Diyala University
College of Engineering
Computer & Software
Engineering Department

Fourth Year 2012/2013

Advanced Encryption Standard Algorithm (AES)

Chapter2: Part5

PRESENTED BY
DR. ALI J. ABBOUD

Objectives

- ❑ To review a short history of AES
- ❑ To define the basic structure of AES
- ❑ To define the transformations used by AES
- ❑ To define the key expansion process
- ❑ To discuss different implementations

2.4.1 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.**

- **In February 2001, NIST announced that a draft of the Federal Information Processing Standard (FIPS) was available for public review and comment. Finally, AES was published as FIPS 197 in the Federal Register in December 2001.**

- **The criteria defined by NIST for selecting AES fall into three areas:**
  1. **Security**
  2. **Cost**
  3. **Implementation.**

- **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.**

2.4.1 Introduction

- RC2 is a block encryption algorithm which may be considered as a proposal for a DES replacement.
- The input and output block sizes are 64 bits each.
- The key size is variable, from one byte up to 128 bytes, although the current implementation uses eight bytes.
- The algorithm is designed to be easy to implement on 16-bit microprocessors. On an IBM AT, the encryption runs about twice as fast as DES (assuming that key expansion has been done)

2.4.2 Algorithm Description

- **We use the term "word" to denote a 16-bit quantity.**
- **The symbol + will denote twos-complement addition.**
- **The symbol & will denote the bitwise "and" operation.**
- **The term XOR will denote the bitwise "exclusive-or" operation.**
- **The symbol ~ will denote bitwise complement.**
- **The symbol ^ will denote the exponentiation operation.**
- **The term MOD will denote the modulo operation.**

2.4.2 Algorithm Description

There are three separate algorithms involved:

- ❑ **Key expansion: This takes a (variable-length) input key and produces an expanded key consisting of 64 words K[0],...,K[63].**
- ❑ **Encryption: This takes a 64-bit input quantity stored in words R[0], ..., R[3] and encrypts it "in place" (the result is left in R[0], ..., R[3]).**
- ❑ **Decryption: The inverse operation to encryption.**

2.4.3 Key Expansion

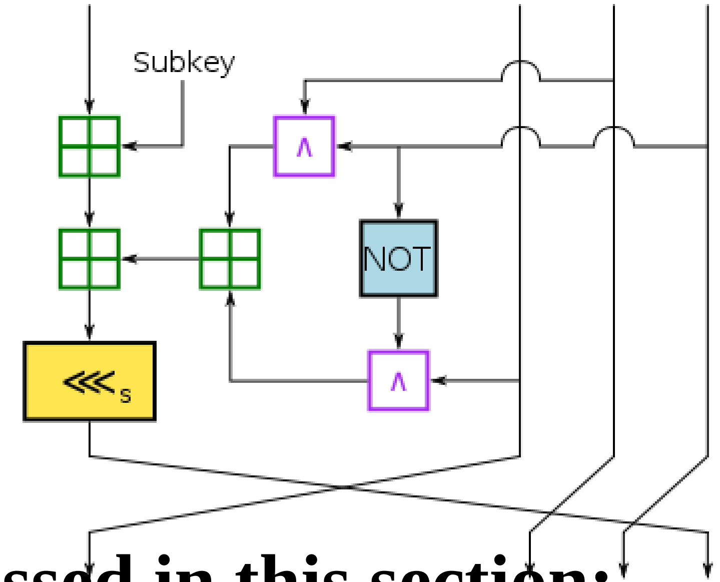
There are three separate algorithms involved:

- ❑ **Key expansion: This takes a (variable-length) input key and produces an expanded key consisting of 64 words K[0],...,K[63].**

- ❑ **Encryption: This takes a 64-bit input quantity stored in words R[0], ..., R[3] and encrypts it "in place" (the result is left in R[0], ..., R[3]).**

- ❑ **Decryption: The inverse operation to encryption.**

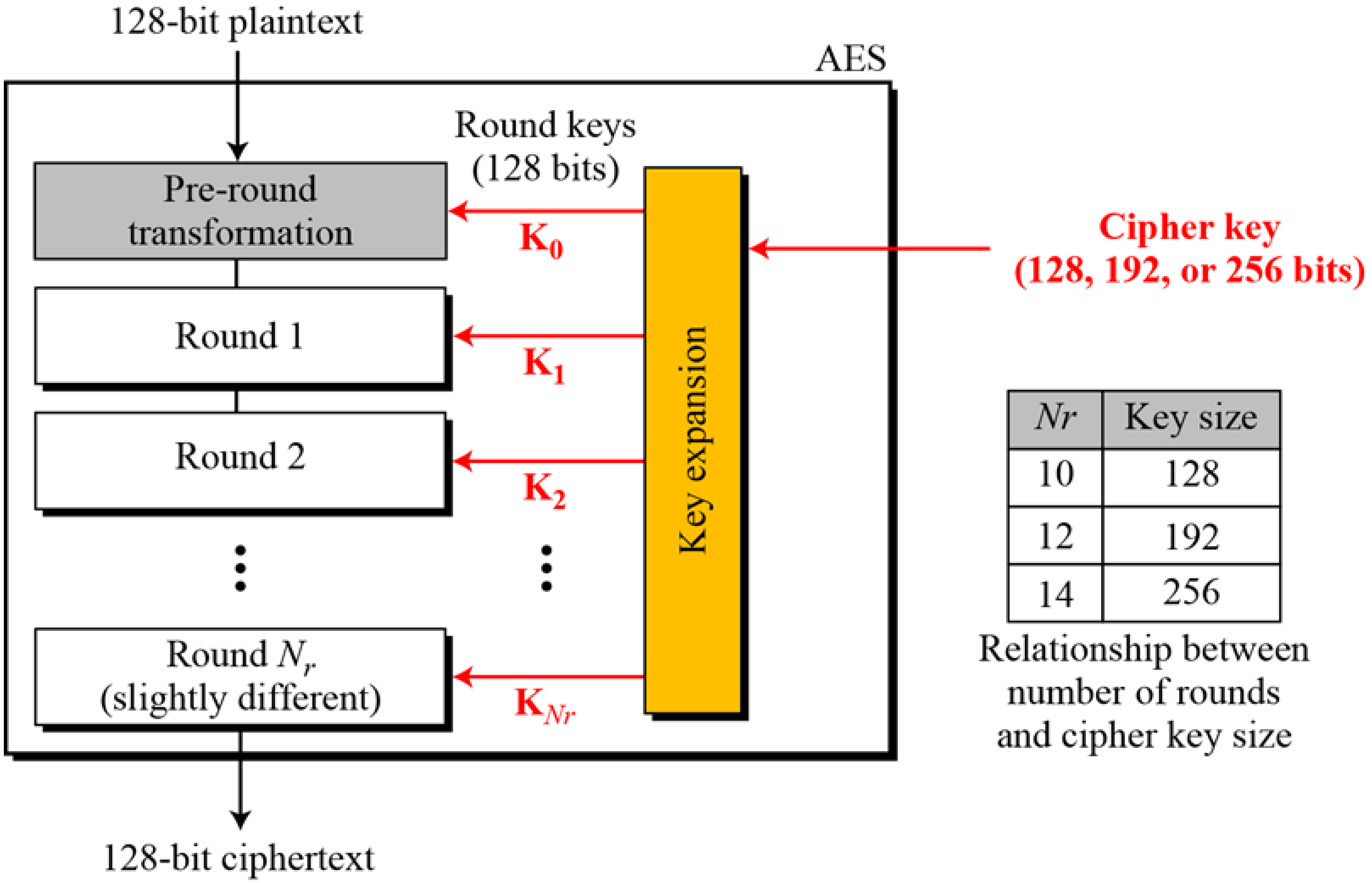
2.4.1 Introduction



Topics discussed in this section:

1. **History**
2. **Criteria**
3. **Rounds**
4. **Data Units**
5. **Structure of Each Round**

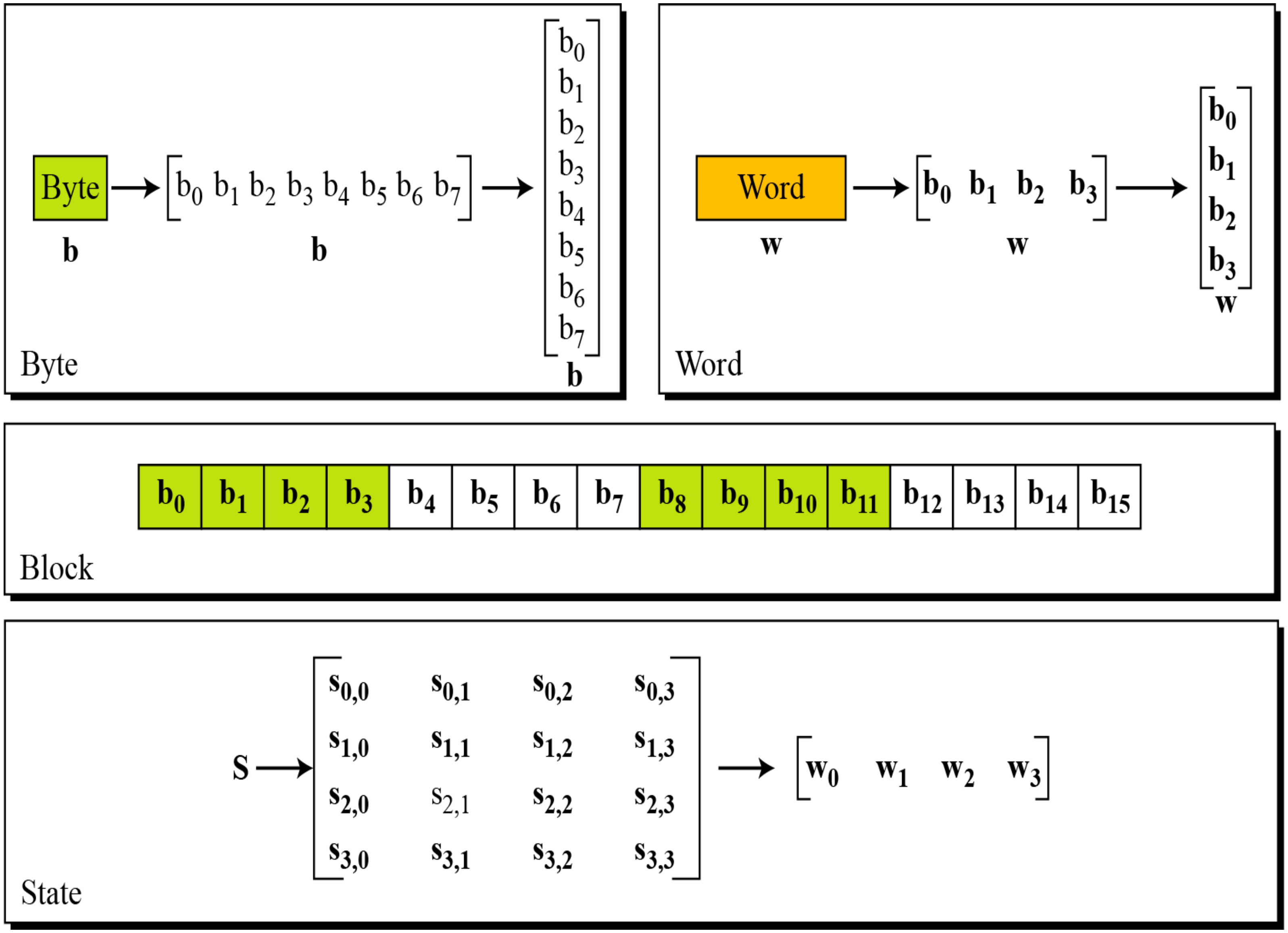
2.4.2 Data Units



Nr	Key size
10	128
12	192
14	256

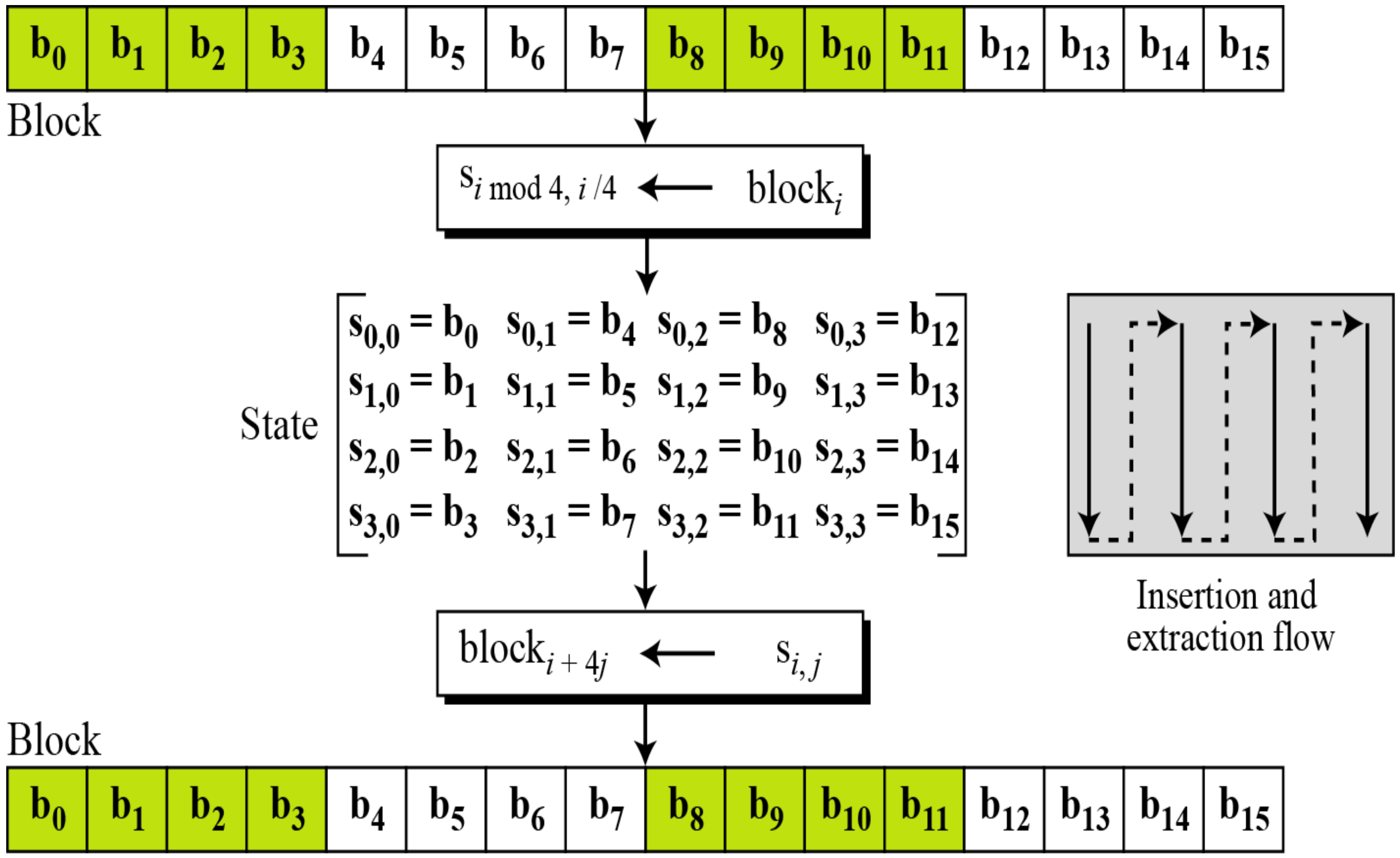
Relationship between number of rounds and cipher key size

2.4.3 Design of AES Encryption Cipher



Byte
b
$b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7$
b
$b_0 \dots b_7$
b
Byte
Word
w
$b_0\ b_1\ b_2\ b_3$
w
$b_0\ b_1\ b_2\ b_3$
w
Word
$b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8\ b_9\ b_{10}\ b_{11}\ b_{12}\ b_{13}\ b_{14}\ b_{15}$
Block
S
$s_{0,0}\ s_{0,1}\ s_{0,2}\ s_{0,3}$
$s_{1,0}\ s_{1,1}\ s_{1,2}\ s_{1,3}$
$s_{2,0}\ s_{2,1}\ s_{2,2}\ s_{2,3}$
$s_{3,0}\ s_{3,1}\ s_{3,2}\ s_{3,3}$
$w_0\ w_1\ w_2\ w_3$
State

2.4.4 Block-to-State and State-to-Block Transformation



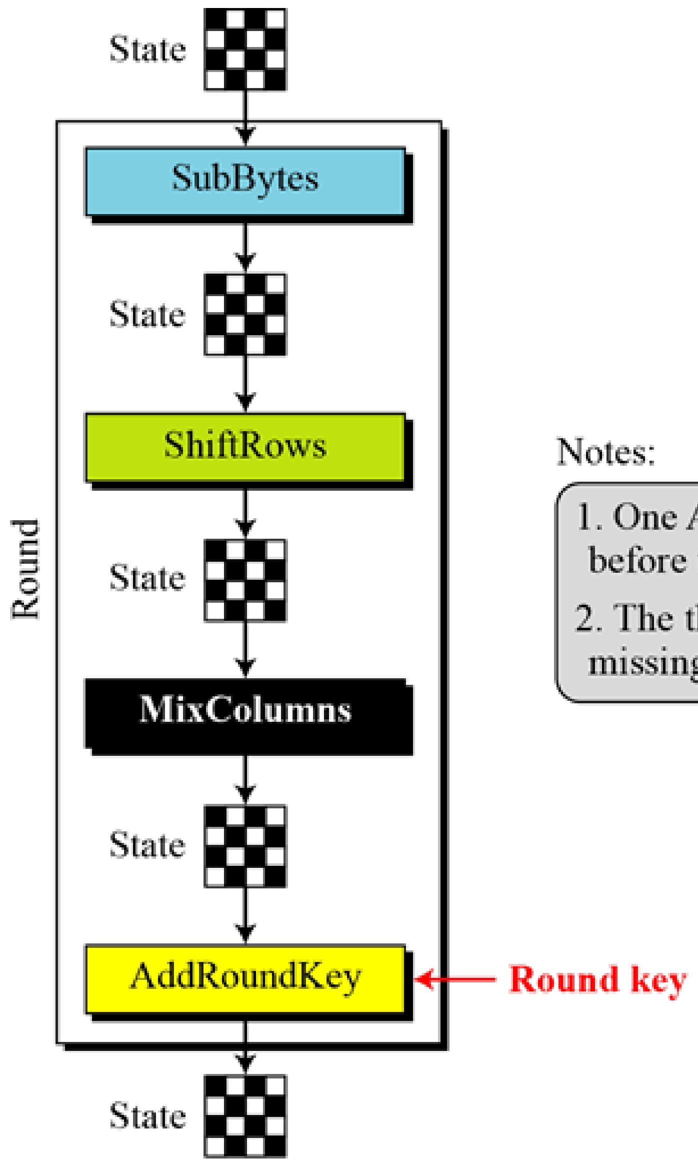
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15}
Block
$s_{i \bmod 4,\, i/4} \leftarrow \text{block}_i$
State
$s_{0,0} = b_0$ $s_{0,1} = b_4$ $s_{0,2} = b_8$ $s_{0,3} = b_{12}$
$s_{1,0} = b_1$ $s_{1,1} = b_5$ $s_{1,2} = b_9$ $s_{1,3} = b_{13}$
$s_{2,0} = b_2$ $s_{2,1} = b_6$ $s_{2,2} = b_{10}$ $s_{2,3} = b_{14}$
$s_{3,0} = b_3$ $s_{3,1} = b_7$ $s_{3,2} = b_{11}$ $s_{3,3} = b_{15}$
$\text{block}_{i+4j} \leftarrow s_{i,j}$
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}
Insertion and extraction flow

2.4.4 Example: 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	23	19	19

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

2.4.5 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.

2.4.6 *TRANSFORMATIONS*

To provide security, AES uses four types of transformations:

1. Substitution
2. Permutation
3. Mixing
4. Key-Adding

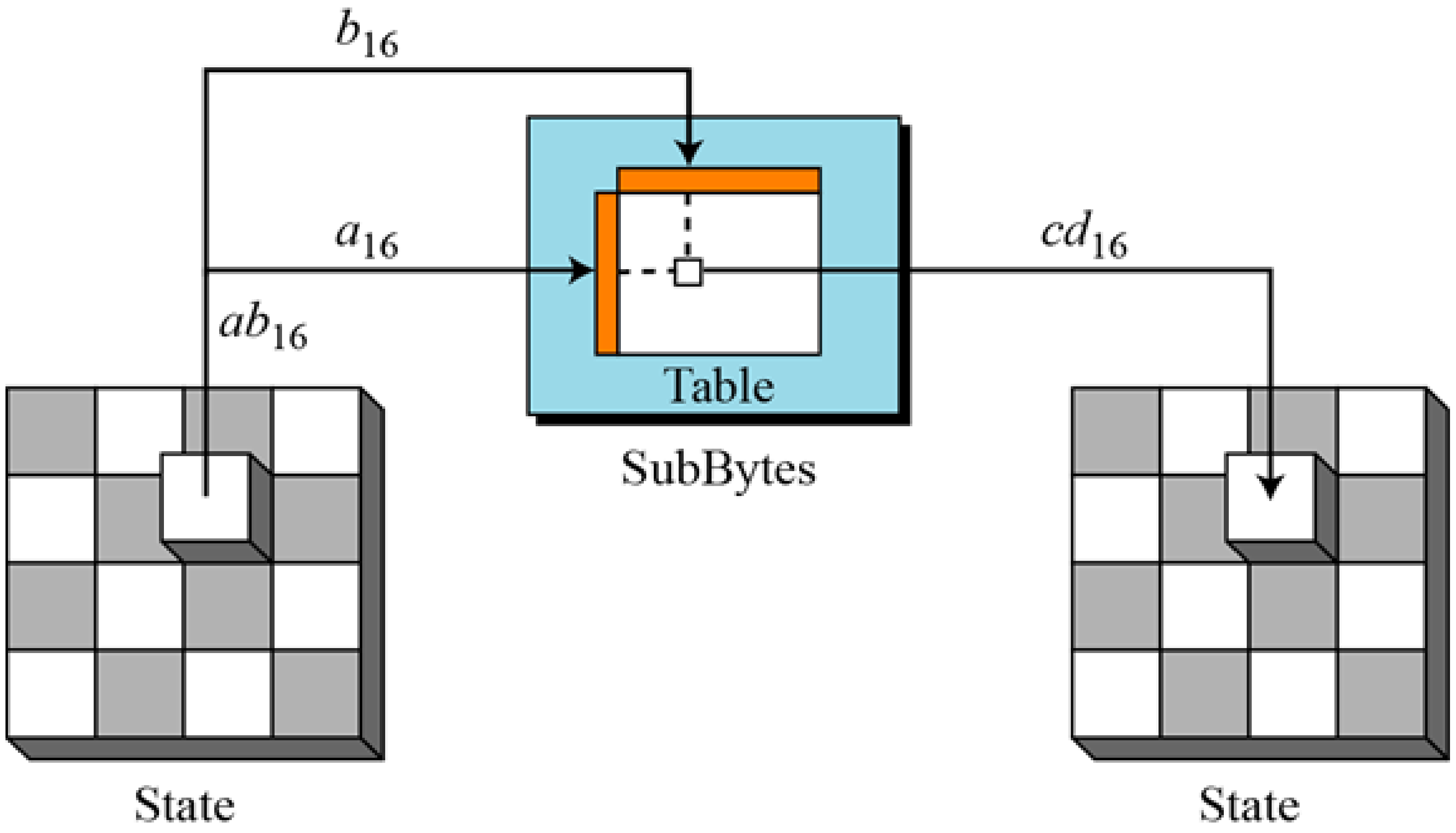
AES, like DES, uses substitution. AES uses two invertible transformations

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.

2.4.6.1 *Substitution*



2.4.6.1 *AES S-Box Lookup Table*

During encryption each value of the state is replaced with the corresponding SBOX value

AES S-Box Lookup 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
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

For example HEX 19 would get replaced with HEX D4

2.4.6.1 AES Inverse S-Box Lookup Table

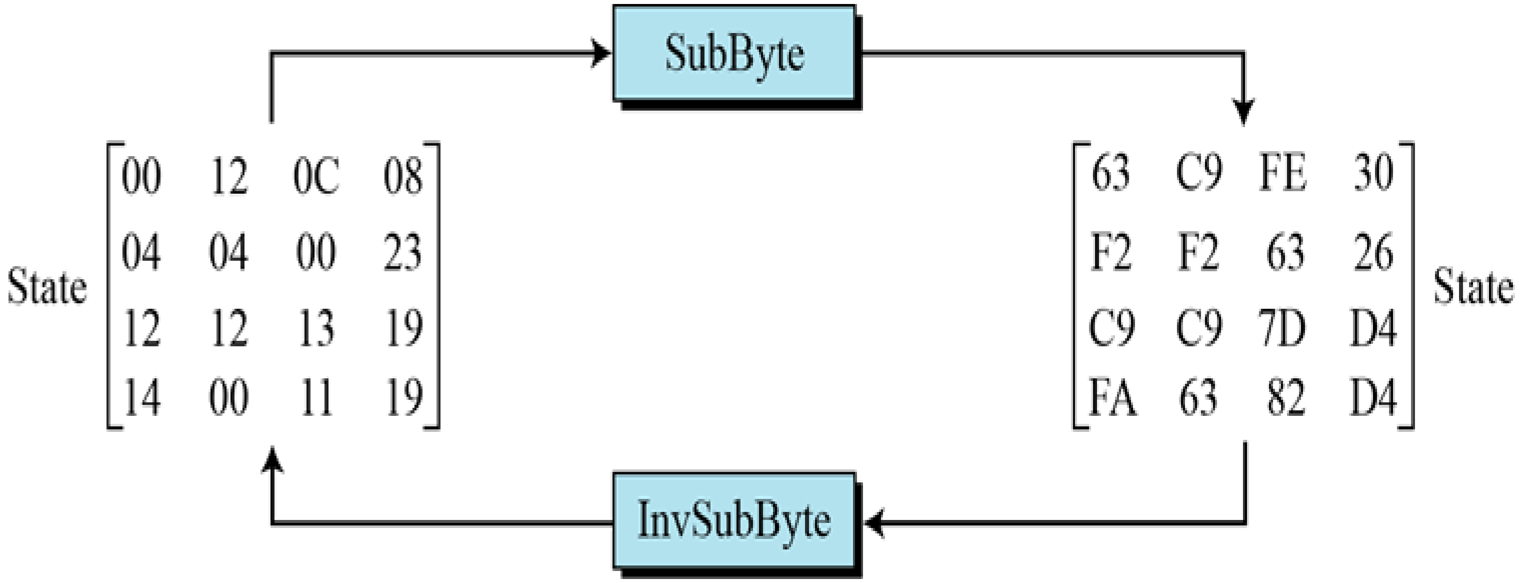
During decryption each value in the state is replaced with the corresponding inverse of the SBOX

	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

For example HEX D4 would get replaced with HEX 19

2.4.6.1 Example: Substitution

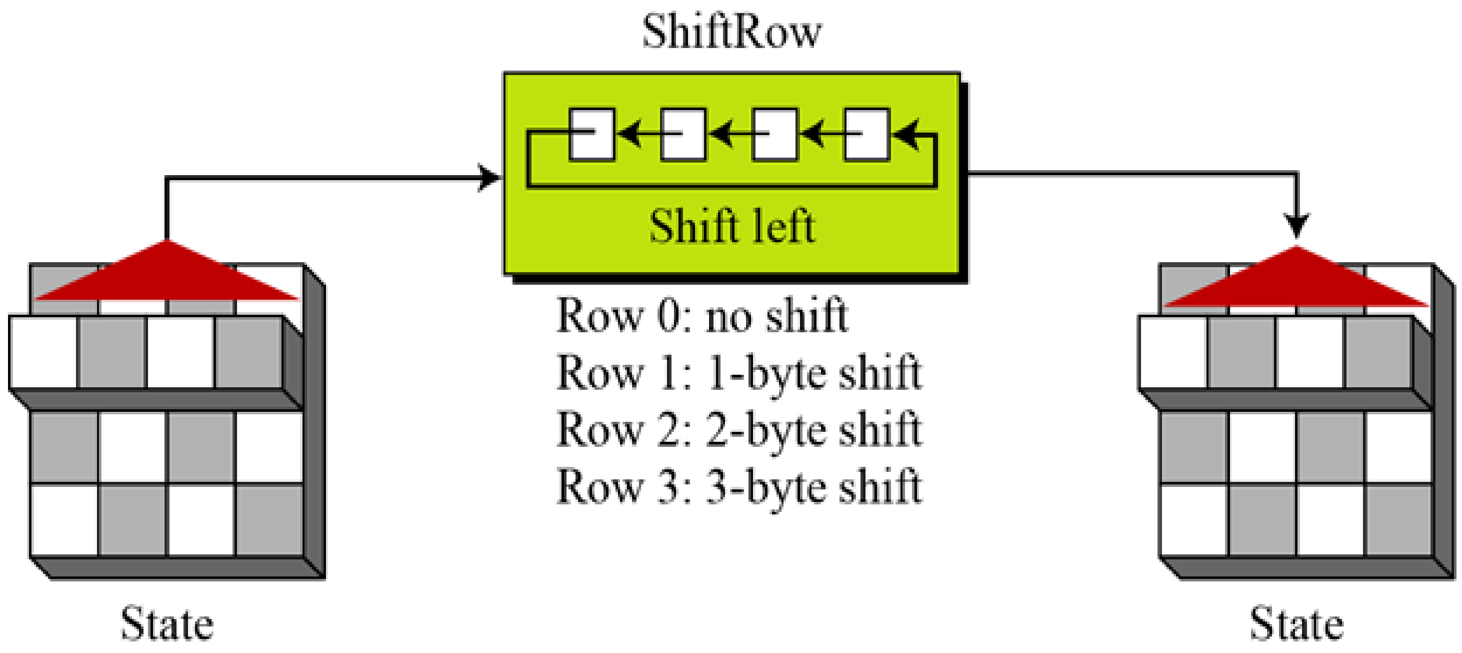
Figure below shows how a state is transformed using the SubBytes transformation. The figure also shows that the InvSubBytes transformation creates the original one. Note that if the two bytes have the same values, their transformation is also the same.



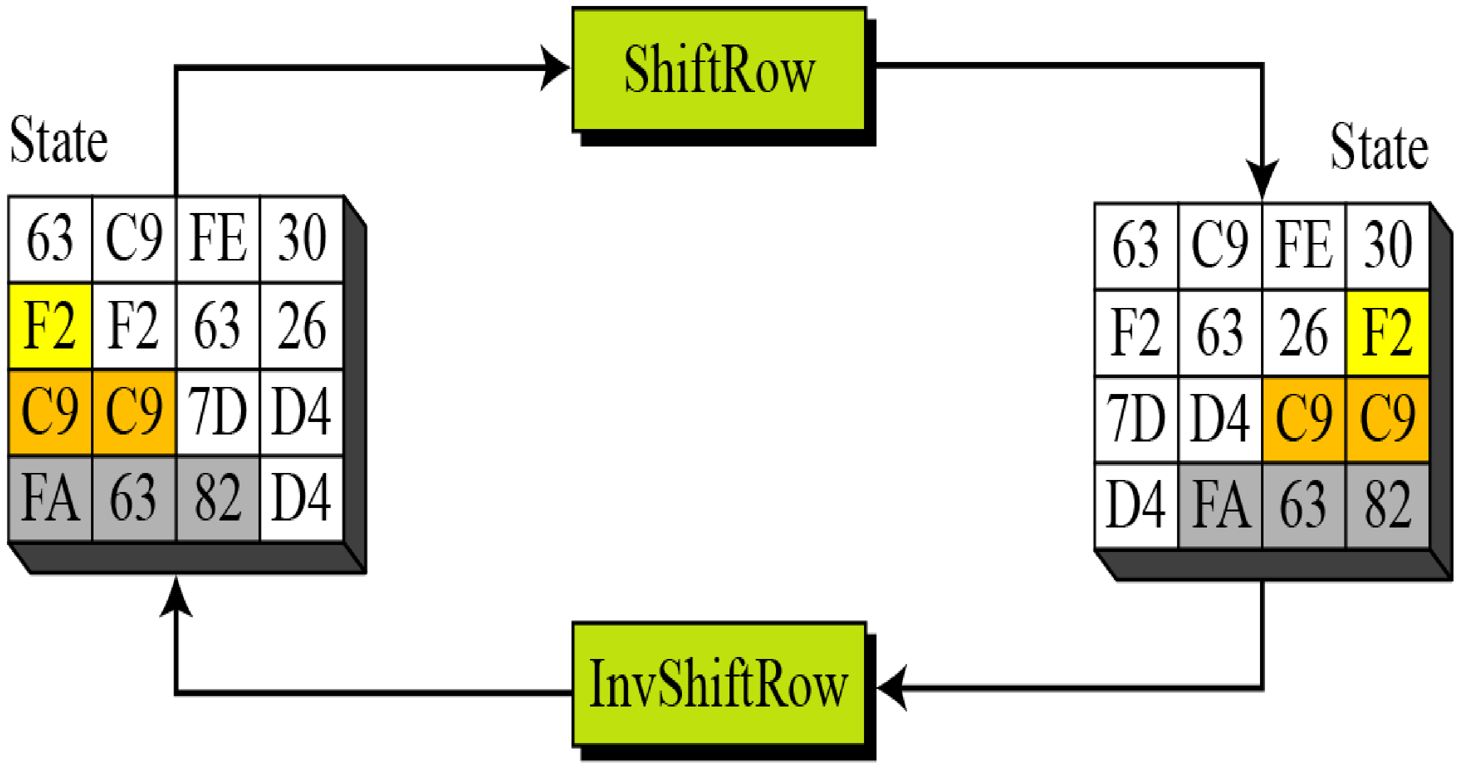
2.4.6.2 Permutation

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

ShiftRows
In the encryption, the transformation is called ShiftRows.



2.4.6.2 *Permutation Shift Rows Example1*



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

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

2.4.6.2 *Permutation Shift Rows Example2*

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Row1	0
Row2	1
Row3	2
Roe4	3

From				To			
1	5	9	13	1	5	9	13
2	6	10	14	6	10	14	2
3	7	11	15	11	15	3	7
4	8	12	16	16	4	8	12

From				To			
1	5	9	13	1	5	9	13
2	6	10	14	14	2	6	10
3	7	11	15	11	15	3	7
4	8	12	16	8	12	16	4

2.4.6.3 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.

$$
\begin{array}{l}
a\mathbf{x} + b\mathbf{y} + c\mathbf{z} + d\mathbf{t} \longrightarrow \\
e\mathbf{x} + f\mathbf{y} + g\mathbf{z} + h\mathbf{t} \longrightarrow \\
i\mathbf{x} + j\mathbf{y} + k\mathbf{z} + l\mathbf{t} \longrightarrow \\
m\mathbf{x} + n\mathbf{y} + o\mathbf{z} + p\mathbf{t} \longrightarrow
\end{array}
\begin{array}{ccccc}
\begin{bmatrix} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{bmatrix} & = & \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} & \times & \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{t} \end{bmatrix} \\
\text{New matrix} & & \textbf{Constant matrix} & & \text{Old matrix}
\end{array}
$$

2.4.6.3 Mixing / Constant Matrices

Constant matrices used by MixColumns and InvMixColumns

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

MixColumns

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

2.4.6.3 Mixing / Example

Multiplication Matrix

```
2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
```

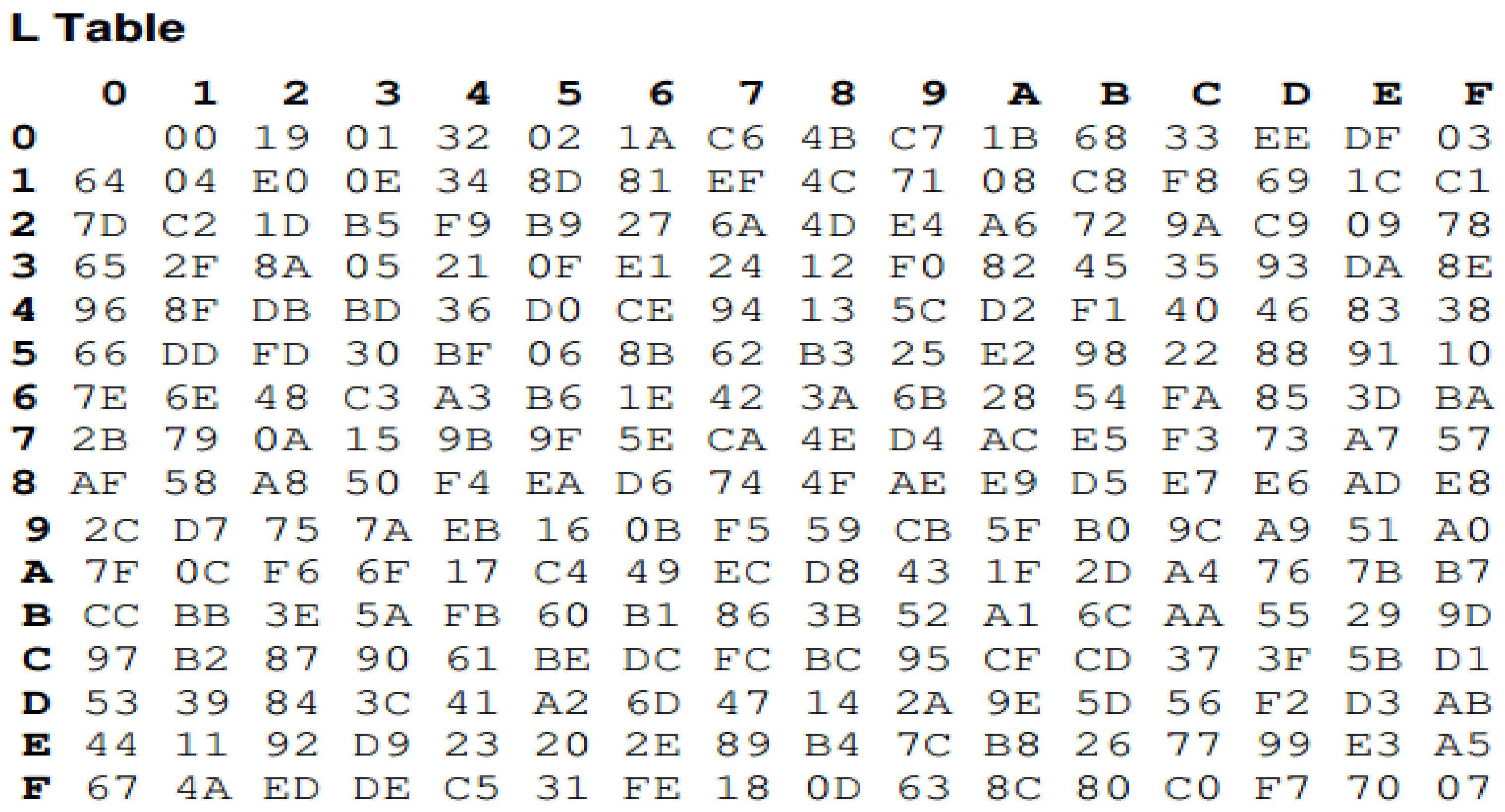
16 byte State

```
b1  b5  b9   b13
b2  b6  b10  b14
b3  b7  b11  b15
b4  b8  b12  b16
```

```
b1 = (b1 * 2) XOR (b2*3) XOR (b3*1) XOR (b4*1)
b2 = (b1 * 1) XOR (b2*2) XOR (b3*3) XOR (b4*1)
b3 = (b1 * 1) XOR (b2*1) XOR (b3*2) XOR (b4*3)
b4 = (b1 * 3) XOR (b2*1) XOR (b3*1) XOR (b4*2)

b5 = (b5 * 2) XOR (b6*3) XOR (b7*1) XOR (b8*1)
b6 = (b5 * 1) XOR (b6*2) XOR (b7*3) XOR (b8*1)
b7 = (b5 * 1) XOR (b6*1) XOR (b7*2) XOR (b8*3)
b8 = (b5 * 3) XOR (b6*1) XOR (b7*1) XOR (b8*2)
```

2.4.6.3 Mixing / Galois Field Multiplication

The result of the multiplication is simply the result of a lookup of the L table, followed by the addition of the results, followed by a lookup to the E table. The addition is a regular mathematical addition represented by +, not a bitwise AND.



L Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		00	19	01	32	02	1A	C6	4B	C7	1B	68	33	EE	DF	03
1	64	04	E0	0E	34	8D	81	EF	4C	71	08	C8	F8	69	1C	C1
2	7D	C2	1D	B5	F9	B9	27	6A	4D	E4	A6	72	9A	C9	09	78
3	65	2F	8A	05	21	0F	E1	24	12	F0	82	45	35	93	DA	8E
4	96	8F	DB	BD	36	D0	CE	94	13	5C	D2	F1	40	46	83	38
5	66	DD	FD	30	BF	06	8B	62	B3	25	E2	98	22	88	91	10
6	7E	6E	48	C3	A3	B6	1E	42	3A	6B	28	54	FA	85	3D	BA
7	2B	79	0A	15	9B	9F	5E	CA	4E	D4	AC	E5	F3	73	A7	57
8	AF	58	A8	50	F4	EA	D6	74	4F	AE	E9	D5	E7	E6	AD	E8
9	2C	D7	75	7A	EB	16	0B	F5	59	CB	5F	B0	9C	A9	51	A0
A	7F	0C	F6	6F	17	C4	49	EC	D8	43	1F	2D	A4	76	7B	B7
B	CC	BB	3E	5A	FB	60	B1	86	3B	52	A1	6C	AA	55	29	9D
C	97	B2	87	90	61	BE	DC	FC	BC	95	CF	CD	37	3F	5B	D1
D	53	39	84	3C	41	A2	6D	47	14	2A	9E	5D	56	F2	D3	AB
E	44	11	92	D9	23	20	2E	89	B4	7C	B8	26	77	99	E3	A5
F	67	4A	ED	DE	C5	31	FE	18	0D	63	8C	80	C0	F7	70	07

2.4.6.3 Mixing / Galois Field Multiplication

E Table

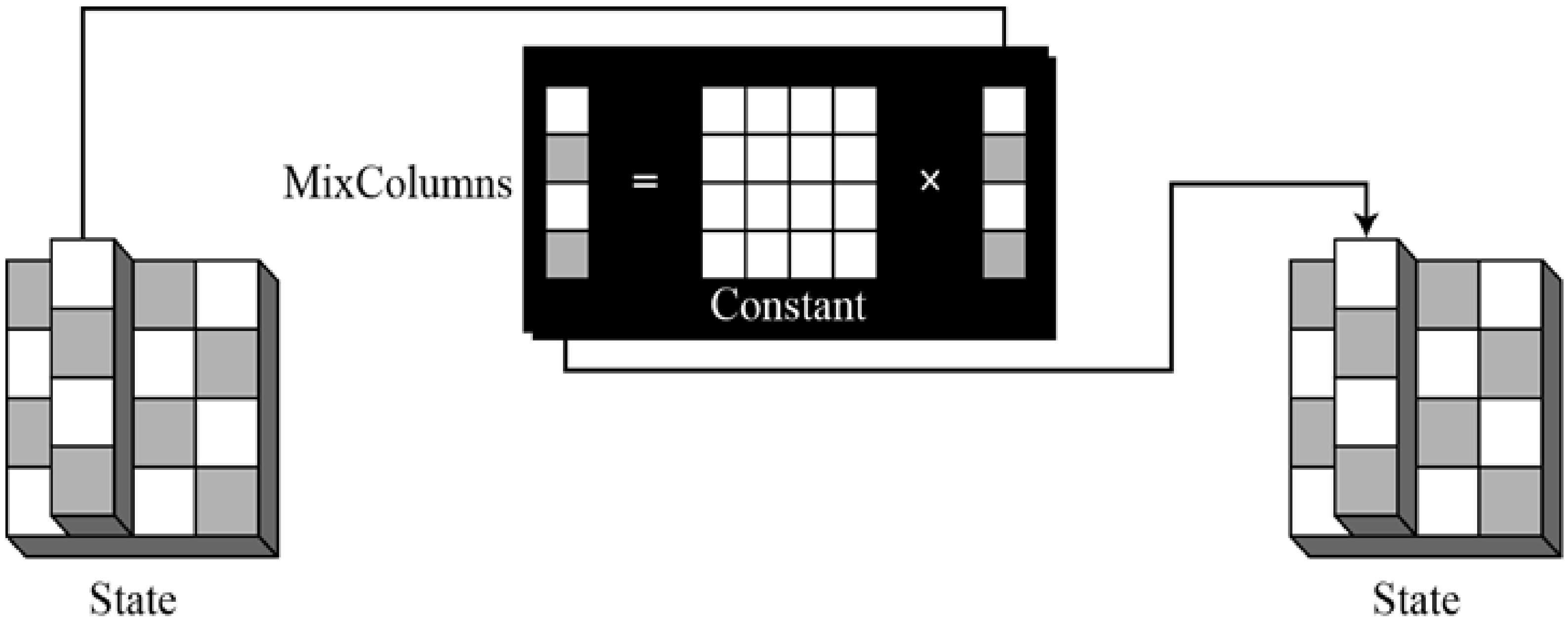
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	01	03	05	0F	11	33	55	FF	1A	2E	72	96	A1	F8	13	35
1	5F	E1	38	48	D8	73	95	A4	F7	02	06	0A	1E	22	66	AA
2	E5	34	5C	E4	37	59	EB	26	6A	BE	D9	70	90	AB	E6	31
3	53	F5	04	0C	14	3C	44	CC	4F	D1	68	B8	D3	6E	B2	CD
4	4C	D4	67	A9	E0	3B	4D	D7	62	A6	F1	08	18	28	78	88
5	83	9E	B9	D0	6B	BD	DC	7F	81	98	B3	CE	49	DB	76	9A
6	B5	C4	57	F9	10	30	50	F0	0B	1D	27	69	BB	D6	61	A3
7	FE	19	2B	7D	87	92	AD	EC	2F	71	93	AE	E9	20	60	A0
8	FB	16	3A	4E	D2	6D	B7	C2	5D	E7	32	56	FA	15	3F	41
9	C3	5E	E2	3D	47	C9	40	C0	5B	ED	2C	74	9C	BF	DA	75
A	9F	BA	D5	64	AC	EF	2A	7E	82	9D	BC	DF	7A	8E	89	80
B	9B	B6	C1	58	E8	23	65	AF	EA	25	6F	B1	C8	43	C5	54
C	FC	1F	21	63	A5	F4	07	09	1B	2D	77	99	B0	CB	46	CA
D	45	CF	4A	DE	79	8B	86	91	A8	E3	3E	42	C6	51	F3	0E
E	12	36	5A	EE	29	7B	8D	8C	8F	8A	85	94	A7	F2	0D	17
F	39	4B	DD	7C	84	97	A2	FD	1C	24	6C	B4	C7	52	F6	01

For example if the two Hex values being multiplied are AF * 8 we first lookup L (AF) index which returns B7 and then lookup L (08) which returns 4B.

Once the L table lookup is complete we can then simply add the numbers together. The only trick being that if the addition result is greater then FF we subtract FF from the addition result.

For example AF+B7= 166. Because 166 > FF, we perform: 166-FF which gives us 67.

2.4.6.3 Mixing



Note

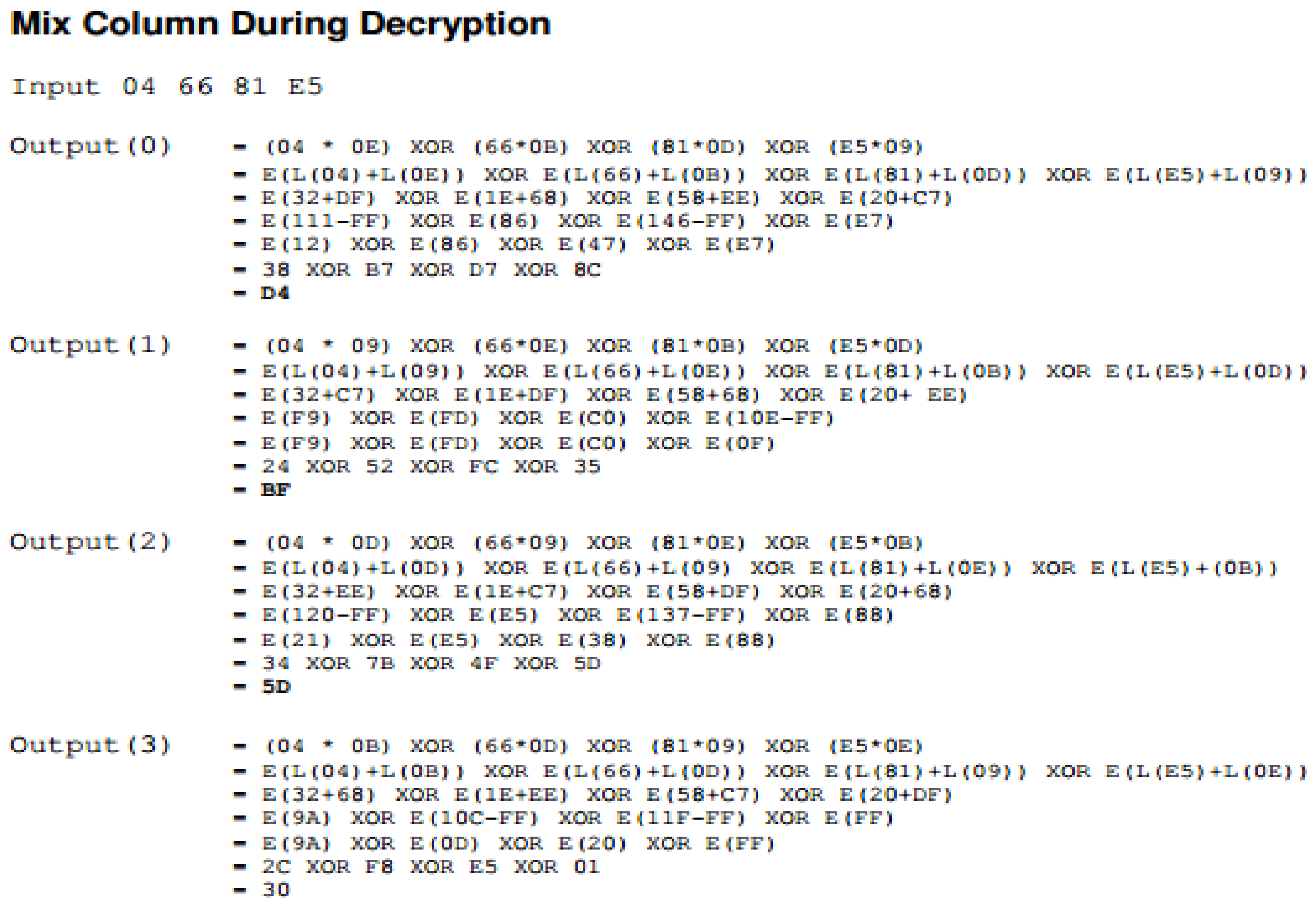
The MixColumns and InvMixColumns transformations are inverses of each other.

2.4.6.3 Mix Column Example

Mix Column Example During Encryption

```
Input = D4 BF 5D 30

Output(0)     = (D4 * 2) XOR (BF*3) XOR (5D*1) XOR (30*1)
              = E(L(D4) + L(02)) XOR E(L(BF) + L(03)) XOR 5D XOR 30
              = E(41 + 19) XOR E(9D + 01) XOR 5D XOR 30
              = E(5A) XOR E(9E) XOR 5D XOR 30

              = B3 XOR DA XOR 5D XOR 30
              = 04

Output(1)     = (D4 * 1) XOR (BF*2) XOR (5D*3) XOR (30*1)
              = D4 XOR E(L(BF)+L(02)) XOR E(L(5D)+L(03)) XOR 30
              = D4 XOR E(9D+19) XOR E(88+01) XOR 30
              = D4 XOR E(B6) XOR E(89) XOR 30
              = D4 XOR 65 XOR E7 XOR 30
              = 66

Output(2)     = (D4 * 1) XOR (BF*1) XOR (5D*2) XOR (30*3)
              = D4 XOR BF XOR E(L(5D)+L(02)) XOR E(L(30)+L(03))
              = D4 XOR BF XOR E(88+19) XOR E(65+01)
              = D4 XOR BF XOR E(A1) XOR E(66)
              = D4 XOR BF XOR BA XOR 50
              = 81

Output(3)     = (D4 * 3) XOR (BF*1) XOR (5D*1) XOR (30*2)
              = E(L(D4)+L(3)) XOR BF XOR 5D XOR E(L(30)+L(02))
              = E(41+01) XOR BF XOR 5D XOR E(65+19)
              = E(42) XOR BF XOR 5D XOR E(7E)
              = 67 XOR BF XOR 5D XOR 60
              = E5
```

2.4.6.3 Mix Column Example



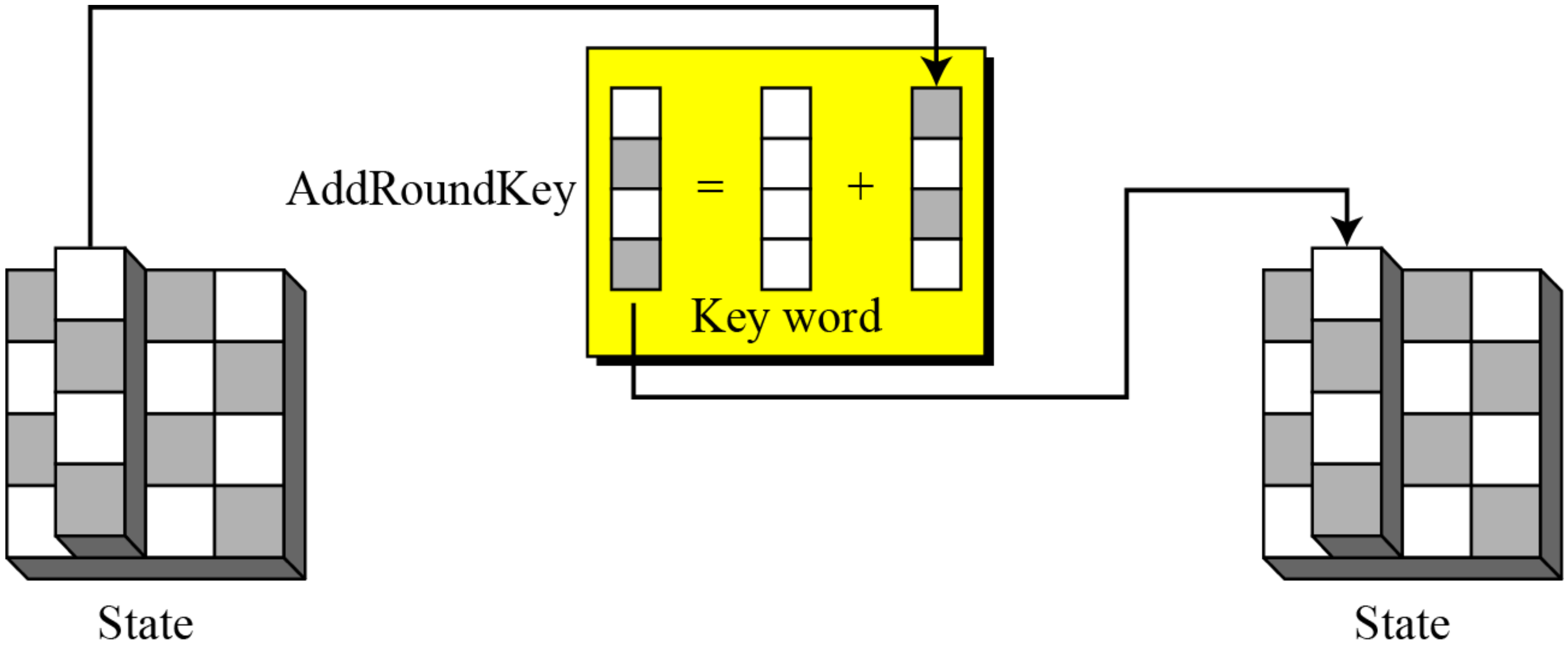
Mix Column During Decryption

```
Input 04 66 81 E5

Output(0)     = (04 * 0E) XOR (66*0B) XOR (81*0D) XOR (E5*09)
              = E(L(04)+L(0E)) XOR E(L(66)+L(0B)) XOR E(L(81)+L(0D)) XOR E(L(E5)+L(09))
              = E(32+DF) XOR E(1E+68) XOR E(58+EE) XOR E(20+C7)
              = E(111-FF) XOR E(86) XOR E(146-FF) XOR E(E7)
              = E(12) XOR E(86) XOR E(47) XOR E(E7)
              = 38 XOR B7 XOR D7 XOR 8C
              = D4

Output(1)     = (04 * 09) XOR (66*0E) XOR (81*0B) XOR (E5*0D)
              = E(L(04)+L(09)) XOR E(L(66)+L(0E)) XOR E(L(81)+L(0B)) XOR E(L(E5)+L(0D))
              = E(32+C7) XOR E(1E+DF) XOR E(58+68) XOR E(20+ EE)
              = E(F9) XOR E(FD) XOR E(C0) XOR E(10E-FF)
              = E(F9) XOR E(FD) XOR E(C0) XOR E(0F)
              = 24 XOR 52 XOR FC XOR 35
              = BF

Output(2)     = (04 * 0D) XOR (66*09) XOR (81*0E) XOR (E5*0B)
              = E(L(04)+L(0D)) XOR E(L(66)+L(09) XOR E(L(81)+L(0E)) XOR E(L(E5)+(0B))
              = E(32+EE) XOR E(1E+C7) XOR E(58+DF) XOR E(20+68)
              = E(120-FF) XOR E(E5) XOR E(137-FF) XOR E(88)
              = E(21) XOR E(E5) XOR E(38) XOR E(88)
              = 34 XOR 7B XOR 4F XOR 5D
              = 5D

Output(3)     = (04 * 0B) XOR (66*0D) XOR (81*09) XOR (E5*0E)
              = E(L(04)+L(0B)) XOR E(L(66)+L(0D)) XOR E(L(81)+L(09)) XOR E(L(E5)+L(0E))
              = E(32+68) XOR E(1E+EE) XOR E(58+C7) XOR E(20+DF)
              = E(9A) XOR E(10C-FF) XOR E(11F-FF) XOR E(FF)
              = E(9A) XOR E(0D) XOR E(20) XOR E(FF)
              = 2C XOR F8 XOR E5 XOR 01
              = 30
```

2.4.6.4 Add Round Key

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.



2.4.6.4 Add Round Key

Each of the 16 bytes of the state is XORed against each of the 16 bytes of a portion of the expanded key for the current round. The Expanded Key bytes are never reused. So once the first 16 bytes are XORed against the first 16 bytes of the expanded key then the expanded key bytes 1-16 are never used again. The next time the Add Round Key function is called bytes 17-32 are XORed against the state.

The first time Add Round Key gets executed

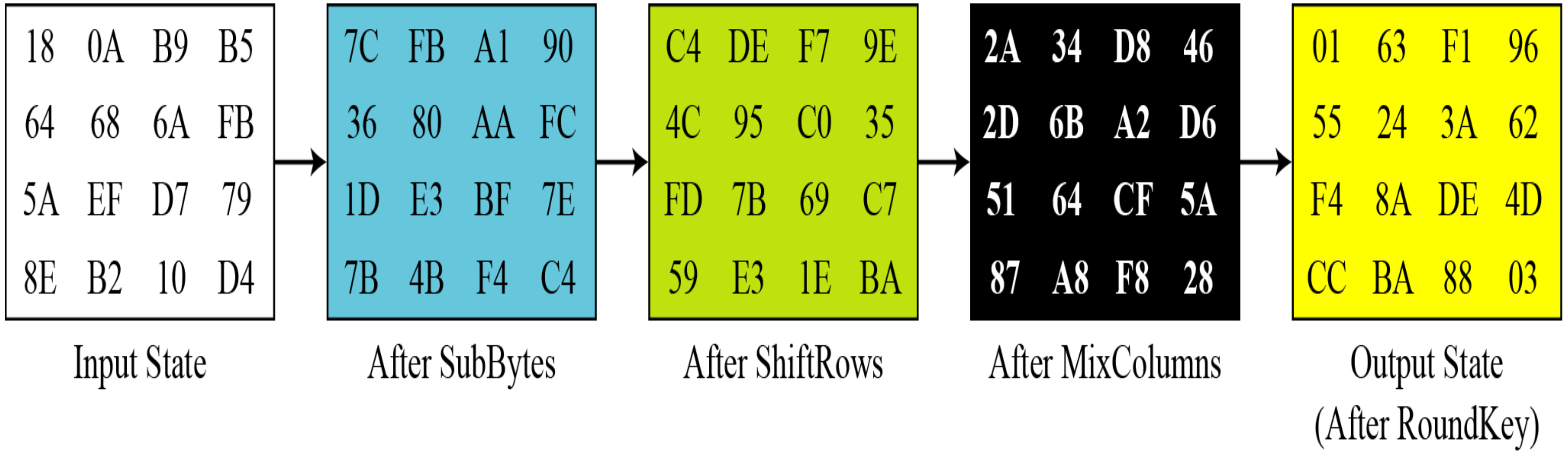
State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR
Exp Key	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

The second time Add Round Key is executed

State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR	XOR
Exp Key	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

And so on for each round of execution.

2.4.6.5 Whole AES Example



2.4.6.6 ANALYSIS OF AES

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.

2.4.6.6 ANALYSIS OF AES

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.

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.

The algorithms used in AES are so simple that they can be easily implemented using cheap processors and a minimum amount of memory.

End of Chapter2/ Part5