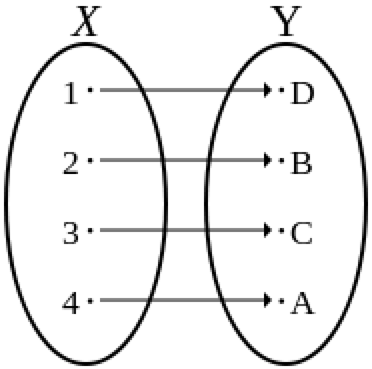
ENEE 459-C
Computer Security

Symmetric key encryption in practice: DES and AES algorithms

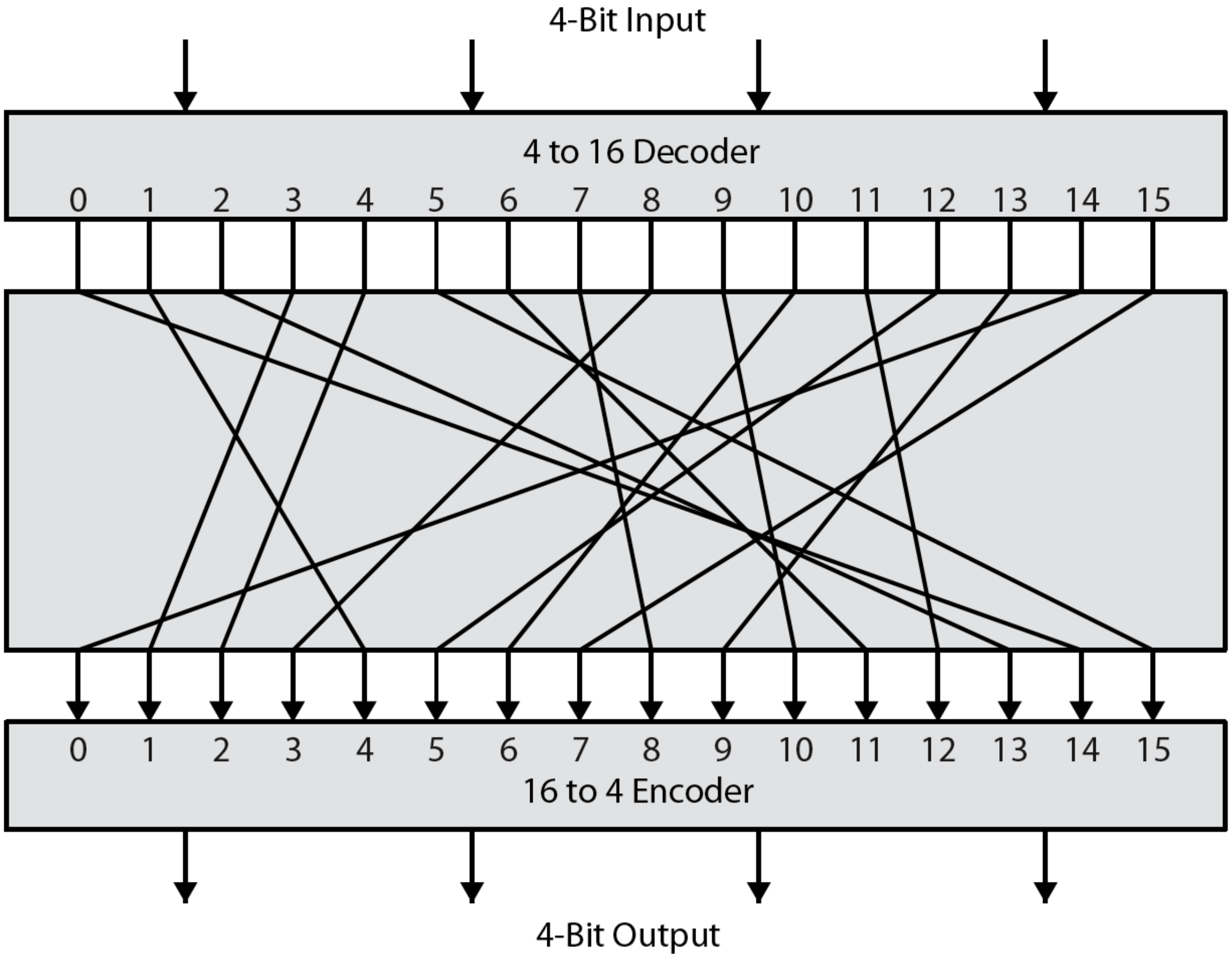
UNIVERSITY OF
MARYLAND

A perfect encryption of a block

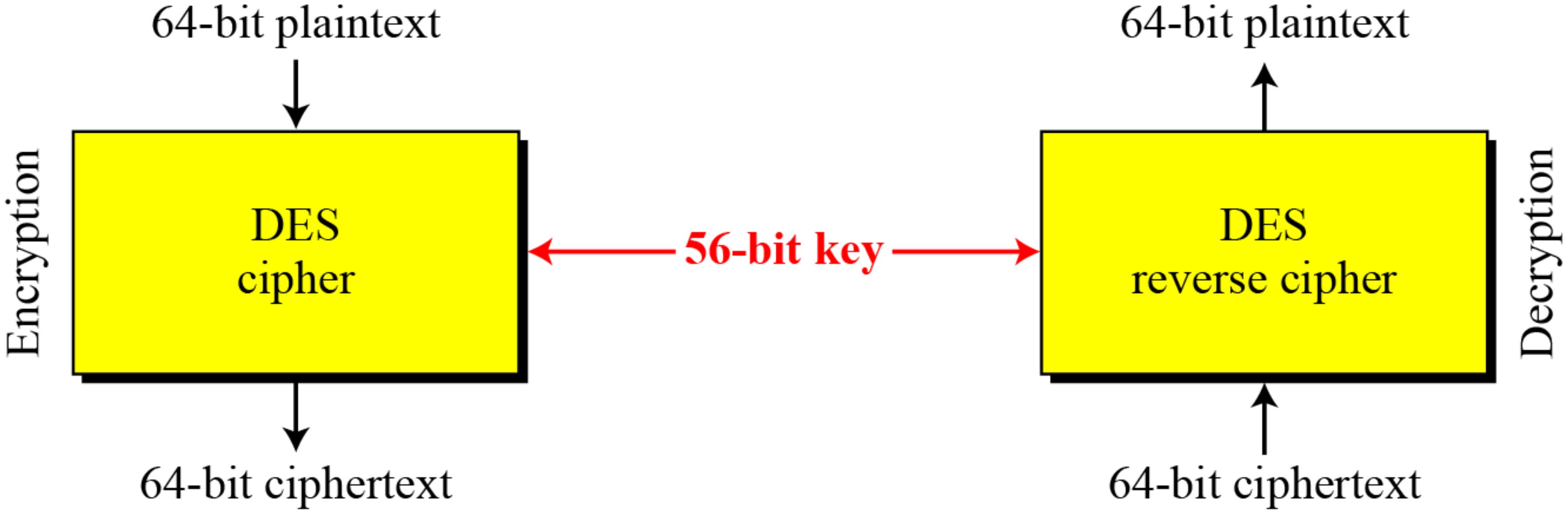
- Say you have a block of n bits
- You want to encrypt it
- You want to use the same key all the time but NOT have the problem of ONE TIME PAD (i.e., be semantically secure)
- Consider a bijective mapping T from $\{0,1\}^n$ to $\{0,1\}^n$
- The pairs are computed uniformly at random
- To encrypt x, just output T[x]
- To decrypt y, just output T^{-1}[y]
- Your secret key is T
- Problem with this approach: T has size ~ n 2^n
- Can you make it randomized (and semantically-secure)?
    - Encrypt x (pick random r): y=T[r] XOR x, r
    - Decrypt (y,r): y XOR T^{-1}[r]



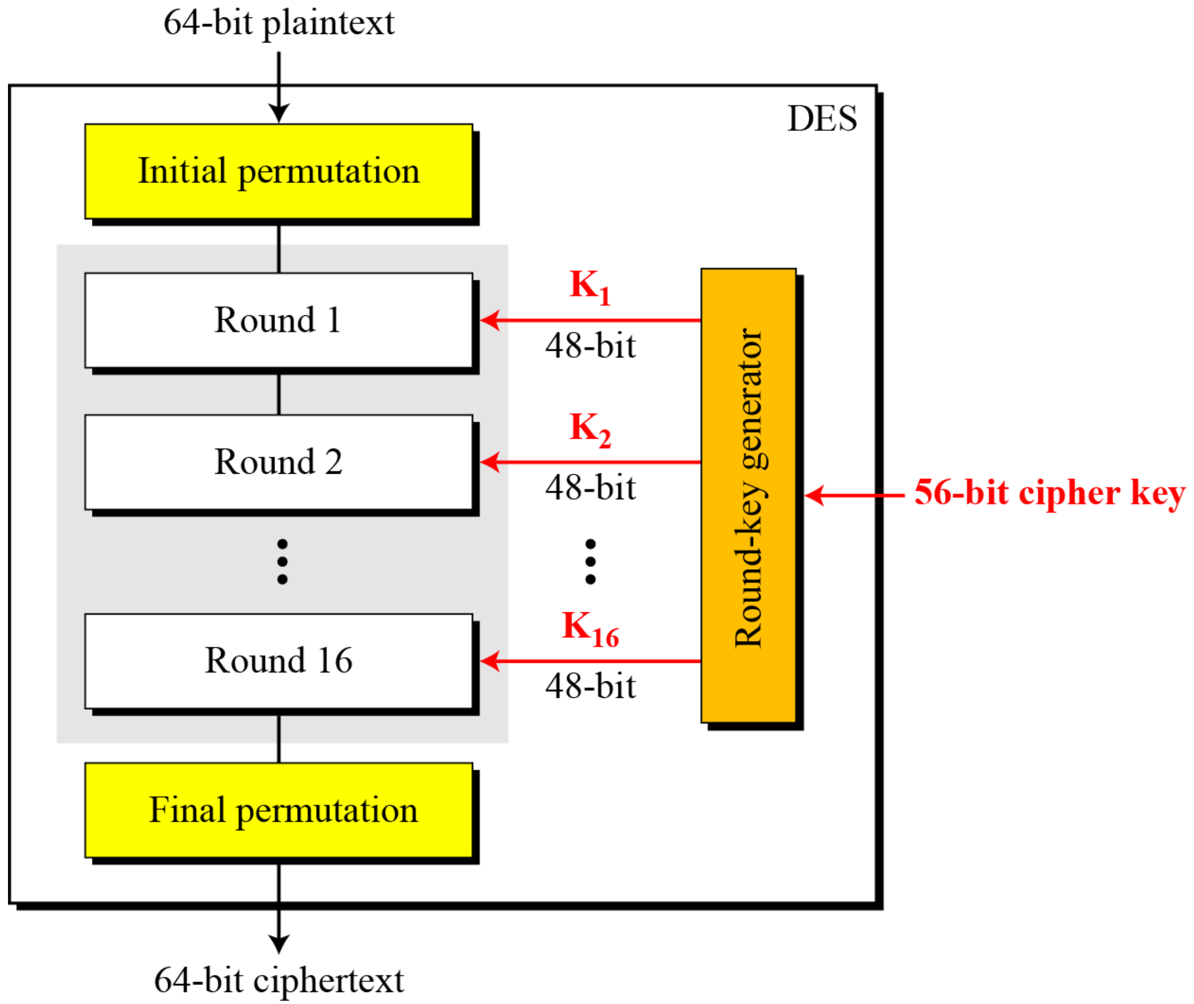
Reminder: Ideal block cipher



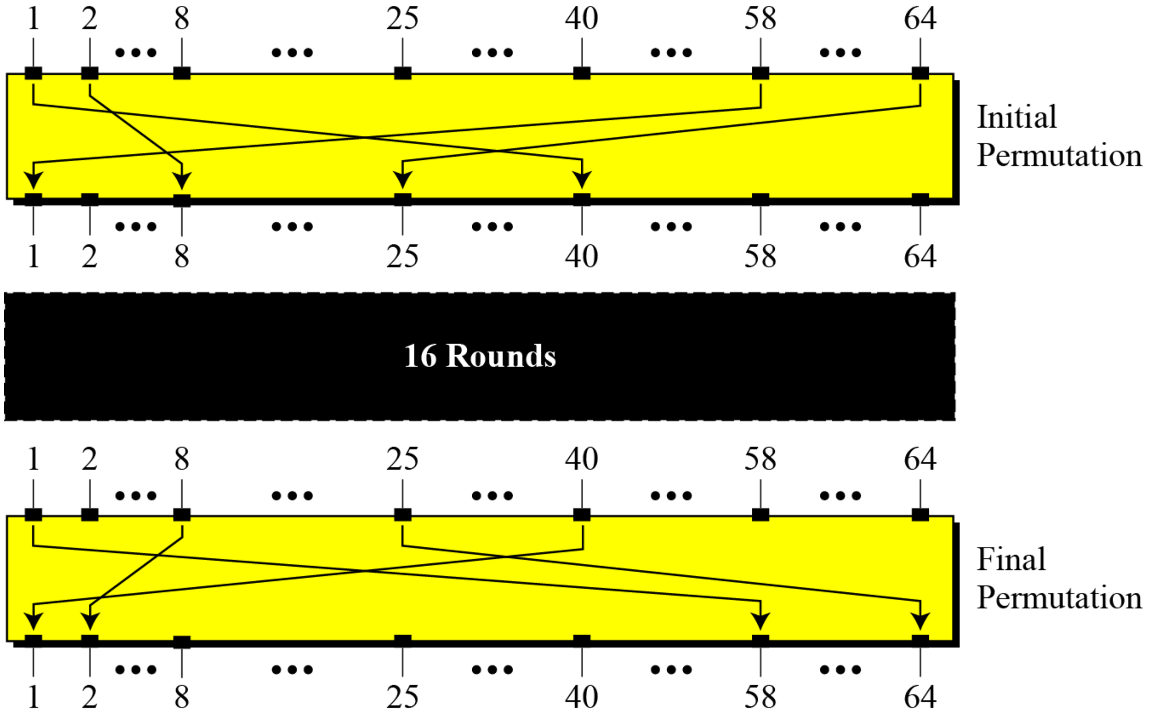
DES algorithm



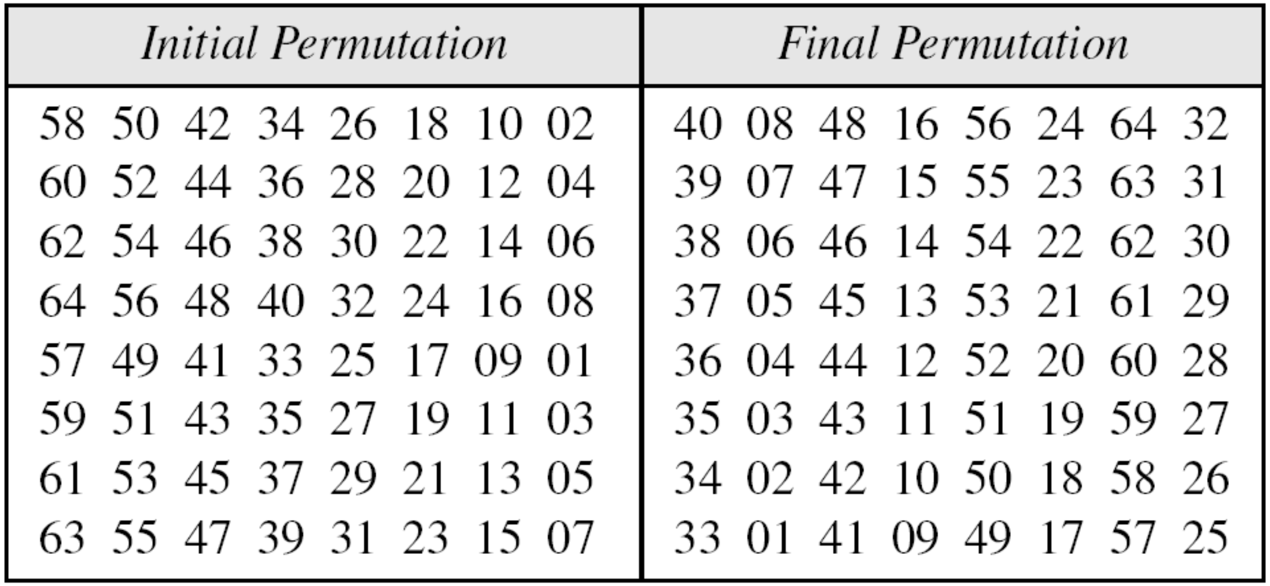
DES structure



Initial and final permutation



The table of the permutations



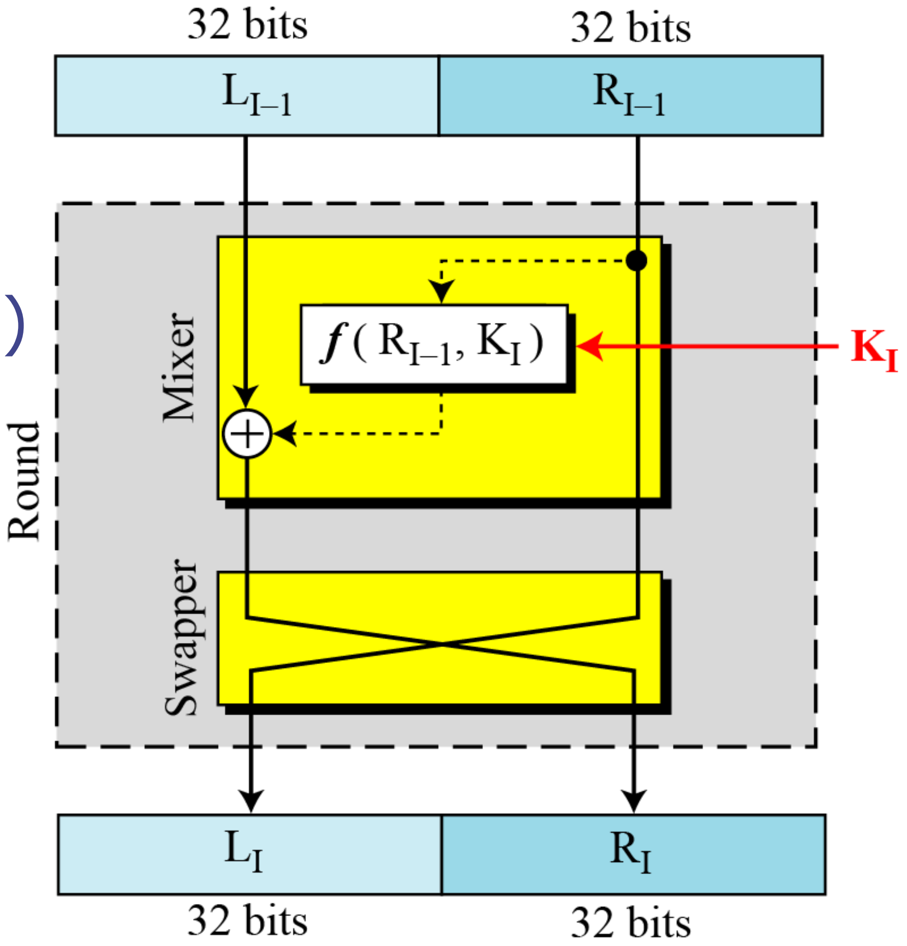
Initial Permutation								*Final Permutation*							
58	50	42	34	26	18	10	02	40	08	48	16	56	24	64	32
60	52	44	36	28	20	12	04	39	07	47	15	55	23	63	31
62	54	46	38	30	22	14	06	38	06	46	14	54	22	62	30
64	56	48	40	32	24	16	08	37	05	45	13	53	21	61	29
57	49	41	33	25	17	09	01	36	04	44	12	52	20	60	28
59	51	43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

- The initial and final permutations are straight P-boxes that are inverses of each other
- They have no cryptography significance in DES

DES round: Feistel network

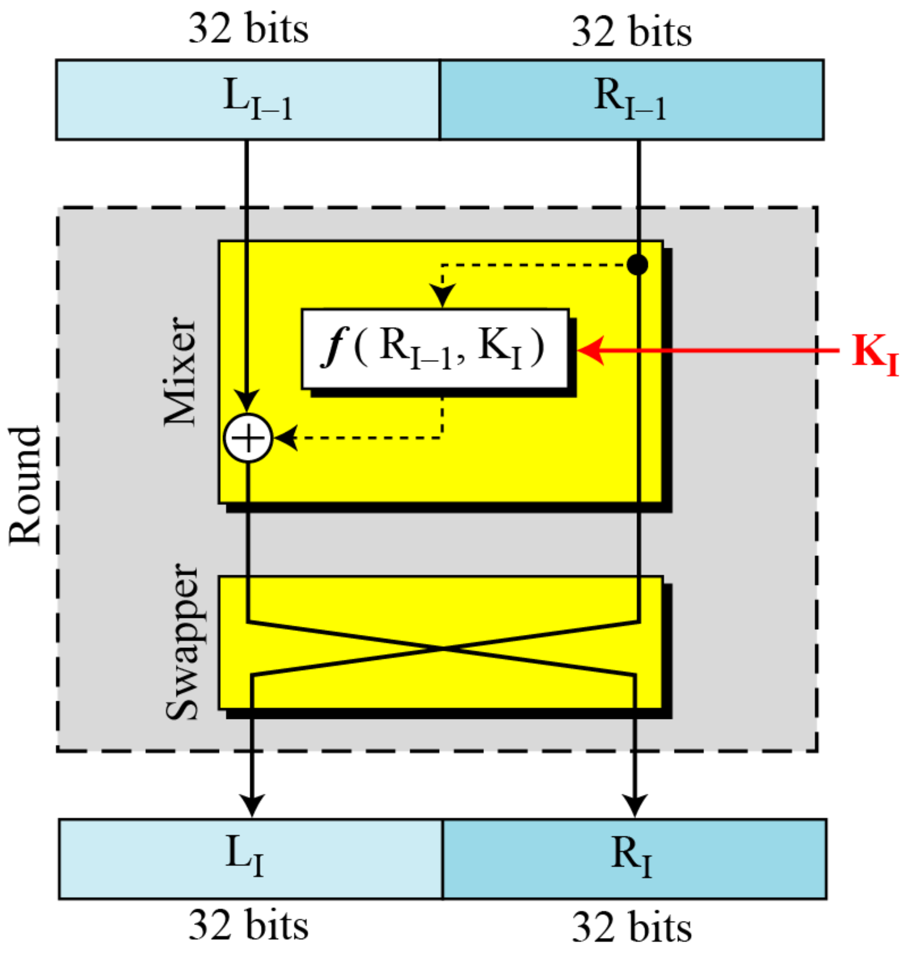
- DES uses 16 rounds
- Each round of DES is a Feistel cipher

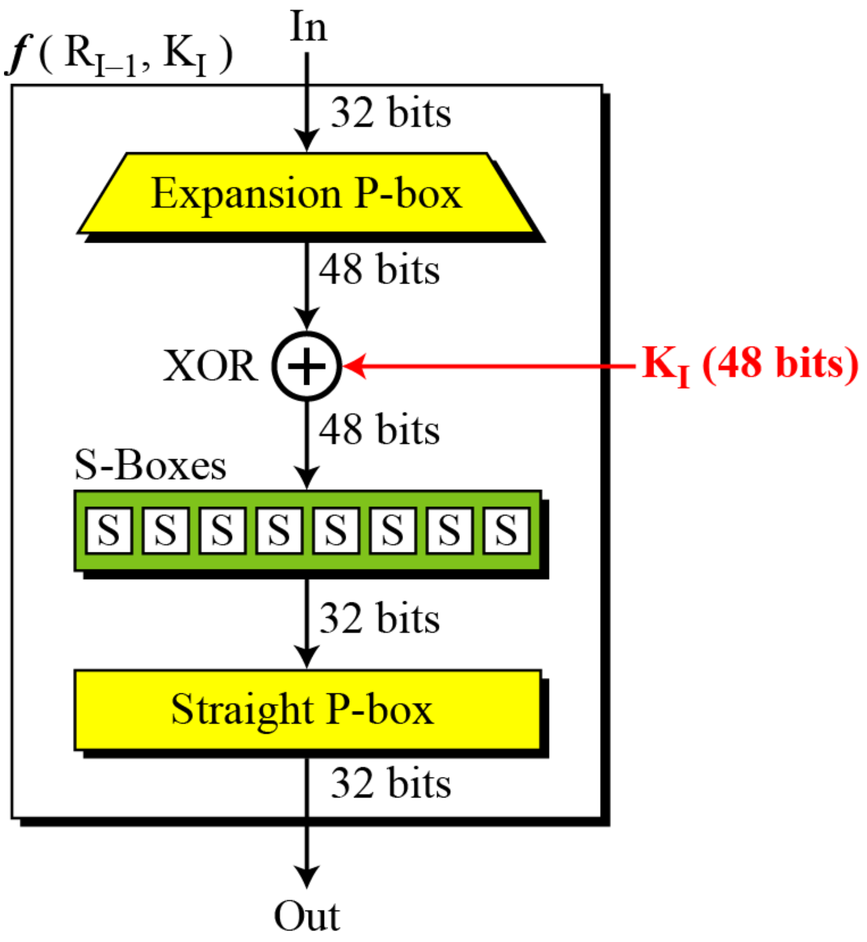
- L(i) = R(i-1)
- R(i) = L(i-1) XOR f(K(i),R(i-1))



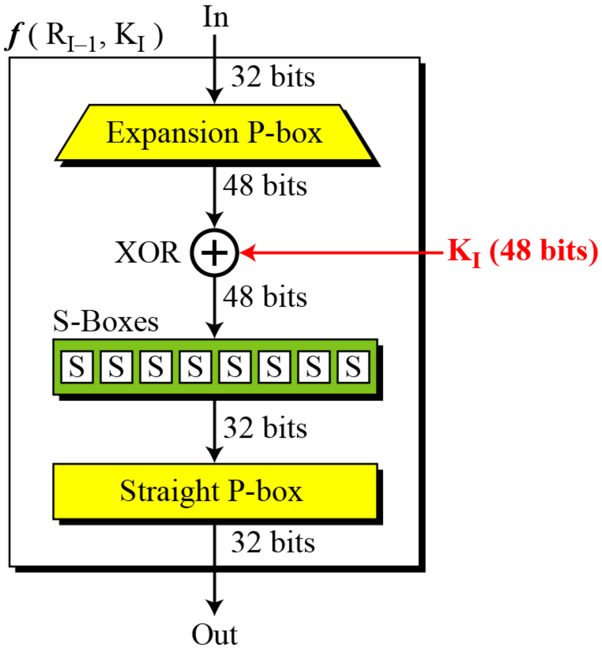
DES function

The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output

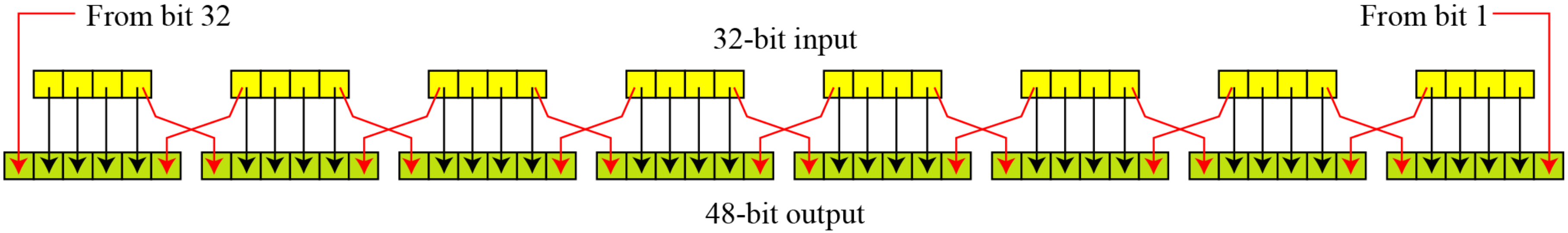




Expansion box

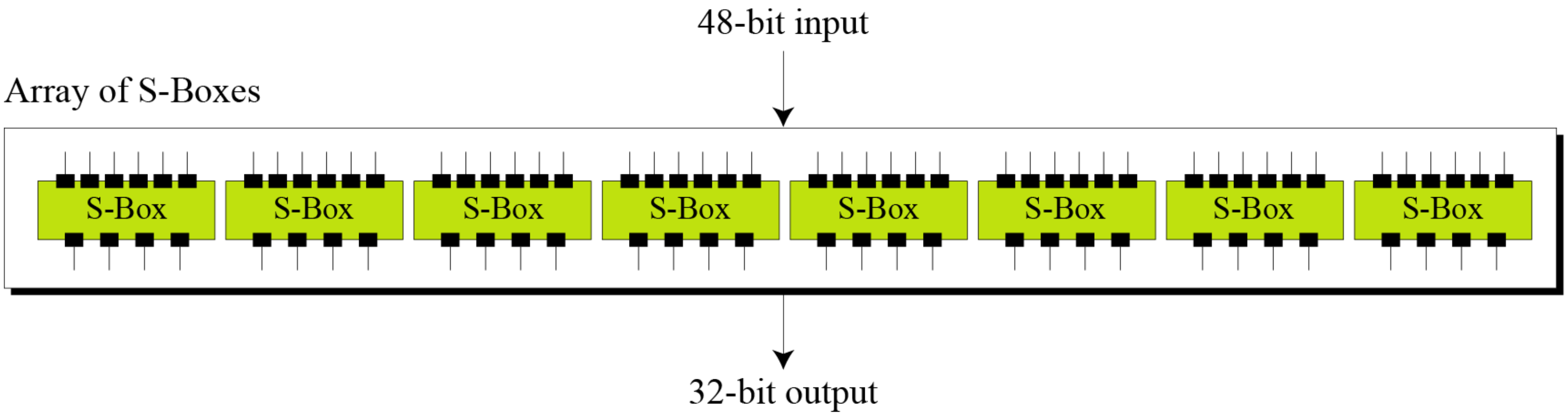


Since R_{I-1} is a 32-bit input and K_I is a 48-bit key, we first need to expand R_{I-1} to 48 bits

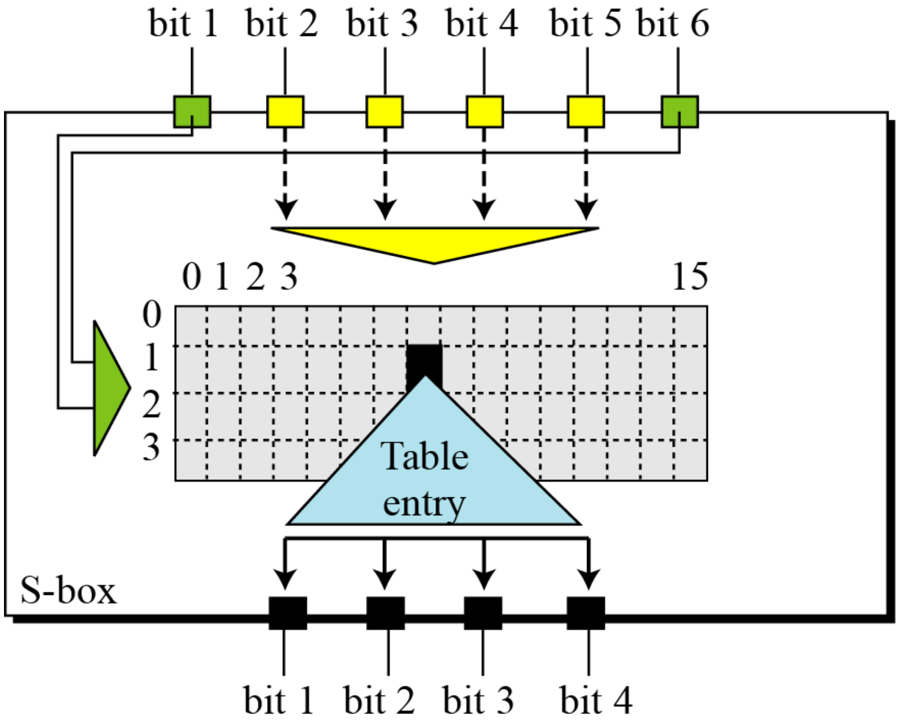


S-box and the avalanche effect

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



S-box in detail



	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Execute one round on 8-bit plaintext

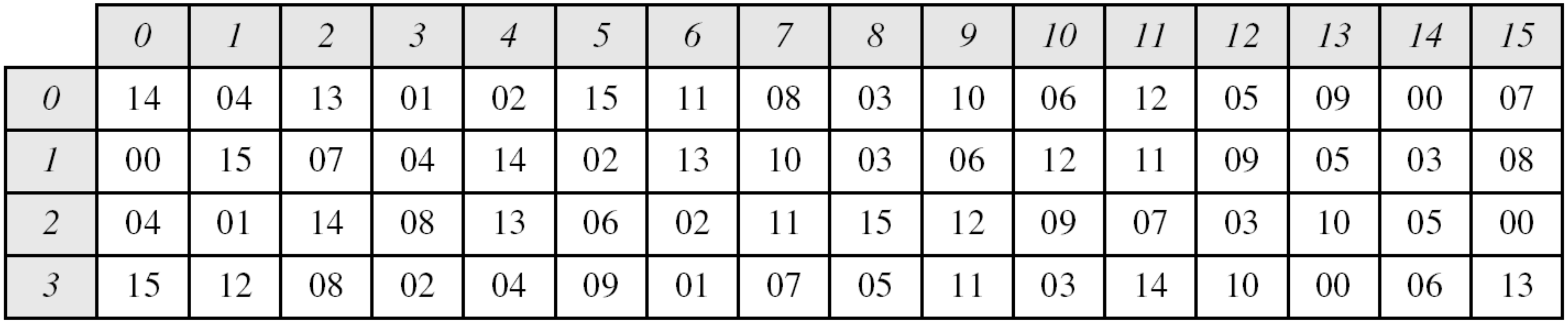
plaintext: 0101 1111

exp-box

key: 010011 010010

s-box (see below)

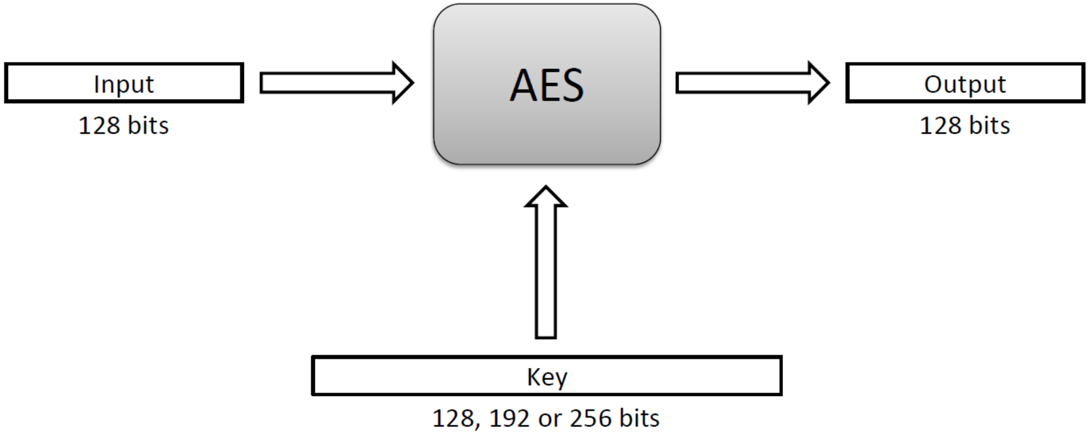
p-box: [1 15 0 2 14 13 5 7 11 10 9 8 3 4 12 6]



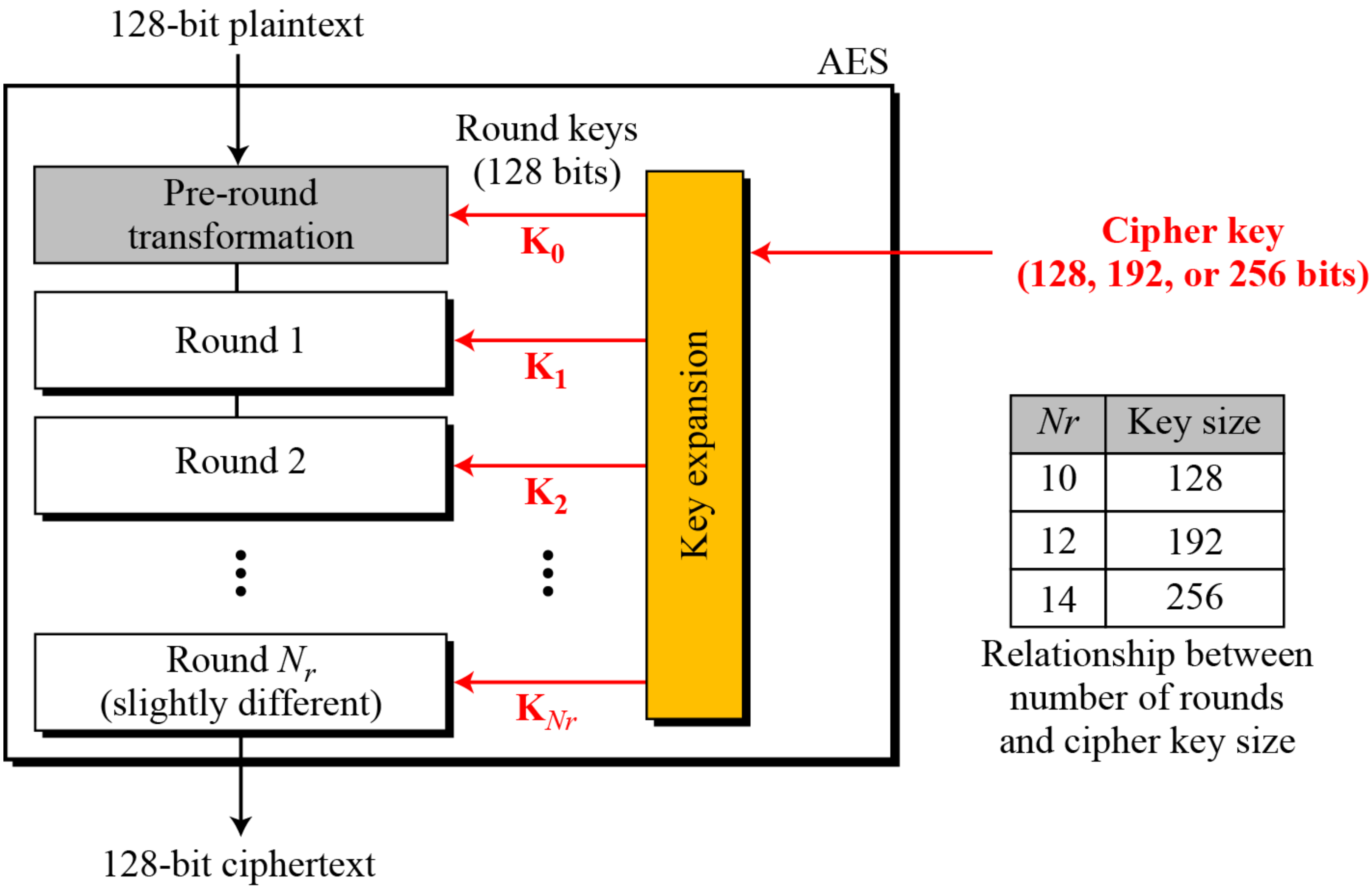
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

The Advanced Encryption Standard (AES)

- In 1997, the U.S. National Institute for Standards and Technology (NIST) put out a public call for a replacement to DES.
- It narrowed down the list of submissions to five finalists, and ultimately chose an algorithm that is now known as the **Advanced Encryption Standard** (**AES**).
- AES is a block cipher that operates on 128-bit blocks. It is designed to be used with keys that are 128, 192, or 256 bits long, yielding ciphers known as AES-128, AES-192, and AES-256.



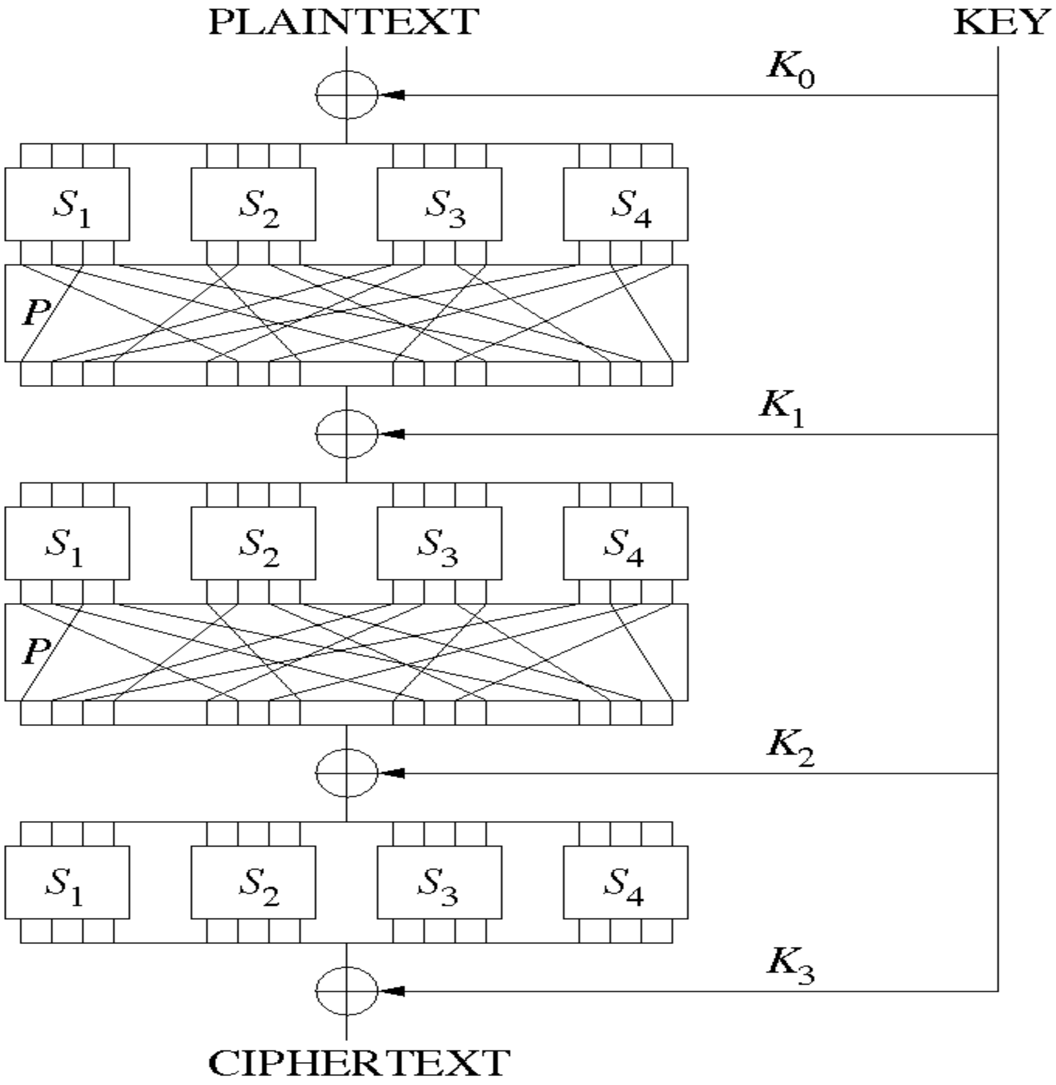
AES structure



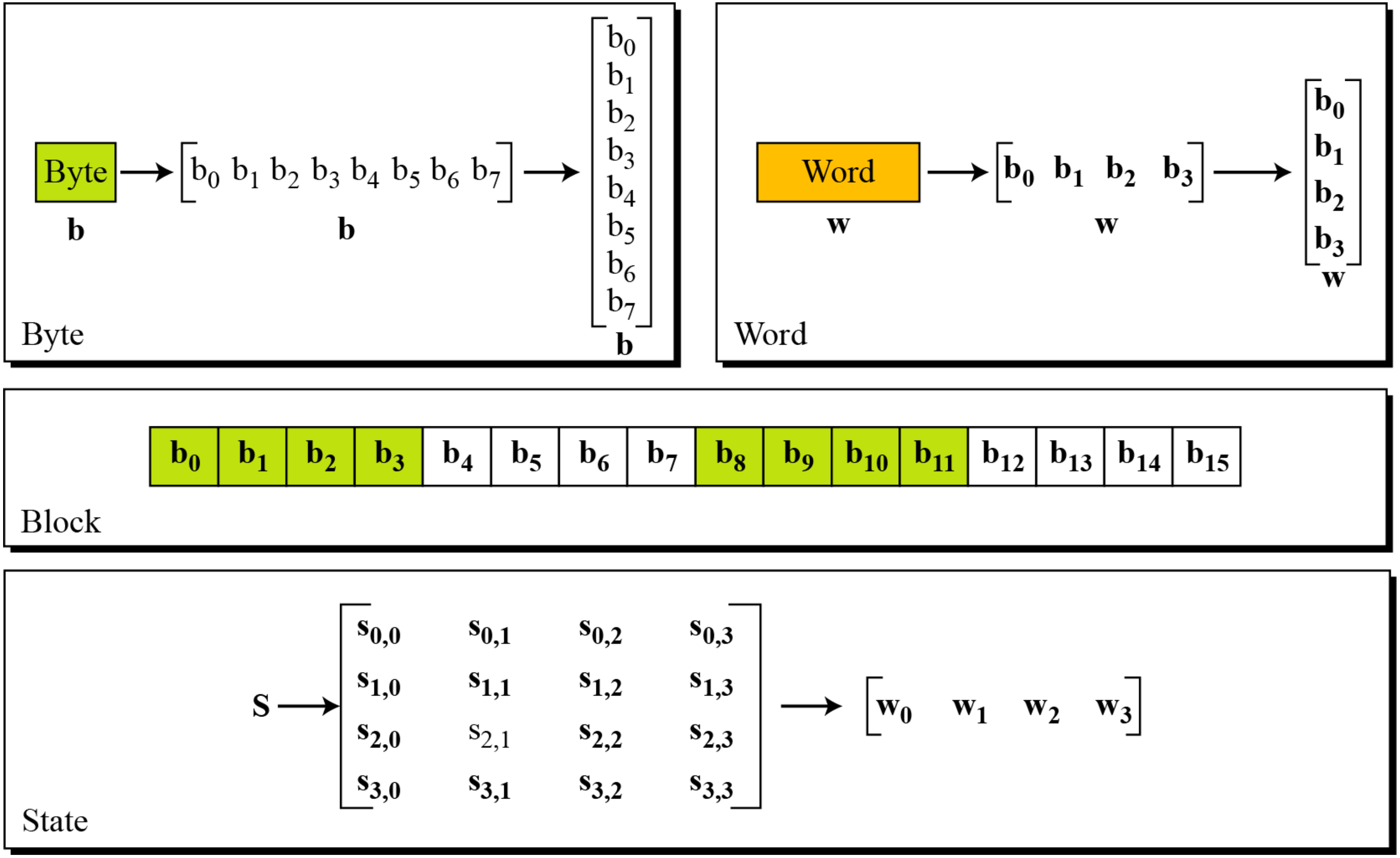
Nr	Key size
10	128
12	192
14	256

Relationship between
number of rounds
and cipher key size

Substitution/permutation network



Data Units in AES



$$\underset{\mathbf{b}}{\boxed{\text{Byte}}} \longrightarrow \underset{\mathbf{b}}{\begin{bmatrix} b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \end{bmatrix}} \longrightarrow \underset{\mathbf{b}}{\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}}$$

Byte

$$\underset{\mathbf{w}}{\boxed{\text{Word}}} \longrightarrow \underset{\mathbf{w}}{\begin{bmatrix} \mathbf{b_0} & \mathbf{b_1} & \mathbf{b_2} & \mathbf{b_3} \end{bmatrix}} \longrightarrow \underset{\mathbf{w}}{\begin{bmatrix} \mathbf{b_0} \\ \mathbf{b_1} \\ \mathbf{b_2} \\ \mathbf{b_3} \end{bmatrix}}$$

Word

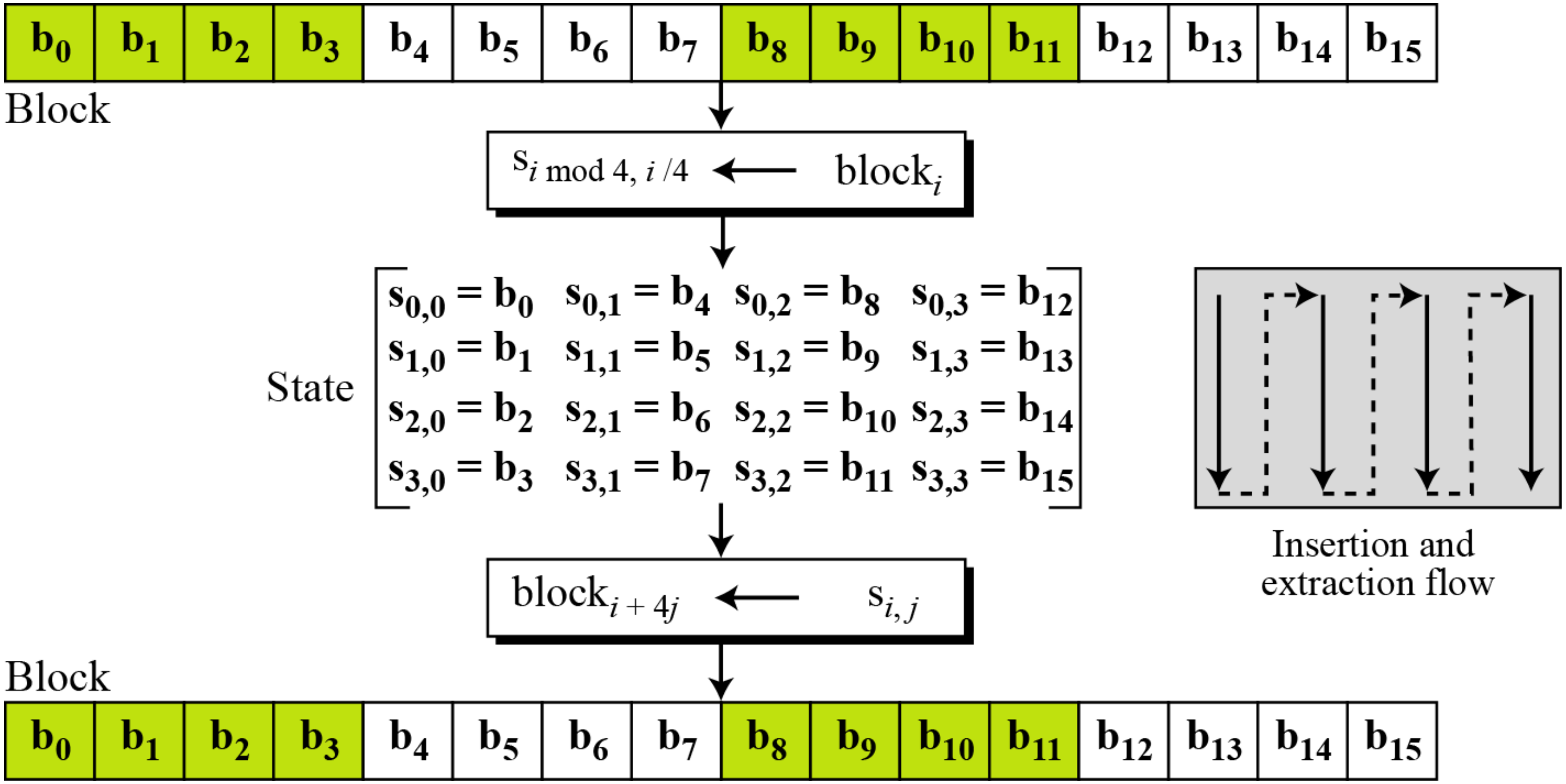
$\mathbf{b_0}$	$\mathbf{b_1}$	$\mathbf{b_2}$	$\mathbf{b_3}$	$\mathbf{b_4}$	$\mathbf{b_5}$	$\mathbf{b_6}$	$\mathbf{b_7}$	$\mathbf{b_8}$	$\mathbf{b_9}$	$\mathbf{b_{10}}$	$\mathbf{b_{11}}$	$\mathbf{b_{12}}$	$\mathbf{b_{13}}$	$\mathbf{b_{14}}$	$\mathbf{b_{15}}$

Block

$$\mathbf{S} \longrightarrow \begin{bmatrix} \mathbf{s_{0,0}} & \mathbf{s_{0,1}} & \mathbf{s_{0,2}} & \mathbf{s_{0,3}} \\ \mathbf{s_{1,0}} & \mathbf{s_{1,1}} & \mathbf{s_{1,2}} & \mathbf{s_{1,3}} \\ \mathbf{s_{2,0}} & s_{2,1} & \mathbf{s_{2,2}} & \mathbf{s_{2,3}} \\ \mathbf{s_{3,0}} & \mathbf{s_{3,1}} & \mathbf{s_{3,2}} & \mathbf{s_{3,3}} \end{bmatrix} \longrightarrow \begin{bmatrix} \mathbf{w_0} & \mathbf{w_1} & \mathbf{w_2} & \mathbf{w_3} \end{bmatrix}$$

State

Transformations from block to state and vice versa



b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}

Block

$s_{i \bmod 4,\, i/4} \leftarrow \text{block}_i$

State
$$\begin{bmatrix} s_{0,0} = b_0 & s_{0,1} = b_4 & s_{0,2} = b_8 & s_{0,3} = b_{12} \\ s_{1,0} = b_1 & s_{1,1} = b_5 & s_{1,2} = b_9 & s_{1,3} = b_{13} \\ s_{2,0} = b_2 & s_{2,1} = b_6 & s_{2,2} = b_{10} & s_{2,3} = b_{14} \\ s_{3,0} = b_3 & s_{3,1} = b_7 & s_{3,2} = b_{11} & s_{3,3} = b_{15} \end{bmatrix}$$

Insertion and extraction flow

$\text{block}_{i+4j} \leftarrow s_{i,j}$

Block

b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}

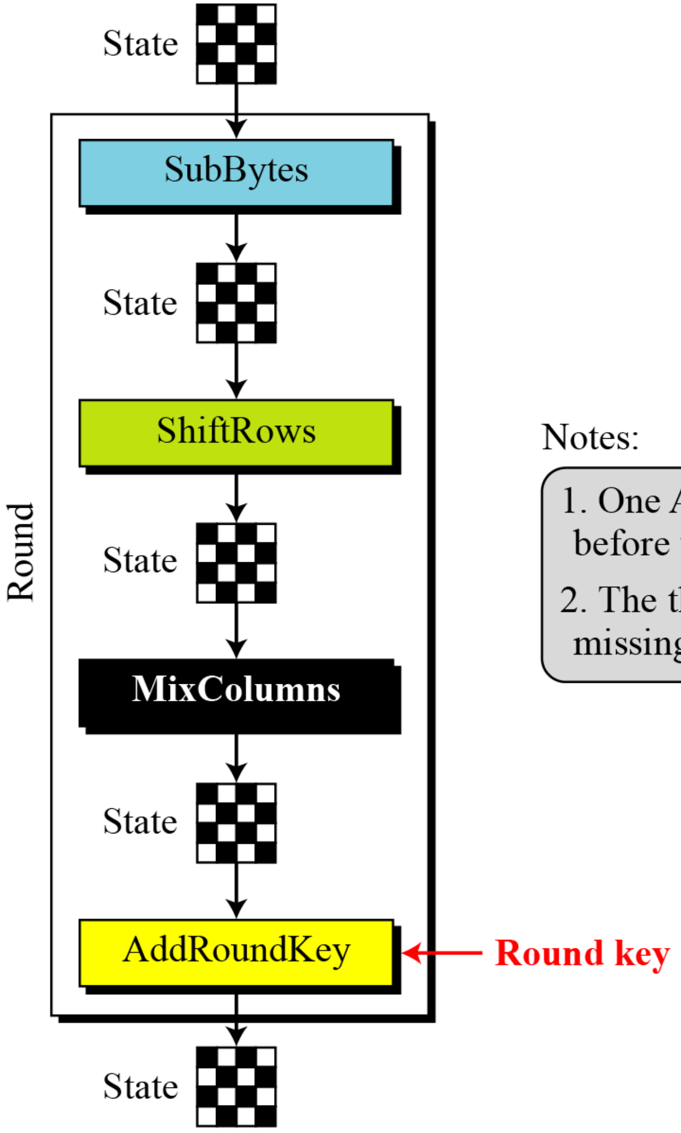
Text to state

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

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

Structure of each round

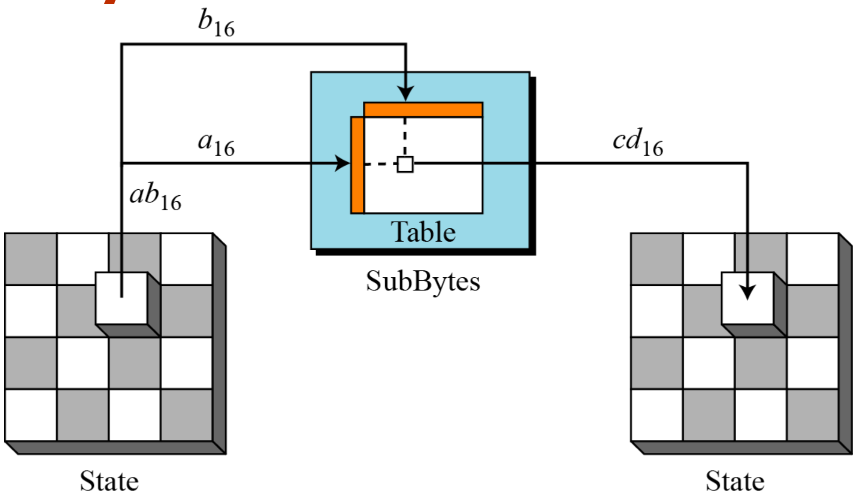
- To provide security, AES uses four types of transformations:
- substitution
- Permutation
- mixing
- key-adding

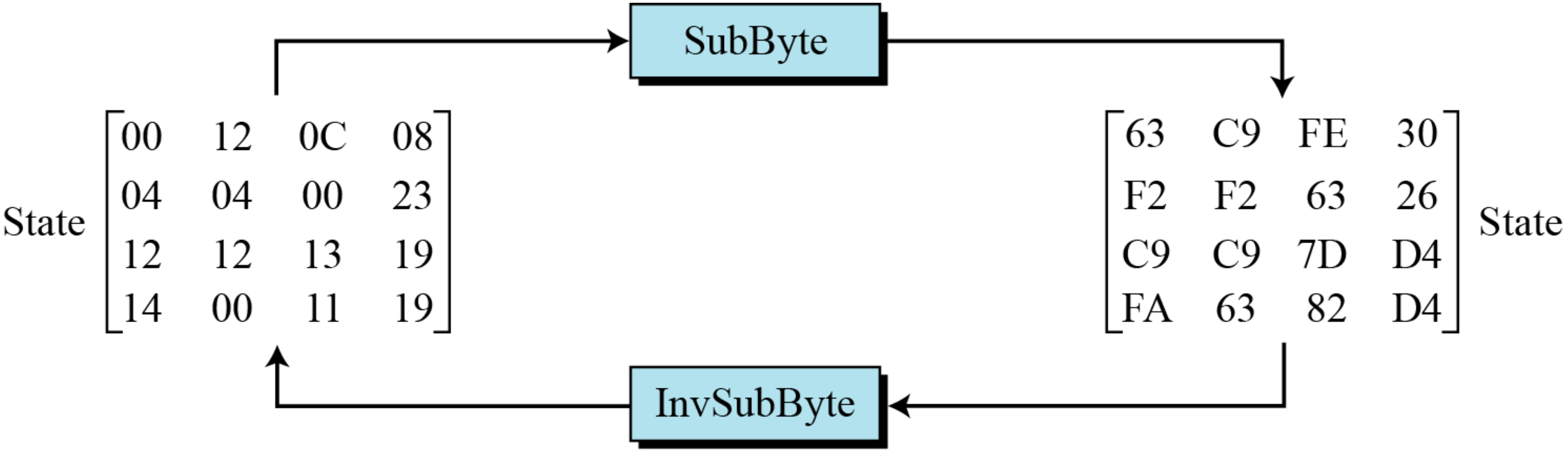


Notes:

1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

SubBytes





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

$$\begin{bmatrix} 63 & C9 & FE & 30 \\ F2 & F2 & 63 & 26 \\ C9 & C9 & 7D & D4 \\ FA & 63 & 82 & D4 \end{bmatrix}\text{State}$$

Tables

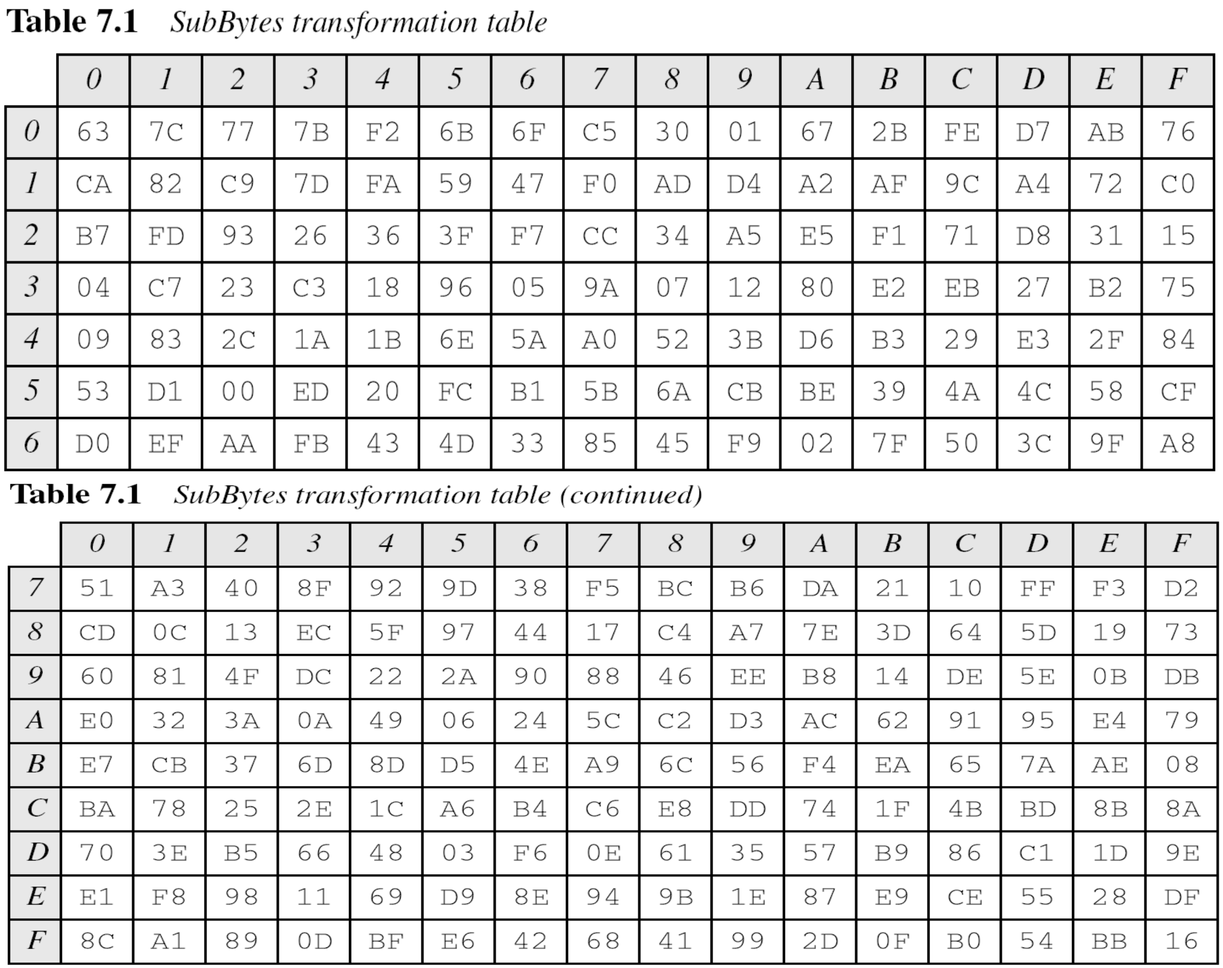


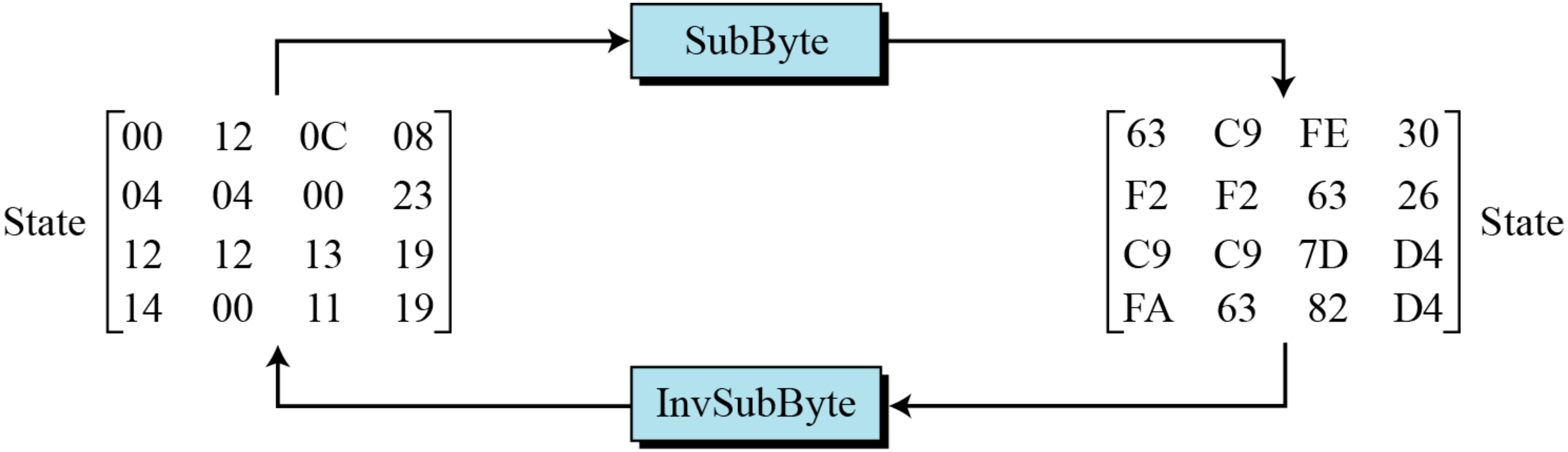
Table 7.1 *SubBytes transformation table*

	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

Table 7.1 *SubBytes transformation table (continued)*

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
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

Example

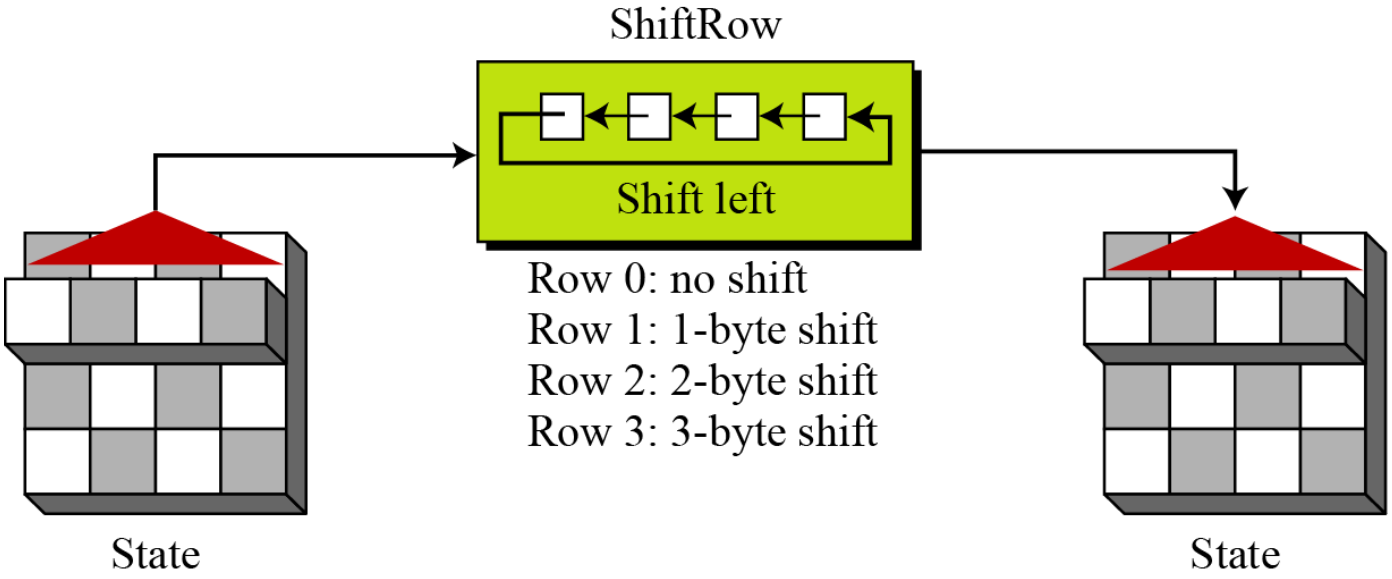


$$\text{State}\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix}$$

$$\begin{bmatrix} 63 & C9 & FE & 30 \\ F2 & F2 & 63 & 26 \\ C9 & C9 & 7D & D4 \\ FA & 63 & 82 & D4 \end{bmatrix}\text{State}$$

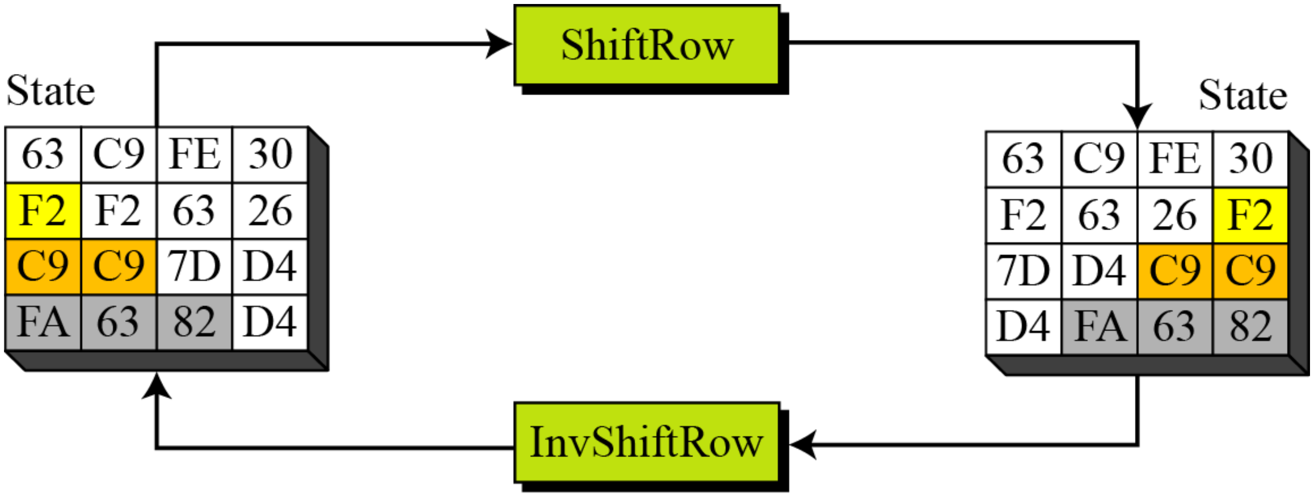
ShiftRows

Another transformation found in a round is shifting, which permutes the bytes



Example

- ShiftRow is used in encryption
- InvShiftRow is used in decryption



State

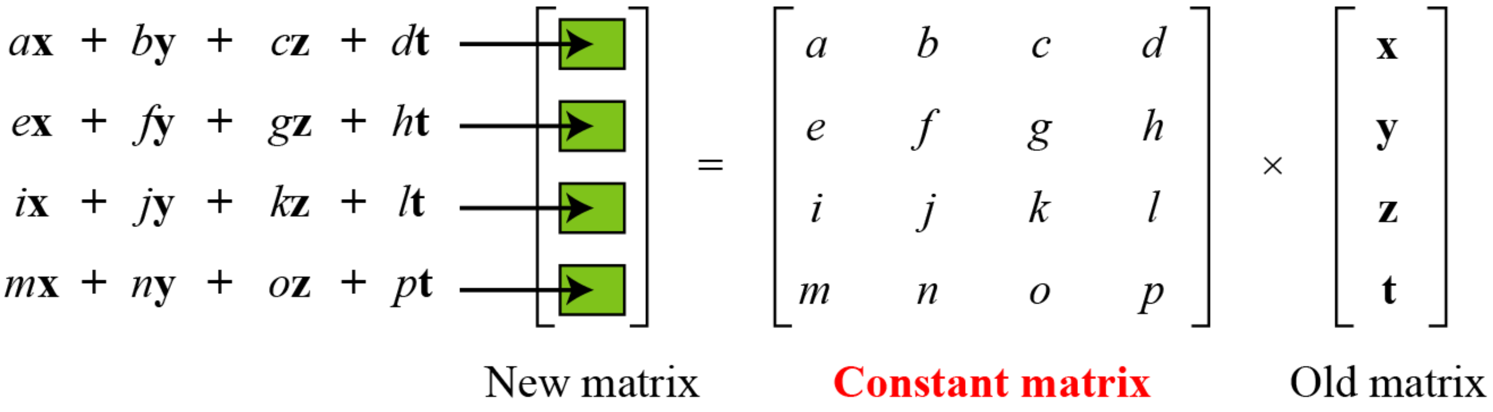
63	C9	FE	30
F2	F2	63	26
C9	C9	7D	D4
FA	63	82	D4

State

63	C9	FE	30
F2	63	26	F2
7D	D4	C9	C9
D4	FA	63	82

Mixing

We 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



$$\begin{matrix} a\mathbf{x} + b\mathbf{y} + c\mathbf{z} + d\mathbf{t} \\ e\mathbf{x} + f\mathbf{y} + g\mathbf{z} + h\mathbf{t} \\ i\mathbf{x} + j\mathbf{y} + k\mathbf{z} + l\mathbf{t} \\ m\mathbf{x} + n\mathbf{y} + o\mathbf{z} + p\mathbf{t} \end{matrix} \rightarrow \begin{bmatrix} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{t} \end{bmatrix}$$

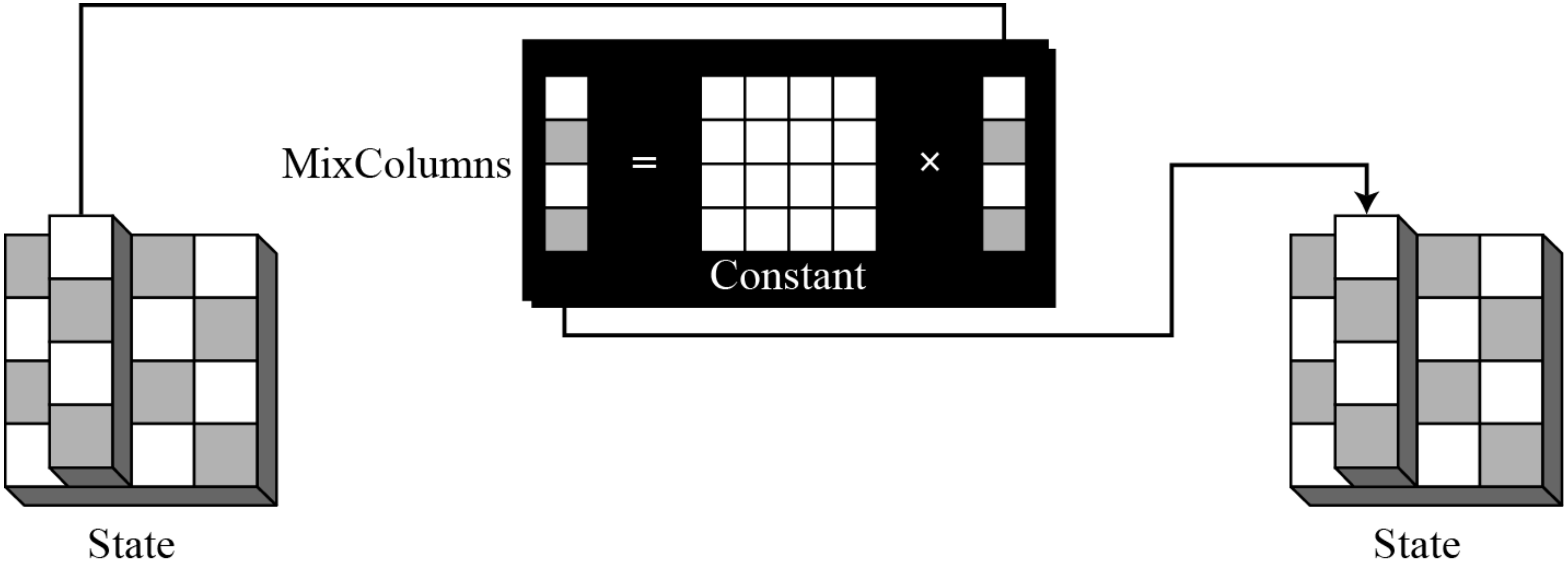
New matrix

Constant matrix

Old matrix

Transforming columns

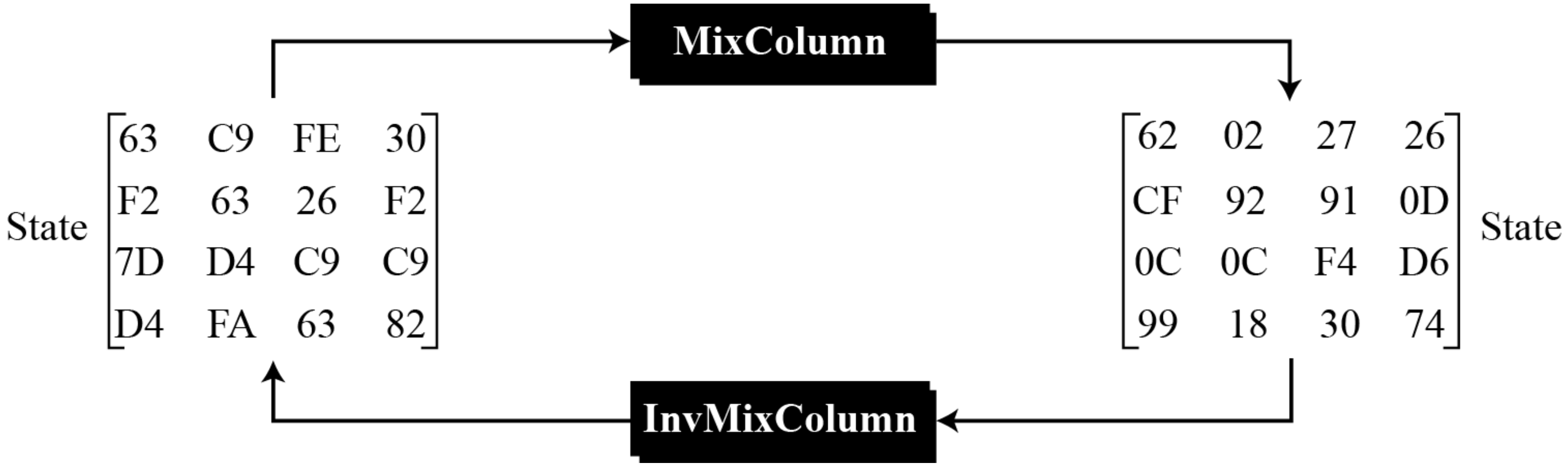
The MixColumns transformation operates at the column level; it transforms each column of the state to a new column



The constant matrices

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

Example



$$\text{State}\quad\begin{bmatrix} 63 & C9 & FE & 30 \\ F2 & 63 & 26 & F2 \\ 7D & D4 & C9 & C9 \\ D4 & FA & 63 & 82 \end{bmatrix}$$

MixColumn

InvMixColumn

$$\begin{bmatrix} 62 & 02 & 27 & 26 \\ CF & 92 & 91 & 0D \\ 0C & 0C & F4 & D6 \\ 99 & 18 & 30 & 74 \end{bmatrix}\quad\text{State}$$

Algebra

- The arithmetic operations are performed in the **GF(2^8)** field modulo $\mathbf{(x^8 + x^4 + x^3 + x + 1)}$
- Map a byte to a polynomial
- Example

{53} • {CA}=

$(x^6 + x^4 + x + 1)(x^7 + x^6 + x^3 + x) =$

$(x^{13} + x^{12} + x^9 + \mathbf{x^7}) + (x^{11} + x^{10} + \mathbf{x^7} + x^5) +$

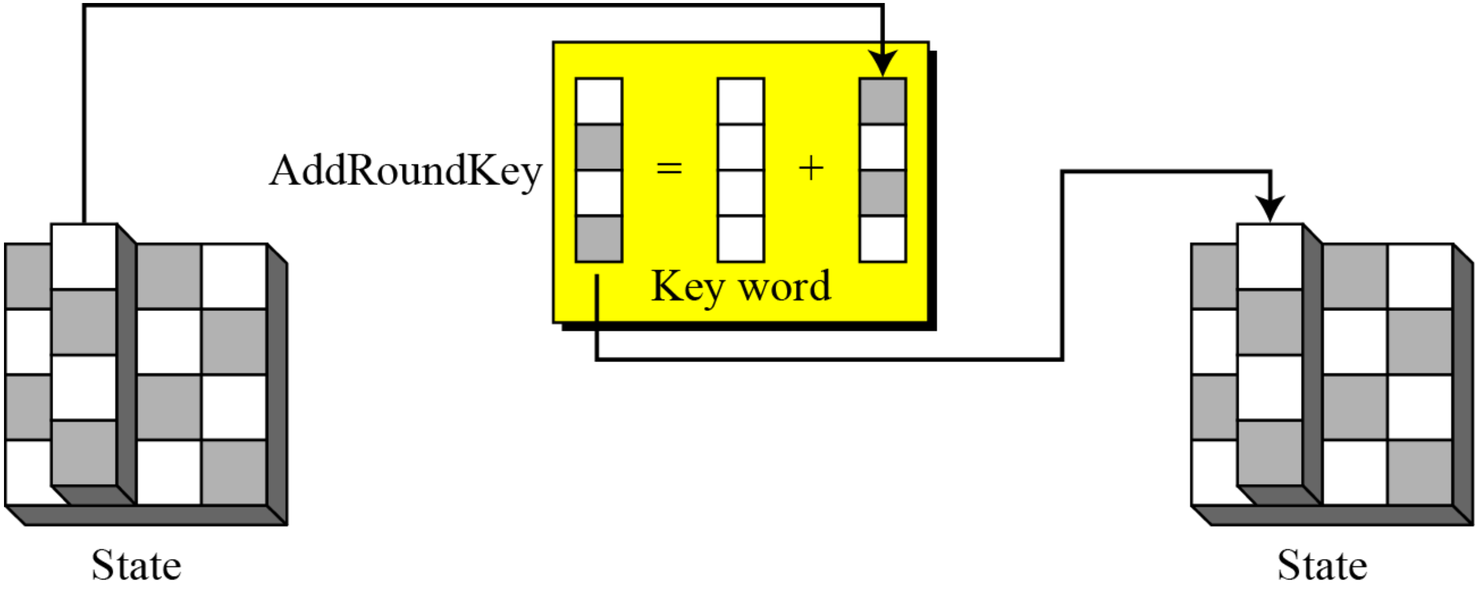
$(x^8 + \mathbf{x^7} + x^4 + x^2) + (\mathbf{x^7} + x^6 + x^3 + x) =$

$x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x$

and

$x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x$ modulo $x^8 + x^4 + x^3 + x^1 + 1$ = (11111101111110 mod 100011011) = {3F7E mod 11B} = {01} = 1

Add round key (final step)

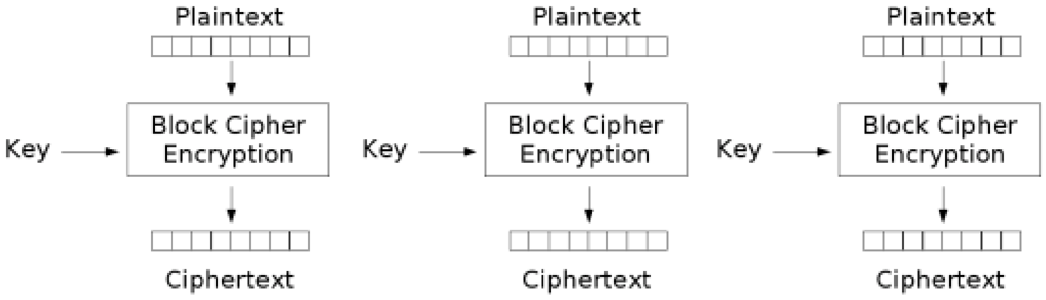


Algorithm 7.4 *Pseudocode for AddRoundKey transformation*

```
AddRoundKey (S)
{
    for ($c$ = 0 to 3)
        $\mathbf{s}_c \leftarrow \mathbf{s}_c \oplus \mathbf{w}_{\text{round} + 4c}$
}
```

Block Cipher Modes

- A block cipher mode describes the way a block cipher encrypts and decrypts a sequence of message blocks.
- Electronic Code Book (ECB) Mode (is the simplest):
  - Block P[i] encrypted into ciphertext block C[i] = $E_K(P[i])$
  - Block C[i] decrypted into plaintext block M[i] = $D_K(C[i])$

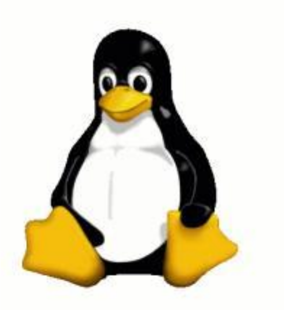


Electronic Codebook (ECB) mode encryption

Public domain images from http://en.wikipedia.org/wiki/File:Ecb_encryption.png and http://en.wikipedia.org/wiki/File:Ecb_decryption.png

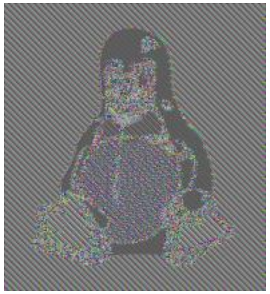
Strengths and Weaknesses of ECB

- Strengths:
  - Is very simple
  - Allows for parallel encryptions of the blocks of a plaintext
  - Can tolerate the loss or damage of a block

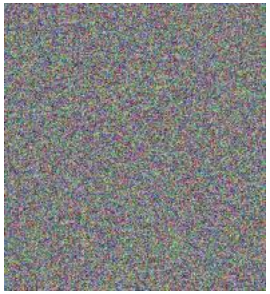


- Weakness:
  - Documents and images are not suitable for ECB encryption since patterns in the plaintext are repeated in the ciphertext:

ECB

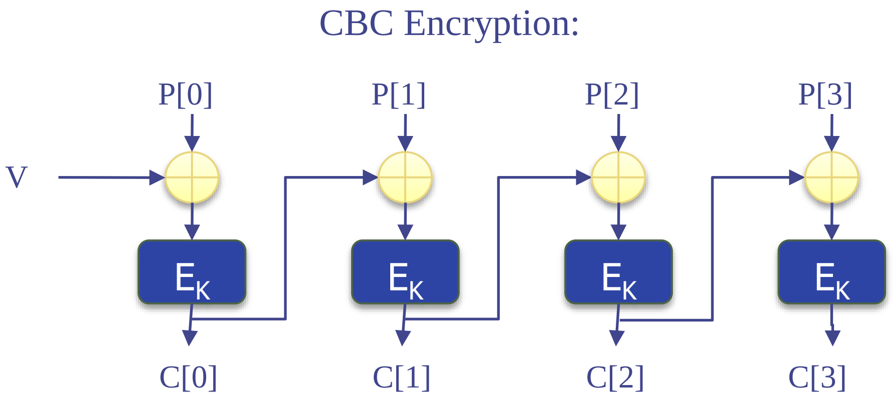


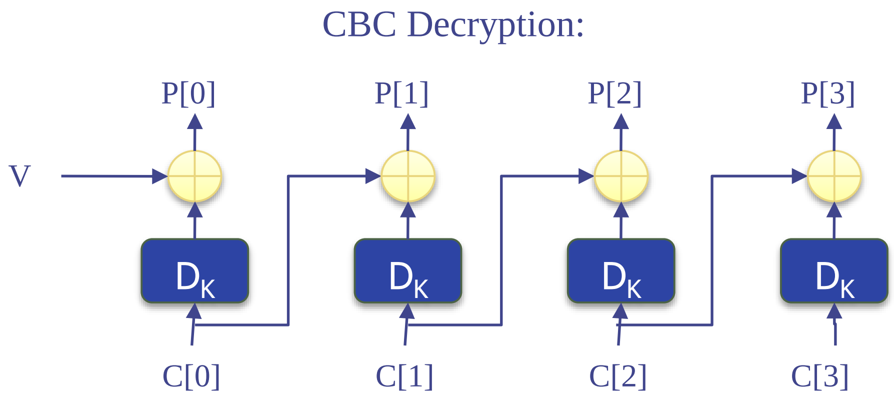
CBC



Cipher Block Chaining (CBC) Mode

- In Cipher Block Chaining (CBC) Mode
  - The previous ciphertext block is combined with the current plaintext block $C[i] = E_K(C[i-1] \oplus P[i])$
  - $C[-1] = V$, a random block separately transmitted encrypted (known as the initialization vector)
  - Decryption: $P[i] = C[i-1] \oplus D_K(C[i])$





Strengths and Weaknesses of CBC

- Strengths:
  - Doesn’t show patterns in the plaintext
  - Is the most common mode
  - Is fast and relatively simple

- Weaknesses:
  - CBC requires the reliable transmission of all the blocks sequentially
  - CBC is not suitable for applications that allow packet losses (e.g., music and video streaming)

Java AES Encryption Example

- Source

  http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html

- Generate an AES key

```
          keygen =                 .getInstance("AES");
      aesKey = keygen.generateKey();
```

- Create a cipher object for AES in ECB mode and PKCS5 padding

```
          aesCipher;
aesCipher =         .getInstance("AES/ECB/PKCS5Padding");
```

- Encrypt

```
aesCipher.init(        .ENCRYPT_MODE, aesKey);
byte[] plaintext = "My secret message".getBytes();
byte[] ciphertext = aesCipher.doFinal(plaintext);
```

- Decrypt

```
aesCipher.init(        .DECRYPT_MODE, aesKey);
byte[] plaintext1 = aesCipher.doFinal(ciphertext);
```