Advanced Encryption Standard (AES)

Chapter 7

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

7-1 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.

7.1.1 History.

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.

7.1.2 Criteria

The criteria defined by NIST for selecting AES fall into three areas:

1. ***Security***
2. ***Cost***
3. ***Implementation***

7.1.2 Criteria

The criteria defined by NIST for selecting AES fall into three areas:

1. ***Security***

- ***Emphasis was on security, resistance to cryptanalysis.***
- ***NIST explicitly demanded a 128 bit key to focus on resistance to cryptanalysis attacks other than brute force attack***

7.1.2 Criteria

The criteria defined by NIST for selecting AES fall into three areas:

2. Cost

- ***Second criterion was cost which covers computational efficiency and storage requirements***
- ***for different implementations such as hardware, software or smart cards***

7.1.2 Criteria

The criteria defined by NIST for selecting AES fall into three areas:

3. Implementation

- ***Platform flexibility-algorithm must be implementable on any platform***
- ***Simplicity***

1. Feistel ciphers- Uses both invertible and non-invertible components

2. Non-Feistel ciphers-Uses only invertible components

7.1.3 Rounds.

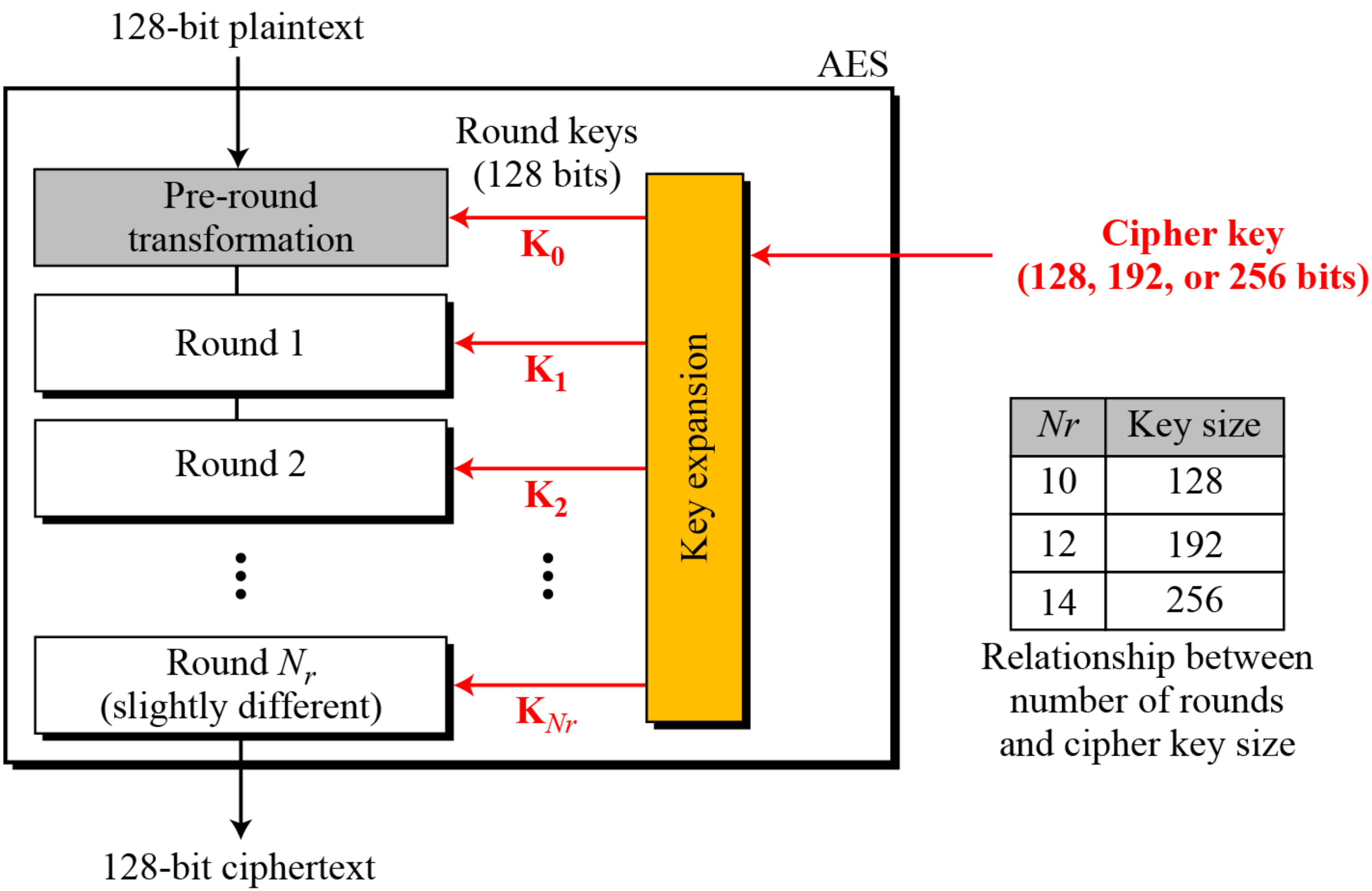
AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits. It uses 10, 12, or 14 rounds. The key size, which can be 128, 192, or 256 bits, depends on the number of rounds.

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.

7.1.3 Continue

Figure 7.1 *General design of AES encryption cipher*

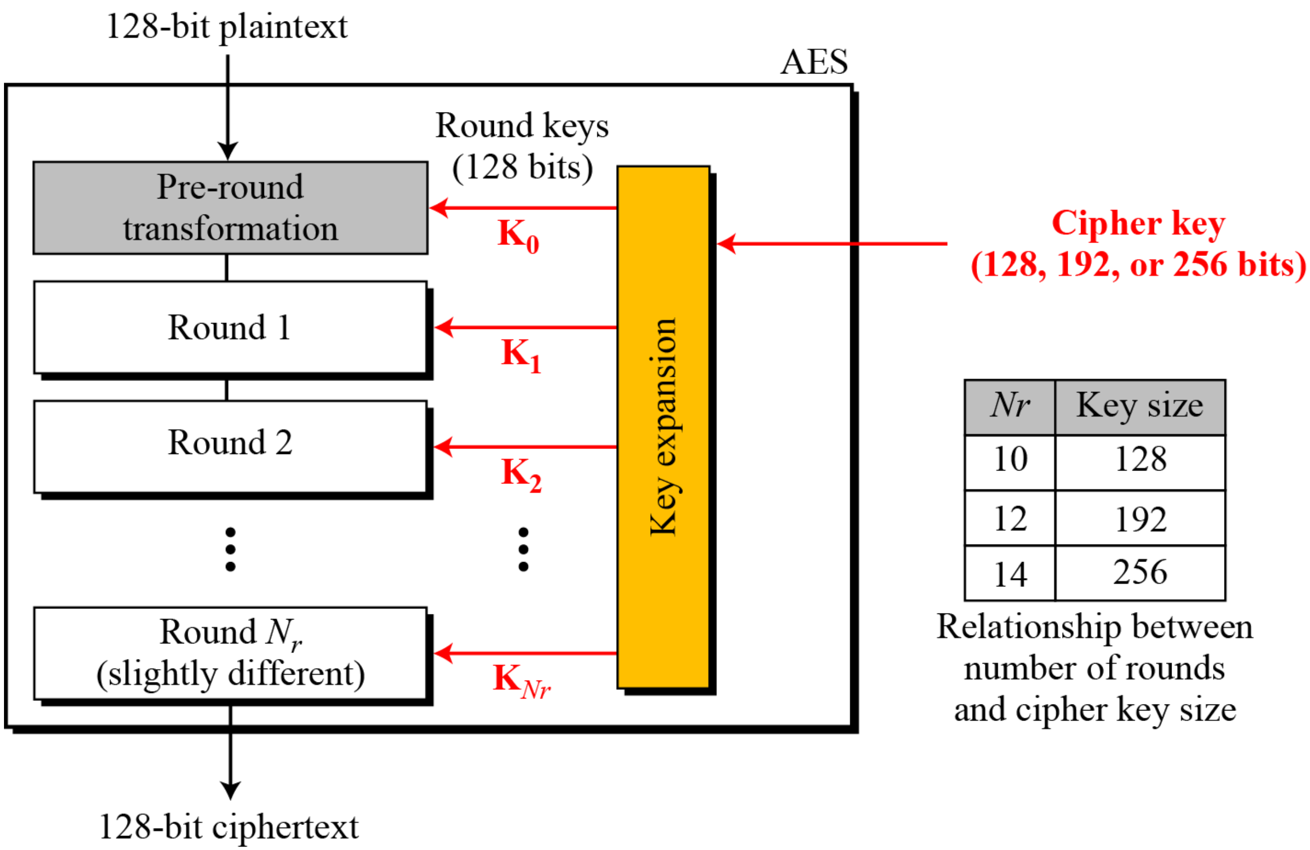


Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

7.1.3 Continue

- *AES encryption algorithm is called Cipher*
- *The Decryption algorithm called Inverse Cipher is similar but round keys are applied in reverse order*

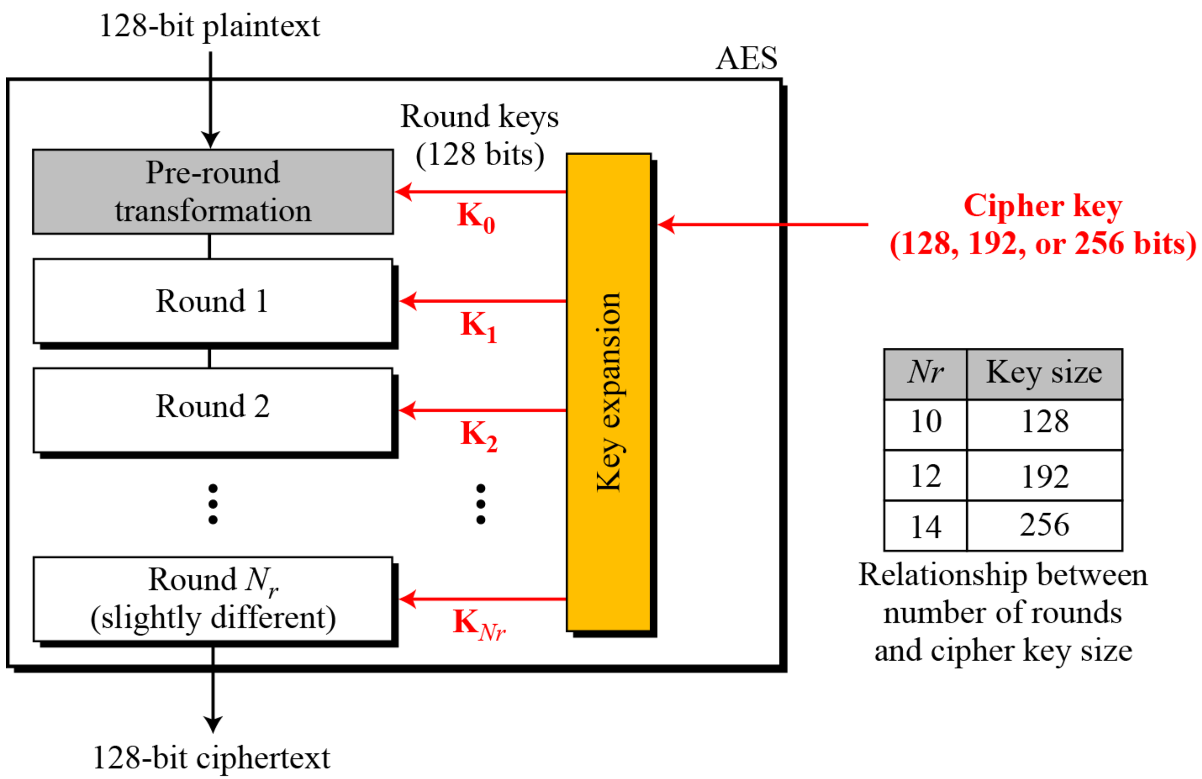


Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

7.1.3 Continue

- *Round Keys created by the Key expansion algorithm are always 128 bits , the same size as the plaintext or ciphertext*
- *The number of round keys generated by the key expansion algorithm is always one more than the number of rounds*
- *Number of Round Keys=Nr+1*
- *Round Keys are K_0,K_1,K_2,……………….K_{Nr}*

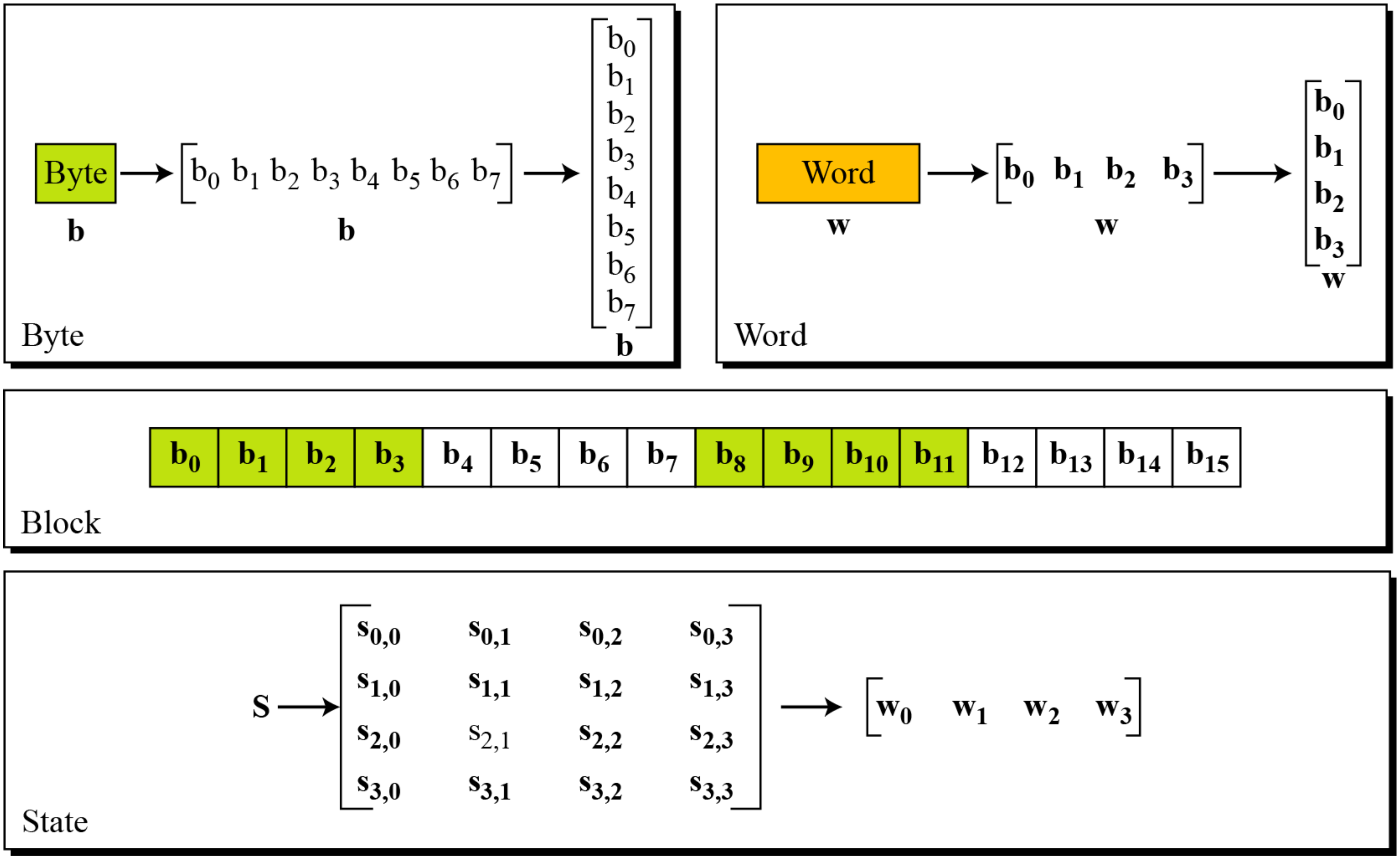


Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

7.1.4 Data Units.

Figure 7.2 ***Data units used in AES***



7.1.4 Data Units.

Data units used in AES

AES uses 5 units of measurement to refer to data:

- ***Bits-Smallest and atomic unit***

Other units can be expressed in terms of smaller ones

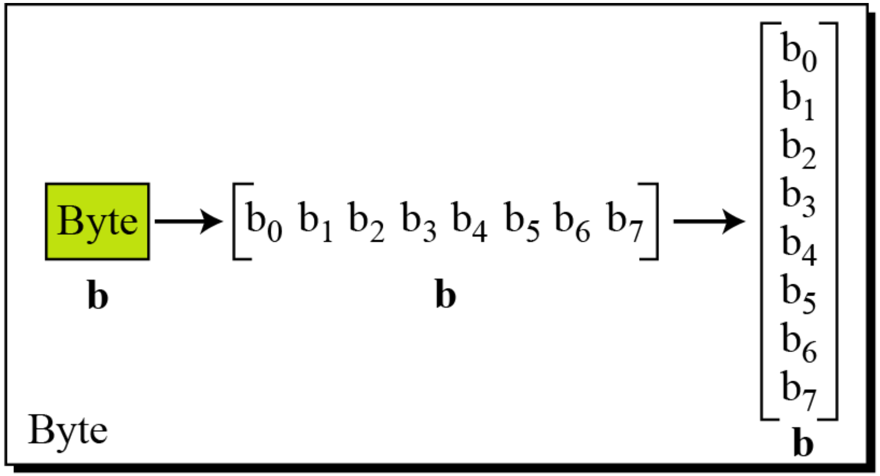
- ***Bytes***
- ***Words***
- ***Blocks***
- ***State***

7.1.4 Data Units.

Data units used in AES

Bytes-

- ***Group of 8 bits,***
- ***Row Matrix (1X8) or***
- ***Column Matrix (8X1),***
- ***Row Matrix –bits are inserted from into the matrix left to right,***
- ***Column Matrix-bits are inserted into the matrix from top to bottom***
- ***Represented with lowercase bold letter “b”***

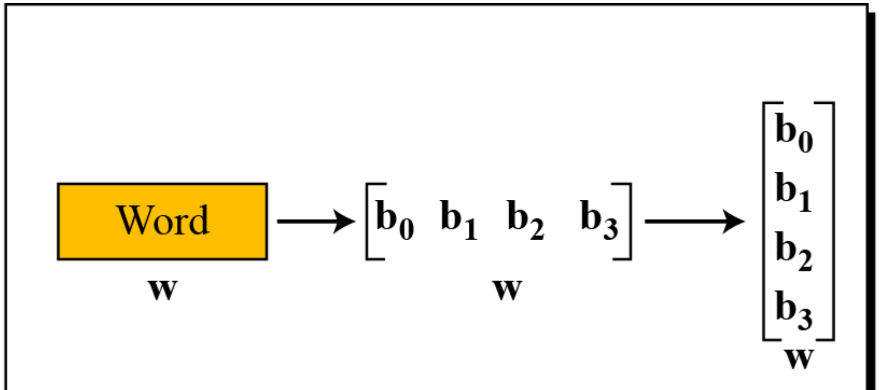


7.1.4 Data Units.

Data units used in AES

Words

- *Group of 32 bits*
- *Row Matrix of 4 Bytes*
- *Column Matrix of 4 Bytes*
- *Row Matrix-Bytes are inserted from into the matrix left to right,*
- *Column Matrix-Bytes are inserted into the matrix from top to bottom*



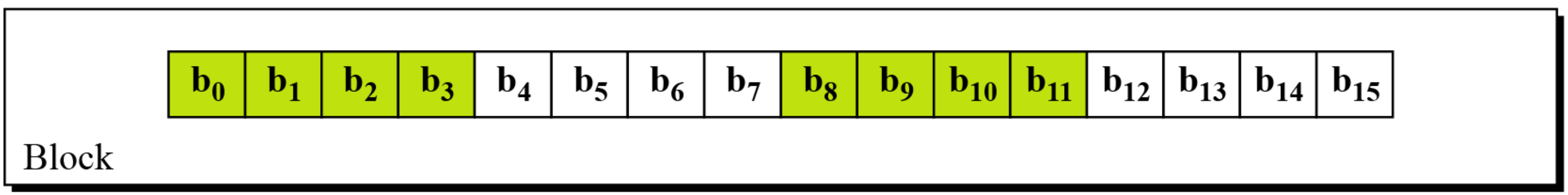
Word

7.1.4 Data Units.

Data units used in AES

Block

- ***Group of 128 bits***
- ***Row Matrix of 16 bytes***
- ***AES encrypts and decrypts data block***



7.1.4 Data Units.

Data units used in AES

State

- ***AES uses several rounds***
- ***Each Round is made of several stages***
- ***Data block is transformed from one stage to other***
- ***At the beginning and end of the cipher, AES uses the term data block***
- ***Before and after each stage, the data block is referred to as state***
- ***Represented as Bold letter S***
- ***Bold Letter T=Temporary State***

7.1.4 Data Units.

Data units used in AES

State

- ***Like Blocks, States are made of 16 bytes***
- ***Normally treated as matrix of 4X4 bytes***
- ***Each element of a state is referred to as $S_{r,c}$ where***
- ***r(0 to3) defines the row and***
- ***c (0 to 3) defines the column***
- ***Occasionally, a state is treated as a row matrix (1X4) of words where words are column matrix***

$$\mathbf{S} \longrightarrow \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \longrightarrow \begin{bmatrix} w_0 & w_1 & w_2 & w_3 \end{bmatrix}$$

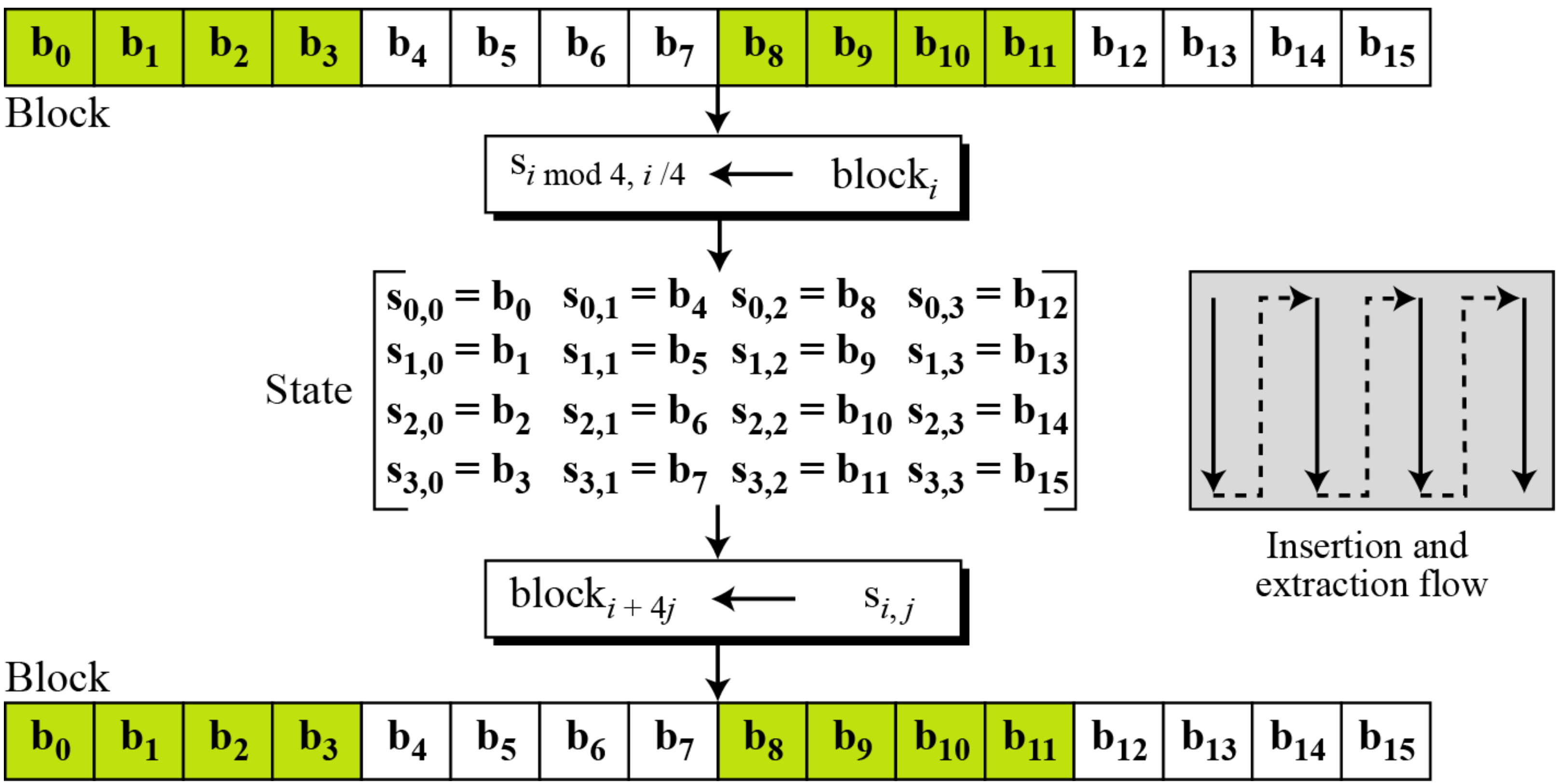
State

7.1.4 Continue

Figure 7.3 *Block-to-state and state-to-block transformation*

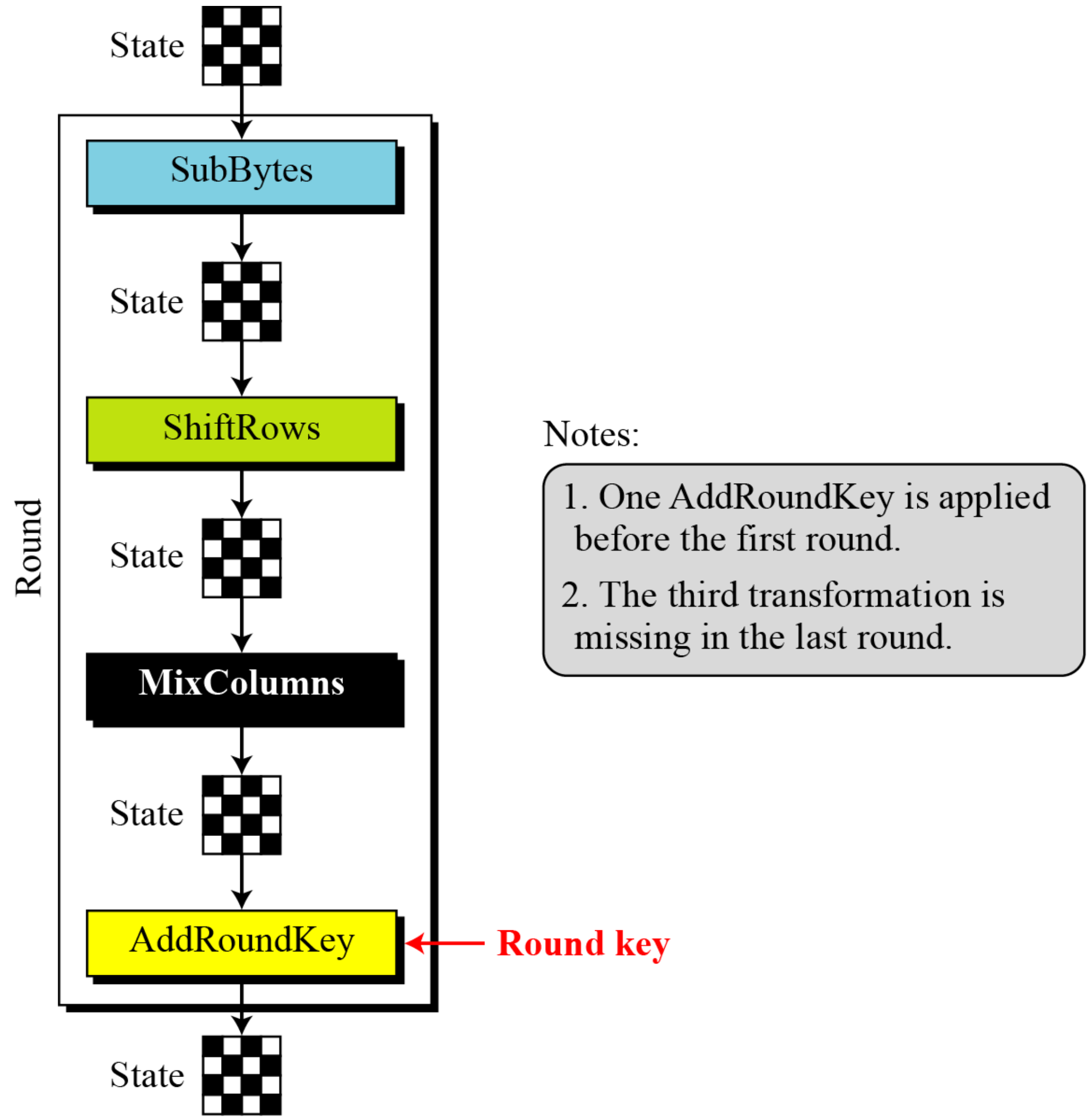
At the beginning of the cipher, the bytes are inserted into a state, column by column and in each column from top to bottom

At the end of cipher, the bytes in data block are extracted in the same way



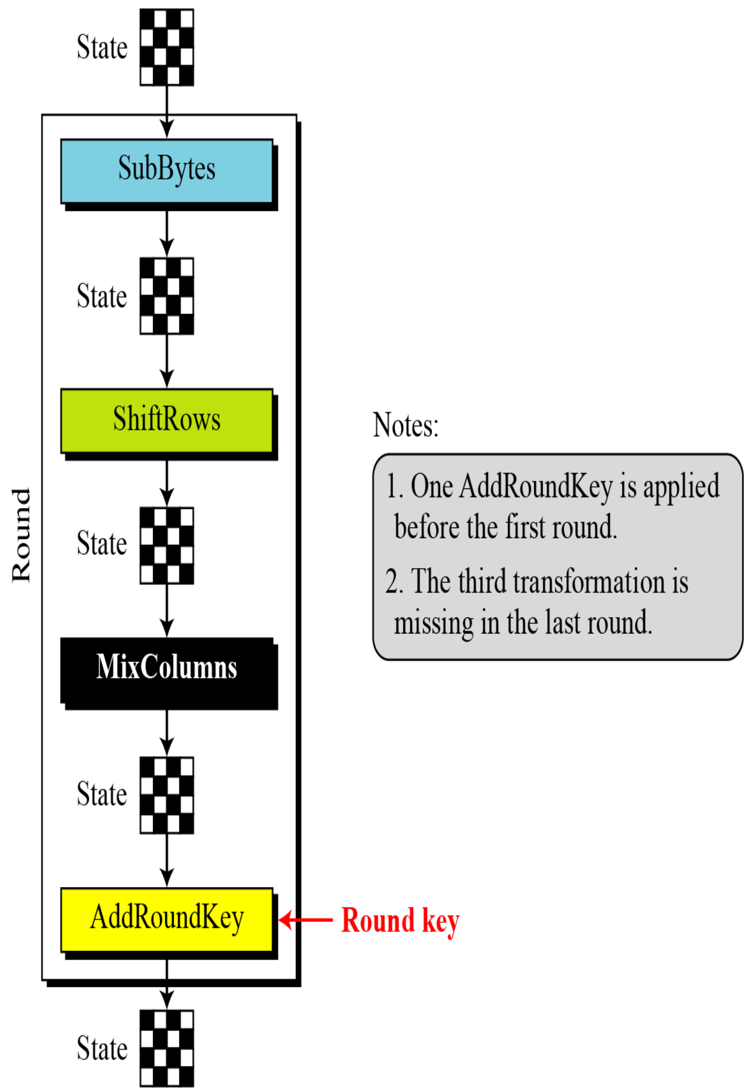
7.1.5 Structure of Each Round

Figure 7.5 ***Structure of each round at the encryption site***



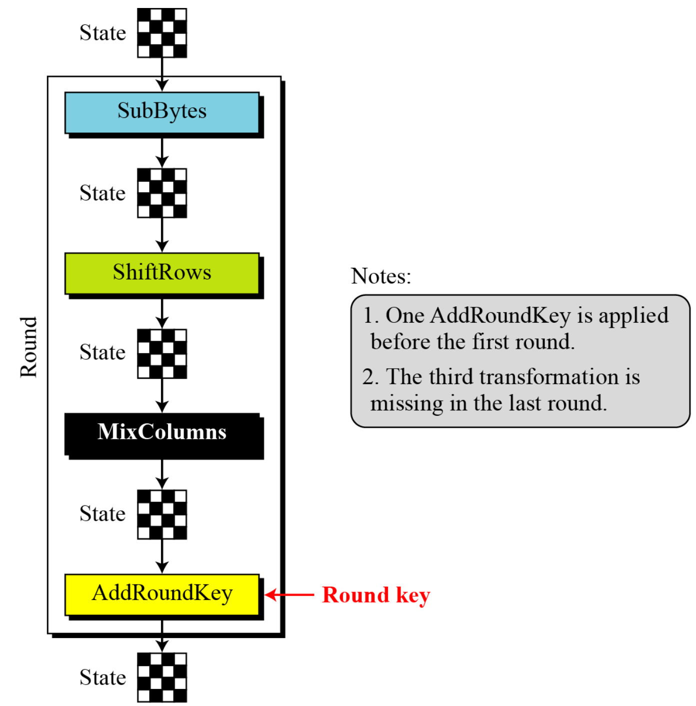
7.1.5 Structure of Each Round

- ***Structure of each round at the encryption site***
- ***Each round except the last uses 4 transformations that are invertible***
- ***The last round has only three transformations***
- ***Each transformation takes a state and creates another state to be used for the next transformation or the next round***



7.1.5 Structure of Each Round

- ***The pre-round section uses only one transformation(AddRoundKey)***
- ***In the last round, MixColumns transformation is missing***
- ***At the Decryption site, The inverse transformations are used –***
  - ***InvSubByte,***
  - ***InvShiftRows,***
  - ***InvMixColumns***
  - ***AddRoundKey(self invertible)***



7-2 TRANSFORMATIONS

To provide security, AES uses four types of transformations: substitution, permutation, mixing, and key-adding.

Topics discussed in this section:

7.2.1 Substitution
7.2.2 Permutation
7.2.3 Mixing
7.2.4 Key Adding

Substitution

- Like DES , AES also uses Substitution
- The Mechanism is different
- First Substitution is done for every byte
- Only one table is used for transformation of every byte, i.e., if two bytes are same, transformation is also same.
- Transformation is defined by a lookup table or mathematical calculation

7.2.1 Substitution

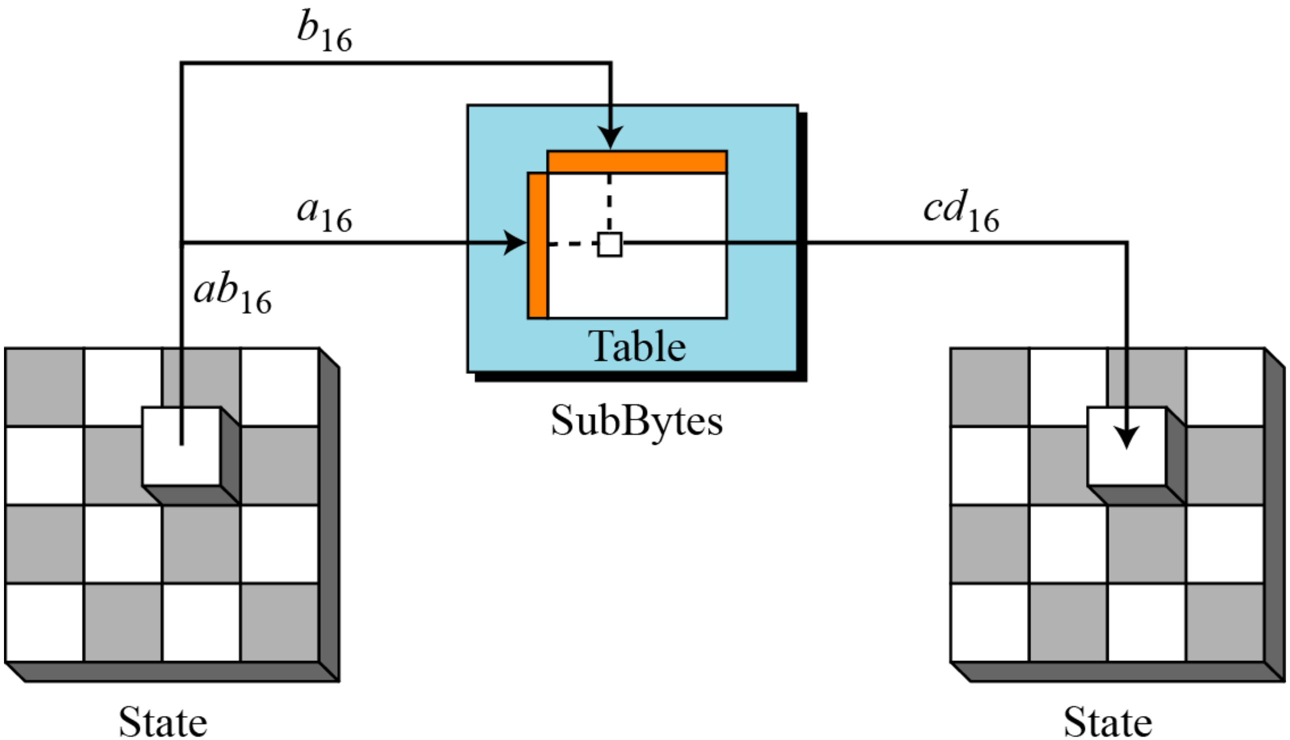
AES uses two invertible transformations.

SubBytes

- ***The first transformation, SubBytes, is used at the encryption site.***
- ***To substitute a byte, we interpret the byte as two hexadecimal digits.***
- ***Left digit defines row while right digit defines column in substitution table.***
- ***The two Hexadecimal digits at the junction of the row and the column are the new byte***

7.2.1 Continue

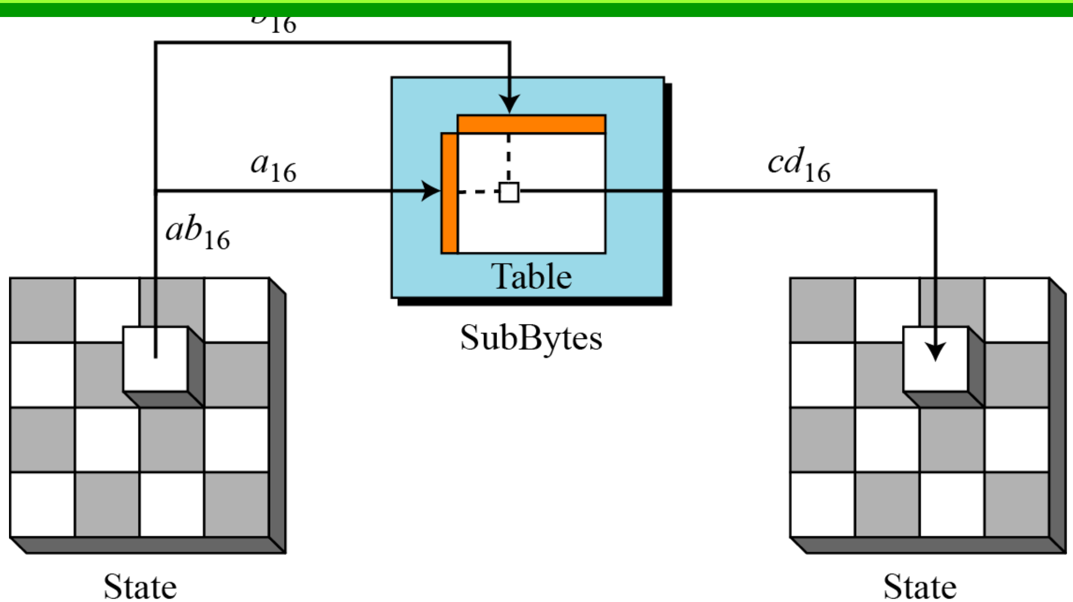
Figure 7.6 ***SubBytes transformation***



7.2.1 SubBytes transformation

- *State is treated as 4X4 matrix of bytes*
- *Transformation is done one byte at a time*
- *The content of each byte is changed but the arrangement of the bytes in the matrix remains the same*
- *Each byte is transformed independently*

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



7.2.1 Continue

- ***The Substitution Table (S Box) for SubBytes transformation***
- ***Provides confusion effect***
- ***Two bytes $5A_{16}$ and $5B_{16}$ which differ only in one bit are transformed to BE_{16} and 39_{16} which differ in four bits***

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

7.2.1 Continue

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

7.2.1 Continue

InvSubBytes

- ***Inverse of SubBytes***

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

7.2.1 Continue

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

Example 7.2

Figure 7.7 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.

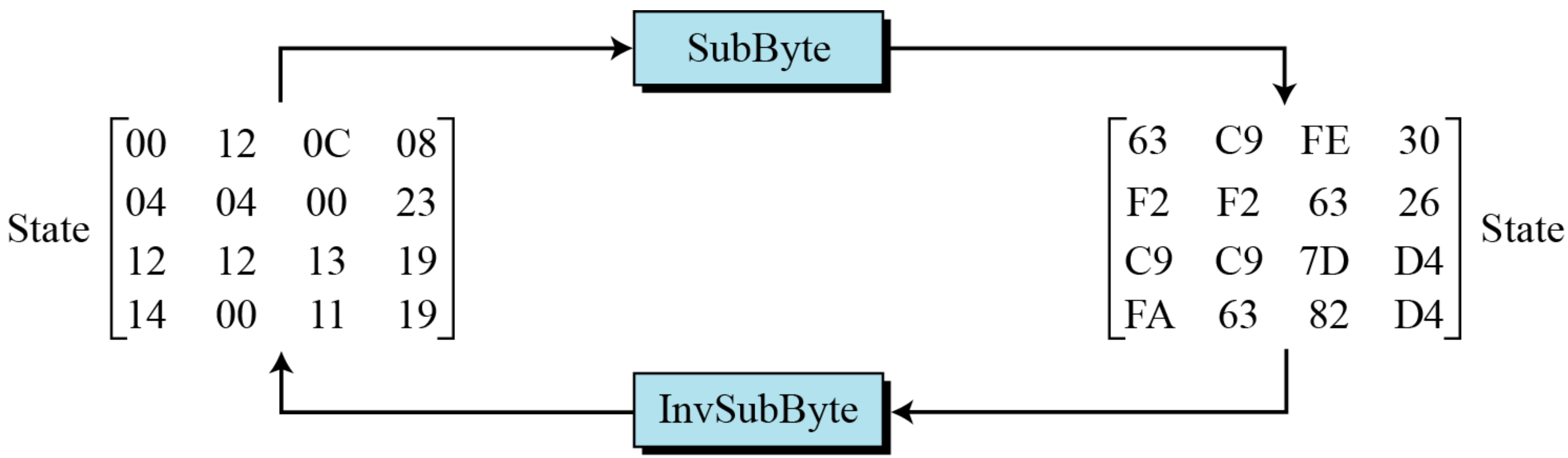


Figure 7.7 *SubBytes transformation for Example 7.2*

Example 7.2

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

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

SubByte

$$\text{State}\ \begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix} \qquad \begin{bmatrix} 63 & C9 & FE & 30 \\ F2 & F2 & 63 & 26 \\ C9 & C9 & 7D & D4 \\ FA & 63 & 82 & D4 \end{bmatrix}\ \text{State}$$

InvSubByte

Figure 7.7 ***SubBytes transformation for Example 7.2***

7.2.1 Continue

Example 7.2

Table 7.2 *InvSubBytes transformation table*

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
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

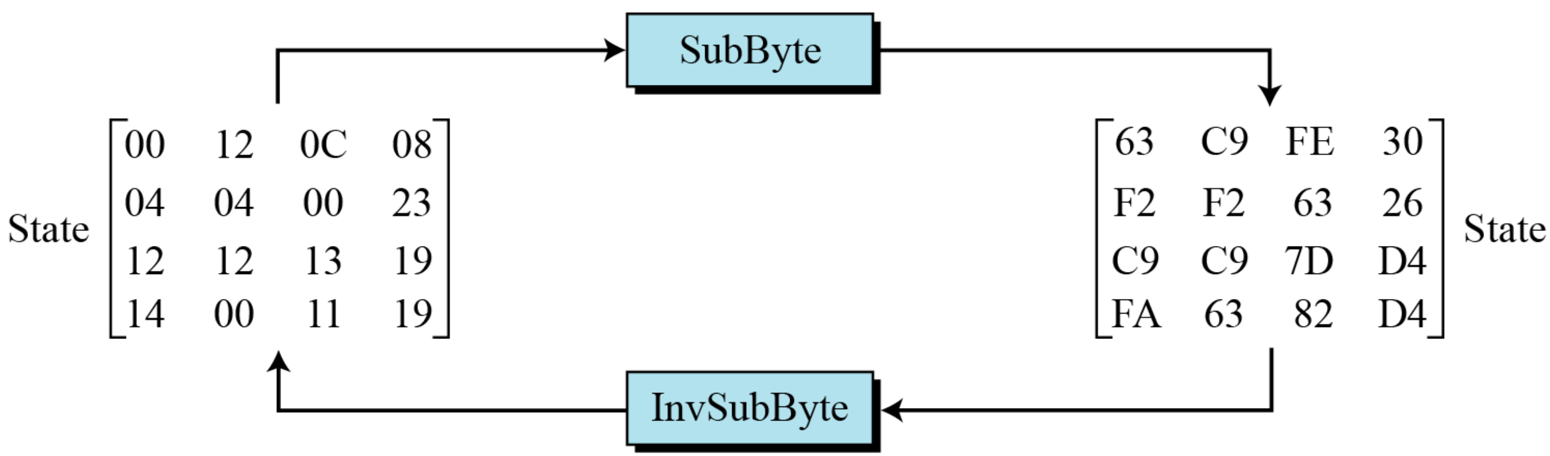


Figure 7.7 ***SubBytes transformation for Example 7.2***

7.2.2 Permutation

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

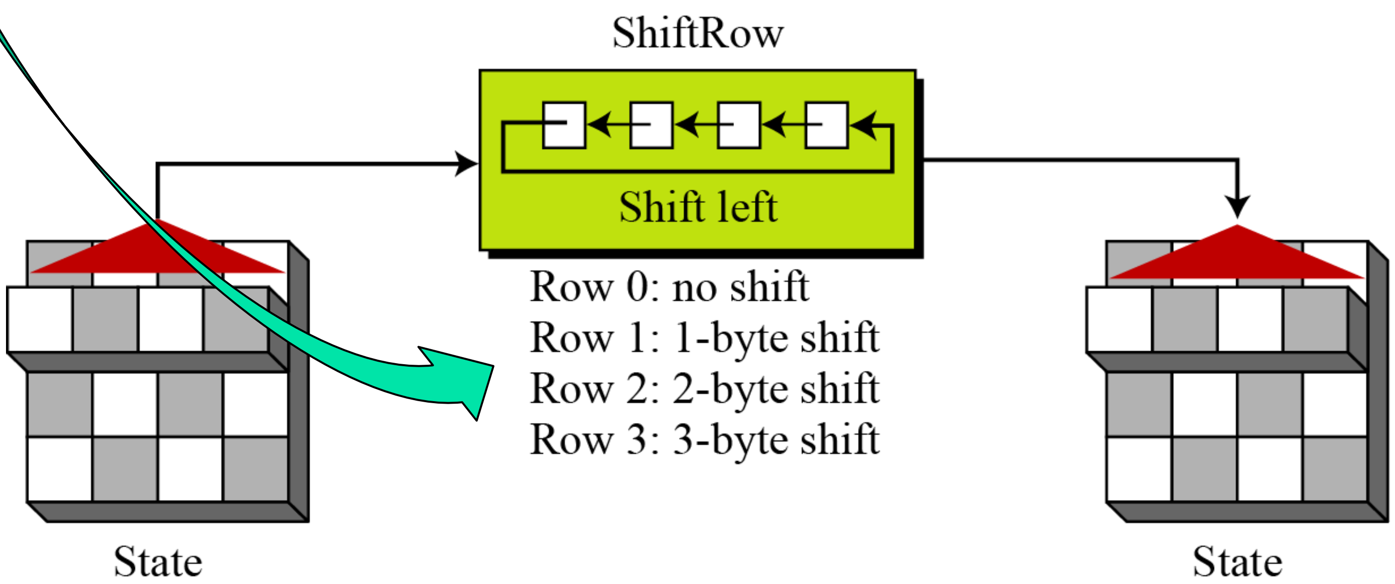
- *In DES, Permutation is done at the bit level,*
- *Shifting transformation in AES is done at the byte level*

- *The order of the bits in the byte is not changed*

ShiftRows

- *In the encryption, the transformation is called ShiftRows.*
- *Shifting is to the left*
- *The no of shifts depends on the row number(0,1,2,3) of the state matrix*
- *Shift row transformation operates one row at a time*

Figure 7.9 *ShiftRows transformation*



7.2.2 Continue

InvShiftRows

- *In the decryption, the transformation is called InvShiftRows and the shifting is to the right.*

- *The number of shifts is the same as the row number (0,1,2 and 3) of the state matrix*

- *ShiftRows and InvShiftRows transformations are inverses of each other*

7.2.2 Continue

Algorithm for ShiftRows-

- ***Function called shift row that shifts byte in a single row***
- ***This function is called three times***
- ***Function copies the row into a temporary row matrix t and then shift row***

Algorithm 7.2 *Pseudocode for ShiftRows transformation*

```
ShiftRows (S)
{
     for (r = 1 to 3)
          shiftrow (s_r, r)                    // s_r is the rth row
}
```

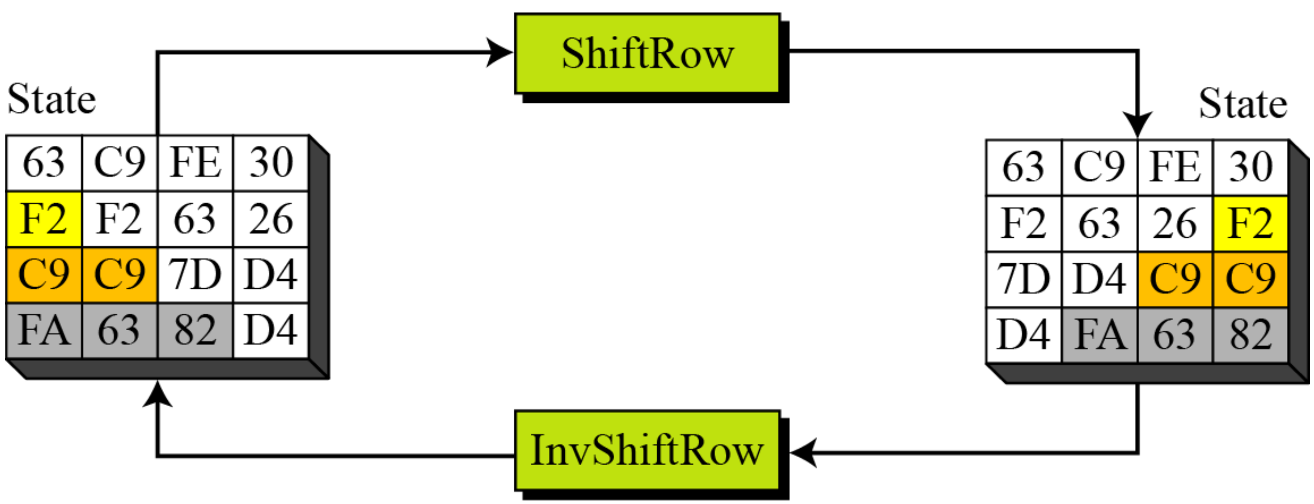
```
shiftrow (row, n)                         // n is the number of bytes to be shifted
{
   CopyRow (row, t)                        // t is a temporary row
   for (c = 0 to 3)
         row_((c − n) mod 4) ← t_c
}
```

7.2.2 Continue

Example 7.4

Figure 7.10 shows how a state is transformed using ShiftRows transformation. The figure also shows that InvShiftRows transformation creates the original state.

Figure 7.10 *ShiftRows transformation in Example 7.4*



7.2.3 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.

Figure 7.11 *Mixing bytes using matrix multiplication*

$$\begin{matrix} a\mathbf{x} + b\mathbf{y} + c\mathbf{z} + d\mathbf{t} \\ e\mathbf{x} + f\mathbf{y} + g\mathbf{z} + h\mathbf{t} \\ i\mathbf{x} + j\mathbf{y} + k\mathbf{z} + l\mathbf{t} \\ m\mathbf{x} + n\mathbf{y} + o\mathbf{z} + p\mathbf{t} \end{matrix} \rightarrow \begin{bmatrix} \square \\ \square \\ \square \\ \square \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{t} \end{bmatrix}$$

New matrix   **Constant matrix**   Old matrix

7.2.3 Mixing

- Takes 4 bytes at a time, combining them to recreate four new bytes
- Each new byte is different, even if all 4 bytes are the same
- Multiplies each byte with a different constant and mixes them

Figure 7.11 *Mixing bytes using matrix multiplication*

$$
\begin{bmatrix} a\mathbf{x} + b\mathbf{y} + c\mathbf{z} + d\mathbf{t} \\ e\mathbf{x} + f\mathbf{y} + g\mathbf{z} + h\mathbf{t} \\ i\mathbf{x} + j\mathbf{y} + k\mathbf{z} + l\mathbf{t} \\ m\mathbf{x} + n\mathbf{y} + o\mathbf{z} + p\mathbf{t} \end{bmatrix}
=
\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}
\times
\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{t} \end{bmatrix}
$$

New matrix | **Constant matrix** | Old matrix

7.2.3 Continue

- **AES defines a transformation called Mix columns to acheive this goal**
- **There is also an inverse transformation called InvMixColumns**

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

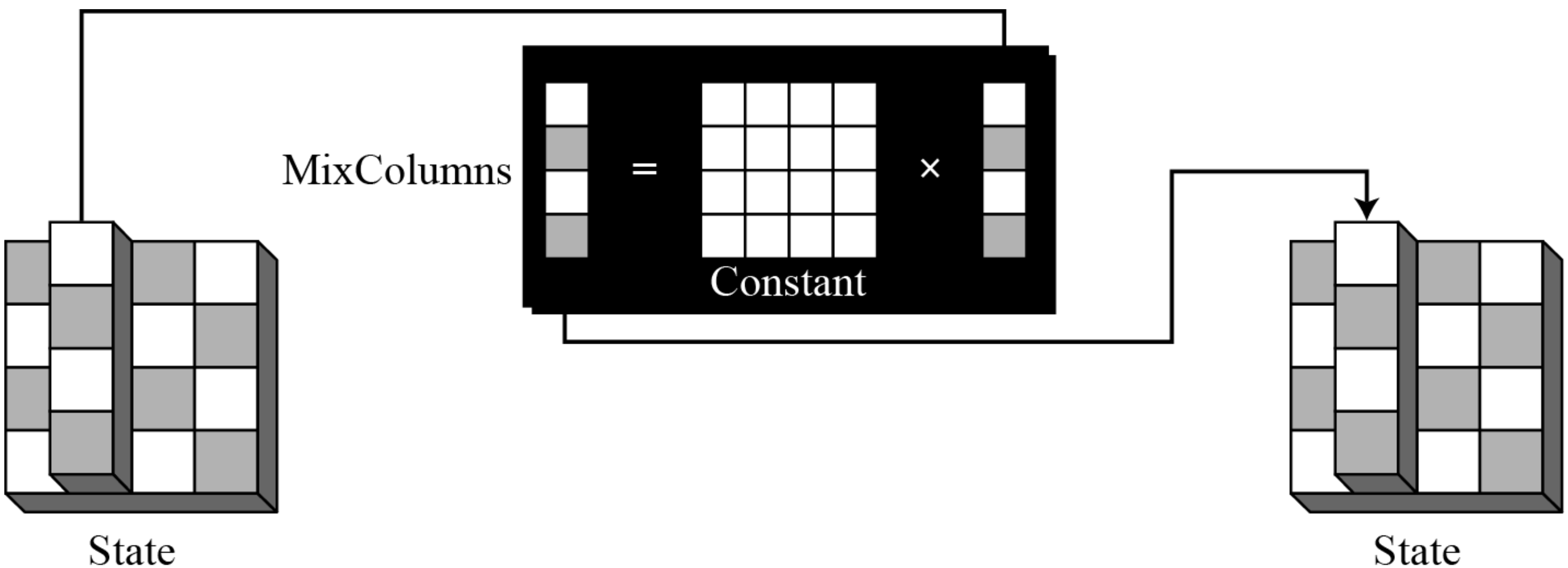
Constant matrices used by MixColumns and InvMixColumns

MixColumns

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

Matrix multiplication of a state column by a constant square matrix

Figure 7.13 *MixColumns transformation*



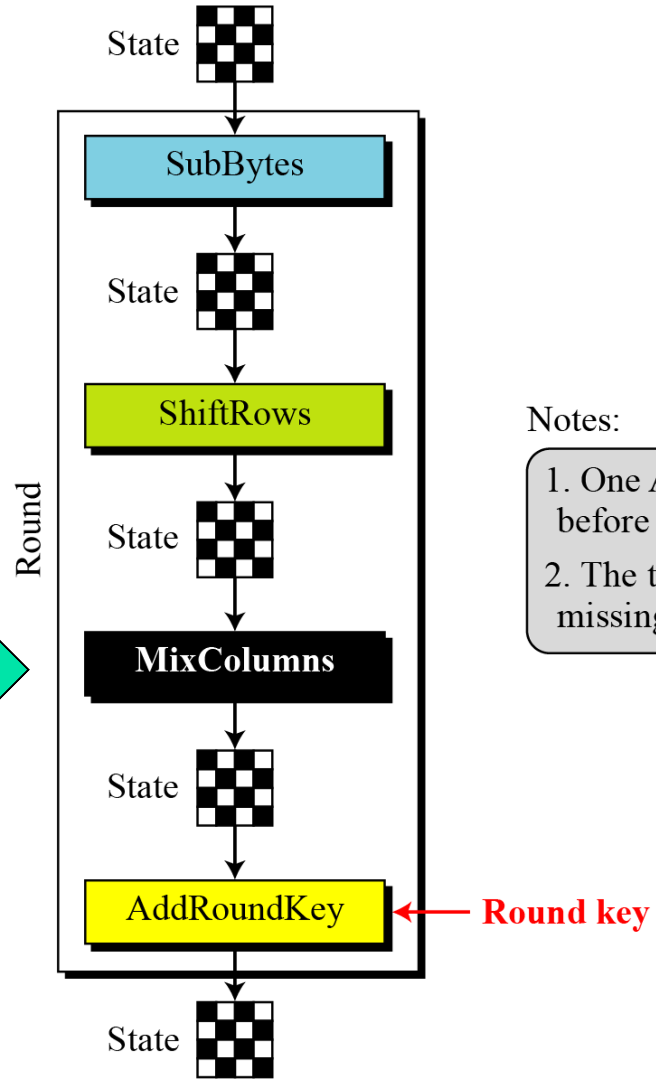
7.2.3 Continue

InvMixColumns

The InvMixColumns transformation is basically the same as the MixColumns transformation.

Note

The two column matrices are inverses of each other,
Thus, The MixColumns and InvMixColumns transformations are inverses of each other.



Notes:

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

7.2.3 Continue

Example 7.5

Figure 7.14 shows how a state is transformed using the MixColumns transformation. The figure also shows that the InvMixColumns transformation creates the original one.

Figure 7.14 *The MixColumns transformation in Example 7.5*

$$
\begin{tikzcd}[column sep=huge]
\text{State}\ \begin{bmatrix} 63 & C9 & FE & 30 \\ F2 & 63 & 26 & F2 \\ 7D & D4 & C9 & C9 \\ D4 & FA & 63 & 82 \end{bmatrix} \arrow[r, shift left=2, "\textbf{MixColumn}"] & \begin{bmatrix} 62 & 02 & 27 & 26 \\ CF & 92 & 91 & 0D \\ 0C & 0C & F4 & D6 \\ 99 & 18 & 30 & 74 \end{bmatrix}\ \text{State} \arrow[l, shift left=2, "\textbf{InvMixColumn}"]
\end{tikzcd}
$$

7.2.4 Key Adding

- ***Most important transformation***

- ***Its the one that includes the cipher key***

- ***If the cipher key is not added to the state at each round, it is very easy for the adversary to find the plaintext, given the ciphertext.***

- ***The cipher key is the only secret between Alice and Bob***

7.2.4 Key Adding

AddRoundKey

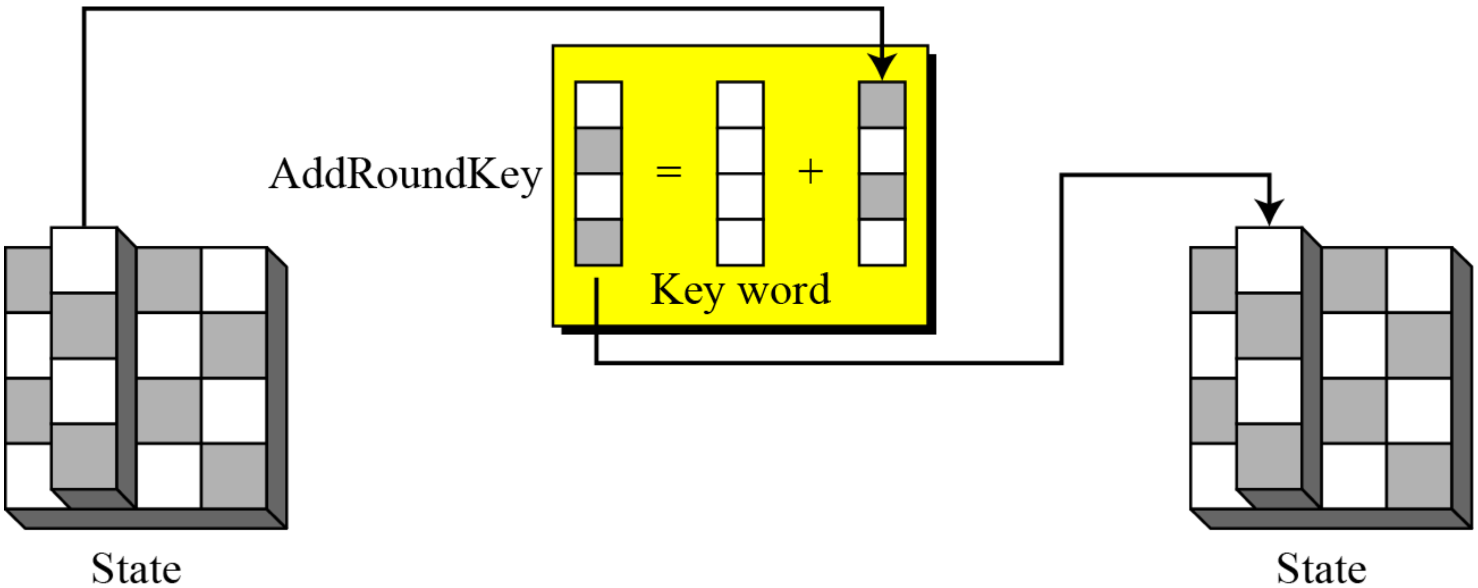
- *Each Round key is 128 bits long*
- *Treated as Four 32 bit words*
- *For adding the key to the state , each word is considered as a column matrix*
- *AddRoundKey proceeds one column at a time.*
- *AddRoundKey adds a round key word with each state column matrix;*
- *The operation in AddRoundKey is matrix addition.*

Note

The AddRoundKey transformation is the inverse of itself.

7.2.4 Continue

Figure 7.15 ***AddRoundKey transformation***



7.2.4 Continue

Algorithm-

- ***XORing of each column of the state with the corresponding keyword***
- ***Cipherkey is expanded into a set of keywords***
- ***S_c and w_{round} are 4X1 column matrices***
- ***XORing of two column matrices , each of 4 bytes***

Algorithm 7.4 *Pseudocode for AddRoundKey transformation*

AddRoundKey (S)
{
    for (c = 0 to 3)
        $\mathbf{s}_c \leftarrow \mathbf{s}_c \oplus \mathbf{w}_{\text{round} + 4c}$
}

7-3 KEY EXPANSION

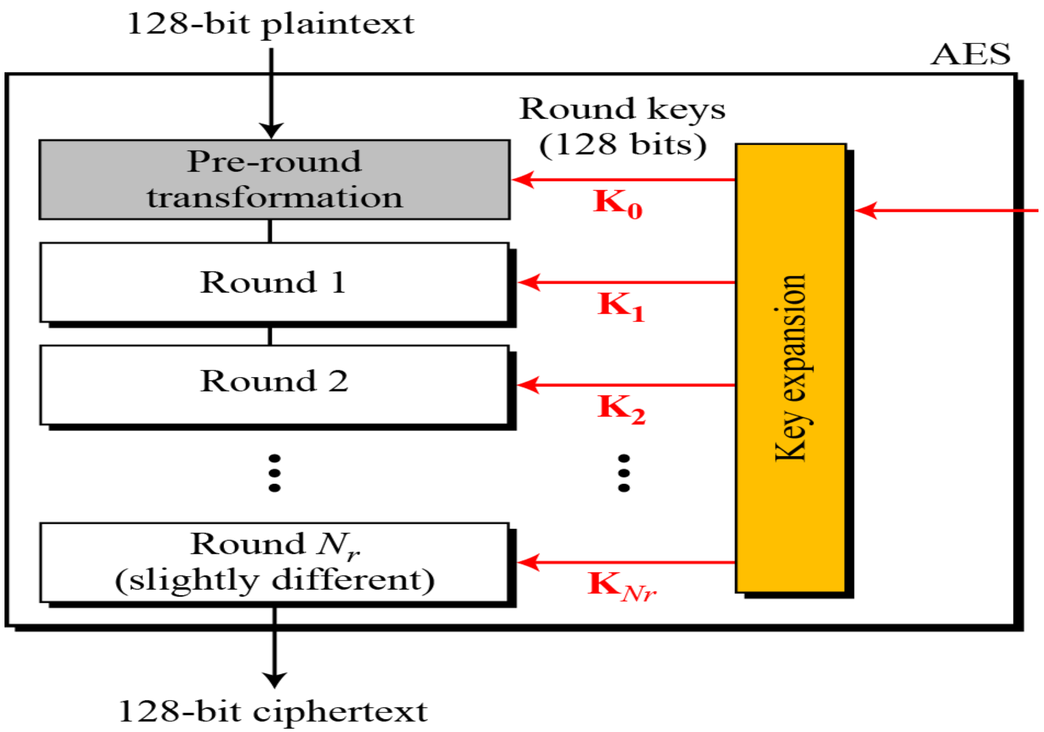
To create round keys for each round, AES uses a key-expansion process. If the number of rounds is N_r , the key-expansion routine creates N_r + 1 128-bit round keys from one single 128-bit cipher key.

Topics discussed in this section:

7-3 KEY EXPANSION

The 1st Round key is used for pre-round transformation

The remaining round keys are used for the last transformation(AddRoundKey) at the end of each round



7-3 KEY EXPANSION

The Key expansion routine creates round keys word by word, where word is an array of four bytes
The routine creates 4X(Nr+1) words called w0,w1,w2…………...w4

In AES-128 version with 10 rounds=>11X4=44 words
In AES-192 version with 12 rounds=>13X4=52 words
In AES-256 version with 14 rounds=>15X4=60 words

7-3 Continued

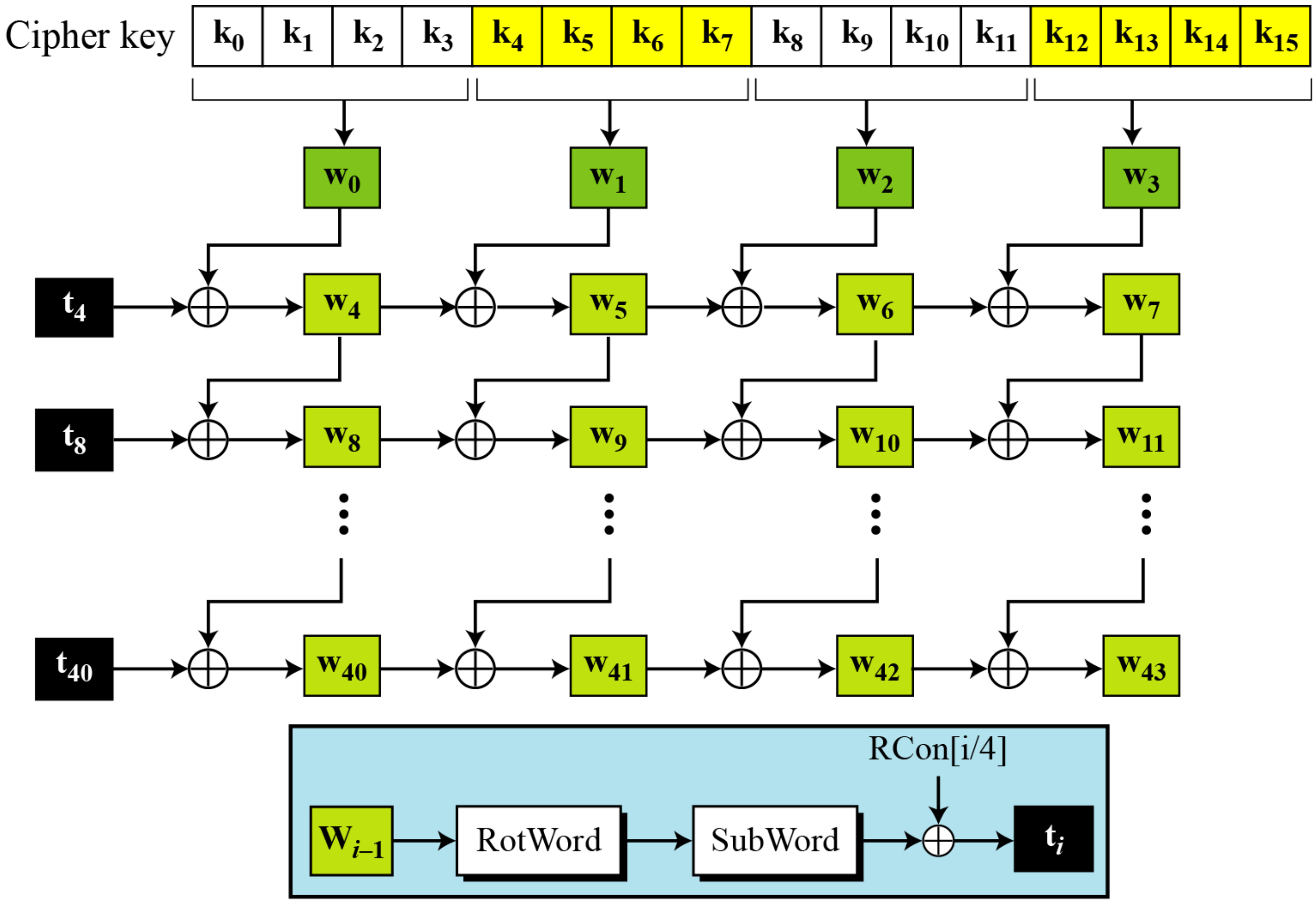
Table 7.3 *Words for each round*

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	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

W40 W41 W42 W43

In AES-128 version with 10 rounds=>11X4=44 words

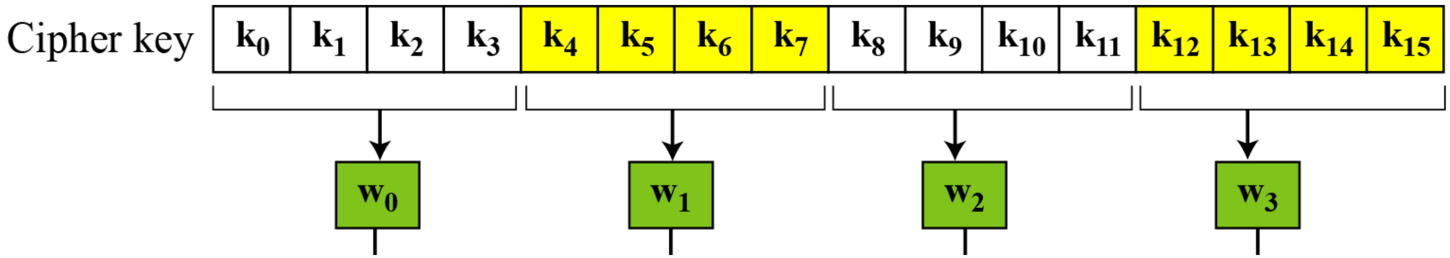
Figure 7.16 *Key expansion in AES*



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

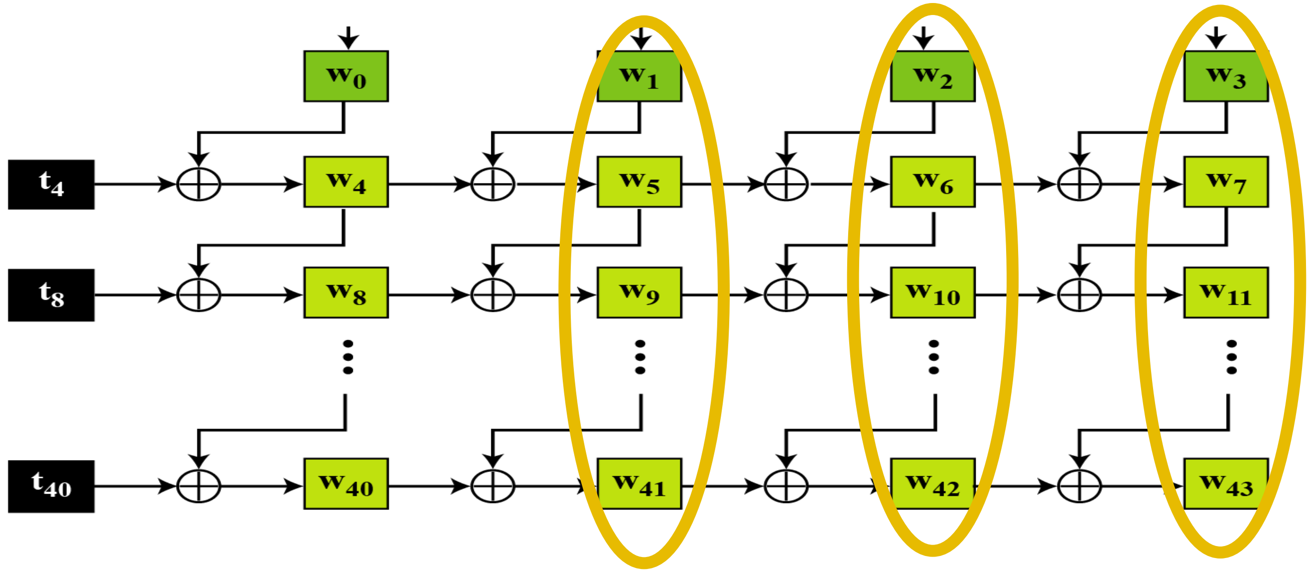
7.3.1 Key Expansion in AES-128

Figure 7.16 *Key expansion in AES*



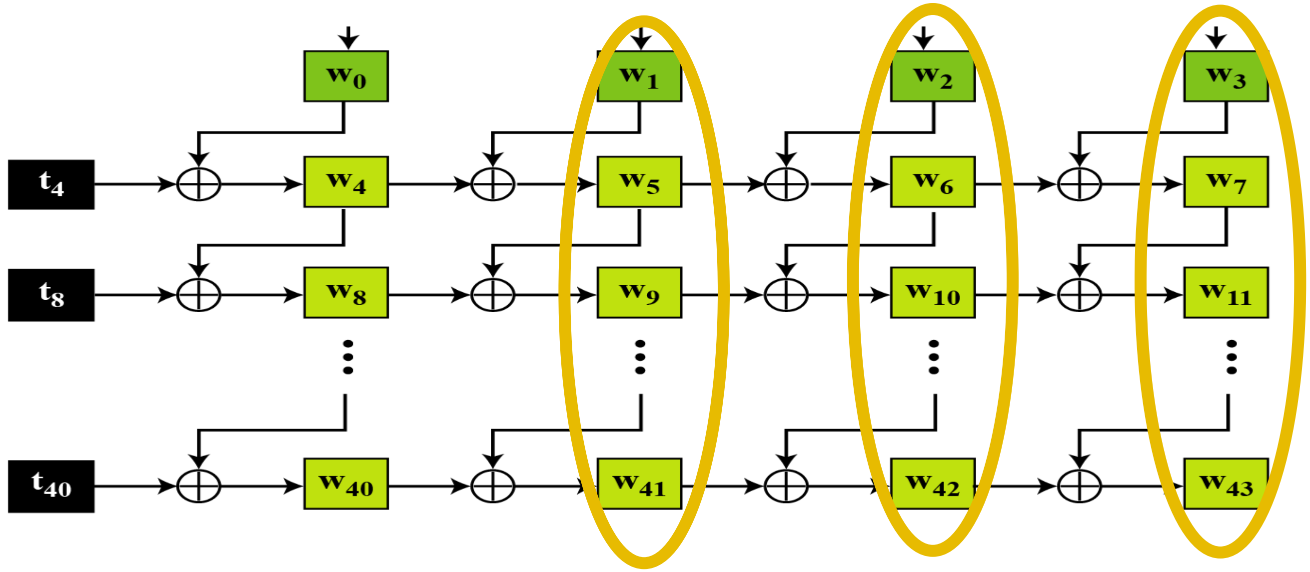
- *The First 4 words are made from the cipher key*
- *Cipher key=array of 16 bytes (ko to k15)*
- *ko,k1,k2,k3=wo*
- *k4,k5,k6,k7=w1*
- *k8,k9,k10,k11=w3*
- *k12,k13,k14,k15=w4*

7.3.1 Key Expansion in AES-128



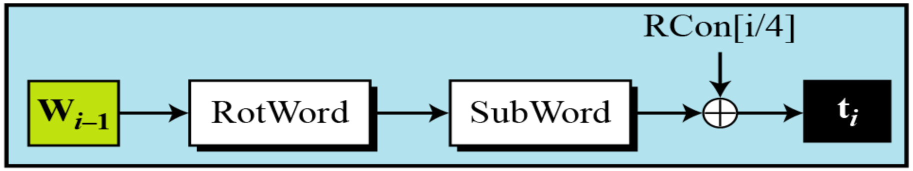
- ***Remaining words are calculated as follows:-***
- ***For i=4 to 43***
  - ***If imod4!=0, wi=wi-1 EXOR wi-4,***
  - ***Each word is made from one at the left and one at the top***

7.3.1 Key Expansion in AES-128



- *If imod4=0,*
  - *wi=t EXOR wi-4,*
  - *t=temporary word=Result of applying two Routines Subword and Rot Word on wi-1 and EXORing the result with a round constants RCon*

7.3.1 Key Expansion in AES-128

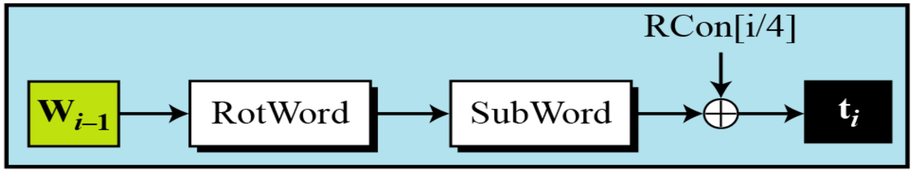


Making of t_i (temporary) words $i = 4\ N_r$.

- ***t=temporary word=Result of applying two Routines Subword and Rot Word on wi-1 and EXORing the result with a round constants Rcon***

- ***t=SubWord(RotWord(wi-1))EXOR RConi/4***

7.3.1 Key Expansion in AES-128



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

- ***RotWord-***
  - ***Rotate Word***
  - ***Takes a word as an array of 4 bytes***
  - ***Shifts each byte to the left with wrapping***
- ***SubWord-***
  - ***Substitute Word***
  - ***Takes each byte in the word and substitute another byte for it***
- ***Round Constants-***
  - ***Rcon=4 byte value, Rightmost three bytes are always Zero***

7.3.1 Continue

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)$_{16}$

- ***Key Expansion can use the above table for Rcon constants***

7.3.1 Continue

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

```
KeyExpansion ([key_0 to key_15], [w_0 to w_43])
{
    for (i = 0 to 3)
        w_i ← key_4i + key_4i+1 + key_4i+2 + key_4i+3

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

7.3.1 Continue

Example 7.6

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} = B822deb8	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

7.3.1 Continue

Example 7.6

- **In each Round, The calculation of the last three words is very simple**
- **For first word, we need to calculate the value of temporary word**

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} = B822deb8	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

7.3.1 Continue

Example 7.7

Each round key in AES depends on the previous round key. The dependency, however, is nonlinear because of SubWord transformation. The addition of the round constants also guarantees that each round key will be different from the previous one.

7.3.1 Continue

Example 7.8

The two sets of round keys can be created from two cipher keys that are different only in one bit.

Cipher Key 1:	12 45 A2 A1 23 31 A4 A3 B2 CC A**A** 34 C2 BB 77 23
Cipher Key 2:	12 45 A2 A1 23 31 A4 A3 B2 CC A**B** 34 C2 BB 77 23

7.3.1 Continue

Example 7.8 *Continue*

There are significant differences between the two corresponding round keys
R=Round
BD=Bit Difference

Table 7.6 *Comparing two sets of round keys*

R.	*Round keys for set 1*	*Round keys for set 2*	*B. D.*
—	1245A2A1 2331A4A3 B2CCAA34 C2BB7723	1245A2A1 2331A4A3 B2CCAB34 C2BB7723	01
1	F9B08484 DA812027 684D8A13 AAF6FD30	F9B08484 DA812027 684D8B13 AAF6FC30	02
2	B9E48028 6365A00F 0B282A1C A1DED72C	B9008028 6381A00F 0BCC2B1C A13AD72C	17
3	A0EAF11A C38F5115 C8A77B09 6979AC25	3D0EF11A 5E8F5115 55437A09 F479AD25	30
4	1E7BCEE3 DDF49FF6 1553E4FF 7C2A48DA	839BCEA5 DD149FB0 8857E5B9 7C2E489C	31
5	EB2999F3 36DD0605 238EE2FA 5FA4AA20	A2C910B5 7FDD8F05 F78A6ABC 8BA42220	34
6	82852E3C B4582839 97D6CAC3 C87260E3	CB5AA788 B487288D 430D4231 C8A96011	56
7	82553FD4 360D17ED A1DBDD2E 69A9BDCD	588A2560 EC0D0DED AF004FDC 67A92FCD	50
8	D12F822D E72295C0 46F948EE 2F50F523	0B9F98E5 E7929508 4892DAD4 2F3BF519	44
9	99C9A438 7EEB31F8 38127916 17428C35	F2794CF0 15EBD9F8 5D79032C 7242F635	51
10	83AD32C8 FD460330 C5547A26 D216F613	E83BDAB0 FDD00348 A0A90064 D2EBF651	52

7.3.1 Continue

Example 7.9

The concept of weak keys, as we discussed for DES in Chapter 6, does not apply to AES. Assume that all bits in the cipher key are 0s. The following shows the words for some rounds:

Pre-round:	00000000	00000000	00000000	00000000
Round 01:	62636363	62636363	62636363	62636363
Round 02:	9B9898C9	F9FBFBAA	9B9898C9	F9FBFBAA
Round 03:	90973450	696CCFFA	F2F45733	0B0FAC99
...	...	...	...	...
Round 10:	B4EF5BCB	3E92E211	23E951CF	6F8F188E

- **The words in the pre-round and the first round are all the same.**
- **In the second round, the first word matches with the third; the second word matches with the fourth.**
- **However, after the second round the pattern disappears; every word is different.**

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, but with few differences

7.3.2 Key Expansion in AES-192 and AES-256

Differences:

- ***In AES-192,***
  - ***The words are generated in groups of six instead of four***
  - ***The Cipher key creates the first six words (wo to w5)***
  - ***If imod6!=0,wi=wi-1+wi-6, else wi=t+wi-6***
- ***In AES-256,***
  - ***The words are generated in groups of eight instead of four***
  - ***The Cipher key creates the first eight words (w0 to w7)***
  - ***If imod8!=0, wi=wi-1+wi-8, else wi=t+wi-8***
  - ***If imod4=0 but imod8!=0, then wi=SubWord(wi-1)+wi-8***

7.3.3 Key-Expansion Analysis

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

- *Two different Cipher keys, no matter how similar to each other, produce two expansions that differ in atleast a few rounds*

- *Each bit of cipher key is diffused into several rounds. Changing a single bit in the cipher key, will change some bits in several rounds*

- *No serious weak keys in AES*

7.3.3 Key-Expansion Analysis

- ***Key expansion can be easily implemented on all platforms***
- ***Even if Eve knows only part of the cipher key or the values of the words in some round keys, she still needs to find the rest of the cipher key before she can find all round keys. Its because of the Non-Linearity produced by SubWord transformation in the key expansion process***

7-4 CIPHERS

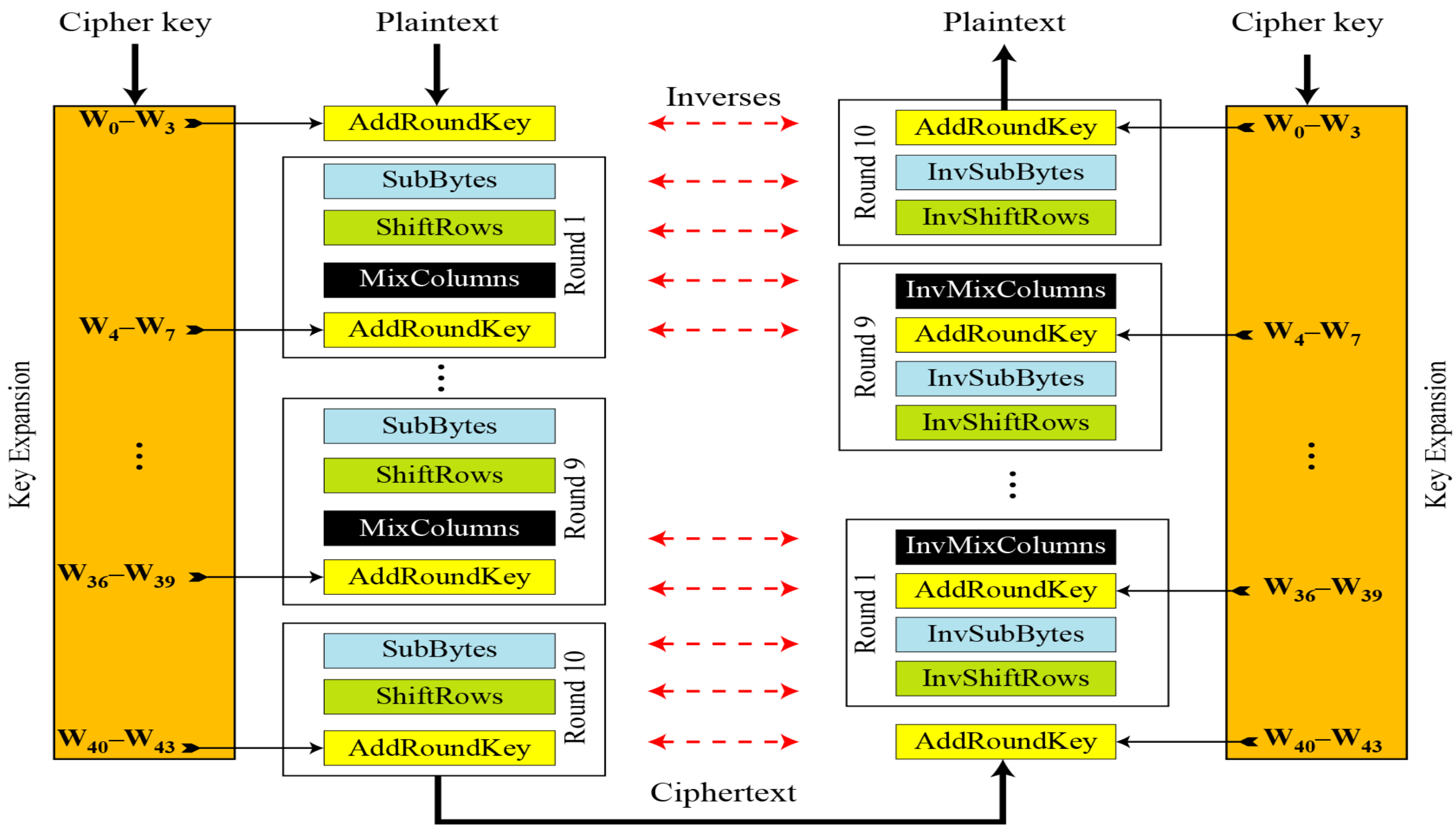
AES uses four types of transformations for encryption and decryption. In the standard, the encryption algorithm is referred to as the cipher and the decryption algorithm as the inverse cipher.

Topics discussed in this section:

7.4.1 Original Design
7.4.2 Alternative Design

7.4.1 Original Design

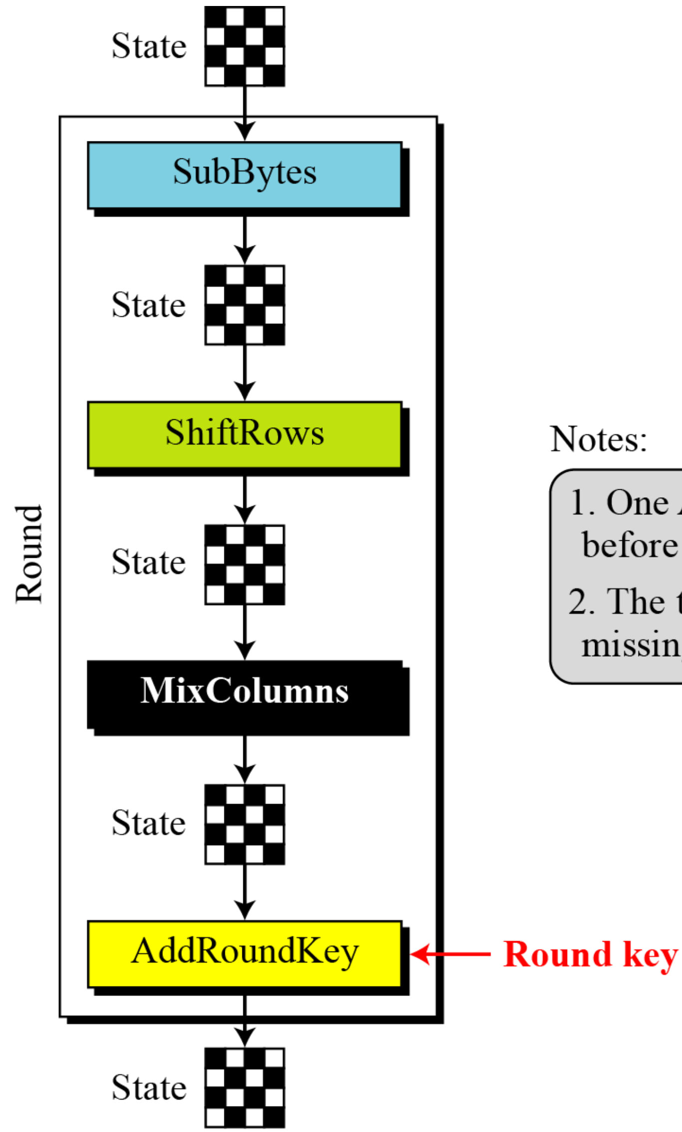
Figure 7.17 *Ciphers and inverse ciphers of the original design*



The order of transformations in each round is not the same in the cipher and reverse cipher

7.1.5 Structure of Each Round

Figure 7.5 ***Structure of each round at the encryption site***



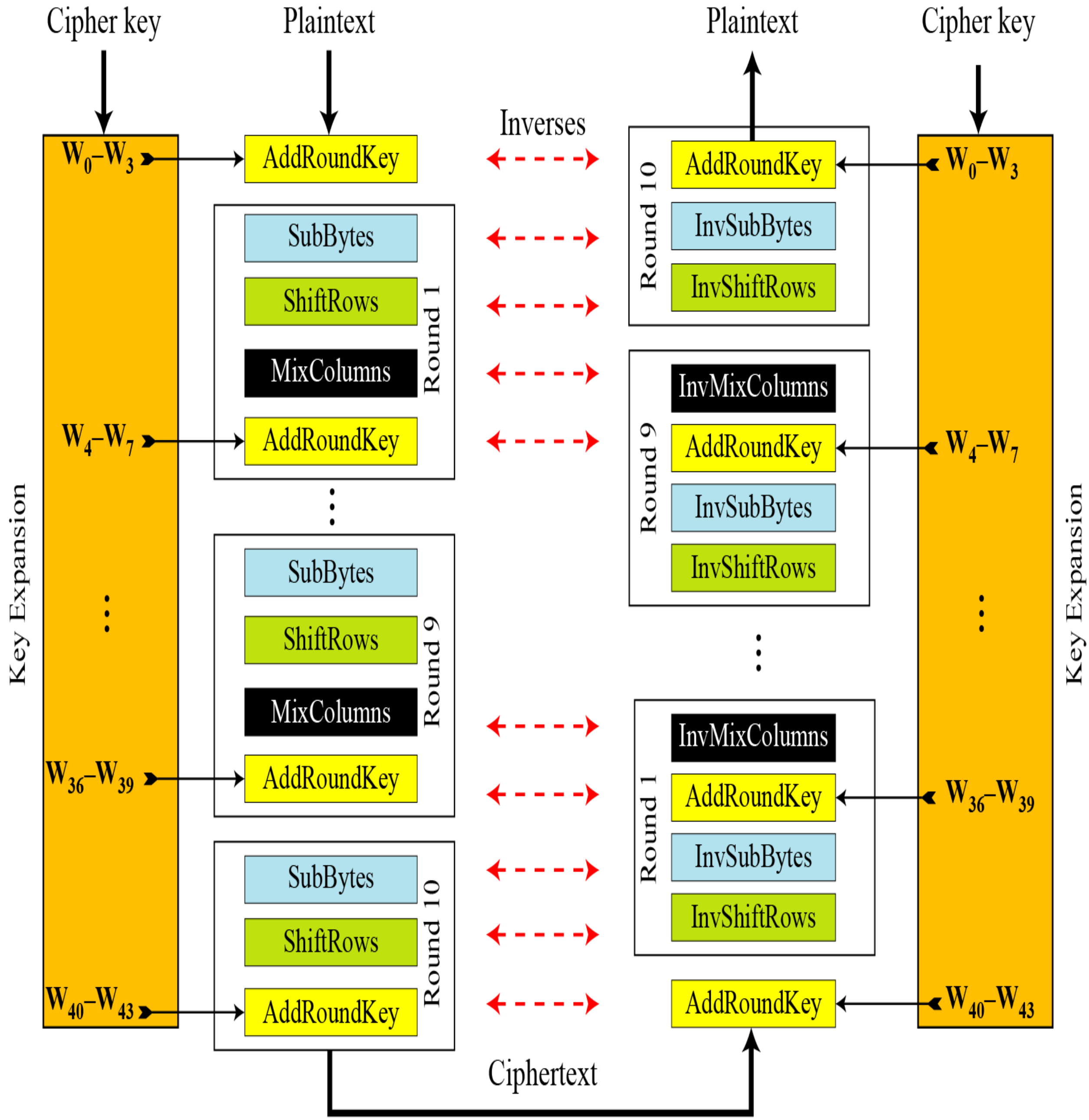
Notes:

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

7.4.1 Original Design

In the Reverse Cipher-

- ***The Order of SubBytes and ShiftRows is changed***
- ***The Order of MixColumns and AddRoundKey is changed***
- ***Decryption Algorithm as a whole is inverse of the encryption algorithm***
- ***Round Keys are used in the reverse order***



Algorithm

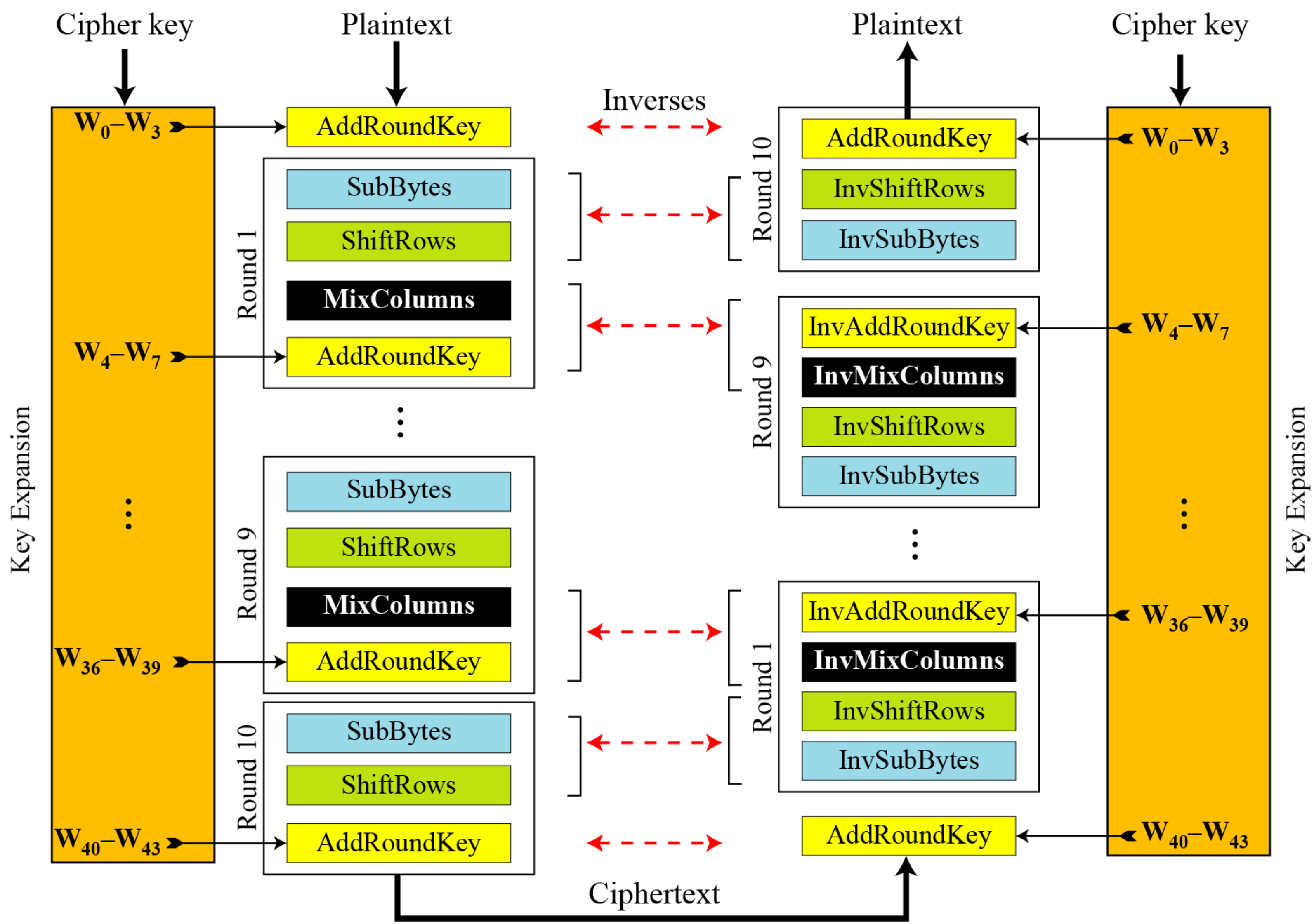
The code for the AES-128 version of this design is shown in Algorithm 7.6.

Algorithm 7.6 *Pseudocode for cipher in the original design*

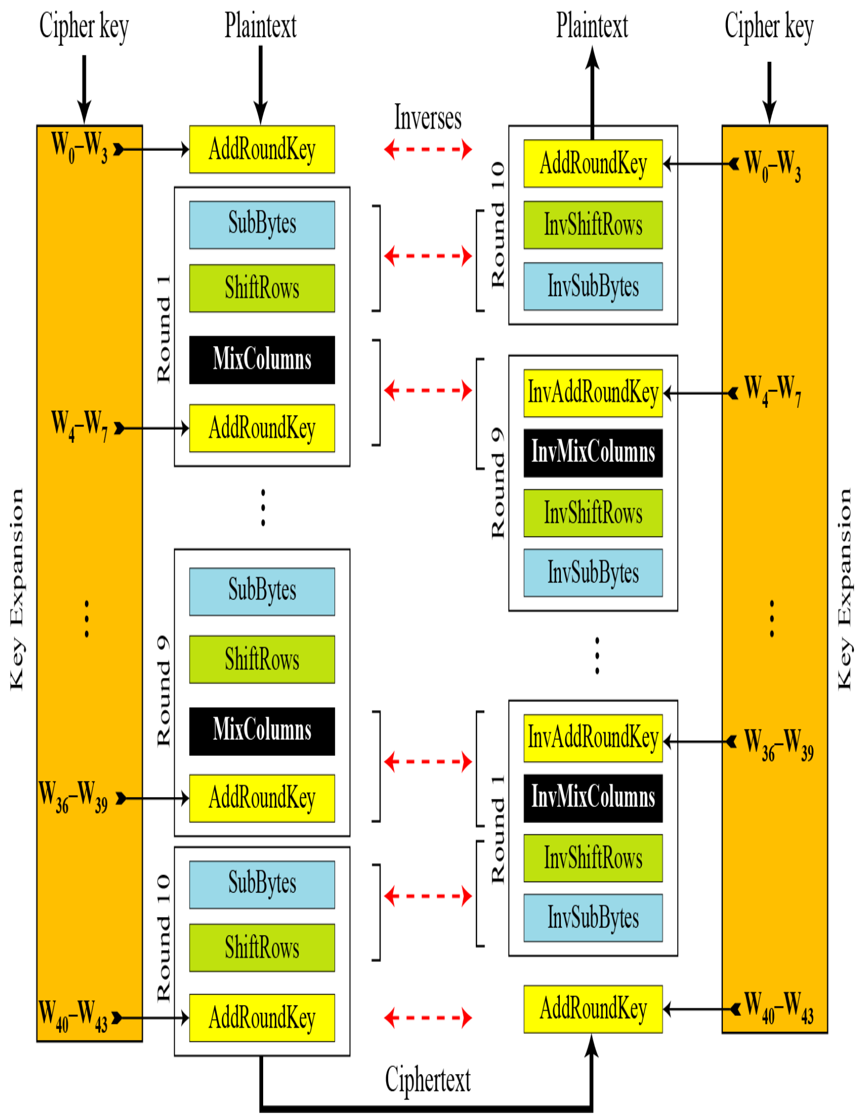
```
Cipher (InBlock [16], OutBlock[16], w[0 … 43])
{
    BlockToState (InBlock, S)

    S ← AddRoundKey (S, w[0…3])
    for (round = 1 to 10)
    {
        S ← SubBytes (S)
        S ← ShiftRows (S)
        if (round ≠ 10)   S ← MixColumns (S)
        S ← AddRoundKey (S, w[4 × round, 4 × round + 3])
    }

    StateToBlock (S, OutBlock);
}
```

7.4.2 Alternate Design

Figure 7.20 *Cipher and reverse cipher in alternate design*

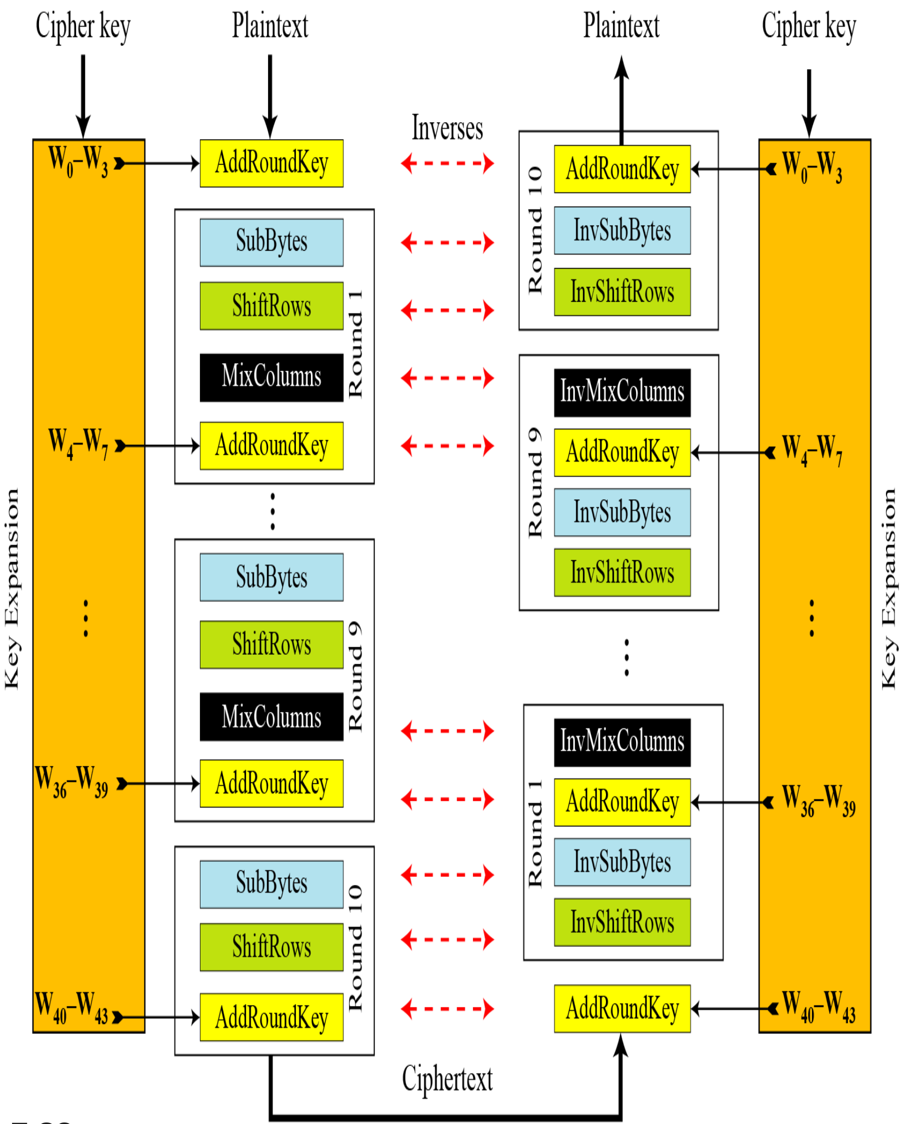


- A different Inverse cipher was developed
- Transformations in the reverse cipher are rearranged to make the order of transformations the same in the cipher and reverse cipher

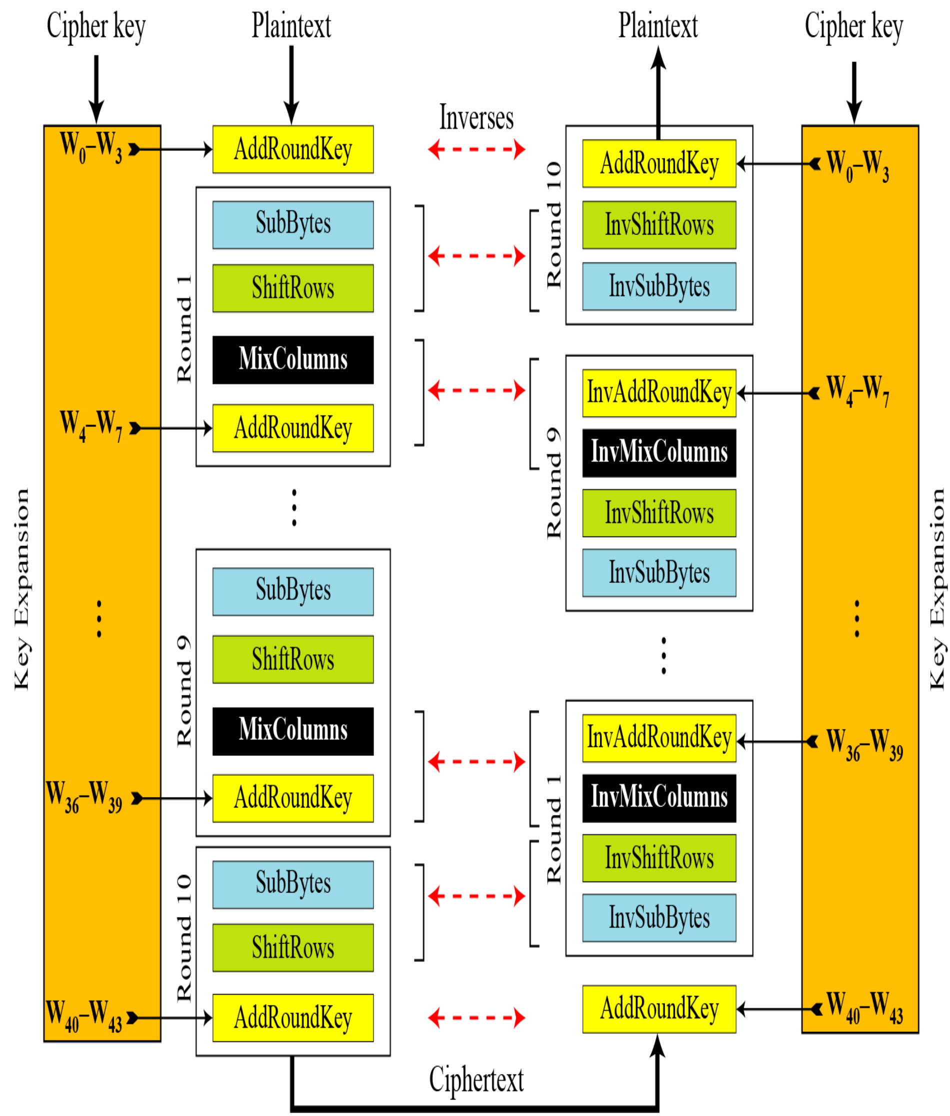


7.4.2 Alternate Design

Original design



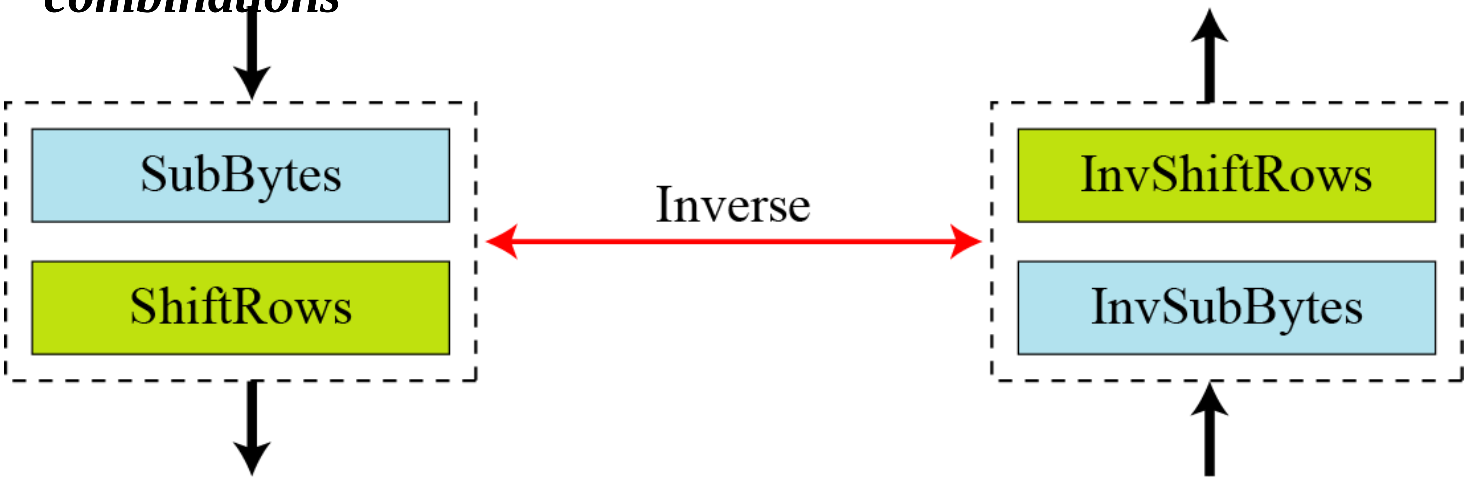
Alternate design



7.4.2 Alternative Design

Invertibility is provided for a pair of transformations and not for each single transformation.

Figure 7.18 ***Invertibility of SubBytes and ShiftRows combinations***

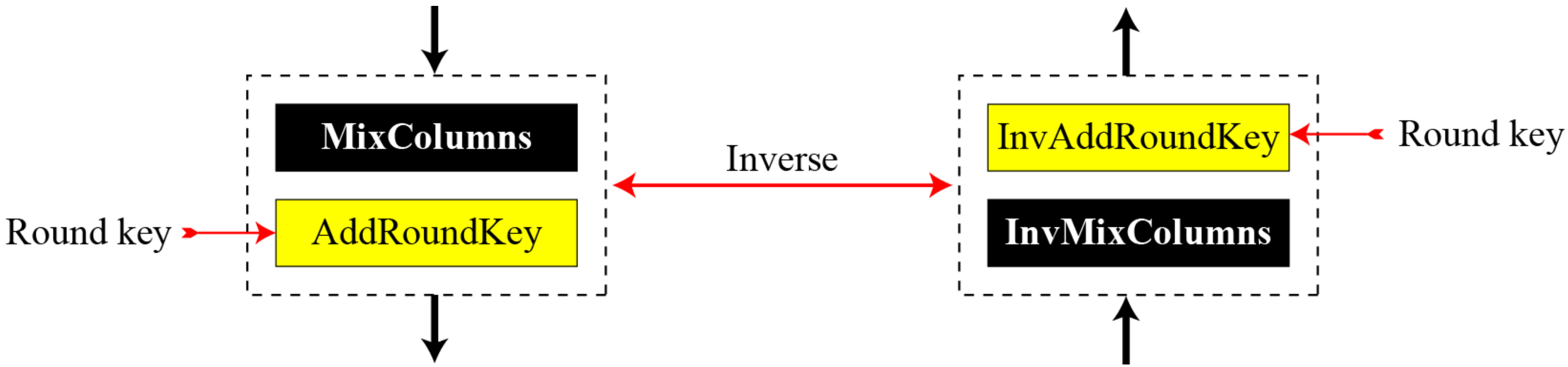


SubBytes change contents without changing order

ShiftRows change order without changing contents

7.4.2 Continue

Figure 7.19 ***Invertibility of MixColumns and AddRoundKey combination***



The pair operation becomes inverses of each other if we multiply the key matrix by inverse of constant matrix used in MixColumns transformation

7-6 ANALYSIS OF AES

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

Topics discussed in this section:

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.

7.6.1 Security

Brute-Force Attack

- ***AES is definitely more secure than DES due to the larger-size key.***
- ***DES had 56 bit cipher key and AES had 128 bit cipher key***
- ***For DES , we need 2^{56} tests to find the key, For AES, we need 2^{128} tests to find the key***
- ***If we break DES in t seconds, we need 2^{72} X t seconds to break AES***
- ***Almost Impossible to break***
- ***AES has 2 other versions with longer cipher keys***
- ***Lack of weak keys is another advantage of AES over DES***

7.6.1 Security

Statistical Attacks

- ***Strong Diffusion and Confusion provided by the combination of SubBytes, ShiftRows and MixColumns transformation removes any frequency pattern in the plain text***

7.6.1 Security

Differential and Linear Attacks

AES was designed after DES, Differential and Linear cryptanalysis attacks were no doubt taken into consideration

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.