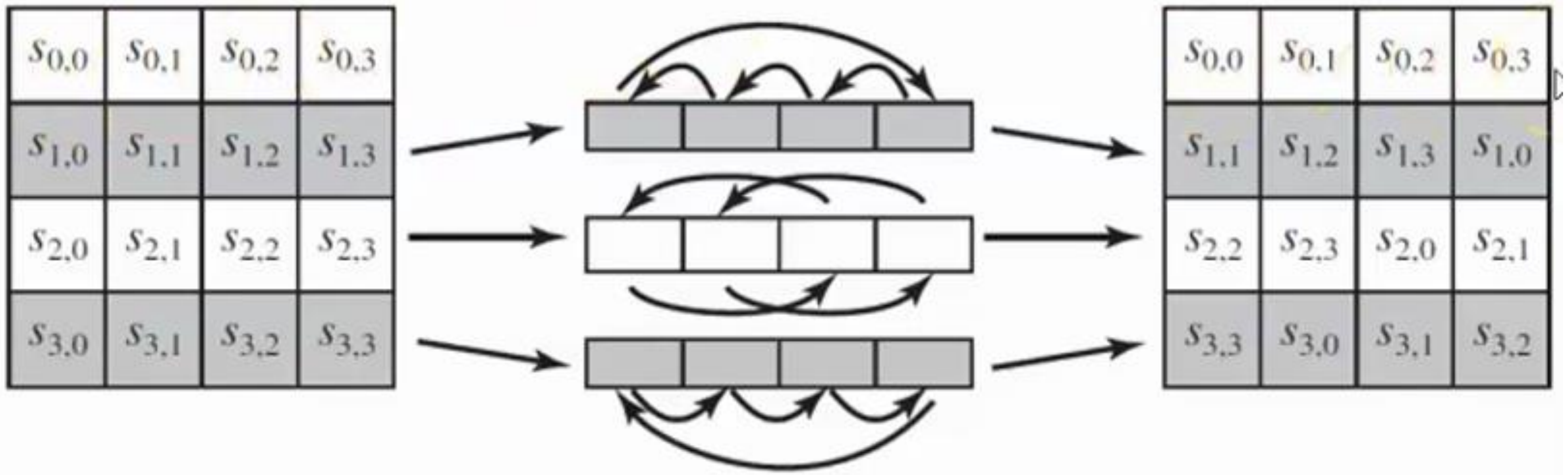
```
SubBytes (S)
{
   for (r = 0 to 3)
     for (c = 0 to 3)
            S_{r,c} = subbyte (S_{r,c})
}
```

```
subbyte (byte)
{
   a ← byte^-1                // Multiplicative inverse in GF(2^8) with inverse of 00 to be 00
    ByteToMatrix (a, b)
    for (i = 0 to 7)
   {
        c_i ← b_i ⊕ b_(i+4)mod 8 ⊕ b_(i+5)mod 8 ⊕ b_(i+6)mod 8 ⊕ b_(i+7)mod 8
        d_i ← c_i ⊕ ByteToMatrix (0x63)
   }
    MatrixToByte (d, d)
    byte ← d
}
```

Permutation

- Another transformation found in a round is shifting, which permutes the bytes.
- Unlike DES, in which permutation is done at the bit level, shifting transformation in AES is done at the byte level; the order of the bits in the byte is not changed.
- The ShiftRows and InvShiftRows transformations are inverses of each other

- Rules of shifting rows,
  - Row 1 → No Shifting
  - Row 2 → 1 byte left shift
  - Row 3 → 2 byte left shift
  - Row 4 → 3 byte left shift



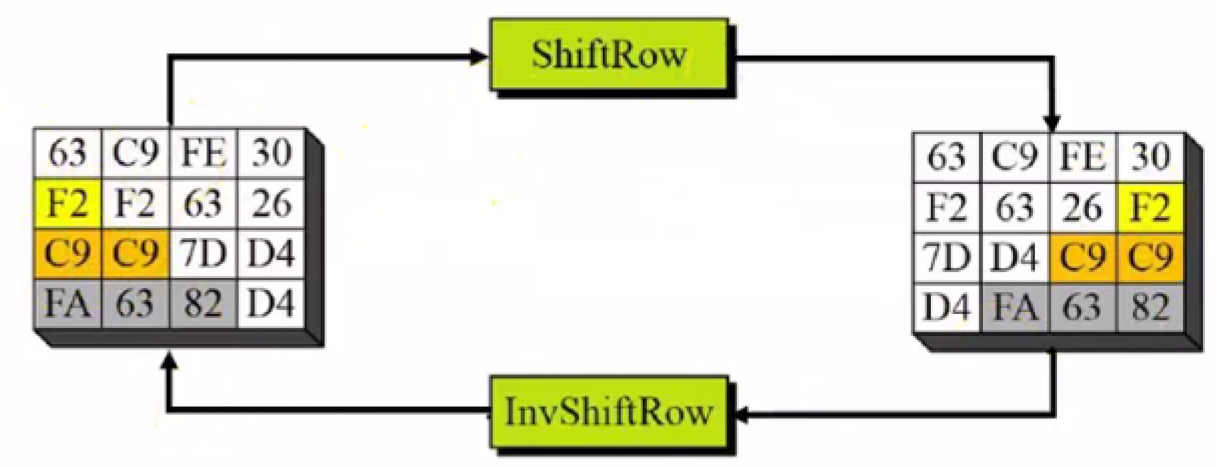
Permutation : Shift Rows

ShiftRows (S)

```
{
    for (r = 1 to 3)
        shiftrow (s_r, r)                  // s_r is the rth row
}
```

```
shiftrow (row, n)                          // n is the number of bytes to be shifted
{
    CopyRow (row, t)                       // t is a temporary row
    for (c = 0 to 3)
        row_((c − n) mod 4) ← t_c
}
```

Example #3



Mixing

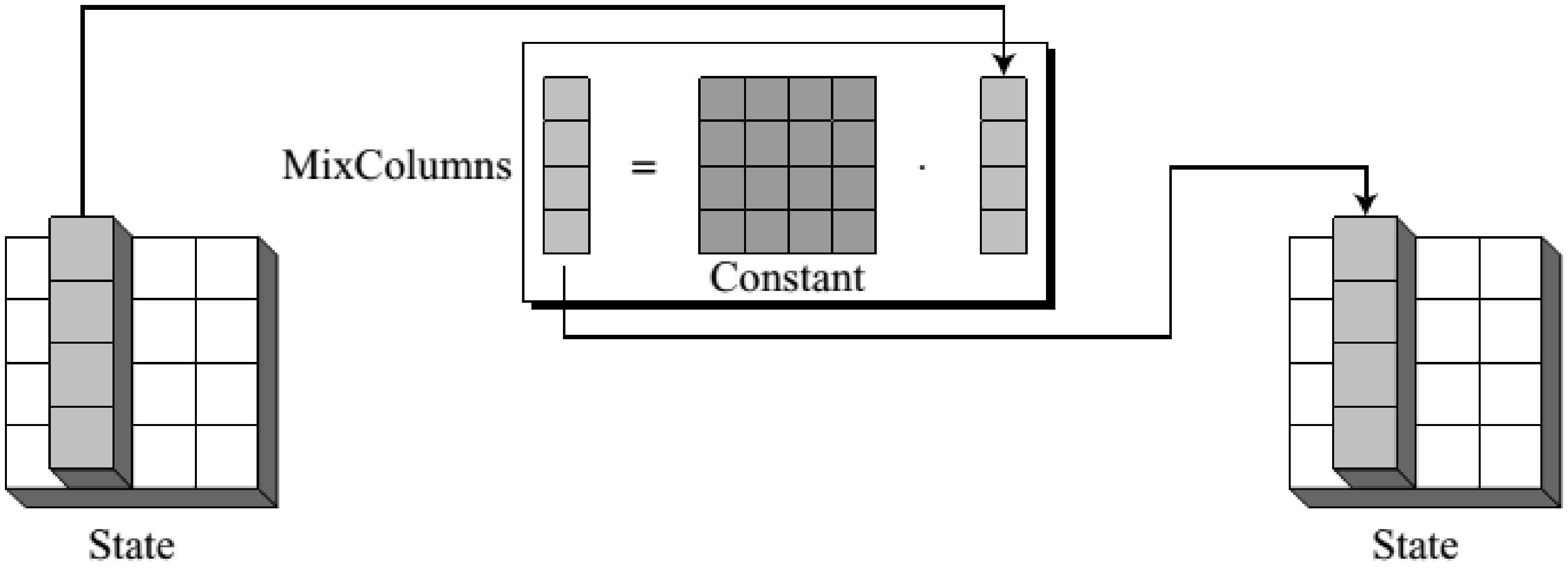
- The substitution provided by the SubBytes transformation changes the value of the byte based only on original value and an entry in the table;
  - The process does not include the neighboring bytes.
  - We can say that SubBytes is an intrabyte transformation.
- The permutation provided by the ShiftRows transformation exchanges bytes without permuting the bits inside the bytes.
  - We can say that ShiftRows is a byte-exchange transformation.
- We also need an interbyte transformation that changes the bits inside a byte, based on the bits inside the neighboring bytes.
- We need to mix bytes to provide diffusion at the bit level.

Mixing

- The mixing transformation changes the contents of each byte by taking four bytes at a time and combining them to recreate four new bytes.
- To guarantee that each new byte is different (even if all four bytes are the same), the combination process first multiplies each byte with a different constant and then mixes them.
- The mixing can be provided by matrix multiplication.

$$\begin{array}{r} a\mathbf{x}+b\mathbf{y}+c\mathbf{z}+d\mathbf{t} \longrightarrow \\ e\mathbf{x}+f\mathbf{y}+g\mathbf{z}+h\mathbf{t} \longrightarrow \\ i\mathbf{x}+j\mathbf{y}+k\mathbf{z}+l\mathbf{t} \longrightarrow \\ m\mathbf{x}+n\mathbf{y}+o\mathbf{z}+p\mathbf{t} \longrightarrow \end{array} \underbrace{\begin{bmatrix} \square \\ \square \\ \square \\ \square \end{bmatrix}}_{\text{New matrix}} = \underbrace{\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}}_{\text{Constant matrix}} \cdot \underbrace{\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{t} \end{bmatrix}}_{\text{Old matrix}}$$

MixColumns & InvMixColumns



$$\underset{C}{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}} \xleftrightarrow{\text{Inverse}} \underset{C^{-1}}{\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}}$$

Pseudocode for MixColumns transformation

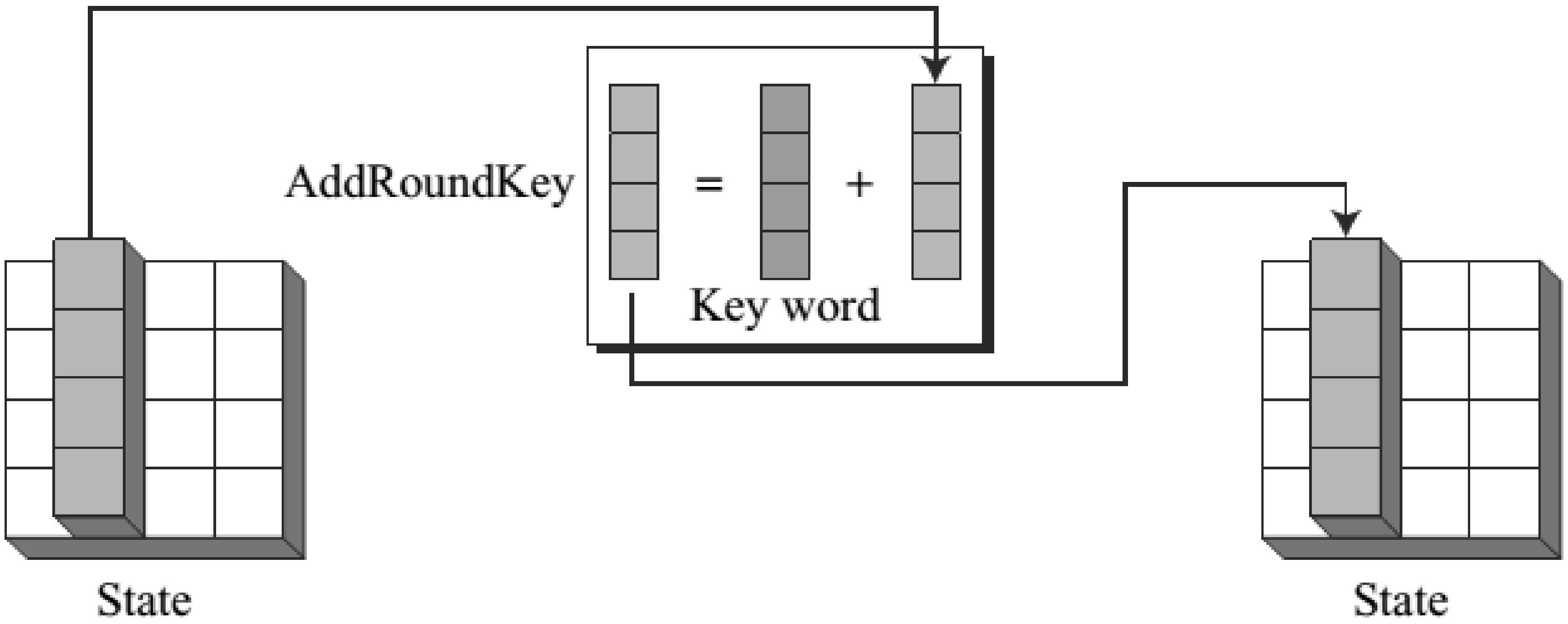
```
MixColumns (S)
{
    for (c = 0 to 3)
        mixcolumn (s_c)
}
```

```
mixcolumn (col)
{
   CopyColumn (col, t)                    // t is a temporary column

    col_0 ← (0x02) • t_0 ⊕ (0x03 • t_1) ⊕ t_2 ⊕ t_3

    col_1 ← t_0 ⊕ (0x02) • t_1 ⊕ (0x03) • t_2 ⊕ t_3

    col_2 ← t_0 ⊕ t_1 ⊕ (0x02) • t_2 ⊕ (0x03) • t_3

    col_3 ← (0x03 • t_0) ⊕ t_1 ⊕ t_2 ⊕ (0x02) • t_3
}
```

Key Adding: AddRoundKey

- AddRoundKey also proceeds one column at a time.
- It is similar to MixColumns in this respect.
- MixColumns multiplies a constant square matrix by each state column;
- AddRoundKey adds a round key word with each state column matrix.
- The operation in MixColumns is matrix multiplication; the operation in AddRoundKey is matrix addition.
- Since addition and subtraction in this field are the same, the AddRoundKey transformation is the inverse of itself.

AddRoundKey transformation



Pseudocode for AddRoundKey transformation

AddRoundKey (S)

{

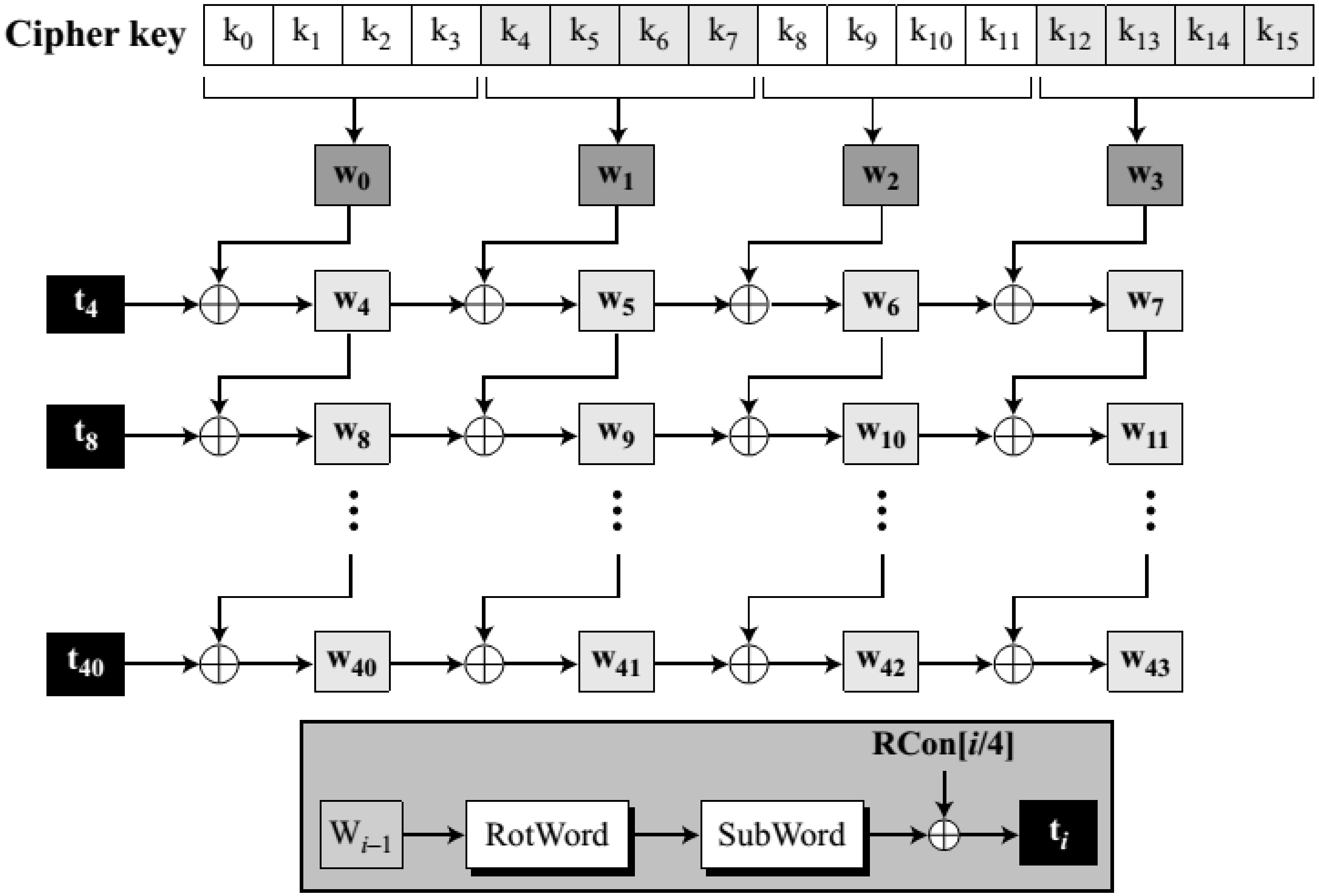
for ($c = 0$ to 3)

$\mathbf{s}_c \leftarrow \mathbf{s}_c \oplus \mathbf{w}_{4\,\text{round} + c}$

}

KEY EXPANSION

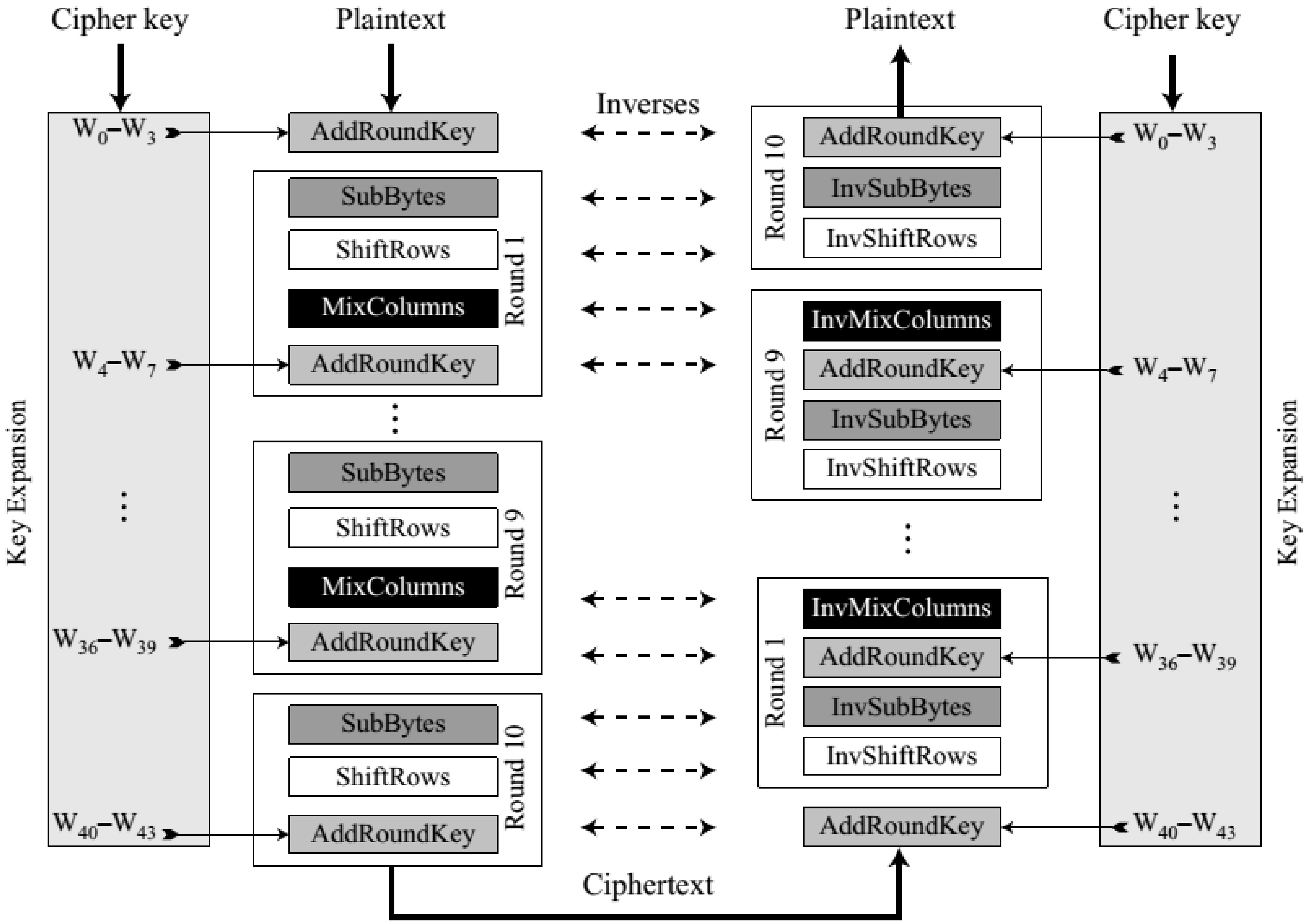
- In the AES-128 version (10 rounds), there are 44 words;
- In the AES-192 version (12 rounds), there are 52 words; and
- In the AES-256 version (with 14 rounds), there are 60 words.
- Each round key is made of four words.



Making of t_i (temporary) words $i = 4\ N_r$

Pseudocode for key expansion in AES-128

```
KeyExpansion ([key_0 to key_15], [w_0 to w_43])
{
    for (i = 0 to 3)
        w_i ← key_4i + key_4i+1 + key_4i+2 + key_4i+3

    for (i = 4 to 43)
    {
        if (i mod 4 ≠ 0)    w_i ← w_i–1 + w_i–4
        else
        {
            t ← SubWord (RotWord (w_i–1)) ⊕ RCon_i/4         // t is a temporary word
            w_i ← t + w_i–4
        }
    }
}
```



Cipher and inverse cipher of the original design

```
Cipher (InBlock [16], OutBlock[16], w[0 … 43])
{
    BlockToState (InBlock, S)

    S ← AddRoundKey (S, w[0…3])
     for (round = 1 to 10)
     {
         S ← SubBytes (S)
         S ← ShiftRows (S)
          if (round ≠ 10)   S ← MixColumns (S)
          S ← AddRoundKey (S, w[4 × round, 4 × round + 3])
     }

    StateToBlock (S, OutBlock);
}
```

Pseudocode for cipher in the original design

ANALYSIS OF AES

- Security
  - AES was designed after DES. Most of the known attacks on DES were already tested on AES; none of them has broken the security of AES so far.
- Brute-Force Attack
  - For DES we need 2^{56}(ignoring the key complement issue) tests to find the key;
  - For AES we need 2^{128} tests to find the key.
  - This means that if we can break DES in t seconds, we need ($2^{72} \times t$) seconds to break AES. This would be almost impossible.
  - AES provides two other versions with longer cipher keys. The lack of weak keys is another advantage of AES over DES.

ANALYSIS OF AES

- Statistical Attacks
  - The strong diffusion and confusion provided by the combination of the SubBytes, ShiftRows, and MixColumns transformations removes any frequency pattern in the plaintext.
  - Numerous tests have failed to do statistical analysis of the ciphertext.
- Differential and Linear Attacks
  - AES was designed after DES.
  - Differential and linear cryptanalysis attacks were no doubt taken into consideration.
  - There are no differential and linear attacks on AES as yet.
- Simplicity and Cost
  - The algorithms used in AES are so simple that they can be easily implemented using cheap processors and a minimum amount of memory.