

杭州电子科技大学

实验报告

课程名称：密码学课程设计 姓名：苏展 学号： 18271329

实验地点：科技馆 620

实验时间：2020-6-2

一、实验名称：RSA 密码实验

二、实验要求：

- 1、了解公钥密码的起源与涵义。
- 2、掌握 RSA 密码的加解密原理。
- 3、用 Visual C++实现 RSA 密码并输出结果。

三、实验内容：

1、公钥密码体制的概念由 Diffie 和 Hellman 于 1976 年提出，用于解决对称密码体制中密钥分配的问题。在公钥密码体制中，密钥被分为公钥与私钥，公钥是公开的，用于加密；私钥是保密的，用于解密。经过四十余年的研究发展，RSA 密码、ElGamal 密码、椭圆曲线密码等等公钥密码体制在商业、军事上都已经得到了广泛的应用。

2、RSA 密码由 R. Rivest、A. Shamir 和 L. Adleman 于 1977 年提出，是最著名的公钥密码，能够抵抗到目前为止已知的绝大多数密码攻击，已被 ISO 推荐为公钥数据加密标准。RSA 密码的安全性基于这样一个事实：将两个大素数 p, q 相乘十分容易，但对其乘积 $N=pq$ 作因子分解却极其困难。在本实验中，假设 p, q 均为 len 比特素数，具体描述如下：

系统参数

大素数 p, q ： len 比特素数（即 $2^{len-1} \leq p, q < 2^{len}$ 且 p, q 为素数）

乘积 N ： $N = p * q$ ，约为 $2 * len$ 比特

N 的欧拉函数值 $\varphi(N)$ ： $\varphi(N) = (p-1)(q-1)$

公钥

(N, e) ：其中 e 满足 $1 < e < \varphi(N)$ 且 $\gcd(e, \varphi(N)) = 1$

私钥

(N,d): 其中 d 满足 $1 < d < \varphi(N)$ 且 $ed \equiv 1 \pmod{\varphi(N)}$, 即 $d = e^{-1} \pmod{\varphi(N)}$

明文

正整数 m: 满足 $1 < m < N$

加密

密文 $c = m^e \pmod{N}$

解密

密文 $m = c^d \pmod{N}$

思考题:

为什么 RSA 密码能正确解密? 请在此写出详细的证明过程。

$pq = N$ p,q 为两个质数

记[N,e], [N,d]分别为算法中的公钥和私钥, 根据算法性质知 $ed = 1 \pmod{(p-1)(q-1)}$ 这里的等号为模等, 下同

则 $ed \equiv 1 \pmod{p-1}$ $ed \equiv 1 \pmod{q-1}$

记 n 为明文, 则 $n^e \equiv c \pmod{N}$, c 为密文

设 $ed = a(p-1) + 1$ $a > 1$

则解密过程 $c^d = n^{ed} = n^{(ed)} = n^{[a(p-1)+1]} = n * n^{[a(p-1)]}$

因 p 为质数, 根据欧拉定理有 $n^{(p-1)} \equiv 1 \pmod{p}$

故 $n^{[a(p-1)]} \equiv 1 \pmod{p}$

$n * n^{[a(p-1)]} \equiv n \pmod{p}$, 即 $c^d \equiv n \pmod{p}$

同理可证 $c^d \equiv n \pmod{q}$

因为 p,q 都为质数, 所以 $c^d \equiv n \pmod{pq}$, 即 $c^d \equiv n \pmod{N}$

3、使用 Visual C++编写程序, 实现 RSA 密码并输出结果。

编程的关键是实现大整数运算, 包括模加、模乘、模幂等等运算, 所以考虑使用无符号字符数组表示大整数, 即使用 256 进制表示大整数, 用结构体实现。

- 1) 新建一个空项目，取名 `rsa`。
- 2) 添加 `cpp` 文件，取名为 `rsa.cpp`。
- 3) 在 `rsa.cpp` 文件中先写入

```
#include<time.h>
#include<iostream>
using namespace std;

#define SIZE 33          //SIZE 是数组长度, 能表示的最大整数的 bit 数是 8*SIZE,
                        //SIZE 可自由选取

typedef struct Bigint    //用于表示大整数的无符号字符数组
{
    unsigned char num[SIZE];
}Bigint;

typedef struct Bigint2   //大整数乘法可能需要的数组长度是原数组的两倍
{
    unsigned char num[2*SIZE];
}Bigint2;
```

- 4) 可能需要的函数

```
//初始化
Bigint Init(unsigned char a[], int length);
//复制
void Copy(Bigint &a, Bigint b);
//打印输出
void Print(Bigint a);
//计算数组长度
int Length(Bigint a);
int Length(Bigint2 a);
//比较大小
int Compare(Bigint a, Bigint b);    //a>b, a=b, a<b 分别输出 1, 0, -1
int Compare(Bigint2 a, Bigint2 b); //a>b, a=b, a<b 分别输出 1, 0, -1
//左移 loop 个字节
Bigint MoveLeft(Bigint a, int loop);
Bigint2 MoveLeft(Bigint2 a, int loop);
//右移一个比特
void BitMoveRight(Bigint &a);
//扩充数组
Bigint2 Extend(Bigint a);
//截断数组
Bigint Narrow(Bigint2 a);
//生成随机数
```

```

Bigint BigRand(Bigint n); //生成 1 到 n 之间的随机数
Bigint BigRandOdd(int bytes); //生成 bit 数是 8*bytes 的随机奇数

//基本运算
Bigint Add(Bigint a, Bigint b); // 加法: 输入 a,b, 返回 a + b
Bigint Sub(Bigint a, Bigint b); // 减法: 输入 a>b, 返回 a - b
Bigint2 Sub(Bigint2 a, Bigint2 b); // 减法: 输入 a>b, 返回 a - b
Bigint2 Mul(Bigint a, Bigint b); // 乘法: 输入 a,b, 返回 a * b
Bigint Div(Bigint a, Bigint b); // 除法: 输入 a,b, 返回 a / b
Bigint Mod(Bigint a, Bigint b); // 求余: 输入 a,b, 返回 a % b
Bigint2 Mod(Bigint2 a, Bigint2 b); // 求余: 输入 a,b, 返回 a % b

Bigint AddMod(Bigint a, Bigint b, Bigint n); // 模加: 计算 a + b mod n
Bigint SubMod(Bigint a, Bigint b, Bigint n); // 模减: 计算 a - b mod n(要求 a>b)
Bigint Sub2Mod(Bigint a, Bigint b, Bigint n); // 模减: 计算 a - b mod n
Bigint MulMod(Bigint a, Bigint b, Bigint n); // 模乘: 计算 a * b mod n
Bigint PowMod(Bigint a, Bigint b, Bigint n); // 模幂: 计算 a ^ b mod n

//MillerRabin 素性检测
bool MillerRabinKnl(Bigint &n); //单次素性检测, 通过返回 1, 否则返回 0
bool MillerRabin(Bigint &n, long loop); //loop 次素性检测, 通过返回 1, 否则返回 0

//生成 bit 数是 8*bytes 的素数
Bigint GenPrime(int bytes);

//生成私钥
bool Inverse(Bigint e, Bigint phiN, Bigint &d); //计算  $d = e^{-1} \bmod \varphi(N)$ 
//加密
Bigint Encrypt(Bigint m, Bigint e, Bigint N); //计算  $c = m^e \bmod N$ 
//解密
Bigint Decrypt(Bigint c, Bigint d, Bigint N); //计算  $m = c^d \bmod N$ 

```

5) 编写主函数, 主要步骤有

生成并输出大素数 p、q 及其乘积 $N=pq$;

输出公钥 e;

输出私钥 d;

随机生成并输出明文;

加密并输出密文;

解密并输出解密后的明文。

如下所示:

```
D:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\vcprojects\RSA\Debug\RSA.exe
生成大素数:
p=279405879024828246713180947787608738547
q=240845450798658244373270493430610093081

系统参数是:
N=67293634889530129061886816553891988938806440043090984832964793595132662693307

公钥是:
e=65537

私钥是:
d=52506634629461413853375104922407506391018852648757904621973708048622442304513

明文是:
m=31628633767000880302747882090455429665585021074826914259669911205492665142248

加密得到的密文是:
c=60340565441035184476991802312618884264757871618352537790312329833011683150744

解密得到的明文是:
31628633767000880302747882090455429665585021074826914259669911205492665142248

请按任意键继续. . .
```

提示点:

- 1) 无符号字符的范围是 0~255，所以大整数是用 256 进制表示的。
假设数组 a 里元素分别为 a.num[0], a.num[1],……,a.num[SIZE-1]
则 a 表示的数为
$$a.num[0] * 1 + a.num[1] * 256 + a.num[SIZE-1] * 256^{SIZE-1}$$
- 2) 为了加密速度快，本次实验的公钥 e 可选定 65537(实际当中也很常用)，
其 256 进制表示是{1,0,1}。
- 3) 实验中涉及到的数都是非负数，所以减法运算需要先判断大小。
- 4) 大整数乘法已用 2 倍长数组保证不溢出，而大整数加法可能会溢出，所以
SIZE 定义时要比 N 的实际大小多一点。比如 N 是 80bit(10 字节)，那么
SIZE 应该定义成 11。
- 4) 测试小整数示例时可用 Bigint a = {0}, Bigint a = {1}, Bigint a={15}类似的
初始化命令。
- 5) 部分函数参考代码在”RSA 密码_部分参考代码.doc”中有提供

四、实验报告:

本次实验要求实现 p,q 均为 128bit(16 字节)的情形，SIZE 可定义为 33。

1) RSA 密码程序代码如下:

```

#!/usr/bin/python
# -*- coding: UTF-8 -*-

from PyQt5 import QtCore, QtGui, QtWidgets
import sys
from Ui_RSA import Ui_MainWindow
import random
import secrets

class Min(QtWidgets.QMainWindow,Ui_MainWindow):
    def __init__(self,parent=None): #ui 部分
        super().__init__()
        self.setupUi(self)
        self.sj.clicked.connect(self.func1)
        self.jm.clicked.connect(self.func2)
        self.jm2.clicked.connect(self.func3)
        self.m1=[]
        self.c=[]

    def func1(self):
        """
        生成随机密钥
        """
        self.nump=self.GenPrime(16)
        self.numq=self.GenPrime(16)
        self.numN=self.nump*self.numq
        self.numPhiN=(self.nump-1)*(self.numq-1)
        self.nume=self.GenE(self.numPhiN)
        self.numd=self.Inverse(self.nume,self.numPhiN)
        #显示
        self.P.setPlainText(str(self.nump))
        self.Q.setPlainText(str(self.numq))
        self.N.setPlainText(str(self.numN))
        self.e.setPlainText(str(self.nume))
        self.d.setPlainText(str(self.numd))

    def func2(self):
        """
        加密
        """
        self.m=list(self.mw.toPlainText())
        for ms in self.m:

```

```

        self.c.append(str(self.fast_power(ord(ms) , self.nume, self
.numN)))
        self.mw2.setPlainText(''.join(self.c))

def func3(self):
    """
    解密
    """
    for cs in self.c:
        cs=int(cs)
        self.m1.append(chr(self.fast_power(cs , self.numd, self.num
N)))

    self.jmw.setPlainText(''.join(self.m1))

def Inverse(self, e, N):

    r1 = e
    r2 = N
    s1 = 1
    s2 = 0
    while (1):
        if (r1 == 0):
            return 0
        if (r1 == 1):
            d = s1
            return d
        q = r1 // r2
        s = self.Sub2Mod(s1, (q* s2% N), N)
        r = r1-(q* r2)
        r1 = r2
        s1 = s2
        s2 = s
        r2 = r

def Sub2Mod(self, a, b, n):

    while (a - b < 0):
        a += n
    return (a - b)

def GenE(self, PhiN):
    e = random.randint(0,PhiN)
    g = self.GCD(PhiN,e)
    while g != 1:

```

```

        e = random.randint(0, PhiN)
        g = self.GCD(PhiN, e)
    return e

def GCD(self, a, b):
    if a % b == 0:
        return b
    else :
        return self.GCD(b, a % b)

def fast_power(self, base, power, n):
    """
    快速模幂运算
    """
    result = 1
    tmp = base
    while power > 0:
        if power & 1 == 1:
            result = (result * tmp) % n
            tmp = (tmp * tmp) % n
            power = power >> 1
    return result

def Miller_Rabin(self, n, iter_num):
    """
    Miller_Rabin 素性检测
    """
    # 2 is prime
    if n == 2:
        return True
    # if n is even or less than 2, then n is not a prime
    a = n & 1
    if n & 1 == 0 or n < 2:
        return False
    #  $n-1 = (2^s)m$ 
    m, s = n - 1, 0
    while m & 1 == 0:
        m = m >> 1
        s += 1
    # M-R test
    for _ in range(iter_num):
        b = self.fast_power(random.randint(2, n-1), m, n)
        if b == 1 or b == n-1:
            continue

```



```

        for __ in range(s-1):
            b = self.fast_power(b, 2, n)
            if b == n-1:
                break
        else:
            return False
    return True

def RandOdd(self, byte):
    """
    生成若干 bytes 的随机奇数
    """
    byte*=8
    num=secrets.randbits(byte)
    if num&1 == 0:
        num+=1
    return num

def GenPrime(self, byte):
    """
    生成若干 bytes 的随机素数
    """
    res=self.RandOdd(byte)
    while self.Miller_Rabin(res,20) == False:
        res=self.RandOdd(byte)
    return res

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ui=Min()
    ui.show()
    sys.exit(app.exec_())

```

2) RSA 密码输出界面截屏如下：

RSA

P

281424712356773602885
196004366073080331

Q

614311060453219244191
81415995827182863

N

1728823134856317855996594511208845442288351441077159448600
7652125000425567653

公钥e

2642569650797215610160405554159183124720023206932596355242
413970277096077219

私钥d

1251235688235199134210393789570004959486190296497128631749
6017218242662126479

随机生成密钥

加密

解密

明文

RSA真是太厉害了太厉害了太厉害啦!!

密文

46242091318229870425155184885942281209199803360188860991679
39881387269737759905111916514112741956480571987669495126455
30152647605825149214622202588561361334590506866223297616881
04442622987387588204992682092850911880880443888461619158974
94431806171011231322695404666141892884417769419939189561198
85283011878261504670977236105931263869029410813292843250532
40989827851691045435264473117855050244990941781012338729040
32729308198913779827378710390985090621859073543617527461180
93668546359592615085882213852988115502390198889627973899890
56448963752965749766343139222772315645130862583170707534787
28073917389305951176867010701937903377639122975685162406631
73522326114670724175421157703303677117855050244990941781012
33872904032729308198913779827378710390985090621859073543617
52746118093668546359592615085882213852988115502390198889627
97389989056448963752965749766343139222772315645130862583170
70753478728073917389305951176867010701937903377639122975685
16240663173522326114670724175421157703303677117855050244990
94178101233872904032729308198913779827378710390985090621859
07354361752746118093668546359592615085882213852988115502390
19888962797389989056448963752965749766343139222772315645130
8625831707075347872807391738930595117038704978026538057222

解密文

RSA真是太厉害了太厉害了太厉害啦!!

附加题：

实现 RSA 密码的 MFC 图形化界面