Advanced Encryption Standard (AES) – Transformation functions

Dr. Shubhangi Sapkal

AES Transformation Functions

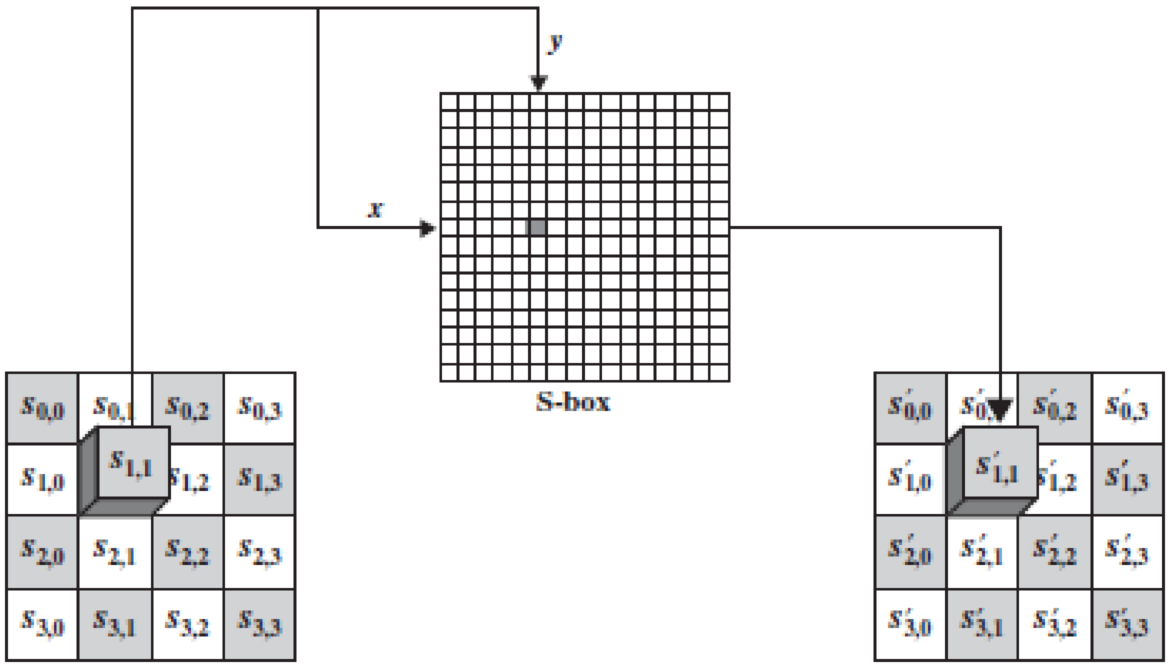
Four different transformations are used, one of permutation and three of substitution:

- **Substitute bytes:** Uses an S-box to perform a byte-by-byte substitution of the block
- **ShiftRows:** A simple permutation
- **MixColumns:** A substitution that makes use of arithmetic over GF(2^8)
- **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded key

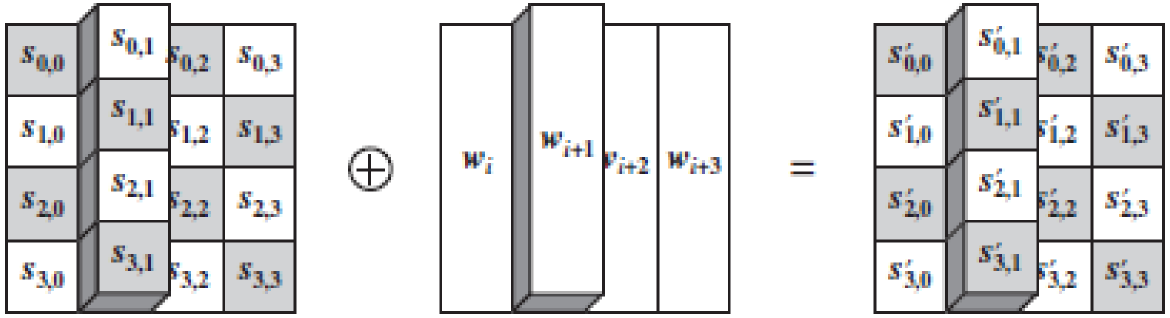
Substitute Bytes Transformation

Forward and Inverse Transformations

- The **forward substitute byte transformation**, called SubBytes, is a simple table lookup. AES defines a 16 * 16 matrix of byte values, called an S-box, that contains a permutation of all possible 256 8-bit values.
- Each individual byte of **State** is mapped into a new byte in the following way:
  The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value.
- These row and column values serve as indexes into the S-box to select a unique 8-bit output value.
- For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.



(a) Substitute byte transformation



(b) Add round key transformation

Figure 5: AES Byte level operations

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

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

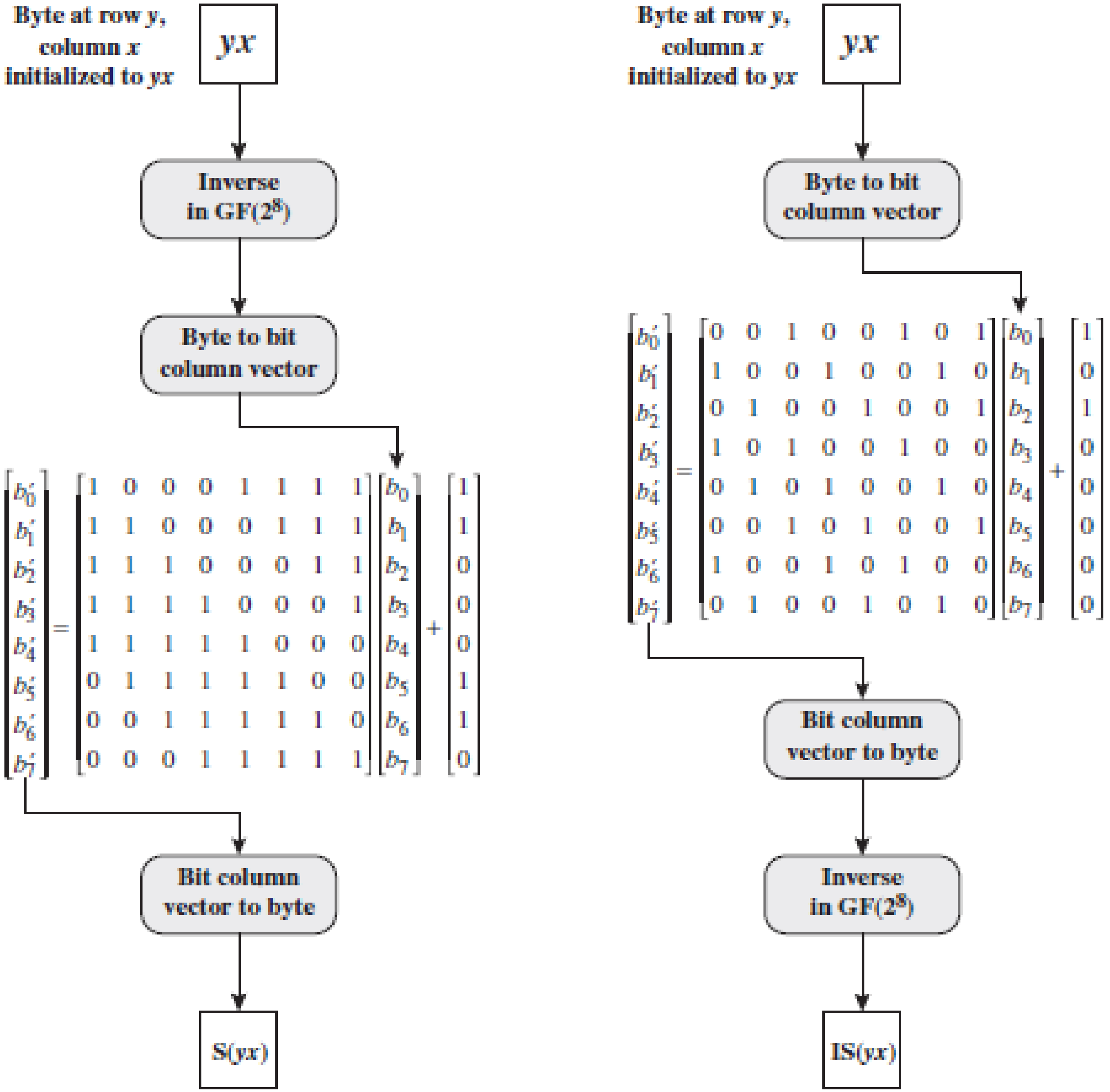
Figure 6: AES S-boxes

Example of SubBytes transformatiom

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

→

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



(a) Calculation of byte at row y, column x of S-box

(a) Calculation of byte at row y, column x of IS-box

Figure 6: Construction of S-box and I-box

Substitute Bytes Transformation

- 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*}.
- 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.

Substitute Bytes Transformation

- Consider that each byte in the S-box consists of 8 bits labeled ($b7$, $b6$, $b5$, $b4$, $b3$, $b2$, $b1$, $b0$). Apply the following transformation to each bit of each byte in the S-box:

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

- where ci is the ith bit of byte c with the value {63}; that is, ($c7c6c5c4c3c2c1c0$) = (01100011).

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

- In ordinary matrix multiplication, for 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 is a bitwise XOR. 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, The result is {2A}, which should appear in row {09} column {05} of the S-box.
- This is verified by checking Table
- The **inverse substitute byte transformation**, called InvSubBytes, makes use of the inverse S-box shown in Table 5.2b.
- for example, the input {2A} produces the output {95}, and the input {95} to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse of the transformation in Equation 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.

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

- 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 (5.2) becomes **B** = **XB C**. We need to show that **Y**(**XB C**) **D** = **B.**
- To multiply out, we must show **YXB YC D** = **B.**

ShiftRows Transformation

Forward and Inverse Transformations

- The **forward shift row transformation**, called ShiftRows.
- 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.

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.

MixColumns Transformation

Forward and Inverse Transformations

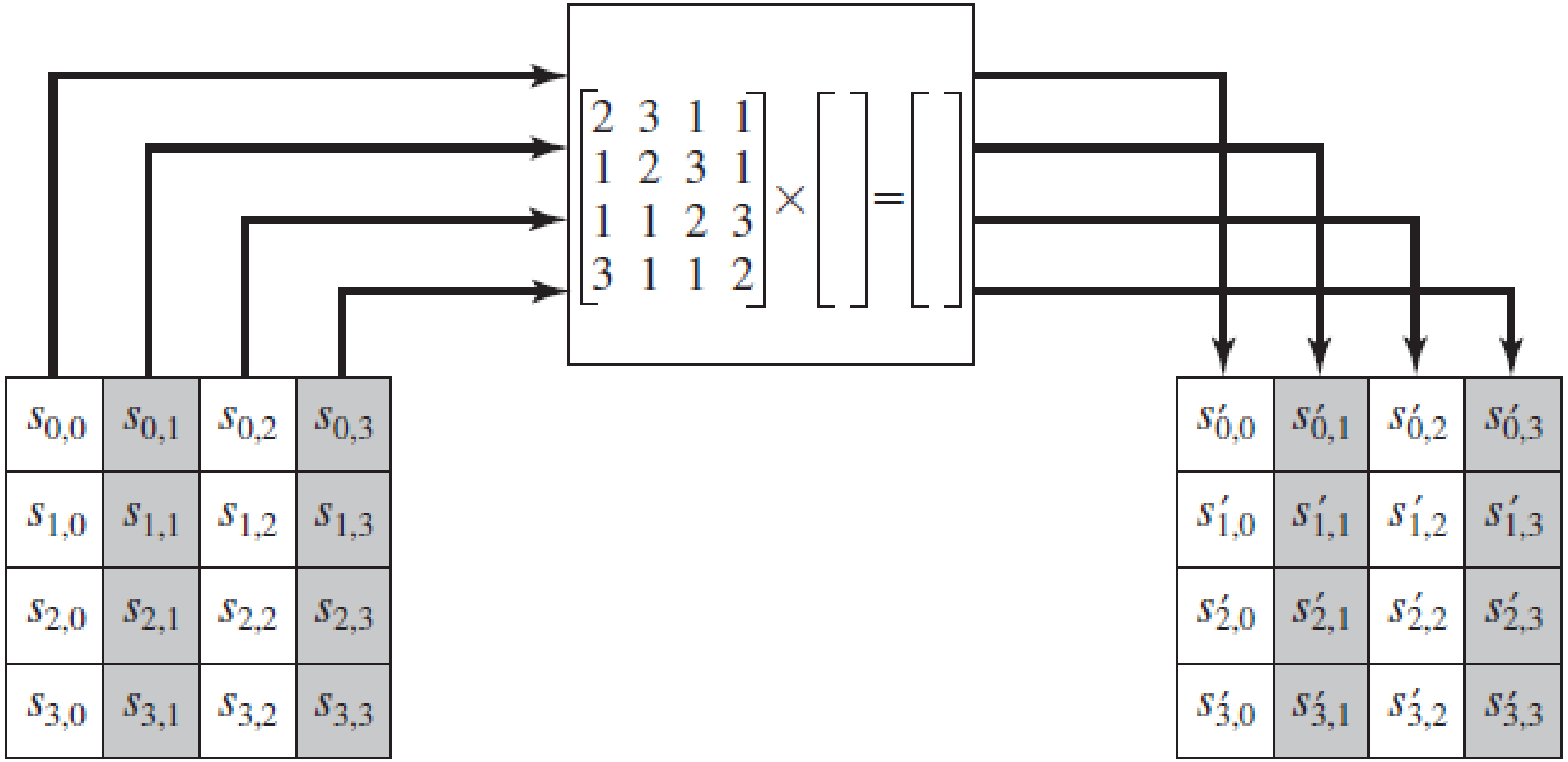
- The **forward mix column transformation**, called MixColumns, operates on each column individually.
- Each byte of a column is mapped into a new value that is a function of all four bytes in that column.

MixColumns Transformation

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

MixColumns Transformation

- 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 are performed in $GF(2^8)$.



(b) Mix column transformation

$$s'_{0,j} = (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$

$$s'_{1,j} = s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j} \oplus s_{3,j}$$

$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j})$$

$$s'_{3,j} = (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j})$$

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

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

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

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

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

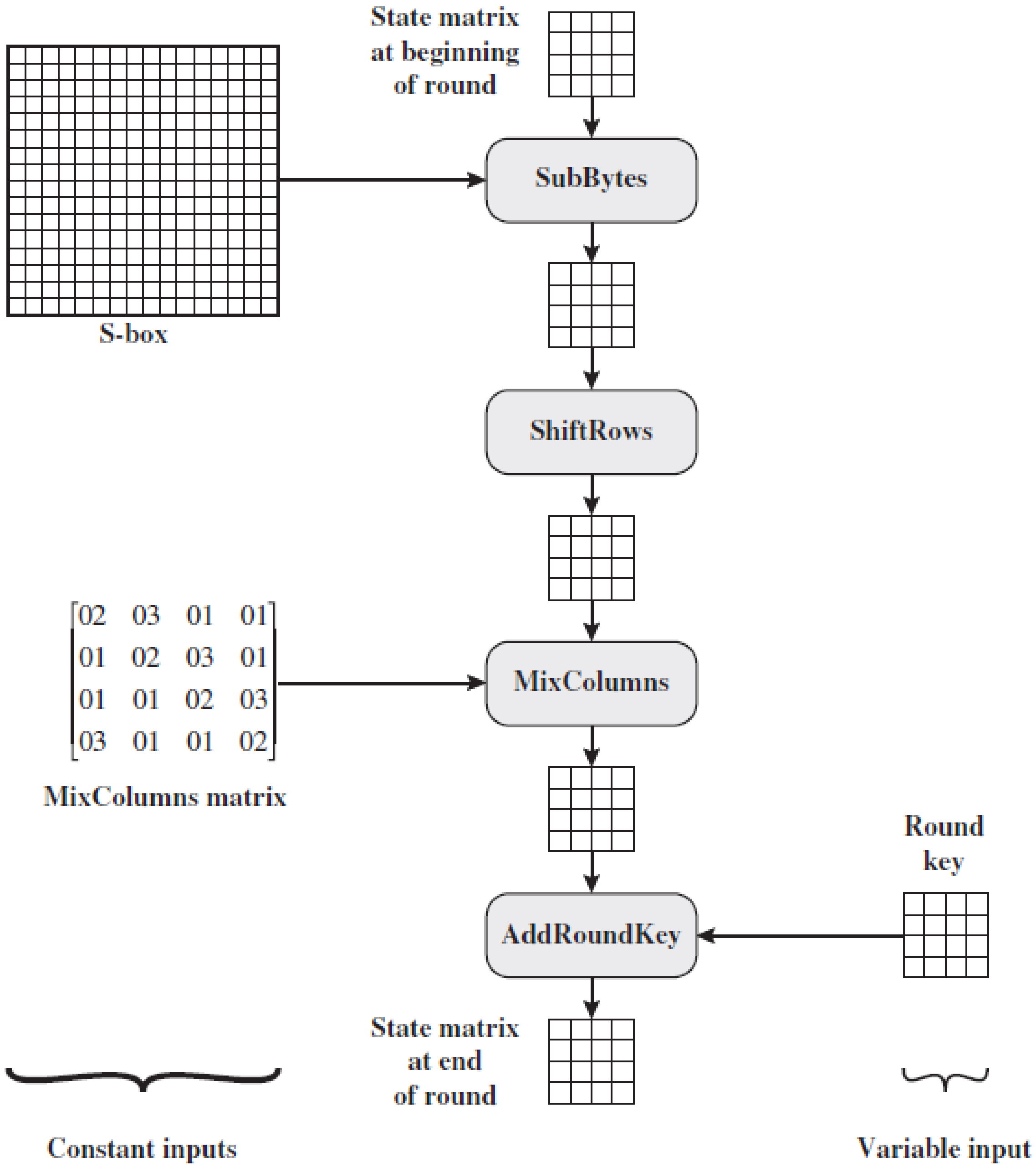
The **inverse mix column transformation**, called InvMixColumns, is defined by the following matrix multiplication:

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

AddRoundKey Transformation

- 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.
- The operation is viewed as a column wise 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.

$$
\begin{array}{|c|c|c|c|}
\hline
47 & 40 & A3 & 4C \\
\hline
37 & D4 & 70 & 9F \\
\hline
94 & E4 & 3A & 42 \\
\hline
ED & A5 & A6 & BC \\
\hline
\end{array}
\oplus
\begin{array}{|c|c|c|c|}
\hline
AC & 19 & 28 & 57 \\
\hline
77 & FA & D1 & 5C \\
\hline
66 & DC & 29 & 00 \\
\hline
F3 & 21 & 41 & 6A \\
\hline
\end{array}
=
\begin{array}{|c|c|c|c|}
\hline
EB & 59 & 8B & 1B \\
\hline
40 & 2E & A1 & C3 \\
\hline
F2 & 38 & 13 & 42 \\
\hline
1E & 84 & E7 & D6 \\
\hline
\end{array}
$$

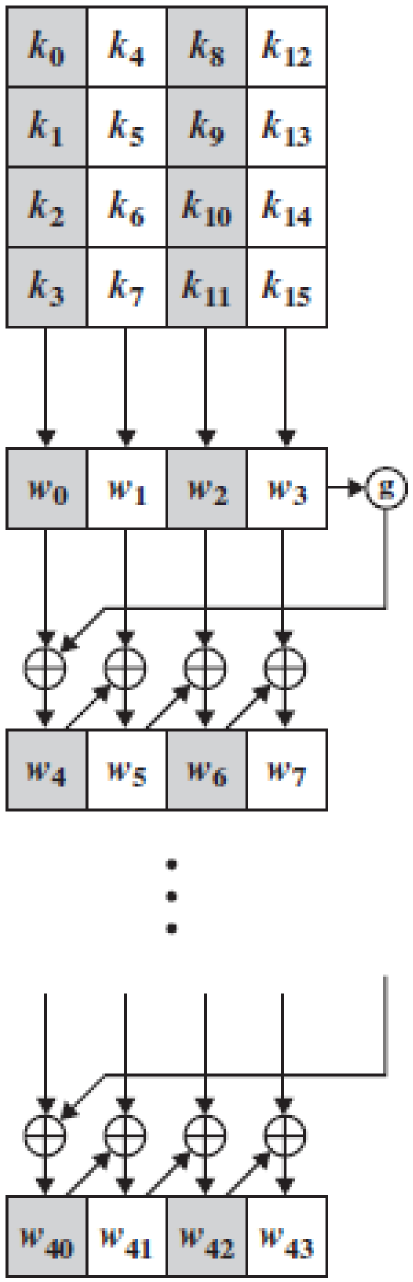

State matrix
at beginning
of round
S-box
SubBytes
ShiftRows
[02 03 01 01]
[01 02 03 01]
[01 01 02 03]
[03 01 01 02]
MixColumns matrix
MixColumns
Round
key
AddRoundKey
State matrix
at end
of round
Constant inputs
Variable input

AES Key Expansion

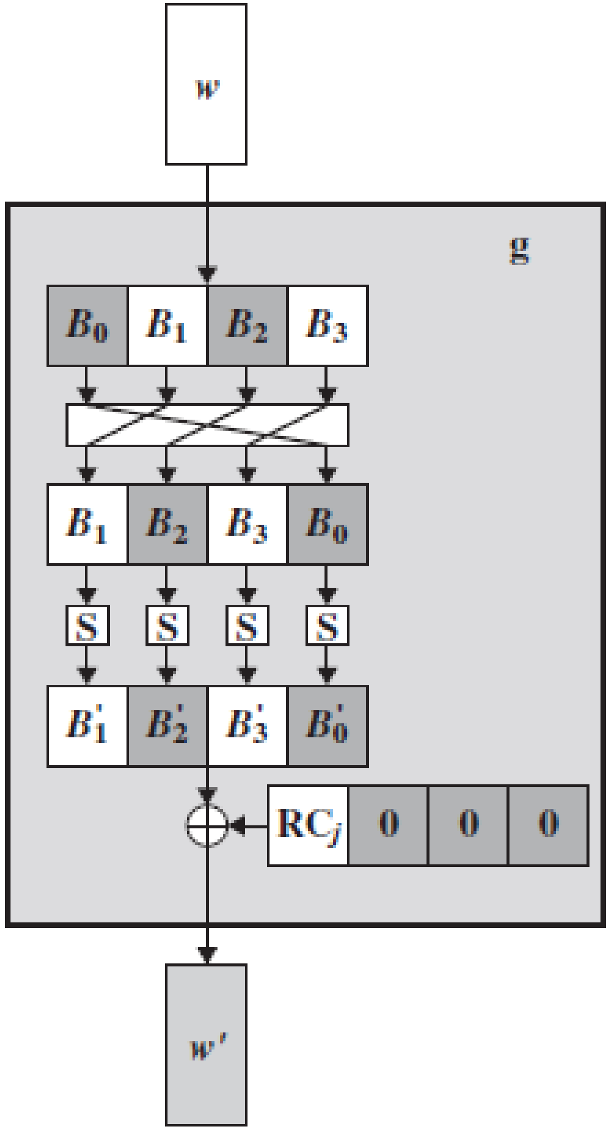
Key Expansion Algorithm

```
KeyExpansion (byte key[16], word w[44])
{
    word 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 = SubWord (RotWord (temp))
                                ⊕ Rcon[i/4];
      w[i] = w[i-4] ⊕ temp
     }
}
```



(a) Overall algorithm



(b) Function g

Figure: AES Key Expansion

- The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes).
- The key is copied into the first four words of the expanded key.
- The remainder of the expanded key is filled in four words at a time. Each added word **w**[i] depends on the immediately preceding word, **w**[i - 1], and the word four positions back, **w**[i - 4].
- In three out of four cases, a simple XOR is used.
- For a word whose position in the **w** array is a multiple of 4, a more complex function is used.
- The function g consists of the following subfunctions.

1. RotWord performs a one-byte circular left shift on a word. This means that an input word [B0, B1, B2, B3] is transformed into [B1, B2, B3, B0].

2. SubWord performs a byte substitution on each byte of its input word, using the S-box.

3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

- The round constant is a word in which the three rightmost bytes are always 0.
- Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as Rcon[j] = (RC[j], 0, 0, 0), with RC[1] = 1, RC[j] = 2 # RC[j-1] and with multiplication defined over the field GF(2^8). The values of RC[j] in hexadecimal are

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

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	$w[i-4]$	$w[i] = \text{temp} \oplus w[i-4]$
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3