Lecture 7
Advanced Encryption Standard

Jason Lin

學習目標

- 回顧 AES 的發展歷史
- 定義 AES 的基本結構
- 定義 AES 的轉換
- 定義金鑰擴展程序
- 討論不同的實作方式

7.1 簡介

- 由於計算機技術的大幅進步使得電腦的運算能力大幅提昇，對於使用 56 位元金鑰的 DES 而言可說是面臨了威脅，因此美國國家標準技術局（NIST）徵求下一代的加密標準。
- 進階加密標準（Advanced Encryption Standard，AES）是 NIST 於 2001 年 12 月所發布的對稱式區塊加密法。
- 本節討論主題
  - 歷史
  - 評選準則
  - 回合數
  - 資料單位
  - 每個回合的結構

7.1.1 歷史

- 在 1977 年，NIST 公開招募取代 DES 的標準加密法，並命名為“**進階加密標準**”AES。而當時對 AES 的需求規格如下：
  - 明文區塊大小 128 位元
  - 金鑰長度為 128、192 及 256 位元三種長度
  - 演算法必須公開
- 1998 年 8 月，舉行第一次 AES 候選研討會（21 取 15）。
- 1999 年 9 月，舉行第二次 AES 候選研討會（15 取 5）。
  - MARS、RC6、Rijndael、Serpent 以及 Twofish

7.1.1 歷史 (續)

- 2000 年 10 月舉行第三次 AES 候選研討會，NIST 宣布由比利時兩位密碼學家 Joan Daemen 和 Vincent Rijment 所設計的 Rijndael 演算法作為新的進階加密標準 AES

Joan Daemen (左) 和 Vincent Rijmen (右)

7.1.1 歷史 (續)

- 2001 年 2 月，NIST 針對 AES 公開發布了一份聯邦資訊處理標準（Federal Information Processing Standard, FIPS）的草案
- 最後，在 2001 年 12 月，AES 正式發表於《聯邦公報》，稱為 FIPS 197

7.1.2 評選準則

- NIST 對 AES 的評選準則
  - 安全性
  - 成本
  - 實作

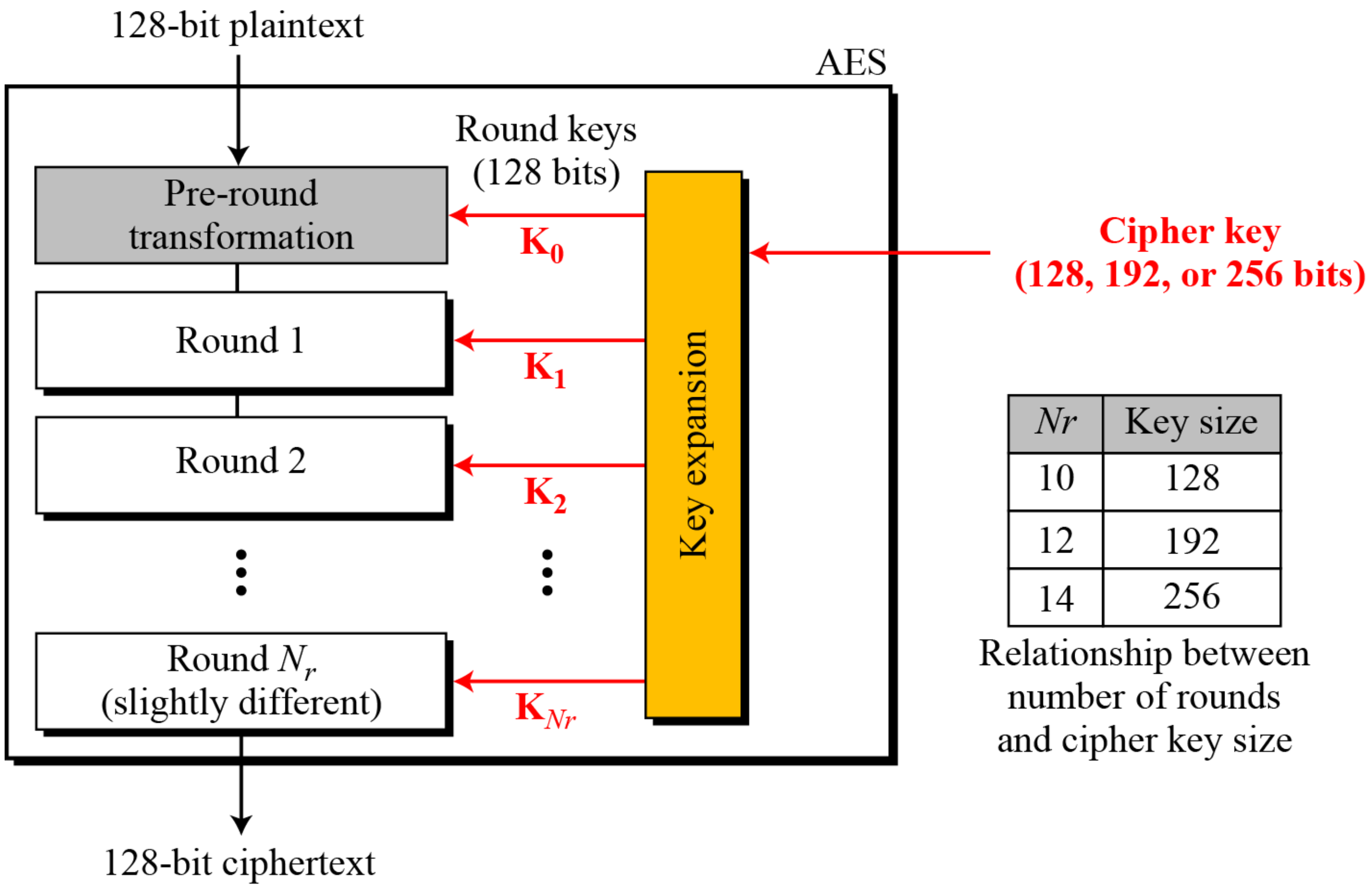
7.1.3 回合數

- AES 是一種非 Feistel 架構，資料區塊的長度為 128 位元，其運算回合數可為十、十二和十四個回合，而秘密金鑰的長度則取決於回合數（Round），可以有 128、192 和 256 位元。

注意

AES 總共定義了三個版本，分別有十、十二和十四個回合的運算，每個版本使用的秘密金鑰長度也都有所不同（分別是128、192或256位元），所有的回合金鑰長度也都是相對應於秘密金鑰長度。

圖 7.1 AES 加密法的設計概念



Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

圖 7.2 AES 所使用的資料單位

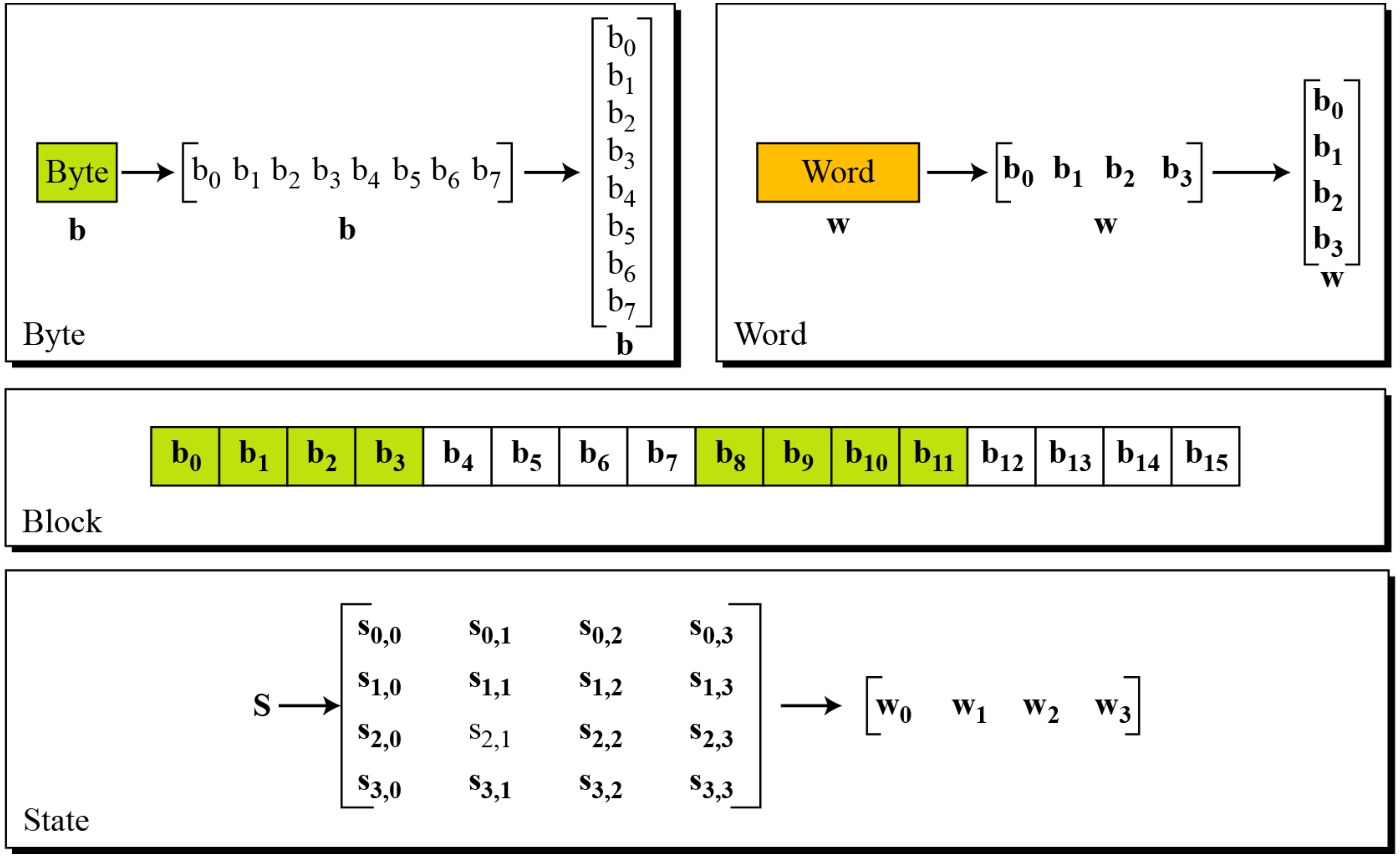


圖 7.3 區塊與狀態之間的轉換

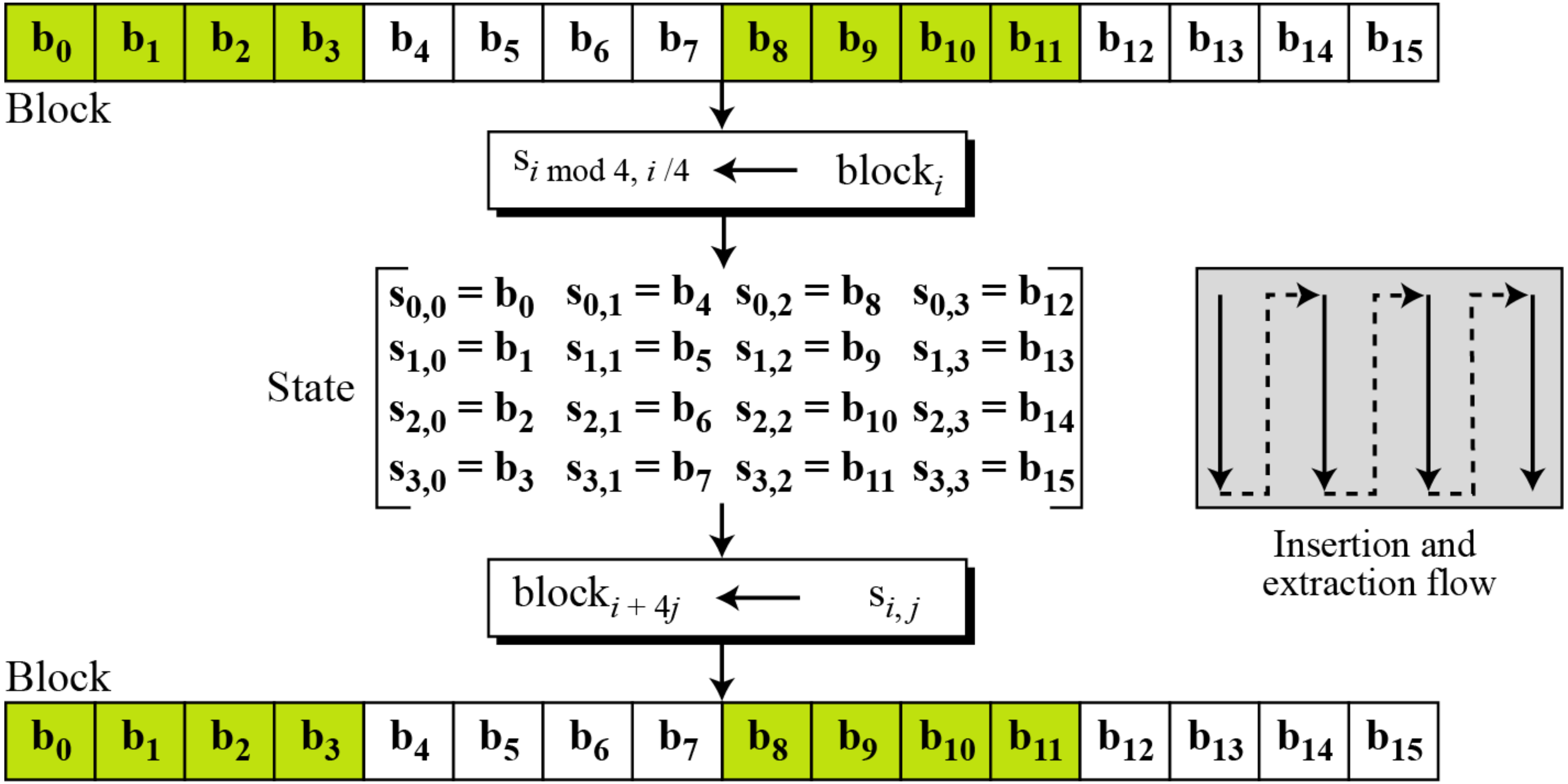
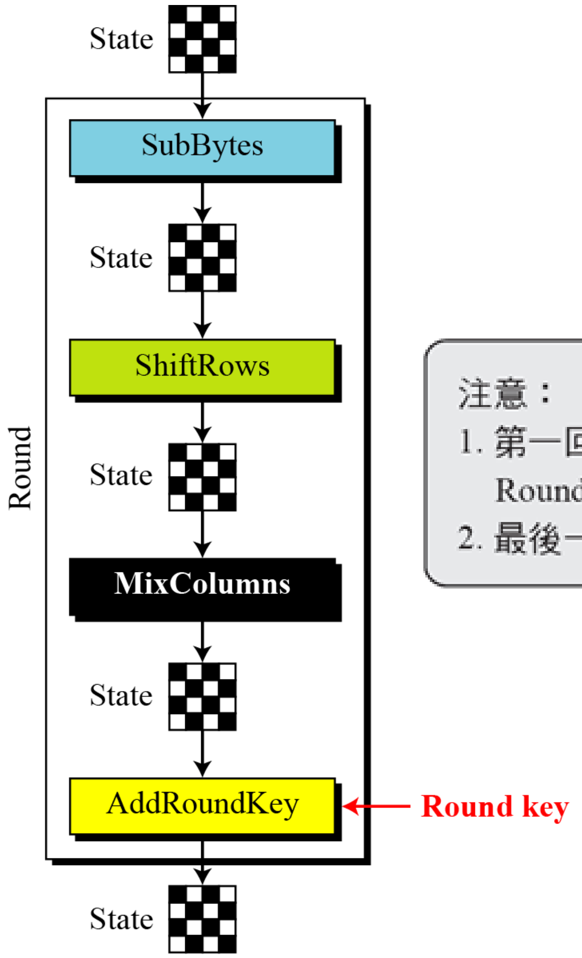


圖 7.4 將密文轉成狀態

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	17	19	19

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

圖 7.5 加密端每個回合的結構



注意：
1. 第一回合前必須先執行一次 Add-RoundKey。
2. 最後一個回合沒有第三個轉換。

7.2 轉換

- 為了安全起見，AES 使用了四種類型的轉換，分別是取代 (SubBytes)、排列 (ShiftRow)、混合 (MixColumn) 以及加入金鑰 (AddRoundKey)。
- 本節討論主題
  - SubBytes
  - ShiftRows
  - MixColumns
  - AddRoundKey

7.2.1 SubBytes

- AES 也像 DES 一樣使用取代，AES 使用兩個可逆的轉換。
- 第一種轉換是在加密端使用，稱為 SubBytes。在取代一個位元組的時候，首先將位元組表示成兩個十六進位的數字。

注意

SubBytes 運算總共有 16 個獨立的位元組與位元組的轉換。

圖 7.6 SubBytes 轉換

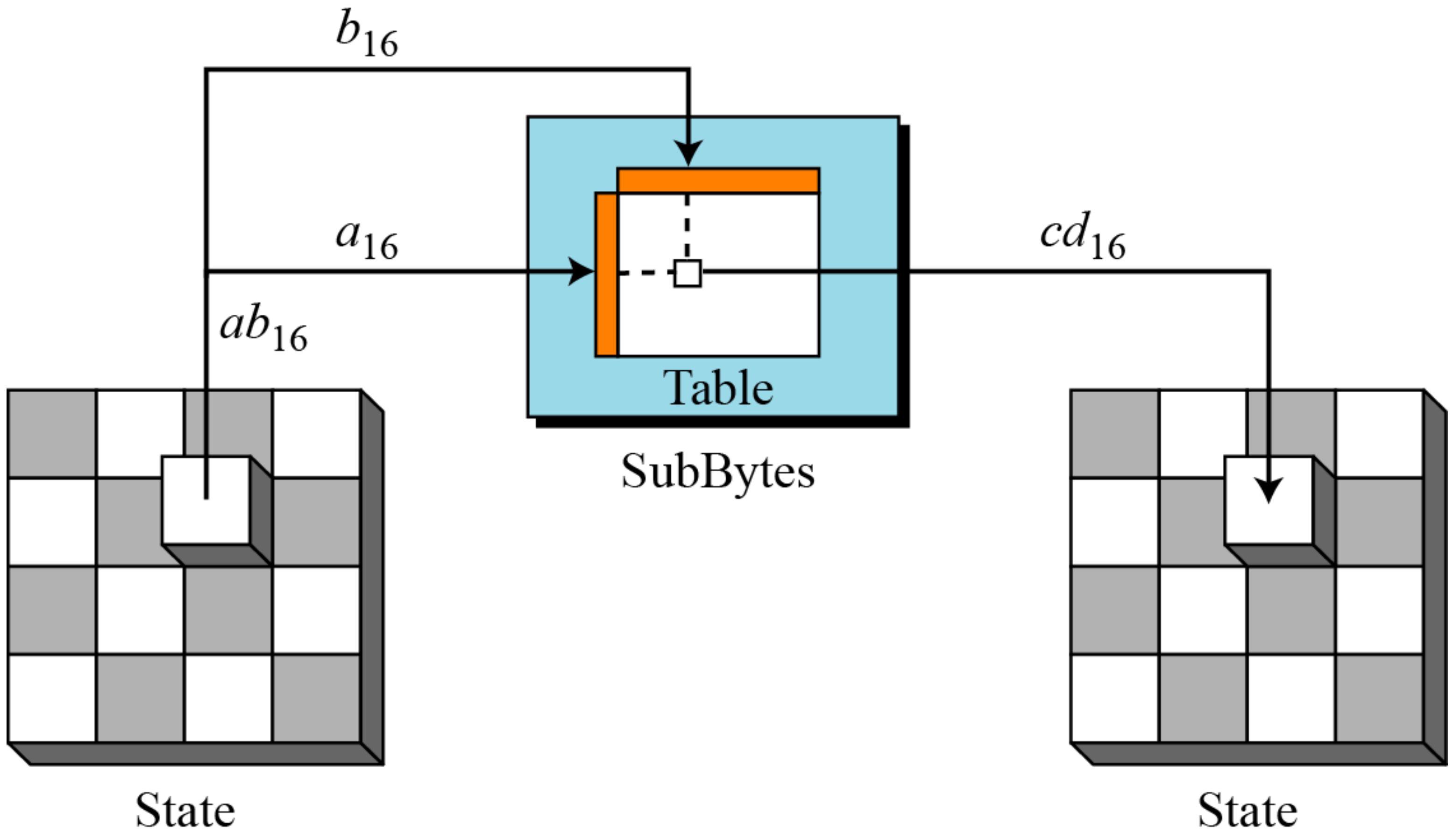
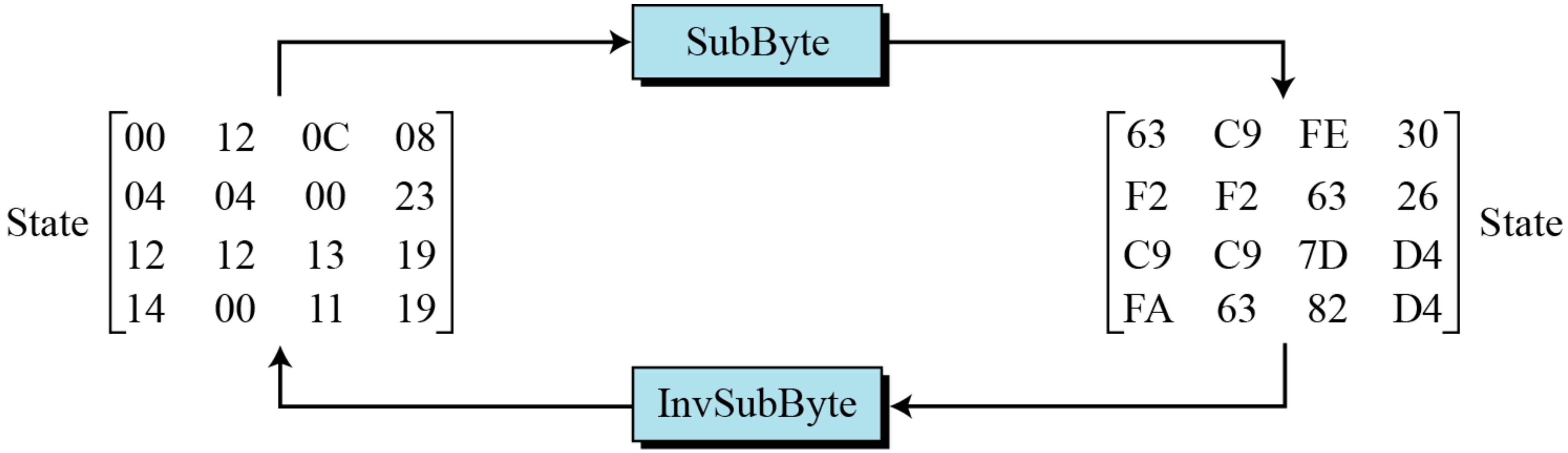


表 7.1 SubBytes 轉換表

	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	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

範例 7.2

- 圖 7.7 展示一組狀態如何使用 SubBytes 進行轉換，該圖同樣也展示出 InvSubBytes 轉換可產生原始的狀態。值得注意的是，若兩個位元組的值相同，則它們的轉換結果也相同。



使用 GF(2^8) 體的轉換

- AES 同時定義了一個代數上的轉換，亦即使用在 **GF**(2^8) 體下的不可分解多項式 ($x^8 + x^4 + x^3 + x + 1$)，請參考圖 7.8。

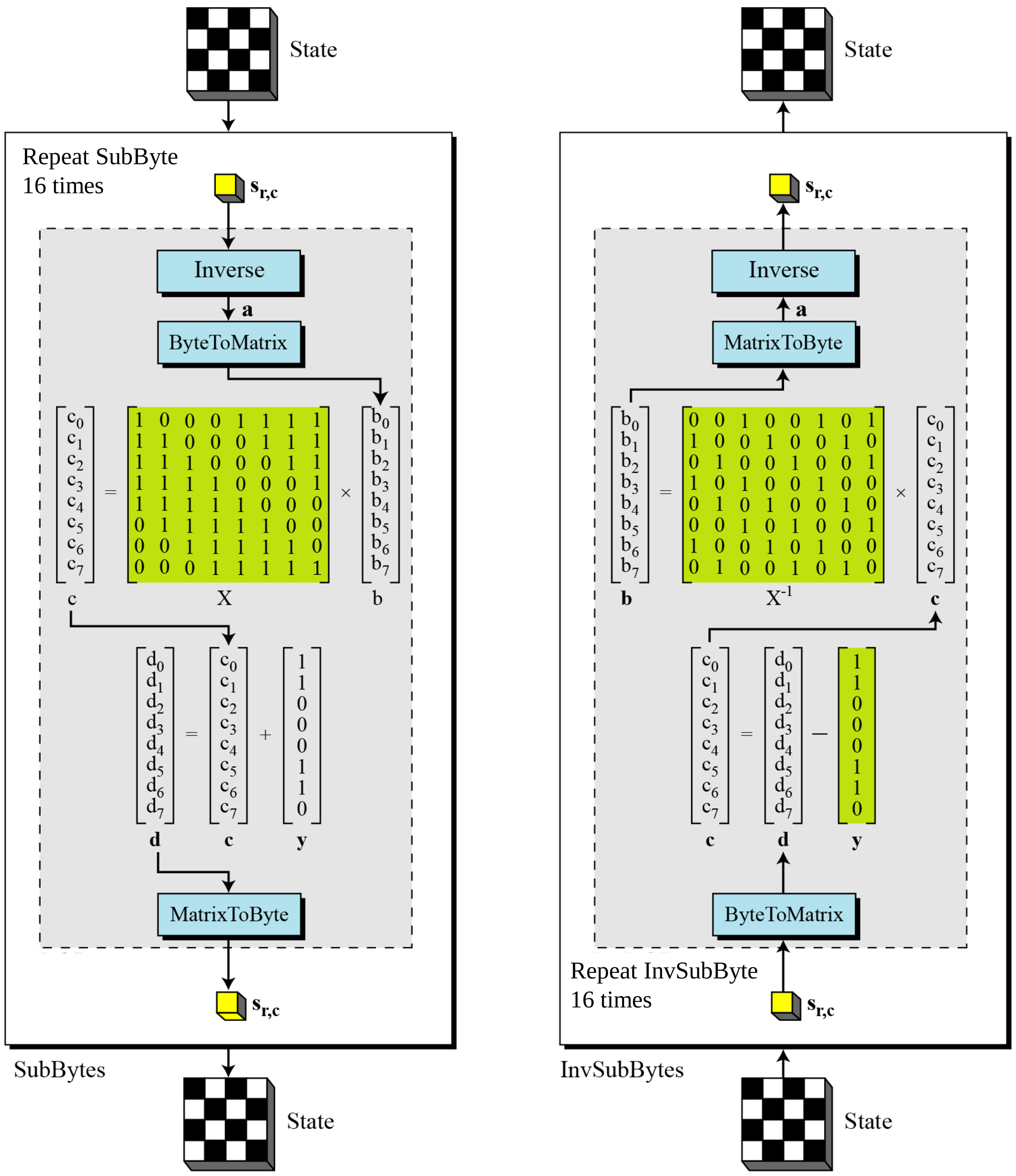
SubBytes: $\rightarrow$ $\mathbf{d} = \mathbf{X}(s_{r,c}) \oplus \mathbf{y}$

InvSubBytes: $\rightarrow$ $[\mathbf{X}^{-1}(\mathbf{d} \oplus \mathbf{y})]^{-1} = [\mathbf{X}^{-1}(\mathbf{X}(s_{r,c})^{-1} \oplus \mathbf{y} \oplus \mathbf{y})]^{-1} = [(s_{r,c})^{-1}]^{-1} = s_{r,c}$

注意

SubBytes 與 InvSubBytes 轉換互為反向。

圖 7.8 SubBytes 與 InvSubBytes 程序



範例 7.3

- 以下我們展示如何經由 SubBytes 程序將位元組 0C 轉換成 FE，再使用 InvSubBytes 程序將其轉換回 0C。
  - SubByte
    - 位元組 0C 在 **GF**(2^8) 體下的乘法反元素為 B0，所以 **b** 就是 (10110000)。
    - 將 **b** 乘上矩陣 **X**，得到矩陣 **c** = (10011101)。
    - 對 **b** 和 **y** 執行 XOR 運算，得到矩陣 **d** = (11111110)，也就是十六進位的 FE。
  - InvSubByte
    - 執行 XOR 運算後得到矩陣 **c** = (10011101)。
    - 乘上矩陣 $\mathbf{X}^{-1}$，得到 (11010000)，即為十六進位的 B0。
    - B0 的乘法反元素即是 0C。

演算法 7.1 SubBytes 轉換的虛擬程式碼

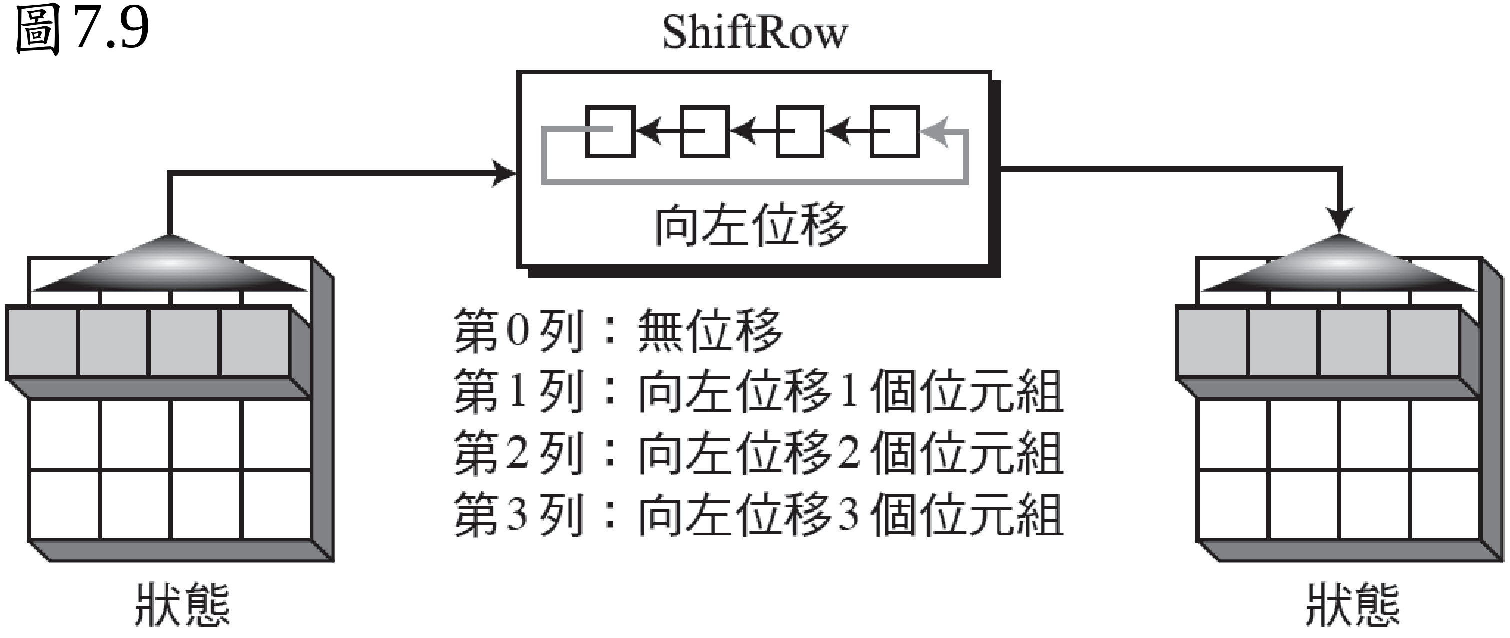
```
SubBytes (S)
{
   for (r = 0 to 3)
     for (c = 0 to 3)
          S_{r,c} = subbyte (S_{r,c})
}
```

```
subbyte (byte)
{
   a ← byte^{-1}                    // 在 GF(2^8) 體之下的乘法反元素 00 的反元素亦為 00
    ByteToMatrix (a, b)
    for (i = 0 to 7)
   {
        c_i ← b_i ⊕ b_{(i+4)mod 8} ⊕ b_{(i+5)mod 8} ⊕ b_{(i+6)mod 8} ⊕ b_{(i+7)mod 8}
        d_i ← c_i ⊕ ByteToMatrix (0x63)
   }
    MatrixToByte (d, d)
    byte ← d
}
```

7.2.2 ShiftRows

- **ShiftRows**
  - 在加密時，這種轉換稱為 **ShiftRows**。

圖7.9



7.2.2 ShiftRows (續)

- **InvShiftRows**
  - 在解密時，這種轉換稱為 **InvShiftRows**，其位移是由左向右。

演算法 7.2 ShiftRows 轉換的虛擬程式碼

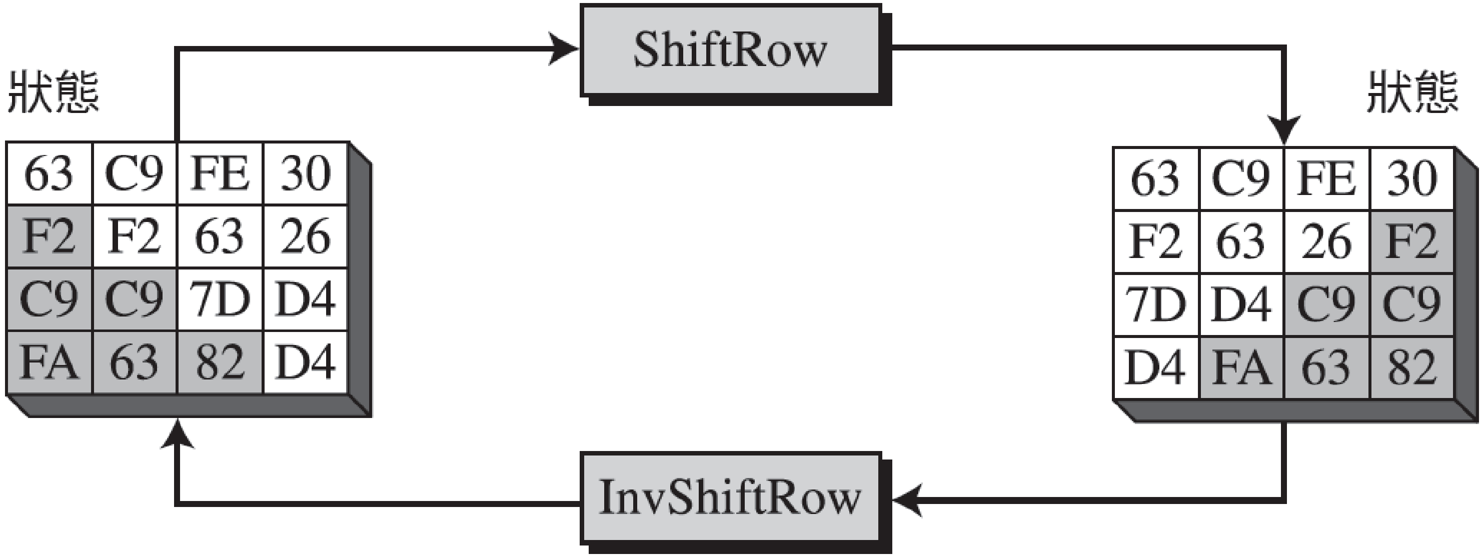
```
ShiftRows (S)
{
    for (r = 1 to 3)
        shiftrow (s_r, r)                          //s_r 為第 r 列
}
```

```
shiftrow (row, n)                                  //n 為位移的位元組數
{
  CopyRow (row, t)                                 //t 為暫時列
  for (c = 0 to 3)
       row_((c − n) mod 4) ← t_c
}
```

範例 7.4

- 圖 7.10 展示一組狀態如何使用 ShiftRows 進行轉換，也展示出 InvShiftRows 轉換可產生原始的狀態。

圖7.10



7.2.3 MixColumns

- 我們需要一個位元組間（Inter-Byte）的轉換，依據鄰近位元組的內容改變目前位元組內的位元。藉由混合位元組，以提供位元的擴散效果。

圖7.11

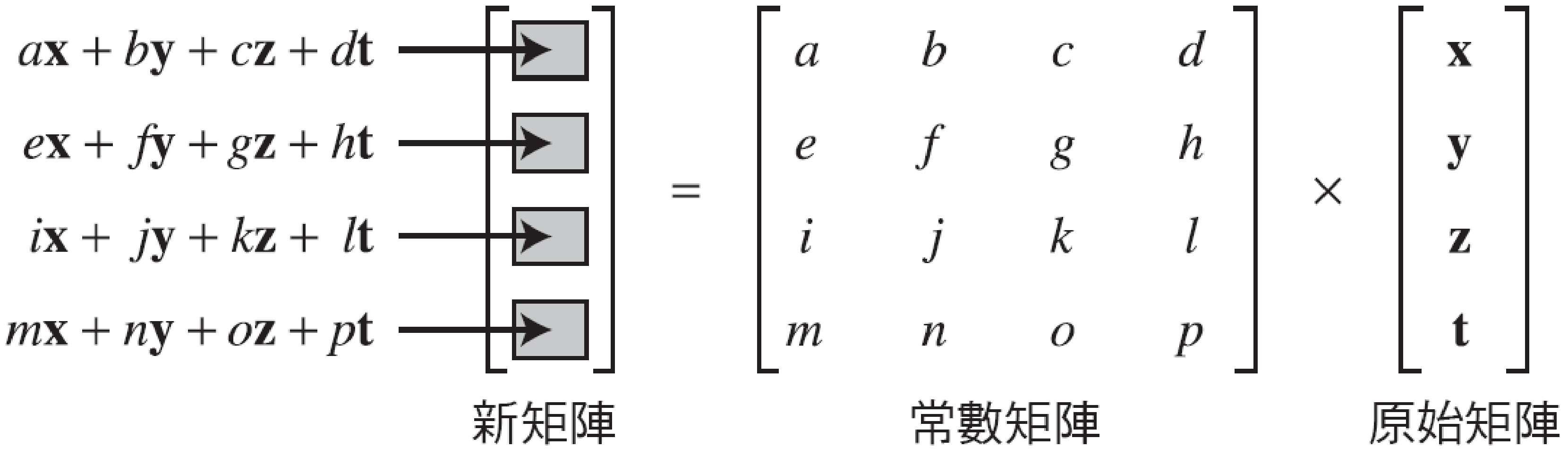


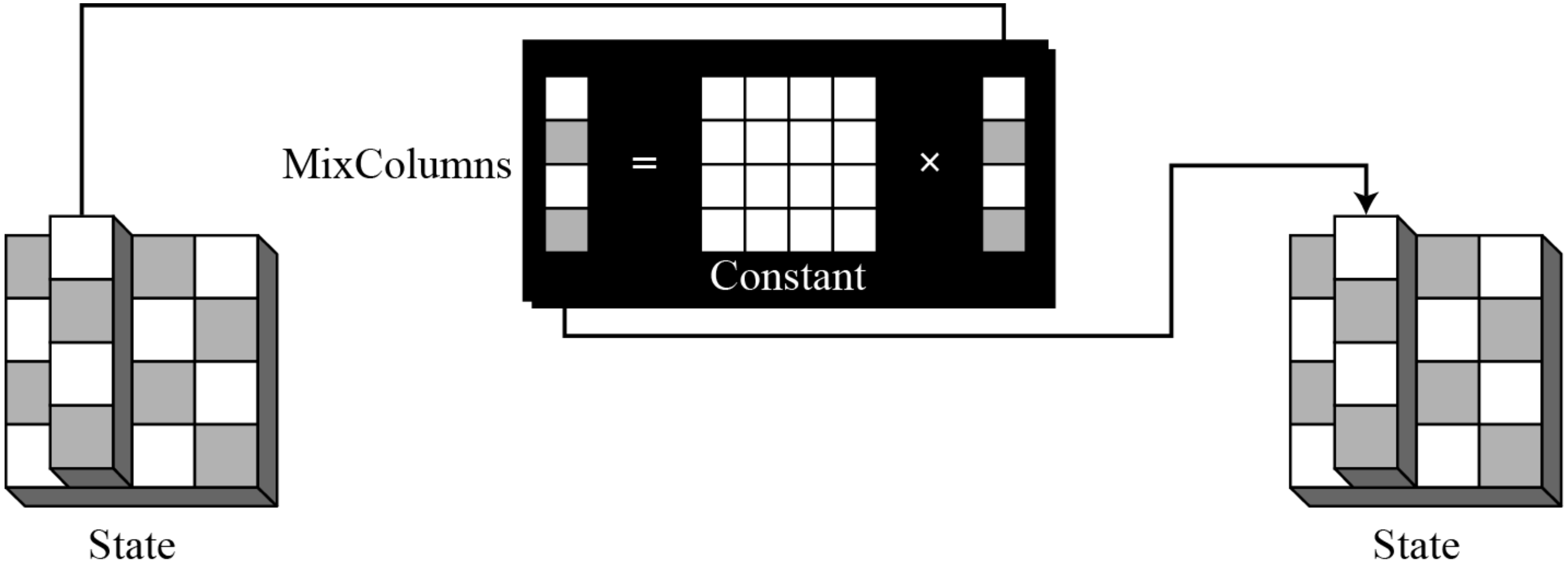
圖 7.12 MixColumns 和 InvMixColumns 所使用的常數矩陣

$$\underbrace{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}}_{\mathbf{C}} \xleftrightarrow{\text{反向}} \underbrace{\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}}_{\mathbf{C}^{-1}}$$

MixColumns

- MixColumns 轉換是以行矩陣為單位來執行；它將每個行矩陣轉換成另一組數值。

圖7.13



MixColumns 之運算 – 舉例

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} \text{D4} \\ \text{BF} \\ \text{5D} \\ \text{30} \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ \text{E5} \end{bmatrix}$$

$$\begin{aligned} & (02)_{16} \times (\text{D4})_{16} + (03)_{16} \times (\text{BF})_{16} + (01)_{16} \times (5\text{D})_{16} + (01)_{16} \times (30)_{16} \\ & = (10110011)_2 + (11011010)_2 + (01011101)_2 + (00110000)_2 \\ & = (00000100)_2 \\ & = (04)_{16} \end{aligned}$$

不可分解多項式的二進制表示：100011011

InvMixColumns

- InvMixColumns 的轉換程序基本上和 MixColumns 一樣。

注意

MixColumns 和 InvMixColumns 轉換互為對方的逆運算。

演算法 7.3 MixColumns 轉換的虛擬程式碼

```
MixColumns (S)
{
     for (c = 0 to 3)
          mixcolumn (s_c)
}
```

```
mixcolumn (col)
{
   CopyColumn (col, t)                                         //t為暫時行
```

$\mathbf{col}_0 \leftarrow (0x02) \bullet \mathbf{t}_0 \oplus (0x03 \bullet \mathbf{t}_1) \oplus \mathbf{t}_2 \oplus \mathbf{t}_3$

$\mathbf{col}_1 \leftarrow \mathbf{t}_0 \oplus (0x02) \bullet \mathbf{t}_1 \oplus (0x03) \bullet \mathbf{t}_2 \oplus \mathbf{t}_3$

$\mathbf{col}_2 \leftarrow \mathbf{t}_0 \oplus \mathbf{t}_1 \oplus (0x02) \bullet \mathbf{t}_2 \oplus (0x03) \bullet \mathbf{t}_3$

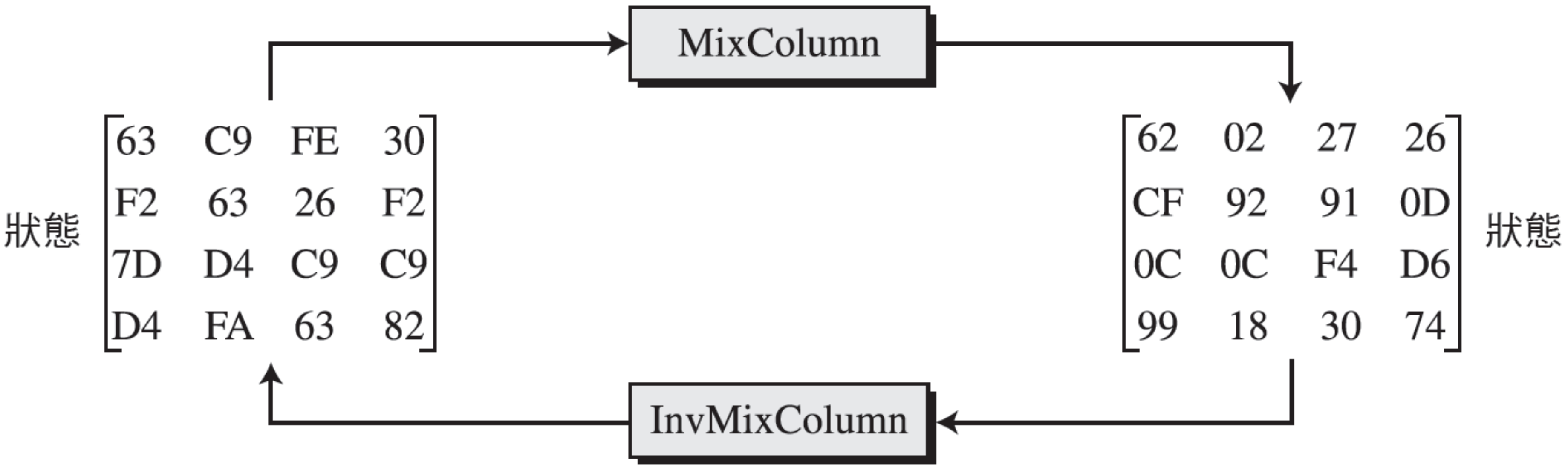
$\mathbf{col}_3 \leftarrow (0x03 \bullet \mathbf{t}_0) \oplus \mathbf{t}_1 \oplus \mathbf{t}_2 \oplus (0x02) \bullet \mathbf{t}_3$

```
}
```

範例 7.5

- 圖 7.14 展示一組狀態如何使用 MixColumns 進行轉換，也展示出 InvMixColumns 轉換可產生原始的狀態。

圖 7.14



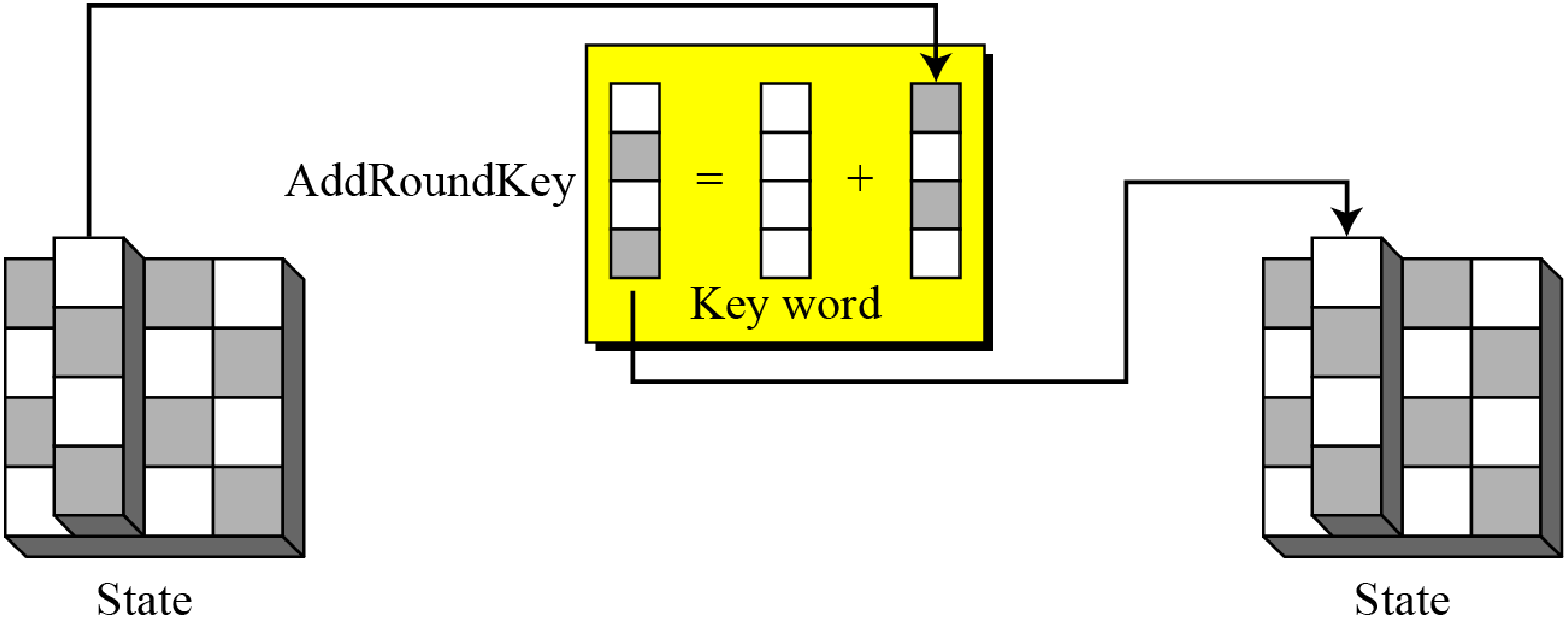
AddRoundKey

- AddRoundKey一次處理一行。AddRoundKey 將回合金鑰加入狀態矩陣行；AddRoundKey 使用矩陣加法。

注意

AddRoundKey 轉換為自己的反向。

圖 7.15 AddRoundKey 轉換



演算法 7.4 AddRoundKey轉換的虛擬碼

AddRoundKey (S)

{

for (c = 0 to 3)

$\mathbf{s}_c \leftarrow \mathbf{s}_c \oplus \mathbf{w}_{\text{round}+4c}$

}

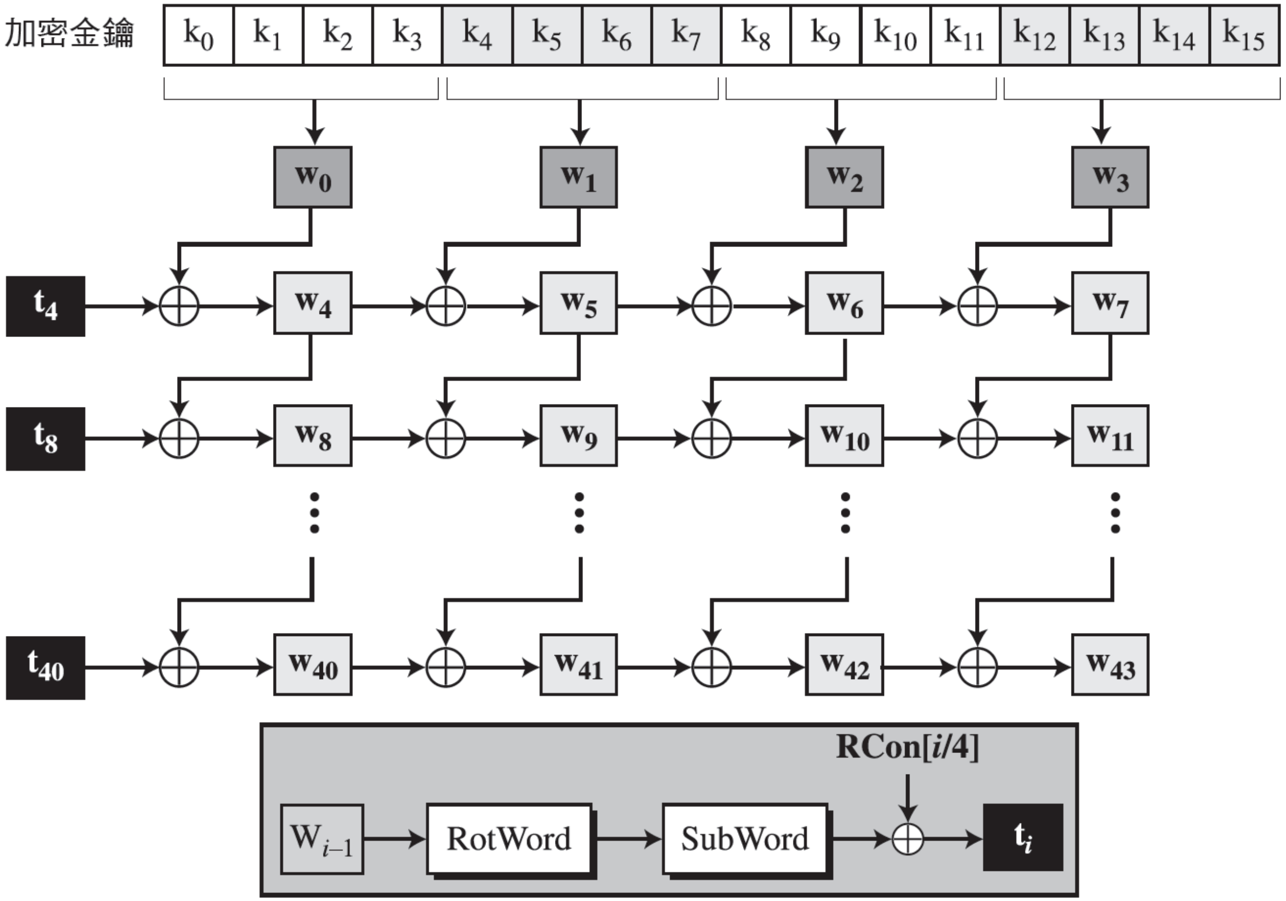
7.3 金鑰擴展

- AES 使用金鑰擴展（Key Expansion）程序來建立每一回合中所使用的回合金鑰。假設回合數為 N_r，則金鑰擴展程序將從一個長度為 128 位元的加密金鑰，產生 $N_r + 1$ 個長度為128 位元的回合金鑰。
- 本節討論主題
  - AES-128 的金鑰擴展程序
  - 金鑰擴展分析
  - AES-192 與 AES-256 的金鑰擴展程序

表 7.3 每一回合的字組

回合	字組			
預先回合	$\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
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	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

7.3.1 AES-128 的金鑰擴展程序



產生 t_i（暫時）字組 $i = 4\ N_r$

RoteWord 與 SubWord

- RoteWord: Return a word in which the bytes are a cyclic permutation of its input

$$(a, b, c, d) \rightarrow (b, c, d, a)$$

- SubWord: Return a 4-byte word in which each byte is the result of applying S-box (表 7.1) to the byte at the corresponding position in the input word

表 7.4 RCon 常數

- $\text{RCon}[i] = (\text{RC}[i], \text{'00'}, \text{'00'}, \text{'00'})$
- $\text{RC}[i]$: Representing an element in $GF(2^8)$ with a value of $x^{(i-1)}$
  - $\text{RC}[1] = 1$, (i.e., '01')
  - $\text{RC}[i] = x \cdot (\text{RC}[i-1]) = x^{(i-1)}$, (i.e., $\text{'02'} \cdot (\text{RC}[i-1])$)

回合	常數（RCon）	回合	常數（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	$(\underline{\mathbf{36}}\ 00\ 00\ 00)_{16}$

回合金鑰

- 金鑰擴展程序在計算字組時，可以選擇使用查表或者在 **GF**(2^8) 體下計算最左邊位元組的值。計算方式如下所列（prime 為一不可分解多項式）：

RC_1	$\rightarrow x^{1-1}$	$=x^0$	mod *prime*	$=1$	$\rightarrow 00000001$	$\rightarrow 01_{16}$
RC_2	$\rightarrow x^{2-1}$	$=x^1$	mod *prime*	$=x$	$\rightarrow 00000010$	$\rightarrow 02_{16}$
RC_3	$\rightarrow x^{3-1}$	$=x^2$	mod *prime*	$=x^2$	$\rightarrow 00000100$	$\rightarrow 04_{16}$
RC_4	$\rightarrow x^{4-1}$	$=x^3$	mod *prime*	$=x^3$	$\rightarrow 00001000$	$\rightarrow 08_{16}$
RC_5	$\rightarrow x^{5-1}$	$=x^4$	mod *prime*	$=x^4$	$\rightarrow 00010000$	$\rightarrow 10_{16}$
RC_6	$\rightarrow x^{6-1}$	$=x^5$	mod *prime*	$=x^5$	$\rightarrow 00100000$	$\rightarrow 20_{16}$
RC_7	$\rightarrow x^{7-1}$	$=x^6$	mod *prime*	$=x^6$	$\rightarrow 01000000$	$\rightarrow 40_{16}$
RC_8	$\rightarrow x^{8-1}$	$=x^7$	mod *prime*	$=x^7$	$\rightarrow 10000000$	$\rightarrow 80_{16}$
RC_9	$\rightarrow x^{9-1}$	$=x^8$	mod *prime*	$=x^4+x^3+x+1$	$\rightarrow 00011011$	$\rightarrow 1B_{16}$
RC_{10}	$\rightarrow x^{10-1}$	$=x^9$	mod *prime*	$=x^5+x^4+x^2+x$	$\rightarrow 00110110$	$\rightarrow 36_{16}$

演算法 7.5 AES-128 金鑰擴展的虛擬程式碼

```
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 為暫時字組
            w_i ← t + w_i−4
        }
    }
}
```

表 7.5 回合金鑰的範例

- 下表顯示使用 $(24\ 75\ A2\ B3\ 34\ 75\ 56\ 88\ 31\ E2\ 12\ 00\ 13\ AA\ 54\ 87)_{16}$ 這個金鑰所產生的十個回合金鑰。

回合	t值	回合中的第一個字組	回合中的第二個字組	回合中的第二個字組	回合中的第四個字組
—		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

7.3.2 金鑰擴展分析

- AES 的金鑰擴展機制設計有幾個特性可以抵抗破密分析。
  - AES 中每個回合金鑰都是由上一個回合金鑰算出來的，然而，因為計算中使用 SubWord 轉換程序，所以回合金鑰之間的關係是非線性的。
  - 加入回合常數的步驟也確保了每個回合金鑰都不會和上一個相同。

範例 7.8

- 即使加密金鑰只差一個位元，所求出來的兩組回合金鑰的差異也很明顯。

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

表 7.6 兩組回合金鑰比較

R.	第一組回合金鑰	第二組回合金鑰	*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

範例 7.9

- 在第六章我們討論過 DES 系統中的弱金鑰，然而這個概念並不適用於 AES。假設加密金鑰的位元全部為 0。以下列出加密過程中數個回合的字組：

預先回合：	00000000	00000000	00000000	00000000
第一回合：	62636363	62636363	62636363	62636363
第二回合：	9B9898C9	F9FBFBAA	9B9898C9	F9FBFBAA
第三回合：	90973450	696CCFFA	F2F45733	0B0FAC99
…	...	...	...	...
第十回合：	B4EF5BCB	3E92E211	23E951CF	6F8F188E

範例 7.9 (續)

- 預先回合和第一回合的字組都完全相同。在第二回合時，第一個字組和第三個相同，而第二個字組和第四個相同。然而，從第二回合以後所有的字組就都完全不一樣了。

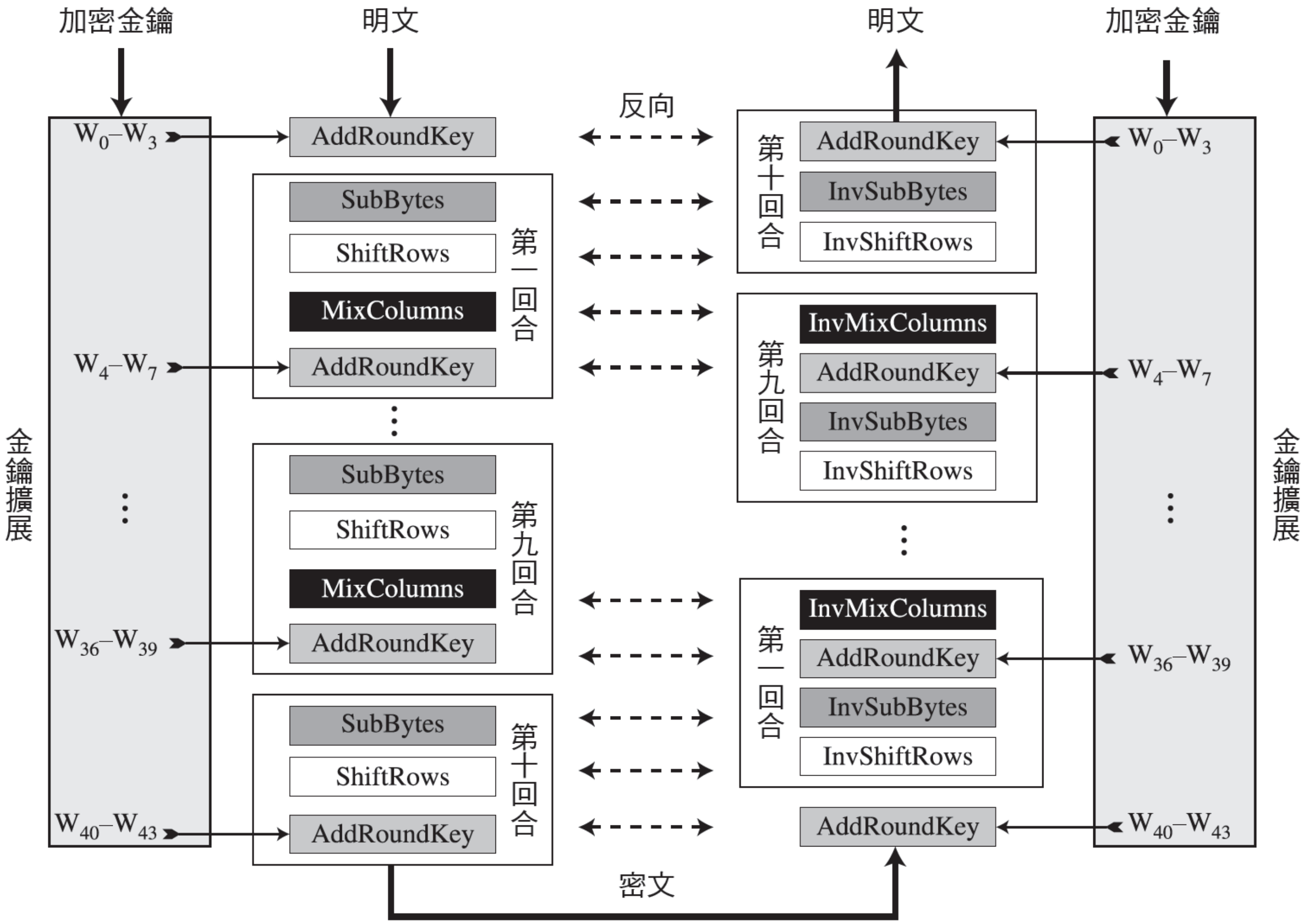
7.3.3 AES-192 和 AES-256 的金鑰擴展程序

- AES-192 和 AES-256 這兩個版本的金鑰擴展程序和 AES-128 非常相近，不同的地方只有以下幾點：
  - 在 AES-192 中，一次產生六個字組而非四個。
    - 使用加密金鑰產生前六個字組（$\mathbf{w}_0$ 至 $\mathbf{w}_5$）。
    - 當 $i \bmod 6 \neq 0$ 時，$\mathbf{w}_i \leftarrow \mathbf{w}_{i-1} + \mathbf{w}_{i-6}$；否則，$\mathbf{w}_i \leftarrow \mathbf{t} + \mathbf{w}_{i-6}$。
  - 在 AES-256 中，一次產生八個字組而非四個。
    - 使用加密金鑰產生前八個字組（$\mathbf{w}_0$ 至 $\mathbf{w}_7$）。
    - 當 $i \bmod 8 \neq 0$ 時，$\mathbf{w}_i \leftarrow \mathbf{w}_{i-1} + \mathbf{w}_{i-8}$；否則，$\mathbf{w}_i \leftarrow \mathbf{t} + \mathbf{w}_{i-8}$。
    - 當 $i \bmod 4 = 0$ 且 $i \bmod 8 \neq 0$ 時，$\mathbf{w}_i = \text{SubWord}(\mathbf{w}_{i-1}) + \mathbf{w}_{i-8}$。

7.4 加密法

- 現在我們來看看 AES 如何在加密和解密的過程中使用以上所介紹的四種轉換。在發布的標準文件中，加密演算法稱為加密法 (Cipher)，而解密演算法則稱為反向加密法 (Inverse Cipher)。
- 本節討論主題
  - 原始設計
  - 替代型設計

圖 7.16 原始設計的加密法與反向加密法



演算法

- AES-128 版本的程式碼如演算法 7.6 所示。

```
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);
}
```

圖 7.17 SubBytes 與 ShiftRows 組合的可逆性

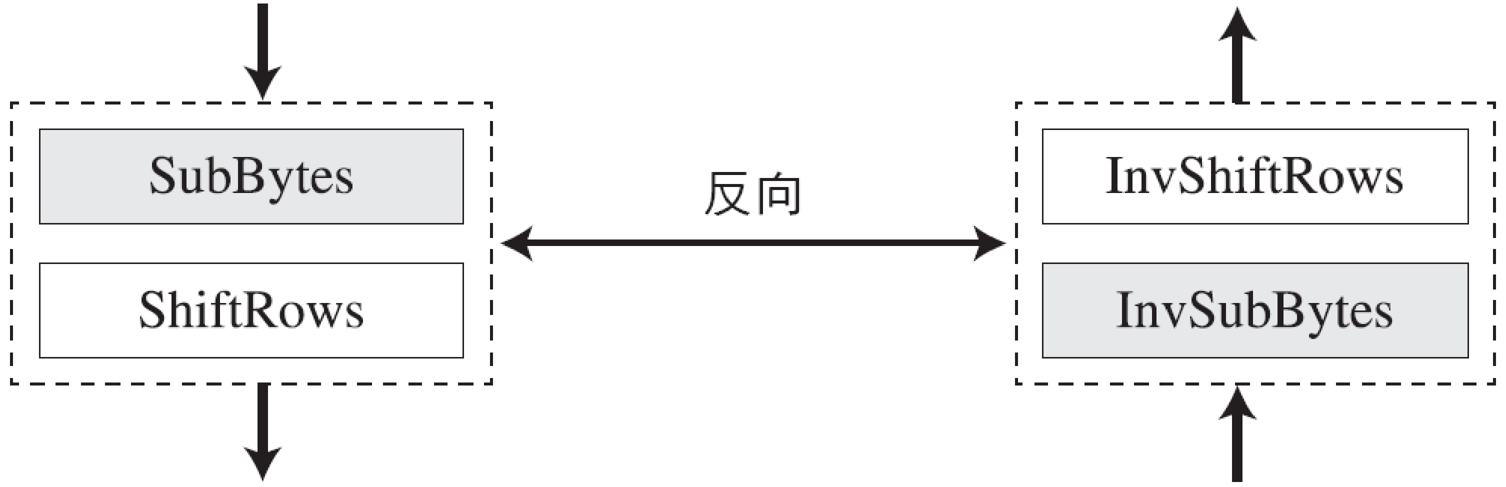


圖 7.18 MixColumns 與 AddRoundKey 組合的可逆性

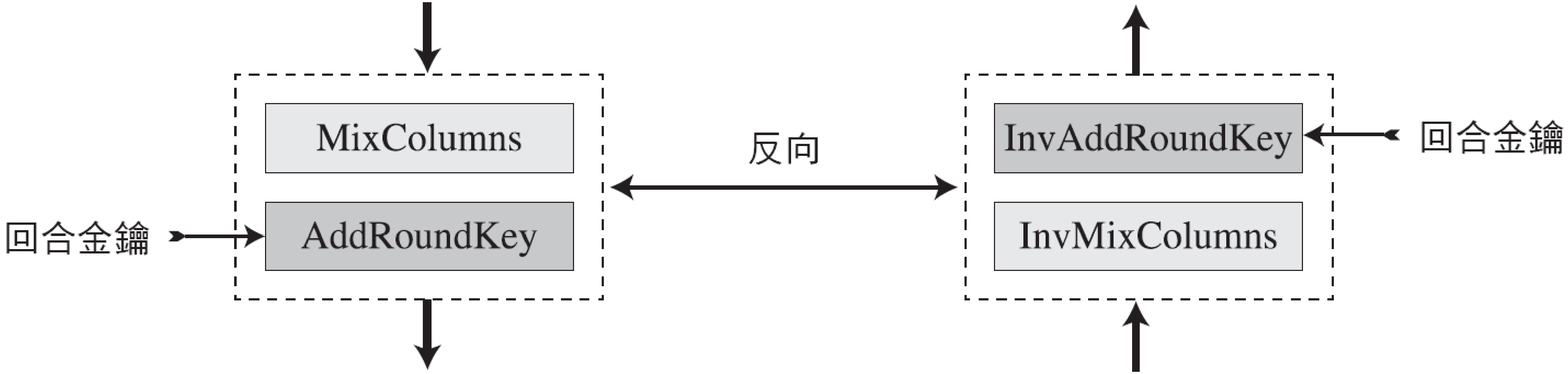
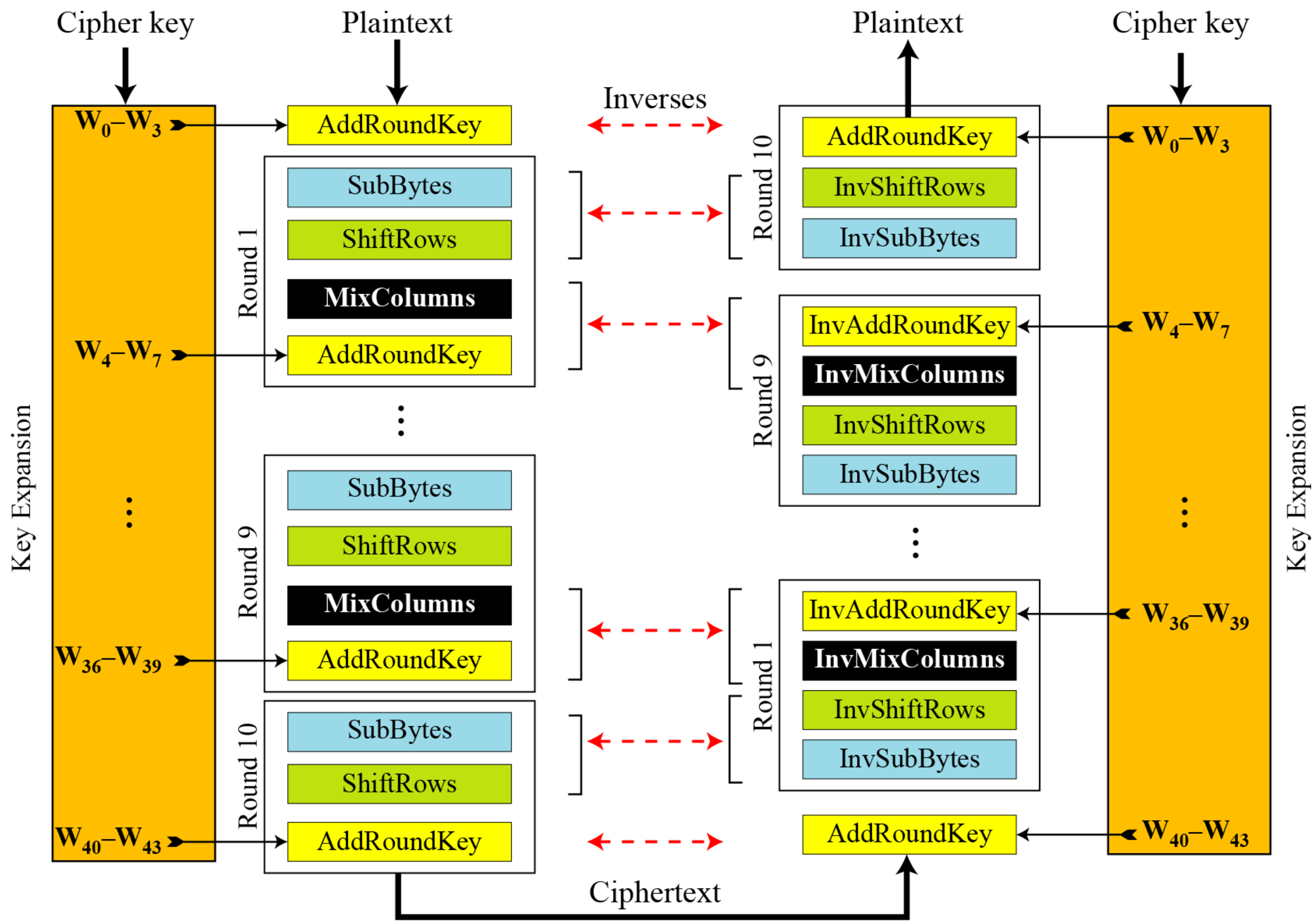


圖 7.19 替代型設計的加密法與反向加密法



改變金鑰擴展演算法

- 若反向加密法不使用 InvRoundKey 轉換，我們可以透過修改金鑰擴展演算法來產生另一組回合金鑰供反向加密法使用。

7.5 金鑰擴展與加密的範例

- 在這一節中，我們提供一些加密／解密和金鑰產生的範例，以強調前面兩節所介紹的概念

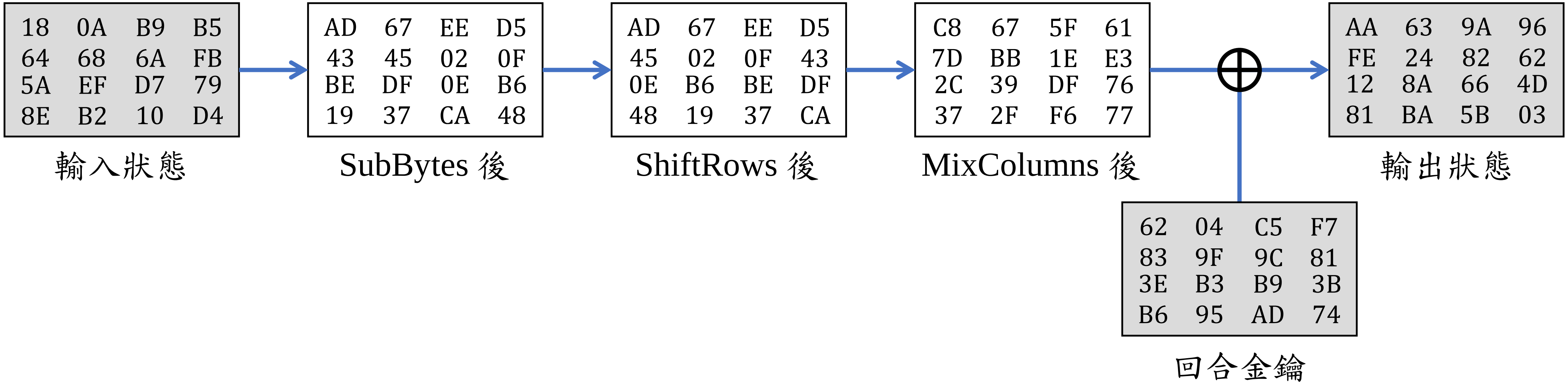
範例 7.10

- 以下是使用一個隨機選擇加密金鑰將明文加密後產生的密文區塊。

明文：	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19
加密金鑰：	24	75	A2	B3	34	75	56	88	31	E2	12	00	13	AA	54	87
密文：	BC	02	8B	D3	E0	E3	B1	95	55	0D	6D	FB	E6	F1	82	41

範例 7.11

- 下圖展示 AES 第七回合的狀態值變化（假設使用以下的輸入狀態搭配表 7.5 的第七把回合金鑰）。



範例 7.12

- 也許有人會對全部由位元 0 所組成的明文經過加密的結果感到好奇。以下的密文便是使用範例 7.10 的金鑰加密後之結果。

明文：	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
加密金鑰：	24	75	A2	B3	34	75	56	88	31	E2	12	00	13	AA	54	87
密文：	63	2C	D4	5E	5D	56	ED	B5	62	04	01	A0	AA	9C	2D	8D

範例 7.13

- 現在來檢查一下第六章討論過的崩塌影響。我們只修改明文的一個位元，然後比較加密後的結果。以下可以看到，光是修改明文的最後一個位元，對於密文的擴散和混淆的效果就非常明顯了。明文修改後的密文和原來的密文相較，可以看出有許多位元都不一樣。

明文1：	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
明文2：	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01
密文1：	63	2C	D4	5E	5D	56	ED	B5	62	04	01	A0	AA	9C	2D	8D
密文2：	26	F3	9B	BC	A1	9C	0F	B7	C7	2E	7E	30	63	92	73	13

範例 7.14

- 以下展示使用全部由位元 0 所組成的金鑰之加密效果。

明文：	00	04	12	14	12	04	12	00	0c	00	13	11	08	23	19	19
金鑰：	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
密文：	5A	6F	4B	67	57	B7	A5	D2	C4	30	91	ED	64	9A	42	72

7.6 AES 安全性分析

- 本節討論主題
  - 安全性
  - 實現性
  - 簡單性與成本
  - 3DES vs. AES

7.6.1 安全性

- AES 本來就是設計來取代 DES 的，所以 AES 幾乎已經測試過所有在 DES 上見過的攻擊，目前尚未發現其中有任何一種方法可以破壞 AES 的安全性。
- SubBytes
  - 提供了加密法非線性的變換能力
- ShiftRow 與 MixColumns
  - 提供了擴散性：每個 input byte 都會對 output byte 造成影響
- AddRoundKey
  - AES 沒有弱金鑰，兩個加密金鑰就算只差 1 bit，金鑰的擴展程序最後會變成完全兩個不同的金鑰

7.6.1 安全性 (續)

- 暴力攻擊 (Brute-Force Attack)
  - 單就金鑰長度來看，AES 裡面最少 128 位元的金鑰絕對比 DES 的 56 位元金鑰要安全得多，需要測試 2^{128} 次。
- 統計攻擊 (Statistical Attack)
  - AES 的轉換提供足夠的擴散和混淆，已經有很多的測試都無法對 AES 所產生的密文進行統計攻擊。
- 差異攻擊與線性攻擊 (Differential and Linear Attacks)
  - AES 系統設計者早已考慮此等攻擊，目前仍然沒有任何已知的差異攻擊或者線性攻擊存在。

7.6.2 實現性

- AES 的設計非常具有彈性，可以在軟體、硬體和韌體上實作，並利用查表或是使用完整定義的算術結構。

7.6.3 簡單性與成本

- AES 設計所使用的演算法都非常簡單，可以很容易地在非常便宜的處理器和非常小的記憶體條件下實作。

3DES vs. AES

	3DES	AES
明文輸入	64 bits	64 bits
密鑰長度	112 = 56 × 2、168 = 56 × 3 (bits)	128、192、256 (bits)
加密回合數	48	10、12、14
加密速度	慢	快

3DES vs. AES (續)

- 輸入明文：256 MB
- 作業系統：Windows
- 假設以每秒 255 個密鑰進行暴力破解

	3DES	AES
加密時間 (秒)	12	5
平均處理資料量 (MB/s) (approx.)	12	51.2
破解時間	46 億年	1490000 億年