

# 網安作業程式書面報告

邱彥閔 B0621101 電機通訊三

1. 題目：請寫一個程式來求  $a \bmod n$  的 inverse，即求解  $ax = 1 \pmod{n}$ 。

一、 程式說明：

(1) 介面：

剛開始跳出之介面如下圖所示，剛開始會以  $a \bmod n$  來計算是否為 inverse，接著會詢問輸入  $a$  的值與  $n$  的值，不過也會提醒  $n$  的數值會比  $a$  大，萬一輸入  $a$  比  $n$  大的值，就會顯示錯誤，就必須重新開始。

```
輸入 a (mod n) 的 inverse，即求解是否為 inverse
n 的數值必須比 a 大
輸入a=>550
輸入n=>1759
```

輸入值正確，就會進到下一步驟，利用 Extended Euclid Algorithm 運算出  $Q$  的值，進而運算出最後一個  $Q$  的值，最後的  $Q$  值如為 1 就是 inverse of  $a \pmod{n}$ ，如值不為 0 就不是 inverse of  $a \pmod{n}$ 。

```
521 輸入 a (mod n) 的 inverse，即求解是否為 inverse
    n 的數值必須比 a 大
    輸入a=>550
    輸入n=>1759
    顯示經過 Extended Euclid's Algorithm 的 Q 變化 ==>3
    顯示經過 Extended Euclid's Algorithm 的 Q 變化 ==>5
    顯示經過 Extended Euclid's Algorithm 的 Q 變化 ==>21
    顯示經過 Extended Euclid's Algorithm 的 Q 變化 ==>1
    餘數==>109
    !!提醒：最後的 Q 值為 1 時，即為 inverse!!
    最後 Q 值為 1，即判斷為 inverse
```

(2) 重點程式碼說明：

大致分成兩部分，第一部分是為控制 Extended Euclid Algorithm 運算的 Extended 的函式(如下圖一)，第二部分就是主程式，負責呼叫 Extended 函式並控制  $n$  的直必須大於  $a$  的偵錯算式(如下圖二)。

```

void Extended() {
    int Q;
    int T1, T2, T3;
    int X = n % a;
    (A1, A2, A3) = (1, 0, n);
    (B1, B2, B3) = (0, 1, a);
    while ((B3 != 0) && (B3 != 1)) {
        Q = A3 / B3;
        T1 = A1 - Q * B1, T2 = A2 - Q * B2, T3 = A3 - Q * B3;
        (A1, A2, A3) = (B1, B2, B3);
        (B1, B2, B3) = (T1, T2, T3);
        printf("顯示經過 Extended Euclid's Algorithm 的 Q 變化 ==>");
        cout << Q << endl;
    }

    if (B3 = 1) {
        printf("餘數==>");
        cout << X << endl;
        printf("\n!! 提醒：最後的 Q 值為 1 時，即為 inverse!!\n\n");

        if (X != 0) {
            printf("最後 Q 值為 1，即判斷為 inverse\n");
        }
        else
        {
            printf("最後 Q 值不為 1，即判斷為 no inverse\n");
        }
    }
}

```

(圖一)

```

int main()
{
    printf("輸入 a (mod n) 的 inverse，即求解是否為 inverse \n");
    printf("n 的數值必須比 a 大 \n");
    printf("輸入a=>");
    cin >> a;
    printf("輸入n=>");
    cin >> n;
    int A;
    A = n % a;
    if (a >= n) {
        printf("輸入錯誤，請重新輸入!!");
    }
    else
        Extended();
}

```

(圖二)

## 二、測試報告：

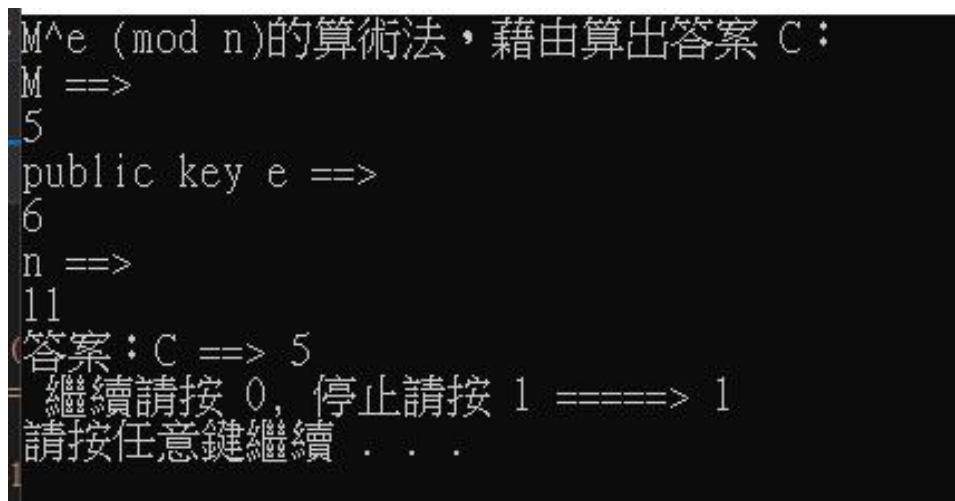
經過測試運算值都蠻正常的，可以有效的分辨出來 inverse 或是 no inverse，不過設置的 Extended()中的函式運算，A1、A2、A3、B1、B2、B3 經過 while 迴圈運算值在最後顯示，都保持著原來的值，並沒有隨著迴圈運算改變。

2. 題目：請寫一個程式來實現 RSA 的快速指數運算， $C = M^e \bmod n$  其中需把  $e$  化為二進位數表示，並依此二進位的各個位元值來實現此快速運算。

### 一、 程式說明：

#### (1)介面：

此程式是用來方便計算 RSA 的快速指數運算，主要就是你輸入 M 值與指數 e 值接著輸入 n 值，再用 mod 運算法，最後得出 C 的值，如下圖。



```
M^e (mod n)的算術法，藉由算出答案 C：
M ==>
5
public key e ==>
6
n ==>
11
(答案：C ==> 5
繼續請按 0，停止請按 1 =====> 1
請按任意鍵繼續 . . .
```

#### (2)重點程式碼說明：

程式主要分成 3 部分來寫：(如下面圖)

- 函式 D\_to\_B()：輸入得 e 值，利用短除法計算，轉成二進位來方便計算。
- 函式 cal(int A, int Bits[], int n)：是用來計算  $C = M^e \bmod n$  的運算式。
- 函式 main：為主程式，主要顯示介面與呼叫前面函式。

```

void D_to_B() {
    for (int i = 15; i >= 0; i--) {
        Bit[i] = E % 2;
        E = E >> 1;
    }
}

```

- 函式 D\_to\_B()

```

int cal(int A, int Bits[], int n) {
    j = 0;
    D = 1;
    for (int i = 0; i <= 15; i++) {
        j = j << 1;
        D = (D * D) % n;
        if (Bits[i] == 1) {
            j++;
            D = (D * A) % n;
        }
    }
    return D;
}

```

- 函式 cal(int A, int Bits[], int n)(如上圖)

```

int main()
{
    do {
        printf("M^e (mod n)的算術法，藉由算出答案 C：\n");
        cout << "M ==>" << endl;
        cin >> A;
        cout << "public key e ==> " << endl;
        cin >> E;
        cout << "n ==> " << endl;
        cin >> n;

        Bit = new int[16];
        for (int i = 0; i < 16; i++)
            Bit[i] = 0;

        D_to_B();

        D = cal(A, Bit, n);
        cout << "答案：C ==> " << D << endl;

        delete[] Bit;

        cout << "繼續請按 0，停止請按 1 ==> ";
        cin >> ans;

    } while (ans == 0);

    system("pause");
}

```

- Main() 主程式 (如左圖)

## 二、 測試報告：

經過測試就是要有  $C = M^e \bmod n$  運算式的基本概念，因我沒有設置防呆模式提醒  $n$  的值必須比  $M^e$  的值還要大，不然出現的  $C$  就會恆為 0，而其他大致上都非常的良好，就像一般的計算機一樣輕鬆簡單。

## 3. 前兩題的總心得：

就前面兩題而言，比起上一份 DES 作業簡單的非常多，大部分就像是計算機上的計算模式，但要用 C++ 寫出其中的運算模式，確實蠻考驗我們的程式能力，不過在網路上其實蠻多相關程式，大致看看然後再改一改，就可以寫得出來，最重要的是講義也有提供 C++ 的程式，所以可以較快寫出運算方式。總覺得大一學完 C++ 總覺得現在不適大部分的人都在使用 JAVA 或是 python 之類的來寫程式，而我徹底的錯了！經過寫這網路安全的作業，發現，其實最根本、最簡單的程式 C++ 才是最容易的，也發現，程式能力必須不停的練習，就像學英文一樣，看似可以記一輩子，殊不知是一個很容易生疏的一門學問。