tags: `2024 年 下學期讀書計畫` `Introduction to Cryptography`

Introduction to Cryptography HW2 DES

學習資源 (References)

- DES 密碼系統 (https://www.tsnien.idv.tw/Security_WebBook/security.htm)
- DES 解說影片 (https://www.youtube.com/watch?v=qkBisYq8ils&t=761s)

debug 好工具

- 每步驟驗算解答: Virtual Labs (https://virtual-labs.github.io/exp-encryption-plaintext-using-des-au/simulation.html)
- 計算 bits 數量: 字數計算&文字計數 (https://www.ifreesite.com/wordcount/)
- 檢驗兩 bits array 是否相等: 編輯距離計算機 (https://zh.planetcalc.com/1721/)
- 16 進位轉成 2 進位: 數字系統換算器 (https://www.digikey.tw/zh/resources/conversion-calculators/conversion-calculator-number-conversion)

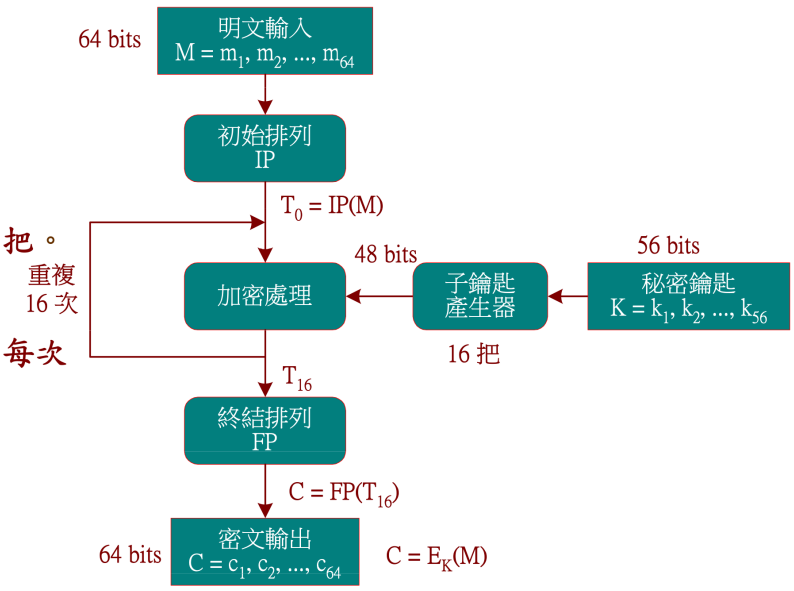
概念流程圖

大架構

- 我們先來看大架構
- 先把比較瑣碎的東西當成一個 black boxs

↓ DES 演算法流程

1. 明文資料分割
2. 初始變換 (Initial Permutation, IP)
3. 選擇子鑰匙，共有 16 把。
4. 加密處理
5. 重複加密處理 16 次，每次使用不同的子鑰匙。
6. 終結變換 (Final Permutation)



- 在這邊你要知道三件事情

1. 明文輸入後 (利用 IP), 及密碼輸出前 (利用 FP), 記得都要經過排列
2. 獲得秘密鑰匙 (64 bits) 後, 我們可以把它轉換成小鑰匙 (48 bits), 接著產生 16 把由小鑰匙產生出來的子鑰匙
3. 我們會有一個神奇的 black box, input 為 16 把子鑰匙, 及排列過後的明文, 協助我們加密得到密文, 他需要經過 16 次

- 所以我們分成三個 Part 來討論

Part 1 排列

- 我們先寫一個 permutation 函數
- 他會協助我們把 64 bits 經過 P[64] 這個 函數重新排列
- 我們只要帶入 P = IP, P = PF, 就能得到我們要的排列結果
- 對了有的排列函數很 xx, 會是 1-base
- 記得要看一下是 1-base 還是 0-base
- table 都是固定的, 照刻就好
- 長下面這樣

初始與終結排列 (換位加密)

(a) 初始排列 (IP) 內容

T_0 輸出	明文區塊 M 輸入							
1- 8	58	50	42	34	26	18	10	2
9 - 16	60	52	44	36	28	20	12	4
17 - 24	62	54	46	38	30	22	14	6
25 - 32	64	56	48	40	32	24	16	8
33 - 40	57	49	41	33	25	17	9	1
41 - 48	59	51	43	35	27	19	11	3
49 - 56	61	53	45	37	29	21	13	5
57 - 64	63	55	47	39	31	23	15	7

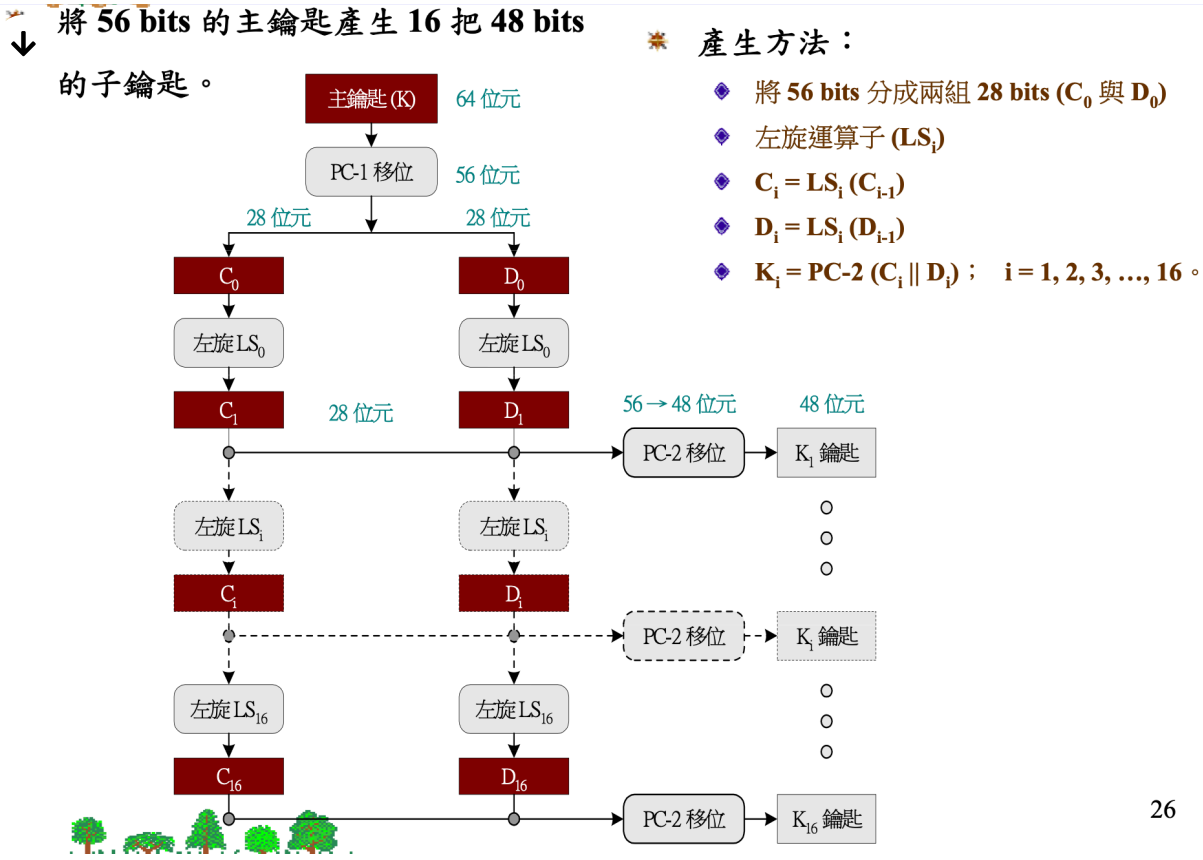
(b) 終結排列 (FP) 內容

密文 C 輸出	T_{16} 輸入							
1- 8	40	8	48	16	56	24	64	32
9 - 16	39	7	47	15	55	23	63	31
17 - 24	38	6	46	14	54	22	62	30
25 - 32	37	5	45	13	53	21	61	29
33 - 40	36	4	44	12	52	20	60	28
41 - 48	35	3	43	11	51	19	59	27
49 - 56	34	2	42	10	50	18	58	26
57 - 64	33	1	41	9	49	17	57	25

```
1
2  const int IP[64] = {58, 50, 42, 34, 26, 18, 10, 2,
3                      60, 52, 44, 36, 28, 20, 12, 4,
4                      62, 54, 46, 38, 30, 22, 14, 6,
5                      64, 56, 48, 40, 32, 24, 16, 8,
6                      57, 49, 41, 33, 25, 17, 9,  1,
7                      59, 51, 43, 35, 27, 19, 11, 3,
8                      61, 53, 45, 37, 29, 21, 13, 5,
9                      63, 55, 47, 39, 31, 23, 15, 7 };
10
11 const int PF[32] = {16, 7, 20, 21,
12                     29, 12, 28, 17,
13                     1, 15, 23, 26,
14                     5, 18, 31, 10,
15                     2, 8, 24, 14,
16                     32, 27, 3, 9,
17                     19, 13, 30, 6,
18                     22, 11, 4, 25};
19
20 vector<int> permutation(vector<int> bits, const int P[64])
21 {
22         vector<int> after_permutation_bits(64);
23         for (int i = 0; i < 64; i++)
24         {
25                 after_permutation_bits[i] = bits[P[i] - 1];
26         }
27         return after_permutation_bits;
28 }
```

Part 2 得到子鑰匙

- 我們先來看個流程圖



- 接著我們看個子鑰匙的 pseudo code

```
Step 1. 輸入主鑰匙 mainKey (64 bits)
Step 2. 把主鑰匙 mainKey 經過 PC1 移位得到 mainKey' (56 bits)
Step 3. mainKey' 把左半部記做 C_0 (28 bits), 右邊半部記做 D_0 (28 bits)
Step 4. 把 C_0, D_0, 按照旋轉左移表得到 C_1, D_1
Step 5. 把 C_1, D_1 連接起來行成 temp_1
STep 6. 把 temp_1 根據 PC2 移位得到 subkey_1
.
.
.
Step. 把 C_k, D_k, 按照旋轉左移表得到 C_k+1, D_k+1
Step. 把 C_k+1, D_k+1 連接起來行成 temp_k+1
STep. 把 temp_k+1 根據 PC2 移位得到 subkey_k+1
.
.
.
重複直到做好最後一把鑰匙 k_16
```

- 然後我們來轉換成 C++ code
- 從上圖可以觀察到: 其實 C_i, D_i 做的操作就只是往左移而已
- 且合併 \(C_i, D_i\), 產生 \(K_{i-1}\) 子鑰匙都會經過 PC2 重新排列

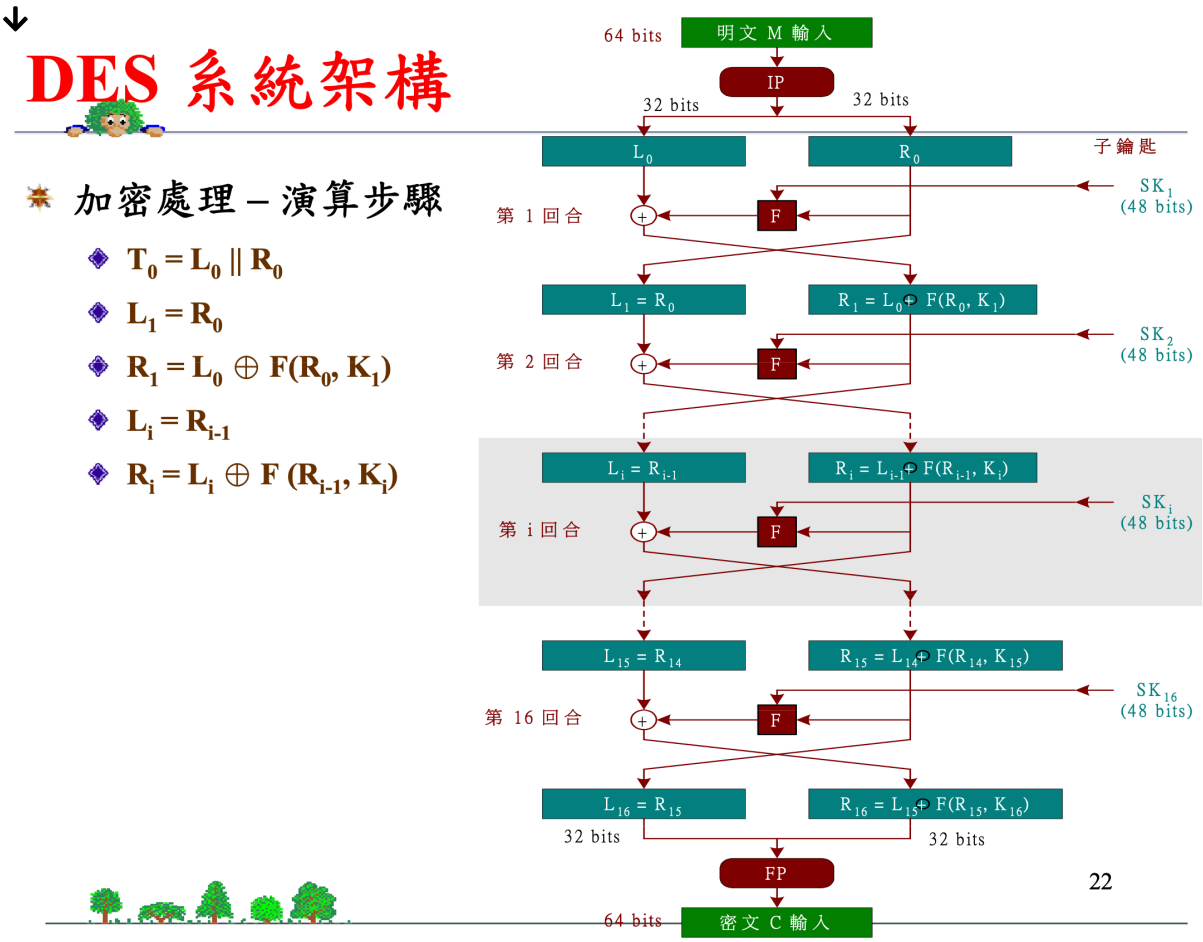
```
const int PC1[56] = {57, 49, 41, 33, 25, 17, 9,
                     1, 58, 50, 42, 34, 26, 18,
                     10, 2, 59, 51, 43, 35, 27,
                     19, 11, 3, 60, 52, 44, 36,
                     63, 55, 47, 39, 31, 23, 15,
                     7, 62, 54, 46, 38, 30, 22,
                     14, 6, 61, 53, 45, 37, 29,
                     21, 13, 5, 28, 20, 12, 4};

const int PC2[48] = {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};

const int num_leftShift[16] = {1, 1, 2, 2, 2, 2, 2, 2,
                               1, 2, 2, 2, 2, 2, 2, 1}; // number of bits to

// Step 2: Build subkey
        vector<int> subkey[16];

        //     initial C, D
        vector<vector<int> > C(17), D(17);
        for (int i = 0; i < 16; i++)
        {
                C[i].resize(28);
                D[i].resize(28);
                subkey[i].resize(48);
        }
        C[16].resize(28);
        D[16].resize(28);
        for (int i = 0; i < 28; i++)
        {
                C[0][i] = mainKey[PC1[i] - 1];
                D[0][i] = mainKey[PC1[i + 28] - 1];
        }
        //   iteration get C_i before permutation
        vector<int> CD(56);
        for (int i = 1; i < 17; i++)
        {
                for (int j = 0; j < 28 - num_leftShift[i - 1]; j++)
                {
                        C[i][j] = C[i - 1][j + num_leftShift[i - 1]];
                        D[i][j] = D[i - 1][j + num_leftShift[i - 1]];
                }
                for (int j = 28 - num_leftShift[i - 1], k = 0;
                         k < num_leftShift[i - 1]; j++, k++)
                {
                        C[i][j] = C[i - 1][k];
                        D[i][j] = D[i - 1][k];
                }
                for (int j = 0; j < 28; j++)
                {
                        CD[j] = C[i][j];
                        CD[j + 28] = D[i][j];
                }
```

```
60                  for (int j = 0; j < 48; j++)
61                  {
62                          subkey[i - 1][j] = CD[PC2[j] - 1];
63                  }
64          }
```

Part 3 加密處理 black box

- 由於細節很多
- 我們一樣先來看大架構

大架構



- 第 \(i\) 輪的輸入會有 \(L_i, R_i\) 以及子鑰匙 subkey\(_i\)
- 然後輸出會是下一輪的 \(L_{i+1}, R_{i+1}\)
- 我們可以把 \(L_i, R_i\) 寫成遞迴式
  - \(L_i = R_{i-1}\)
  - \(R_i = L_{i-1}\oplus F(R_{i-1},~\text{subkey}_i)\)
- 這裡的 \(F\) 可以把它想成小 black box, 他會吃入 \(R\) 和 subkey 吐出 32 bits 的數字
- 然後一路我們可以算到 \(L_{16}, R_{16}\)
- 接著 swap(\(L_{16}, R_{16}\)) 交換變成 \(L_{17}, R_{17}\)
- (記得一定要寫, 這很容易忘記)
- 然後再做最後的排列 FP

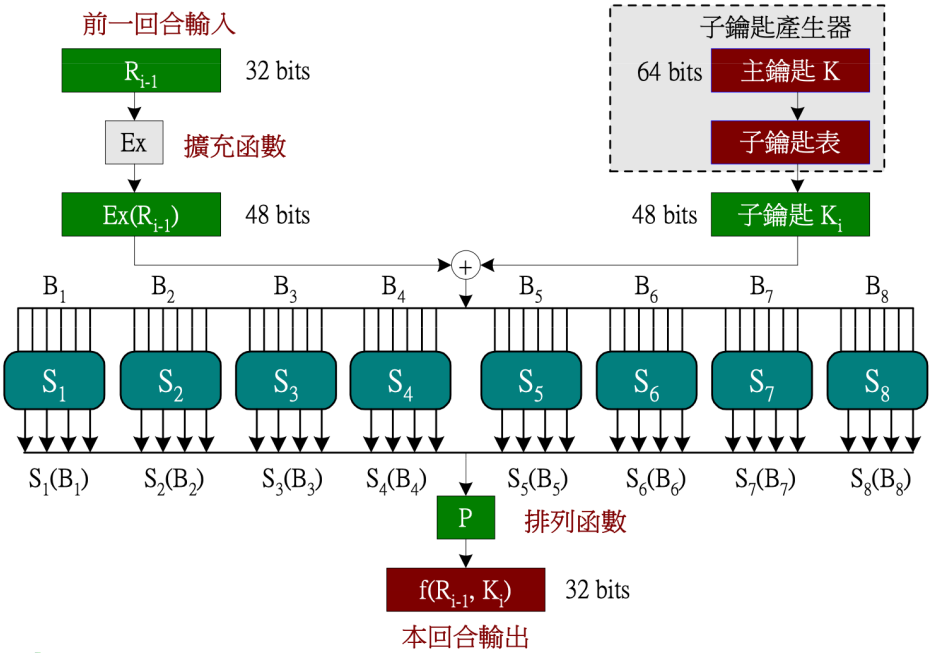
```
// Step 3: calculate the L and R
        //     3-1: Build initial L0, R0
        vector<int> L(32), R(32);
        for (int i = 0; i < 32; i++)
        {
                L[i] = plainText[i];
                R[i] = plainText[i + 32];
        }

        for (int m = 0; m < 16; m++)
        {
                pair<vector<int>, vector<int> > upd_L_R = round(L, R, subkey[
                L = upd_L_R.first;
                R = upd_L_R.second;
        }

        // Step 4: Get the final L_16 and R_16,
        //                 and get cipherText before permutation
        for (int i = 0; i < 32; i++)
        {
                plainText[i] = R[i];
                plainText[i + 32] = L[i];
        }
```

F function

- 我們一樣先來看流程圖

↓ 加密處理 (取代加密)

- $F(R_{i-1}, K_i)$ 函數實現



23

- 我們吃入 \(P_i, \text{subkey}_{i-1}\), 目標輸出 \(F(R_{i-1},~\text{subkey}_i)\)

- 但有個問題是現在 R_{i-1} 是 32 bits, 但 subkey_i 卻有 48 個 bits, 無法做 XOR
- 所以我們利用 extendVector 這個函數協助我們把 32 bits 的 R_{i-1} 變成 48 bits 的 R_{i-1}
- 接著我們把 48 bits 的 R_{i-1} 和 subkey_i 做 XOR
- 得到一個 48 bits 的數字
- 接著我們必須把這個數字由左至右, 連續地以 6 為單位分成一等份 (共有 8 份)
- 我們記做 $B_1,B_2,\cdots,B_8$
- 然後透過 S function, 幫我們把 6 bits 的 B_i 變成 4 bits 的 $S(B_i)$
- 然後把 $S(B_1), S(B_2), \cdots, S(B_6)$ 合成成一個 bits, 接著經過排列函數 P 得到 $F(R_{i-1},~\text{subkey}_i)$
- 寫成程式碼如下

```
vector<int> F_and_permutation(vector<int> R, vector<int> subkey)
{
        R = extendVector(R);
        for (int i = 0; i < 48; i++)
        {
                R[i] ^= subkey[i];
        }

        vector<int> rev(32);
        for (int i = 0; i < 8; i++)
        {
                vector<int> slice_R(6);
                for (int j = 0; j < 6; j++)
                {
                        slice_R[j] = R[i * 6 + j];
                }
                vector<int> temp_S = S(i, slice_R);
                for (int j = 0; j < 4; j++)
                {
                        rev[i * 4 + j] = temp_S[j];
                }
        }
        vector<int> rev_permutation(32);
        for (int i = 0; i < 32; i++)
        {
                rev_permutation[i] = rev[PF[i] - 1];
        }
        return rev_permutation;
}
```

- Q1: 如何計算 extendVector
- Q2: 如何計算 S function

extendVector

- 按照擴充函數增加 bits 就可以了

- 如下圖所示

↓ 加密處理

- **(C) Ex(Ri-1) 擴充函數**
- **(D) 排列函數 P**

$Ex(R_{i-1})$ 輸出	R_{i-1} 輸入					
1 - 6	32	1	2	3	4	5
7 - 12	4	5	6	7	8	9
13 - 18	8	9	10	11	12	13
19 - 24	12	13	14	15	16	17
25 - 30	16	17	18	19	20	21
31 - 36	20	21	22	23	24	25
37 - 42	24	25	26	27	28	29
43 - 48	28	29	30	31	32	1

$F(R_{i-1}, K_i)$ 輸出	S(B) 輸入			
1 - 4	16	7	20	21
5 - 8	29	12	28	17
9 - 12	1	15	23	26
13 - 16	5	18	31	10
17 - 20	2	8	24	14
21 - 24	32	27	3	9
24 - 28	19	13	30	6
29 - 32	22	11	4	25

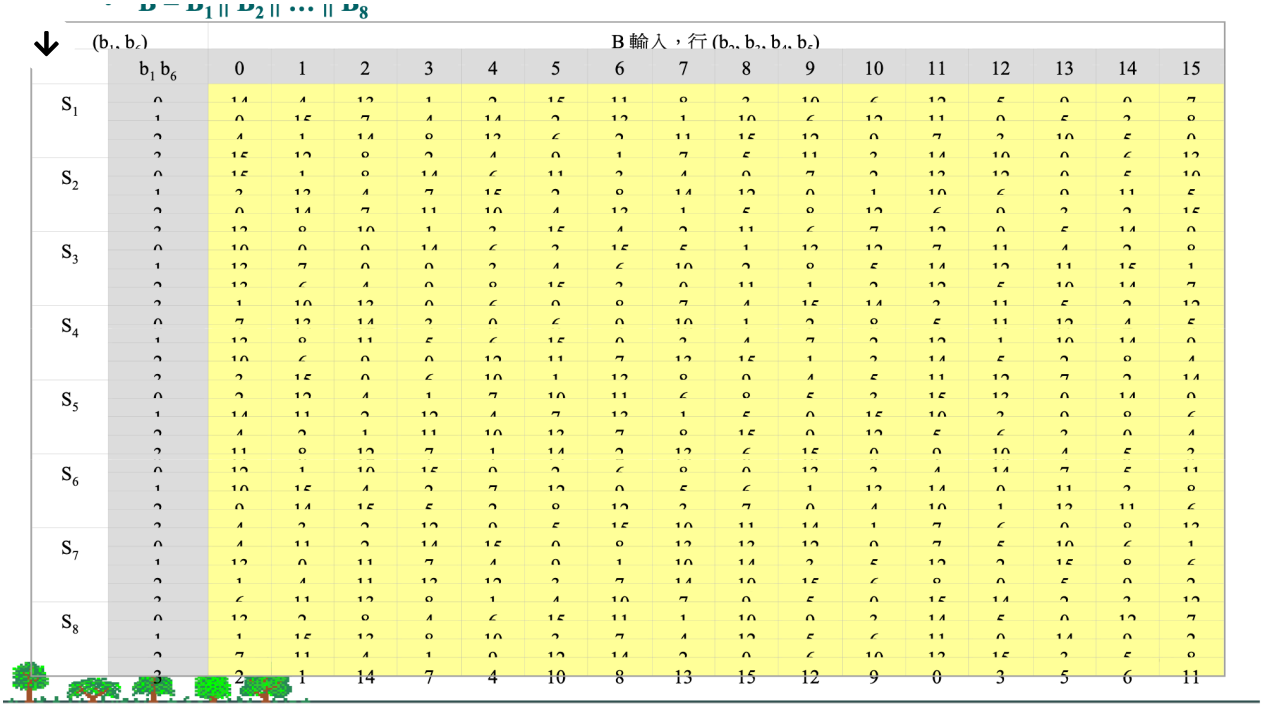
- 寫成程式碼如下

```
const int expand_table[48] = {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};

vector<int> extendVector(vector<int> R)
{
        vector<int> rev(48);
        for (int i = 0; i < 48; i++)
        {
                rev[i] = R[expand_table[i] - 1];
        }
        return rev;
}
```

S function

- 我們把最前面和最後面兩個組在一起 row
- 把中間的 bits 組在一起當成 col
- 然後找對應到 S_box 的 S_box[k][row][col] 的位置
- 其中 k 代表現在是在算 $S(B_k)$

$B = B_1 \| B_2 \| \cdots \| B_8$



	(b_1, b_6)	B 輸入，行 (b_2, b_3, b_4, b_5)															
	b_1 b_6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

- 寫成程式碼如下

```
const int s_box[8][4][16] = {{{14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9
                                {0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5
                                {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10
                                {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0,
                               {{15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0,
                                {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9,
                                {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3,
                                {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5,
                               {{10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4
                                {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11,
                                {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10,
                                {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5,
                               {{7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12,
                                {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10,
                                {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2
                                {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7,
                               {{2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0,
                                {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9
                                {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3,
                                {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4
                               {{12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7,
                                {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11
                                {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13,
                                {4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0,
                               {{4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10
                                {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15
                                {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5
                                {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2,
                               {{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0,
                                {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14
                                {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3
                                {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5,

vector<int> S(int k, vector<int> slice_R)
{
        int num = s_box[k][slice_R[0] * 2 + slice_R[5]][slice_R[1] * 8 + slic
        vector<int> rev(4);
        for (int i = 3; i >= 0; i--)
        {
                rev[i] = num % 2;
                num /= 2;
        }
        return rev;
}
```

題目

題目說明

This homework is to implement DES, which encrypts a 64-bit plaintext block to a 64- bit ciphertext block with a key of 64 bits (with parity bits).Do not call crypto library directly since you need to modify the code during the on-site test.

題目格式

5 ordered pairs of key and plaintext, one in each line, such as, “12345678 Pachinko”. Each character is interpreted as its 8 bit-ASCII code, e.g., ‘A’ = 41 (Hex)

Sample Input

12345678 Pachinko
11111111 abcdefgh
33333333 EFGHabcd
98989898 NYCUhwhw
67766776 CryptoPP

Sample Output

C45077C10E08B3D0
7873EDA876CA0FEA
EC17FEF37EBD566A
051D18E9939892D3
E29E7F4FD8AFAB4B

演算法流程

- 以下是我們的演算法流程
- Step 1: Permutate the plainText
- Step 2: Build subkey
- Step 3: calculate the Li and Ri
- Step 4: Get the cipherText before permutation
- Step 5: Get cipherText

AC Code

有 debugger 版本

AC 畫面

391987	1823. Homework 2	Hsiuyee Liao	4	3408	AC	Homework 2	10558	200	2024-03-13 00:37:59

▶ AC Code

去掉 debugger 版本

AC 畫面

391987	1823. Homework 2	Hsiuyee Liao	4	3408	AC	Homework 2	10558	200	2024-03-13 00:37:59

▶ AC Code