**RSA公钥加密算法实验实验报告**

计21班

2012011394 周界

2012011400 杨俊

2012011401 张梦豪

2012011385 安高乐

2012011403 张光耀

2012011387 石道明

1. **使用说明**
2. **编译运行**

将myRSA.cpp用C++编译器编译，得到可执行程序myRSA.exe，运行可执行程序myRSA.exe。得到如下的选择界面：

1. **使用**

在编译运行后，需要输入进行选择操作，有以下几种选项：R、C、D、T、Q

**R - 初始化设置：**

产生RSA所需要的大素数p,q,以及生成他们的积n，另外，随机一个与φ(n)互质并小于φ(n)的一个随机数d，并求d模φ(n)的逆e。

**C - 加密文件：**

用R得到加密所用的参数之后，即可使用它们对文件进行加密，得到密文。

**D - 解密文件:**

使用相同的参数对文件进行解密，得到明文。

**T - 特殊测试:**

为方便测试，在这里，我们对一个数进行了加密解密的测试，可以直观地看到过程。

**Q - 退出:**

按此键结束加密解密，退出系统。

1. **实现**
2. **数据表示**

在此次试验中，需要素数为2048位，一般的整型都无法完成要求的2048位（2048位已经远远超出了long long的表示范围），所以定义了如下的数据结构来实现：

struct BigNumber{

int len; // length

char num[LEN]; // number

bool neg; // negative or positive

};

其中，len表示长度，num表示每一位的值，而neg表示是否为负。

1. **关于所定义的数的基本操作**
2. **Long和定义数据结构的相互转化：**

num2Long(BigNumber\* p)：

在能表示的前提下，将P转化成long long类型并返回。

long2Num(long long k, BigNumber\* x)：

将K转化成BigNumber型并且赋值给x。

1. **初始化数**

clearNumber(BigNumber\* p)：将p赋值为0，即将len赋值为0，将num的元素都赋值为0，将neg赋值为0。

1. **显示数**

printNumber(BigNumber\* p)：

将数以二进制的形式打印出来。

1. **复制一个数到另外一个数**

copyNumber(BigNumber\* x, BigNumber\* y)：

将数x赋值给y。

1. **加法**

addNumber(BigNumber\* x, BigNumber\* y, BigNumber\* z)：

将x和y的和赋值给z。

1. **左移**

leftShiftNumber(BigNumber\* x)：

将x左移一位。

1. **右移**

rightShiftNumber(BigNumber\* x)：

将x右移一位。

1. **乘法**

mulNumber(BigNumber\* x, BigNumber\* y, BigNumber\* z)：

将x和y的乘积赋值给z。

1. **减一**

decNumber(BigNumber\* x, BigNumber\* y)：  
 将x减一的值赋给y。

1. **比较**

cmpNumber(BigNumber\* x, BigNumber\* y)：

比较两个数x和y：x大于y时赋值1；x等于y时赋值为0；x小于y时赋值为-1。

checkValue(BigNumber\* x, long t)：

比较两种格式的数x和t的值，相等返回true，不相等返回false。

1. **减法**

subNumber(BigNumber\* x, BigNumber\* y, BigNumber\* z)：

将x减y的值赋给z。

1. **模**

modNumber(BigNumber\* x, BigNumber\* y, BigNumber\* z)：

将x模y得到的值赋值给z。

1. **除法**

divNumber(BigNumber\* x, BigNumber\* y, BigNumber\* z)：

将x除以y的值赋值给z。

1. **特殊操作的实现**
2. **生成素数：**

为了保证生成的素数的随机性，我们仅仅限制了素数的位数（最大值）。算法的效率取决于判断素数的算法。

我们用Miller-Rabin检验来检验一个数是否为素数。算法的大概过程如下所示：

首先，将n-1表示成

然后，对i从1到t做循环做以下操作：

* 选择一个随机整数a（2 ≤ *a* ≤ *n*−2）
* 计算*y* ← mod *n*
* 如果*y*≠1且*y*≠ *n*−1循环做下面的操作，否则转结束：
  + 1. *j* ← 1
    2. 当*j* ≤ *s*−1并且*y* ≠ *n*−1做操作3，否则跳到步骤4
    3. 计算*y* ← *y*2 bmod *n*，如果*y* = 1返回“合数”，否则*j* ← *j* + 1
    4. 如果*y* ≠ *n*−1则返回“合数”

结束：返回“素数”

Miller-Rabin检验是一个概率算法，素数判断的成功率很大程度上取决于选取a测试的次数n，并且错误率以-n为指数衰减，可以认为当n取相当大（大于50），此时算法的正确率接近于1。

1. **模幂运算：**

模幂运算可以利用二进制表示的优势，从最低位开始，每次循环都要乘以base值，得到指数次数幂的模，而只有这一位的值为1时才会加到最后的结果中，并且取模。

1. **实验思考和收获**
2. **更加了解了公钥密码体制**

如果要说密码学中革命性的突破的话，公钥密码体制的出现无疑是最重要的。公钥密码体制依赖于复杂的数学问题，也就是大数因式分解。在每一次RSA加密中，都需要先随机地产生一对质数，以及另外一个特定的质数。产生对应的公钥和私钥。并且公开公钥而留下私钥，由于需要解密的时候的速度要快，所以一般需要私钥比较短而公钥比较长。另外，为了避免暴力破解，选取的大素数p和q也是非常的大，2048位为现行比较安全的长度，而一定不要用相同的秘钥对去加密不同的文件，这样会留下漏洞，更容易被破解。

1. **了解了RSA密码体制的优势和不足**

在实现RSA的过程中，由于是二进制实现的数，在软件中可能不是特别好实现，但是由于许多加密算法都是用硬件实现的，而像移位、模幂运算、乘除法、Miller-Rabin检验等，用二进制实现能省不少时间，从这方面说，RSA算法的优势在于它的简单和易实现。从另外一方面说，RSA中最费时间的地方便是产生素数和判断，利用Euler定理和Fermat小定理作为两个有力的判据，使得Miller-Rabin检验能以很快的速度以可以忍受的错误率下得到可靠的判断。然而，为了得到很低的错误率，不得不枚举很多的a来进行判断，而为了保证所用的数的随机性，每次都是先随机，然后再判断是否是质数，这无疑加大了运算量。所以在实际情况中，往往是这两者的折衷，用各种方法使得素数能在很短的时间内得到而近似随机。另外，在所用的位数方面，由于过短的素数使得安全性没有保证，所以2048位的素数成为了新的要求，但是相比于同等安全性的ECC等，它的计算成本太高，所以RSA逐渐被ECC所代替。

1. **关于设计一个好的公钥密码的思考**

首先，需要选择一个合适的数学难题，并且能够适合用计算机进行计算。一般来说，计算机更倾向于计算位数较少的带模运算，而对需要特殊算法实现的如RSA中的素数判定其实效果并不是特别的好。除了加解密时的运算速度的要求，还需要考虑的是尽可能解密的时候要快一些，因为从用户体验出发，解密时更需要得到及时的响应。在存储的角度，一般来说，在公钥体制里，可以用空间来换取时间，而许多优化也是从这方面出发的。从功能上出发，一般是把公钥密码算法和加密算法一起使用的，加密算法提供加密功能，而公钥密码算法提供数字签名。如果能够找到一种陷门单向函数，并且能够追赶上普通加密算法的速度，那末相信密码学又将上升到一个新的台阶。