AES CIPHER
The Inverse Substitute Byte Transformation

Inverse substitute byte transformation, called InvSubBytes, makes use of the inverse of S-box shown in Table 5.4b. 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 by applying the inverse of the transformation in (5.1) 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 d={05}, or 0000 0101. We can depict this transformation as follows:

B_0'		0	0	1	0	0	1	0	1		B_0		1
B_1'		1	0	0	1	0	0	1	0		B_1		0
B_2'		0	1	0	0	1	0	0	1		B_2		1
B_3'	=	1	0	1	0	0	1	0	0	x	B_3	+	0
B_4'		0	1	0	1	0	0	1	0		B_4		0
B_5'		0	0	1	0	1	0	0	1		B_5		0
B_6'		1	0	0	1	0	1	0	0		B_6		0
B_7'		0	1	0	0	1	0	1	0		B_7		0

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

$$B' = XB \oplus C \quad (*)$$

Assume that

$Y = X^{-1}$: YX=E

where E is the unity matrix.

Multiplying both parts of (*) by Y, we have

$$YB' = YXB \oplus YC = B + YC$$

$$B = YB' \oplus YC = YB' \oplus D$$

Let's check partially that Y is the inverse of X: diagonal elements of product should be 1's, other elements – 0's. For example, let's calculate 2nd diagonal element (multiply 2nd row of Y by 2nd column of X, start numbering from 0):

0*0+1*0+0*1+0*1+1*1+0*1+0*1+1*0=1

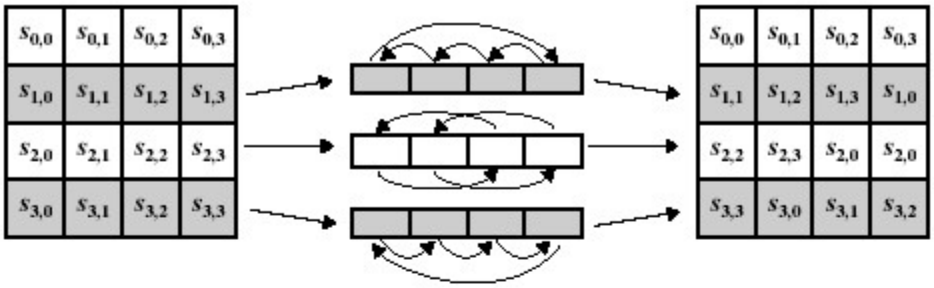
If we multiply 2nd row of Y by 1st column of X, then

The Inverse Substitute Byte Transformation (Cont 1)

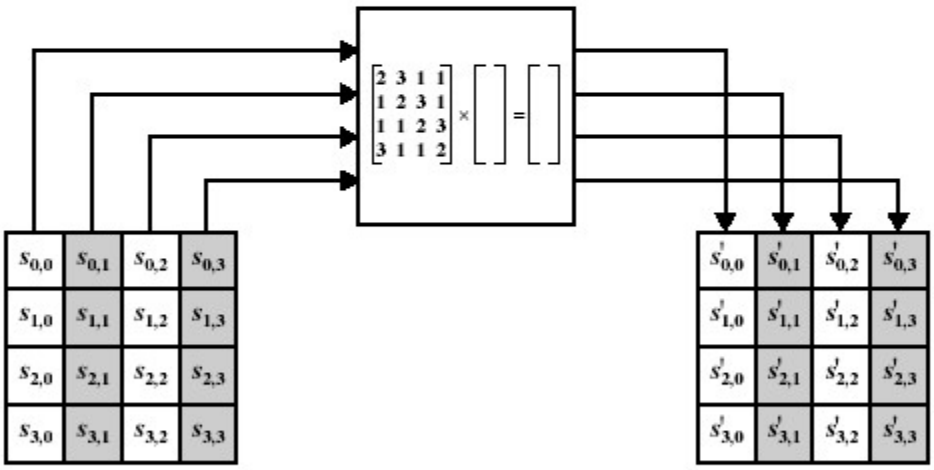
0*0+1*1+0*1+0*1+1*1+0*1+0*0+1*0=1+1=0.
So, we got 1 on the diagonal, and 0 outside of diagonal.
The S-box is designed to be resistant to known cryptanalytic attacks. It provides low correlation between input bits and output bits, output cannot be expressed as simple mathematical function of the input, it has not fixed points (S-box(a)=a).

Shift Row Transformation

The forward shift row transformation, called ShiftRows, is depicted in Fig. 5.5a.



(a) Shift row transformation



(b) Mix column transformation

Figure 5.5 AES Row and Column Operations

Shift Row Transformation (Cont 1)

The 1st row (number 0) is not altered, row number i is shifted left by i-byte circular left shift, i=1, 2, 3. The following is the example of such shift:

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 right circular shift of i-th row by i bytes, i=0,1,2,3.
Shift row transformation ensures that the 4 bytes of one column are spread out to four different columns (Fig. 5.3 illustrates this effect).

Mix Column Transformation

The forward mix column transformation, called MixColumns, operates on each column individually. Each byte is mapped into a new value that is a function of all four bytes in the column. The transformation can be defined as the following matrix multiplication on State (Fig. 5.5b):

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

*

S00	S01	S02	S03
S10	S11	S12	S13
S20	S21	S22	S23
S30	S31	S32	S33

=

S00’	S01’	S02’	S03’
S10’	S11’	S12’	S13’
S20’	S21’	S22’	S23’
S30’	S31’	S32’	S33’

(5.3)

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, multiplications and additions are performed in GF(2^8).
The following is the 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

1st column of the result is obtained by:

{02){87}+{03}{6E}+{46}+{A6} = {47}
{87}+{02}{6E}+{03}{46}+{A6} = {37}
{87}+{6E}+{02}{46}+{03}{A6} = {94}
{03}{87}+{6E}+{46}+{02}{A6} = {ED}

For the 1st equation, we have {02}{87}=(0000 0010)(1000 0111)=
$x(x^7 + x^2 + x + 1) = (x^8 + x^3 + x^2 + x) \bmod (x^8 + x^4 + x^3 + x + 1) = x^4 + x^2 + 1 =$
(0001 0101)={15}

Mix Column Transformation (Cont 1)

{03}{6E}=(0000 0011)(0110 1110)=

$(x+1)(x^6+x^5+x^3+x^2+x) = x^7+x^5+x^4+x =$

(1011 0010) = {B2}

{02){87}+{03}{6E}+{46}+{A6}={15}+{B2}+{46}+{A6}=

(0001 0101)+

(1011 0010)+

(0100 0110)+

(1010 0110)=

(0100 0111)={47}

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

0E	0B	0D	09		S00	S01	S02	S03		S00'	S01'	S02'	S03'	
09	0E	0B	0D	*	S10	S11	S12	S13	=	S10'	S11'	S12'	S13'	(5.5)
0D	09	0E	0B		S20	S21	S22	S23		S20'	S21'	S22'	S23'	
0B	0D	09	0E		S30	S31	S32	S33		S30'	S31'	S32'	S33'	

To show that matrix in (5.5) is inverse of matrix in (5.3), we are to check that their product in GF(2^8) is a unity matrix. Let's make such partial check for S00' (product of 0^{th} row by 0^{th} column):

S00'=(0E}{02}+{0B}{01}+{0D}{01}+{09}{03}=(0E}{02}+{0B}+{0D}+{09}{03}

{0E}{02}=(0000 1110)(0000 0010)= $(x^3+x^2+x)x = x^4+x^3+x^2$ =(0001 1100)={1C}

{09}{03}=(0000 1001)(0000 0011)= $(x^3+1)(x+1) = x^4+x+x^3+1$ =(0001 1011)={1B}

(0E}{02}+{0B}+{0D}+{09}{03}={1C}+{0B}+{0D}+{1B}=

(0001 1100)+

(0000 1011)+

(0000 1101)+

(0001 1011)=

(0000 0001)={01}

The other elements are verified similarly.

The AES document describes MixColumns in terms of polynomial arithmetic. In the standard, MixColumns is defined by considering each column of State to be a four-term polynomial with coefficients in GF(2^8). Each column is multiplied modulo (x^4+1) by the fixed polynomial a(x), given by

Mix Column Transformation (Cont 2)

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \qquad (5.7)$$

Let's show that such multiplication by polynomial (5.7) is equivalent to matrix multiplication, represented by (5.3). Each column of State matrix is viewed as set of coefficients of respective polynomial, e.g., 1st column of State corresponds to the polynomial:

$$S0(x) = S_{30}x^3 + S_{20}x^2 + S_{10}x + S_{00}$$

Then

$$a(x)S0(x) = (\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\})(S_{30}x^3 + S_{20}x^2 + S_{10}x + S_{00}) =$$

$$(\{03\}S_{30}x^6 + (\{03\}S_{20} + S_{30})x^5 + (\{03\}S_{10} + S_{20} + S_{30})x^4 + (\{02\}S_{30} + S_{20} + S_{10} + \{03\}S_{00})x^3 +$$

$$(\{02\}S_{20} + S_{10} + S_{00})x^2 + (\{02\}S_{10} + S_{00})x + \{02\}S_{00}) \bmod(x^4 + 1) =$$

$$(\{03\}S_{30}x^6 \bmod(x^4 + 1) + (\{03\}S_{20} + S_{30})x^5 \bmod(x^4 + 1) + (\{03\}S_{10} + S_{20} + S_{30})x^4 \bmod(x^4 + 1)$$

$$+ (\{02\}S_{30} + S_{20} + S_{10} + \{03\}S_{00})x^3 + (\{02\}S_{20} + S_{10} + S_{00})x^2 + (\{02\}S_{10} + S_{00})x + \{02\}S_{00}) =$$

$$(\{03\}S_{30}x^2 + (\{03\}S_{20} + S_{30})x + (\{03\}S_{10} + S_{20} + S_{30})$$

$$+ (\{02\}S_{30} + S_{20} + S_{10} + \{03\}S_{00})x^3 + (\{02\}S_{20} + S_{10} + S_{00})x^2 + (\{02\}S_{10} + S_{00})x + \{02\}S_{00}) =$$

$$(\{02\}S_{30} + S_{20} + S_{10} + \{03\}S_{00})x^3 + (\{03\}S_{30} + \{02\}S_{20} + S_{10} + S_{00})x^2 + (S_{30} + \{03\}S_{20} + \{02\}S_{10} + S_{00})x +$$

$$(S_{30} + S_{20} + \{03\}S_{10} + \{02\}S_{00}) =$$

$$(\{02\}S_{00} + \{03\}S_{10} + S_{20} + S_{30}) +$$

$$(S_{00} + \{02\}S_{10} + \{03\}S_{20} + S_{30})x +$$

$$(S_{00} + S_{10} + \{02\}S_{20} + \{03\}S_{30})x^2 +$$

$$(\{03\}S_{00} + S_{10} + S_{20} + \{02\}S_{30})x^3$$

In the last polynomial, coefficients are just same as used in matrix multiplication according to (5.3). Actually, 1st column of the result in (5.3) may be written as follows:

S00'={02}S00+{03}S10+S20+S30

S10'=S00+{02)S10+{03}S20+S30

S20'=S00+S10+{02}S20+{03}S30

S30'={03}S00+S10+S20+{02}S30

Similarly, it may be shown that the transformation in the (5.5) corresponds to treating each column as a 4-term polynomial and multiplying each column by b(x), given by

$$b(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \qquad (5.8)$$

Mix Column Transformation (Cont 3)

It can be shown that $b(x) = a^{-1}(x) \bmod (x^4 + 1)$

The mix column transformation combined with the shift row transformation ensures that after a few rounds, all output bits depend on all input bits.

Add Round Key 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 AddRoundKey transformation is as simple as possible and affects every bit of State. The complexity of the round key expansion together with the complexity of other stages of AES, ensures security.

AES Key Expansion

The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (156 bytes). The following pseudo code describes the expansion:

```
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)) XOR Rcon[i/4];
    W[i]=w[i-4] XOR temp;
 }
}
```

The key is copied into the 1st 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 array w is a multiple of 4, a more complex function is used. Figure 5.6 illustrates the generation of the 1st eight words of the expanded key, using the symbol g to represent the complex function. The function g consists of the following subfunctions:

1. RotWord performs a 1-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 (Table 5.4a)
3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j]

AES Key Expansion (Cont 1)

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 1st four bytes (1st 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 XOR w[i-4]
36	7f8d292f	8d292f7f	5da515d2	1b000000	46a515d2	Ead27321	Ac7766f3

The inclusion of a round-dependent constant eliminates the symmetry, or similarity, between the ways in which round keys are generated in different rounds. It is an invertible transformation. Each key bit affects many round key bits. It is a nonlinear transformation.