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.

ASE Rounds.

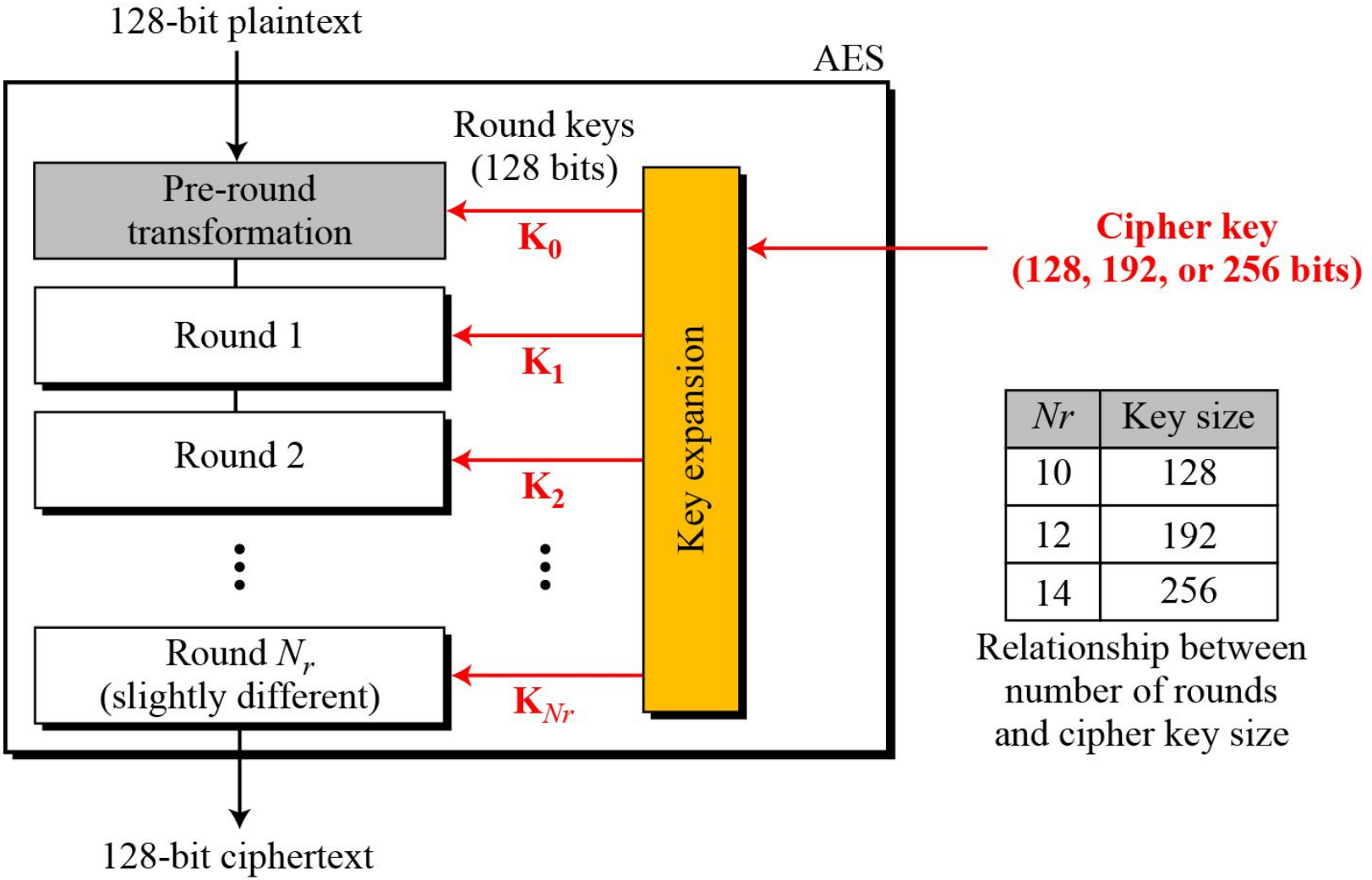
AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits. It uses 10, 12, or 14 rounds. The key size, which can be 128, 192, or 256 bits, depends on the number of rounds.

Note

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.

AES encryption cipher

General design of AES encryption cipher

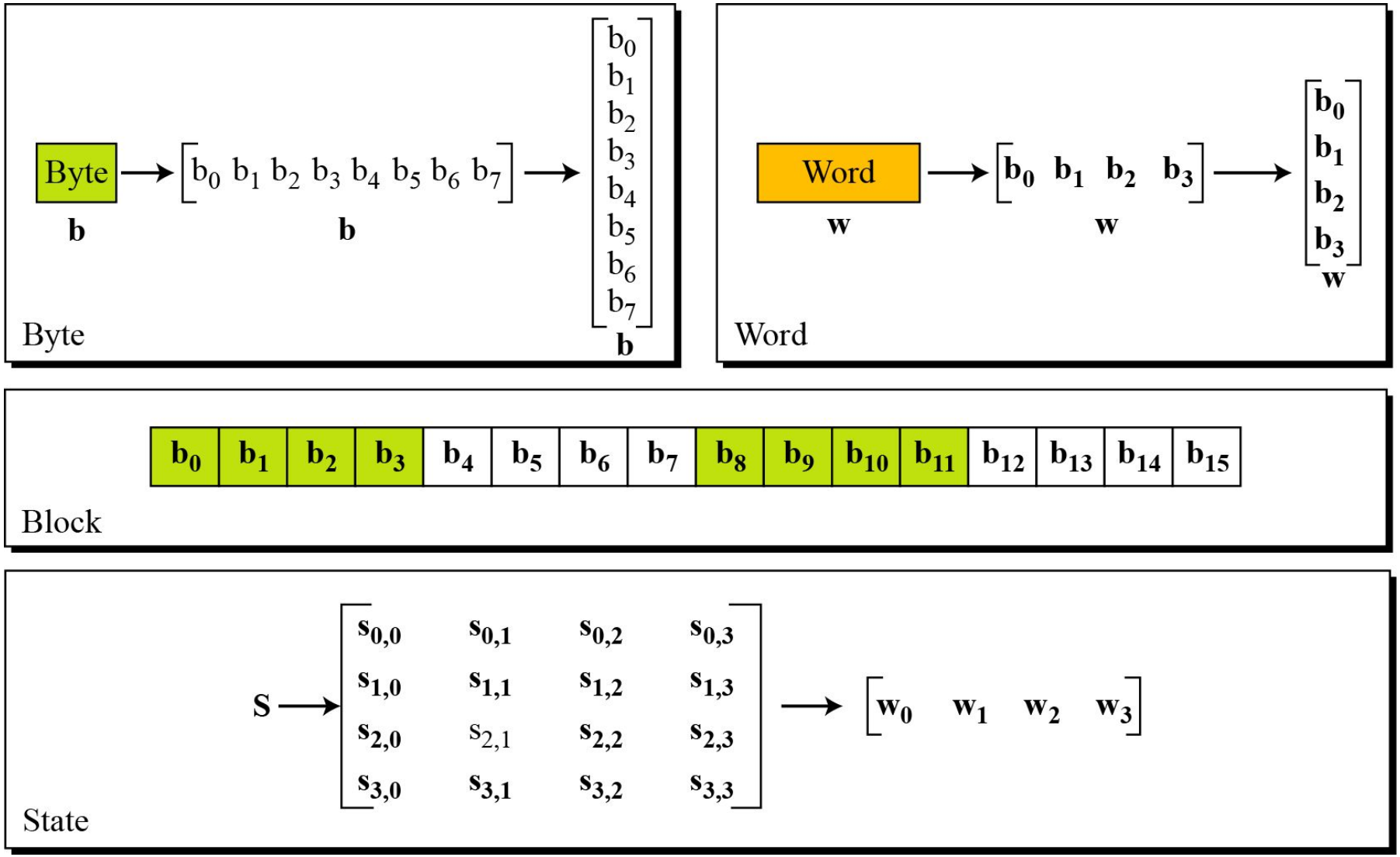


Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

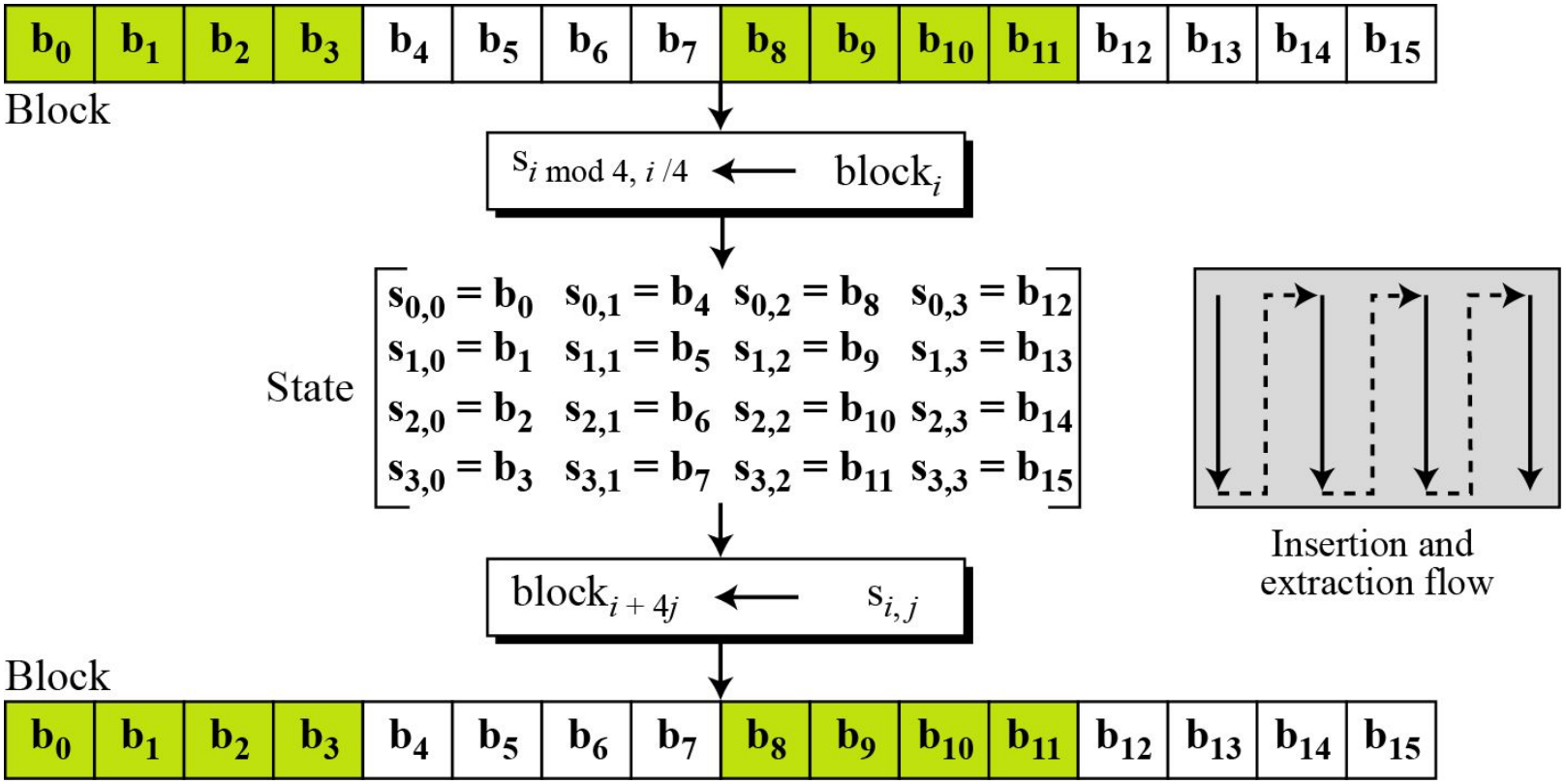
AES Data Units

Data units used in AES



AES Data Units

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



AES Data Units

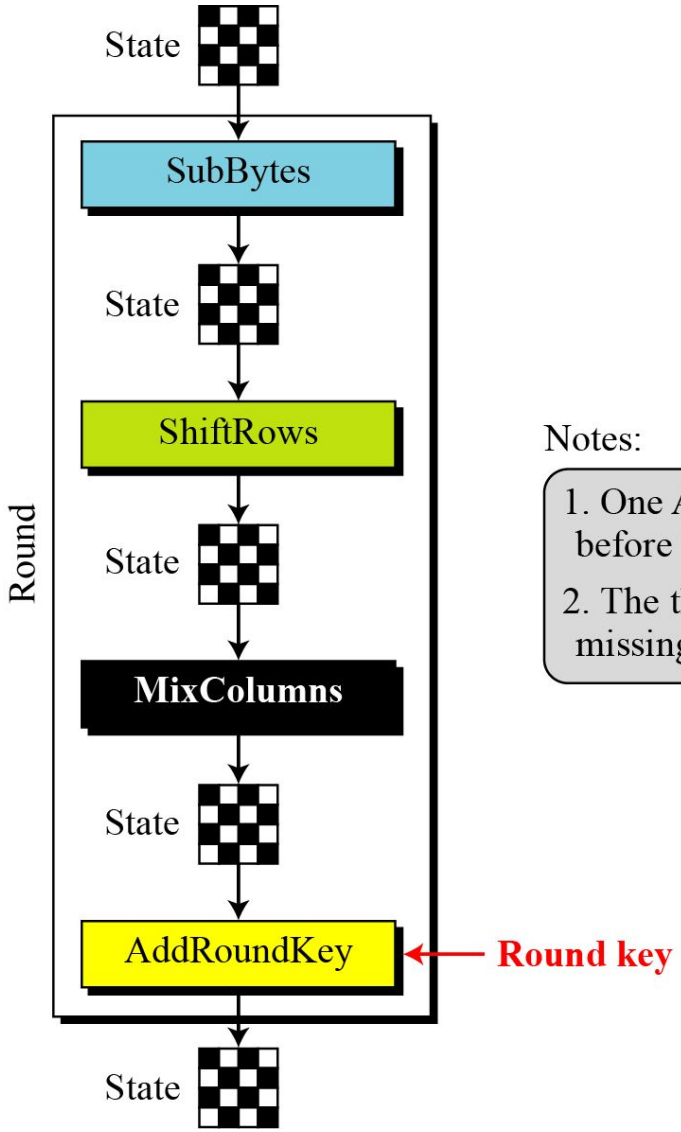
Changing plaintext to state

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	**17**	19	19

$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & \mathbf{17} \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix}$$ State

Structure of Each Round

Structure of each round at the encryption site



Notes:

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

TRANSFORMATIONS IN EACH ROUND

To provide security, AES uses four types of transformations:

1. *Substitution*
2. *Permutation*
3. *Mixing*
4. *Key-adding*

Substitution

AES, like DES, uses substitution. AES uses two invertible transformations.

SubBytes

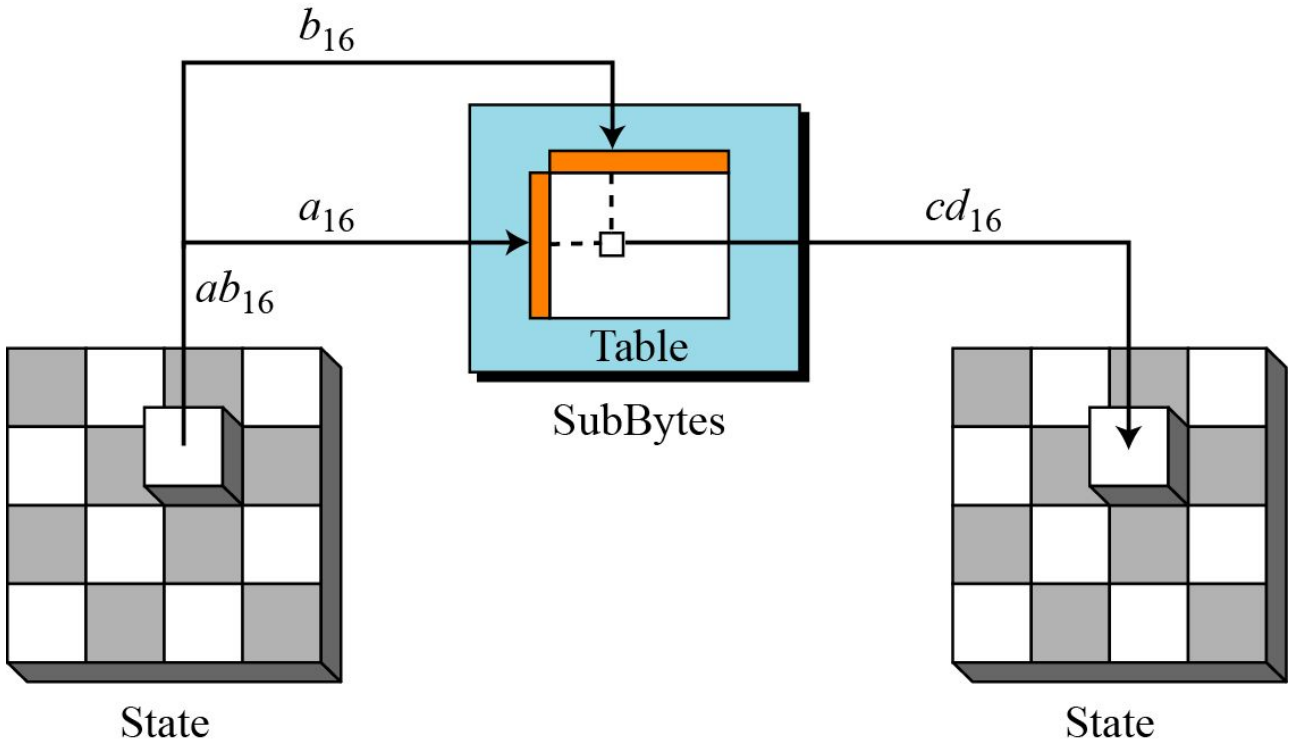
The first transformation, SubBytes, is used at the encryption site. To substitute a byte, we interpret the byte as two hexadecimal digits.

Note

The SubBytes operation involves 16 independent byte-to-byte transformations.

Substitution

SubBytes transformation



Substitution

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

Substitution

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

Substitution

InvSubBytes

Table 7.2 *InvSubBytes transformation table*

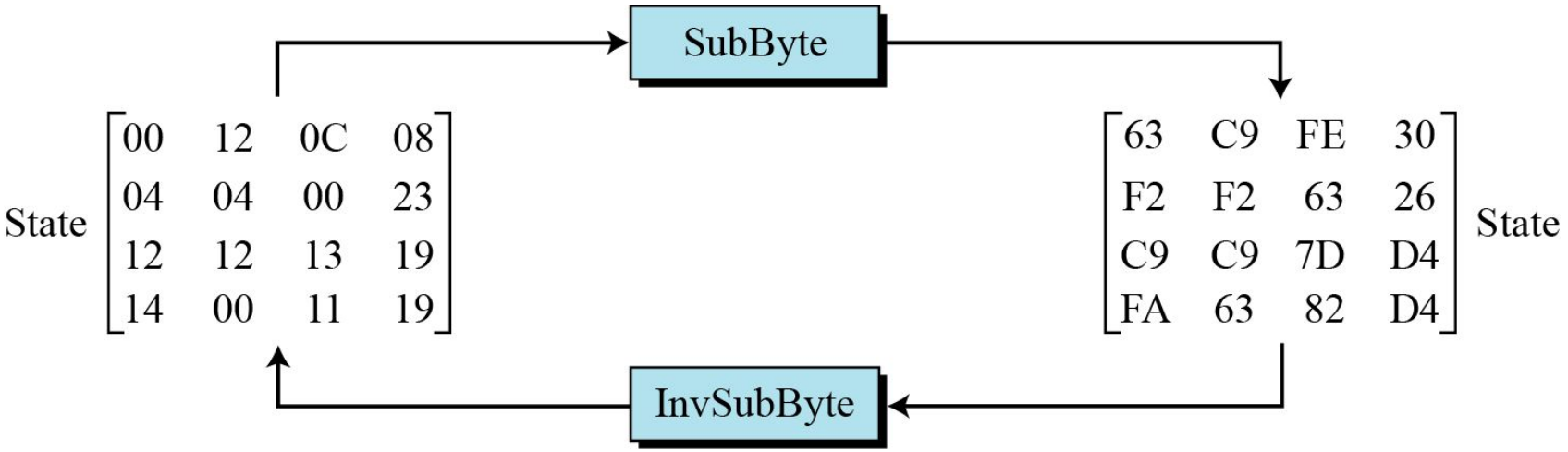
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	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

InvSubBytes (Continued)

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

Substitution

SubBytes transformation

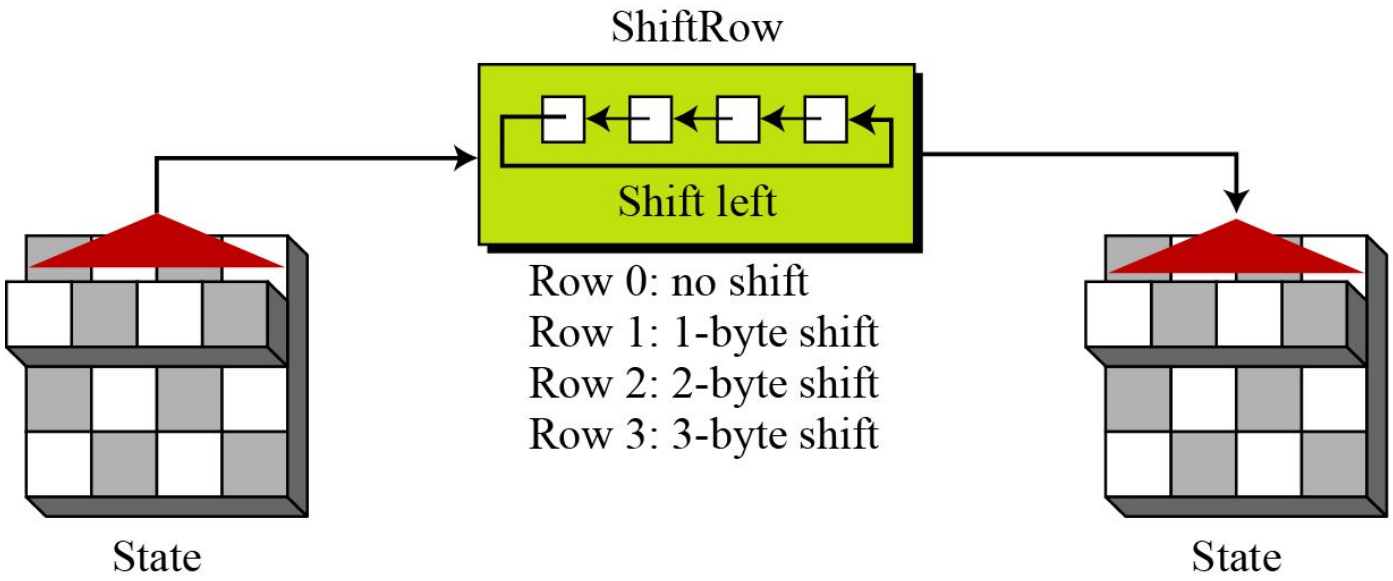


Permutation

ShiftRows

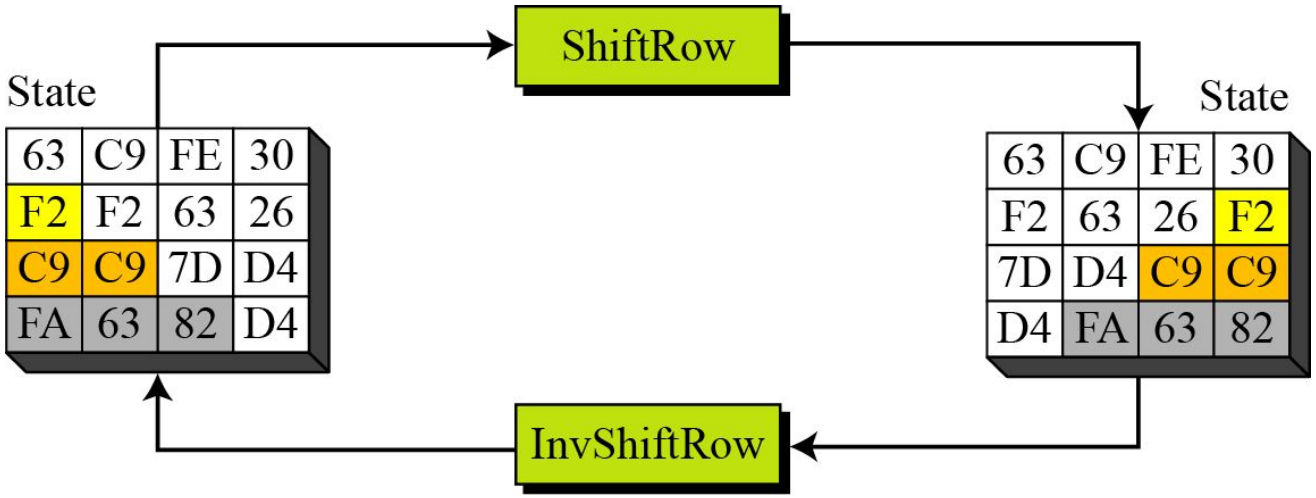
In the encryption, the transformation is called ShiftRows, which permutes the bytes.

ShiftRows transformation



Permutation

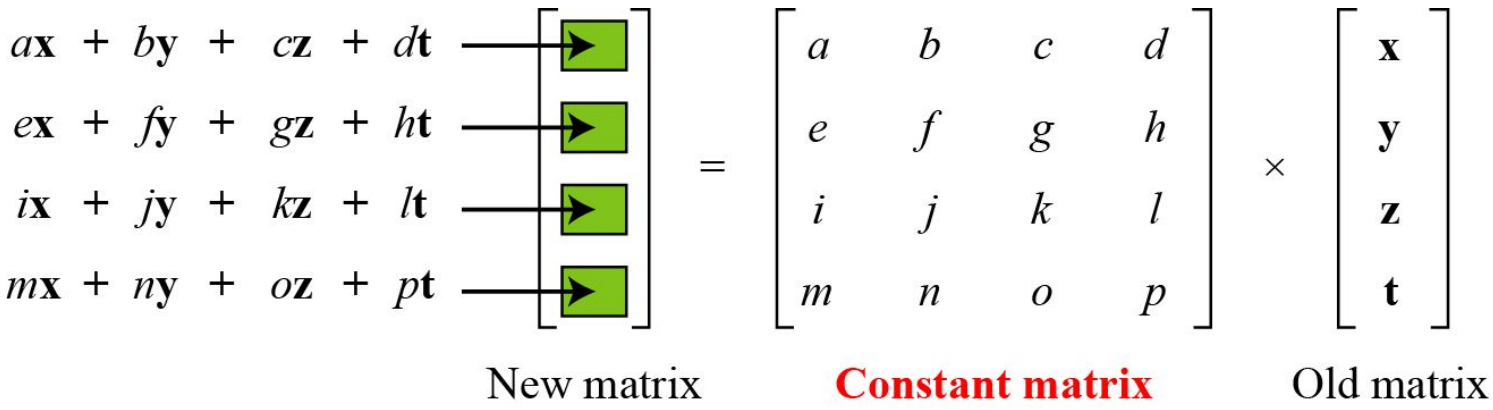
- Figure shows how a state is transformed using ShiftRows transformation.
- The figure also shows that InvShiftRows transformation creates the original state.



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.

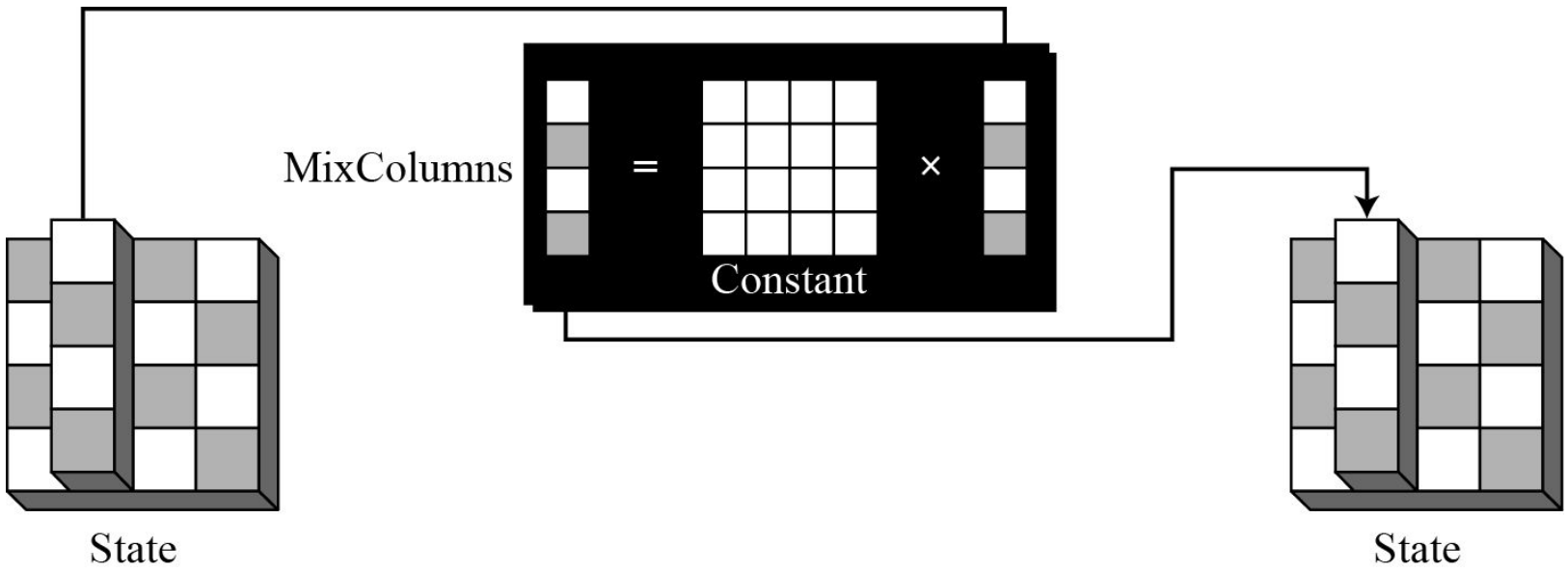
Mixing bytes using matrix multiplication



MixColumns

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

MixColumns transformation



Key Adding

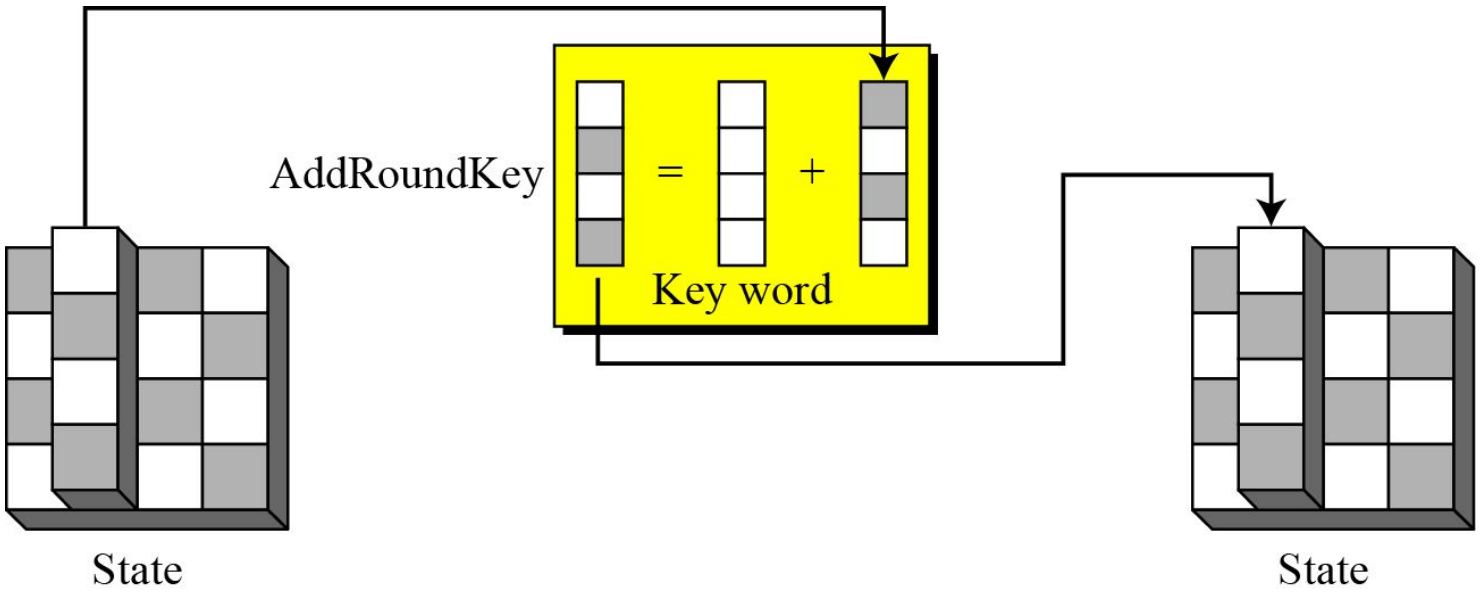
AddRoundKey

AddRoundKey proceeds one column at a time. AddRoundKey adds a round key word with each state column matrix; the operation in AddRoundKey is matrix addition.

Note

The AddRoundKey transformation is the inverse of itself.

AddRoundKey transformation



Algorithm 7.4 *Pseudocode for AddRoundKey transformation*

```
AddRoundKey (S)
{
    for (c = 0 to 3)
        s_c ← s_c ⊕ w_round + 4c
}
```

KEY EXPANSION

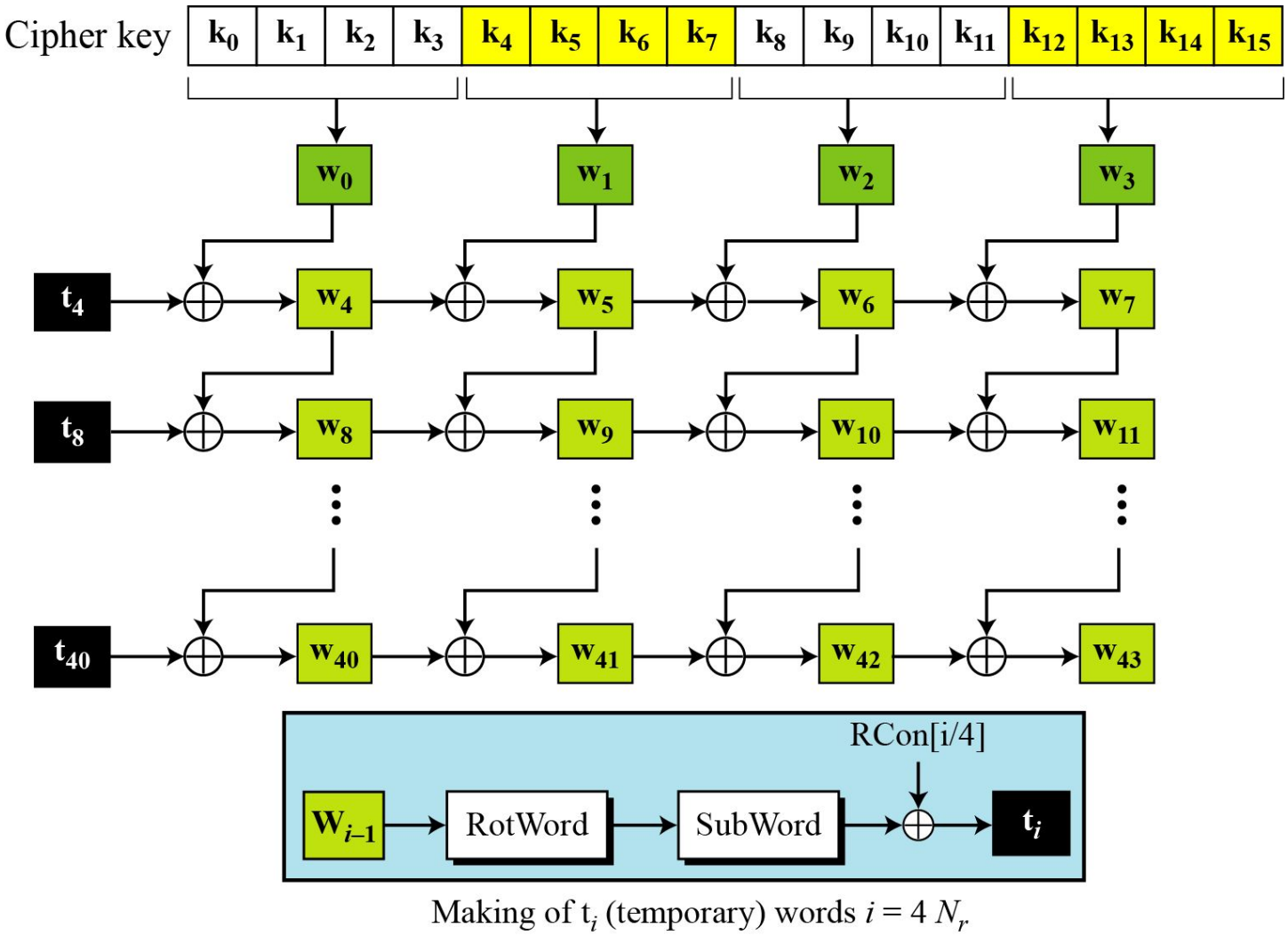
- *To create round keys for each round, AES uses a key-expansion process.*
- *If the number of rounds is N_r , the key-expansion routine creates $N_r + 1$ 128-bit round keys from one single 128-bit cipher key.*

KEY EXPANSION

Table 7.3 *Words for each round*

Round	*Words*			
Pre-round	$\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
1	$\mathbf{w}_4$	$\mathbf{w}_5$	$\mathbf{w}_6$	$\mathbf{w}_7$
2	$\mathbf{w}_8$	$\mathbf{w}_9$	$\mathbf{w}_{10}$	$\mathbf{w}_{11}$
. . .	. . .			
N_r	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

Key Expansion in AES-128



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

Table 7.4 *RCon constants*

Round	*Constant (RCon)*	*Round*	*Constant (RCon)*
1	$(\underline{\mathbf{01}}\ 00\ 00\ 00)_{16}$	6	$(\underline{\mathbf{20}}\ 00\ 00\ 00)_{16}$
2	$(\underline{\mathbf{02}}\ 00\ 00\ 00)_{16}$	7	$(\underline{\mathbf{40}}\ 00\ 00\ 00)_{16}$
3	$(\underline{\mathbf{04}}\ 00\ 00\ 00)_{16}$	8	$(\underline{\mathbf{80}}\ 00\ 00\ 00)_{16}$
4	$(\underline{\mathbf{08}}\ 00\ 00\ 00)_{16}$	9	$(\underline{\mathbf{1B}}\ 00\ 00\ 00)_{16}$
5	$(\underline{\mathbf{10}}\ 00\ 00\ 00)_{16}$	10	$(\underline{\mathbf{36}}\ 00\ 00\ 00)_{16}$

Algorithm 7.5 *Pseudocode for key expansion in AES-128*

```
KeyExpansion ([key0 to key15], [w0 to w43])
{
    for (i = 0 to 3)
        wi ← key4i + key4i+1 + key4i+2 + key4i+3

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

Table 7.5 shows how the keys for each round are calculated assuming that the 128-bit cipher key agreed upon by Alice and Bob is (24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87)$_{16}$.

Table 7.5 *Key expansion example*

Round	*Values of t's*	*First word in the round*	*Second word in the round*	*Third word in the round*	*Fourth word in the round*
—		$w_{00} = 2475A2B3$	$w_{01} = 34755688$	$w_{02} = 31E21200$	$w_{03} = 13AA5487$
1	AD20177D	$w_{04} = 8955B5CE$	$w_{05} = BD20E346$	$w_{06} = 8CC2F146$	$w_{07} = 9F68A5C1$
2	470678DB	$w_{08} = CE53CD15$	$w_{09} = 73732E53$	$w_{10} = FFB1DF15$	$w_{11} = 60D97AD4$
3	31DA48D0	$w_{12} = FF8985C5$	$w_{13} = 8CFAAB96$	$w_{14} = 734B7483$	$w_{15} = 2475A2B3$
4	47AB5B7D	$w_{16} = B822dcb8$	$w_{17} = 34D8752E$	$w_{18} = 479301AD$	$w_{19} = 54010FFA$
5	6C762D20	$w_{20} = D454F398$	$w_{21} = E08C86B6$	$w_{22} = A71F871B$	$w_{23} = F31E88E1$
6	52C4F80D	$w_{24} = 86900B95$	$w_{25} = 661C8D23$	$w_{26} = C1030A38$	$w_{27} = 321D82D9$
7	E4133523	$w_{28} = 62833EB6$	$w_{29} = 049FB395$	$w_{30} = C59CB9AD$	$w_{31} = F7813B74$
8	8CE29268	$w_{32} = EE61ACDE$	$w_{33} = EAFE1F4B$	$w_{34} = 2F62A6E6$	$w_{35} = D8E39D92$
9	0A5E4F61	$w_{36} = E43FE3BF$	$w_{37} = 0EC1FCF4$	$w_{38} = 21A35A12$	$w_{39} = F940C780$
10	3FC6CD99	$w_{40} = DBF92E26$	$w_{41} = D538D2D2$	$w_{42} = F49B88C0$	$w_{43} = 0DDB4F40$

7.3.2 Key Expansion in AES-192 and AES-256

Key-expansion algorithms in the AES-192 and AES-256 versions are very similar to the key expansion algorithm in AES-128, with the following differences:

Key-Expansion Analysis

The key-expansion mechanism in AES has been designed to provide several features that thwart the cryptanalyst.

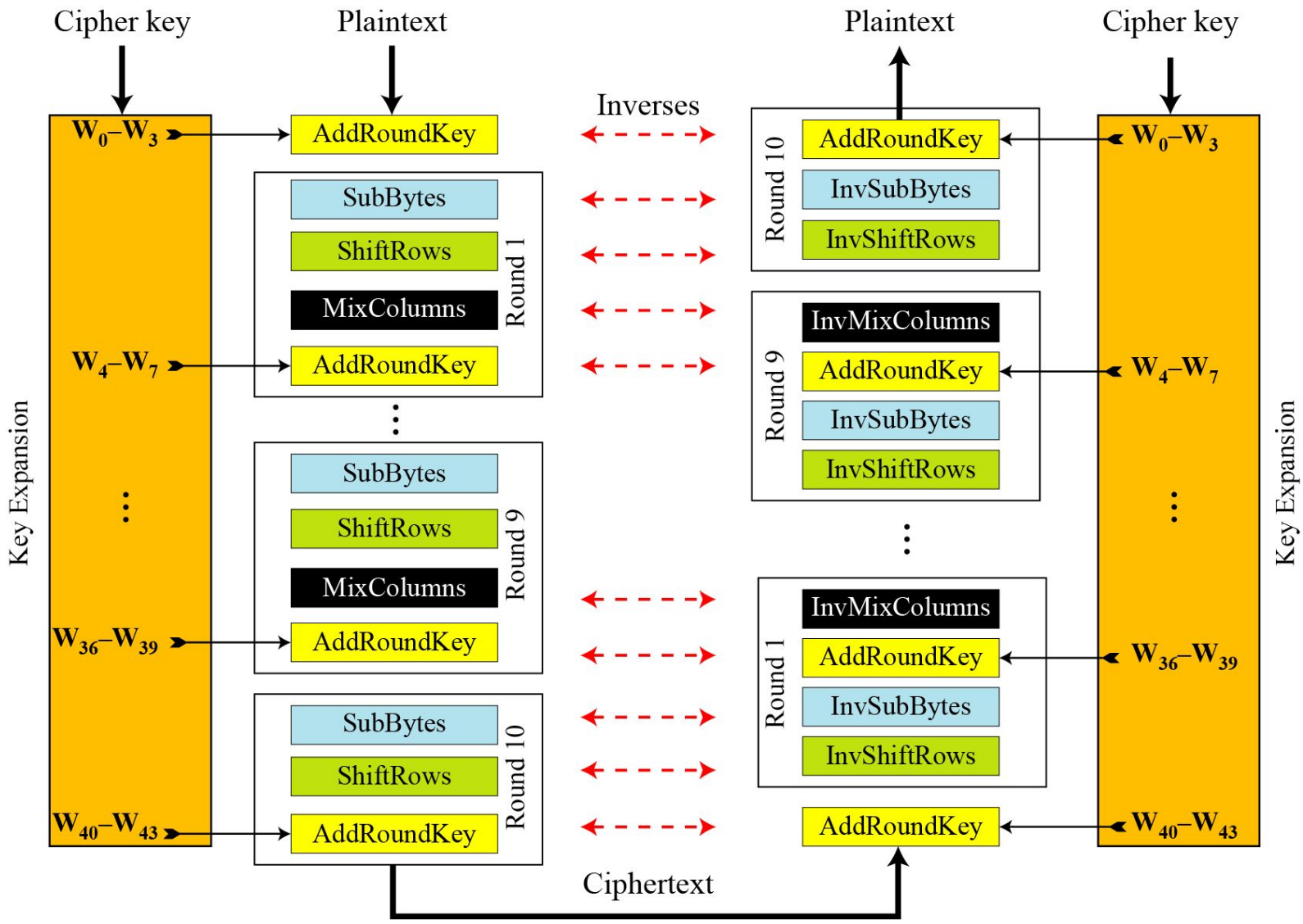
7-4 CIPHERS

AES uses four types of transformations for encryption and decryption. In the standard, the encryption algorithm is referred to as the cipher and the decryption algorithm as the inverse cipher.

Topics discussed in this section:

Original Design

Ciphers and inverse ciphers of the original design



ANALYSIS OF AES

This section is a brief review of the three characteristics of AES.

Topics discussed in this section:

1. Security
2. Implementation
3. Simplicity and Cost

7.6.1 Security

AES was designed after DES. Most of the known attacks on DES were already tested on AES.

Brute-Force Attack

AES is definitely more secure than DES due to the larger-size key.

Statistical Attacks

Numerous tests have failed to do statistical analysis of the ciphertext.

Differential and Linear Attacks

There are no differential and linear attacks on AES as yet.

7.6.2 Implementation

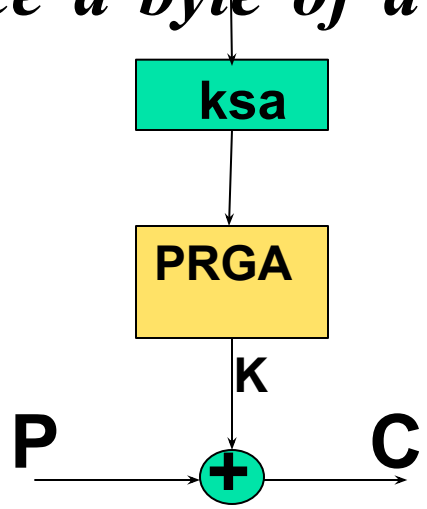
AES can be implemented in software, hardware, and firmware. The implementation can use table lookup process or routines that use a well-defined algebraic structure.

7.6.3 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.

8.2.1 RC4

Developed by RSA Labs, RC4 is a symmetric, byte-oriented stream cipher with a variable length key size, in which a byte (8 bits) of a plaintext is exclusive-ored with a byte of key to produce a byte of a ciphertext.



RC4 HAS two main parts:

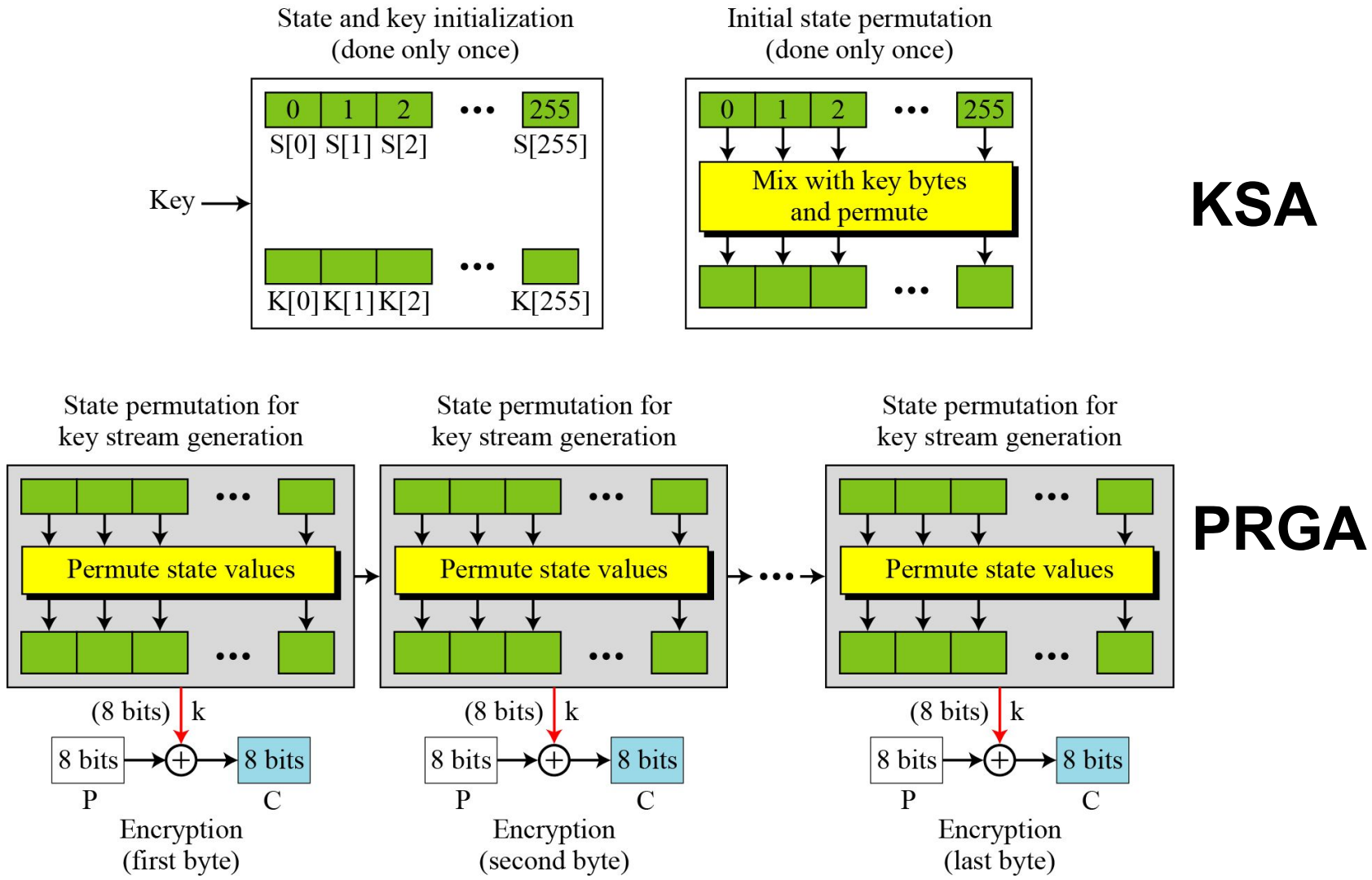
- **KSA (Key Scheduling Algorithm)**
- **PRGA (Pseudo Random Generation Algorithm)**

State

RC4 is based on the concept of a state.

S[0]	S[1]	S[2]	···	S[255]

Figure 8.10 *The idea of RC4 stream cipher*



RC4 Key Schedule KSA

Starts with an array S of numbers: 0..255

- Use key to truly shuffle S
- S forms **internal state** of the cipher
- Given a key k of length L bytes

Scrambling Pseudocode :

for i = 0 to 255 do
S[i] = i
j = 0
for i = 0 to 255 do
j = (j + S[i] + k[i]) (mod 256)
swap (S[i], S[j])

RC4 PRGA and Encryption

Encryption involves XORing data bytes with output of the PRGA

- The PRGA initializes i and j to 0 and then loops over 4 basic operations: increase j, increase j using s[i], swap and output s[i]+s[j]

- **PRGA Pseudocode is:**

i = j = 0
for each message byte Mi
i = (i + 1) (mod 256)
j = (j + S[i]) (mod 256)
swap(S[i], S[j])
t = (S[i] + S[j]) (mod 256) ; Ki = S[t]
Encryption : Ci = Mi XOR S[t]

RC4 Encryption Example

Lets consider the stream cipher RC4, but instead of the full 256 bytes, we will use 8 x 3-bits.
That is, the state vector **S** is 8 x 3-bits.
We will operate on 3-bits of plaintext at a time since S can take the values 0 to 7, which can be represented as 3 bits.
Assume we use a 4 x 3-bit key of **K** = [1 2 3 6]. And a plaintext **P** = [1 2 2 2]

RC4 PRGA and Encryption

The first step is to generate the stream.
Initialise the state vector **S** and temporary vector **T**. **S** is initialised so the S[i] = i, and **T** is initialised so it is the key **K** (repeated as necessary).

S = [0 1 2 3 4 5 6 7]

T = [1 2 3 6 1 2 3 6]

Now perform the initial permutation on S.

```
j = 0;
for i = 0 to 7 do
j = (j + S[i] + T[i]) mod 8
Swap(S[i],S[j]);
end
```

For i = 0:

j = (0 + 0 + 1) mod 8 = 1

Swap(S[0],S[1]);

S = [1 0 2 3 4 5 6 7]

RC4 PRGA and Encryption

For i = 1:

j = 3

Swap(S[1],S[3])

S = [1 3 2 0 4 5 6 7];

For i = 2:

j = 0

Swap(S[2],S[0]);

S = [2 3 1 0 4 5 6 7];

For i = 3:

j = 6;

Swap(S[3],S[6])

S = [2 3 1 6 4 5 0 7];

RC4 PRGA and Encryption

For i = 4:

j = 3

Swap(S[4],S[3])

S = [2 3 1 4 6 5 0 7];

For i = 5:

j = 2

Swap(S[5],S[2]);

S = [2 3 5 4 6 1 0 7];

For i = 6:

j = 5;

Swap(S[6],S[4])

S = [2 3 5 4 0 1 6 7];

For i = 7:

j = 2;

Swap(S[7],S[2])

S = [2 3 7 4 0 1 6 5];

Hence, our initial permutation of **S = [2 3 7 4 0 1 6 5];**

RC4 PRGA and Encryption

Now we generate 3-bits at a time, k, that we XOR with each 3-bits of plaintext to produce the ciphertext. The 3-bits k is generated by:

```
i, j = 0;
while (true) {
i = (i + 1) mod 8;
j = (j + S[i]) mod 8;
Swap (S[i], S[j]);
t = (S[i] + S[j]) mod 8;
k = S[t]; }
```

The first iteration:

S = [2 3 7 4 0 1 6 5]
i = (0 + 1) mod 8 = 1
j = (0 + S[1]) mod 8 = 3
Swap(S[1],S[3])
S = [2 4 7 3 0 1 6 5]
t = (S[1] + S[3]) mod 8 = 7
k = S[7] = 5
Remember, **P** = [1 2 2 2]

RC4 PRGA and Encryption

Remember, **P** = [1 2 2 2]
So our first 3-bits of ciphertext is obtained by: k XOR P
5 XOR 1 = 101 XOR 001 = 100 = **4**

The second iteration:
S = [2 4 7 3 0 1 6 5]
i = (1 + 1) mod 8 = 2
j = (2 + S[2]) mod 8 = 1
Swap(S[2],S[1])
S = [2 7 4 3 0 1 6 5]
t = (S[2] + S[1]) mod 8 = 3
k = S[3] = 3
Second 3-bits of ciphertext are:
3 XOR 2 = 011 XOR 010 = 001 = **1**

RC4 PRGA and Encryption

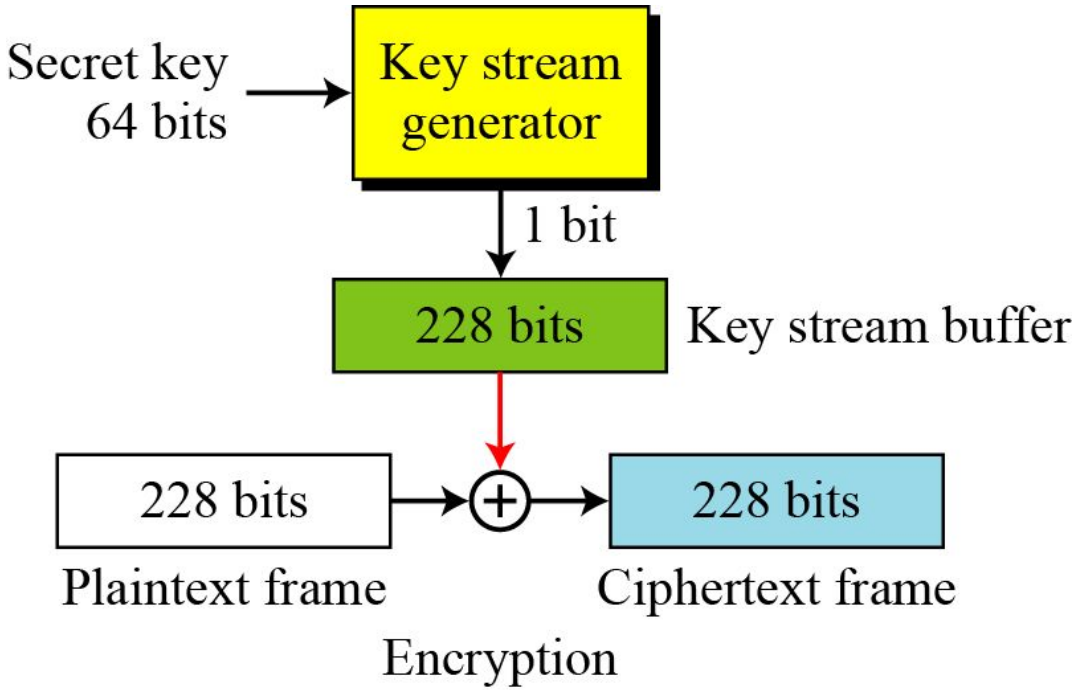
After 4 iterations:

To encrypt the plaintext stream **P** = [1 2 2 2] with key **K** = [1 2 3 6] using our simplified RC4 stream cipher we get **C** = [4 1 2 0].
(or in binary: **P** = 001010010010, **K** = 001010011110 and **C** = 100001010000)
Simplified

8.2.2 A5/1

A5/1 (a member of the A5 family of ciphers) is used in the Global System for Mobile Communication (GSM), a network for mobile telephone communication..

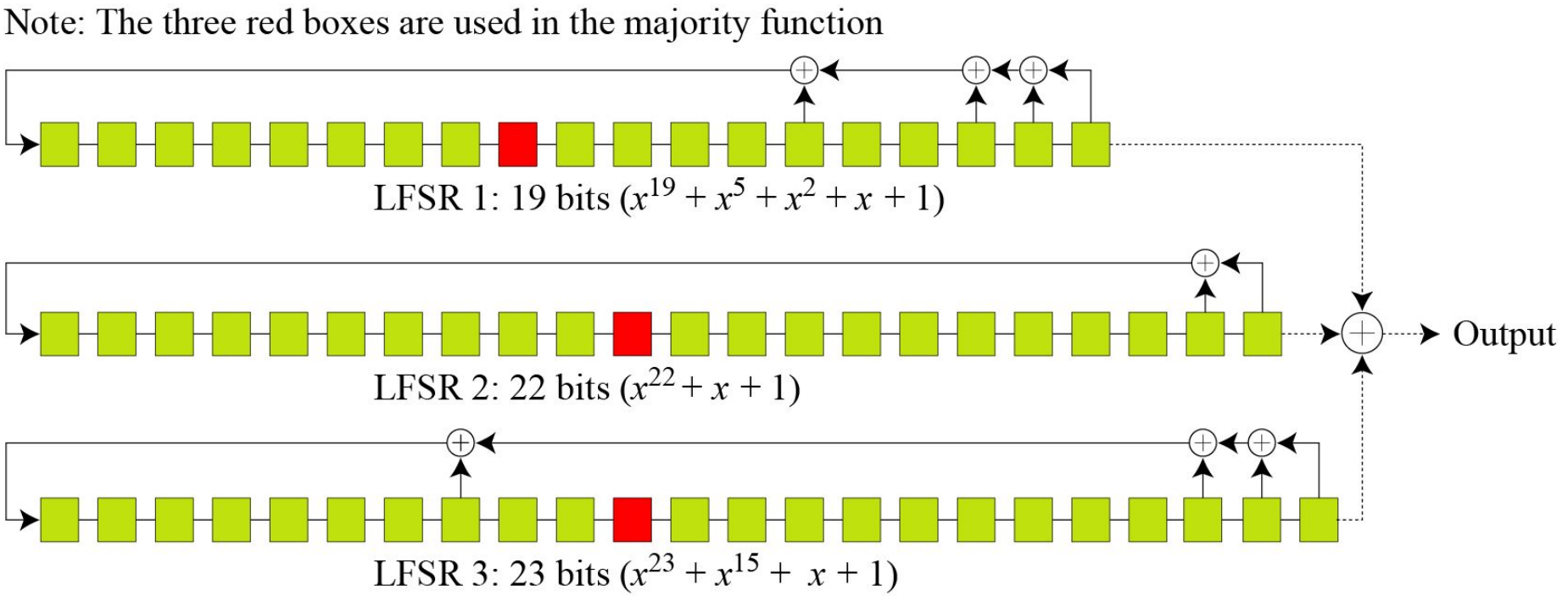
Figure 8.11 *General outline of A5/1*



Key Generator

A5/1 uses three LFSRs with 19, 22, and 23 bits.

Figure 8.12 *Three LFSR's in A5/1*



Example 8.7

At a point of time the clocking bits are 1, 0, and 1. Which LFSR is clocked (shifted)?

Solution

The result of Majority (1, 0, 1) = 1. LFSR1 and LAFS3 are shifted, but LFSR2 is not.

Encryption/Decryption

The bit streams created from the key generator are buffered to form a 228-bit key that is exclusive-ored with the plaintext frame to create the ciphertext frame. Encryption/decryption is done one frame at a time.

8-3 OTHER ISSUES

Encipherment using symmetric-key block or stream ciphers requires discussion of other issues.

Topics discussed in this section:

8.3.1 Key Management

8.3.2 Key Generation

8.3.1 Key Management

Alice and Bob need to share a secret key between themselves to securely communicate using a symmetric-key cipher. If there are n entities in the community,
n(n − 1)/2 keys are needed.

Note

Key management is discussed in Chapter 15.

8.3.2 Key Generation

Different symmetric-key ciphers need keys of different sizes. The selection of the key must be based on a systematic approach to avoid a security leak. The keys need to be chosen randomly. This implies that there is a need for random (or pseudorandom) number generator.

Note

Random number generators are discussed in Appendix K.