# 網路安全概論

# 期中專題報告

DES 實作

系級：電機系統三

姓名：B0721251 楊仁傑

日期：2021/05/20

## 壹、DES 原理：

DES 加密算法主要可以分為以下四個步驟：

（1）初始置換

（2）生成子密鑰

（3）迭代過程

（4）逆置換

### 一、初始置換(IP 置換)：

初始替換是將原始明文通過 IP 置換表處理。IP 置換表如下：

```
# 初始置換表
IP = [58, 50, 42, 34, 26, 18, 10, 2,
      60, 52, 44, 36, 28, 20, 12, 4,
      62, 54, 46, 38, 30, 22, 14, 6,
      64, 56, 48, 40, 32, 24, 16, 8,
      57, 49, 41, 33, 25, 17, 9,  1,
      59, 51, 43, 35, 27, 19, 11, 3,
      61, 53, 45, 37, 29, 21, 13, 5,
      63, 55, 47, 39, 31, 23, 15, 7]
```

圖 1. IP 置換表

其中，IP 置換表中的數字指的是指原文位置，例如 58 指將 M 第 58 位放置第 1 位。

例如：

輸入 64 位明文數據 M（64 bits）：

明文 M（64 bits）＝

0110001101101111010110110101110000011101010111010001100101011110010

將 M 經過 IP 置換後為 M’

M’（64 bits）＝

1111111110111000011101100101011100000000111111110000011010000011

## 二、生成子密鑰

在 DES 加密中會執行 16 次迭代，每次重複過程的數據長度為 48bits，因此需要 16 個 48bits 的子密鑰來進行加密，生成子密鑰的過程如下：

### I. 第一輪置換：

#### A. PC1 置換：

密鑰 K = 0001001100110100010101110111100110011011110111100110111111110001 經過 PC-1 表置換，可得到 K’。

```
# 金鑰置換表，將64位金鑰變成56位
PC_1 = [57, 49, 41, 33, 25, 17, 9,
        1,  58, 50, 42, 34, 26, 18,
        10, 2,  59, 51, 43, 35, 27,
        19, 11, 3,  60, 52, 44, 36,
        63, 55, 47, 39, 31, 23, 15,
        7,  62, 54, 46, 38, 30, 22,
        14, 6,  61, 53, 45, 37, 29,
        21, 13, 5,  28, 20, 12, 4]
```

圖 2. PC1 置換表

K'（56 位）= 11110000110011001010101011110101010101100110011110001111

取 K'的前 28 位作為 C0，且取 K'的後 28 位作為 D0，，則有

C0（28 位）= 1111000011001100101010101111

D0（28 位）= 0101010101100110011110001111

#### B. leftShift 置換：

生成 C0，D0 後進行左移操作，需要查詢移動位數表：

每輪移動移動位數表如下：

```
# 每輪左移的位數
shiftBits = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
```

圖 3. 左移置換表

進行第一輪移位，輪數為 1，查表得左移位數為 1。

C0 左移 1 位為 C1，且 D0 左移 1 位為 D1：

C1（28 位）= 1110000110011001010101011111

D1（28 位）= 1010101011001100111100011110

C. **PC2 置換：**

將 C1 和 D1 合併後，經過 PC-2 表置換得到子密鑰 K1。

由於 PC-2 表為 6x8 的表，經 PC-2 置換後的數據為 48 位，置換後得到密鑰 K1，

```
# 壓縮置換，將56位金鑰壓縮成48位子金鑰
PC_2 = [14, 17, 11, 24, 1,  5,
        3,  28, 15, 6,  21, 10,
        23, 19, 12, 4,  26, 8,
        16, 7,  27, 20, 13, 2,
        41, 52, 31, 37, 47, 55,
        30, 40, 51, 45, 33, 48,
        44, 49, 39, 56, 34, 53,
        46, 42, 50, 36, 29, 32]
```

圖 4. PC2 置換表

K1（48 位）= 000110110000001011101111111111000111000001110010

II. **第二輪置換**

A. **leftShift 置換：**

C1 和 D1 再次左移，輪數 =2，查表左移位數 =1，則 C1 和 D1 左移 1 位得到 C2 和 D2。

C2（28 位）= 1100001100110010101010111111

D2（28 位）= 0101010110011001111000111101

B. **PC2 置換：**

C2 和 D2 合併後為 56 位，經過 PC-2 表置換得到密鑰 K2（48 位）

K2（48 位）= 011110011010111011011001110110111100100111100101

重複進行加密動作，即可得到 K3-K16 子密鑰，且須注意 Ci 和 Di 左移的位數。

C3（28 位） = 0000110011001010101011111111

D3（28 位） = 0101011001100111100011110101

K3（48 位） = 010101011111110010001010010000101100111110011001

C4（28 位） = 0011001100101010101111111100

D4（28 位） = 0101100110011110001111010101

K4（48 位） = 011100101010110111010110110110110011010100011101

C5（28 位） = 1100110010101010111111110000

D5（28 位） = 0110011001111000111101010101

K5（48 位） = 011111001110110000000111111010110101001110101000

C6（28 位） = 0011001010101011111111000011

D6（28 位） = 1001100111100011110101010101

K6（48 位） = 011000111010010100111110010100000111101100101111

C7（28 位） = 1100101010101111111100001100

D7（28 位） = 0110011110001111010101010110

K7（48 位） = 111011001000010010110111111011000011000101111100

C8（28 位） = 0010101010111111110000110011

D8（28 位） = 1001111000111101010101011001

K8（48 位） = 111101111000101000111010110000010011101111111011

C9（28 位） = 0101010101111111100001100110

D9（28 位） = 0011110001111010101010110011

K9（48 位） = 111000001101101111101011111011011110011110000001

C10（28 位） = 0101010111111110000110011001

D10（28 位） = 1111000111101010101011001100

K10（48 位） = 101100011111001101000111101110100100011001001111

C11（28 位） = 0101011111111000011001100101

D11（28 位） = 1100011110101010101100110011

K11（48 位） = 001000010101111111010011110111101101001110000110

C12（28 位） = 0101111111100001100110010101

D12（28 位） = 0001111010101010110011001111

K12（48 位） = 011101010111000111110101100101000110011111101001

C13（28 位） = 0111111110000110011001010101

D13（28 位） = 0111101010101011001100111100

K13（48 位） = 100101111100010111010001111110101011101001000001

C14（28 位） = 1111111000011001100101010101

D14（28 位） = 1110101010101100110011110001

K14（48 位） = 010111110100001110110111111100101110011100111010

C15（28 位） = 1111100001100110010101010111

D15（28 位） = 1010101010110011001111000111

K15（48 位） = 101111111001000110001101001111010011111100001010

C16（28 位） = 1111000011001100101010101111

D16（28 位） = 0101010101100110011110001111

K16（48 位） = 110010110011110110001011000011100001011111110101

## 三、Feistel 函式：

Feistel 函式主要可以由下列四個部分組成：

### I. 擴展置換 E：

右半部分 Ri 的位數為 32 bits，而密鑰長度 Ki 為 48 bits，為了使 Ri 與 Ki 可以進行 xor 運算，所以需要利用擴展置換表 E 將 Ri 由 32 bits 擴充為 48 bits。

```
# 擴充置換表，將 32bits 擴充至 48bits
E = [32, 1,  2,  3,  4,  5,
     4,  5,  6,  7,  8,  9,
     8,  9,  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, 30, 31, 32, 1]
```

圖 5. 擴充置換表 E

L0（32 位） = 11111111101110000111011001010111

R0（32 位） = 00000000111111110000011010000011

R0（32 位）經過擴展置換後變為 48 bits 數據：

E(R0)（48 位） = 100000000001011111111110100000000110101000000110

### II. XOR 運算：

若兩輸入相異，輸出為 0；若兩輸出相同，輸出為 0。

| A | B | A **XOR** B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

圖 6. XOR 真值表

將 E(R0) 與 K1 作 XOR 運算：

E(R0) = 100000000001011111111110100000000110101000000110

K1 = 000110110000000101110111111111100011100000111 0010

E(R0)^K1 = 100110110001010100010001011111001010010001110100

**III. S 盒置換：**

置換運算由 8 個不同的置換盒（S 盒）完成。每個 S 盒有 6 bits 輸入，4 bits 輸出。運算流程如下：

若 S-盒 1 的輸入為 110111，第一位與最後一位構成 11，十進位值為 3，則對應第 3 行，中間 4 位為 1011 對應的十進位值為 11，則對應第 11 列。查找 S-盒 1 表的值為 14，則 S-盒 1 的輸出為 1110。8 個 S 盒將輸入的 48 位數據輸出為 32 位數據。

```
# S盒，每個S盒是4x16的置換表，6位 -> 4位
S = [
    [
        [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
        [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
        [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
        [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]
    ],
    [
        [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
        [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
        [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
        [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]
    ],
    [
        [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
        [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
        [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
        [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]
    ],
    [
        [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
        [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
        [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
        [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]
    ],
    [
        [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
        [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
        [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
        [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]
    ],
    [
        [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
        [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
        [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
        [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]
    ],
    [
        [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
        [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
        [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
        [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]
    ],
    [
        [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
        [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
        [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
        [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]
    ]
]
```

圖 7. S 盒置換表

將 E(R0)^K1（48 bits）= 100110110001010100010001011111001010010001110100，通過 S 盒替換得到輸出為 10001011110001000110001011101010（32 bits）。

## IV. P 置換：

將 S 盒置換的輸出結果作為 P 盒置換的輸入

```
# P置換，32位 -> 32位
P = [16, 7,  20, 21,
     29, 12, 28, 17,
     1,  15, 23, 26,
     5,  18, 31, 10,
     2,  8,  24, 14,
     32, 27, 3,  9,
     19, 13, 30, 6,
     22, 11, 4,  25]
```

圖 8. P 置換表

將 S 盒輸出 10001011110001000110001011101010（32 bits）經過 P 置換，得到輸出 01001000101111110101010110000001（32 bits）。

綜上所述，將 Feistel 函式重複執行 16 次即可得到 L16 與 R16。

第一次疊代過程 f(R0,K1) = 01001000101111110101010110000001

計算 L1（32 位）= R0 = 00000000111111110000011010000011

計算 R1（32 位）= L0 ^ f(R0,K1) = 10110111000001110010001111010110

經過 16 次疊代後輸出：

L16（32 位）= 0011000001000001001101101100101000

R16（32 位）= 10110001011001010011000000011000

**四、逆置換(IP$^{-1}$)：**

逆置換是初始置換的逆運算。從初始置換規則中可以看到，原始數據的第 1 位置換到了第 40 位，第 2 位置換到了第 8 位。則逆置換就是將第 40 位置換到第 1 位，第 8 位置換到第 2 位。以此類推，逆置換規則表如下：

```
# 結尾置換表
IIP = [40, 8, 48, 16, 56, 24, 64, 32,
       39, 7, 47, 15, 55, 23, 63, 31,
       38, 6, 46, 14, 54, 22, 62, 30,
       37, 5, 45, 13, 53, 21, 61, 29,
       36, 4, 44, 12, 52, 20, 60, 28,
       35, 3, 43, 11, 51, 19, 59, 27,
       34, 2, 42, 10, 50, 18, 58, 26,
       33, 1, 41, 9,  49, 17, 57, 25]
```

圖 9. IP$^{-1}$ 置換表

將 L16 與 R16 構成 64 位數據，經過逆置換表輸出密文為：

L16+R16　：0011000010000100110110110010100010110001011001010011000000011000

Ciphertext　：0101100000000100000011000000001011110011011101011000011000001101000

## 貳、DES 各模式示意圖：

1. ECB 模式：

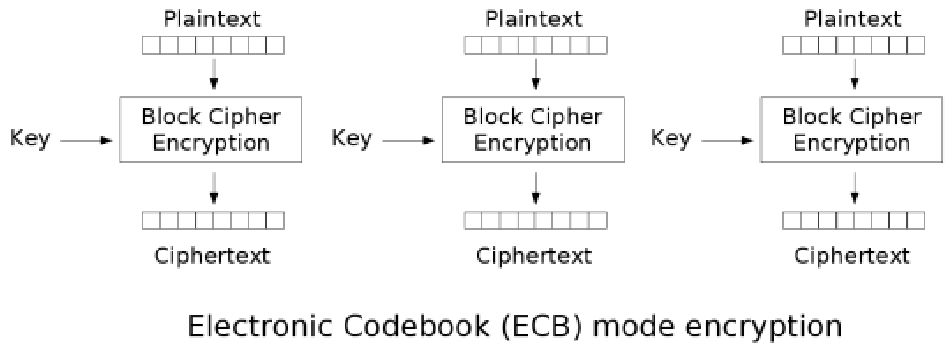



圖 1. ECB 加密模式示意圖

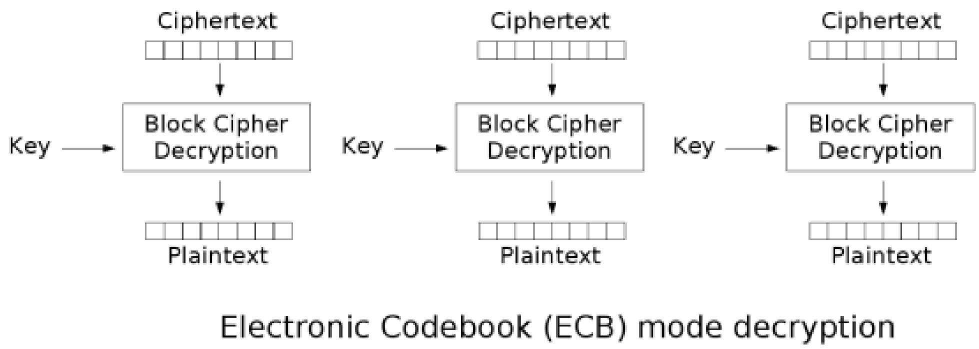



圖 2. ECB 解密模式示意圖

2.　CBC 模式：

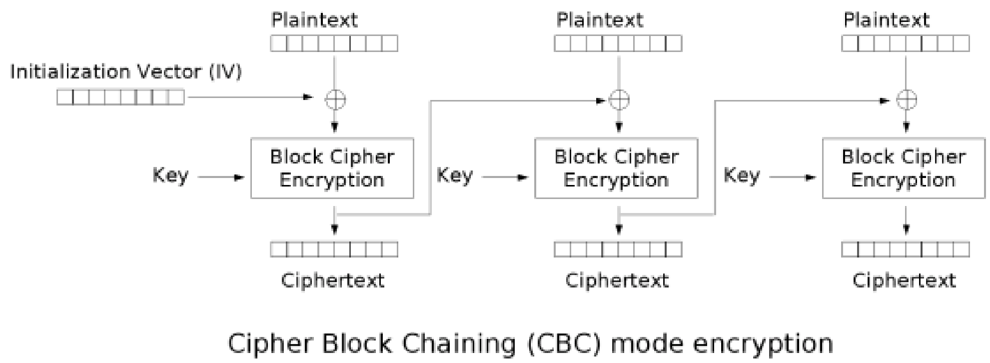



圖 3. CBC 加密模式示意圖

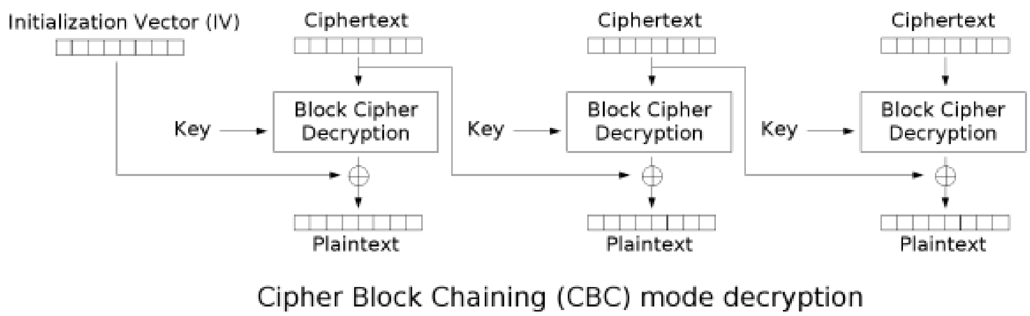



圖 4. CBC 解密模式示意圖

3. CFB 模式：

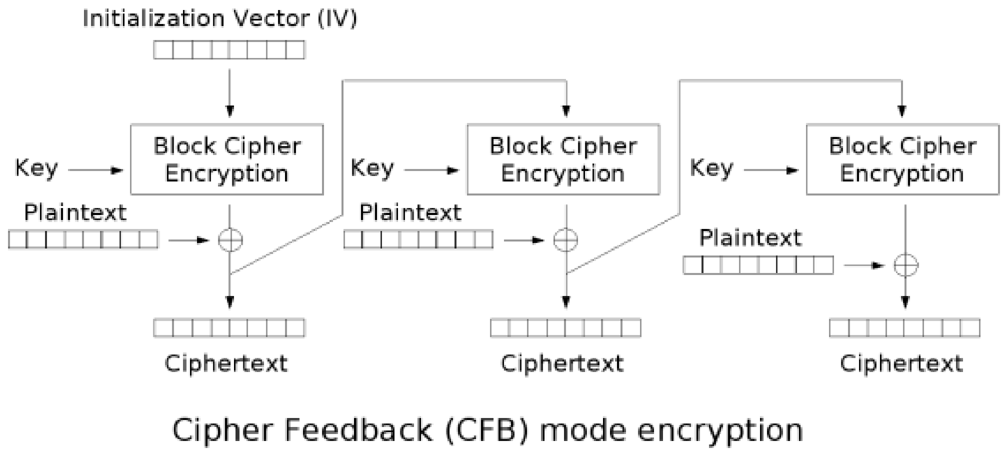



圖 5. CFB 加密模式示意圖

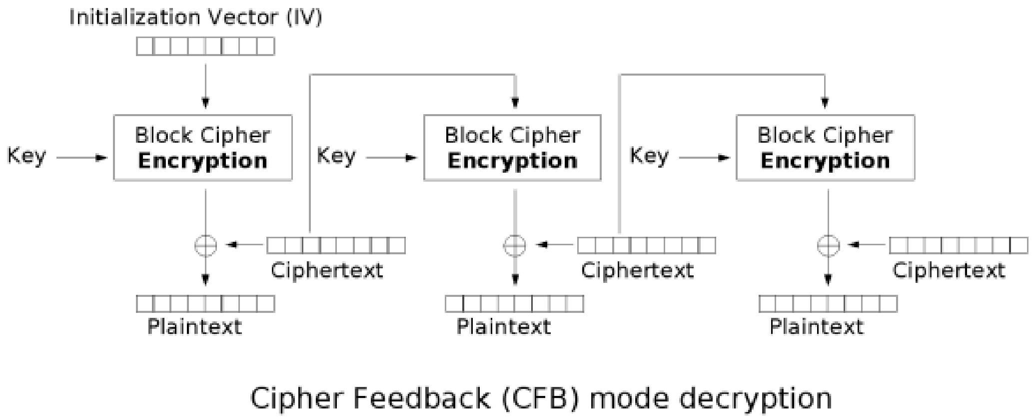



圖 6. CFB 解密模式示意圖

4.　OFB 模式：

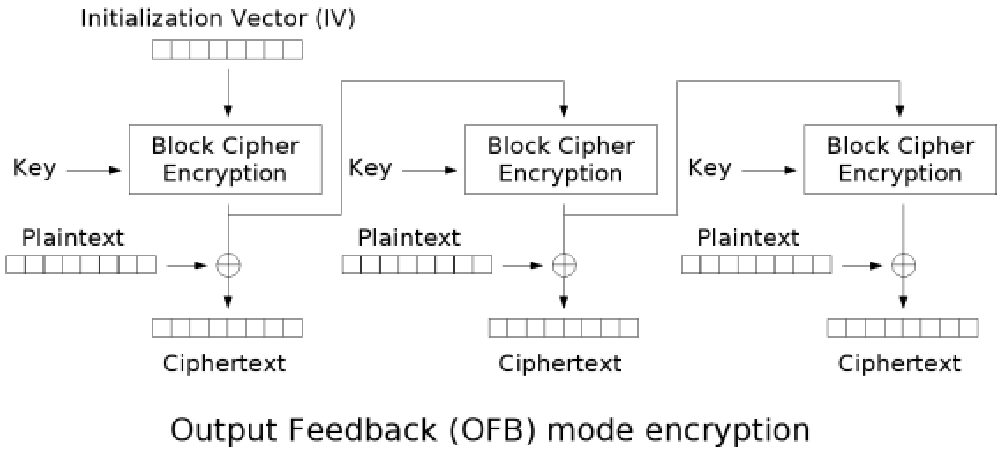



圖 7. OFB 加密模式示意圖

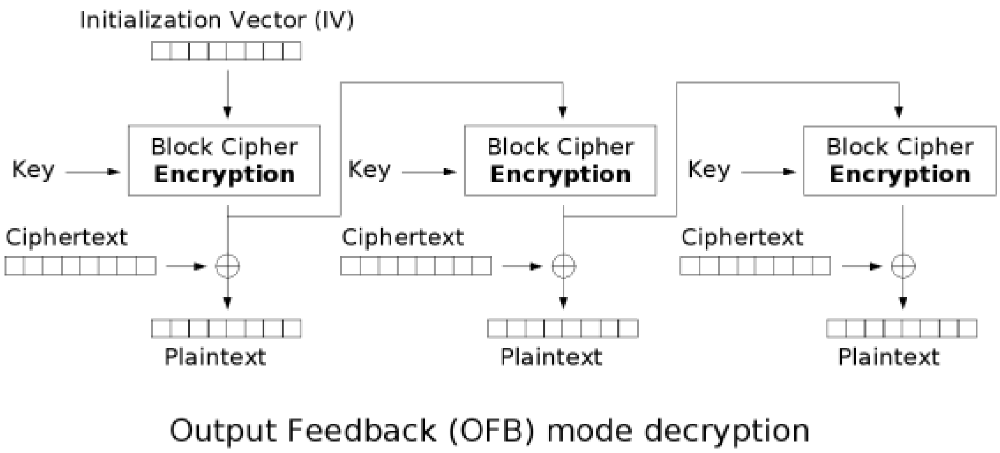



圖 8. OFB 解密模式示意圖

5.　CTR 模式：

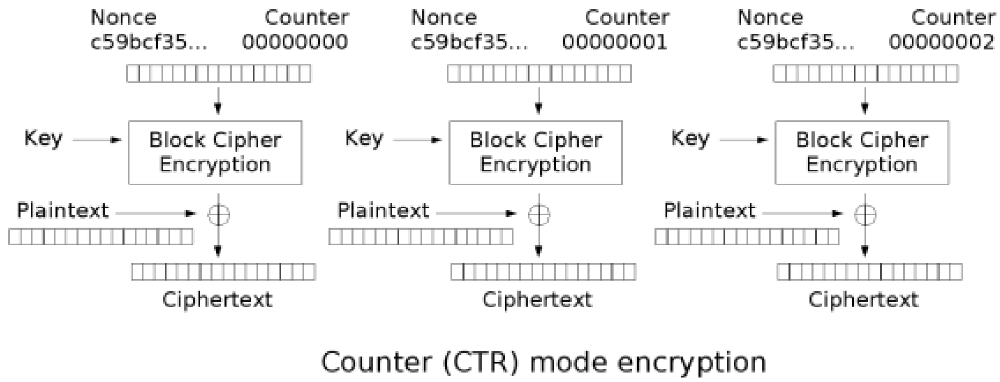



圖 9. CTR 加密模式示意圖

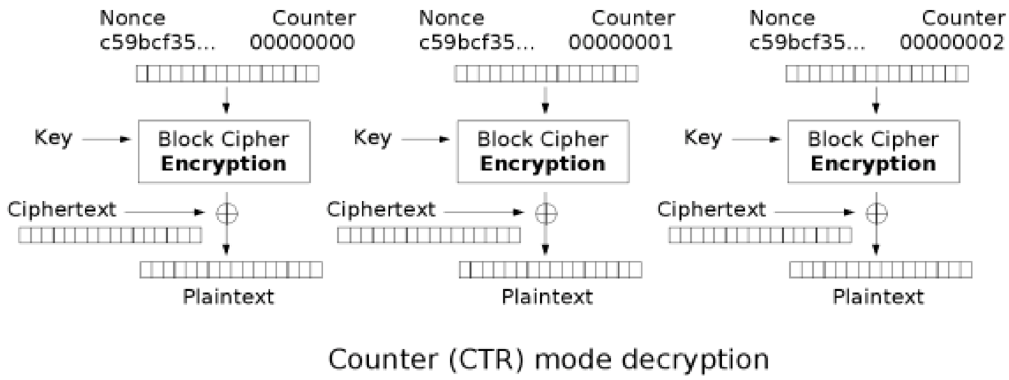
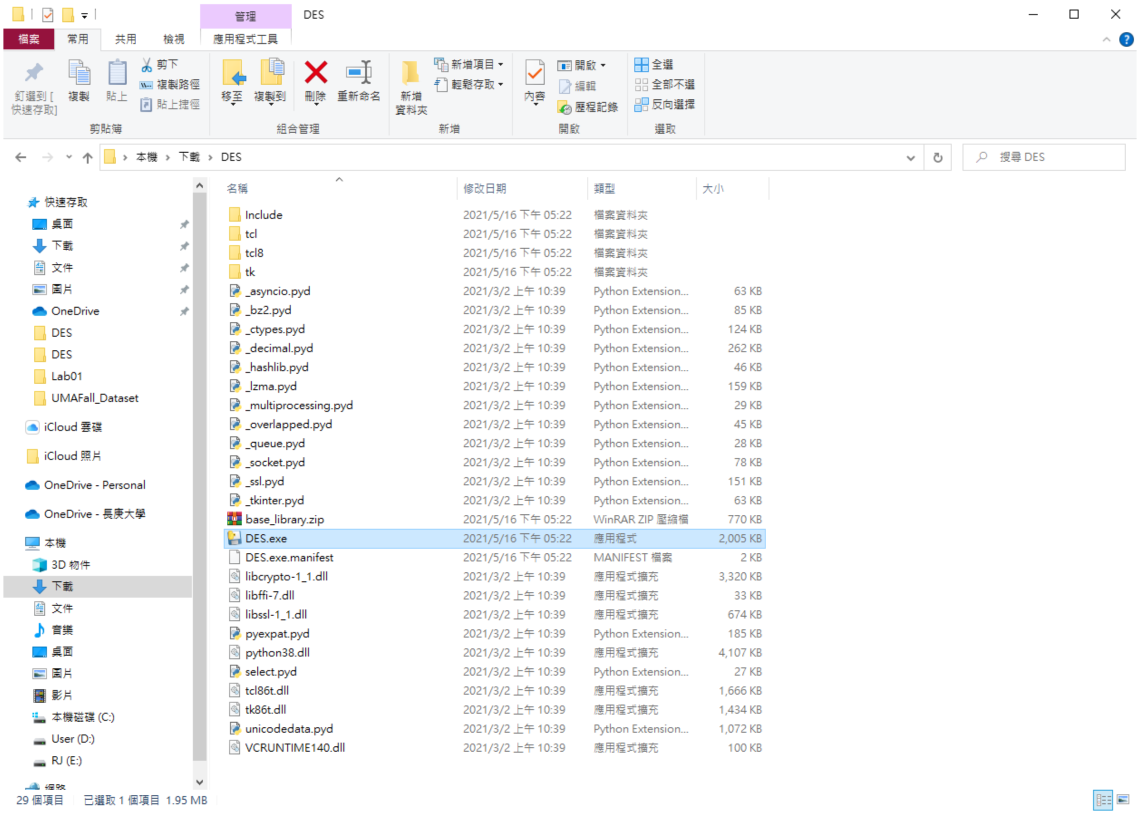



圖 10. CTR 解密模式示意圖

## 參、實作操作：

1. 將 DES.rar 解壓縮後，選取資料夾內的 DES.exe。




2. 打開 DES.exe 並選取特定檔案後即可使用。

3. 加密功能展示：

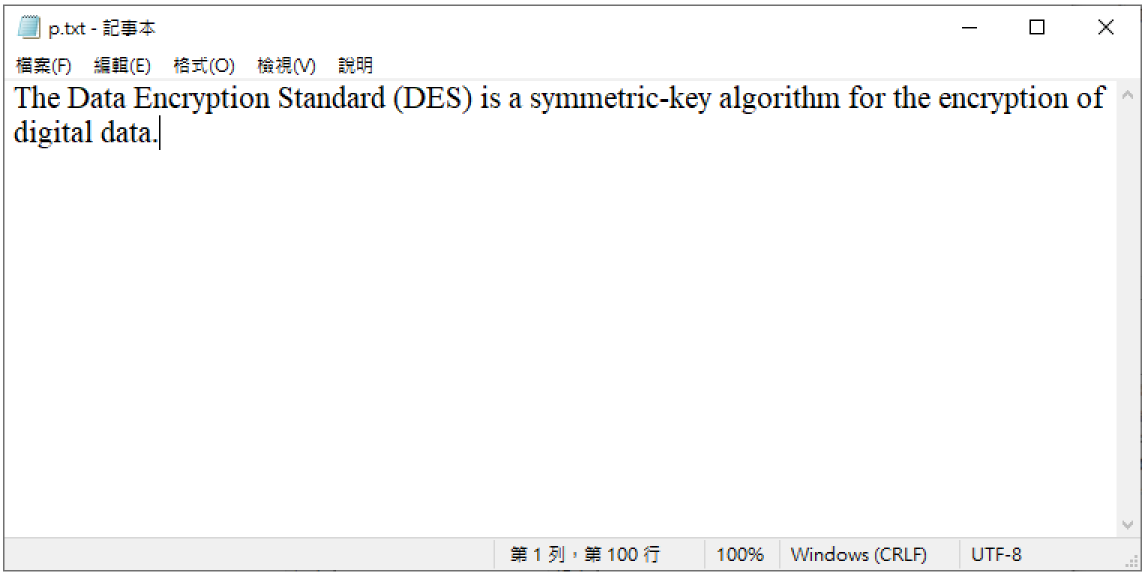



圖 11. 原始明文內容(plaintext)

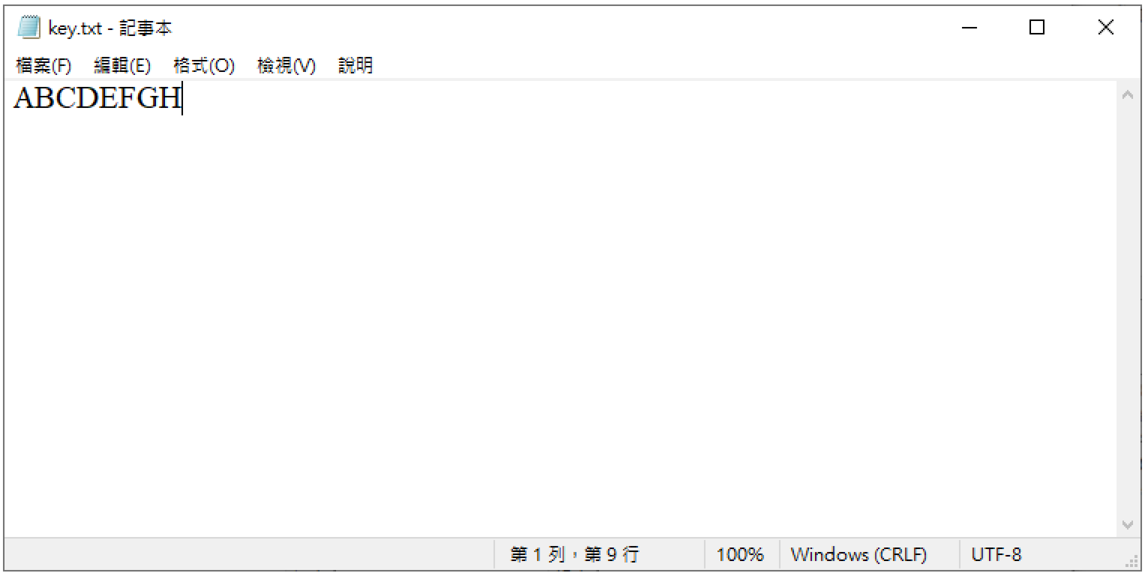



圖 12. 金鑰內容(key)

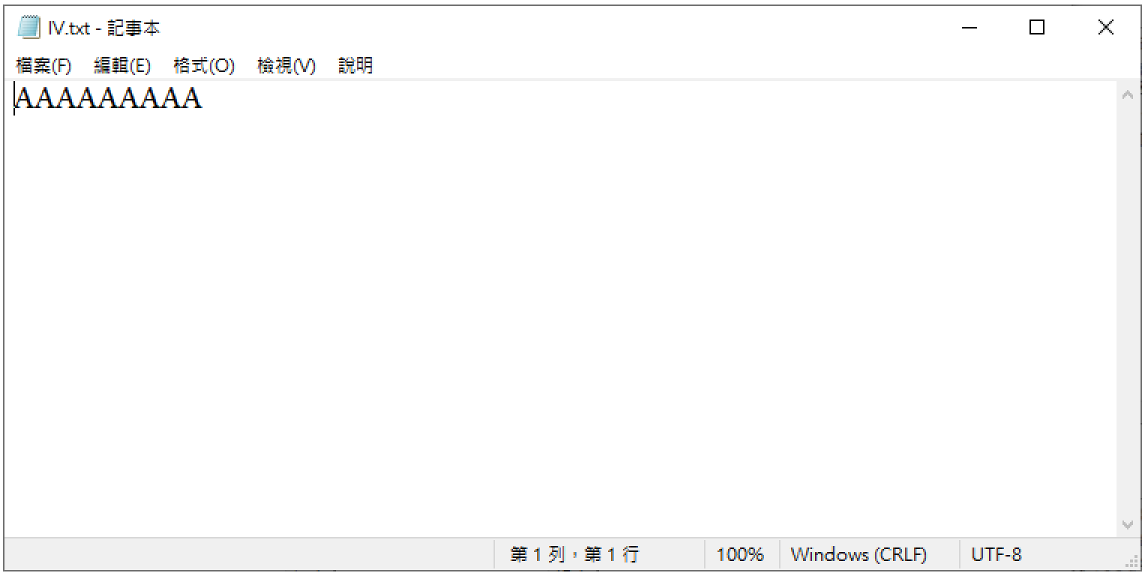



圖 13. 向量內容(IV)

A. ECB 模式

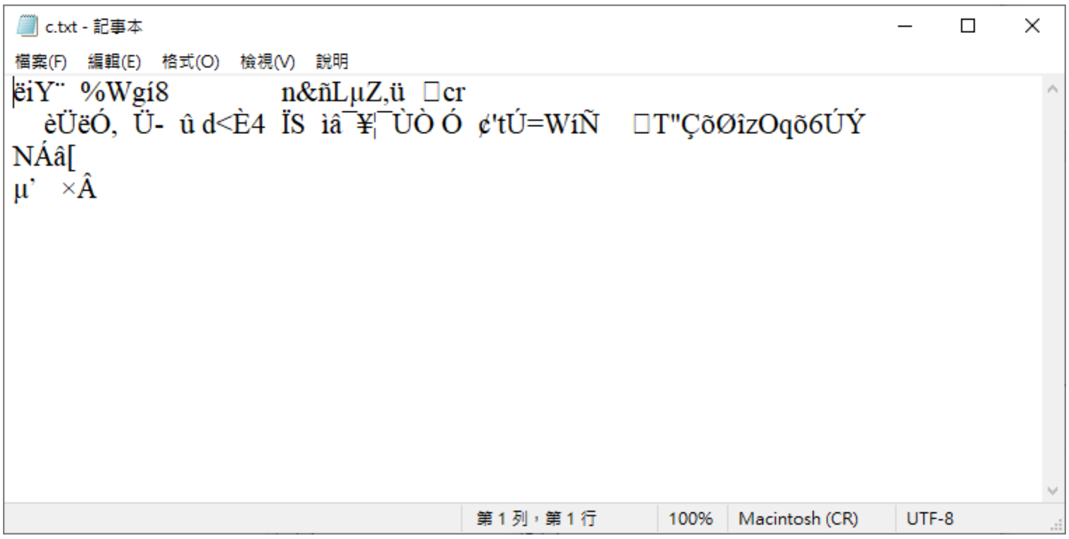



圖 12. 由 ASCII 編碼輸出的密文內容

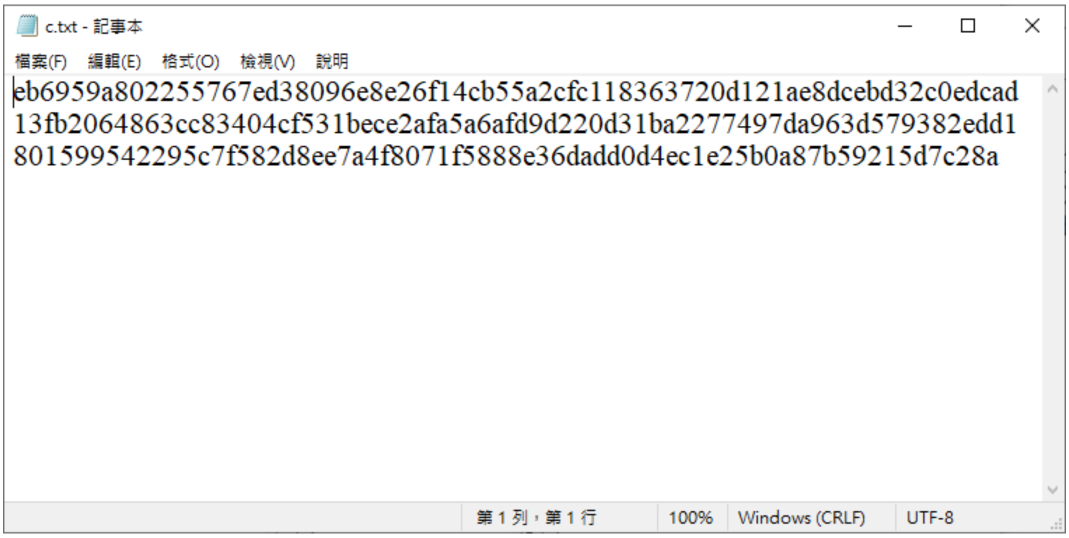



圖 13. 由 Hex 編碼輸出的密文內容

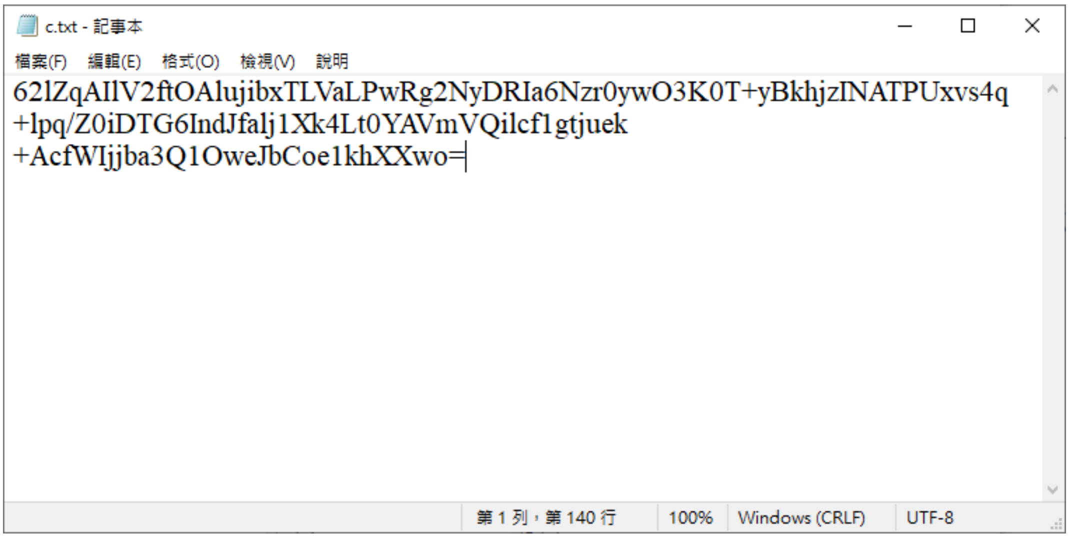



圖 14. 由 Base64 編碼輸出的密文內容

B. CBC 模式

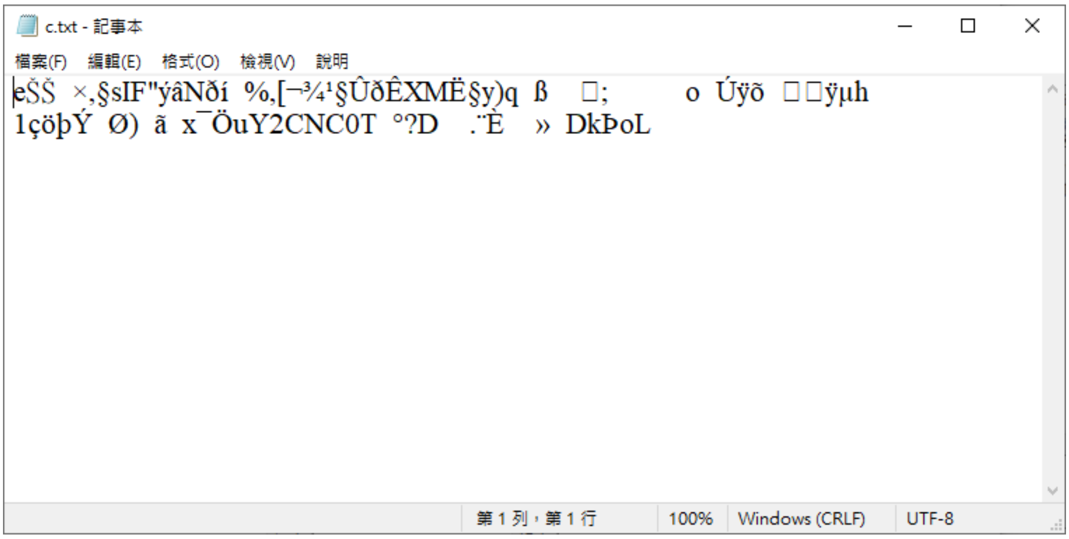



圖 12. 由 ASCII 編碼輸出的密文內容

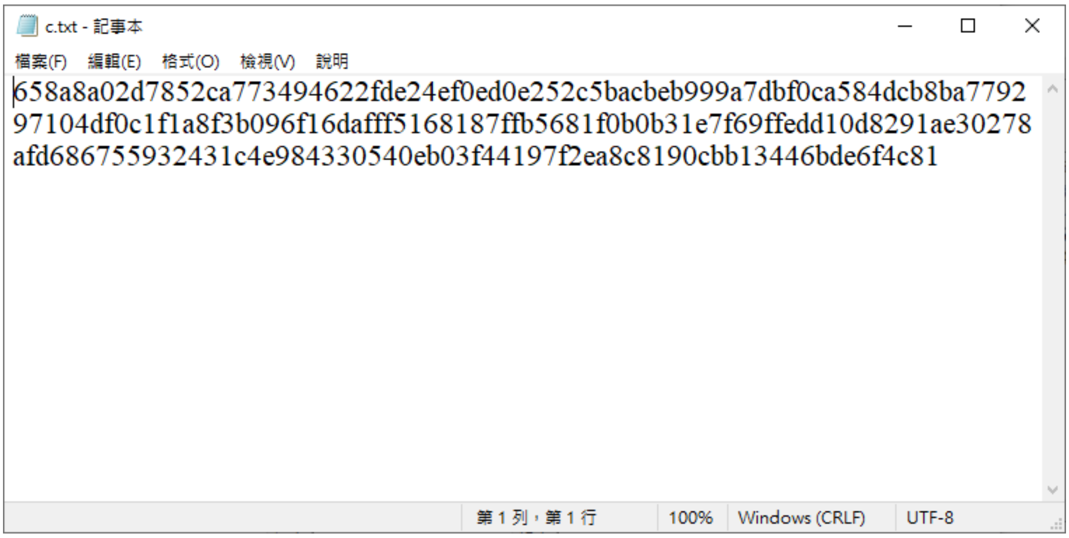



圖 13. 由 Hex 編碼輸出的密文內容


圖 14. 由 Base64 編碼輸出的密文內容

C. CFB 模式

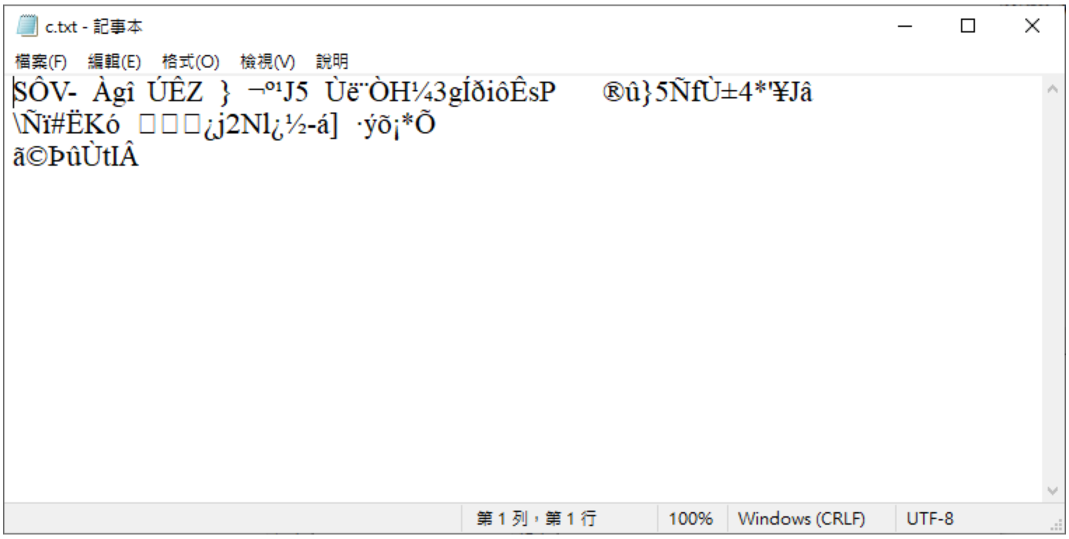



圖 12. 由 ASCII 編碼輸出的密文內容

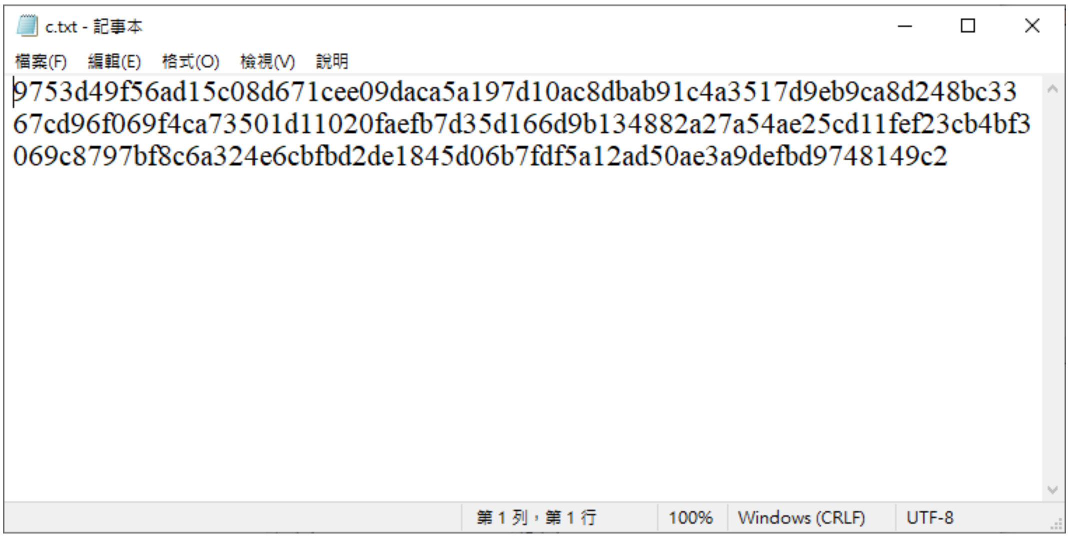



圖 13. 由 Hex 編碼輸出的密文內容

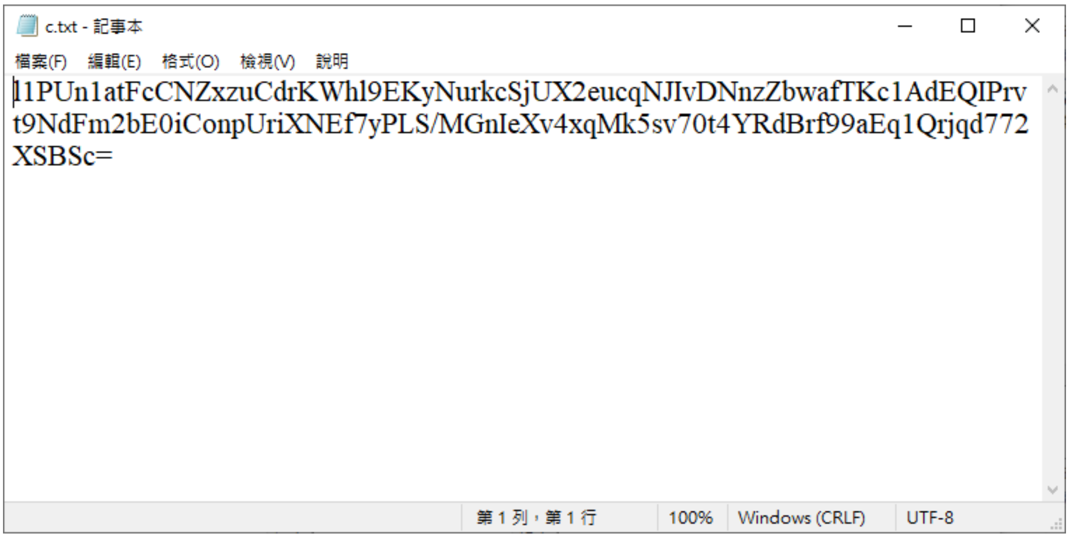



圖 14. 由 Base64 編碼輸出的密文內容

D. OFB 模式

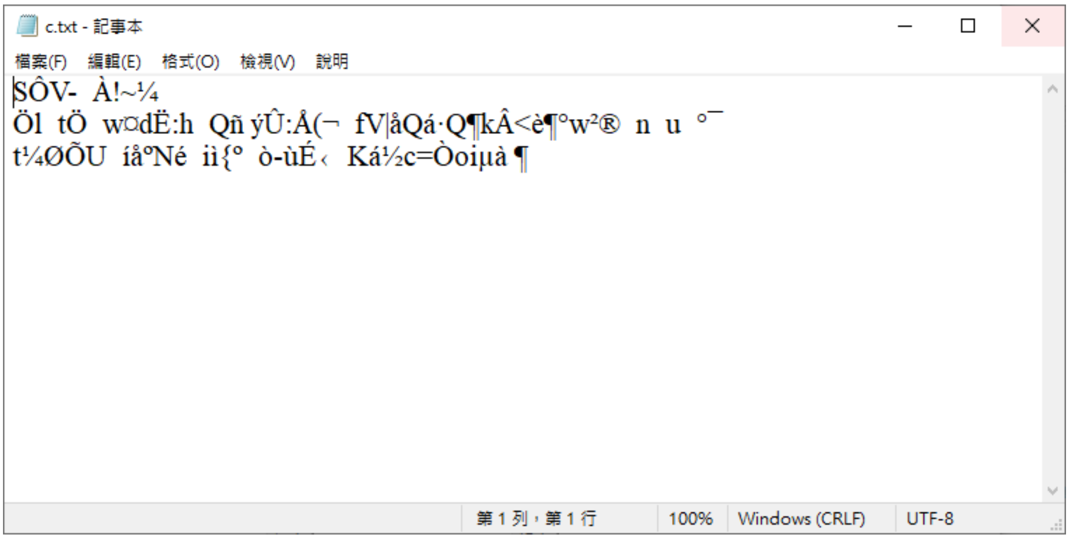



圖 12. 由 ASCII 編碼輸出的密文內容

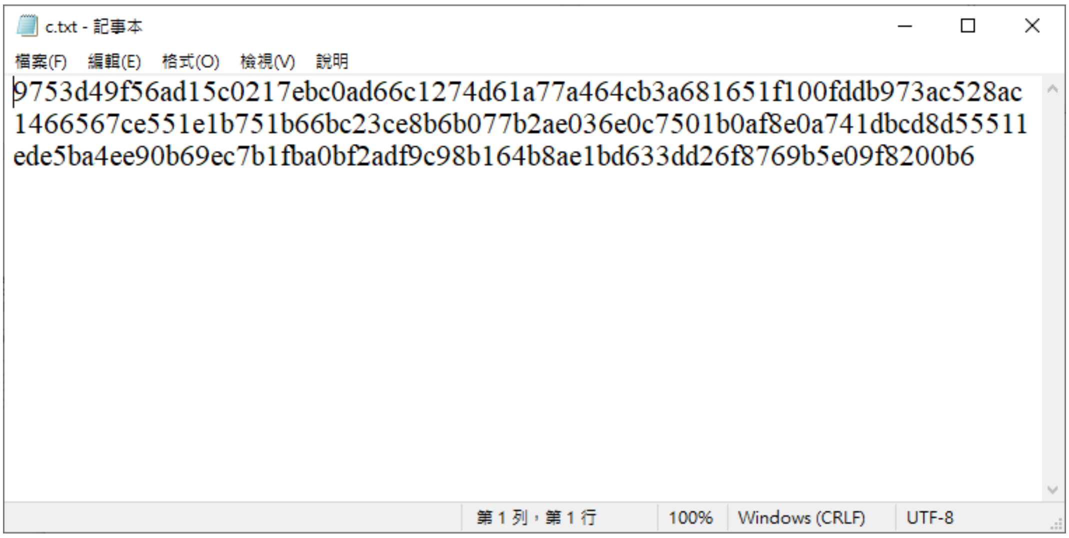



圖 13. 由 Hex 編碼輸出的密文內容

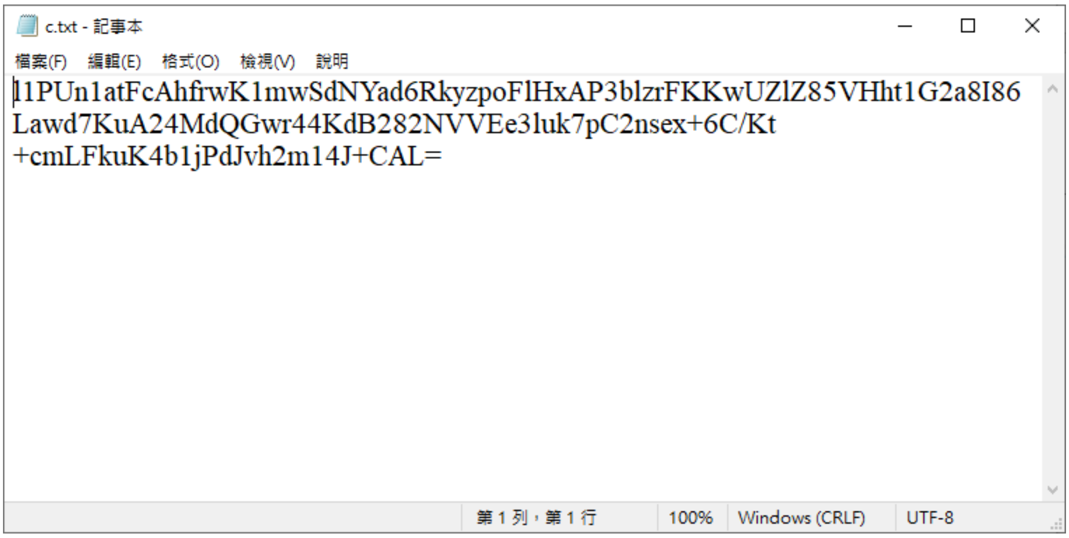



圖 14. 由 Base64 編碼輸出的密文內容

## E. CTR 模式

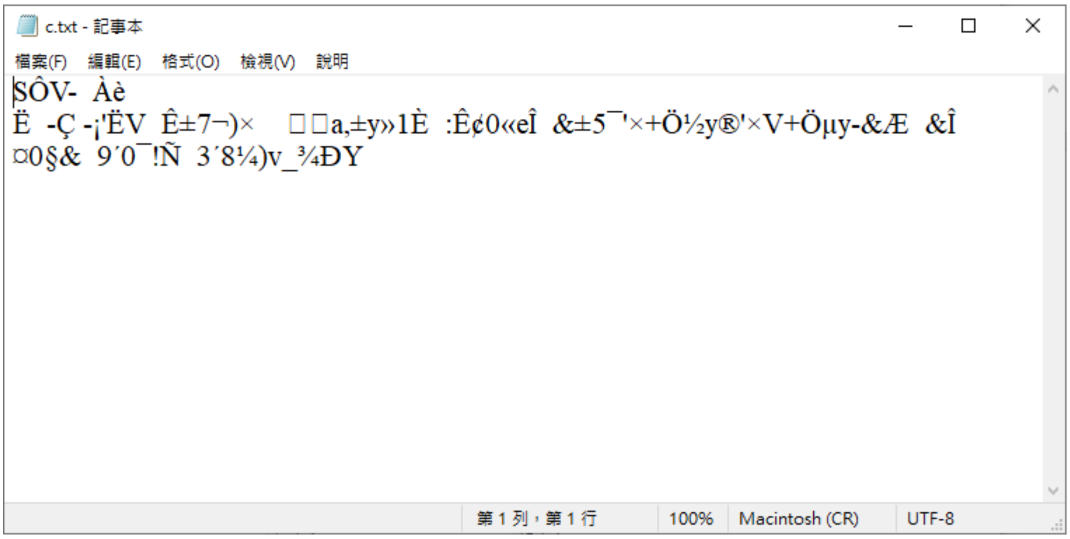



圖 12. 由 ASCII 編碼輸出的密文內容

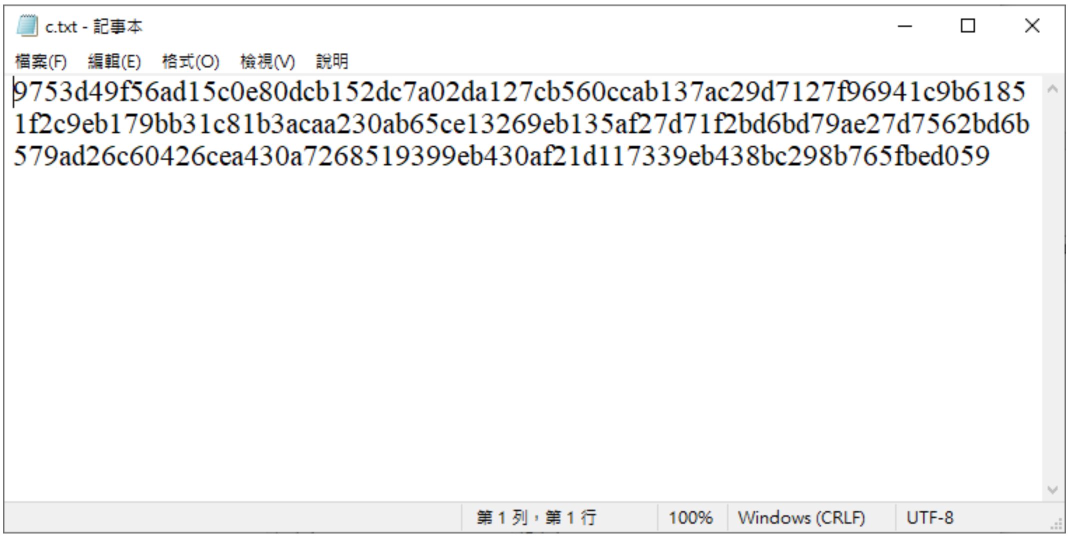



圖 13. 由 Hex 編碼輸出的密文內容

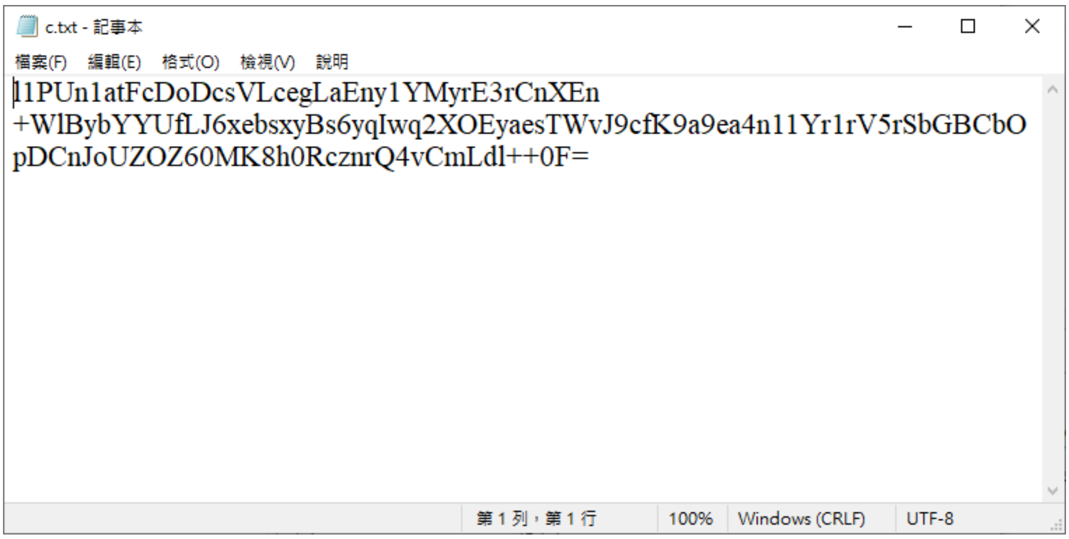



圖 14. 由 Base64 編碼輸出的密文內容

## 肆、程式碼：

### 一、table.py

```
# DES 加解密中所應用的置換表。

class Table:
    # 初始置換表
    IP = [58, 50, 42, 34, 26, 18, 10, 2,
          60, 52, 44, 36, 28, 20, 12, 4,
          62, 54, 46, 38, 30, 22, 14, 6,
          64, 56, 48, 40, 32, 24, 16, 8,
          57, 49, 41, 33, 25, 17, 9,  1,
          59, 51, 43, 35, 27, 19, 11, 3,
          61, 53, 45, 37, 29, 21, 13, 5,
          63, 55, 47, 39, 31, 23, 15, 7]

    # 結尾置換表
    IIP = [40, 8, 48, 16, 56, 24, 64, 32,
           39, 7, 47, 15, 55, 23, 63, 31,
           38, 6, 46, 14, 54, 22, 62, 30,
           37, 5, 45, 13, 53, 21, 61, 29,
           36, 4, 44, 12, 52, 20, 60, 28,
           35, 3, 43, 11, 51, 19, 59, 27,
           34, 2, 42, 10, 50, 18, 58, 26,
           33, 1, 41, 9,  49, 17, 57, 25]

    # 金鑰置換表，將 64 位金鑰變成 56 位
    PC_1 = [57, 49, 41, 33, 25, 17, 9,
            1,  58, 50, 42, 34, 26, 18,
            10, 2,  59, 51, 43, 35, 27,
            19, 11, 3,  60, 52, 44, 36,
            63, 55, 47, 39, 31, 23, 15,
            7,  62, 54, 46, 38, 30, 22,
            14, 6,  61, 53, 45, 37, 29,
            21, 13, 5,  28, 20, 12, 4]

    # 壓縮置換，將 56 位金鑰壓縮成 48 位子金鑰
    PC_2 = [14, 17, 11, 24, 1,  5,
            3,  28, 15, 6,  21, 10,
            23, 19, 12, 4,  26, 8,
            16, 7,  27, 20, 13, 2,
```

```
        41, 52, 31, 37, 47, 55,
        30, 40, 51, 45, 33, 48,
        44, 49, 39, 56, 34, 53,
        46, 42, 50, 36, 29, 32]

# 每輪左移的位數
shiftBits = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

# 擴充置換表，將 32bits 擴充至 48bits
E = [32, 1,   2,   3,   4,   5,
     4,   5,   6,   7,   8,   9,
     8,   9,   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, 30, 31, 32, 1]

# S 盒，每個 S 盒是 4x16 的置換表，6 位 -> 4 位
S = [
    [
        [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
        [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
        [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
        [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]
    ],
    [
        [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
        [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
        [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
        [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]
    ],
    [
        [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
        [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
        [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
        [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]
    ],
    [
        [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
```

```
            [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
            [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
            [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]
        ],
        [
            [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
            [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
            [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
            [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]
        ],
        [
            [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
            [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
            [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
            [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]
        ],
        [
            [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
            [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
            [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
            [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]
        ],
        [
            [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
            [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
            [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
            [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]
        ]
    ]

# P 置换，32 位 -> 32 位
P = [16, 7,  20, 21,
     29, 12, 28, 17,
     1,  15, 23, 26,
     5,  18, 31, 10,
     2,  8,  24, 14,
     32, 27, 3,  9,
     19, 13, 30, 6,
     22, 11, 4,  25]
```

**二、fileProcess.py**

```
# 檔案 or 文字處理

def inputTxt(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        text = f.read()
        return text


def outputTxt(filename, text):
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(text)


# 將 list 轉換成 string
def listToString(s):
    str1 = ''
    return str1.join('%s' % ID for ID in s)


def padding_64(s):
    if len(s) % 64 != 0:
        s += '0' * (64 - len(s) % 64)
    return s


def ASCII(mode, string):
    if mode == 'in':
        encode = ''
        for i in range(len(string)):
            encode += bin(ord(string[i])).replace('0b', '').zfill(8)
        return encode
    if mode == 'out':
        decode = ''
        count = int(len(string) / 8)
        for i in range(count):
            decode += chr(int(string[i * 8:i * 8 + 8], 2))
        return decode
```

```
def Hex(mode, string):
    if mode == 'in':
        decode = ''
        for i in range(len(string)):
            decode += bin(int(string[i], 16)).replace('0b', '').zfill(4)
        return decode
    if mode == 'out':
        decode = ''
        count = int(len(string) / 4)
        for i in range(count):
            decode += hex(int(string[i * 4:i * 4 + 4], 2)).replace('0x', '')
        return decode


def Base64(mode, string):
    index = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
    if mode == 'in':
        decode = ''
        if string[-1] == '=' and string[-2] == '=':
            count = 2
        elif string[-1] == '=':
            count = 4
        else:
            count = 0
        for i in range(len(string)):
            for j in range(len(index)):
                if string[i] == index[j]:
                    decode += bin(j).replace('0b', '').zfill(6)
        decode += '0' * count
        return decode
    if mode == 'out':
        encode = ''

        mod = len(string) % 6
        count = int(len(string) / 6)

        if mod == 2:
            string += '0' * 4
        elif mod == 4:
            string += '0' * 2
```

```
        for i in range(count):
            tmp = index[int(string[i * 6:i * 6 + 6], 2)]
            encode += tmp

        if mod == 2:
            encode += '=' * 2
        elif mod == 4:
            encode += '=' * 1
        return encode


def convert_in(mode, string):
    # 編碼模式 to bin
    encode = ''
    if mode == 'ASCII':
        encode = ASCII('in', string)
    if mode == 'Hex':
        encode = Hex('in', string)
    if mode == 'Base64':
        encode = Base64('in', string)
    return encode


def convert_out(mode, string):
    # bin to 編碼模式
    decode = ''
    # 將 64bits，每 8bits 轉換成 ASCII
    if mode == 'ASCII':
        decode = ASCII('out', string)
    # 將 64bits，每 4bits 轉換成 Hex
    if mode == 'Hex':
        decode = Hex('out', string)
    # 將 64bits，每 6bits 轉換成 Base64
    if mode == 'Base64':
        decode = Base64('out', string)
    return decode
```

**三、subKey.py**

```
# 生成 16 組子金鑰。

from Table import Table


def PC1(key64):
    key56 = []
    for i in Table.PC_1:
        key56.append(int(key64[int(i) - 1]))
    C = key56[:28]
    D = key56[28:]
    return C, D

def PC2(C, D):
    key56 = C + D
    key48 = []
    for i in Table.PC_2:
        key48.append(int((key56[int(i) - 1])))
    return key48

def leftShift(num, C, D):
    Shift = int(Table.shiftBits[num - 1])
    new_C = C[Shift:] + C[:Shift]
    new_D = D[Shift:] + D[:Shift]
    return new_C, new_D

def generateKeys(realKey):
    C = [0] * 17
    D = [0] * 17
    subKey = [0] * 17
    C[0], D[0] = PC1(realKey)
    subKey[0] = PC2(C[0], D[0])
    for i in range(1, 17):
        C[i], D[i] = leftShift(i, C[i - 1], D[i - 1])
        subKey[i] = PC2(C[i], D[i])
    return subKey
```

## 四、Feistel.py

```
# Feistel 函式

from Table import Table


# IP 置換
def IP(key_64):
    # 宣告 new_key(list)，用於存放置換結果。
    new_key = [0] * 64
    # 利用 IP 置換表進行置換。
    for i in range(64):
        new_key[i] = int(key_64[Table.IP[i] - 1])
    # 將置換結果分為左半部及右半部。
    L = new_key[:32]
    R = new_key[32:]
    return L, R

# IIP 逆置換
def IIP(key_64):
    # 宣告 new_key(list)，用於存放置換結果。
    new_key = [0] * 64
    # 利用 IIP 置換表進行置換。
    for i in range(64):
        new_key[i] = int(key_64[Table.IIP[i] - 1])
    return new_key

# 將明文 R 從 32 位 擴展成 48 位
def R_expand(R):
    new_R = [0] * 48
    for i in range(48):
        new_R[i] = int(R[Table.E[i] - 1])
    return new_R

# 將兩 list 做 xor
def xor(list1, list2):
    xor_result = []
    for i in range(len(list1)):
        xor_result.append(int(list1[i]) ^ int(list2[i]))
    return xor_result
```

```
# S 盒置換
def S_Box(xor_result):
    S_result = ''
    for i in range(8):
        tmp = xor_result[i * 6:i * 6 + 6]
        row = int(tmp[0] * 2 + tmp[5])
        col = int(tmp[1] * 8 + tmp[2] * 4 + tmp[3] * 2 + tmp[4])
        S_result += bin(int(Table.S[i][row][col])).replace('0b', '').zfill(4)
    # 將 S_result 從 str 轉型成 list。
    S_result = list(S_result)
    S_result = [int(i) for i in S_result]
    return S_result

# P 置換
def P_Box(S_result):
    P_result = [0] * 32
    for i in range(32):
        P_result[i] = int(S_result[Table.P[i] - 1])
    return P_result

# feistel 函式
def F(R, K):
    new_R = R_expand(R)
    RK_xor = xor(new_R, K)
    s_result = S_Box(RK_xor)
    p_result = P_Box(s_result)
    return p_result

# DES 加密用完整 feistel 函式 (k[1] -> k[16])
def F_16(L, R, K):
    L_16 = [0] * 17
    R_16 = [0] * 17
    L_16[0] = L
    R_16[0] = R
    for i in range(16):
        R_16[i+1] = xor(L_16[i], F(R_16[i], K[i+1]))
        L_16[i+1] = R_16[i]
    result = R_16[16] + L_16[16]
    return result
```

```
# DES 解密用完整 feistel 函式  (k[16] -> k[1])
def IF_16(L, R, K):
    L_16 = [0] * 17
    R_16 = [0] * 17
    L_16[0] = L
    R_16[0] = R
    for i in range(16):
        R_16[i+1] = xor(L_16[i], F(R_16[i], K[16 - i]))
        L_16[i+1] = R_16[i]
    result = R_16[16] + L_16[16]
    return result
```

**五、DES_Func.py**

```
# DES 加解密函式

from math import ceil

from subKey import *
from Feistel import *
from fileProcess import *


def Encrypt(mode, encode, plainTxt, keyTxt, cipherTxt, IVTxt=None):
    # 讀取明文(字元)
    plainText = inputTxt(plainTxt)
    # 宣告暫存密文
    temp = ''

    # 讀取金鑰
    k = convert_in('ASCII', inputTxt(keyTxt))
    # 生成 16 組子金鑰
    key_16 = generateKeys(k)

    # 計算總共需要跑幾次
    # math.ceil()：無條件進位
    count = ceil(len(plainText) / 8)

    # 若有設定向量檔案的話，則讀取其值。
    if IVTxt is not None:
        # 讀取初始向量(字元)
        IVText = inputTxt(IVTxt)
        # 讀取初始向量(64bits)，若初始向量長度不足 64bits，則 padding 0 至 64bits
        IV = padding_64(convert_in('ASCII', IVText))

    # 宣告向量(儲存上一輪數據(64bits))
    vector = []

    for i in range(count):
        # 讀取明文(64bits)，若明文長度不足 64bits，則 padding 0 至 64bits
        P = padding_64(convert_in('ASCII', plainText[i * 8:i * 8 + 8]))
```

```
# DES_ECB 加密
# Ci = Ek(Pi)
if mode == 'ECB':
    # 使用 Feistel，生成密文(64bits)
    L, R = IP(P)
    result = listToString(IIP(F_16(L, R, key_16)))

# DES_CBC 加密
# Ci = Ek(Ci-1 ⊕ Pi), C0 = IV
if mode == 'CBC':
    # 將明文與向量進行 XOR 處理
    if i == 0:
        new_M = xor(P, IV)
    else:
        new_M = xor(P, vector[i - 1])

    # 使用 Feistel，生成密文(64bits)
    L, R = IP(new_M)
    result = listToString(IIP(F_16(L, R, key_16)))

    # 將密文存入 vector 中，以便下一輪的調用
    vector.append(result)

# DES_CFB 加密
# Ci = Ek(Ci-1) ⊕ Pi, C0 = IV
if mode == 'CFB':
    if i == 0:
        L, R = IP(IV)
    else:
        L, R = IP(vector[i - 1])

    Key_IV_xor = listToString(IIP(F_16(L, R, key_16)))

    # 將金鑰與向量加密結果與明文做 XOR
    result = listToString(xor(P, Key_IV_xor))

    # 將密文存入 vector 中，以便下一輪的調用
    vector.append(result)
```

```
        # DES_OFB 加密
        # Ci = Pi ⊕ Ek(Oi-1), O0 = IV
        if mode == 'OFB':
            if i == 0:
                L, R = IP(IV)
            else:
                L, R = IP(vector[i - 1])

            tmp = listToString(IIP(F_16(L, R, key_16)))

            # 將密文存入 vector 中，以便下一輪的調用
            vector.append(tmp)

            # 將金鑰與向量加密結果與明文做 XOR
            result = listToString(xor(P, tmp))

        # DES_CTR 加密
        # Ci = Pi ⊕ Ek(counter)
        if mode == 'CTR':
            L, R = IP(IV)
            tmp = listToString(IIP(F_16(L, R, key_16)))

            IV = bin(int(IV, 2) + 1).replace('0b', '').zfill(64)
            # 將金鑰與向量加密結果與明文做 XOR
            result = listToString(xor(P, tmp))

        temp += result
    C = convert_out(encode, temp)
    # 將密文寫入.txt 文件中
    outputTxt(cipherTxt, C)



def Decrypt(mode, encode, plainTxt, keyTxt, cipherTxt, IVTxt=None):
    # 讀取密文
    ciphertext = inputTxt(cipherTxt)
    # 宣告空明文
    temp = ''

    # 讀取金鑰
    k = convert_in('ASCII', inputTxt(keyTxt))
```

```
# 生成 16 組子金鑰
key_16 = generateKeys(k)

# 計算總共需要跑幾次
# math.ceil()：無條件進位
if encode == 'ASCII':
    count = ceil(len(ciphertext) / 8)
if encode == 'Hex':
    count = ceil(len(ciphertext) / 16)
if encode == 'Base64':
    ciphertext = convert_in('Base64', ciphertext)
    count = ceil(len(ciphertext) / 64)

# 若有設定向量檔案的話，則讀取其值。
if IVTxt is not None:
    # 讀取初始向量(字元)
    IVText = inputTxt(IVTxt)
    # 讀取初始向量(64bits)，若初始向量長度不足 64bits，則 padding 0 至 64bits
    IV = padding_64(convert_in('ASCII', IVText))

# 宣告向量(儲存上一輪密文(64bits))
vector = []
for i in range(count):
    # 讀取密文(64Bits)
    if encode == 'ASCII':
        C = convert_in('ASCII', ciphertext[i * 8:i * 8 + 8])
    if encode == 'Hex':
        C = convert_in('Hex', ciphertext[i * 16:i * 16 + 16])
    if encode == 'Base64':
        C = ciphertext[i * 64:i * 64 + 64]

    # DES_ECB 解密
    # Pi = Dk(Ci)
    if mode == 'ECB':
        # 使用 Feistel，生成密文(64bits)
        # 使用 Feistel，生成明文(64Bin)
        L, R = IP(C)
        result = listToString(IIP(IF_16(L, R, key_16)))
```

```
# DES_CBC 解密
# Pi = Dk(Ci) ⊕ Ci-1 , C0 = IV
if mode == 'CBC':
    # 將密文存入 vector 中，以便下一輪的調用
    vector.append(C)
    # 使用 Feistel，生成明文(64Bits)
    L, R = IP(C)
    Ci = listToString(IIP(IF_16(L, R, key_16)))
    # 將明文與向量進行 XOR 處理
    if i == 0:
        result = xor(Ci, IV)
    else:
        result = xor(Ci, vector[i - 1])
    result = listToString(result)

# DES_CFB 解密
# Pi = Ek(Ci-1) ⊕ Ci, C0 = IV
if mode == 'CFB':
    # 將密文存入 vector 中，以便下一輪的調用
    vector.append(C)
    # 使用 Feistel，生成明文(64Bits)
    if i == 0:
        L, R = IP(IV)
    else:
        L, R = IP(list(vector[i - 1]))
    EC = listToString(IIP(F_16(L, R, key_16)))
    # 將明文與向量進行 XOR 處理
    result = listToString(xor(EC, C))

# DES_OFB 解密
# Pi = Ci ⊕ Ek(Oi-1), O0 = IV
if mode == 'OFB':
    # 使用 Feistel，生成明文(64Bits)
    if i == 0:
        L, R = IP(IV)
    else:
        L, R = IP(list(vector[i - 1]))
    EO = listToString(IIP(F_16(L, R, key_16)))
    # 將密文存入 vector 中，以便下一輪的調用
    vector.append(EO)
```

```
        # 將明文與向量進行 XOR 處理
        result = listToString(xor(EO, C))

    # DES_CTR 加密
    # Ci = Pi ⊕ Ek(counter)
    if mode == 'CTR':
        # 使用 Feistel，生成明文(64Bits)
        L, R = IP(IV)
        EO = listToString(IIP(F_16(L, R, key_16)))

        IV = bin(int(IV, 2) + 1).replace('0b', '').zfill(64)
        # 將明文與向量進行 XOR 處理
        result = listToString(xor(EO, C))

    temp += result

P = convert_out('ASCII', temp)
# 將密文寫入.txt 文件中
outputTxt(plainTxt, P)
```

## 六、DES.py

```
# 主程式(GUI 模式)。
from DES_Func import *
from tkinter import *
from tkinter import filedialog
from os import startfile

root = Tk()
# 設置視窗標題為 DES
root.title('DES')
# 設置視窗大小為 800x600
root.geometry('800x600')
# 固定視窗大小為 800x600
root.resizable(0, 0)
# 設置視窗背景色為 白色
root.configure(background='white')

# 獲取 明文文件路徑
def get_plain_filePath():
    filename = filedialog.askopenfilename()
    plain_file_label.configure(text=filename)
    return filename

# 獲取 金鑰文件路徑
def get_key_filePath():
    filename = filedialog.askopenfilename()
    key_file_label.configure(text=filename)
    return filename

# 獲取 向量文件路徑
def get_IV_filePath():
    filename = filedialog.askopenfilename()
    IV_file_label.configure(text=filename)
    return filename

# 獲取 密文文件路徑
def get_cipher_filePath():
    filename = filedialog.askopenfilename()
    cipher_file_label.configure(text=filename)
    return filename
```

```
# 打開 明文
def open_plain_file():
    filePath = plain_file_label['text']
    startfile(filePath)

# 打開 金鑰
def open_key_file():
    filePath = key_file_label['text']
    startfile(filePath)

# 打開 向量
def open_IV_file():
    filePath = IV_file_label['text']
    startfile(filePath)

# 打開 密文
def open_cipher_file():
    filePath = cipher_file_label['text']
    startfile(filePath)

# 按下加密按鈕後，執行此函式
def des_encrypt():
    # 讀取 DES 模式
    mode_sel = modes[var.get()]
    # 讀取編碼模式
    encode_sel = encode[var2.get()]
    # 讀取 明文路徑
    plain = plain_file_label['text']
    # 讀取 金鑰路徑
    key = key_file_label['text']
    # 讀取 密文路徑
    cipher = cipher_file_label['text']

    # 讀取 向量路徑
    # 若 DES 模式為 ECB，則向量路徑設置為空
    if mode_sel == 'ECB':
        IV = None
    else:
        IV = IV_file_label['text']
```

```
    # 執行 DES 加密
    Encrypt(mode_sel, encode_sel, plain, key, cipher, IV)
    # 設置 state 標籤為'加密成功'
    state_label.configure(text='加密成功')


# 按下解密按鈕後，執行此函式
def des_decrypt():
    # 讀取 DES 模式
    mode_sel = modes[var.get()]
    # 讀取編碼模式
    encode_sel = encode[var2.get()]
    # 讀取 明文路徑
    plain = plain_file_label['text']
    # 讀取 金鑰路徑
    key = key_file_label['text']
    # 讀取 密文路徑
    cipher = cipher_file_label['text']

    # 讀取 向量路徑
    # 若 DES 模式為 ECB，則向量路徑設置為空
    if mode_sel == 'ECB':
        IV = None
    else:
        IV = IV_file_label['text']

    # 執行 DES 解密
    Decrypt(mode_sel, encode_sel, plain, key, cipher, IV)
    # 設置 state 標籤為'解密成功'
    state_label.configure(text='解密成功')


plain_label = Label(root, text='DES 明文', fg='black', bg='white', height=1, width=8, anchor='center',
font='Helvetica 18')
plain_file_label = Label(root, text='', font='Helvetica 12', height=1, width=36, bg='white', anchor='center')
plain_file_btn = Button(root, text='瀏覽檔案', bg='white', command=get_plain_filePath)
open_plain_file_btn = Button(root, text='開啟檔案', bg='white', command=open_plain_file)

key_label = Label(root, text='DES 金鑰', fg='black', bg='white', height=1, width=8, anchor='center', font='Helvetica
18')
```

```
key_file_label = Label(root, text='', font='Helvetica 12', height=1, width=36, bg='white', anchor='center')
key_file_btn = Button(root, text='瀏覽檔案', bg='white', command=get_key_filePath)
open_key_file_btn = Button(root, text='開啟檔案', bg='white', command=open_key_file)

IV_label = Label(root, text='DES 向量', fg='black', bg='white', height=1, width=8, anchor='center', font='Helvetica 18')
IV_file_label = Label(root, text='', font='Helvetica 12', height=1, width=36, bg='white', anchor='center')
IV_file_btn = Button(root, text='瀏覽檔案', bg='white', command=get_IV_filePath)
open_IV_file_btn = Button(root, text='開啟檔案', bg='white', command=open_IV_file)

cipher_label = Label(root, text='DES 密文', fg='black', bg='white', height=1, width=8, anchor='center', font='Helvetica 18')
cipher_file_label = Label(root, text='', font='Helvetica 12', height=1, width=36, bg='white', anchor='center')
cipher_file_btn = Button(root, text='瀏覽檔案', bg='white', command=get_cipher_filePath)
open_cipher_file_btn = Button(root, text='開啟檔案', bg='white', command=open_cipher_file)

plain_label.place(x=320, y=0)
plain_file_label.place(x=200, y=40)
plain_file_btn.place(x=550, y=40)
open_plain_file_btn.place(x=620, y=40)

key_label.place(x=320, y=80)
key_file_label.place(x=200, y=120)
key_file_btn.place(x=550, y=120)
open_key_file_btn.place(x=620, y=120)

IV_label.place(x=320, y=160)
IV_file_label.place(x=200, y=200)
IV_file_btn.place(x=550, y=200)
open_IV_file_btn.place(x=620, y=200)

cipher_label.place(x=320, y=240)
cipher_file_label.place(x=200, y=280)
cipher_file_btn.place(x=550, y=280)
open_cipher_file_btn.place(x=620, y=280)

# 設置 DES 模式選項按鈕
modes = {0: 'ECB', 1: 'CBC', 2: 'CFB', 3: 'OFB', 4: 'CTR'}
var = IntVar()
var.set(0)
```

```
x, y = 240, 320
for val, mode in modes.items():
    Radiobutton(root, text=mode, variable=var, value=val, bg='white').place(x=x, y=y)
    x += 75

# 設置編碼模式選項按鈕
encode = {0: 'ASCII', 1: 'Hex', 2: 'Base64'}
var2 = IntVar()
var2.set(0)

x, y = 240, 360
for val, mode in encode.items():
    Radiobutton(root, text=mode, variable=var2, value=val, bg='white').place(x=x, y=y)
    x += 75

mode_label = Label(root, text='加密模式：', fg='black', bg='white', height=1, width=8, anchor='center',
font='Helvetica 12')
encode_label = Label(root, text='編碼模式：', fg='black', bg='white', height=1, width=8, anchor='center',
font='Helvetica 12')
mode_label.place(x=150, y=320)
encode_label.place(x=150, y=360)

encrypt_btn = Button(root, text='加密', command=des_encrypt)
encrypt_btn.place(x=275, y=400)

decrypt_btn = Button(root, text='解密', command=des_decrypt)
decrypt_btn.place(x=450, y=400)

state_label = Label(root, text='', bg='white', font='Helvetica 12')
state_label.place(x=350, y=440)

root.mainloop()
```

**伍、心得：**

在本次的期中專題中，遇到的問題主要有兩個部分。第一部分是對於 DES 算法以及 Python 的不熟悉，導致在撰寫程式時會有各種莫名的錯誤，例如在 Feistel 函式中會因為格式上的不同，導致不斷報錯；抑或是在編寫程式碼時，沒有注意到細節，導致在加密過程中，無法輸出想要的結果；又或者是在封裝副函式時不夠精簡，導致大量的程式碼重複利用，造成儲存空間及應用上的冗餘。

第兩部分則是 GUI 圖形介面的封裝。由於在之前撰寫 Python 時，都是以命令列為優秀控制，並沒有將其封裝成 GUI 介面的經歷。故在這次專題中，需要從零開始，經由課外書籍、網際網路補充設定 GUI 的相關知識，並且多次嘗試後，選擇適當的函式，並不斷的調試成最佳化。對我而言，這次封裝 GUI 是一次充實的經歷，使我對於基本的圖形介面操作有一定程度的了解。

總體來說，在本次專題中，需要改進的部分有加速程式碼的執行效率，撰寫程式碼的時間效率，以及優化 GUI 圖形介面使其更加好看。若之後還有相關的專題，希望自己能夠完成的更加良好。