Lecture 6
Data Encryption Standard

Jason Lin

學習目標

- 回顧 DES 的發展歷史
- 定義 DES 的基本結構
- 描述 DES 建構元件的詳細情形
- 描述回合金鑰產生程序
- 分析 DES

6.1 簡介

- 資料加密標準（Data Encryption Standard, DES）是一個對稱式金鑰區塊加密法，由美國國家標準技術局（National Institute of Standards and Technology, NIST）發表
- 本節討論主題
  - 歷史
  - 概述

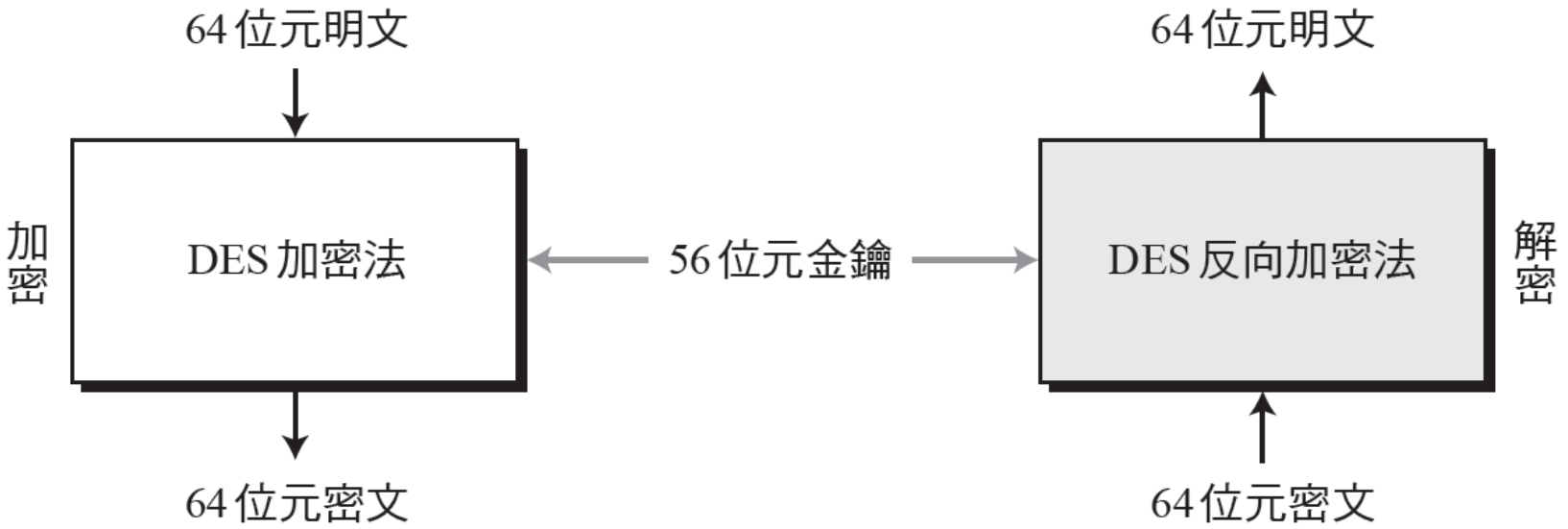
6.1.1 歷史

- 在 1973 年，NIST 發布一個國家對稱式金鑰系統的需求提案。一個由 IBM 所修改的 Lucifer 計畫被接受成為 DES
- DES 於 1975 年 3 月發表在《聯邦公報》（*Federal Register*）上而成為聯邦資訊處理標準（Federal Information Processing Standard，FIPS）的草案
- DES 於 1976 年 11 月被確定為聯邦標準，並在 1977 年 1 月作為 FIPS PUB 46 發布
- 由於 DES 在加密時有安全性上的問題，所以 NIST 將其授權應用於非機密資料的標準，並於 1999 年發布 FIPS-46-3，建議在未來的應用上使用 3-DES (重複 DES 的加密 3 次)

6.1.1 概述

- DES 是一個區塊加密法，如下圖 6.1 所示

圖 6.1



6.2 DES 的結構

- 加密程序由兩個排列（P-box，初始排列與最終排列）以及十六個 Feistel 回合所組成
- 本節討論的主題
  - 初始排列與最終排列
  - Feistel 回合
  - 加密法與解密法
  - 範例

圖 6.2 DES 的一般結構

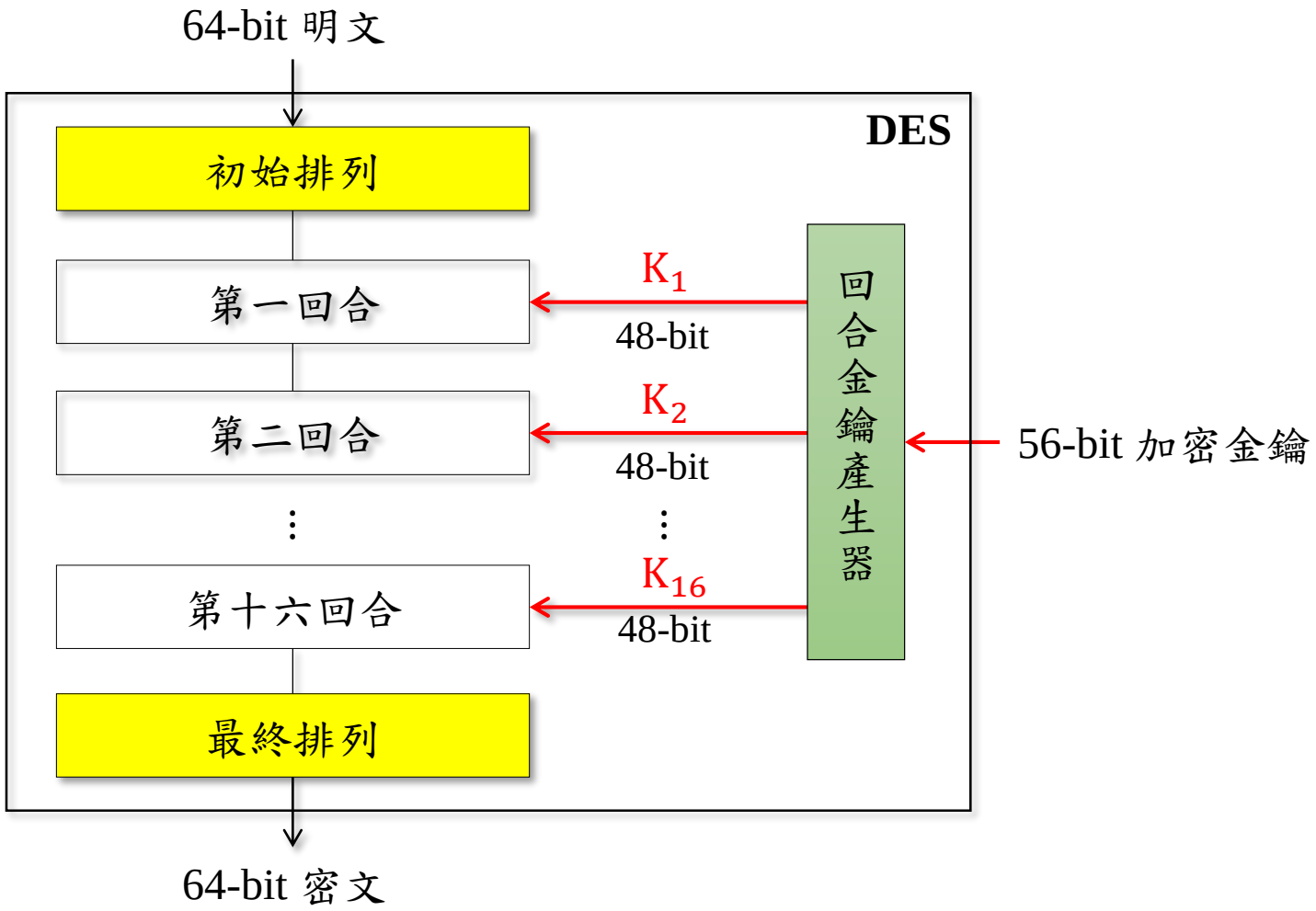
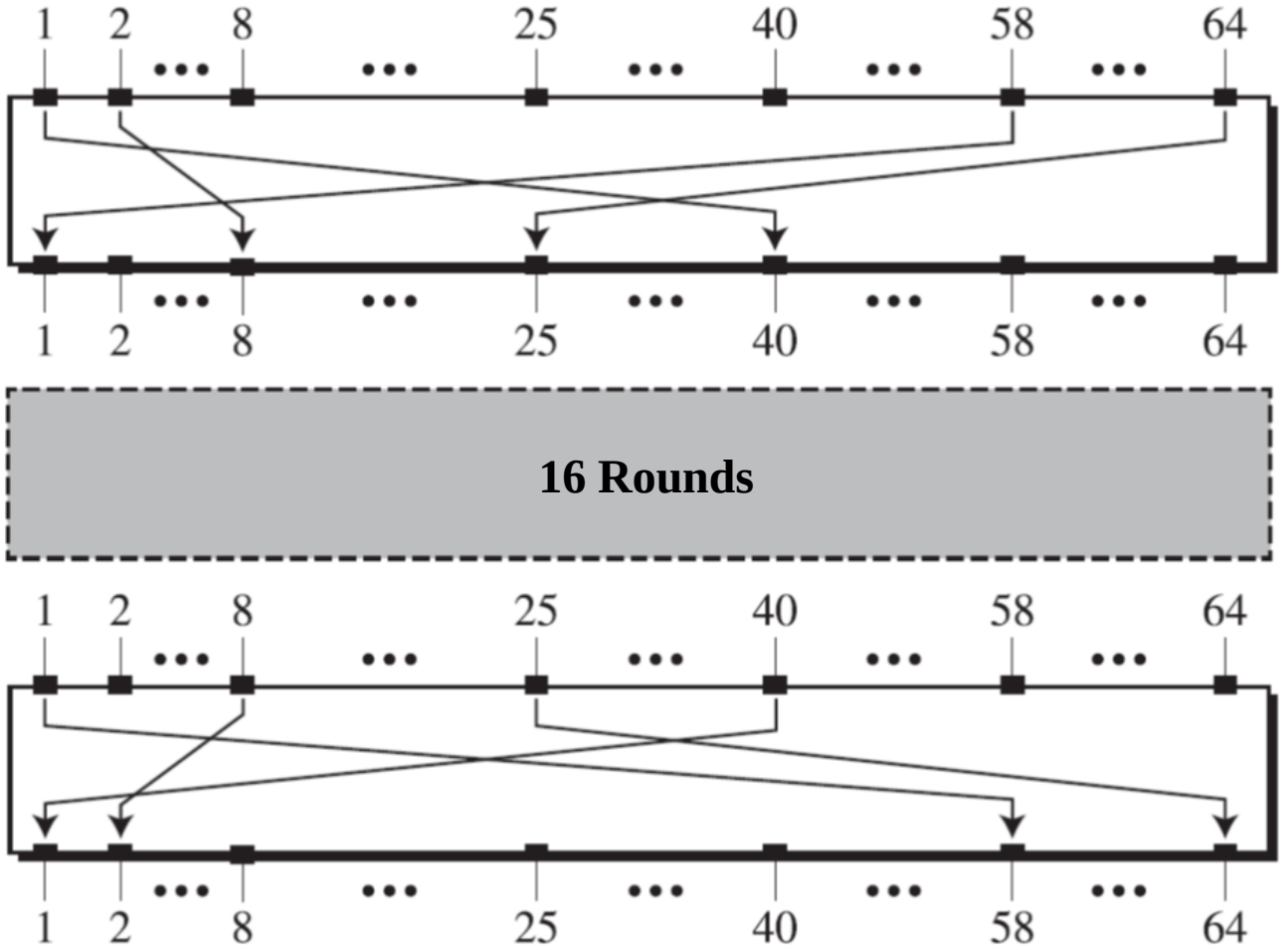


圖 6.3 DES 初始排列與最終排列的步驟



Initial Permutation

Final Permutation

表 6.1 初始排列與最終排列表

Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

範例 6.1

- 找出初始排列的輸出結果，假設輸入（由左到右）以十六進位表示如下：

0x0002 0000 0000 0001

- 解法：輸入僅有兩個為 1 的位元（第 15 個位元及第 64 個位元），因此輸出必定也只有兩個位元為 1（標準排列的性質）。使用表 6.1，我們可以找到這兩個位元的相對輸出。輸入的第 15 個位元將變為輸出的第 63 個位元；輸入的第 64 個位元將變為輸出的第 25 個位元。亦即輸出只有兩個 1，分別在第 25 個位元及第 63 個位元。以十六進位表示如下：

0x0000 0080 0000 0002

範例 6.2

- 假設輸入如下，請找出最終排列之輸出，以證明初始與最終排列互為反向

0x0000 0080 0000 0002

- 解法：只有第 25 個位元及第 63 個位元為 1，其餘為 0。在最終排列中，輸入的第 25 個位元將變為輸出的第 64 個位元，而輸入的第 63 個位元將變為輸出的第 15 個位元。因此結果為

0x0002 0000 0000 0001

初始排列與最終排列

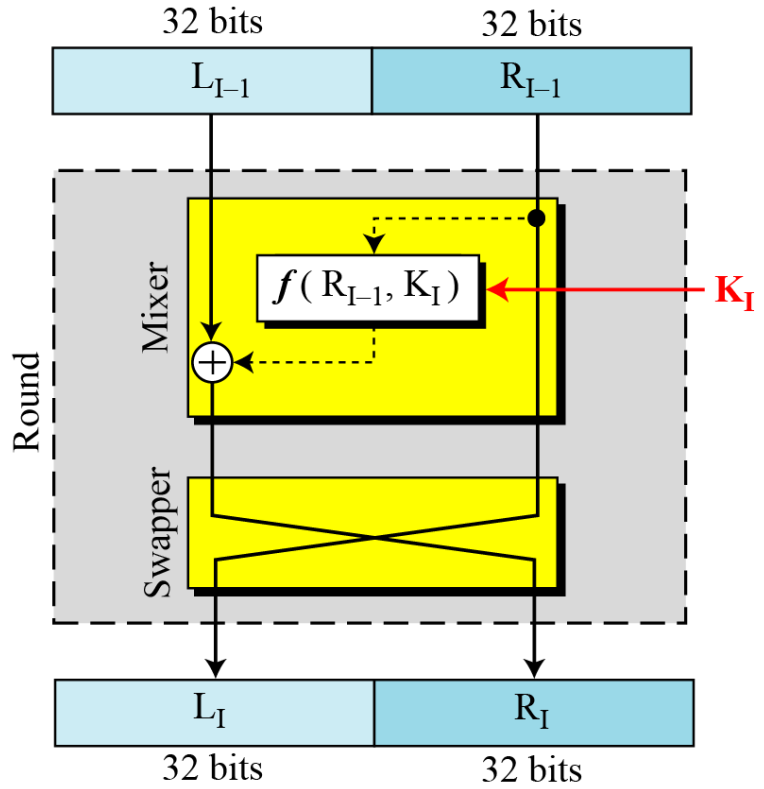
注意

初始排列與最終排列皆為標準的 P-box 且互為反向。它們在 DES 中均與密碼學無太大關係。

6.2.1 Feistel 回合

- DES 使用十六個回合，每一個回合是一個 Feistel 加密法，如下圖 6.4 所示

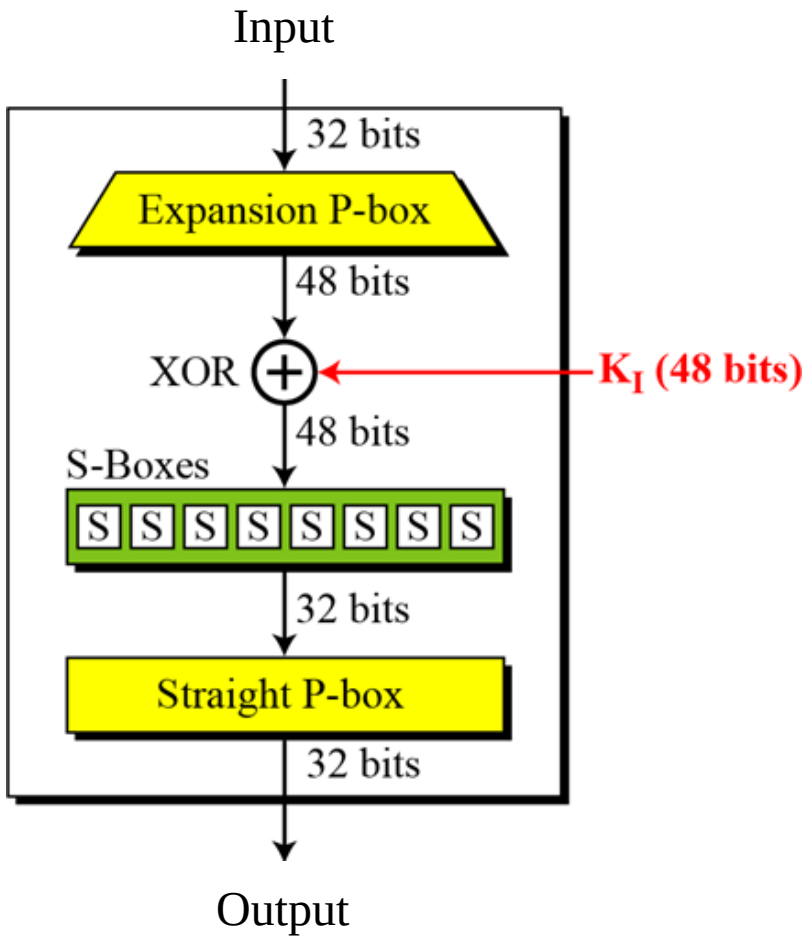
圖 6.4



DES 函數

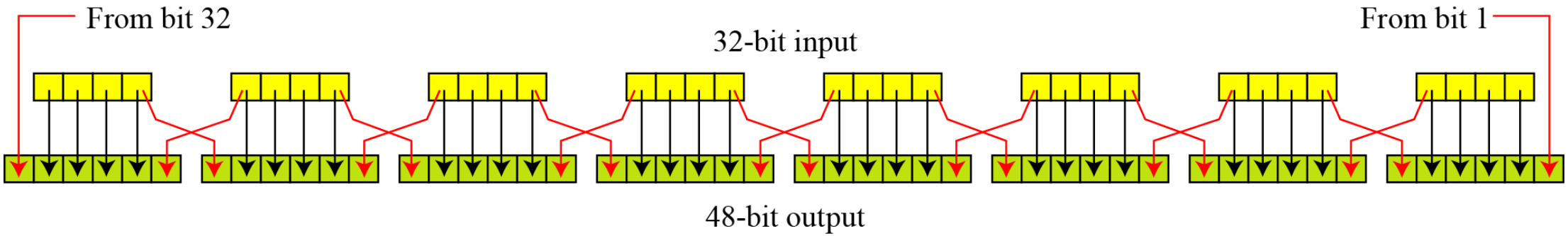
- DES 的核心為 DES 函數
- DES 函數在最右邊的 32 位元（R_{I-1}）上運用一個 48 位元的金鑰，以產生一個 32 位元的輸出

$f(R_{I-1}, K_I)$



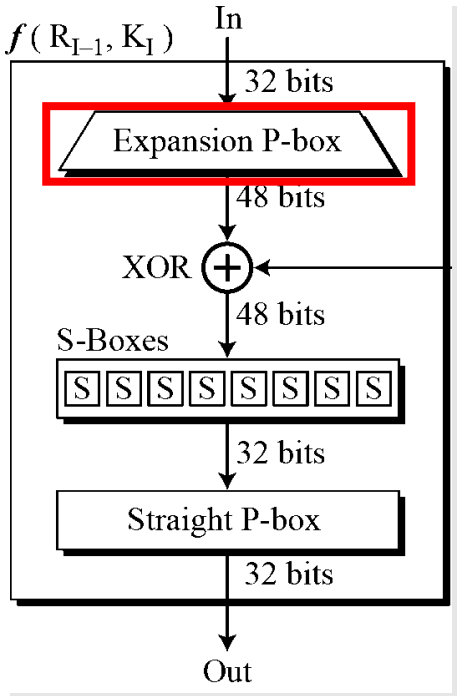
Expansion P-box

- 因為 R_{I-1} 是一個 32 位元輸入且 K_I 是一個 48 位元金鑰，一開始需要先將 R_{I-1} 擴展到 48 位元



擴展的 P-box (續)

- 雖然輸入與輸出的關係可用數學方式來定義，但 DES 採用下表來定義這個 P-box



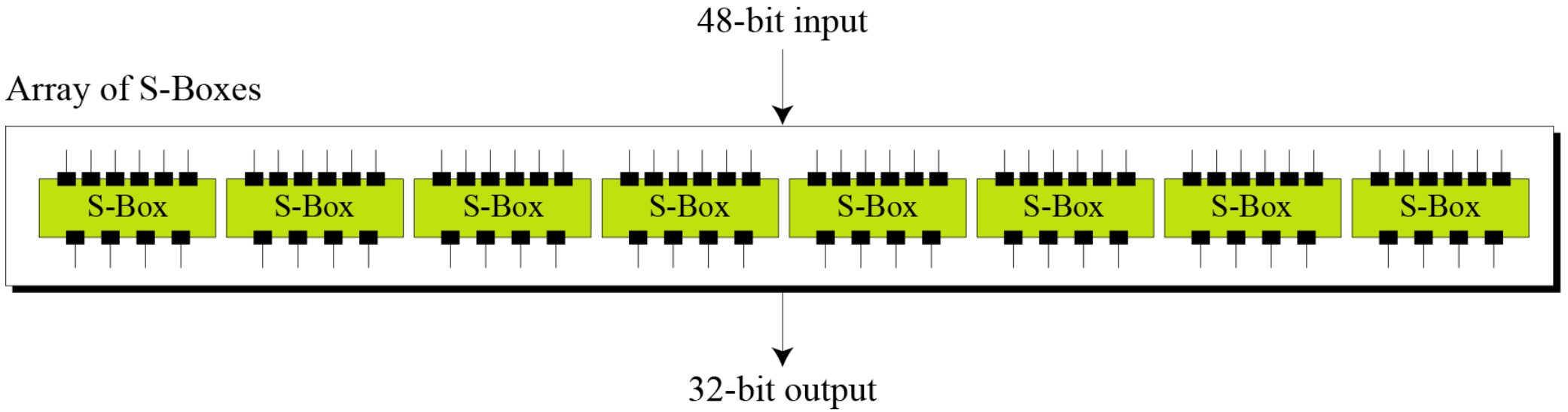
32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

漂白器（XOR）

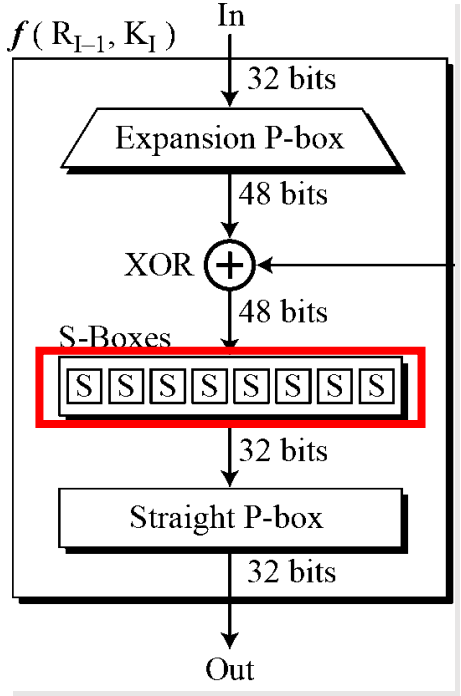
- 在擴展排列之後，DES 將擴展的右半部分與回合金鑰進行 XOR 運算
- 注意右半部與金鑰長度均為 48 位元，而且回合金鑰僅使用在這個運算上

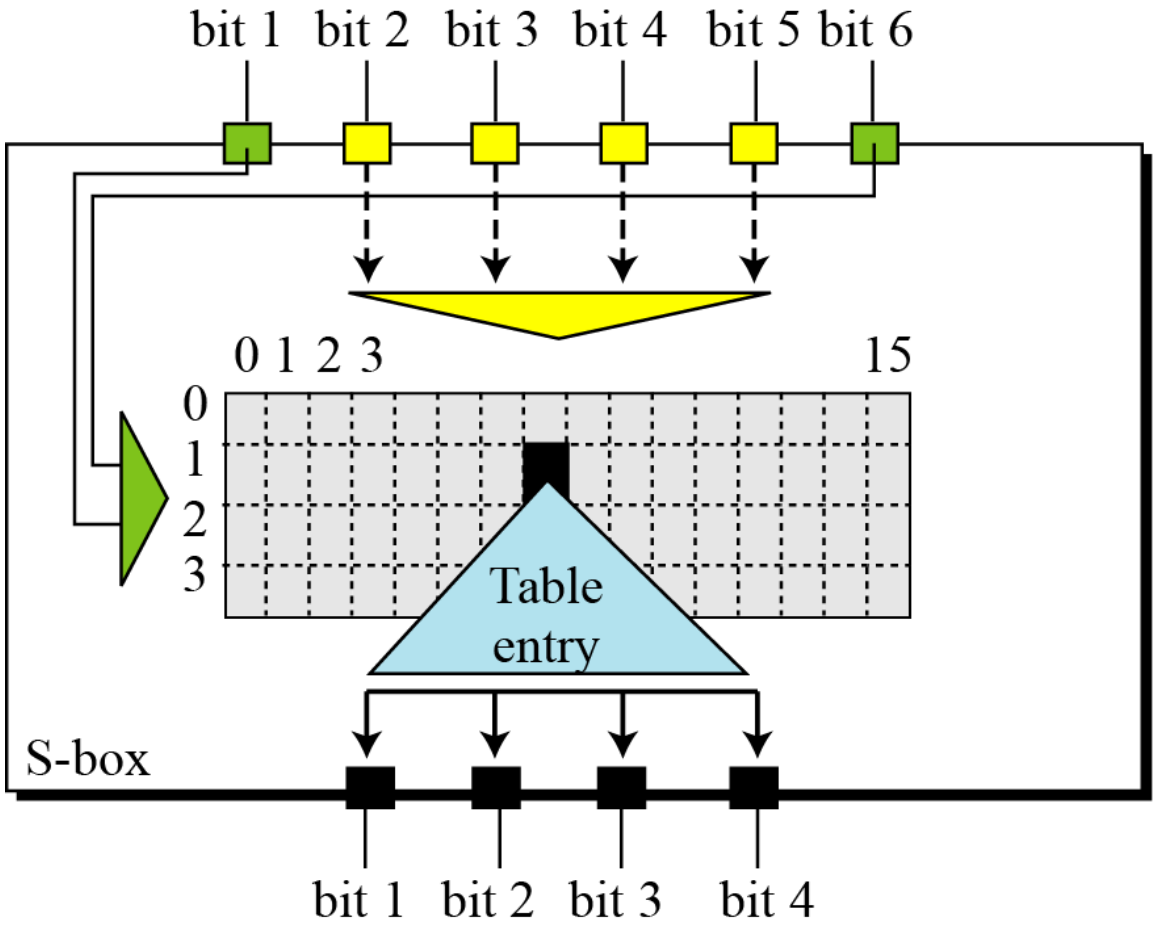
S-box

- S-box 進行實際的混淆（Confusion）
- DES 使用 8 個 S-box，每一個 S-box 有 6 位元 的輸入及 4 位元的輸出，如下圖所示



一種 S-box 的規則


f (R I–1, K I)
In
32 bits
Expansion P-box
48 bits
XOR
48 bits
S-Boxes
S S S S S S S S
32 bits
Straight P-box
32 bits
Out


bit 1 bit 2 bit 3 bit 4 bit 5 bit 6
0 1 2 3
15
0
1
2
3
Table
entry
S-box
bit 1 bit 2 bit 3 bit 4

一種 S-box 的規則 (續)

- 下表顯示所舉 S-box 規則的排列

表 6.3

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

範例 6.3

- 若 S-box 的輸入為 100011，其輸出為何？
- 解法：若將第 1 位元及第 6 位元寫在一起，以二進位表示為 11，以十進位表示為 3。剩下的位元 為 0001，以十進位表示為 1。我們查表 6.3（S-box）的第 3 列與第 1 行，結果為 12（十進位），以 二進位表示為 1100。因此若輸入為 100011，則輸出為 1100

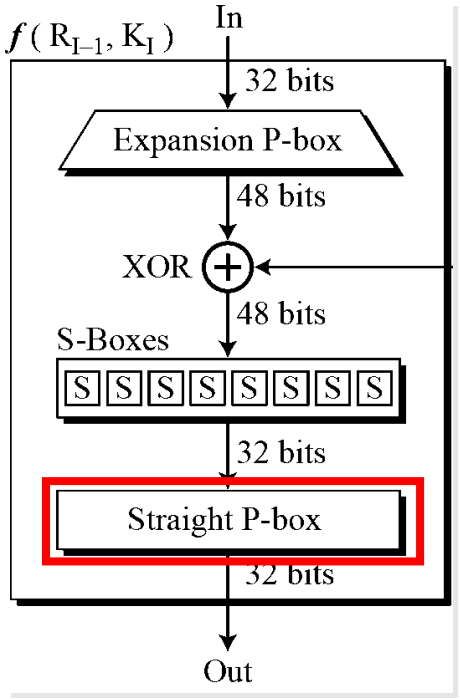
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

範例 6.4

- 若 S-box 的輸入為 **0**00000**0**，其輸出為何？
- 解法：若將第 1 位元及第 6 位元寫在一起，以二進位表示為 00，以十進位表示為 0。剩下的位元 為 0000，以十進位表示為 0。我們查表 6.3（S-box）的第 0 列與第 0 行，結果為 14（十進位），以二進位表示為 1110。因此，若輸入為 000000，則輸出為 1110

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

表 6.11 Straight P-box



16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

6.2.3 加密法與解密法

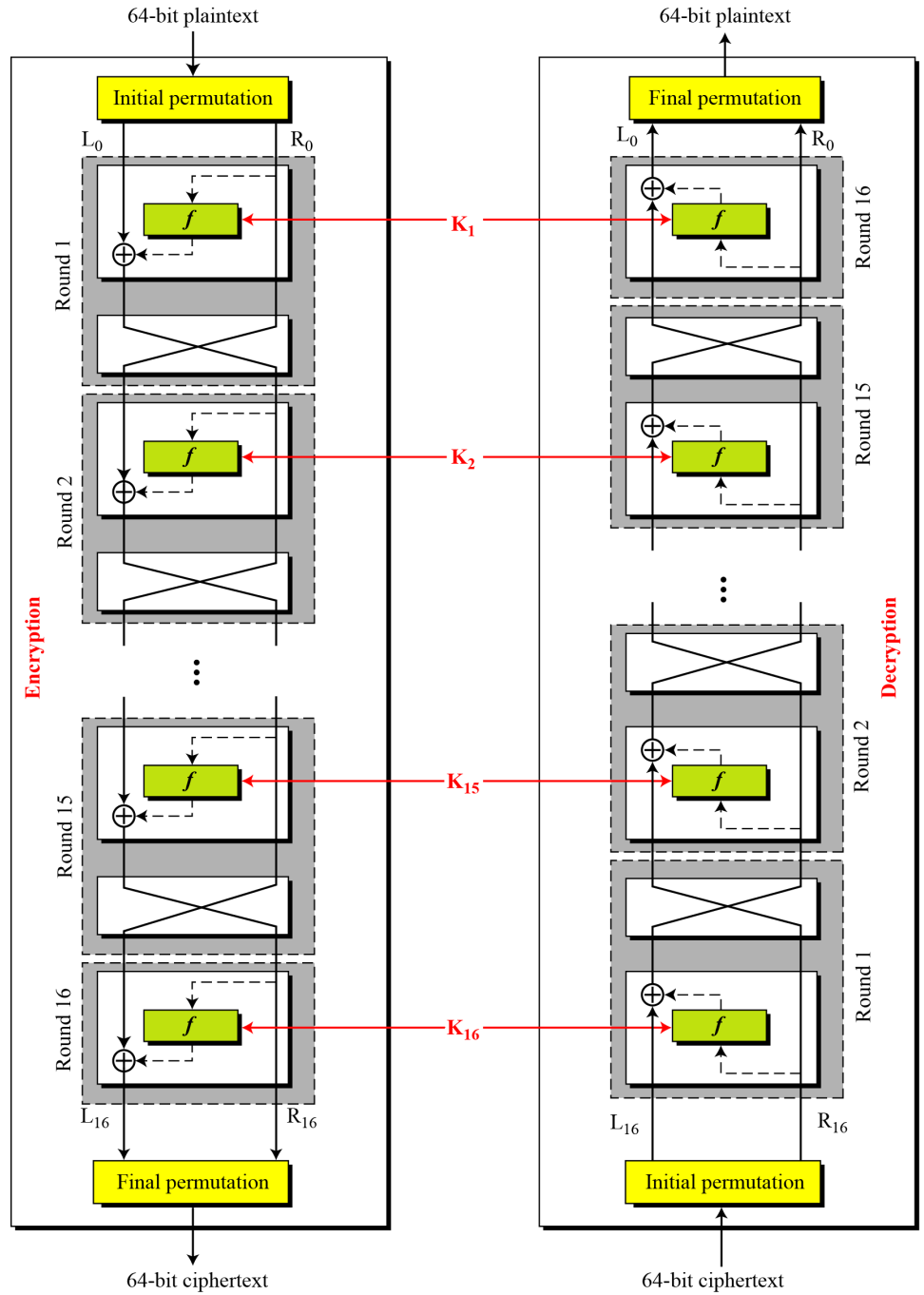
- 使用混合器與交換器可以建立加密法與解密法，均有十六個回合。整個想法是讓加密法與解密法的演算法變得十分類似，將有助於硬體上的實現

DES 加密法與解密法的第一種方式

- 為達到上述目的，第一種方式即是讓最後一個回合（第十六回合）與其他回合不同；其僅有一個混合器而無交換器

注意

在第一種方式中，最後一個回合沒有交換器。


64-bit plaintext
Initial permutation
L_0
R_0
Round 1
Round 2
Round 15
Round 16
f
K_1
K_2
K_{15}
K_{16}
Encryption
L_{16}
R_{16}
Final permutation
64-bit ciphertext
64-bit plaintext
Final permutation
L_0
R_0
Round 16
Round 15
Round 2
Round 1
Decryption
L_{16}
R_{16}
Initial permutation
64-bit ciphertext

演算法 6.1 DES 加密法的虛擬碼

```
Cipher (plainBlock[64], RoundKeys[16, 48], cipherBlock[64])
{
    permute (64, 64, plainBlock, inBlock, InitialPermutationTable)
    split (64, 32, inBlock, leftBlock, rightBlock)
    for (round = 1 to 16)
    {
        mixer (leftBlock, rightBlock, RoundKeys[round])
        if (round!=16) swapper (leftBlock, rightBlock)
    }
    combine (32, 64, leftBlock, rightBlock, outBlock)
    permute (64, 64, outBlock, cipherBlock, FinalPermutationTable)
}
```

演算法 6.1 DES 加密法的虛擬碼 (續)

```
mixer (leftBlock[48], rightBlock[48], RoundKey[48])
{
    copy (32, rightBlock, T1)
    function (T1, RoundKey, T2)
    exclusiveOr (32, leftBlock, T2, T3)
    copy (32, T3, rightBlock)
}

swapper (leftBlock[32], rigthBlock[32])
{
    copy (32, leftBlock, T)
    copy (32, rightBlock, leftBlock)
    copy (32, T, rightBlock)
}
```

演算法 6.1 DES 加密法的虛擬碼 (續)

```
function (inBlock[32], RoundKey[48], outBlock[32])
{
    permute (32, 48, inBlock, T1, ExpansionPermutationTable)
    exclusiveOr (48, T1, RoundKey, T2)
    substitute (T2, T3, SubstituteTables)
    permute (32, 32, T3, outBlock, StraightPermutationTable)
}
```

演算法 6.1 DES 加密法的虛擬碼 (續)

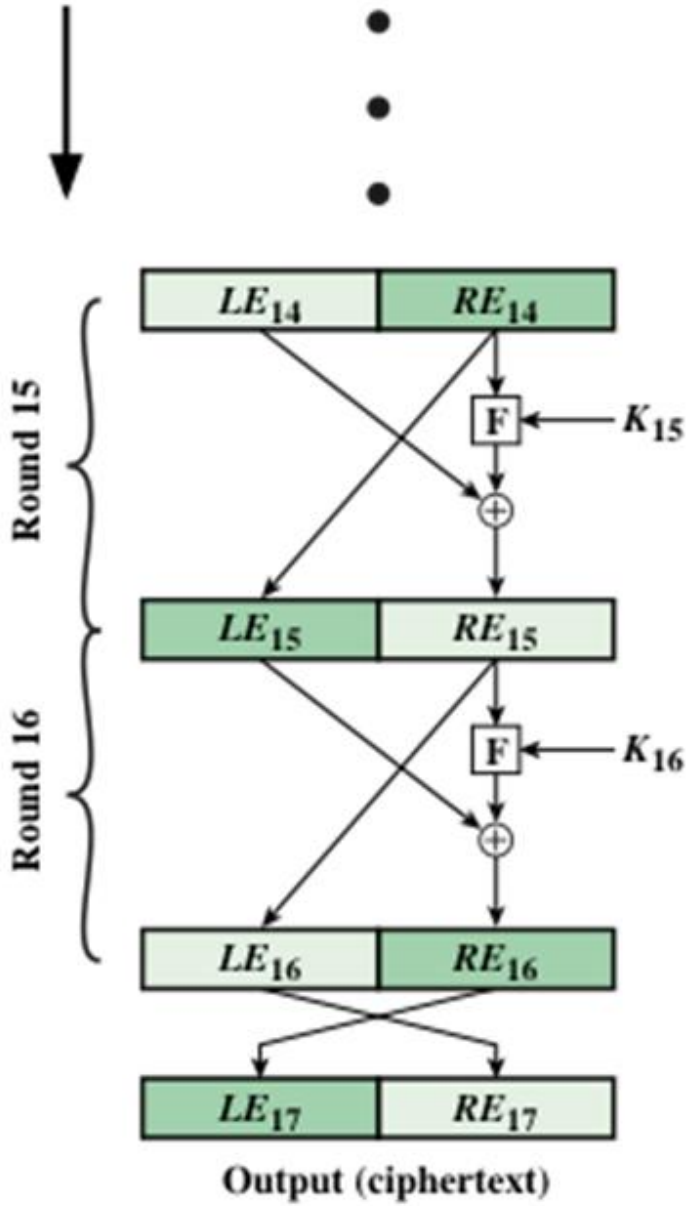
```
substitute (inBlock[32], outBlock[48], SubstitutionTables[8, 4, 16])
{
    for (i = 1 to 8)
    {
        row ← 2 × inBlock[i × 6 + 1] + inBlock [i × 6 + 6]
        col ← 8 × inBlock[i × 6 + 2] + 4 × inBlock[i × 6 + 3] +
              2 × inBlock[i × 6 + 4] + inBlock[i × 6 + 5]

        value = SubstitutionTables [i][row][col]

        outBlock[[i × 4 + 1] ← value / 8;      value ← value mod 8
        outBlock[[i × 4 + 2] ← value / 4;      value ← value mod 4
        outBlock[[i × 4 + 3] ← value / 2;      value ← value mod 2
        outBlock[[i × 4 + 4] ← value
    }
}
```

另一種方式

- 我們可以在第十六回合中包含交換器，然後在其後加入一個額外的交換器（兩個交換器的效果相互抵銷）



回合金鑰的產生

- 雖然給定的金鑰表面上是 64 位元，但只有其中的 56 位元被實際用於演算法，其餘 8 位元會被用於奇偶校驗，並在演算法中被丟棄。因此，DES 的有效加密金鑰長度僅為 56 位元。
- 這裡回合金鑰產生器（Round-Key Generator）建立 16 個 48 位元的回合金鑰（Round Key），而這些回合金鑰是從 56 位元的加密金鑰而來。

- DES 使用 64 位元的金鑰，以 8 個 8 位元的位元組為金鑰的內容，其中每個位元組的第 8 個位元做為同位位元
- 同位位元（Parity Bit）是用來表示一個給定的二進位數中 1 的個數為奇數還是偶數的檢查位元（Check Bit），它是最簡單的錯誤檢測碼

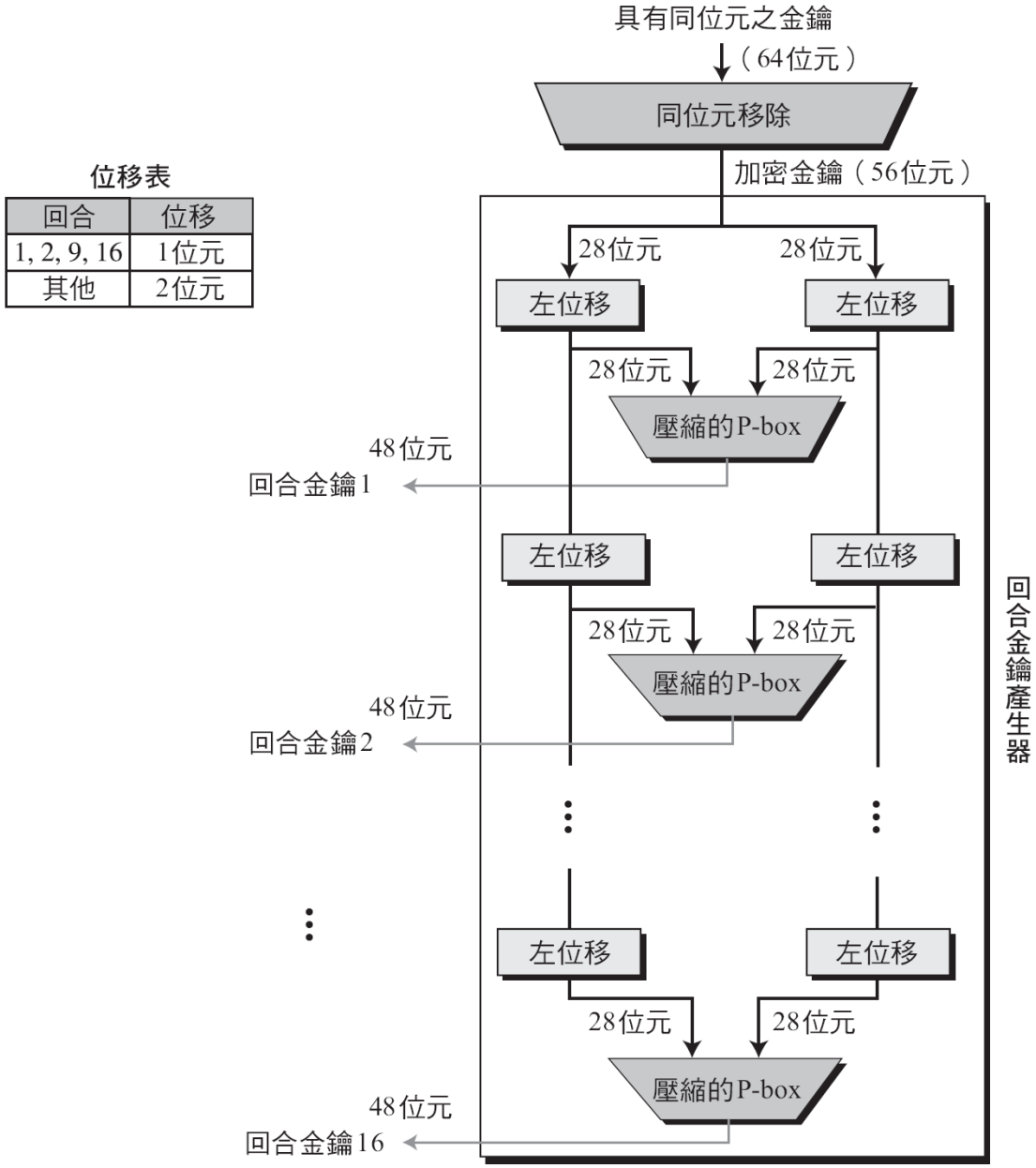


表 6.12 同位元移除表

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

表 6.13 每一個回合的位移數量

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

表 6.14 金鑰壓縮表

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

```
Key_Generator (keyWithParities[64], RoundKeys[16, 48], ShiftTable[16])
{
    permute (64, 56, keyWithParities, cipherKey, ParityDropTable)
    split (56, 28, cipherKey, leftKey, rightKey)
    for (round = 1 to 16)
    {
        shiftLeft (leftKey, ShiftTable[round])
        shiftLeft (rightKey, ShiftTable[round])
        combine (28, 56, leftKey, rightKey, preRoundKey)
        permute (56, 48, preRoundKey, RoundKeys[round], KeyCompressionTable)
    }
}
shiftLeft (block[28], numOfShifts)
{
    for (i = 1 to numOfShifts)
    {
        T ← block[1]
        for (j = 2 to 28)
        {
            block [j−1] ← block [j]
        }
        block[28] ← T
    }
}
```

範例 6.5

- 我們選擇一個隨機的明文區塊以及一個隨機的金鑰，並決定將所產生的密文區塊，全部均以十六進位來表示：

明文：123456ABCD132536　　金鑰：AABB09182736CCDD

密文：C0B7A8D05F3A829C

明文：123456ABCD132536

在初始排列之後：14A7D67818CA18AD

在分割之後：L_0=14A7D678　R_0=18CA18AD

回合	左半部	右半部	回合金鑰
第一回合	18CA18AD	5A78E394	194CD072DE8C
第二回合	5A78E394	4A1210F6	4568581ABCCE
第三回合	4A1210F6	B8089591	06EDA4ACF5B5
第四回合	B8089591	236779C2	DA2D032B6EE3
第五回合	236779C2	A15A4B87	69A629FEC913
第六回合	A15A4B87	2E8F9C65	C1948E87475E
第七回合	2E8F9C65	A9FC20A3	708AD2DDB3C0
第八回合	A9FC20A3	308BEE97	34F822F0C66D
第九回合	308BEE97	10AF9D37	84BB4473DCCC
第十回合	10AF9D37	6CA6CB20	02765708B5BF
第十一回合	6CA6CB20	FF3C485F	6D5560AF7CA5
第十二回合	FF3C485F	22A5963B	C2C1E96A4BF3
第十三回合	22A5963B	387CCDAA	99C31397C91F
第十四回合	387CCDAA	BD2DD2AB	251B8BC717D0
第十五回合	BD2DD2AB	CF26B472	3330C5D9A36D
第十六回合	19BA9212	CF26B472	181C5D75C66D

在合併之後：19BA9212CF26B472

密文：C0B7A8D05F3A829C　（在最終排列之後）

範例 6.6

- 讓我們來看在目的地的 Bob 如何使用相同的金鑰解開 Alice 傳送的密文，下表僅顯示部分回合的結果

密文：C0B7A8D05F3A829C			
在初始排列之後：19BA9212CF26B472 在分割之後：L_0=19BA9212　R_0=CF26B472			
回合	左半部	右半部	回合金鑰
第一回合	CF26B472	BD2DD2AB	181C5D75C66D
第二回合	BD2DD2AB	387CCDAA	3330C5D9A36D
...	...	...	...
第十五回合	5A78E394	18CA18AD	4568581ABCCE
第十六回合	14A7D678	18CA18AD	194CD072DE8C
在合併之後：14A7D67818CA18AD			
明文：123456ABCD132536			（在最終排列之後）

6.3 DES 分析

- 評論者使用放大鏡來分析 DES，測量區塊加密法之某些預期特性強度的測試已經進行，DES 的元件也被仔細審查是否符合建構的準則
- 本節討論主題
  - 特性
  - 設計準則
  - DES 的弱點

6.3.1 特性

- 區塊加密法想要達成的兩個特性為崩塌影響以及完整性影響

崩塌影響

- 崩塌影響（**Avalanche Effect**）意指在明文（或金鑰）少數位元的一個小改變會造成在密文中的重大改變

範例 6.7

- 為了檢查 DES 的崩塌影響，我們（以相同的金鑰）加密兩個明文區塊。兩個明文區塊僅有一個位元不同，試著觀察在每個回合中位元差異的數量

明文：0000000000000000　　金鑰：22234512987ABB23

密文：4789FD476E82A5F1

明文：0000000000000001　　金鑰：22234512987ABB23

密文：0A4ED5C15A63FEA3

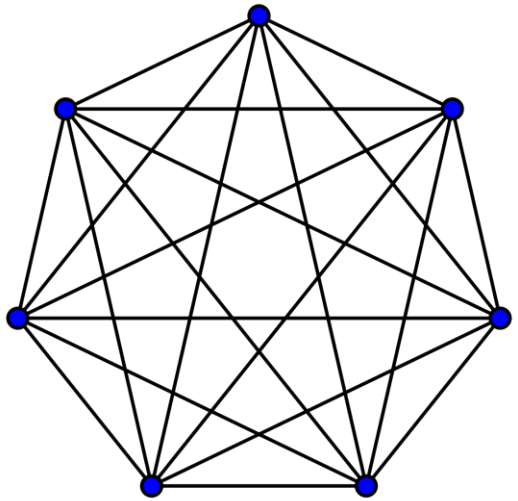
範例 6.7 (續)

- 雖然兩個明文區塊僅有最右邊的位元不同，其密文卻有 29 個位元不同。這表示改變大約 1.5% 的明文將產生大約 45% 的密文變化。下表顯示每一個回合的改變情形，其中早在第三回合就已發生重大改變

回合	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
位元差異	1	6	20	29	30	33	32	29	32	39	33	28	30	31	30	29

完整性影響

- 完整性影響（**Completeness Effect**）意指密文的每一個位元需要由明文的許多位元來決定
- DES 的 S-box 與 P-box 會產生混淆與擴散，具有非常強的完整性影響



6.3.2 設計準則

- S-box
  - 在設計上提供位元從每個回合至下一回合的混淆與擴散
- P-box
  - 提供位元擴散
- 回合數量
  - DES 使用十六個回合的 Feistel 加密法，使得所產生的密文完全是明文與密文的隨機函數

6.3.3 DES 的弱點

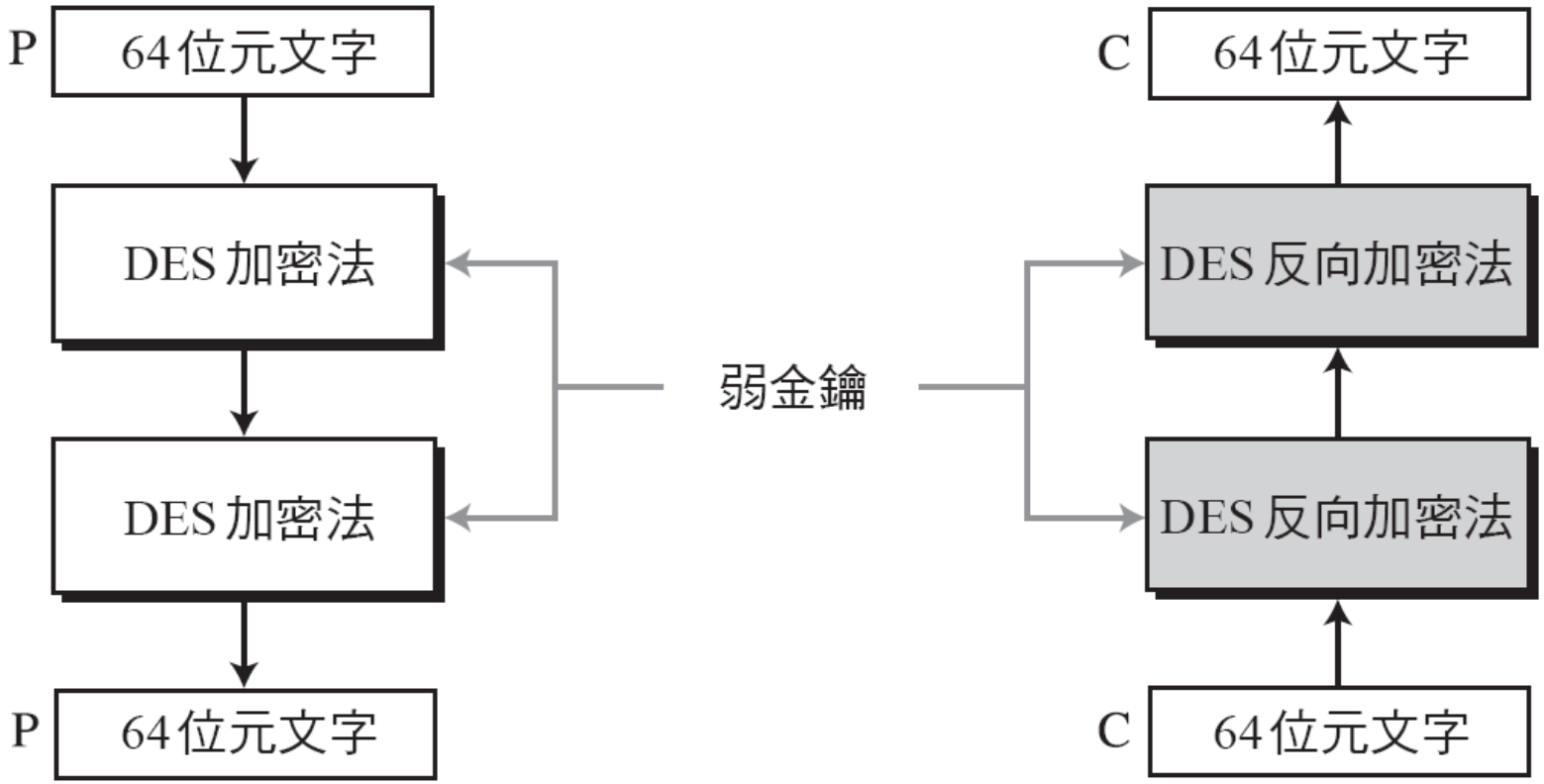
- S-box 的弱點
  - 兩個特定選擇的輸入可得到相同的輸出
- P-box 的弱點
  - 初始排列與最終排列的運算對於安全性並無幫助
- 加密金鑰的弱點
  - 存在弱金鑰，使得加密和解密具有相同的效果

表 6.18 弱金鑰

- DES 有 4 個所謂的弱金鑰。若使用弱金鑰，連續加密和解密兩次有可能會造成相同的效果，也就是 $E_k(E_k(P)) = P$

同位元移除前的金鑰（64位元）	實際金鑰（56位元）
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 0E0E 0E0E	0000000 FFFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

圖 6.11 使用弱金鑰進行兩次加密及解密



範例 6.18

- 我們嘗試使用表 6.18 的第一把弱金鑰來加密一個區塊內容兩次。在使用相同弱金鑰加密一個區塊兩次後，將會出現原始明文區塊。注意我們使用加密演算法兩次，並非進行一次加密後再接著進行一次解密。

金鑰：0x0101010101010101

明文：*0x1234567887654321*　　密文：0x814FE938589154F7

金鑰：0x0101010101010101

明文：0x814FE938589154F7　　密文：*0x1234567887654321*

表 6.19 半弱金鑰

- DES 有 6 對半弱金鑰，若使用某個半弱金鑰 k_1 對明文進行加密，則相當於使用其對應的半弱金鑰 k_2 進行解密的效果，也就是 $E_{k_2}(E_{k_1}(P)) = P$。

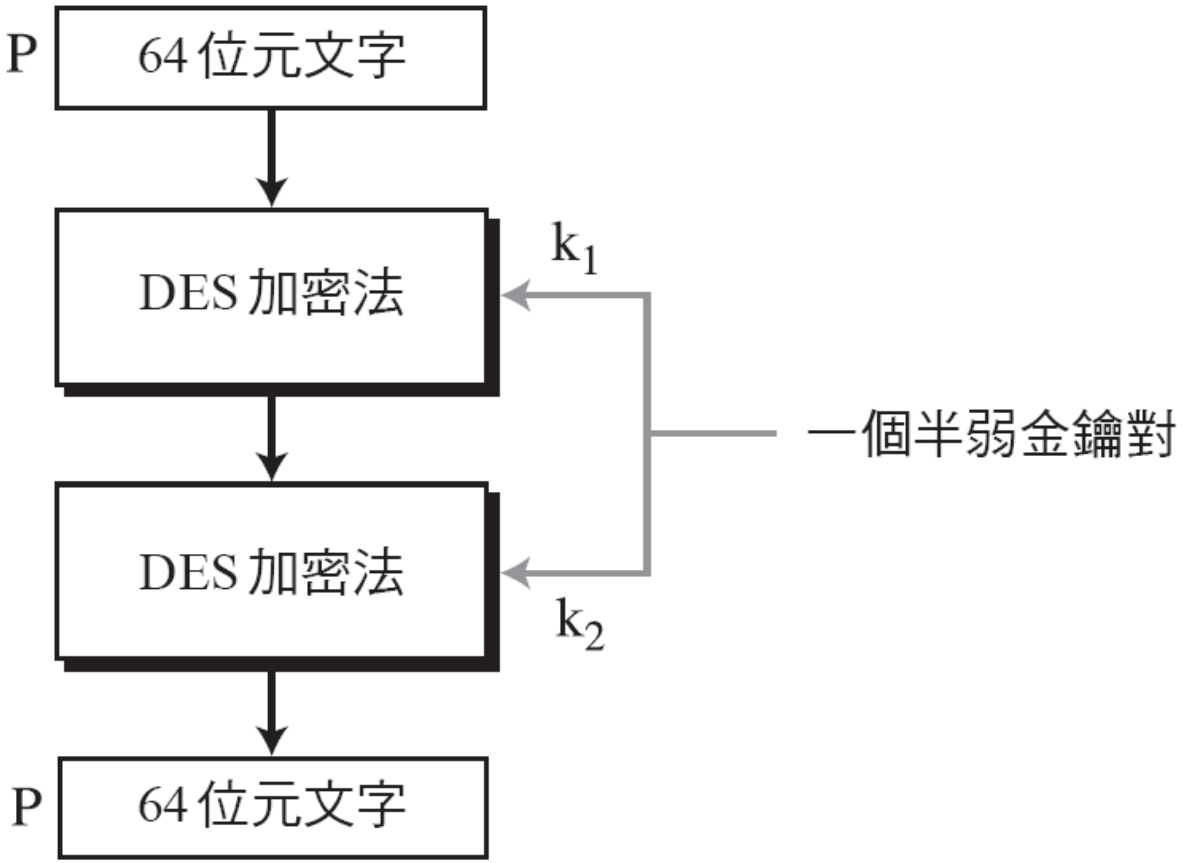
金鑰對的第一把金鑰	金鑰對的第二把金鑰
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E1 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

6.3.3 加密金鑰的弱點 (續)

- 使用半弱金鑰對時，回合金鑰的變化情形

回合金鑰1	9153E54319BD	6EAC1ABCE642
回合金鑰2	6EAC1ABCE642	9153E54319BD
回合金鑰3	6EAC1ABCE642	9153E54319BD
回合金鑰4	6EAC1ABCE642	9153E54319BD
回合金鑰5	6EAC1ABCE642	9153E54319BD
回合金鑰6	6EAC1ABCE642	9153E54319BD
回合金鑰7	6EAC1ABCE642	9153E54319BD
回合金鑰8	6EAC1ABCE642	9153E54319BD
回合金鑰9	9153E54319BD	6EAC1ABCE642
回合金鑰10	9153E54319BD	6EAC1ABCE642
回合金鑰11	9153E54319BD	6EAC1ABCE642
回合金鑰12	9153E54319BD	6EAC1ABCE642
回合金鑰13	9153E54319BD	6EAC1ABCE642
回合金鑰14	9153E54319BD	6EAC1ABCE642
回合金鑰15	9153E54319BD	6EAC1ABCE642
回合金鑰16	6EAC1ABCE642	9153E54319BD

圖 6.12 一個半弱金鑰對的加密與解密



範例 6.9

- 隨機挑選到一把弱金鑰或一把半弱金鑰的機率為何？

- 解法：DES 的金鑰範圍大小為 2^{56}，上述所有的金鑰數目為 16（即為 $4+12$）。因此，挑選到其中一把金鑰的機率為 2.2×10^{-16}，這幾乎是不可能的。

金鑰補數

- 在金鑰範圍（2^{56}）裡，很明顯地一半的金鑰與另一半的金鑰互為補數。金鑰補數（**Key Complement**）可以經由每一位元反向（將 0 改變為 1，或將 1 改變為 0）來得到。金鑰補數是否會簡化密碼分析的工作呢？答案是肯定的。攻擊者可以僅使用一半的可能金鑰（2^{55}）來執行暴力攻擊法。因為：

$$C = E(K, P) \quad \rightarrow \quad \overline{C} = E(\overline{K}, \overline{P})$$

換句話說，如果使用金鑰補數來加密明文的補數，會得到密文的補數，所以攻擊者不需測試所有 2^{56} 把可能的金鑰，可以僅測試一半，然後再取其結果的補數。

範例 6.10

- 測試我們所宣稱的金鑰補數，使用任意一把金鑰以及一個明文來計算其相對的密文，如果我們擁有金鑰補數及明文，則可以得到密文的補數（見表 6.20）。

表 6.20

	原始值	補數
金鑰	1234123412341234	EDCBEDCBEDCBEDCB
明文	12345678ABCDEF12	EDCBA987543210ED
密文	E112BE1DEFC7A367	1EED41E210385C98

6.4 DES 安全性

- 由於 DES 是第一個重要的區塊加密法，因此經過許多嚴密的審查。在所有的意圖攻擊當中，有三個非常重要：暴力攻擊法、差異破密分析以及線性破密分析。
- 本節討論主題
  - 暴力攻擊法 (Brute-Force Attack)
  - 差異破密分析 (Differential Cryptanalysis)
  - 線性破密分析 (Linear Cryptanalysis)

6.4.1 暴力攻擊法

- 我們已經討論過 DES 其短加密金鑰的弱點，結合這個弱點與金鑰補數的弱點，很清楚地，DES 可以使用 2^{55} 次加密來破解
- 每秒檢查一百萬個 key 的電腦 ➔ 超過 2000 年
- 擁有一百萬個晶片的平行處理電腦 ➔ 20 小時
- 1977 年，Diffie 和 Hellman 提出了一部造價約兩千萬美元的破解器，可以在一天內找到一個 DES 金鑰，但並無公開的實作
- 2008 年，由德國魯爾大學與英國基爾大學的工作群組所建造實作的 COPACOBANA RIVYERA 成功在一天的時間內破解 DES

6.4.2 差異破密分析

- DES 的設計者已知關於這類的攻擊，並且設計 S-box 及選擇使用十六個回合，使得 DES 特別能夠抵抗這類的攻擊

6.4.3 線性破密分析

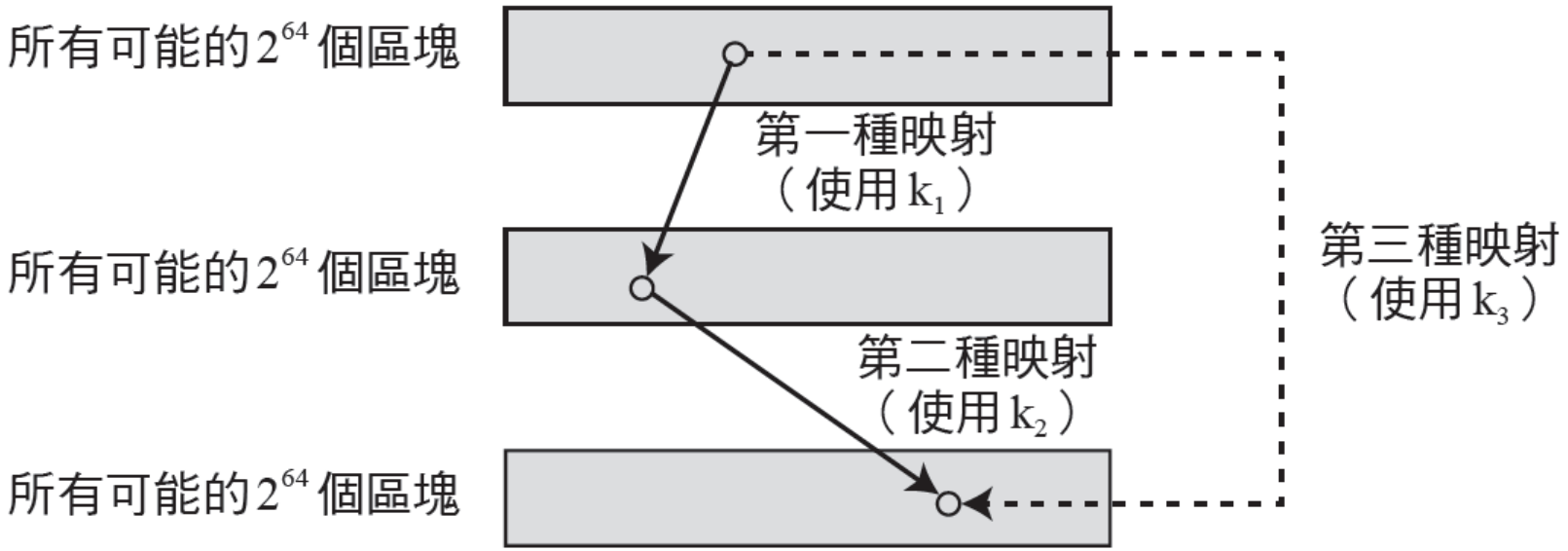
- 線性破密分析較差異破密分析來得新穎，在 1994 年由 Kaliski 和 Robshaw 所提。
- 比起差異破密分析，DES 較易遭受線性破密分析的攻擊，這是因為 DES 設計者尚未知曉此類攻擊。
- 設法得到 S-box 輸入與輸出的某幾個位元之間的線性關係式，並推出其成立的機率，然後串聯各回合的線性關係式，以統計的方法找出最可能的子金鑰。
- S-box 幾乎不能抵抗線性破密分析，目前已證明使用 2^{43} 對已知明文便可破解 DES。然而，從實際的角度來看，找到這麼多對的已知明文是不可行的。

6.5 多重 DES

- 對 DES 主要的批評是金鑰長度，由於電腦運算能力的增強，原版 DES 的金鑰容易被暴力破解。
- 幸運的是，DES 並非一個群，這表示我們可以使用 Double DES（2DES）或者 Triple DES（3DES）來增加金鑰長度。
- 本節討論主題
  - Double DES
  - Triple DES

6.5 多重 DES (續)

- 一個取代動作會將群中所有可能的輸入映射至所有可能的輸出。



6.5.1 Double DES

- 中間相遇攻擊
  - Diffie 和 Hellman 於 1977 年首先提出稱為中間相遇攻擊（Meet-in-the-Middle Attack）的已知明文攻擊，證明 Double DES 僅些微地改善此弱點（僅達 2^{57} 次測試），而並非達到理想上的提升（2^{112} 次測試）。

圖 6.14 Double DES 的中間相遇攻擊

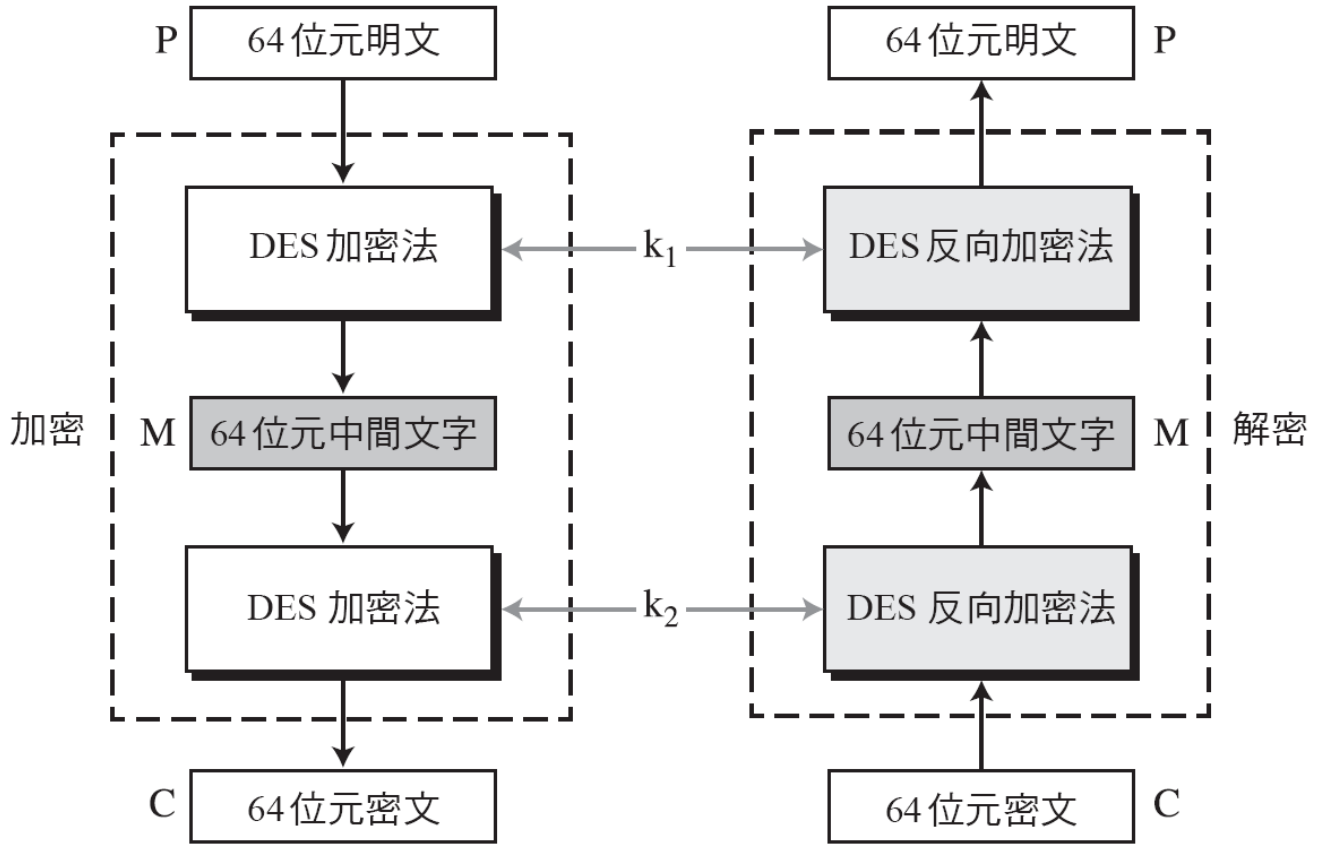
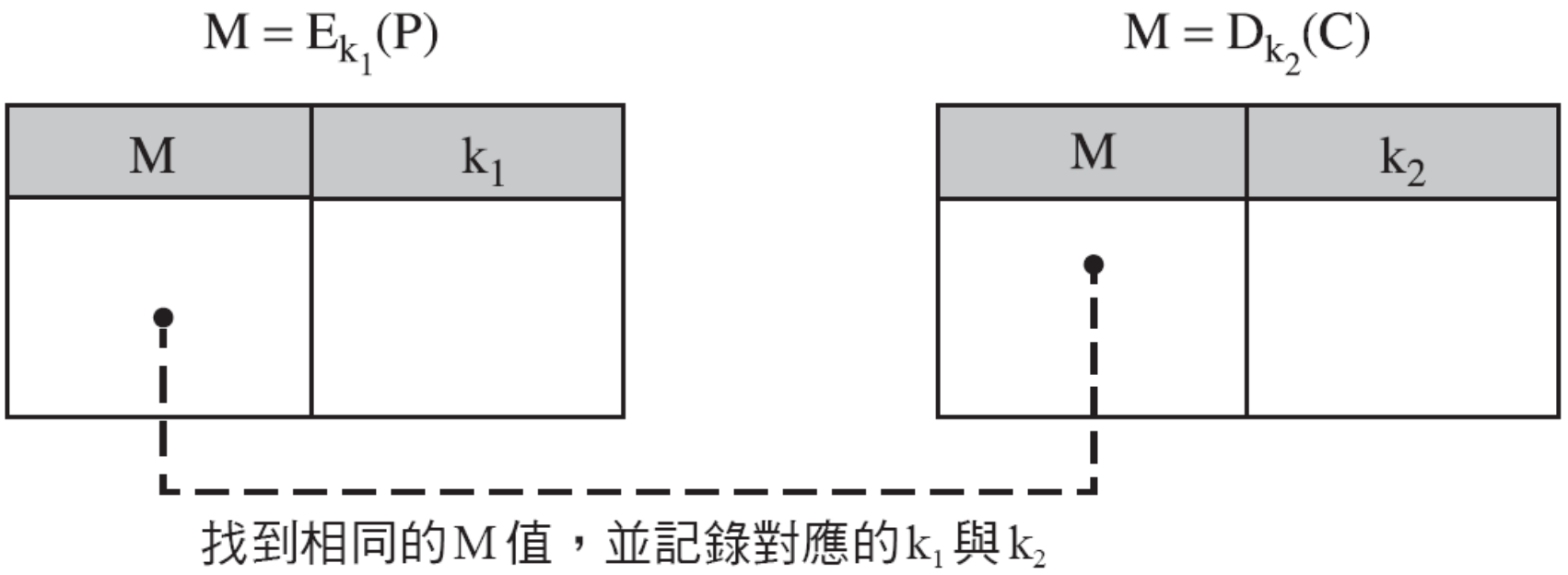


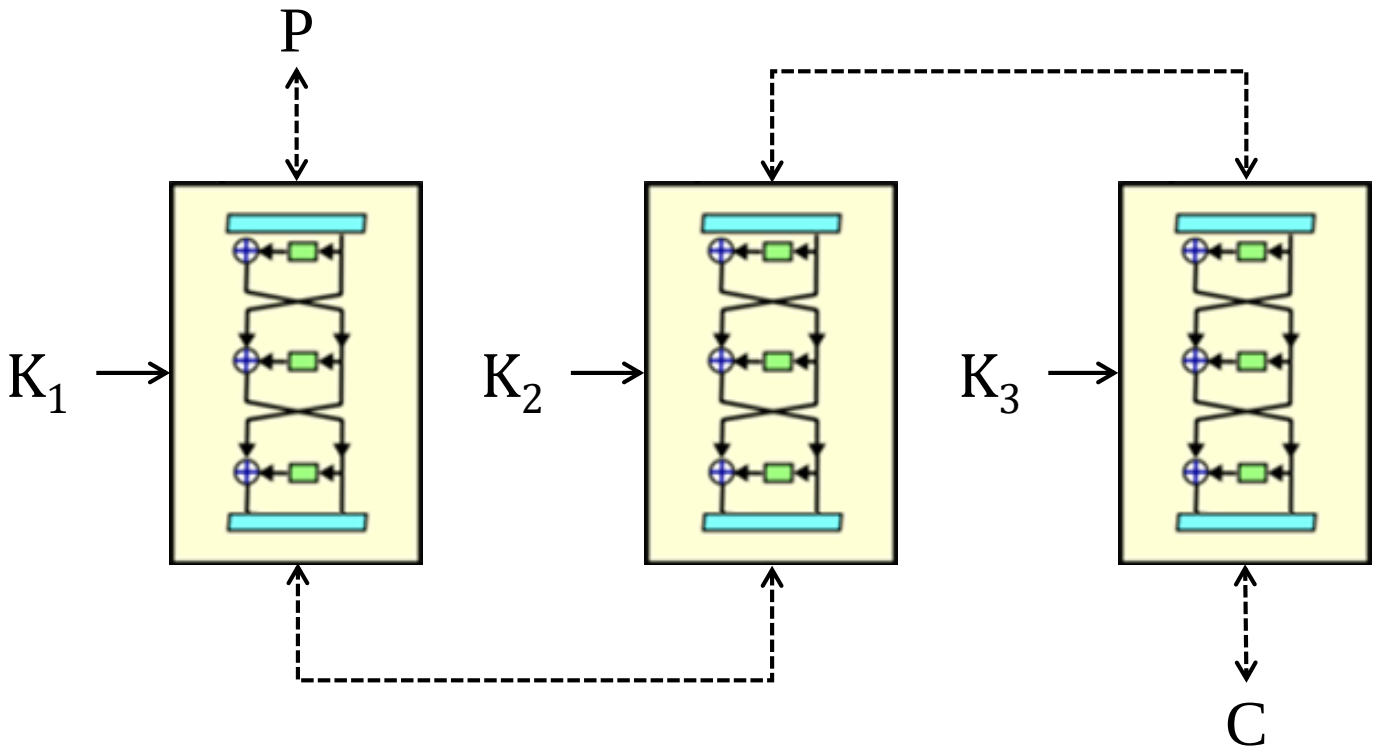
圖 6.15 中間相遇攻擊的表格

- 兩邊的中間值 M 各有 2^{56} 個，所以 Eve 必須執行 $2 \cdot 2^{56} = 2^{57}$ 次測試



6.5.2 Triple DES

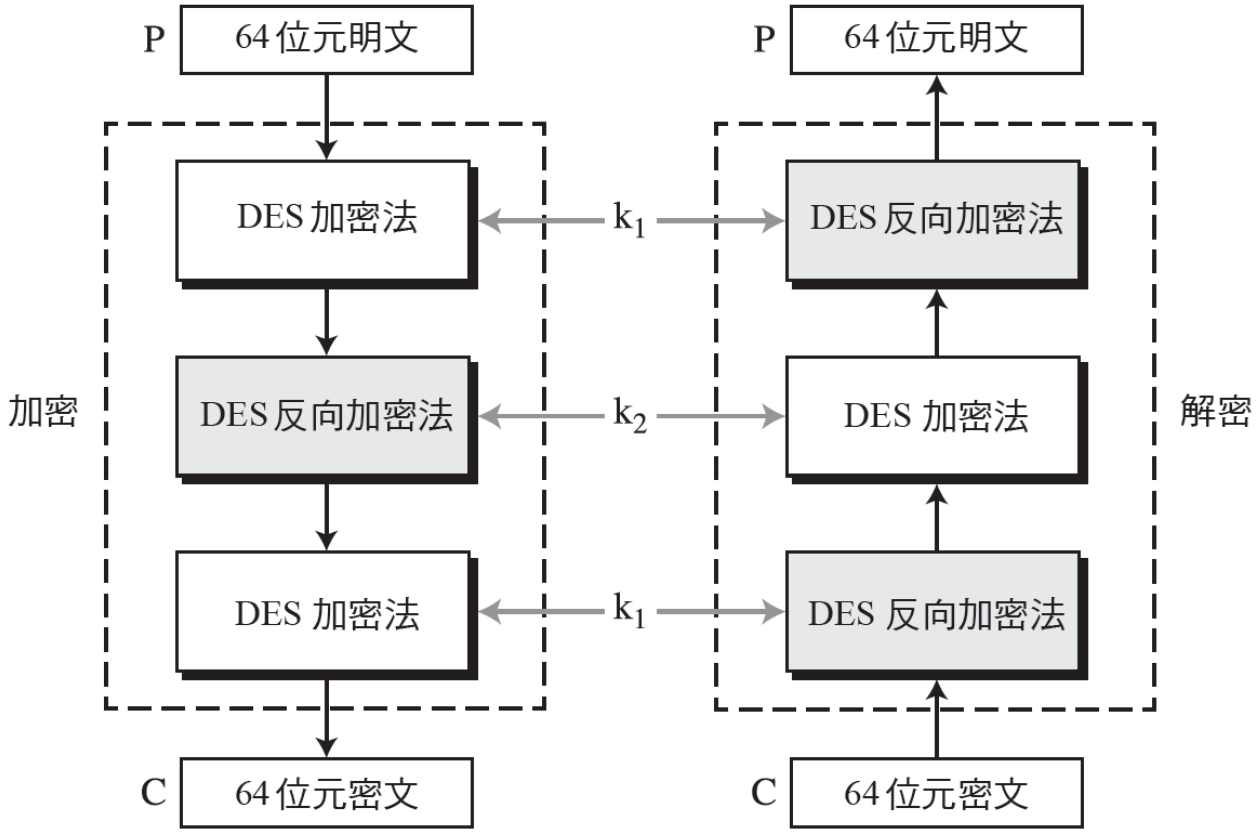
- Triple DES 的一般結構



6.5.2 Triple DES (續)

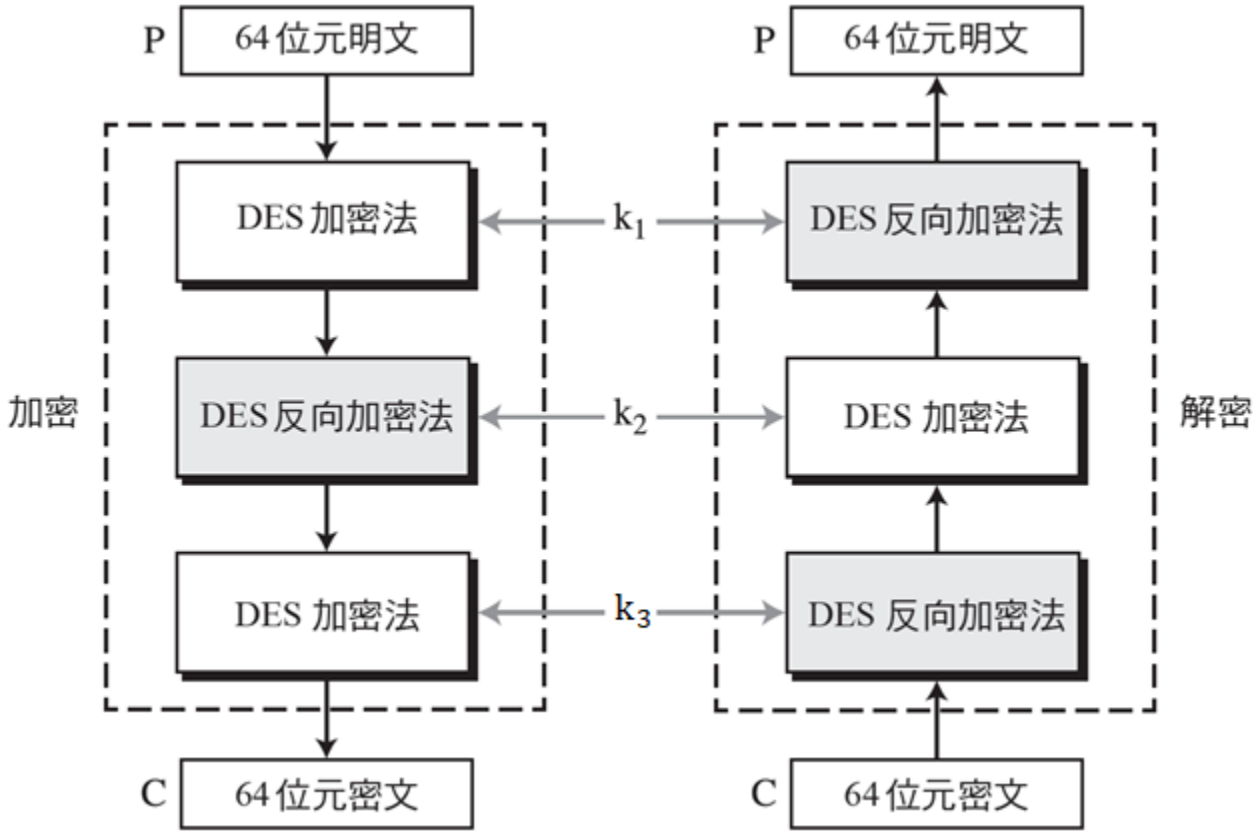
- Triple DES 的金鑰選取
  - 金鑰選擇一：K_1、K_2 及 K_3 皆獨立選取，又稱 3TDEA，強度最高（擁有金鑰長度 $3 \times 56 = 168$ 個位元）
  - 金鑰選擇二：僅 K_1 和 K_2 是獨立選取，而 $K_3 = K_1$，又稱 2TDEA，安全性稍低（擁有金鑰長度 $2 \times 56 = 112$ 個位元）。此選項會比單純應用 DES 兩次的強度還高，因為它可以防禦中間相遇攻擊。
  - 金鑰選擇三：三個金鑰均相等（$K_1 = K_2 = K_3$），安全性等同於 DES（金鑰長度 56 個位元），也就是第一次和第二次的 DES 會相互抵消，當使用此選擇時，Triple DES = Single DES。

使用兩把金鑰的 3DES



$$C = E_{k_1}(D_{k_2}(E_{k_1}(P)))$$
$$P = D_{k_1}(E_{k_2}(D_{k_1}(C)))$$

使用三把金鑰的 3DES



$$C = E_{k_3}(D_{k_2}(E_{k_1}(P)))$$
$$P = D_{k_1}(E_{k_2}(D_{k_3}(C)))$$

使用三把金鑰的 3DES (續)

- 由於存在選擇明文和已知明文攻擊有可能成功破解使用兩把金鑰的 3DES，使得 NIST 認定它只有 80 位元的安全性，也因而促使某些應用採取使用三把金鑰的 3DES（Triple DES with Three Keys）
- 許多應用均採取使用三把金鑰的 3DES，例如用於訊息加密以及驗證的商業應用程式 PGP（Pretty Good Privacy）協定

結語

- 雖然 3DES 增強了 DES的安全性，但也存在理論上的攻擊方法
- 由於越來越多針對 DES 的破解法出現，使得其安全性逐漸降低，所以目前 DES 已經不再作為 NIST 的一個加密標準