Advanced Encryption Standard (AES)

G.P. Biswas
Prof/CSE, IIT-Dhanbad

Outline

- Overview of AES
- Inside Algorithm
- Its Mathematics
- Conclusions

Motivation behind AES

- A replacement of DES was needed due to its small Key and to have another standard symmetric encryption technique.
- In 1990's the cracking of DES algorithm became possible.
- Around 50 hrs. of brute-force attack allowed to crack system.
- 3DES secure but slow and DESX is developed.
- In 1997, National Institute of Standards and Technology (NIST) called for new proposal for a secure encryption system.

NIST requirements

- Block cipher must be supported with 128 bit block size
- Three key sizes like 128, 192 and 256 bits must be supported.
- It must be efficient both in software and hardware

About AES

- AES is an encryption standard chosen by the National Institute of Standards and Technology (NIST) and accepted in worldwide as a desirable algorithm to encrypt sensitive data.
- It is the most widely used symmetric ciphers .
- In 2001, Rijndael algorithm designed by Vincent Rijmen and John Daemon was declared as the winner of the competition.

In 1999, five finalist algorithms were announced:

- **Mars** by IBM Corporation
- **RC6** by RSA Laboratories
- **Rijndael**, by Joan Daemen and Vincent Rijmen
- **Serpent**, by Ross Anderson, Eli Biham and Lars Knudsen
- **Twofish**, by Bruce Schneier, John Kelsey, Doug Whiting, DavidWagner, Chris Hall and Niels Ferguson

- On 2, October 2000, **NIST** announced the selection of **Rijndael** as the **AES**.
- On 26, November , **AES** was formally approved as a US federal standard.

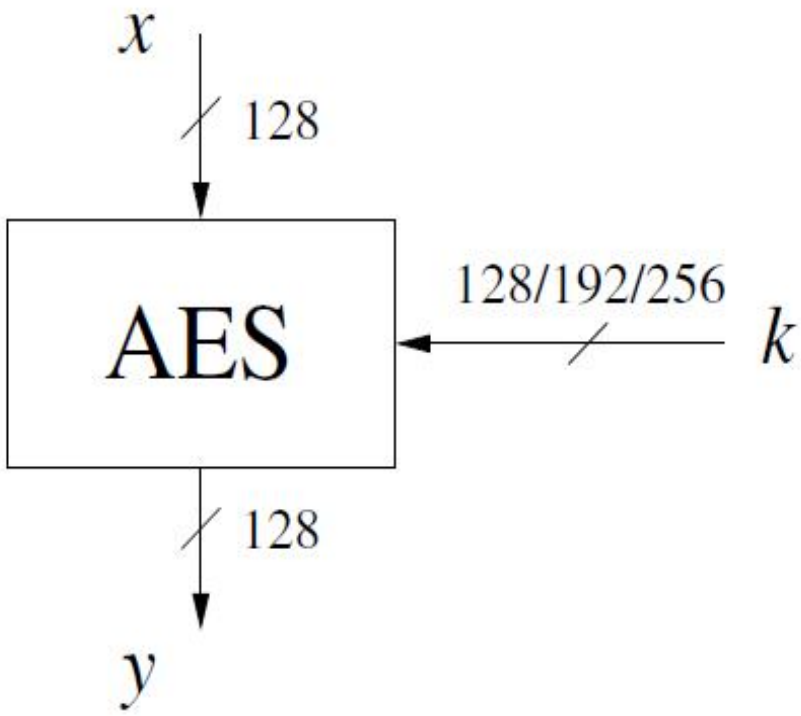
The internal Structure of AES is as follows

- **Byte substitution layer**
- **Diffusion layer**
- **Key addition layer**
- **Key schedule**

Key lengths and number of rounds for AES

key lengths	# rounds = n_r
128 bit	10
192 bit	12
256 bit	14

AES input/output parameters



Brief description of the layers

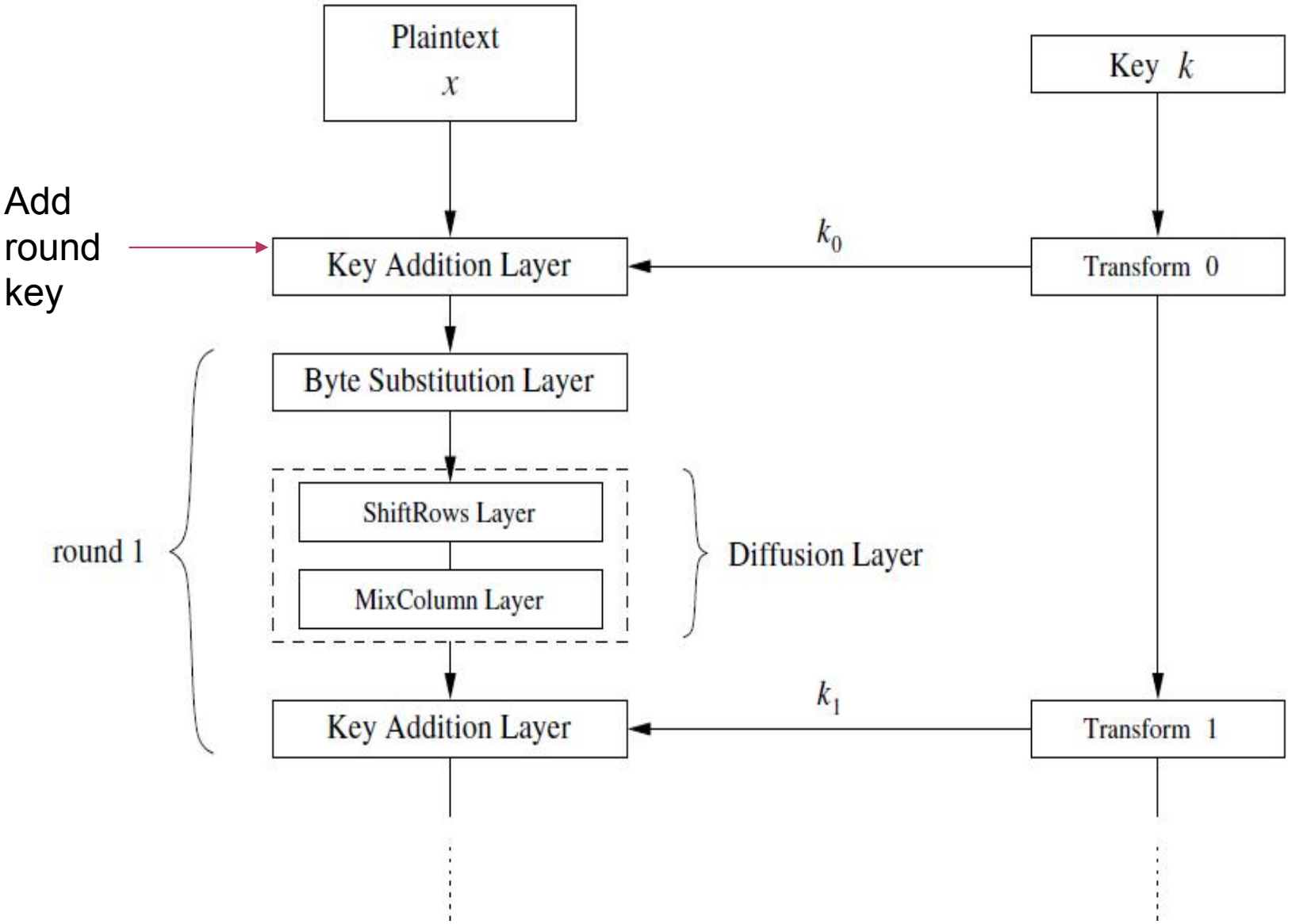
- **Key Addition layer:** A **128-bit** round key, or subkey, which has been derived from the main key in the key schedule, is XORed a state input (16 Bytes, defined later).

- **Byte Substitution layer (S-Box):** Each element of the state is nonlinearly transformed using a lookup table having special mathematical properties. This introduces confusion to the data.

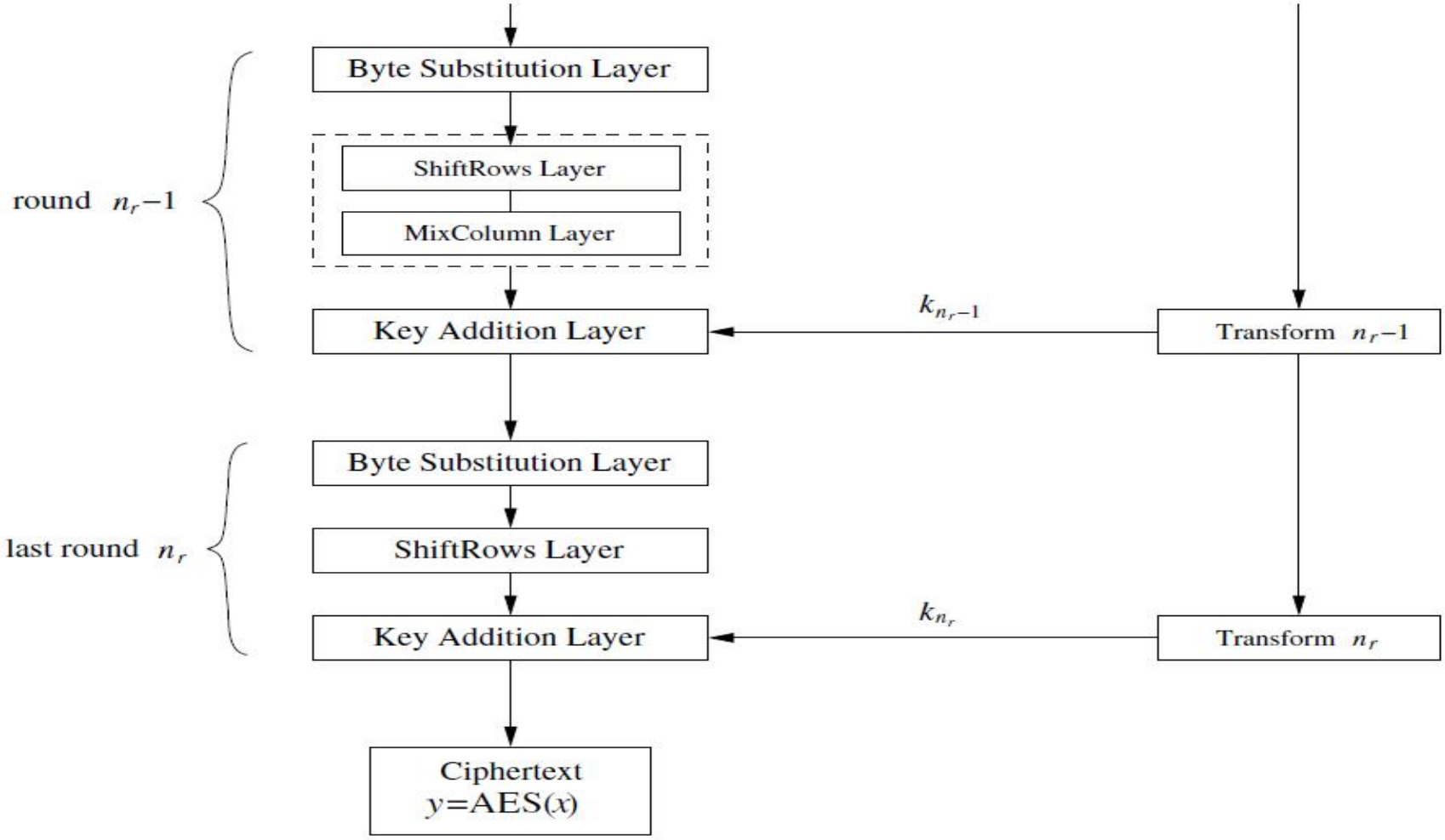
Contd…

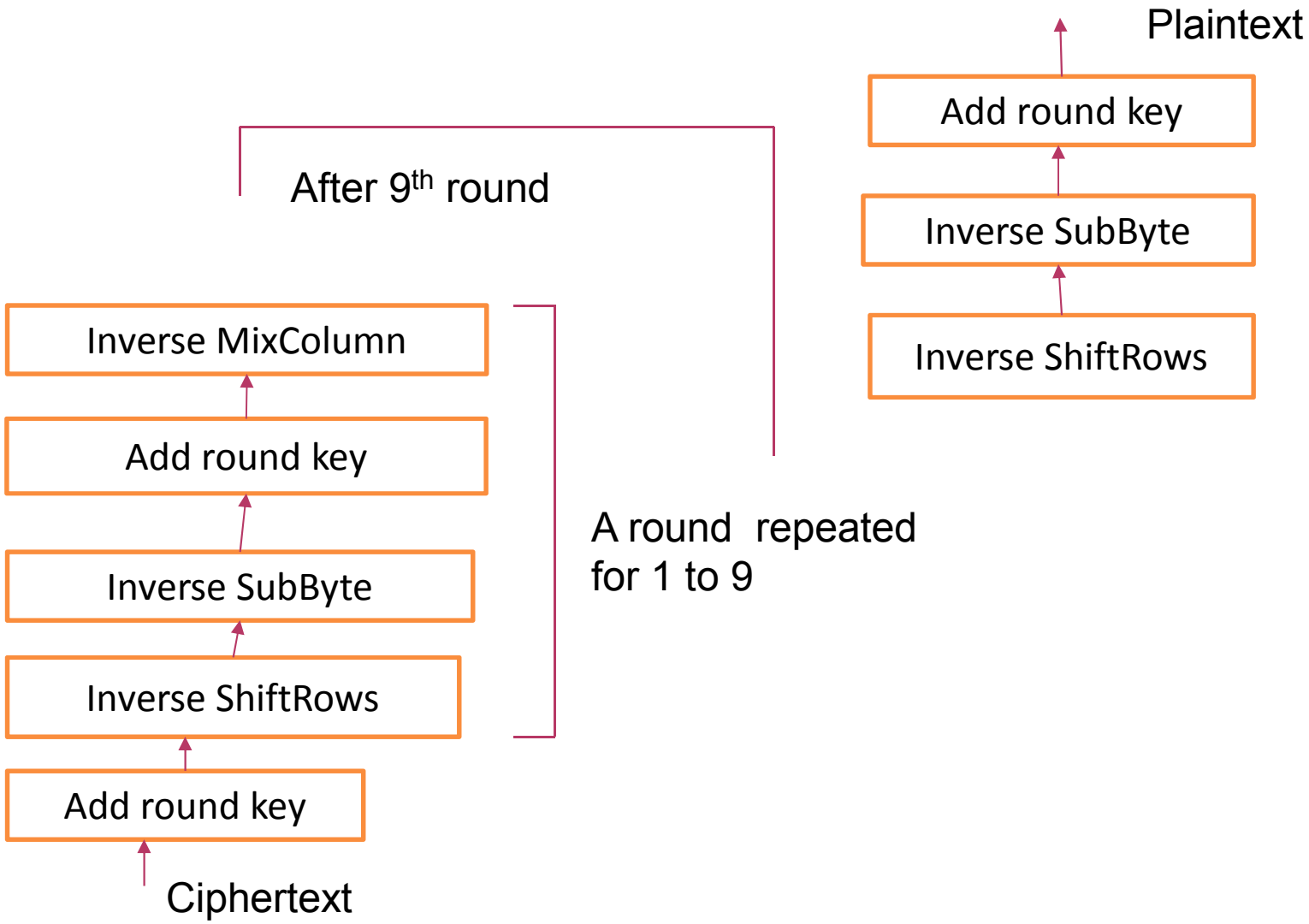
- **Diffusion layer:** It provides **diffusion** over all state bits. It consists of **two sublayers**, both of which perform linear operations:
  - **ShiftRows** layer permutes the data on a byte level.
  - **MixColumn** layer is a matrix operation which combines (mixes) blocks of four bytes.
- **Key scheduling algorithm:** Similar to DES, the key scheduling algorithm computes round subkeys (k0, k1, . . . , kn, n indicates no. of rounds used) from an AES secret key.

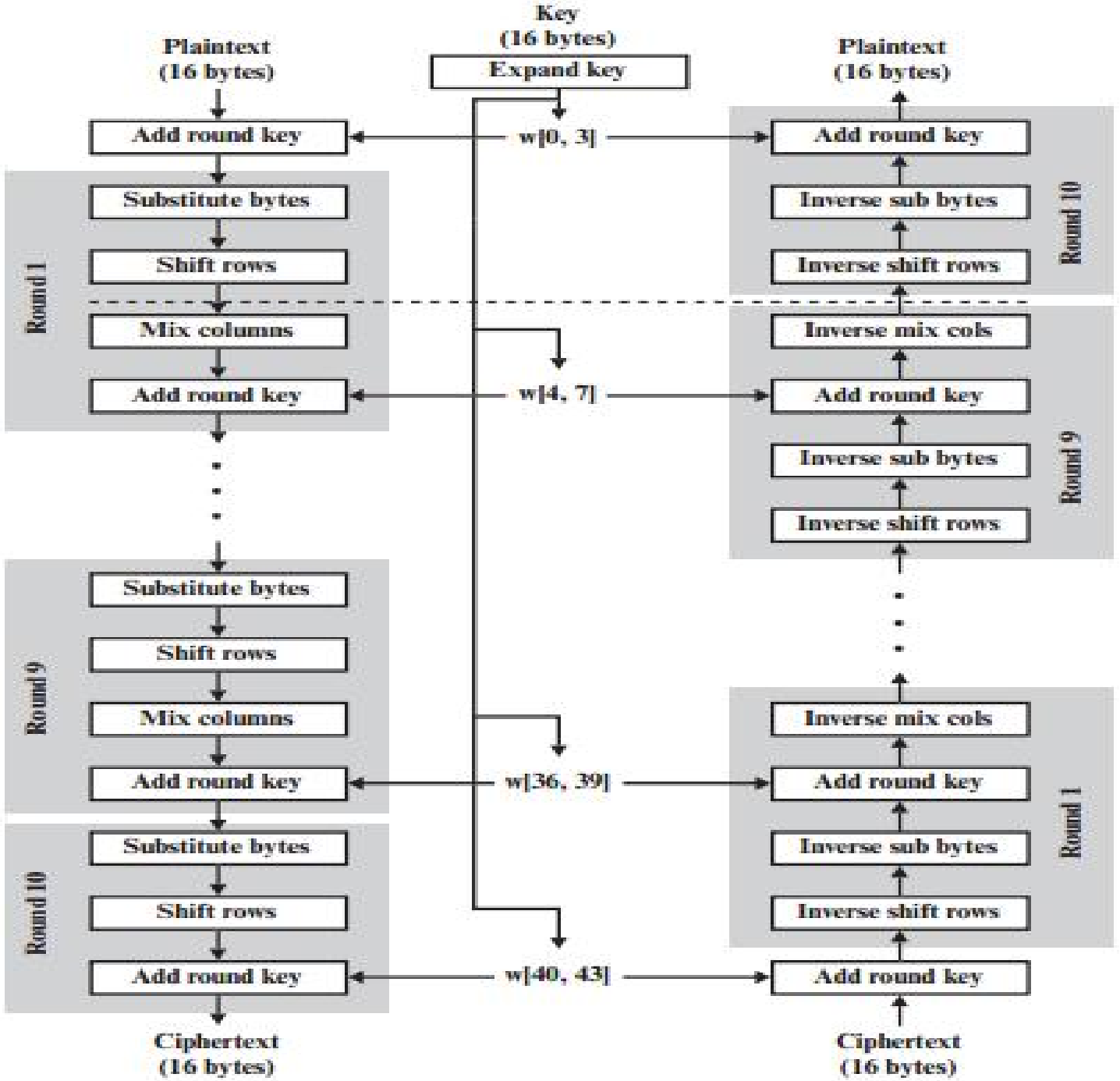
AES encryption block diagram



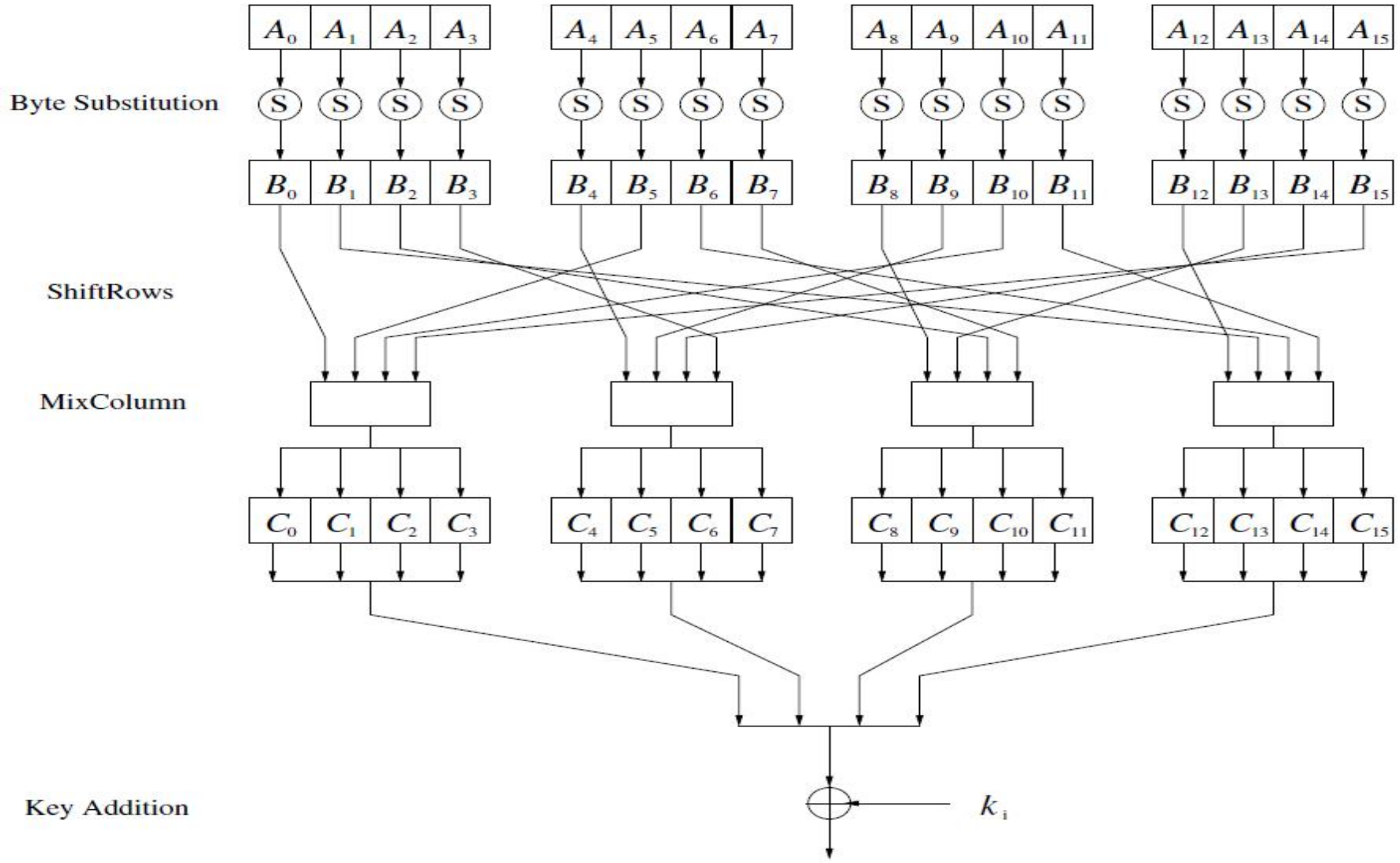
Contd…


Byte Substitution Layer
ShiftRows Layer
MixColumn Layer
round n_r-1
Key Addition Layer
k_{n_r-1}
Transform n_r-1
Byte Substitution Layer
ShiftRows Layer
last round n_r
Key Addition Layer
k_{n_r}
Transform n_r
Ciphertext
y=AES(x)


Plaintext
Add round key
Inverse SubByte
Inverse ShiftRows
After 9th round
Inverse MixColumn
Add round key
Inverse SubByte
Inverse ShiftRows
Add round key
Ciphertext
A round repeated
for 1 to 9


Plaintext
(16 bytes)
Key
(16 bytes)
Expand key
Plaintext
(16 bytes)
Add round key
w[0, 3]
Add round key
Substitute bytes
Shift rows
Inverse sub bytes
Inverse shift rows
Round 1
Round 10
Mix columns
Inverse mix cols
Add round key
w[4, 7]
Add round key
Inverse sub bytes
Inverse shift rows
Round 9
Round 9
Substitute bytes
Shift rows
Mix columns
Inverse mix cols
Add round key
w[36, 39]
Add round key
Inverse sub bytes
Inverse shift rows
Round 1
Round 10
Substitute bytes
Shift rows
Add round key
w[40, 43]
Add round key
Ciphertext
(16 bytes)
Ciphertext
(16 bytes)

AES round function for rounds 1,2, . . . ,*nr* −1



Contd…

- A 16-byte $B_0, \ldots, B_{15}$ is fed into the S-Box, which is a 16×16 Bytes size with each element of size 8 bits (0-255 values in 256 locations)

B0	B4	B8	B12
B1	B5	B9	B13
B2	B6	B10	B14
B3	B7	B11	B15

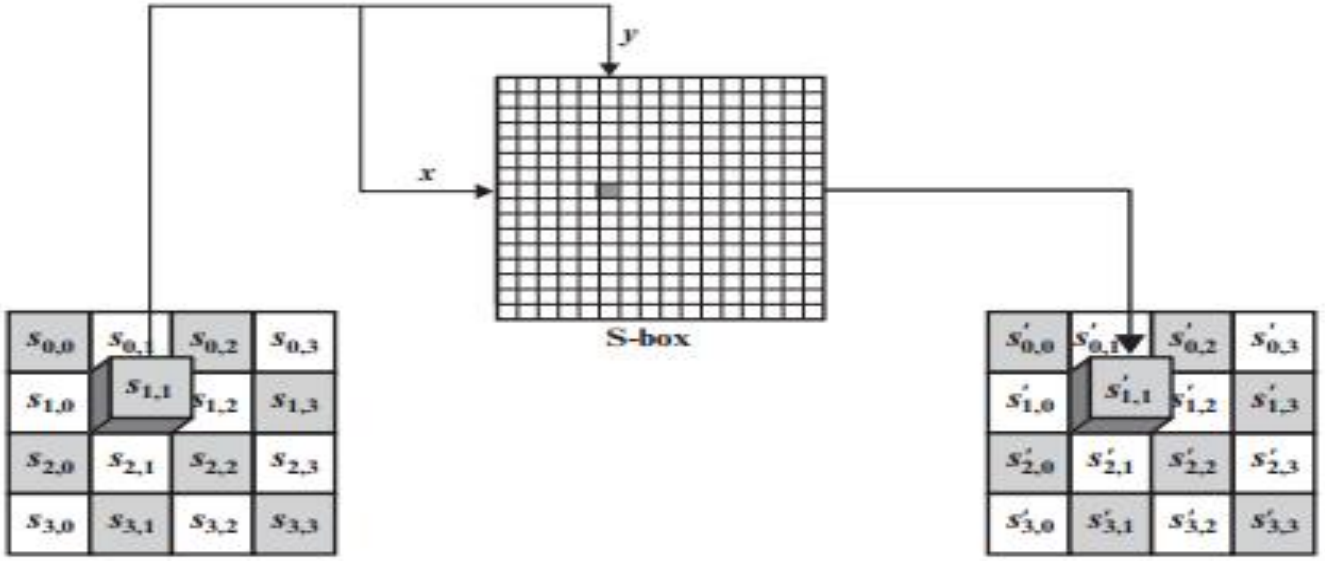
- The 16-byte output $B_0, \ldots, B_{15}$ *is permuted byte-wise in the ShiftRows layer* and mixed by the MixColumn transformation *c(x) by a matrix multiplication.*
- *Finally, the 128-bit subkey* k_i *is* XORed with the intermediate result

Byte Substitution Transformation

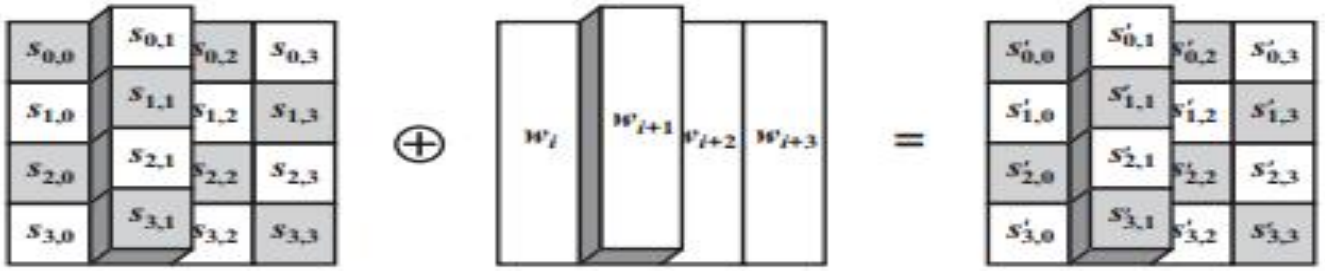
- In Byte Substitution layer (can be viewed as a row of 16 parallel S-Boxes), each 8-bit input is substituted using a S-Box and output 8 bits produced.
- Note that all 16 S-Boxes are identical, unlike DES where eight different S-Boxes are used.
- In the layer, each state byte A_i is replaced, i.e., substituted by B_i

$$S(A_i) = B_i.$$

- Each byte can be represented as xy (say), x and y are hexadecimal digits, then x and y represent Rows and Columns of the S-box, and their intersection gives the corresponding output byte.

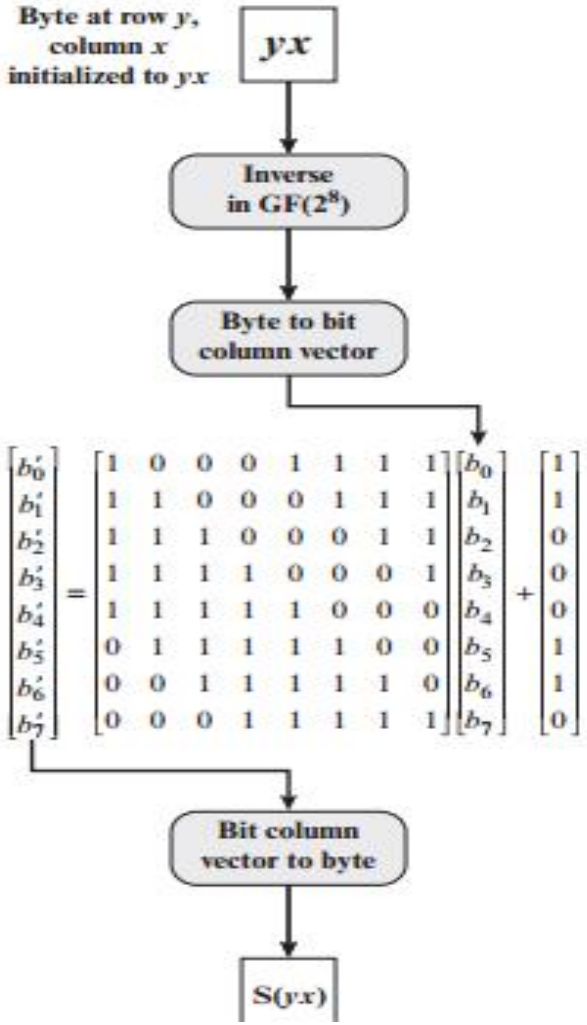


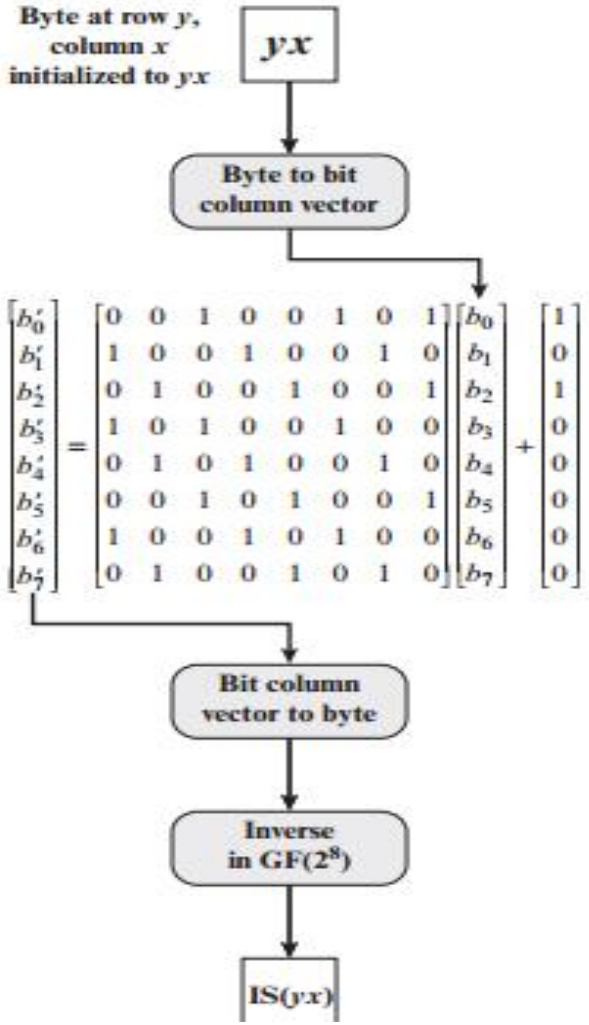
(a) Substitute byte transformation



(b) Add round key transformation

The S-box is constructed in the following fashion (Figure 6.6a).





S-Box Construction

Table 6.2 AES S-Boxes

x \ y	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

(a) S-box

x \ y	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

(b) Inverse S-box

S-Box Construction

1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02}, . . . , {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row y, column x is $\{yx\}$.
2. Map each byte in the S-box to its multiplicative inverse in the finite field GF(2^8); the value {00} is mapped to itself.
3. Consider that each byte in the S-box consists of 8 bits labeled $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$. Apply the following transformation to each bit of each byte in the S-box:

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad \textbf{(6.1)}$$

where c_i is the ith bit of byte c with the value {63}; that is, $(c_7c_6c_5c_4c_3c_2c_1c_0) = (01100011)$. The prime (′) indicates that the variable is to be updated by the value on the right. The AES standard depicts this transformation in matrix form as follows.

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \textbf{(6.2)}$$

Equation (6.2) has to be interpreted carefully. In ordinary matrix multiplication,[4] each element in the product matrix is the sum of products of the elements of one row and one column. In this case, each element in the product matrix is the bitwise XOR of products of elements of one row and one column. Furthermore, the final addition shown in Equation (6.2) is a bitwise XOR. Recall from Section 5.6 that the bitwise XOR is addition in GF(2^8).

As an example, consider the input value {95}. The multiplicative inverse in $GF(2^8)$ is $\{95\}^{-1} = \{8A\}$, which is 10001010 in binary. Using Equation (6.2),

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The result is {2A}, which should appear in row {09} column {05} of the S-box. This is verified by checking Table 6.2a.

The **inverse substitute byte transformation**, called InvSubBytes, makes use of the inverse S-box shown in Table 6.2b. Note, for example, that the input {2A} produces the output {95}, and the input {95} to the S-box produces {2A}. The inverse S-box is constructed (Figure 6.6b) by applying the inverse of the transformation in Equation (6.1) followed by taking the multiplicative inverse in GF(2^8). The inverse transformation is

$$b'_i = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i$$

where byte d = {05}, or 00000101. We can depict this transformation as follows.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

To see that InvSubBytes is the inverse of SubBytes, label the matrices in SubBytes and InvSubBytes as **X** and **Y**, respectively, and the vector versions of constants c and d as **C** and **D**, respectively. For some 8-bit vector **B**, Equation (6.2) becomes $\mathbf{B}' = \mathbf{XB} \oplus \mathbf{C}$. We need to show that $\mathbf{Y}(\mathbf{XB} \oplus \mathbf{C}) \oplus \mathbf{D} = \mathbf{B}$. To multiply out, we must show $\mathbf{YXB} \oplus \mathbf{YC} \oplus \mathbf{D} = \mathbf{B}$. This becomes

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

We have demonstrated that **YX** equals the identity matrix, and the **YC** = **D**, so that **YC** $\oplus$ **D** equals the null vector.

Rationale The S-box is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits and the property that the output is not a linear mathematical function of the input [DAEM01]. The nonlinearity is due to the use of the multiplicative inverse. In addition, the constant in Equation (6.1) was chosen so that the S-box has no fixed points [S-box(a) = a] and no "opposite fixed points" [S-box(a) = $\bar{a}$], where $\bar{a}$ is the bitwise complement of a.

Of course, the S-box must be invertible, that is, IS-box[S-box(a)] = a. However, the S-box does not self-inverse in the sense that it is not true that S-box(a) = IS-box(a). For example, S-box({95}) = {2A}, but IS-box({95}) = {AD}.

Polynomial Arithmetic

With the appropriate definition of arithmetic operations, each such set S is a finite field. The definition consists of the following elements.

1. Arithmetic follows the ordinary rules of polynomial arithmetic using the basic rules of algebra, with the following two refinements.
2. Arithmetic on the coefficients is performed modulo p. That is, we use the rules of arithmetic for the finite field Z_p.
3. If multiplication results in a polynomial of degree greater than $n - 1$, then the polynomial is reduced modulo some irreducible polynomial $m(x)$ of degree n. That is, we divide by $m(x)$ and keep the remainder. For a polynomial $f(x)$, the remainder is expressed as $r(x) = f(x) \bmod m(x)$.

The Advanced Encryption Standard (AES) uses arithmetic in the finite field $GF(2^8)$, with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. Consider the two polynomials $f(x) = x^6 + x^4 + x^2 + x + 1$ and $g(x) = x^7 + x + 1$. Then

$$\begin{aligned} f(x) + g(x) &= x^6 + x^4 + x^2 + x + 1 + x^7 + x + 1 \\ &= x^7 + x^6 + x^4 + x^2 \end{aligned}$$

$$\begin{aligned} f(x) \times g(x) &= x^{13} + x^{11} + x^9 + x^8 + x^7 \\ &\quad + x^7 + x^5 + x^3 + x^2 + x \\ &\quad + x^6 + x^4 + x^2 + x + 1 \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \end{aligned}$$

$$\begin{array}{r} x^5 + x^3 \\ x^8 + x^4 + x^3 + x + 1 \overline{)\, x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1} \\ \underline{x^{13} + x^9 + x^8 + x^6 + x^5} \\ x^{11} + x^4 + x^3 \\ \underline{x^{11} + x^7 + x^6 + x^4 + x^3} \\ x^7 + x^6 + 1 \end{array}$$

Therefore, $f(x) \times g(x) \bmod m(x) = x^7 + x^6 + 1$.

Extended Euclidean Algorithm for Polynomials			
Calculate	**Which satisfies**	**Calculate**	**Which satisfies**
$r_{-1}(x) = a(x)$		$v_{-1}(x) = 1; w_{-1}(x) = 0$	$a(x) = a(x)v_{-1}(x) + bw_{-1}(x)$
$r_0(x) = b(x)$		$v_0(x) = 0; w_0(x) = 1$	$b(x) = a(x)v_0(x) + b(x)w_0(x)$
$r_1(x) = a(x) \bmod b(x)$ $q_1(x) =$ quotient of $a(x)/b(x)$	$a(x) = q_1(x)b(x) + r_1(x)$	$v_1(x) = v_{-1}(x) - q_1(x)v_0(x) = 1$ $w_1(x) = w_{-1}(x) - q_1(x)w_0(x) = -q_1(x)$	$r_1(x) = a(x)v_1(x) + b(x)w_1(x)$
$r_2(x) = b(x) \bmod r_1(x)$ $q_2(x) =$ quotient of $b(x)/r_1(x)$	$b(x) = q_2(x)r_1(x) + r_2(x)$	$v_2(x) = v_0(x) - q_2(x)v_1(x)$ $w_2(x) = w_0(x) - q_2(x)w_1(x)$	$r_2(x) = a(x)v_2(x) + b(x)w_2(x)$
$r_3(x) = r_1(x) \bmod r_2(x)$ $q_3(x) =$ quotient of $r_1(x)/r_2(x)$	$r_1(x) = q_3(x)r_2(x) + r_3(x)$	$v_3(x) = v_1(x) - q_3(x)v_2(x)$ $w_3(x) = w_1(x) - q_3(x)w_2(x)$	$r_3(x) = a(x)v_3(x) + b(x)w_3(x)$
• • •	• • •	• • •	• • •
$r_n(x) = r_{n-2}(x) \bmod r_{n-1}(x)$ $q_n(x) =$ quotient of $r_{n-2}(x)/r_{n-2}(x)$	$r_{n-2}(x) = q_n(x)r_{n-1}(x) + r_n(x)$	$v_n(x) = v_{n-2}(x) - q_n(x)v_{n-1}(x)$ $w_n(x) = w_{n-2}(x) - q_n(x)w_{n-1}(x)$	$r_n(x) = a(x)v_n(x) + b(x)w_n(x)$
$r_{n+1}(x) = r_{n-1}(x) \bmod r_n(x) = 0$ $q_{n+1}(x) =$ quotient of $r_{n-1}(x)/r_n(x)$	$r_{n-1}(x) = q_{n+1}(x)r_n(x) + 0$		$d(x) = \gcd(a(x), b(x)) = r_n(x)$ $v(x) = v_n(x); w(x) = w_n(x)$

Table 5.4 Extended Euclid $[(x^8 + x^4 + x^3 + x + 1), (x^7 + x + 1)]$

Initialization	$a(x) = x^8 + x^4 + x^3 + x + 1; v_{-1}(x) = 1; w_{-1}(x) = 0$ $b(x) = x^7 + x + 1; v_0(x) = 0; w_0(x) = 1$
Iteration 1	$q_1(x) = x; r_1(x) = x^4 + x^3 + x^2 + 1$ $v_1(x) = 1; w_1(x) = x$
Iteration 2	$q_2(x) = x^3 + x^2 + 1; r_2(x) = x$ $v_2(x) = x^3 + x^2 + 1; w_2(x) = x^4 + x^3 + x + 1$
Iteration 3	$q_3(x) = x^3 + x^2 + x; r_3(x) = 1$ $v_3(x) = x^6 + x^2 + x + 1; w_3(x) = x^7$
Iteration 4	$q_4(x) = x; r_4(x) = 0$ $v_4(x) = x^7 + x + 1; w_4(x) = x^8 + x^4 + x^3 + x + 1$
Result	$d(x) = r_3(x) = \gcd(a(x), b(x)) = 1$ $w(x) = w_3(x) = (x^7 + x + 1)^{-1} \bmod (x^8 + x^4 + x^3 + x + 1) = x^7$

Table 5.4 shows the calculation of the multiplicative inverse of $(x^7 + x + 1)$ mod $(x^8 + x^4 + x^3 + x + 1)$. The result is that $(x^7 + x + 1)^{-1} = (x^7)$. That is, $(x^7 + x + 1)(x^7) \equiv 1 (\bmod (x^8 + x^4 + x^3 + x + 1))$.

Diffusion Layer

- It consists of two sublayers as mentioned earlier:
  - (1) **ShiftRows transformation**
  - **(2) MixColumn transformation**

(1) ShiftRows Sublayer Transformation

- If the input to ShiftRows sublayer (Transformation) is given as a state matrix, i.e., B = (B_0,B_1, . . . ,B_{15}) as

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

the output is the new state:

B_0	B_4	B_8	B_{12}	no shift
B_5	B_9	B_{13}	B_1	⟵ one position left shift
B_{10}	B_{14}	B_2	B_6	⟵ two positions left shift
B_{15}	B_3	B_7	B_{11}	⟵ three positions left shift

Forward and Inverse Transformations The **forward shift row transformation**, called ShiftRows, is depicted in Figure 6.7a. The first row of **State** is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of ShiftRows.

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

The **inverse shift row transformation**, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a 1-byte circular right shift for the second row, and so on.

Rationale The shift row transformation is more substantial than it may first appear. This is because the **State**, as well as the cipher input and output, is treated as an array of four 4-byte columns. Thus, on encryption, the first 4 bytes of the plaintext are copied to the first column of **State**, and so on. Furthermore, as will be seen, the round key is applied to **State** column by column. Thus, a row shift moves an individual byte from one column to another, which is a linear distance of a multiple of 4 bytes. Also note that the transformation ensures that the 4 bytes of one column are spread out to four different columns. Figure 6.4 illustrates the effect.

MixColumn Sublayer Transformation

- If a 16-byte state *B* is the input to MixColumn and 16-byte *C* is the output state, then, we can write

$$\text{MixColumn}(B) = C,$$

- Each 4-byte column is considered as a vector and multiplied by a fixed 4×4 matrix. Here, the multiplication and addition are done in GF(2^8) Galois Field (explained later).

Contd…

- As an example, we show how the first four output bytes are computed:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}.$$

Fixed Matrix

- The second column of output bytes (C_4,C_5,C_6,C_7) is computed by multiplying the four input bytes (B_4,B_9,B_{14},B_3) by the same constant matrix, and so on.

MixColumns Transformation

Forward and Inverse Transformations The **forward mix column transformation**, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on **State** (Figure 6.7b):

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad \textbf{(6.3)}$$

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications[5] are

performed in GF(2^8). The MixColumns transformation on a single column of State can be expressed as

$$
\begin{aligned}
s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\
s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \\
s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\
s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j})
\end{aligned}
\tag{6.4}
$$

The following is an example of MixColumns:

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

Let us verify the first column of this example. Recall from Section 5.6 that, in GF(2^8), addition is the bitwise XOR operation and that multiplication can be performed according to the rule established in Equation (4.14). In particular, multiplication of a value by x (i.e., by {02}) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original value (prior to the shift) is 1. Thus, to verify the MixColumns transformation on the first column, we need to show that

$$
\begin{aligned}
(\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\} &= \{47\} \\
\{87\} \oplus (\{02\} \cdot \{6E\}) \oplus (\{03\} \cdot \{46\}) \oplus \{A6\} &= \{37\} \\
\{87\} \oplus \{6E\} \oplus (\{02\} \cdot \{46\}) \oplus (\{03\} \cdot \{A6\}) &= \{94\} \\
(\{03\} \cdot \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \cdot \{A6\}) &= \{ED\}
\end{aligned}
$$

For the first equation, we have {02} · {87} = (0000 1110) ⊕ (0001 1011) = (0001 0101) and {03} · {6E} = {6E} ⊕ ({02} · {6E}) = (0110 1110) ⊕ (1101 1100) = (1011 0010). Then,

$$
\begin{aligned}
\{02\} \cdot \{87\} &= 0001\ 0101 \\
\{03\} \cdot \{6E\} &= 1011\ 0010 \\
\{46\} &= 0100\ 0110 \\
\{A6\} &= \underline{1010\ 0110} \\
&\quad 0100\ 0111 = \{47\}
\end{aligned}
$$

- 87: 1000 0111 X 10 10 X 10 = 100

  1 0000 1110 100

  0000 1110

x^8 + x^4 + X^3 + X + 1: 1 0 0 0 1 1 0 1 0

The **inverse mix column transformation**, called InvMixColumns, is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad \textbf{(6.5)}$$

It is not immediately clear that Equation (6.5) is the **inverse** of Equation (6.3). We need to show

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{0,3} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

which is equivalent to showing

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(6.6)}$$

That is, the inverse transformation matrix times the forward transformation matrix equals the identity matrix. To verify the first column of Equation (6.6), we need to show

$$\begin{aligned} (\{0E\} \cdot \{02\}) \oplus \{0B\} \oplus \{0D\} \oplus (\{09\} \cdot \{03\}) &= \{01\} \\ (\{09\} \cdot \{02\}) \oplus \{0E\} \oplus \{0B\} \oplus (\{0D\} \cdot \{03\}) &= \{00\} \\ (\{0D\} \cdot \{02\}) \oplus \{09\} \oplus \{0E\} \oplus (\{0B\} \cdot \{03\}) &= \{00\} \\ (\{0B\} \cdot \{02\}) \oplus \{0D\} \oplus \{09\} \oplus (\{0E\} \cdot \{03\}) &= \{00\} \end{aligned}$$

For the first equation, we have $\{0E\} \cdot \{02\} = 00011100$ and $\{09\} \cdot \{03\} = \{09\} \oplus (\{09\} \cdot \{02\}) = 00001001 \oplus 00010010 = 00011011$. Then

$$\begin{aligned} \{0E\} \cdot \{02\} &= 00011100 \\ \{0B\} &= 00001011 \\ \{0D\} &= 00001101 \\ \{09\} \cdot \{03\} &= \underline{00011011} \\ & \quad 00000001 \end{aligned}$$

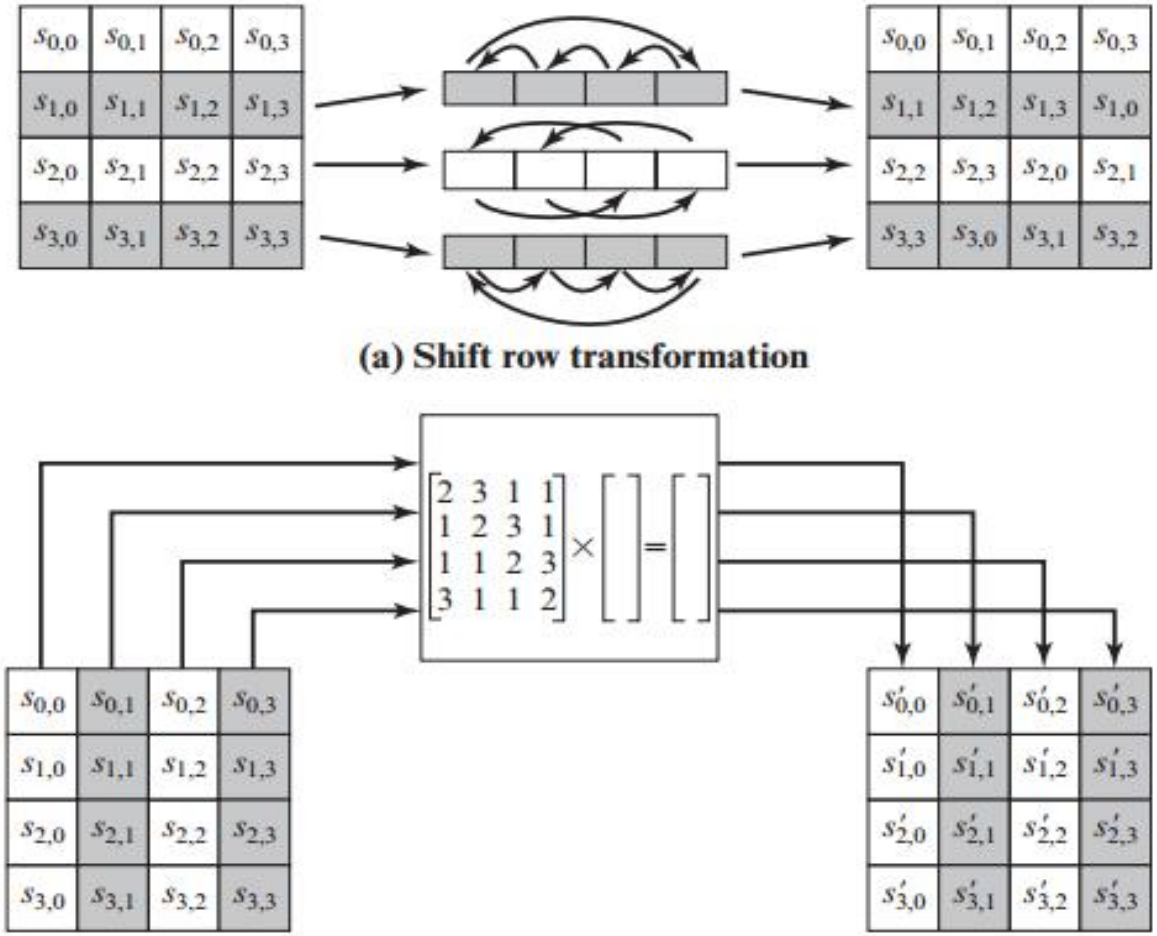


Figure 6.7 AES Row and Column Operations

Key Addition Layer

- The two inputs to the Key Addition layer are the current 16-byte state matrix and a subkey which also consists of 16 bytes (128 bits).
- The two inputs are combined through a bitwise XOR operation.
- Note that the XOR operation is equal to addition in the Galois field GF(2).

AddRoundKey Transformation

Forward and Inverse Transformations In the **forward add round key transformation**, called AddRoundKey, the 128 bits of **State** are bitwise XORed with the 128 bits of the round key. As shown in Figure 6.5b, the operation is viewed as a columnwise operation between the 4 bytes of a **State** column and one word of the round key; it can also be viewed as a byte-level operation. The following is an example of AddRoundKey:

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

$\oplus$

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

=

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D6

The first matrix is **State**, and the second matrix is the round key.

The **inverse add round key transformation** is identical to the forward add round key transformation, because the XOR operation is its own inverse.

Rationale The add round key transformation is as simple as possible and affects every bit of **State**. The complexity of the round key expansion, plus the complexity of the other stages of AES, ensure security.

Figure 6.8 is another view of a single round of AES, emphasizing the mechanisms and inputs of each transformation.

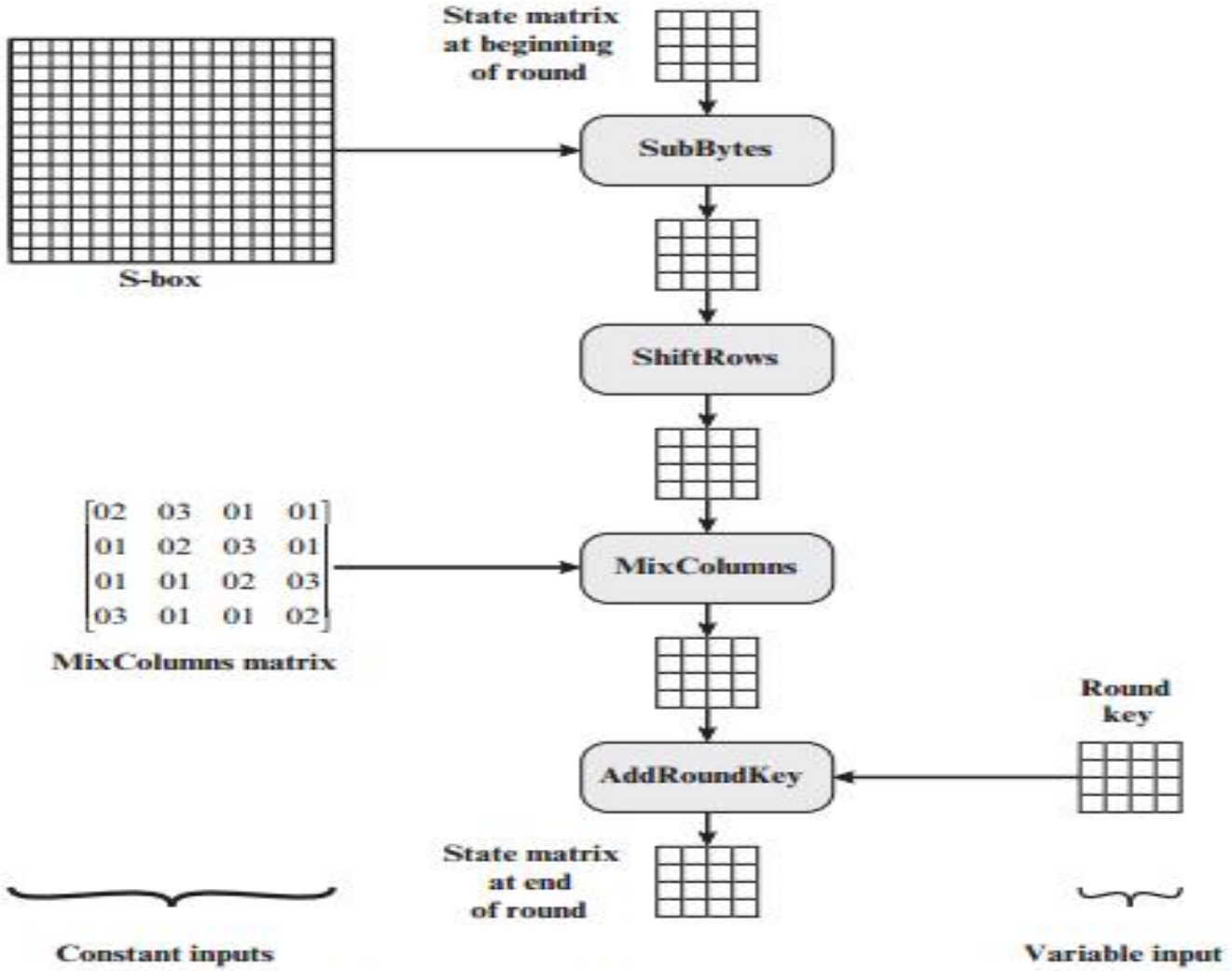
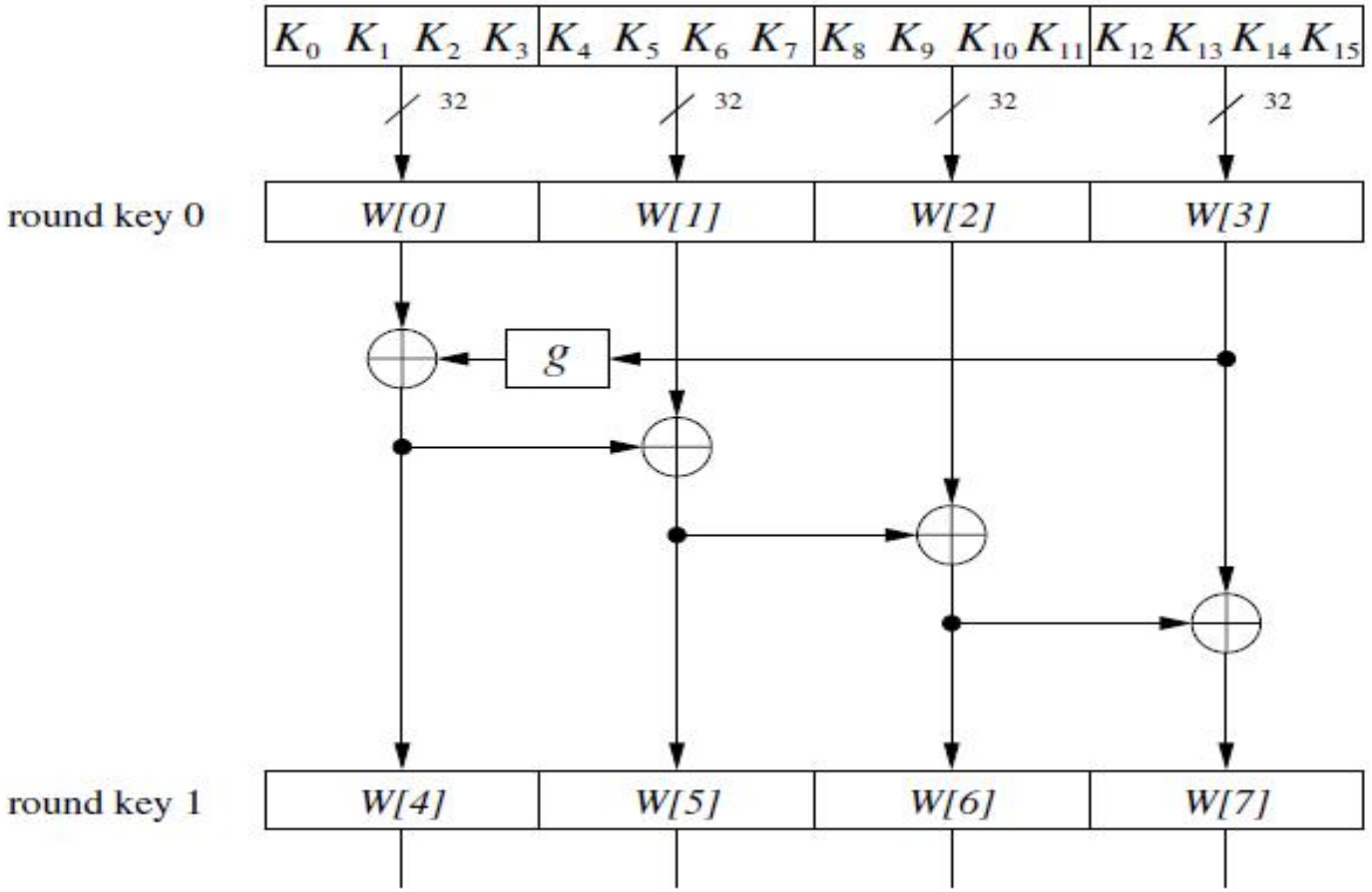


Figure 6.8 Inputs for Single AES Round

Key Schedule

- The key schedule takes the original input key (of length 128, 192 or 256 bit) and derives the subkeys used in AES.
- Note that an XOR addition of a subkey is used both at the input and output of AES.
- This process is sometimes referred to as key **whitening**.
- The number of subkeys is equal to the number of rounds plus one.

AES key schedule Algorithm for 128-bit key size (for 10 rounds AES)



Contd…

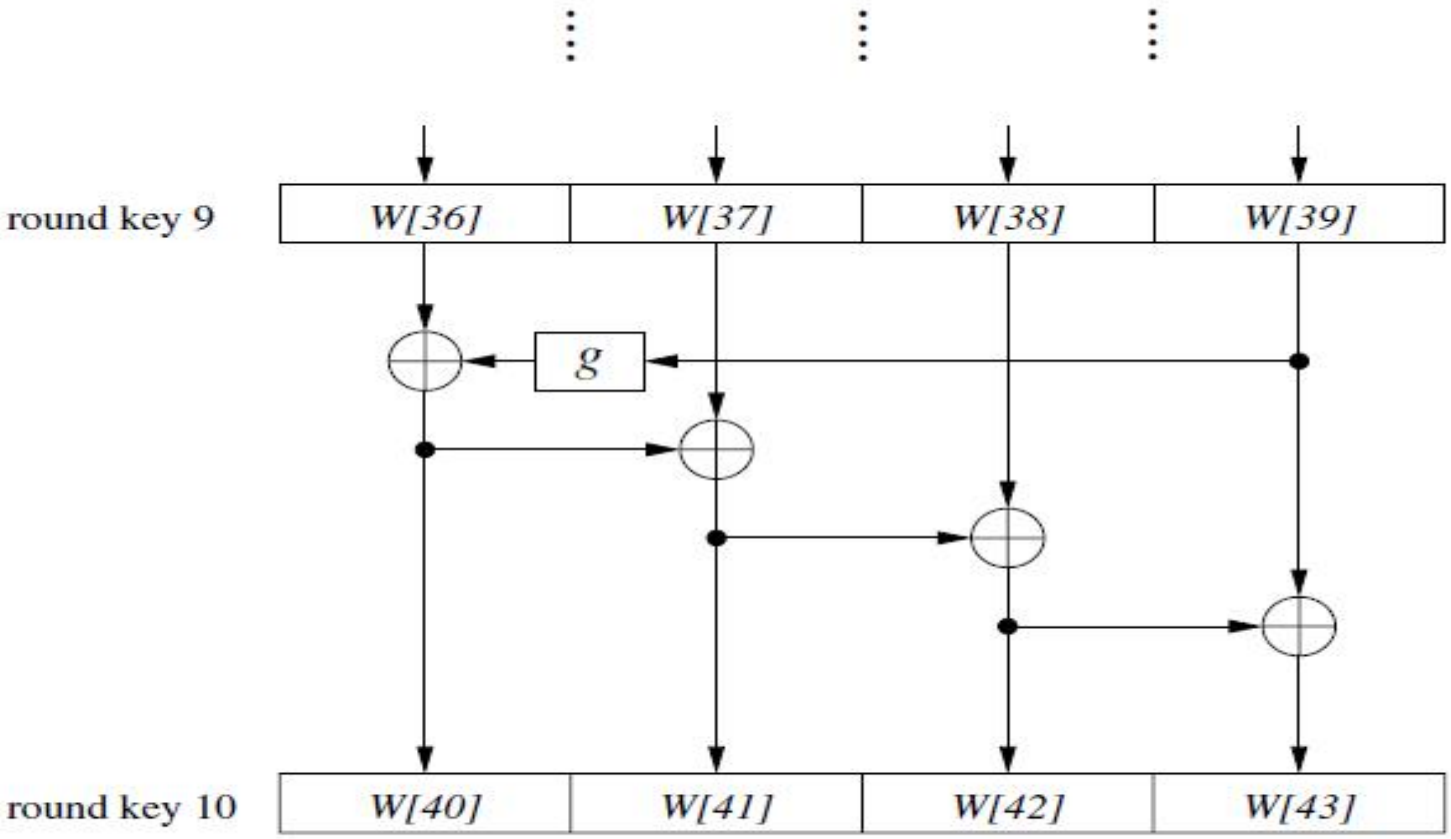


Fig. 4.5 AES key schedule for 128-bit key size

AES Key scheduling or expansion Algorithm

```
KeyExpansion (byte key[16], w[44])
{  w temp;
   for(i=0; i<4; i++)
      w[i] = (key[4*i, key[4*i+1], key[4*i+2], key[4*i+3]);
  for(i=4; i<44; i++) {
      temp= w(i-1);
      if(i mod 4 = 0)
         temp = SubByte(Rotw(temp)) xor Rcon;
      w[i]= w[i-4] xor temp;
  }
}
```

Rotw (b0, b1, b2, b3) = b1, b2, b3, b0)- left circular shift

Round constants (Rcon]

80×2 = 1000 0000×2= (1.x^7+ 0.x^6+ 0. x^5+…)× 10
= x^7× x) = x^8
X^8 mod x^8 +x^4 +x^3 +x +1 = x^4 +x^3 +x +1 = 0001 1011 =1B

1	2	3	4	5	6	7	8	9	10
01	02	04	08	10	20	40	80	1B	36

This first byte, and three rightmost bytes are zero.

That is, Rcon 00 00 00

For example, suppose that the round key for round 8 is

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

Then the first 4 bytes (first column) of the round key for round 9 are calculated as follows:

i (decimal)	temp	After RotWord	After SubWord	Rcon (9)	After XOR with Rcon	w[i − 4]	w[i] = temp ⊕ w[i − 4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

Rationale

The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the ways in which round keys are generated in different rounds. The specific criteria that were used are [DAEM99]

- Knowledge of a part of the cipher key or round key does not enable calculation of many other round-key bits.
- An invertible transformation [i.e., knowledge of any Nk consecutive words of the expanded key enables regeneration of the entire expanded key (Nk = key size in words)].
- Speed on a wide range of processors.
- Usage of round constants to eliminate symmetries.
- Diffusion of cipher key differences into the round keys; that is, each key bit affects many round key bits.
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only.
- Simplicity of description.

Contd…

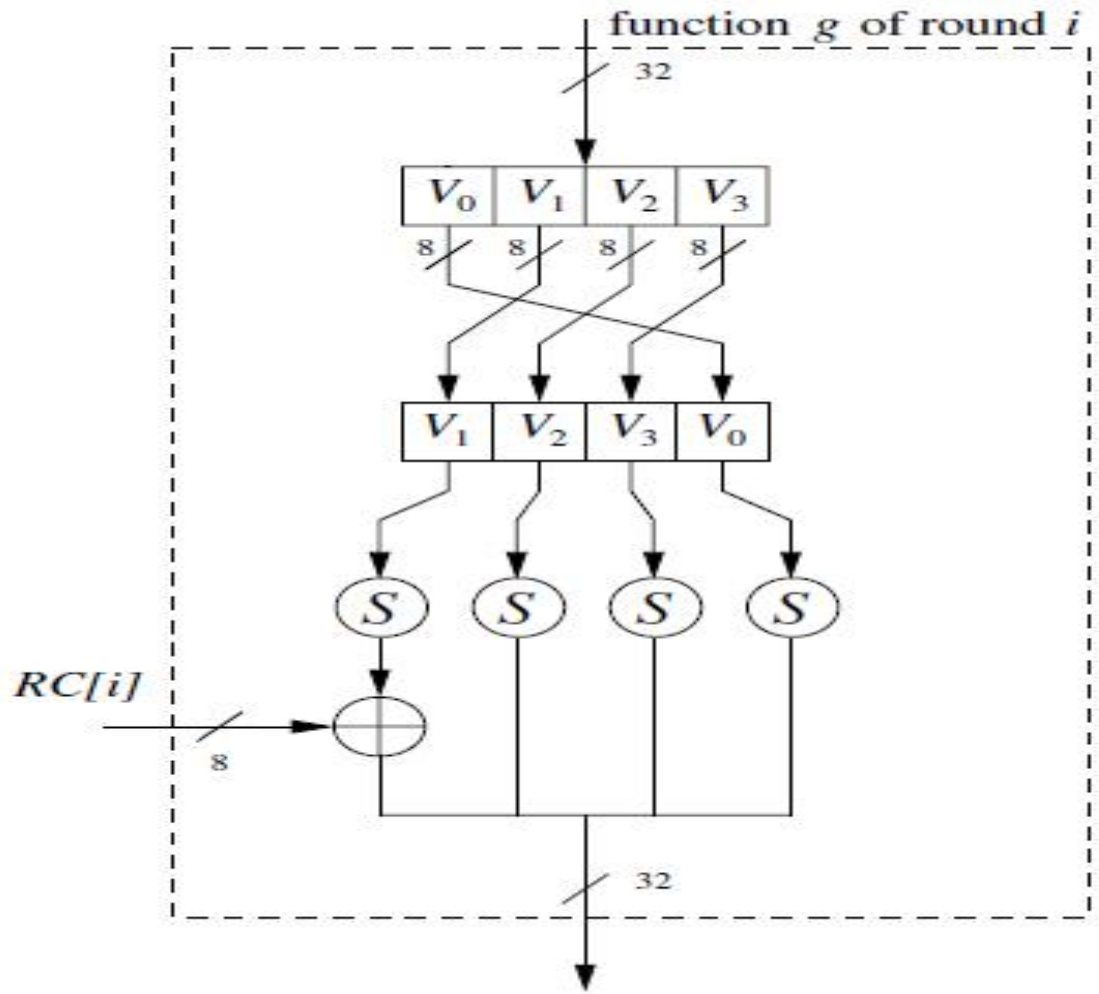
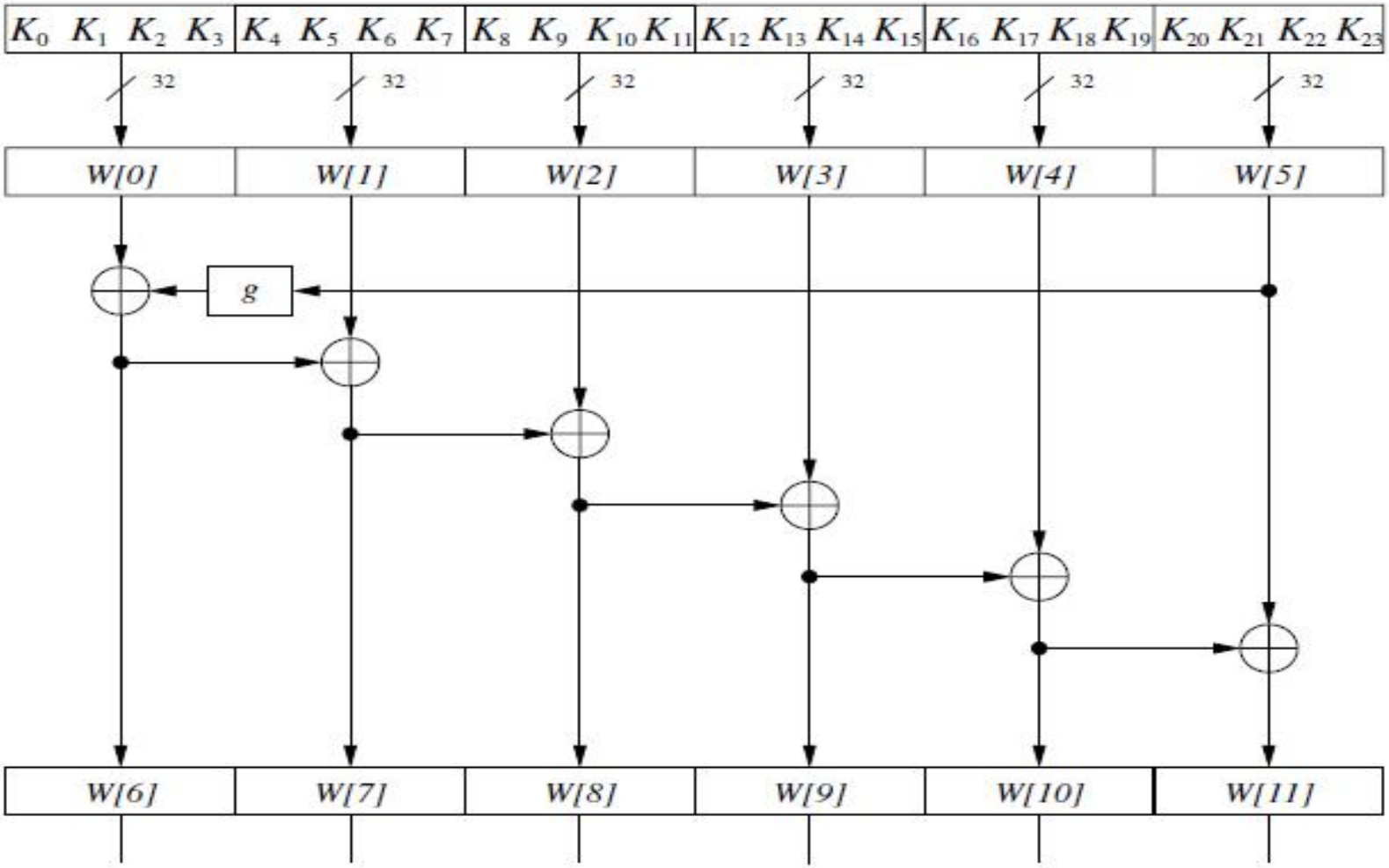


Figure of AES key schedule for 192-bit key size



Contd…

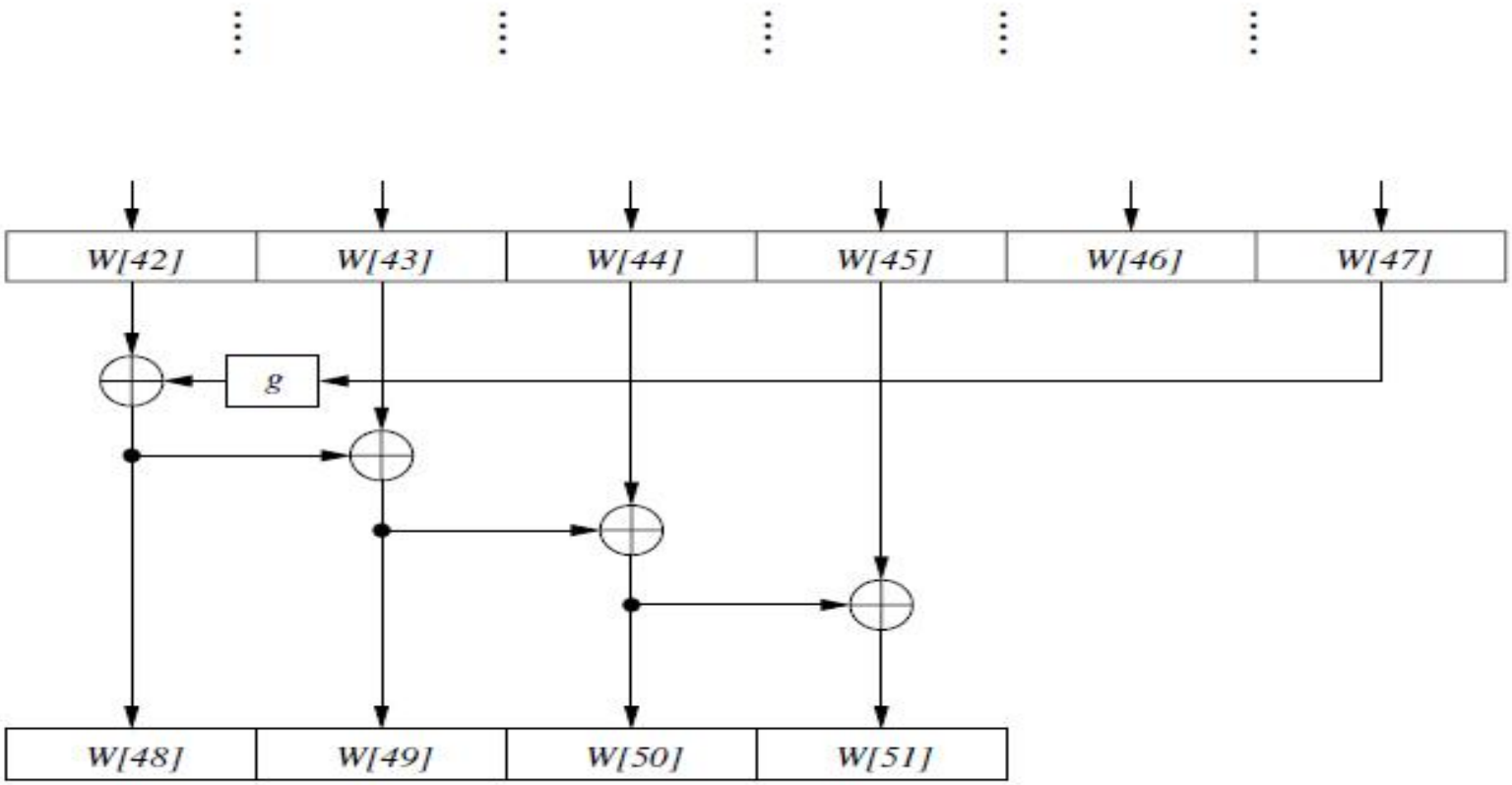


Fig. 4.6 AES key schedule for 192-bit key sizes

Contd…

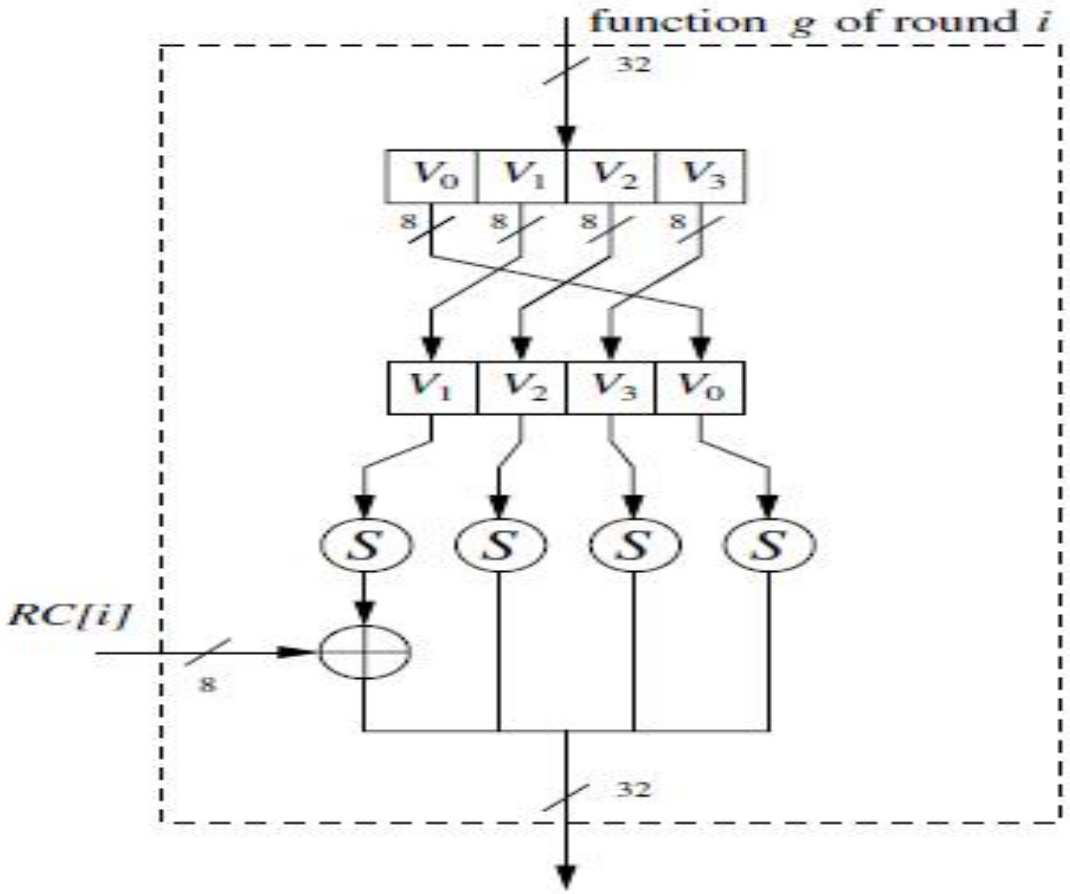
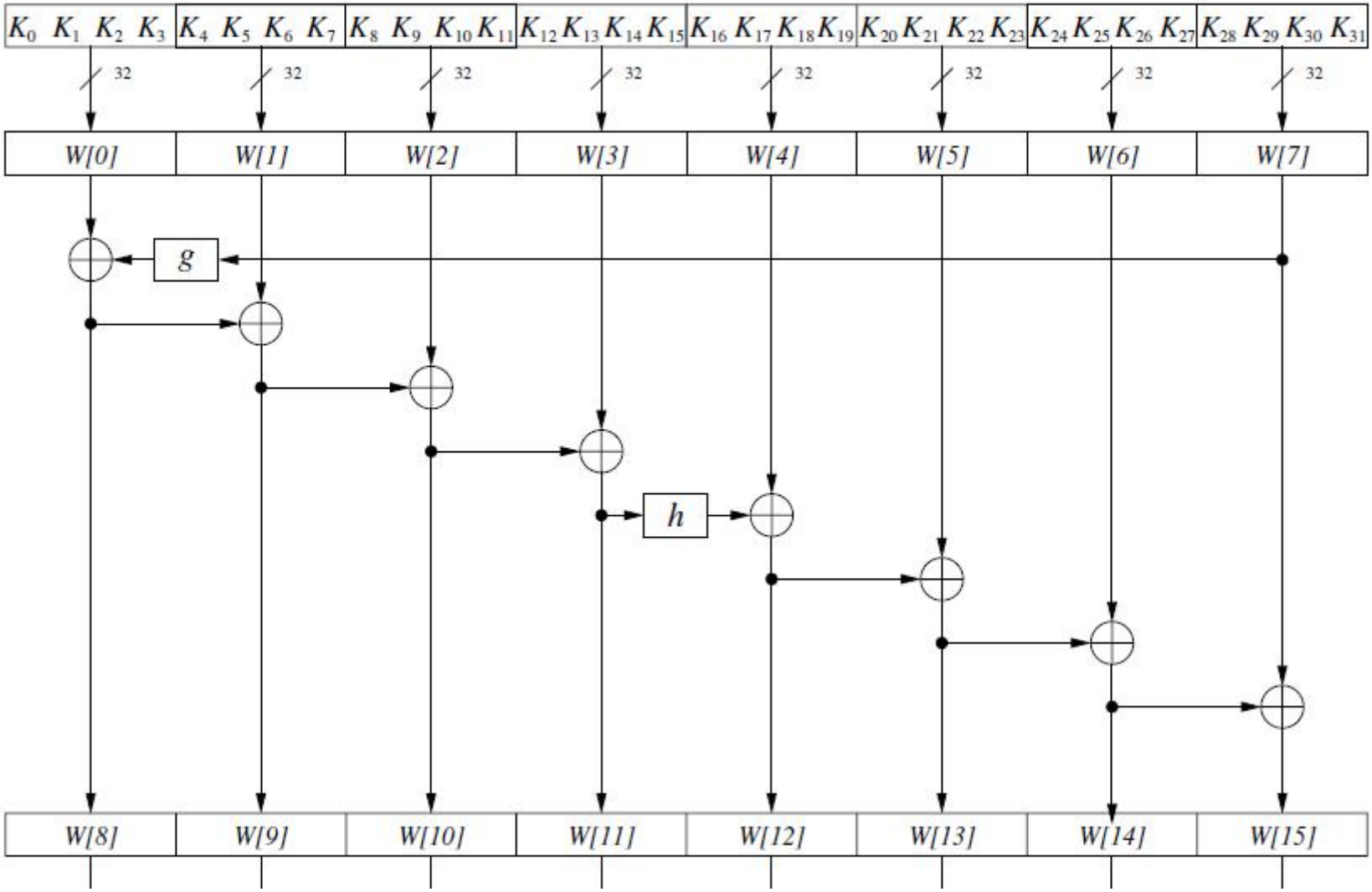


Figure of AES key schedule for 256-bit key size



Contd…

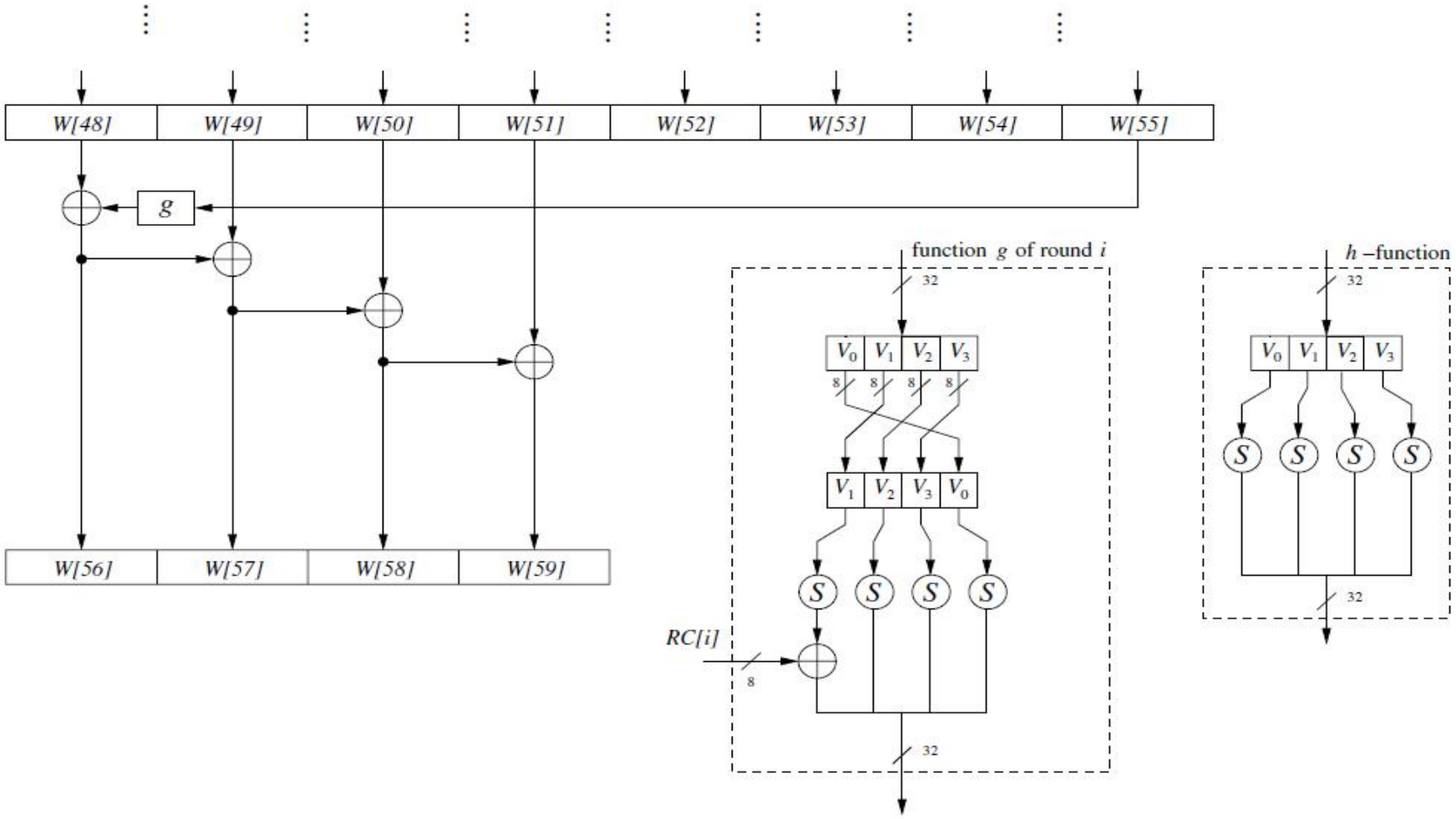
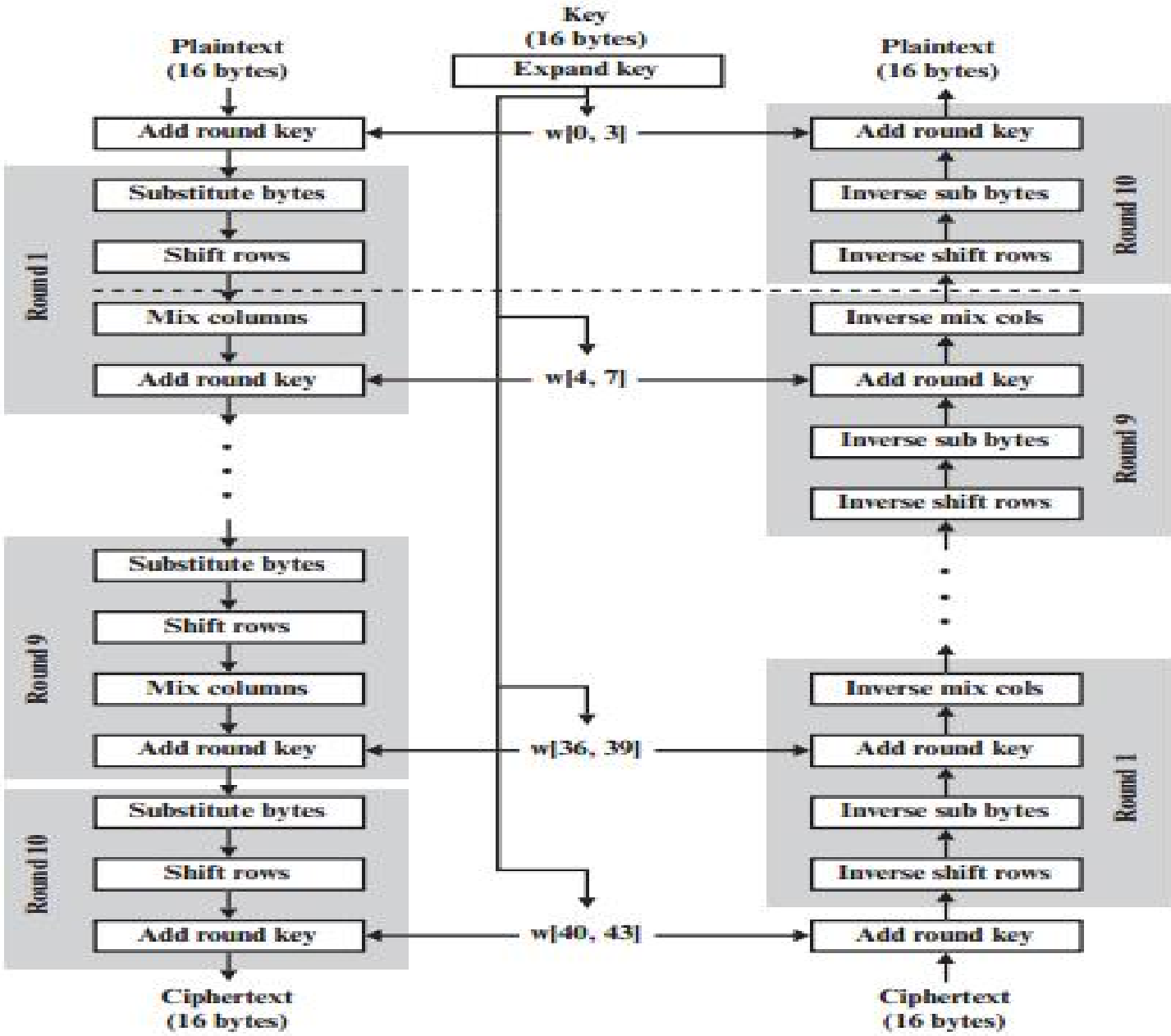


Fig. 4.7 AES key schedule for 256-bit key size

AES Decryption

- AES is not based on a Feistel network.
- The Byte Substitution layer becomes the Inv Byte Substitution layer.
- The ShiftRows layer becomes the Inv ShiftRows layer, and the MixColumn layer becomes Inv MixColumn layer.
- The order of the subkeys is reversed


Plaintext
(16 bytes)
Key
(16 bytes)
Expand key
Plaintext
(16 bytes)
Add round key
w[0, 3]
Add round key
Substitute bytes
Shift rows
Inverse sub bytes
Inverse shift rows
Round 1
Round 10
Mix columns
Add round key
w[4, 7]
Inverse mix cols
Add round key
Inverse sub bytes
Inverse shift rows
Round 9
Substitute bytes
Shift rows
Mix columns
Add round key
w[36, 39]
Inverse mix cols
Add round key
Inverse sub bytes
Inverse shift rows
Round 9
Round 1
Substitute bytes
Shift rows
Add round key
w[40, 43]
Add round key
Round 10
Ciphertext
(16 bytes)
Ciphertext
(16 bytes)

Equivalent Inverse Cipher for AES

- As was mentioned, the AES decryption cipher is not identical to the encryption cipher (Figure 6.3). That is, the sequence of transformations for decryption differs from that for encryption, although the form of the key schedules for encryption and decryption is the same.
- This has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption.
- There is, however, an equivalent version of the decryption algorithm that has the same structure as the encryption algorithm. The equivalent version has the same sequence of transformations as the encryption algorithm (with transformations replaced by their inverses).
- To achieve this equivalence, a change in key schedule is needed.

- Two separate changes are needed to bring the decryption structure in line with the encryption structure. As illustrated in Figure 6.3, an encryption round has the structure SubBytes, ShiftRows, MixColumns, AddRoundKey.
- The standard decryption round has the structure InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns.
- Thus, the first two stages of the decryption round need to be interchanged, and the second two stages of the decryption round need to be interchanged.

Interchanging InvShiftRows and InvSubBytes InvShiftRows affects the sequence of bytes in **State** but does not alter byte contents and does not depend on byte contents to perform its transformation. InvSubBytes affects the contents of bytes in **State** but does not alter byte sequence and does not depend on byte sequence to perform its transformation. Thus, these two operations commute and can be interchanged. For a given **State** S_i,

$$\text{InvShiftRows}\,[\text{InvSubBytes}\,(S_i)] = \text{InvSubBytes}\,[\text{InvShiftRows}\,(S_i)]$$

Interchanging AddRoundKey and InvMixColumns The transformations AddRoundKey and InvMixColumns do not alter the sequence of bytes in **State**. If we view the key as a sequence of words, then both AddRoundKey and InvMixColumns operate on **State** one column at a time. These two operations are linear with respect to the column input. That is, for a given **State** S_i and a given round key w_j,

$$\text{InvMixColumns}\,(S_i \oplus w_j) = [\text{InvMixColumns}\,(S_i)] \oplus [\text{InvMixColumns}\,(w_j)]$$

To see this, suppose that the first column of **State** S_i is the sequence (y_0, y_1, y_2, y_3) and the first column of the round key w_j is (k_0, k_1, k_2, k_3). Then we need to show

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \oplus k_0 \\ y_1 \oplus k_1 \\ y_2 \oplus k_2 \\ y_3 \oplus k_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \oplus \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

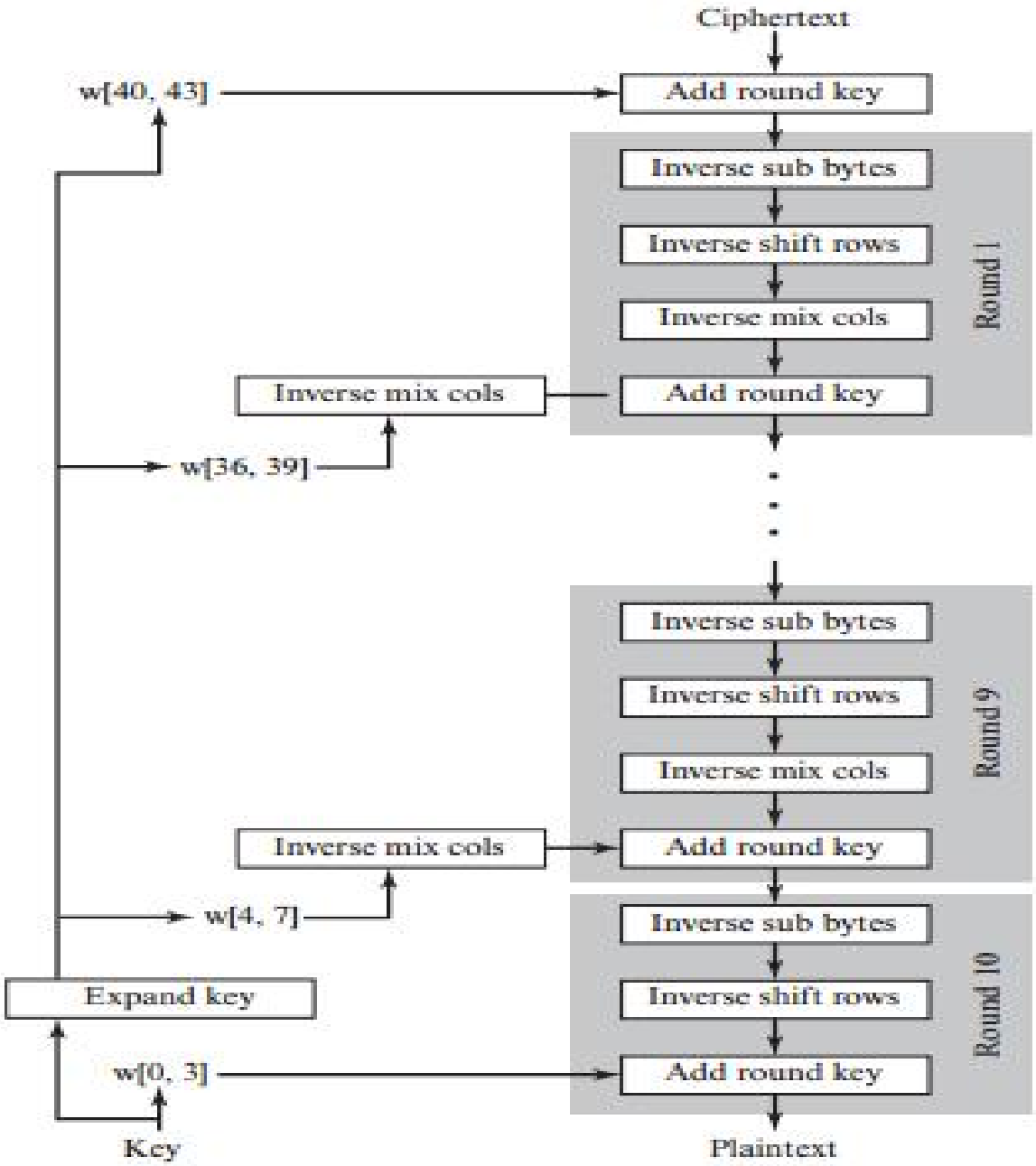
To see this, suppose that the first column of **State** S_i is the sequence (y_0, y_1, y_2, y_3) and the first column of the round key w_j is (k_0, k_1, k_2, k_3). Then we need to show

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \oplus k_0 \\ y_1 \oplus k_1 \\ y_2 \oplus k_2 \\ y_3 \oplus k_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \oplus \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

Let us demonstrate that for the first column entry. We need to show

$$\begin{aligned} &[\{0E\} \cdot (y_0 \oplus k_0)] \oplus [\{0B\} \cdot (y_1 \oplus k_1)] \oplus [\{0D\} \cdot (y_2 \oplus k_2)] \oplus [\{09\} \cdot (y_3 \oplus k_3)] \\ &= [\{0E\} \cdot y_0] \oplus [\{0B\} \cdot y_1] \oplus [\{0D\} \cdot y_2] \oplus [\{09\} \cdot y_3] \oplus \\ &\quad [\{0E\} \cdot k_0] \oplus [\{0B\} \cdot k_1] \oplus [\{0D\} \cdot k_2] \oplus [\{09\} \cdot k_3] \end{aligned}$$

This equation is valid by inspection. Thus, we can interchange AddRoundKey and InvMixColumns, provided that we first apply InvMixColumns to the round key. Note that we do not need to apply InvMixColumns to the round key for the input to the first AddRoundKey transformation (preceding the first round) nor to the last AddRoundKey transformation (in round 10). This is because these two AddRoundKey transformations are not interchanged with InvMixColumns to produce the equivalent decryption algorithm.


Ciphertext
w[40, 43]
Add round key
Inverse sub bytes
Inverse shift rows
Inverse mix cols
Add round key
Round 1
Inverse mix cols
w[36, 39]
Inverse sub bytes
Inverse shift rows
Inverse mix cols
Add round key
Round 9
Inverse mix cols
w[4, 7]
Inverse sub bytes
Inverse shift rows
Add round key
Round 10
Expand key
w[0, 3]
Key
Plaintext

- AES was designed after DES
- Most of the known attacks on DES were already tested on AES.
- Brute-Force attack is huge time consuming (2^128)
- There are no differential and linear attacks on AES yet.
- It can be easily implemented using cheap processors.

Thank You