Information Security
CS3002

Lecture 5
7th September 2023

Dr. Rana Asif Rehman
Email: r.asif@lhr.nu.edu.pk

Advanced Encryption Standard (AES)

Origins

- Clear a replacement for DES was needed.
  - Have theoretical attacks that can break it.
  - Have demonstrated exhaustive key search attacks.
- US NIST issued call for ciphers in 1997.
- 15 candidates accepted in Jun 98.
- 5 were shortlisted in Aug-99.
- Rijndael was selected as the AES in Oct-2000.
- Issued as FIPS PUB 197 standard in Nov-2001 .

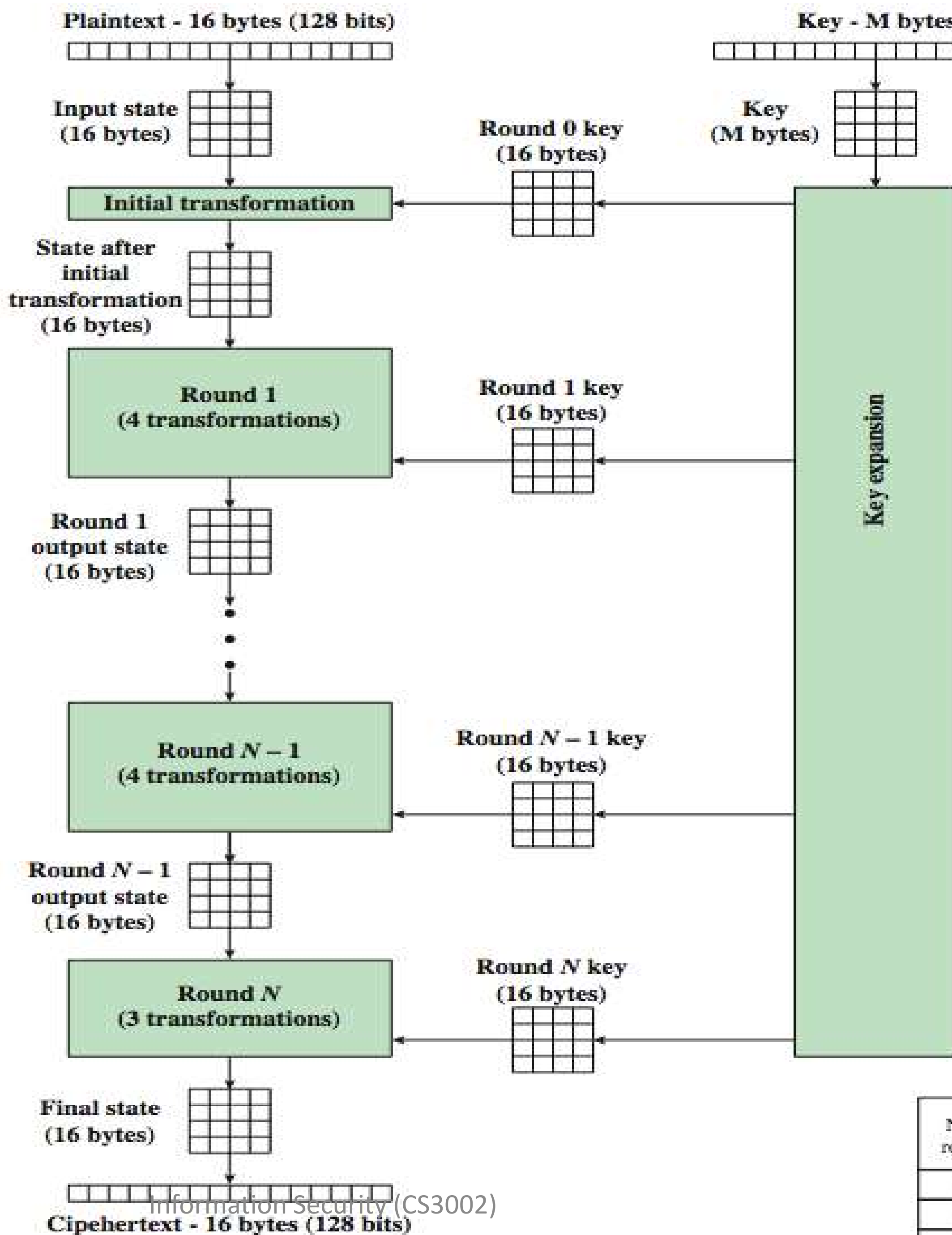
The AES Cipher

- Allows *128, 192, and 256*-bit key sizes .
- Variable block length of *128, 192, or 256* bits. All nine combinations of key/block length possible.
  - A block is the smallest data size the algorithm will encrypt
- Vast speed improvement over DES in both hardware and software implementations
  - with fast XOR & table lookup implementation

The AES Cipher - Rijndael

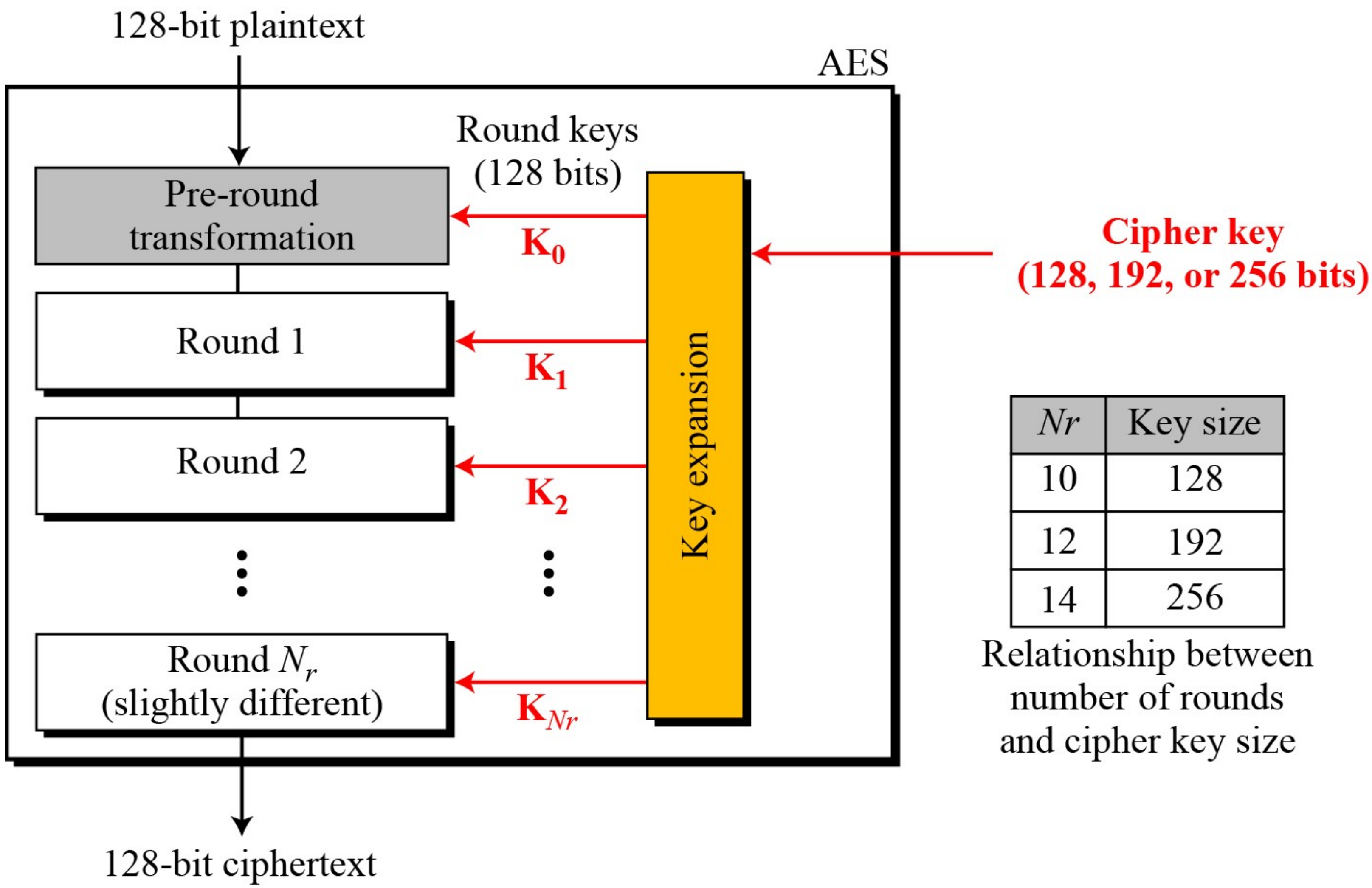
- An **iterative** rather than **feistel** cipher
  - Processes data as block of 4 columns of 4 bytes.
  - Operates on entire data block in every round.
- Designed to be:
  - Resistant against known attacks.
  - Speed and code compactness on many CPUs.
  - Design simplicity.

AES Encryption Process



No.of rounds	Key Length (bytes)
10	16
12	24
14	32

AES Encryption Process

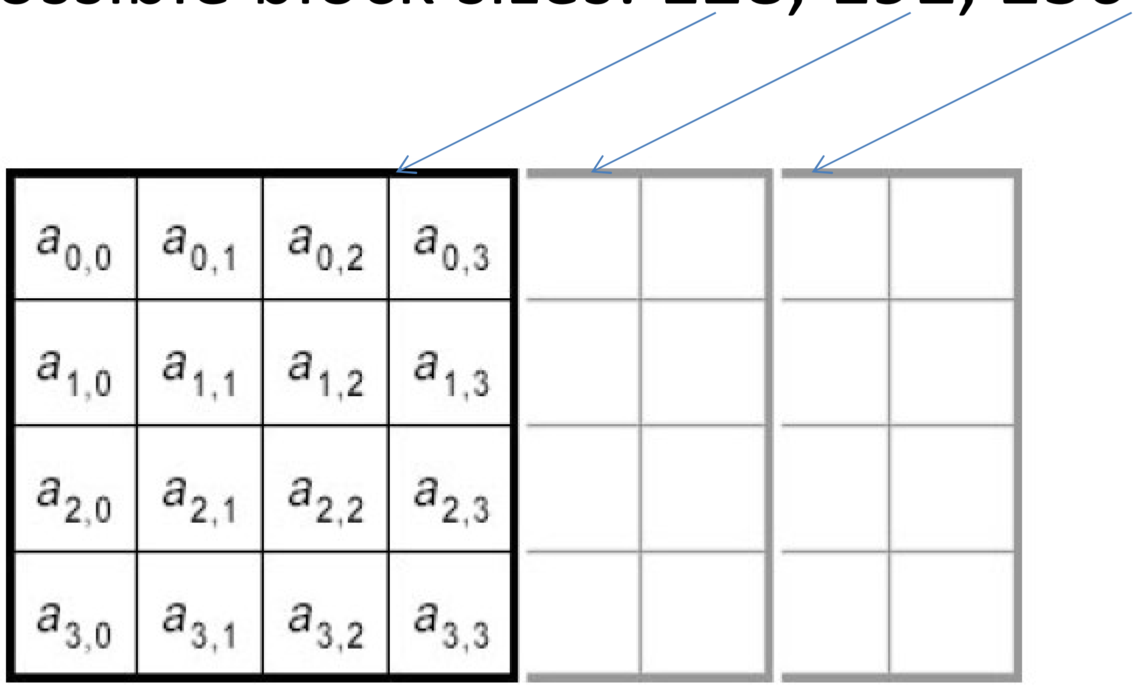


Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

The AES Cipher- Input Text

- Possible block sizes: 128, 192, 256 bit



The state

The AES Cipher

- Assume 128 bit block as input
- Input blocks represented as states at intermediates stages.

Input

in_0	in_4	in_8	in_{12}
in_1	in_5	in9	in13
in2	in6	in10	in14
in3	in7	in11	in15

→

State array

S0,0	S0,1	S0,2	S0,3
S1,0	S1,1	S1,2	S1,3
S2,0	S2,1	S2,2	S2,3
S3,0	S3,1	S3,2	S3,3

→

Output

out0	out 4	out8	out12
out1	out5	out9	out13
out2	out6	out10	out14
out3	out7	out11	out15

The AES Cipher

- Key received as input array of 4 rows and N_k columns.
- N_k = 4,6, or 8, parameter which depends key size 128,192 or 256.
- Input key is expanded into an array of 44/52/60 words of 32 bits each depending upon key size.
- 4 different words serve as a key for each round.

k0	k4	k8	k12
k1	k5	k9	k13
k2	k6	k10	k14
k3	k7	k11	k15

$\longrightarrow$

w0	w1	w2	……	w42	w43

Nk = 4

Steps in Rijndael (AES)

Confusion

Step1: Byte Substitution

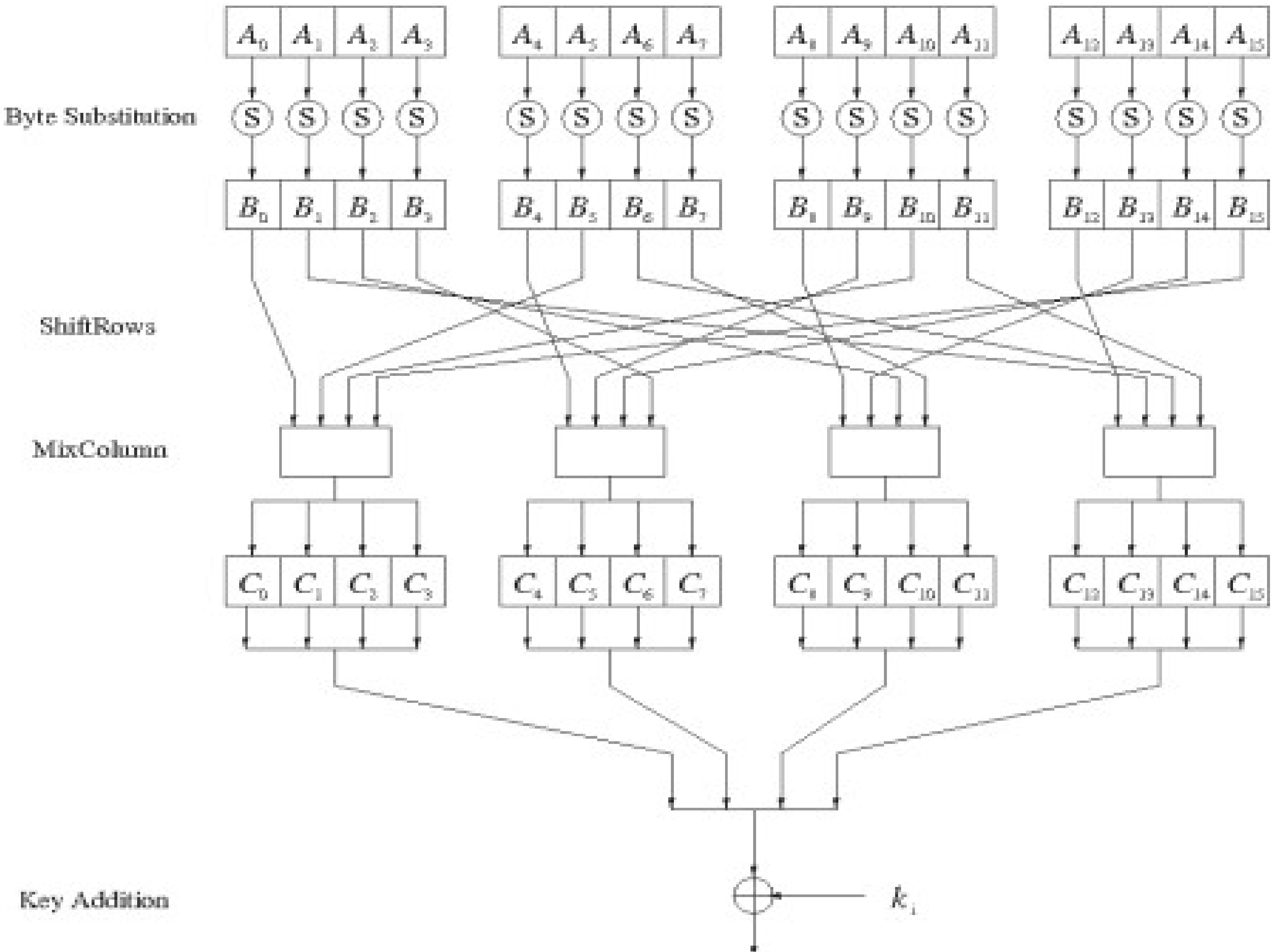
Diffusion

Step2: Shift Rows

Step3: Mix Cloumns

Step4: Round Key addition

Repeat these four steps 10,12,14 times



Courtesy: Book: Understanding cryptography

Steps in Rijndael (AES)

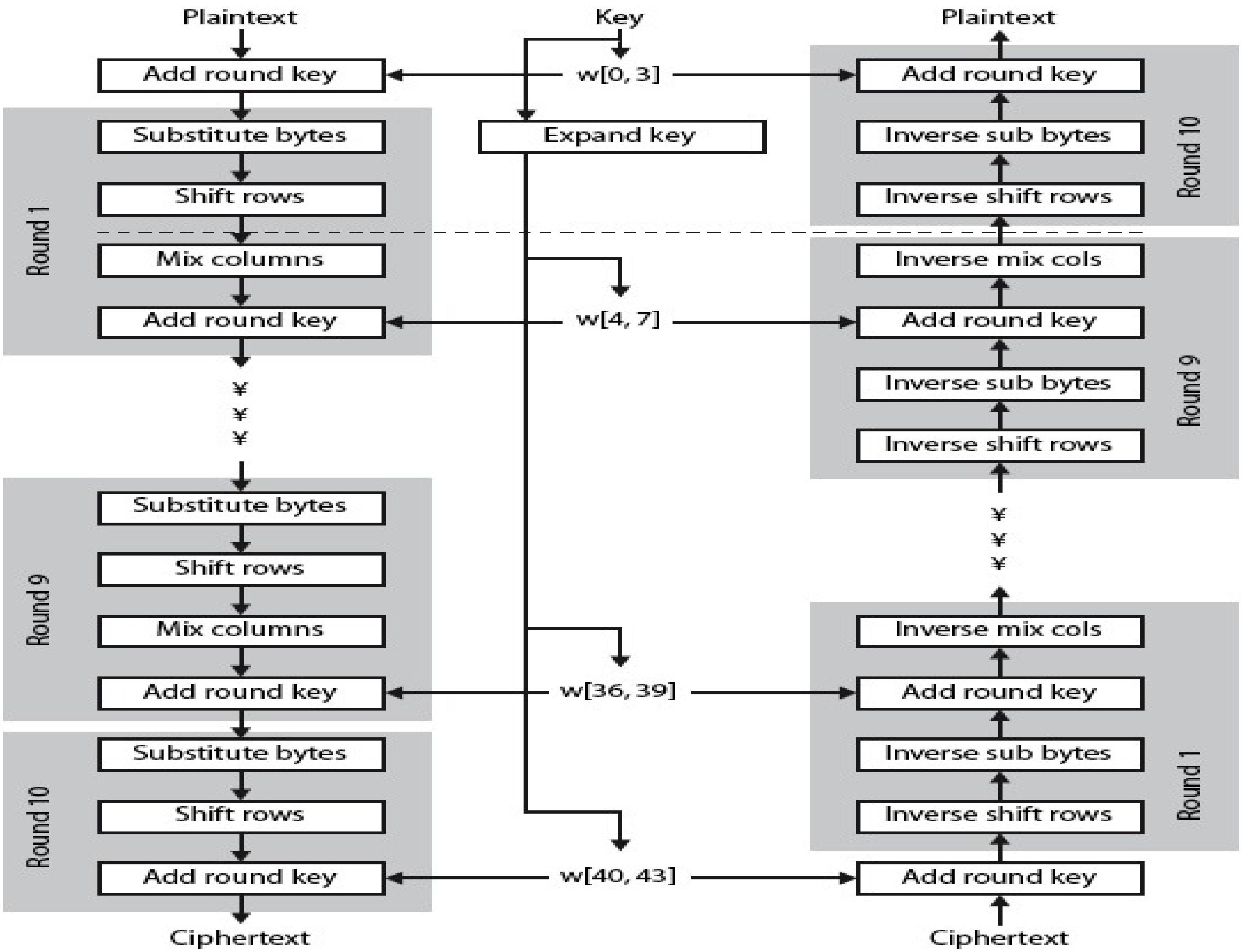
- SubBytes() – uses S-box to perform a byte-by-byte substitution of State, making use of arithmetic over GF(2^8).
- ShiftRows() – processes the State by cyclically shifting the last three rows of the State by different offsets.
- MixColumns() – takes all the columns of the State and mixes their data, independently of one another, making use of arithmetic over GF(2^8).
- AddRoundKey() – round key is added to the State using XOR operation.

Rijndael (AES)

- Data block of 4 columns of 4 bytes is state and key is expanded to array of words.
- Has 10/12/14 rounds in which state undergoes:
  - Byte substitution (1 S-box used on every byte)
  - Shift rows (permute bytes between groups/columns)
  - Mix columns (subs using matrix multipy of groups)
  - Add round key (XOR state with key material)

    Diffusion
  - View as alternating XOR key & scramble data bytes
- Initial XOR key material (1op.) & incomplete last (3ops.) round (so 10 rounds each of 4 ops.)

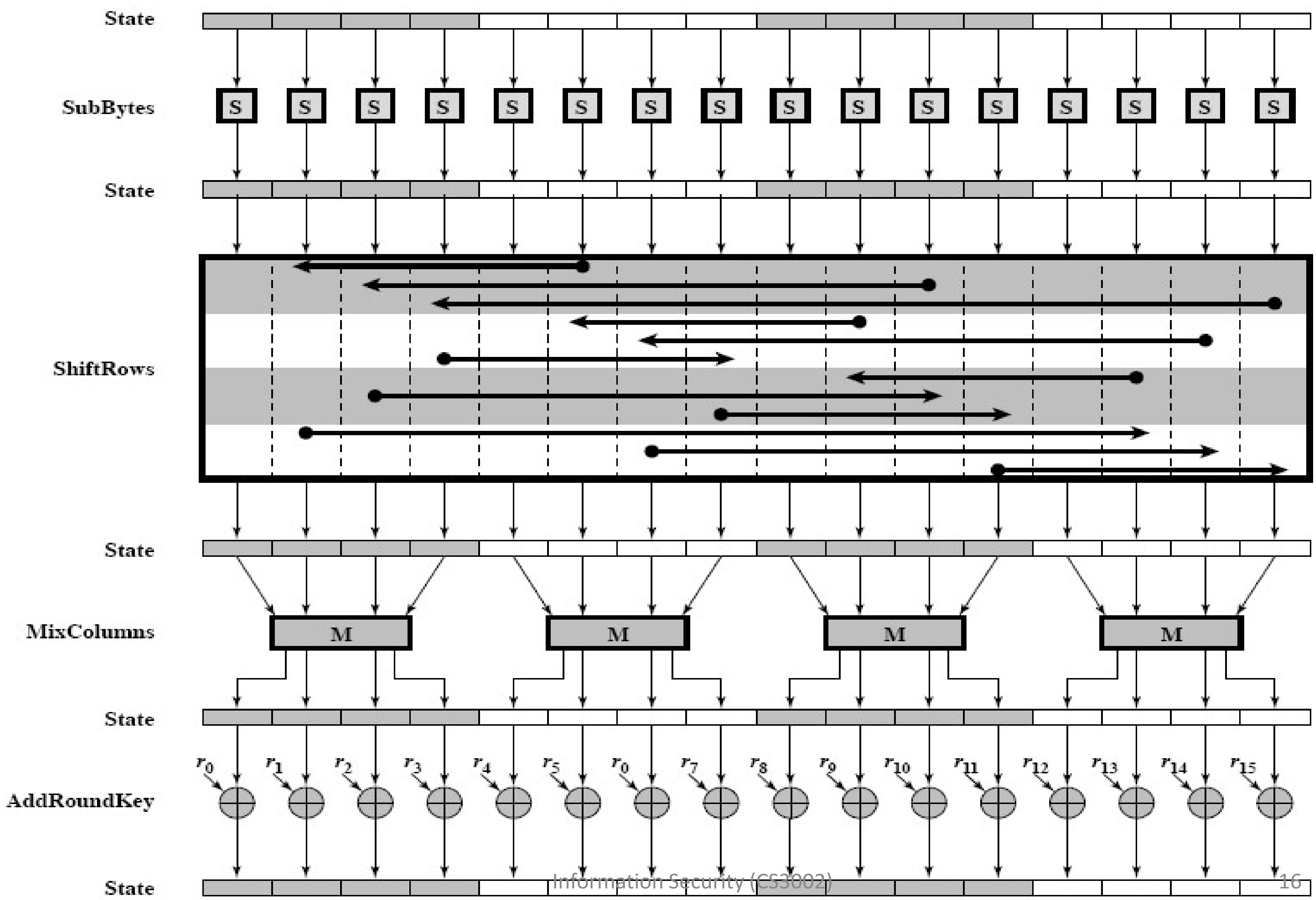
Rijndael (AES)



(a) Encryption

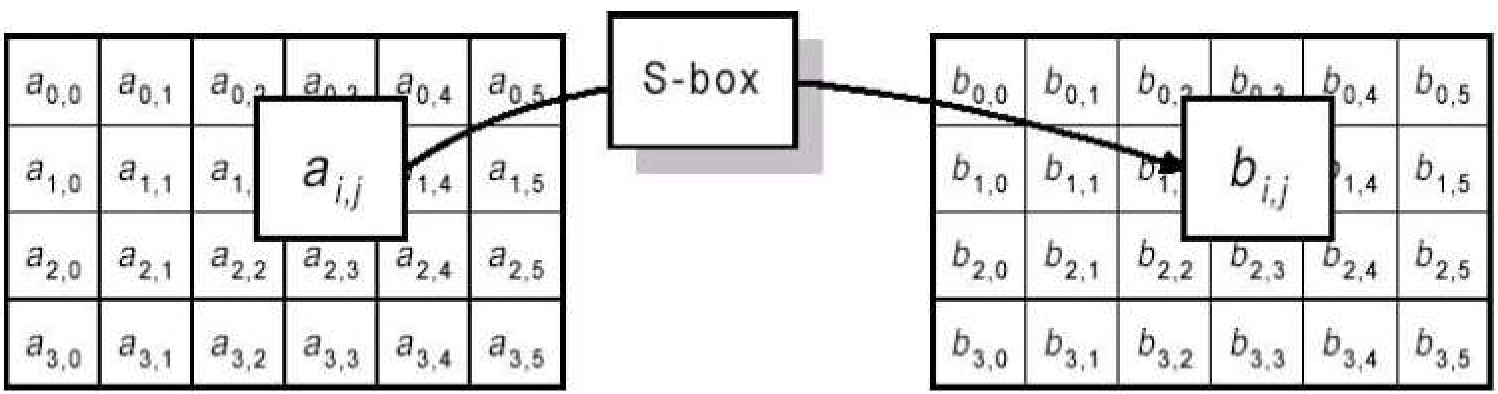
(b) Decryption

AES Encryption Round



1. Byte Substitution

- Non linear byte substitution
  - Multiplicative inverse in GF(2^8)



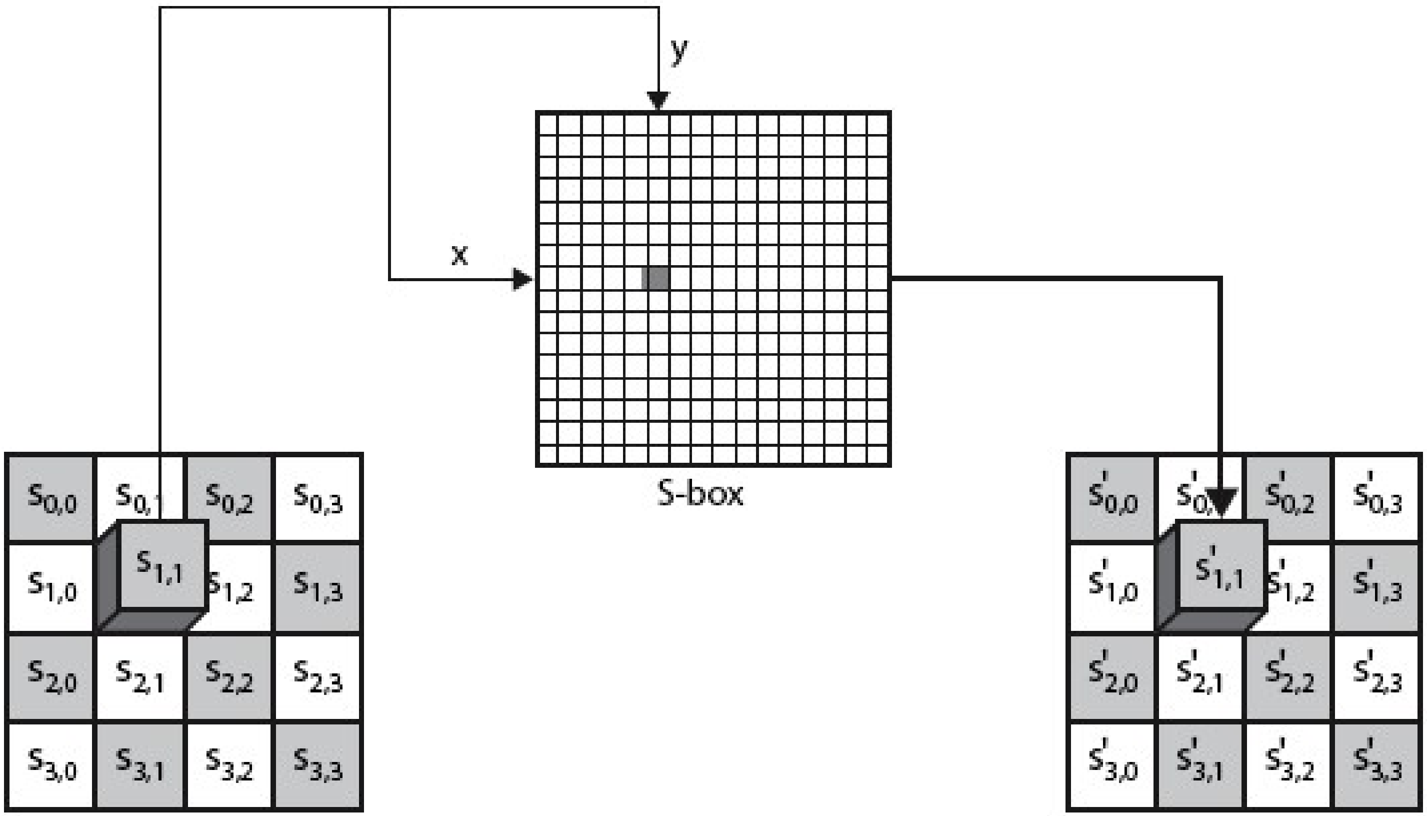
ByteSub acts on individual bytes of the state

Table 5.2 AES S-Boxes

		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

Byte Substitution (cont.)



Byte Substitution (cont.)

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

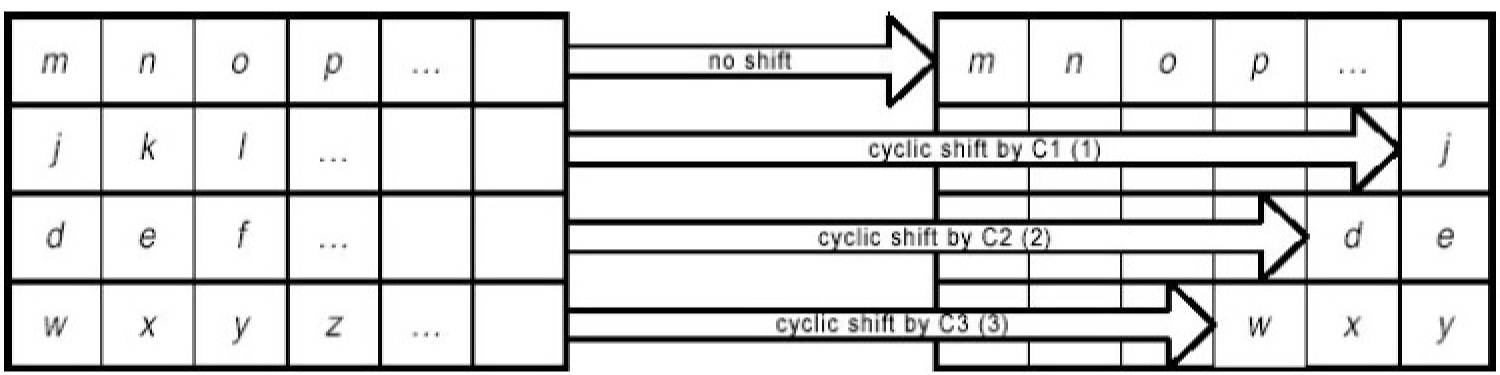
→

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

2. Shift Rows

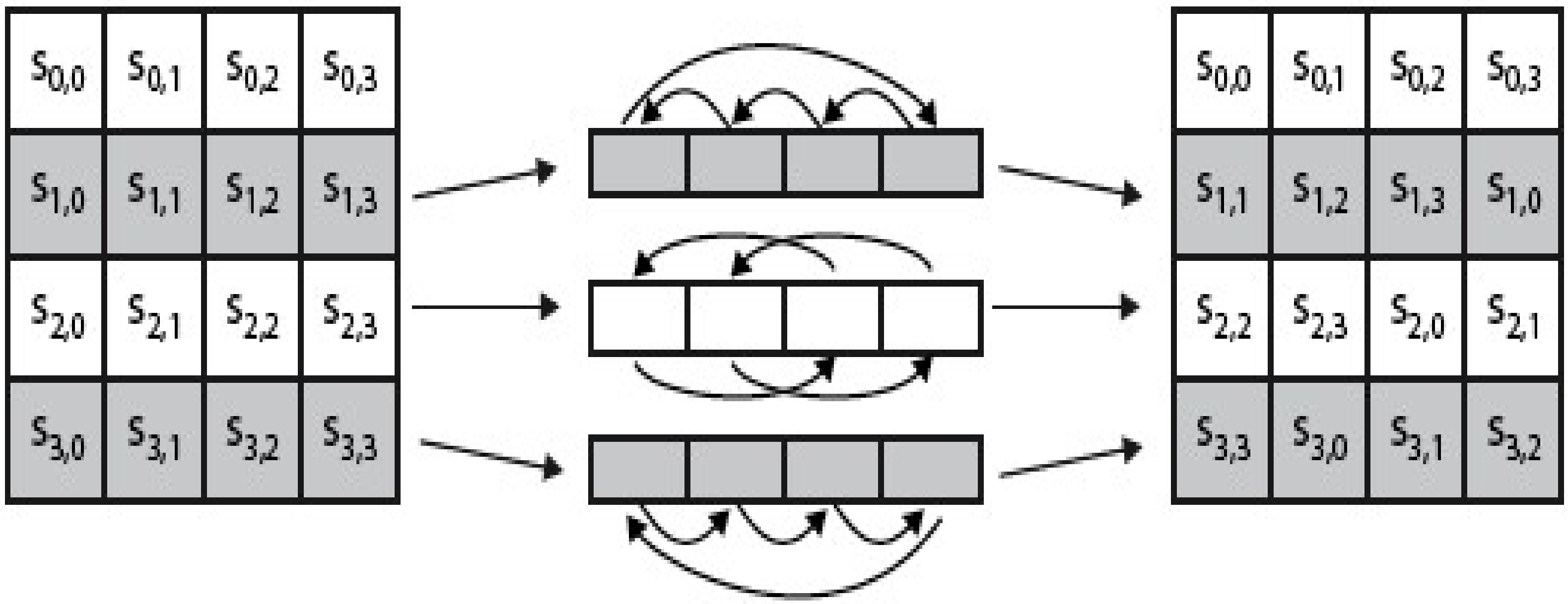
- rotating the rows with different offsets
  – Depending on block size
  – Cyclic Shift left

BlockSize	C1	C2	C3
128	1	2	3
196	1	2	3
256	1	3	4



ShiftRow operates on the rows of the state

Shift Rows (cont.)



- Decrypt inverts using *shifts to right*
- Since state is processed by columns, this step permutes bytes between the columns

3. Mix Columns

- Each column is processed separately.
- Each byte is replaced by a value dependent on all 4 bytes in the column.
- Can express each column as 4 equations
  - to derive each new byte in column.
- Decryption requires use of inverse matrix
  - with larger coefficients, hence a little harder

Mix Columns (cont.)

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

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

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

Mix Columns (cont.)

$$(\{02\} \bullet \{87\}) \oplus (\{03\} \bullet \{6E\}) \oplus \{46\} \oplus \{A6\} = \{47\}$$

$$\{87\} \oplus (\{02\} \bullet \{6E\}) \oplus (\{03\} \bullet \{46\}) \oplus \{A6\} = \{37\}$$

$$\{87\} \oplus \{6E\} \oplus (\{02\} \bullet \{46\}) \oplus (\{03\} \bullet \{A6\}) = \{94\}$$

$$(\{03\} \bullet \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \bullet \{A6\}) = \{ED\}$$

Multiplication and Addition is in GF(2^8).

Mix Columns (cont.)

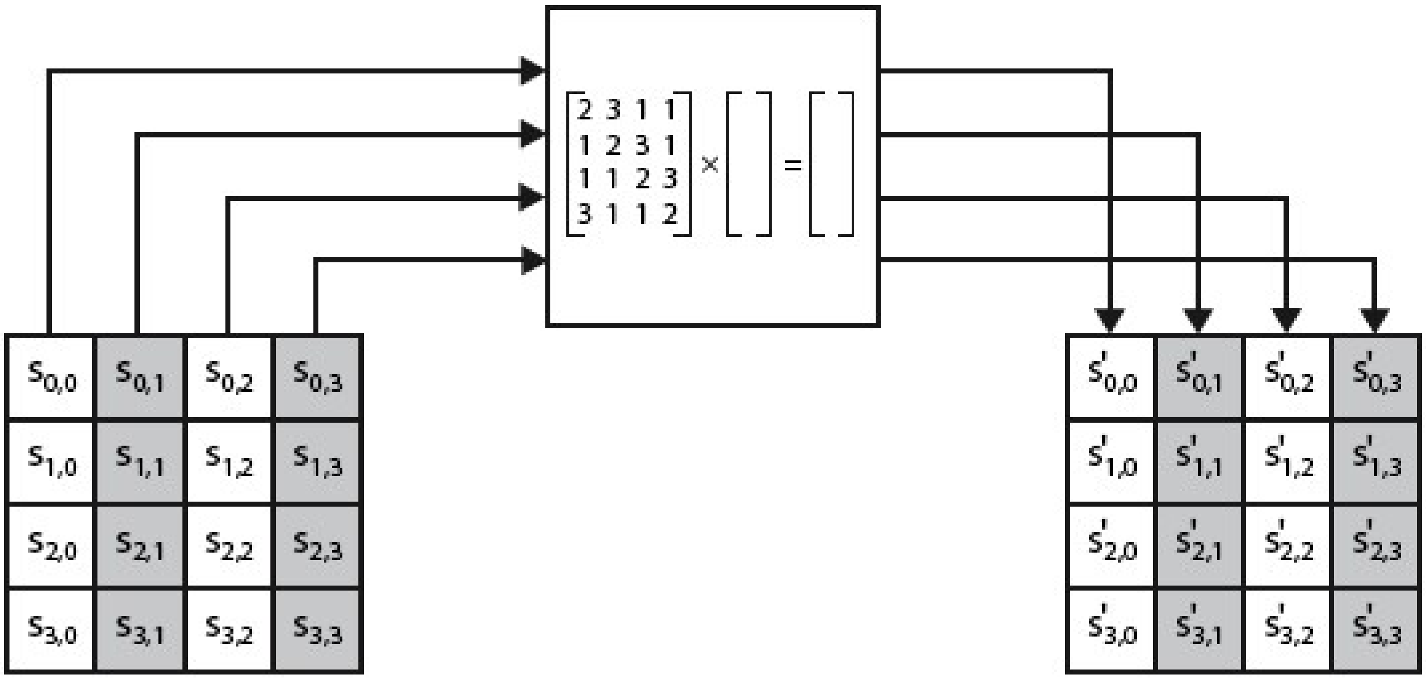


Fig.5.5 (b)

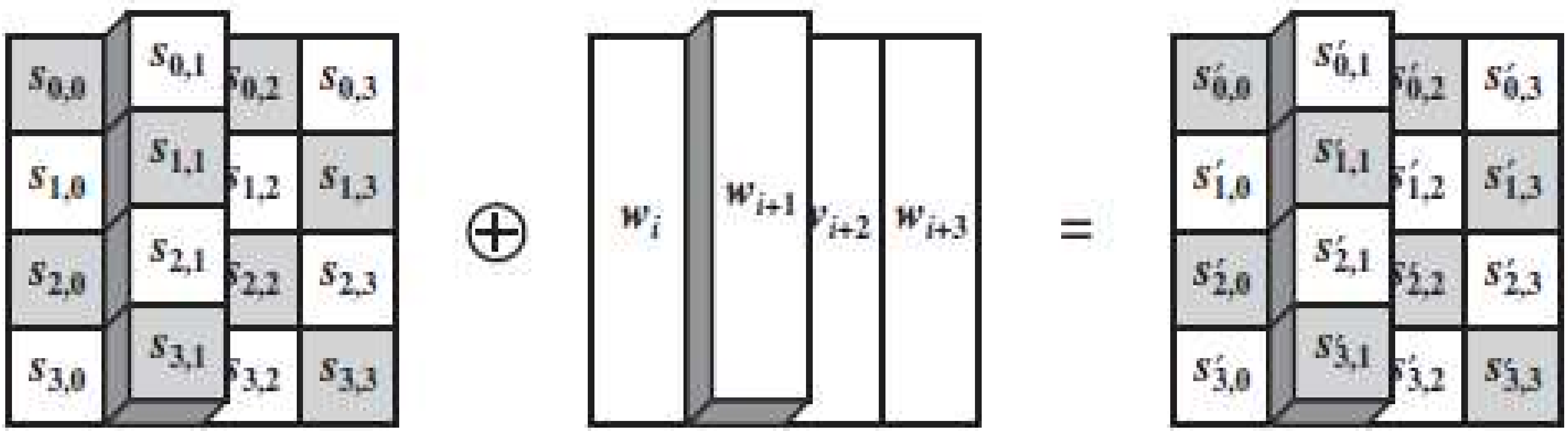
4. Add Round Key

- Apply the roundkey with a bitwise XOR.
- Size of roundkey equals block size.

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & a_{0,4} & a_{0,5} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} & k_{0,4} & k_{0,5} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} & k_{1,4} & k_{1,5} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} & k_{2,4} & k_{2,5} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} & k_{3,4} & k_{3,5} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} & b_{0,4} & b_{0,5} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \end{bmatrix}$$

Bitwise round key addition

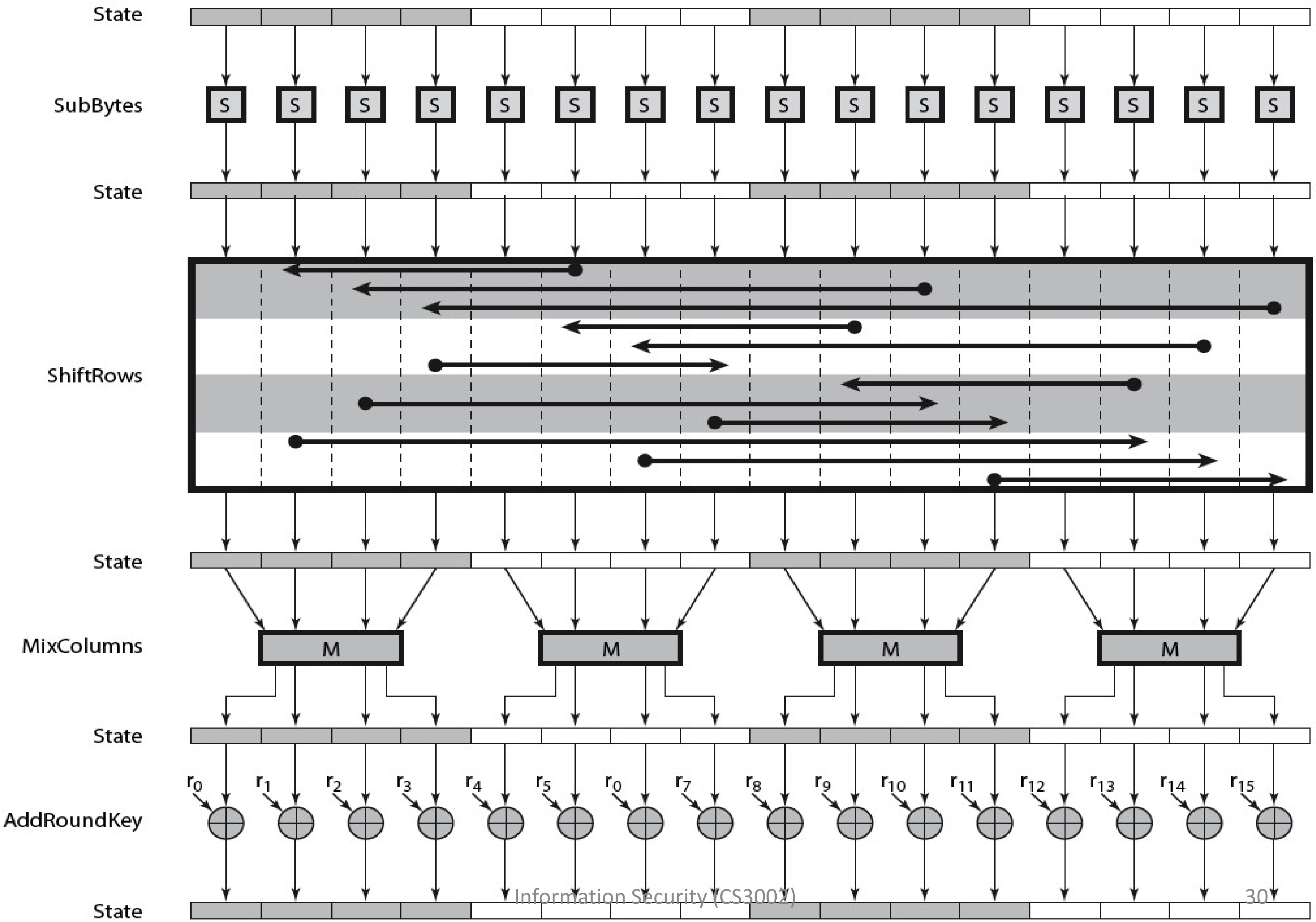
Add Round Key (cont.)



Add Round Key (cont.)

- Inverse for decryption identical.
  - Since XOR own inverse, with reversed keys.
- Designed to be as simple as possible.
  - A form of Vernam cipher on expanded key.
  - Requires other stages for complexity / security.

AES Round



State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_0 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

AES Key Expansion

Nk=4,6,8

Nr=10,12,14

AES Key Expansion

- Takes as input a N_k word key and produces a linear array of N_k * (N_r+1) words.
- Expanded key provide a N_k word round key for the initial AddRoundKey() stage and for each of the N_r rounds of the cipher.
- The key is first copied into the first N_k words, the remainder of the expanded key is filled N_k words at a time.

Key Expansion (cont.) Pseudo Code 16 byte key

```
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;
   }
}
```

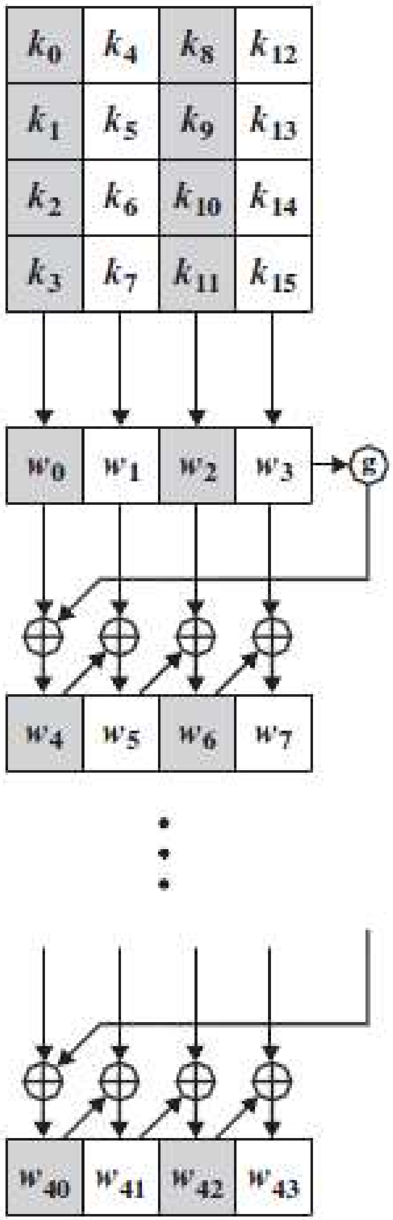
Key Expansion (cont.)

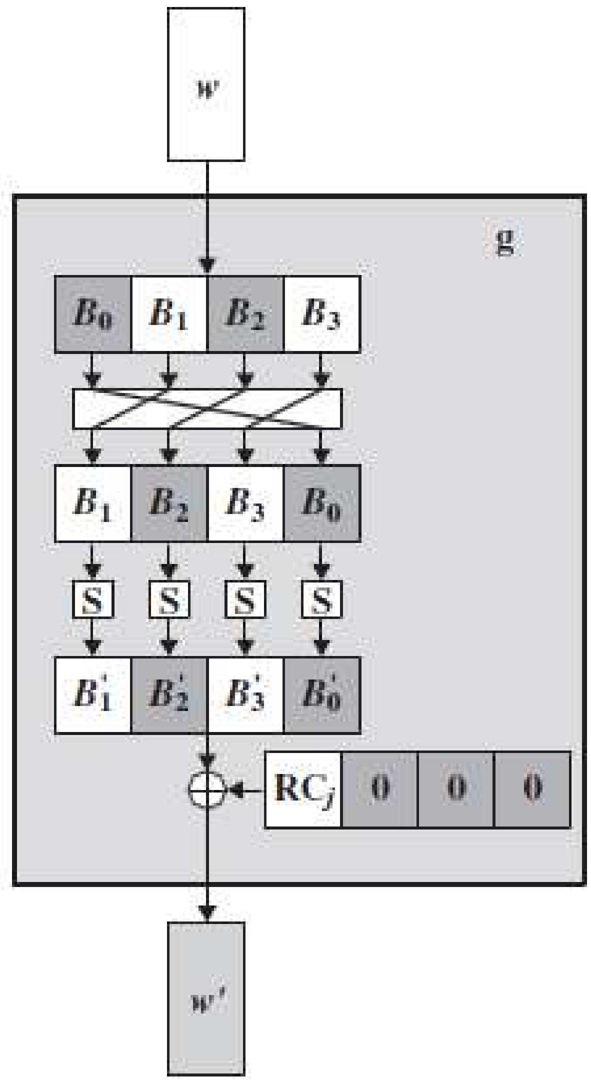
- **`RotWord`** performs a one byte circular left shift on a word. For example:

  `RotWord[b0,b1,b2,b3] = [b1,b2,b3,b0]`

- **`SubWord`** performs a byte substitution on each byte of input word using the S-box (similar to SubBytes())

- **`SubWord(RotWord(temp))`** is XORed with Rcon[j] – the round constant.





(b) Function g

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

Nk=4,6,8

Nr=10,12,14

Pseudo Code for All Key Sizes

```
KeyExpansion(int* Key[4*Nk], int* EKey[Nb*(Nr+1)])
{
    for(i = 0; i < Nk; i++)
       EKey[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
    for(i = Nk; i < Nb * (Nr + 1); i++)
    {
       temp = EKey[i - 1];
       if (i % Nk == 0)
          temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
       EKey[i] = EKey[i - Nk] ^ temp;
    }
}
```

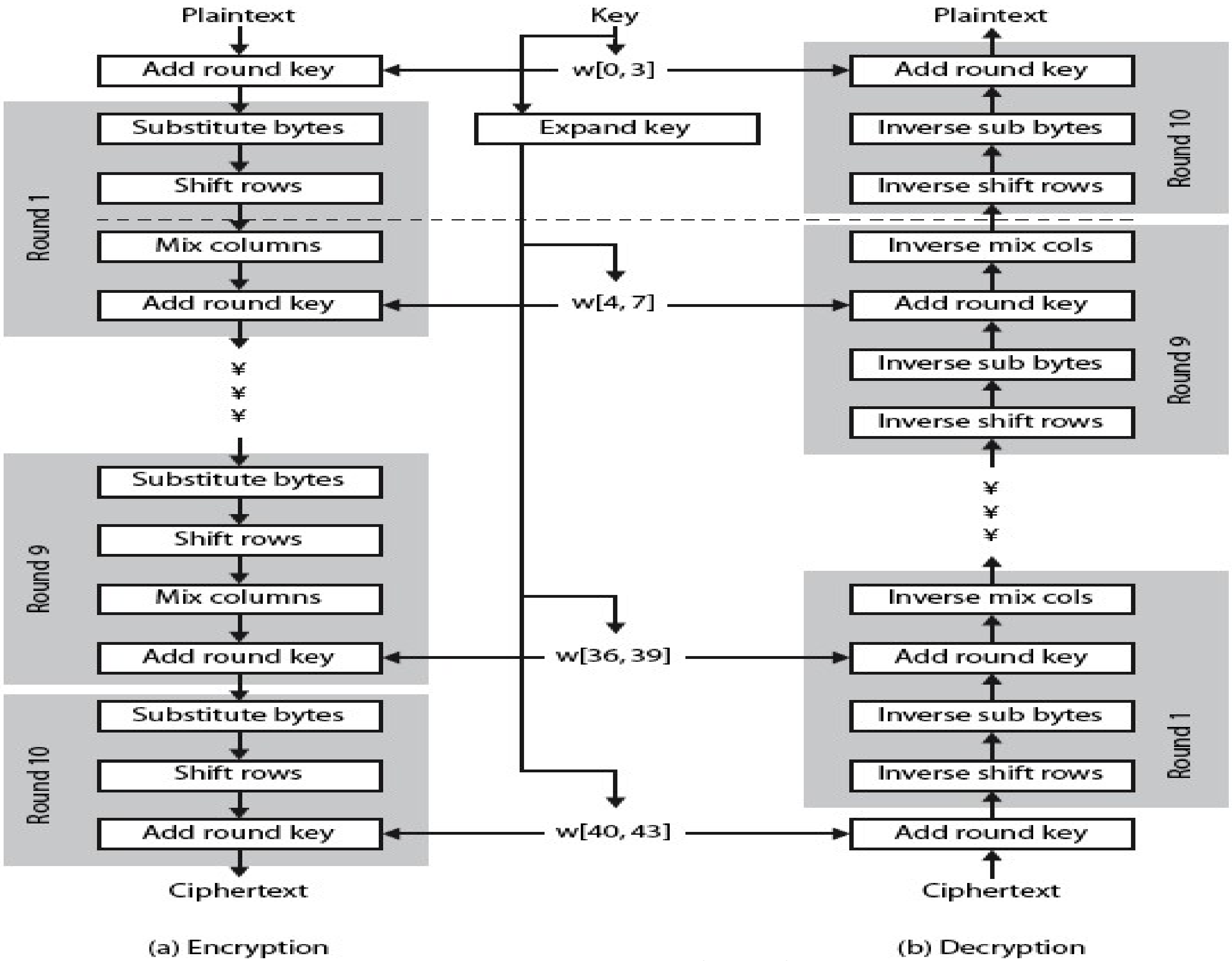
Decryption of AES (Self-Study)

tweak -adjust finely

AES Decryption

- AES decryption is not identical to encryption since steps done in reverse.
- But can define an equivalent inverse cipher with steps as for encryption.
  - but using inverses of each step
  - with a different key schedule
- Works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key

Rijndael


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

Equivalent Inverse Cipher

- The original sequence is :

Encryption:

SubBytes ShiftRows MixColumns AddRoundKey

Decryption:

InvShiftRows InvSubBytes AddRoundKey InvMixColumns

- Thus **InvShiftRows** needs to be interchanged with **InvSubBytes** and **AddRoundKey** with **InvMixColumns**.

Equivalent Inverse Cipher

- **`InvShiftRows`** – Affects sequence of bytes but does not alter byte content and does not depend on the byte content to perform transformation.
- **`InvSubBytes`** – Affects content of bytes but does not alter byte sequence and does not depend on the byte sequence to perform transformation.
- Thus **`InvShiftRows and InvSubBytes`** can be interchanged. For given state **`S`**,

`InvShiftRows(InvSubBytes(S))`

`=`

`InvSubBytes(InvShiftRows(S))`

Equivalent Inverse Cipher

- If key is viewed as sequence of words then both **`AddRoundKey`** and **`InvMixColumns`** operate on state one column at a time.

- These operations are linear with respect to the column input: State – S and key - w

  `InvMixColumns`(S XOR w) =
  [**`InvMixColumns`**(S)] XOR **`InvMixColumns`**(w)]

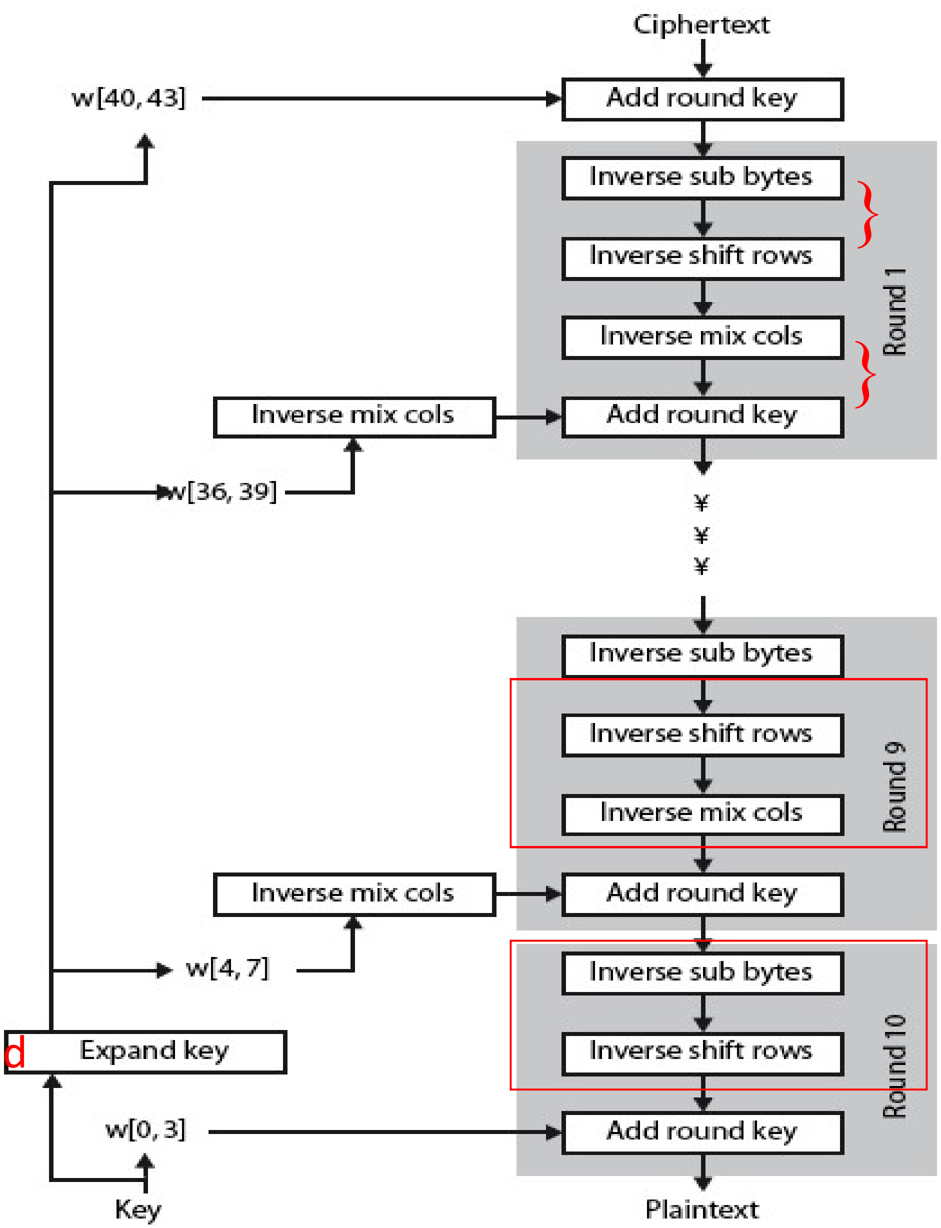
Equivalent Inverse Cipher

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y0 \text{ XOR } k0 \\ y1 \text{ XOR } k1 \\ y2 \text{ XOR } k2 \\ y3 \text{ XOR } k3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \end{bmatrix}$$

$$+ \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \end{bmatrix}$$

- Thus **`InvMixColumns`** and **`AddRoundKey`** can be interchanged.

AES Decryption



- **InvShiftRows ↔ InvSubBytes** and
- **AddRoundKey ↔ InvMixColumns**.

Two Decryptions compared

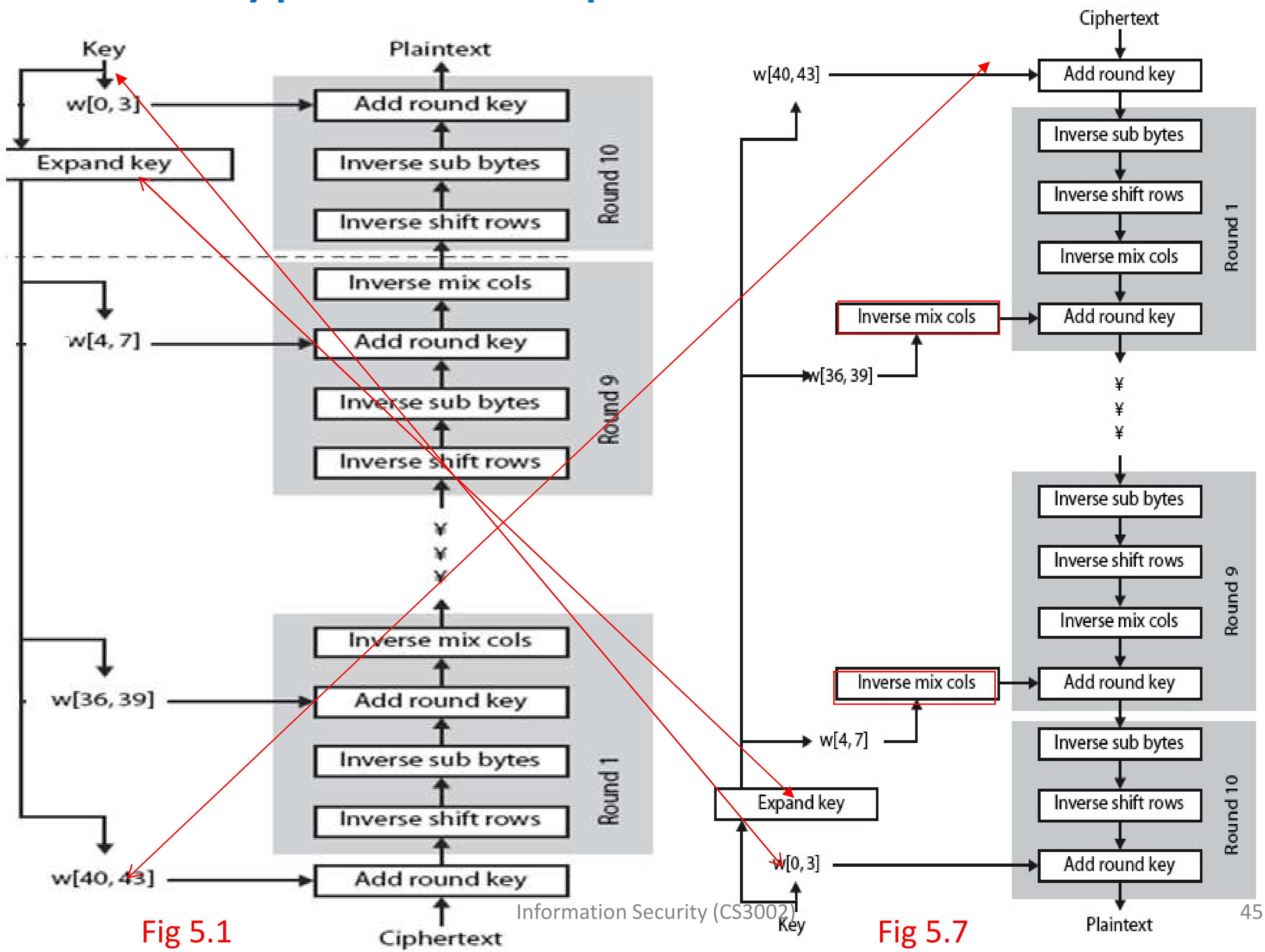


Fig 5.1

Fig 5.7

Equivalent Inverse Cipher Advantages

- AS AES decryption cipher is not identical to the encryption cipher (Fig 5.1)
- Disadvantage – Two separate software or hardware modules are required if performing both encryption and decryption.
- So equivalent version of the decryption algorithm that has the same structure (the same sequence of transformations) as the encryption algorithm, can use same software (Fig 5.7)