哈工大计算机科学与技术学院

《[RSA加密实验](http://219.217.226.44/pages/student/javascript:void(0))》

实验报告

计算机科学与技术学院

计算机系网络教研室制

|  |  |
| --- | --- |
| 课程名称： | 信息安全概论 |
| 实验名称： | [RSA加密实验](http://219.217.226.44/pages/student/javascript:void(0)) |
| 指导教师： | 韩琦 |
| 学生姓名： | 田钧锐1152010110 |
| 组 号： | 12 |
| 实验日期： | 2018/3/8 |
| 实验地点： |  |
| 实验成绩： |  |

实验报告撰写要求

实验操作是教学过程中理论联系实际的重要环节，而实验报告的撰写又是知识系统化的吸收和升华过程，因此，实验报告应该体现完整性、规范性、正确性、有效性。现将实验报告撰写的有关内容说明如下：

1、 实验报告模板为电子版。

2、 下载统一的实验报告模板，学生自行完成撰写和打印。报告的首页包含本次实验的一般信息：

*  组 号：例如：2-5 表示第二班第5组。
*  实验日期：例如：05-10-06 表示本次实验日期。(年-月-日)……
*  实验编号：例如：No.1 表示第一个实验。
*  实验时间：例如：2学时 表示本次实验所用的时间。

实验报告正文部分，从六个方面（目的、内容、步骤等）反映本次实验的要点、要求以及完成过程等情况。模板已为实验报告正文设定统一格式，学生只需在相应项内填充即可。续页不再需要包含首页中的实验一般信息。

3、 实验报告正文部分具体要求如下：

一、实验目的

本次实验所涉及并要求掌握的知识点。

二、实验环境

实验所使用的设备名称及规格，网络管理工具简介、版本等。

三、实验内容与实验要求

实验内容、原理分析及具体实验要求。

四、实验过程与分析

根据具体实验，记录、整理相应命令、运行结果等，包括截图和文字说明。

详细记录在实验过程中发生的故障和问题，并进行故障分析，说明故障排除的过程及方法。

五、实验结果总结

对实验结果进行分析，完成思考题目，总结实验的心得体会，并提出实验的改进意见。

六、附录

## 一、实验目的

RSA是第一个比较完善的公开密钥算法，它既能用于加密，也能用于数字签名。RSA以它的三个发明者Ron Rivest, Adi Shamir, Leonard Adleman的名字首字母命名，这个算法经受住了多年深入的密码分析，虽然密码分析者既不能证明也不能否定RSA的安全性，但这恰恰说明该算法有一定的可信性，目前它已经成为最流行的公开密钥算法。

RSA是目前最有影响力和最常用的公钥加密算法，它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。

通过用RSA算法对实际数据进行加密和解密来深刻了解RSA的运行原理及其特点， 能够编译并分析RSA算法，进而加深对非对称加密算法的理解与认识。

## 二、实验环境

1.操作系统：运行Windows ，VS2010编译环境

2.验证软件：CAP(Cryptographic Analysis Program v4)软件

## 三、实验内容与实验要求

### 任务一：RSA加解密算法的原理

首先产生密钥，过程如下：

      随机产生两个长度为K/2位的素数P 和 Q；

      计算公钥 publicKey=P\*Q;(publicKey 是k位的长度)

      随机产生一个加密密钥 keyE, 2<=keyE<=Φ(n)-1其中GCD(keyE, Φ(n))=1；注：这是保证解密密钥keyE \*keyD mod Φ(n)=1 有解的充要条件， Φ(n)称为n的欧拉函数,值为: Φ(n)=(P-1)\*(Q-1)

      求解解密密钥keyD=keyE-1 mod (n) ，keyE-1为解密密钥keyD的逆元 ，此公式原方程为(keyE\*keyD mod (n)=1)

      由此公钥，加密密钥，解密密钥全部产生。

      其次，对明文加密或对密文进行解密,过程如下：

      (1) 加密: C = MkeyE mod publicKey;其中M表示明文，C表示密文。

      (2) 解密: M = CkeyD mod publicKey. 其中M表示明文，C表示密文。

### 任务二：使用C语言实现IDEA加密算法

      编译运行下列代码，并对代码进行分析

|  |
| --- |
| \*/  #include "stdafx.h"  #include <iostream>  #include <stdlib.h>  #include <time.h>    using namespace std;//RSA算法所需参数  typedef struct  RSA\_PARAM\_Tag  {       unsigned \_\_int64    p, q;   //两个素数，不参与加密解密运算      unsigned \_\_int64    f;      //f=(p-1)\*(q-1)，不参与加密解密运算      unsigned \_\_int64    n, e;   //公匙，n=p\*q，gcd(e,f)=1      unsigned \_\_int64    d;      //私匙，e\*d=1 (mod f)，gcd(n,d)=1      unsigned \_\_int64    s;      //块长，满足^s<=n的最大的s，即log2(n)  } RSA\_PARAM;//小素数表    const static long g\_PrimeTable[]=                {3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97  };  const static long   g\_PrimeCount=sizeof(g\_PrimeTable) / sizeof(long);  const unsigned \_\_int64  multiplier=12747293821;  const unsigned \_\_int64  adder=1343545677842234541;//随机数类    class  RandNumber  {  private:      unsigned \_\_int64    randSeed;    public:      RandNumber(unsigned \_\_int64 s=0);      unsigned \_\_int64    Random(unsigned \_\_int64 n);  };    RandNumber::RandNumber(unsigned \_\_int64 s)  {       if(!s)      {      randSeed= (unsigned \_\_int64)time(NULL);       }      else      {      randSeed=s;       }  }    unsigned \_\_int64 RandNumber::Random(unsigned \_\_int64 n)  {       randSeed=multiplier \* randSeed + adder;      return randSeed % n;  }    static RandNumber   g\_Rnd;/\* 模乘运算，返回值x=a\*b mod n  \*/  inline unsigned \_\_int64 MulMod(unsigned \_\_int64 a, unsigned \_\_int64 b, unsigned \_\_int64 n)  {       return a \* b % n;  }    /\* 模幂运算，返回值x=base^pow mod n \*/  unsigned \_\_int64 PowMod(unsigned \_\_int64 &base, unsigned \_\_int64 &pow, unsigned \_\_int64 &n)  {       unsigned \_\_int64    a=base, b=pow, c=1;      while(b)      {   while(!(b & 1))          {              b>>=1;  //a=a \* a % n;    //函数看起来可以处理位的整数，但由于这里a\*a在a>=2^32时已经造成了溢出，因此实际处理范围没有位              a=MulMod(a, a, n);          }        b--;        //c=a \* c % n;        //这里也会溢出，若把位整数拆为两个位整数不知是否可以解决这个问题。          c=MulMod(a, c, n);      }       return c;  }    /\*  Rabin-Miller素数测试，通过测试返回，否则返回。n是待测素数。注意：通过测试并不一定就是素数，非素数通过测试的概率是/4  \*/  long RabinMillerKnl(unsigned \_\_int64 &n)  {       unsigned \_\_int64    b, m, j, v, i;      m=n - 1;      j=0;         //0、先计算出m、j，使得n-1=m\*2^j，其中m是正奇数，j是非负整数      while(!(m & 1))      {      ++j;          m>>=1;      }         //1、随机取一个b，<=b<n-1      b=2 + g\_Rnd.Random(n - 3);       //2、计算v=b^m mod n      v=PowMod(b, m, n);       //3、如果v==1，通过测试      if(v == 1)      {      return 1;      }       //4、令i=1      i=1;      b=2;       //5、如果v=n-1，通过测试      while(v != n - 1)      {      //6、如果i==l，非素数，结束          if(i == j)          {          return 0;           }           //7、v=v^2 mod n，i=i+1          v=PowMod(v, b, n);          ++i;           //8、循环到      }       return 1;  }    /\*   Rabin-Miller素数测试，循环调用核心loop次全部通过返回，否则返回   \*/  long RabinMiller(unsigned \_\_int64 &n, long loop)  {       //先用小素数筛选一次，提高效率      for(long i=0; i < g\_PrimeCount; i++)      {      if(n % g\_PrimeTable[i] == 0)          {          return 0;           }      }         //循环调用Rabin-Miller测试loop次，使得非素数通过测试的概率降为(1/4)^loop      for(long i=0; i < loop; i++)      {    if(!RabinMillerKnl(n))          {   return 0;    }      }    return 1;  }    /\* 随机生成一个bits位(二进制位)的素数，最多位\*/  unsigned \_\_int64 RandomPrime(char bits)  {       unsigned \_\_int64    base;      do      {   base= (unsigned long)1 << (bits - 1);   //保证最高位是          base+=g\_Rnd.Random(base);               //再加上一个随机数          base|=1;    //保证最低位是，即保证是奇数      }       while(!RabinMiller(base, 30));    //进行拉宾－米勒测试次      return base;    //全部通过认为是素数  }    /\*欧几里得法求最大公约数\*/  unsigned \_\_int64 EuclidGcd(unsigned \_\_int64 &p, unsigned \_\_int64 &q)  {       unsigned \_\_int64    a=p > q ? p : q;      unsigned \_\_int64    b=p < q ? p : q;      unsigned \_\_int64    t;      if(p == q)      {      return p;   //两数相等，最大公约数就是本身      }      else      {      while(b)    //辗转相除法，gcd(a,b)=gcd(b,a-qb)          {   a=a % b;              t=a;              a=b;              b=t;           }           return a;       }  }    /\*  Stein法求最大公约数 \*/  unsigned \_\_int64 SteinGcd(unsigned \_\_int64 &p, unsigned \_\_int64 &q)  {       unsigned \_\_int64    a=p > q ? p : q;      unsigned \_\_int64    b=p < q ? p : q;      unsigned \_\_int64    t, r=1;      if(p == q)      {      return p;           //两数相等，最大公约数就是本身      }      else      {      while((!(a & 1)) && (!(b & 1)))          {          r<<=1;          //a、b均为偶数时，gcd(a,b)=2\*gcd(a/2,b/2)              a>>=1;              b>>=1;          }             if(!(a & 1))          {   t=a;            //如果a为偶数，交换a，b              a=b;              b=t;           }        do          {          while(!(b & 1))              {                b>>=1; //b为偶数，a为奇数时，gcd(b,a)=gcd(b/2,a)              }                  if(b < a)              {                t=a;        //如果b小于a，交换a，b                  a=b;                  b=t;                }                b=(b - a) >> 1; //b、a都是奇数，gcd(b,a)=gcd((b-a)/2,a)          } while(b);            return r \* a;       }  }    /\*  已知a、b，求x，满足a\*x =1 (mod b)相当于求解a\*x-b\*y=1的最小整数解  \*/  unsigned \_\_int64 Euclid(unsigned \_\_int64 &a, unsigned \_\_int64 &b)  {       unsigned \_\_int64    m, e, i, j, x, y;      long                xx, yy;      m=b;      e=a;      x=0;      y=1;      xx=1;      yy=1;      while(e)      {      i=m / e;          j=m % e;          m=e;          e=j;          j=y;          y\*=i;          if(xx == yy)          {          if(x > y)              {                y=x - y;                }              else              {                y-=x;    yy=0;                }           }          else          {          y+=x;   xx=1 - xx;  yy=1 - yy;           }           x=j;           }           if(xx == 0)           {           x=b - x;           }       return x;  }    /\* 随机产生一个RSA加密参数\*/  RSA\_PARAM RsaGetParam(void)  {       RSA\_PARAM           Rsa={ 0 };      unsigned \_\_int64    t;      Rsa.p=RandomPrime(16);          //随机生成两个素数      Rsa.q=RandomPrime(16);      Rsa.n=Rsa.p \* Rsa.q;      Rsa.f=(Rsa.p - 1) \* (Rsa.q - 1);        do      {      Rsa.e=g\_Rnd.Random(65536);  //小于^16，=2^16           Rsa.e|=1;   //保证最低位是，即保证是奇数，因f一定是偶数，要互素，只能是奇数      }while(SteinGcd(Rsa.e, Rsa.f) != 1);         Rsa.d=Euclid(Rsa.e, Rsa.f);      Rsa.s=0;   t=Rsa.n >> 1;       while(t)      {      Rsa.s++;   //s=log2(n)          t>>=1;       }       return Rsa;  }    /\* 拉宾－米勒测试 \*/  void TestRM(void)  {       unsigned long   k=0;      cout << " - Rabin-Miller prime check.\\\\n" << endl;      for(unsigned \_\_int64 i=4197900001; i < 4198000000; i+=2)      {      if(RabinMiller(i, 30))          {          k++;              cout << i << endl;           }       }       cout << "Total: " << k << endl;  }    /\*  RSA加密解密 \*/  void TestRSA(void)  {       RSA\_PARAM   r;      char     pSrc[]="wellcome to bjhit!";      const unsigned long n=sizeof(pSrc);      unsigned char       \*q, pDec[n];      unsigned \_\_int64    pEnc[n];      r=RsaGetParam();      cout << "p=" << r.p << endl;      cout << "q=" << r.q << endl;      cout << "f=(p-1)\*(q-1)=" << r.f << endl;      cout << "n=p\*q=" << r.n << endl;      cout << "e=" << r.e << endl;      cout << "d=" << r.d << endl;      cout << "s=" << r.s << endl;      cout << "Source:" << pSrc << endl;      q= (unsigned char \*)pSrc;      cout << "Encode:";      for(unsigned long i=0; i < n; i++)      {          unsigned \_\_int64 temp = (unsigned \_\_int64)q[i];      pEnc[i]=PowMod(temp, r.e, r.n);          cout << hex << pEnc[i] << " ";      }       cout << endl;      cout << "Decode:";      for(unsigned long i=0; i < n; i++)      {      pDec[i]=PowMod(pEnc[i], r.d, r.n);          cout << hex << (unsigned long)pDec[i] << " ";      }       cout << endl;      cout << (char \*)pDec << endl;  }    int main(void)  {       TestRSA();      return 0;  } |

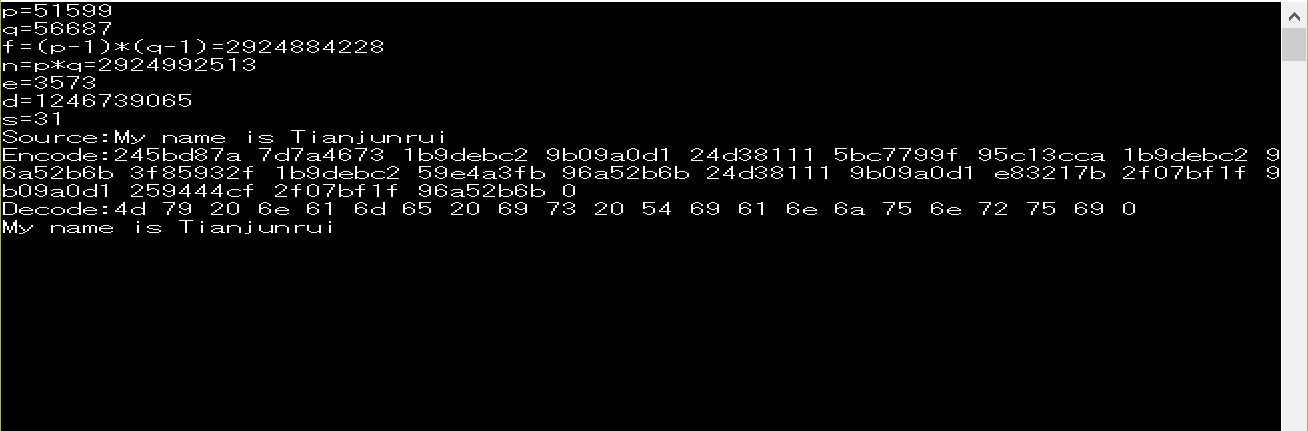
## 四、实验过程与分析

#### 任务一：RSA加解密算法流程图

|  |
| --- |
| 开始  随机产生两个长度为K/2位的素数P和Q  计算公钥publicKey = P \* Q  计算Φ(n) = (P-1)\*(Q-1)  生成密钥  随机产生一个加密密钥keyE  GCD(keyE,Φ(n)) = 1  求解解密密钥keyD = keyE-1 mod Φ(n)  输入明文M  加密过程  密文 C = 明文 MkeyE mod publicKey  输出密文C  M = CkeyD mod publicKey  解密过程  解密过程  输出明文M |

#### 任务二

**我将字符明文改为“My name is Tianjunrui”**



## 五、实验结果总结

**实验结果分析：**

申明原文 pSrc = “My name is Tianjunrui”

根据流程图所示的原理，获取RSA加密模块的参数信息，p,q,f,n,e,d,s。

随机产生了两个大的质数，p=51599，q=56687；

计算f=(p－1)\*(q－1)=2924884228，n=p\*q=2924992513；

诸位取 pSrc 的十进制表示，按照加密模块的信息 e,n 进行加密。

然后就算出加密密钥e；再计算出解密密钥d；

输出解密后的字符串，和原来加密前的文本一致说明程序执行正确。

## 六、思考题目：

#### 1）通过本实验，论述RSA算法的加密原理是什么？

输入待加密的明文，记为C，调用RsaGetParam随机产生两个大素数，记为P和Q，求以P和Q的乘积N为参数的欧拉函数值，记为F，其中，F = （P - 1）\*（Q - 1），计算加密指数e，其中e与F有最大公因数1，即e与F互质, 且1<e<F。根据e产生解密指数d，d为e在模F下的乘法逆元，即（e\*d）= 1 mod F.即e\*d = k \* F + 1，可写成裴蜀等式的形式：e\*d + y \* F= gcd（e，F）=1，用扩展的欧几里得算法求解该式即可求得e的乘法逆元d。现求得了加密指数e和解密指数d，用加密指数e对明文c作指数操作再模P，Q乘积N，即密文m= c^e mod N.解密时再用解密指数d来解密，即c = m ^d mod N.

#### 2）在上述算法中哪些模块是该算法的核心模块？

RandomPrime(16)：随机生成一个bits位(二进制位)的素数

|  |
| --- |
| unsigned \_\_int64 RandomPrime(char bits)  {  unsigned \_\_int64 base;  do{  base= (unsigned long)1 << (bits - 1); //保证最高位是  base+=g\_Rnd.Random(base); //再加上一个随机数  base|=1; //保证最低位是，即保证是奇数  }  while(!RabinMiller(base, 30)); //进行拉宾－米勒测试次  return base; //全部通过认为是素数  } |

Euclid：欧几里得法求最大公约数

|  |
| --- |
| unsigned \_\_int64 EuclidGcd(unsigned \_\_int64 &p, unsigned \_\_int64 &q)  {  unsigned \_\_int64 a=p > q ? p : q;  unsigned \_\_int64 b=p < q ? p : q;  unsigned \_\_int64 t;  if(p == q){  return p; //两数相等，最大公约数就是本身  }  else{  while(b){ //辗转相除法，gcd(a,b)=gcd(b,a-qb)  a=a % b;  t=a;  a=b;  b=t;  }  return a;  }  } |

MulMod：模乘运算

|  |
| --- |
| static RandNumber g\_Rnd;/\* 模乘运算，返回值x=a\*b mod n \*/  inline unsigned \_\_int64 MulMod(unsigned \_\_int64 a, unsigned \_\_int64 b, unsigned \_\_int64 n)  {  return a \* b % n;  } |

PowMod：模幂运算 返回值x=base^pow mod n

|  |
| --- |
| unsigned \_\_int64 PowMod(unsigned \_\_int64 &base, unsigned \_\_int64 &pow, unsigned \_\_int64 &n)  {  unsigned \_\_int64 a=base, b=pow, c=1;  while(b){  while(!(b & 1)){  b>>=1; //a=a \* a % n; //函数看起来可以处理位的整数，但由于这里a\*a在a>=2^32时已经造成了溢出，因此实际处理范围没有位  a=MulMod(a, a, n);  }  b--; //c=a \* c % n; //这里也会溢出，若把位整数拆为两个位整数不知是否可以解决这个问题。  c=MulMod(a, c, n);  }  return c;  } |

SteinGcd： 利用Stein法求最大公约数

|  |
| --- |
| unsigned \_\_int64 SteinGcd(unsigned \_\_int64 &p, unsigned \_\_int64 &q)  {  unsigned \_\_int64 a=p > q ? p : q;  unsigned \_\_int64 b=p < q ? p : q;  unsigned \_\_int64 t, r=1;  if(p == q){  return p; //两数相等，最大公约数就是本身  }  else{  while((!(a & 1)) && (!(b & 1))){  r<<=1; //a、b均为偶数时，gcd(a,b)=2\*gcd(a/2,b/2)  a>>=1;  b>>=1;  }  if(!(a & 1)){  t=a; //如果a为偶数，交换a，b  a=b;  b=t;  }  do{  while(!(b & 1)){  b>>=1; //b为偶数，a为奇数时，gcd(b,a)=gcd(b/2,a)  }  if(b < a){  t=a; //如果b小于a，交换a，b  a=b;  b=t;  }  b=(b - a) >> 1; //b、a都是奇数，gcd(b,a)=gcd((b-a)/2,a)  } while(b);  return r \* a;  }  } |

#### 3）**对于一个RSA加密算法的密文，要得到明文需要哪些要素？**

需要公钥publickey，和解密密钥keyD或者强行分解公钥得到两个质数来得到明文。

## 七、实验心得体会：

RSA公开密钥密码体制。所谓的公开密钥密码体制就是使用不同的加密密钥与解密密钥，是一种“由已知加密密钥推导出解密密钥在计算上是不可行的”密码体制。

RSA加密密钥是非对称的，一般是成对出现分为公钥和私钥，所以也叫非对称加密，公钥加密，私钥解密。并且产生的两个素数越大，公钥和密钥越难破解，越安全。RSA是一个安全性很高的加密算法。

RSA的安全性依赖于大数分解，但是否等同于大数分解一直未能得到理论上的证明，因为没有证明破解RSA就一定需要作大数分解。假设存在一种无须分解大数的算法，那它肯定可以修改成为大数分解算法。 RSA 的一些变种算法已被证明等价于大数分解。不管怎样，分解n是最显然的攻击方法。人们已能分解多个十进制位的大素数。因此，模数n必须选大一些，因具体适用情况而定。