实验项目名称： RSA密码算法 实验学时： 6

同组学生姓名： 无 实验地点：科技楼4-1204

实验日期： 2023.12.6 实验成绩：

批改教师： 黄丹丹 批改时间： 2023.12.15

实验6 RSA密码算法

一、基于OBE模式的实验目的和要求

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、实现RSA密码并输出结果。

二、实验仪器和设备

Visual C/C++

三、实验过程

1、RSA密码由R. Rivest、A. Shamir和L. Adleman于1978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](http://baike.baidu.com/view/1519129.htm" \t "http://baike.baidu.com/_blank)。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 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：

**公钥**

(N, e)：其中e满足1<e<且gcd (e, ) = 1

**私钥**

d：其中d满足1<d<且，即

**明文：**选取正整数m，满足1<m<N-1

**加密**

密文 

**解密**

明文 

1. 编写程序，实现RSA密码并输出结果。

编程的关键是实现整数运算，包括模加、模乘、模幂等等运算。

3、可能需要的函数

#include<time.h>

#include<iostream>

//复制

void Copy(int &a, int b);

//打印输出

void Print(int a);

//生成随机数

int Rand(int n); //生成1到n之间的随机数

int Rand(int bytes); //生成bit数是8\*Bytes的随机数

//基本运算

int AddMod(int a, int b, int n); // 模加: 计算 a + b mod n

int SubMod(int a, int b, int n); // 模减: 计算 a - b mod n(要求a>b)

int MulMod(int a, int b, int n); // 模乘: 计算 a \* b mod n

int Inv(int a, int n); // 求逆元: 输入a, n, 返回 a^(-1)mod n

int PowMod(int a, int b, int n); // 模幂：计算 a ^ b mod n

//MillerRabin素性检测

bool MillerRabinKnl(int &n); //单次素性判定

bool MillerRabin(int &n, int loop); //素性检测，通过返回1，否则返回0

//生成bit数是8\*Bytes的随机素数

int GenPrime(int bytes);

//生成私钥（求逆元）

int Inverse(int e, int N, int d); //计算

//加密

int Encrypt(int m, int e, int N); //计算c = m^e mod N

//解密

int Decrypt(int c, int d, int N); //计算m = c^d mod N

4、编写主函数，主要步骤有

生成并输出素数p、q及其乘积N=pq；

输出公钥e，计算私钥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。

提示点：

1. 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。
2. 本次实验要求实现p, q约为16bit(2字节)素数的情形，测试可选加密DES算法密钥，例如1334577 99BBCDF F1（8个字节），也可自选。

参考：例如取 p = 11113， q = 33113，N = 367984769，log2N = 29，明文可按照28比特一组进行加密。

5、程序代码：

#include **<**stdio**.**h**>**#include **<**stdlib**.**h**>**#include **<**time**.**h**>  
  
long** **long** **int** gcd**(long** **long** **int** a**,long** **long** **int** b**)** **{**    **while** **(**b **!=** 0**)** **{**        **long** **long** **int** t **=** b**;**        b **=** a **%** b**;**        a **=** t**;**    **}**    **return** a**;  
}  
  
long** **long** **int** extended\_gcd**(long** **long** **int** a**,long** **long** **int** b**,long** **long** **int** **\***x**,long** **long** **int** **\***y**)** **{**    **if** **(**b **==** 0**)** **{**        **\***x **=** 1**;**        **\***y **=** 0**;**        **return** a**;**    **}**    **long** **long** **int** x1**,** y1**;**    **long** **long** **int** gcd **=** extended\_gcd**(**b**,** a **%** b**,** **&**x1**,** **&**y1**);**    **\***x **=** y1**;**    **\***y **=** x1 **-** **(**a **/** b**)** **\*** y1**;**    **return** gcd**;  
}  
  
long** **long** **int** inverse**(long** **long** **int** a**,long** **long** **int** n**)** **{**    **long** **long** **int** x**,** y**;**    **long** **long** **int** gcd **=** extended\_gcd**(**a**,** n**,** **&**x**,** **&**y**);**    **if** **(**gcd **!=** 1**)** **{**        **return** **-**1**;** *// 没有逆元*    **}** **else** **{**        **return** **(**x **%** n **+** n**)** **%** n**;**    **}  
}  
  
long** **long** **int** mod\_pow**(long** **long** **int** a**,long** **long** **int** b**,long** **long** **int** n**)** **{**    **long** **long** **int** r **=** 1**;**    **while** **(**b **>** 0**)** **{**        **if** **(**b **&** 1**)** **{**            r **=** **((long** **long)**r **\*** a**)** **%** n**;**        **}**        a **=** **((long** **long)**a **\*** a**)** **%** n**;**        b **>>=** 1**;**    **}**    **return** r**;  
}  
  
long** **long** **int** is\_prime**(long** **long** **int** n**)** **{**    **if** **(**n **<=** 1**)** **{**        **return** 0**;**    **}**    **if** **(**n **<=** 3**)** **{**        **return** 1**;**    **}**    **if** **(**n **%** 2 **==** 0 **||** n **%** 3 **==** 0**)** **{**        **return** 0**;**    **}**    **for** **(int** i **=** 5**;** i **\*** i **<=** n**;** i **+=** 6**)** **{**        **if** **(**n **%** i **==** 0 **||** n **%** **(**i **+** 2**)** **==** 0**)** **{**            **return** 0**;**        **}**    **}**    **return** 1**;  
}  
  
long** **long** **int** gen\_prime**(int** bits**)** **{**    **long** **long** **int** p**,** count **=** 0**;**    **do** **{**        p **=** rand**()** **%** **(**1 **<<** bits**)** **+** **(**1 **<<** **(**bits **-** 1**));**        count**++;**    **}** **while** **(!**is\_prime**(**p**)** **&&** count **<** 1000**);**    **return** p**;  
}  
  
void** gen\_key\_pair**(int** bits**,long** **long** **int** **\***e**,long** **long** **int** **\***d**,long** **long** **int** **\***N**)** **{**    srand**(**time**(**NULL**));**    printf**("生成p，q...**\n**");**    *// 生成两个大素数p和q*    **long** **long** **int** p **=** gen\_prime**(**bits **/** 2**);**    **long** **long** **int** q **=** gen\_prime**(**bits **/** 2**);**    printf**("p=%lld,q=%lld**\n**",**p**,**q**);**    *//欧拉函数值*    **long** **long** **int** phiN **=** **(**p **-** 1**)** **\*** **(**q **-** 1**);**      
    *// 选择公钥e，满足1 < e < phi(N)且e与phi(N)互质*    **\***e **=** 65537**;**    printf**("计算公钥（N，e）...**\n**");**    *// 计算模数N*    **\***N **=** p**\***q**;**      
    *// 计算私钥d*    **\***d **=** inverse**(\***e**,** phiN**);  
}  
  
long** **long** **int** encrypt**(long** **long** **int** m**,long** **long** **int** e**,long** **long** **int** N**)** **{**    **return** mod\_pow**(**m**,** e**,** N**);  
}  
  
long** **long** **int** decrypt**(long** **long** **int** c**,long** **long** **int** d**,long** **long** **int** N**)** **{**    **return** mod\_pow**(**c**,** d**,** N**);  
}  
  
int** main**()** **{**    **int** bits **=** 32**;**    **long** **long** **int** e**,** d**,** N**;**      
    gen\_key\_pair**(**bits**,** **&**e**,** **&**d**,** **&**N**);**    *//printf("%d\n",count);*    printf**("公钥:（N，e）=（%lld, %lld）**\n**",** N**,** e**);**      
    printf**("计算私钥d...**\n**");**    printf**("私钥: d = %lld**\n**",** d**);**      
    **long** **long** **int** m **=** rand**()** **%** N**;**    **long** **long** **int** c **=** encrypt**(**m**,** e**,** N**);**    **long** **long** **int** decrypted **=** decrypt**(**c**,** d**,** N**);**    printf**("明文m = %lld, 密文c = %lld**\n**解密后的明文(16进制) = %llx**\n**解密后的明文(十进制) = %lld**\n**",** m**,** c**,** decrypted**,** decrypted**);**      
    *//printf("明文m = %lld, 密文c = %lld\n解密后的明文 = %lld\n", m, c, decrypted);*      
    **return** 0**;  
}**四、实验结果与分析

测试结果截图与分析如下：

文本

描述已自动生成

文本

描述已自动生成

先随机生成素数p，q，再利用素数p，q计算公钥私钥，然后再生成明文并对明文进行加密，解密并且输出结果。

并且对代码进行修改，使之能按照16进制输出数据。

五、基于OBE模式的学生自我评价与体会

（该实验对自身分析、设计、思辨、创新等个人综合能力与素质的影响）

本次实验是通过C语言实现rsa算法，rsa算法需要一对公私钥来实现数据的加解密，生成密钥时需要生成大素数，并且素数需要足够大才能满足安全性要求。在这里有一个小坑，就是C语言的int范围不能满足运算过程中的需求，会出现两个正整数相乘的结果出现负数，这就是结果超出int范围导致的，甚至连long int的范围都不够用，只有long long能满足需求。这次实验需要用到前面好几次实验的代码，但是不能直接拿来用，需要进行修改，这样更能增强我的综合能力。