資訊安全導論 HW2

B10815044 謝鈞曜 解密

B10815057 廖聖郝 加密

加密

讀取輸入，用 strtoull 讀取 16 進制數字，放到 bitset 裡

```
    bitset<64> plaintext;
    bitset<64> key64;
    //read parameter to bitset
    for (int i = 1; i < 5; i++) {
        if (strcmp(argv[i], "-i") == 0) {
            plaintext = bitset<64>(strtoull(argv[i + 1], NULL, 16));
        }
        else if (strcmp(argv[i], "-k") == 0) {
            key64 = bitset<64>(strtoull(argv[i + 1], NULL, 16));
        }
    }
```

Permutation function，輸出的每一位元為輸入經過 index 對應 table 轉換

因為 bitset 與 table 的位元順序是相反的，所以要做一些特別處理

```
template<int inSize,int outSize>
bitset<outSize> permutation(bitset<inSize> plaintext, vector<int> perm)
{
    //table is reverse order, so reverse permutation table back
    reverse(perm.begin(), perm.end());
    bitset<outSize> result;
    for (int i = 0; i < outSize; i++) {
        //index in table is base on reverse order, so minus from input
size to reverse back
        result[i] = plaintext[inSize - perm[i]];
    }
    return result;
}
```

根據 DES 演算法，將 plaintext 經過 IP permutation 轉換

Key 經過 PC1 permutation 轉換，從 64bit 變為 56bit

```
    //Initial Permutation  64 bit -> 64 bit
    plaintext = permutation<64, 64>(plaintext, initial_perm);
    //PC1 key 64 bit -> 56 bit
    bitset<56> key56 = permutation<64, 56>(key64, PC1);
```

Plaintext 與 key 都切割為左右部分

```
    //split plaintext 64 bit -> 2 * 32 bit
    bitset<32> L = bitset<32>(((plaintext >> 32) & divider64).to_ullong());
    bitset<32> R = bitset<32>((plaintext & divider64).to_ullong());
    //split key 56 bit -> 2 * 28 bit
    bitset<28> L_key = bitset<28>(((key56 >> 28) & divider56).to_ullong());
    bitset<28> R_key = bitset<28>((key56 & divider56).to_ullong());
```

進行 16 輪的加密，每輪根據 shift table 對 Key 做左旋位元，我的做法是將左

旋 1bit 與 2bit 直接寫成 permutation table，讓程式一致性更高。

```
vector<int> shift_left1{
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 1};
vector<int> shift_left2{
3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 1, 2};

    //16 round
    for (int i = 0; i < 16; i++) {
        //shift L R key
        if (shift_table[i] == 1) {
            L_key = permutation<28, 28>(L_key, shift_left1);
            R_key = permutation<28, 28>(R_key, shift_left1);
        }
        else if (shift_table[i] == 2) {
            L_key = permutation<28, 28>(L_key, shift_left2);
            R_key = permutation<28, 28>(R_key, shift_left2);
        }
```

然後將左右部分的 key combine 起來，concat 的原理是將 bitset 轉成 string 去相連，再轉回 bitset。

```
template <size_t N1, size_t N2 >
inline bitset <N1 + N2> bitset_concat(const bitset <N1>& b1, const bitset <N2>& b2) { return bitset <N1 + N2>(b1.to_string() + b2.to_string()); }

    //combine L R key
     bitset<56> combine = bitset_concat(L_key, R_key);
```

對 combine 後的 key 做 PC2 permutation，產生這個 round 的 48bit key

```
        //PC2 key 56 bit -> 48 bit
        bitset<48> round_key = permutation<56, 48>(combine, PC2);
```

F function 輸入右半部的 plaintext 與 round key，輸出與 plaintext 左半部 XOR，最後將 plaintext 左右部分交換

```
        //F function and xor with L plaintext
        L = L ^ F(R, round_key);
        //swap L and R
        swap(L, R);
```

組合 plaintext 左右部分，然後經過 FP 就是加密完成的密文

```
    //combine L R plaintext and Final Permutation  64 bit -> 64 bit
    bitset<64> ciphertext = permutation<64, 64>(bitset_concat(L, R), final_perm);
```

輸出經過 16 進制與大寫轉換，並且設定不到 16 個字最前面要補零

```
    //output
    cout << "0x" << setw(16) << setfill('0') << hex << uppercase << ciphertext.to_ullong() << endl;
```

F function 內容:

將右半部 plaintext 從 32bit 擴充成 48bit，再與 round key XOR

```
    //expand function 32 bit -> 48 bit
    bitset<48> exR = permutation<32, 48>(R, E);
    //xor with round key
    bitset<48> B = exR ^ round_key;
```

跑過 8 個 S box，迴圈每次會計算該 6 bit 對應到 S box 中的 row col

```
    //loop through 8 S box
    for (int i = 0; i < 8; i++) {
        //base index on 6 bit input
        int base = i * 6;
        //calculate row col in S box
        int row = B[base + 0] + B[base + 5] * 2;
        int col = B[base + 1] + B[base + 2] * 2 + B[base + 3] * 4 +
B[base + 4] * 8;
```

根據 row col 從當前 S box 中取出 4bit 輸出，assign 到輸出變數

```
        //get 4 bit output
        bitset<4> out = bitset<4>(S_BOX[7 - i][row][col]);
        //assign to result
        for (int j = 0; j < 4; j++) {
            result[output_index++] = out[j];
```

輸出再經過 P permutation 轉換

```
    //P function 32 bit -> 32 bit
    result = permutation<32, 32>(result, P);
```

解密

先將會使用到的 table 全部列出來

```
// Init Permutation
int IP[] = { ... }

// Final Permutation
int FP[] = { ... }

// Key generate
int PC1[] = { ... }

int PC2[] = { ... }

int shiftBits[] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

// For f function
int E[] = { ... }

int S_BOX[8][4][16] = { ... }

int P[] = { ... }

char _hex[16] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
string bits4[16] = { "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111" };
```

將 input 做一些基本的處理，例如補 0

```
string input_process(string input) {

    input.erase(0, 2);
    string output = "";
    for (int i = 0; i < input.size(); i++) {
        for (int k = 0; k < 16; k++) {
            if (toupper(input[i]) == _hex[k]) {
                output += bits4[k];
                break;
            }
        }
    }
    int count = output.size();
    for (int i = 0; i < 64 - count; i++) {
        output = '0' + output;
    }
    return output;
}
```

生成 16 個 subkey

```
void keygeneration(bitset<64> key) {
    bitset<56> PC1_Key;
    bitset<28> C;
    bitset<28> D;
    bitset<48> PC2_Key;
    for (int i = 0; i < 56; i++) {
        PC1_Key[55 - i] = key[64 - PC1[i]];
    }
    for (int r = 0; r < 16; r++)
    {
        for (int i = 0; i < 28; i++) {
            C[i] = PC1_Key[i + 28];
            D[i] = PC1_Key[i];
        }
        C = leftShift(C, shiftBits[r]);
        D = leftShift(D, shiftBits[r]);
        for (int i = 0; i < 28; ++i) {
            PC1_Key[i] = D[i];
            PC1_Key[i+28] = C[i];
        }
        for (int i = 0; i < 48; ++i) {
            PC2_Key[47 - i] = PC1_Key[56 - PC2[i]];
        }
        subKey[r] = PC2_Key;
    }
}
```

F function 實做

```
bitset<32> f(bitset<32> R, bitset<48> key){
    bitset<48> expandR;
    for (int i = 0; i < 48; i++) {
        expandR[47-i] = R[32 - E[i]];
    }
    expandR = expandR ^ key;

    bitset<32> output;
    int count = 0;

    for (int i = 0; i < 48; i += 6) {
        int row = expandR[47 - i] * 2 + expandR[47 - i - 5];
        int col = expandR[47 - i - 1] * 8 + expandR[47 - i - 2] * 4 +
expandR[47 - i - 3] * 2 + expandR[47 - i - 4];
        bitset<4> binary(S_BOX[i / 6][row][col]);
        output[31 - count] = binary[3];
        output[31 - count - 1] = binary[2];
        output[31 - count - 2] = binary[1];
        output[31 - count - 3] = binary[0];
        count += 4;
    }
    bitset<32> temp = output;
    for (int i = 0; i < 32; i++)
        output[31 - i] = temp[32 - P[i]];
    return output;
}
```

解密主要演算法

```
// Initial permutation
for (int i = 0; i < 64; i++) {
    IP_plain[63-i] = plain[64-IP[i]];
}
// Divide IP_plain into left and right
for (int i = 0; i < 32; i++) {
    left[i] = IP_plain[i + 32];
    right[i] = IP_plain[i];
}
for (int round = 0; round < 16; round++)
{
    newLeft = right;
    right = left ^ f(right, subKey[15-round]);
    left = newLeft;
}
for (int i = 0; i < 32; i++) {
    plain[i] = left[i];
    plain[i + 32] = right[i];
}
IP_plain = plain;
for (int i = 0; i < 64; ++i) {
    plain[63 - i] = IP_plain[64 - FP[i]];
}
```

將輸出從 bitset<64>轉成 0x

```
void output(bitset<64> bits)
{
    cout << "0x";
    for (int i = 63; i >= 0; i-=4) {
        bitset<4> temp;
        temp[3] = bits[i];
        temp[2] = bits[i-1];
        temp[1] = bits[i-2];
        temp[0] = bits[i-3];
        for (int k = 0; k < 16; k++) {
            if (temp.to_string() == bits4[k]) {
                cout << _hex[k];
            }
        }
    }
}
```