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

一、 使用说明

1. 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：

2. 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。

二、 实现

1. 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。

2. 关于所定义的数的基本操作

1) 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。

2) 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。

3) 显示数

printNumber(BigNumber* p)：

将数以二进制的形式打印出来。

4) 复制一个数到另外一个数

copyNumber(BigNumber* x, BigNumber* y)：

将数 x 赋值给 y。

5) 加法

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

将 x 和 y 的和赋值给 z。

6) 左移

leftShiftNumber(BigNumber* x)：

将 x 左移一位。

7) 右移

rightShiftNumber(BigNumber* x)：

将 x 右移一位。

8) 乘法

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

将 x 和 y 的乘积赋值给 z。

9) 减一

decNumber(BigNumber* x, BigNumber* y)：

将 x 减一的值赋给 y。

10) 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。

11) 减法

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

将 x 减 y 的值赋给 z。

12) 模

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

将 x 模 y 得到的值赋值给 z。

13) 除法

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

将 x 除以 y 的值赋值给 z。

3. 特殊操作的实现

1) 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2^sr

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（$2 \le a \le n-2$）
- 计算 $y \leftarrow a^r \mod n$
- 如果 $y \neq 1$ 且 $y \neq n-1$ 循环做下面的操作，否则转结束：

1. $j \leftarrow 1$
2. 当 $j \le s-1$ 并且 $y \neq n-1$ 做操作 3，否则跳到步骤 4
3. 计算 $y \leftarrow y2 \mod n$，如果 $y = 1$ 返回“合数”，否则 $j \leftarrow j + 1$
4. 如果 $y \neq n-1$ 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。

2) 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。

三、 实验思考和收获

1. 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。

2. 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。

3. 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。