南開大學
Nankai University

南　开　大　学

网 络 空 间 安 全 学 院

密码学实验报告

密码学大作业：保密通信

罗功成 1910487

年级：2019 级

专业：信息安全

2022 年 6 月 16 日

目录

一、 实验目的

利用 rsa 公钥加密算法和 MD5 哈希算法，实现消息的完整性检验和发送者身份验证的功能。

二、 实验原理

1. 套接字 (Socket)，就是对网络中不同主机上的应用进程之间进行双向通信的端点的抽象。一个套接字就是网络上进程通信的一端，提供了应用层进程利用网络协议交换数据的机制

主要的工作流程为服务器监听，客户端请求，连接建立确认，数据传输。

2.Hash 函数是将任意长的输入转换成一个较短的固定长度输出数字串的函数，可用于数字签名、消息的完整性检验、消息的来源认证检测等, 具有快速性、单向性、无碰撞等特点

MD5 算法对任意长度的输入值处理后产生 128 位的 Hash 值，具体算法实现为：

(1) 首先需要对信息进行填充，使其字节长度与 448 模 512 同余，在信息的后面填充第一位为 1，其余各位均为 0，直到满足上面的条件时才停止用 0 对信息的填充。

(2) 设置好这 4 个链接变量后，就开始进入算法的 4 轮循环运算。循环的次数是信息中 512 位信息分组数目

(3) 然后进入主循环，主循环有 4 轮，每轮循环都很相似。第一轮进行 16 次操作，每次操作对 A、B、C、D 中的 3 个做一次非线性函数运算，然后将所得结果加上第四个变量，文本的一个子分组（32 位）和一个常数。再将所得结果向左循环移 S 位

(4)4 轮 64 次操作的运算结束后，将 A，B，C，D 分别加上 A0，B0，C0，D0。然后用下一分组数据继续进行运算，最后得到一组 A，B，C，D。把这组数据级联起来，即得到 128 比特的 Hash 结果。

3.RSA 算法安全性建立在“大数分解和素性检测”基础上, 具体算法实现如下：

(1) 公钥：选择两个不同的大素数 p 和 q，n 是二者的乘积，即 n = pq, 随机选取正整数 e，使 e 和 n 的欧拉函数互素，则将（n，e）作为公钥。

(2) 私钥：求出正数 d，使其满足 e,d 在 mod n 的欧拉函数时互为乘法逆元，则将（n，d）作为私钥

(3) 加密/解密：由 c=m 的 e 次方 mod n 得到密文 c, 由 m=c 的 d 次方 mod n 得到明文 m

三、 实验要求

假设通讯双方为 A、B，并假设发方 A 拥有自己的 RSA 公钥 PK_A 和私钥 SK_A，同时收方 B 已经通过某种方式知道了发方的公钥 PK_A 。

要求对发方 A 发来的消息，收方 B 通过检验，能够确定：

1、B 收到的消息是完整的，即消息在传送过程中没有遭到非法修改；

2、B 收到的消息来源是真实的，即该消息的确是由 A 发来的，而不是由其他人伪造的。

四、 程序流程图

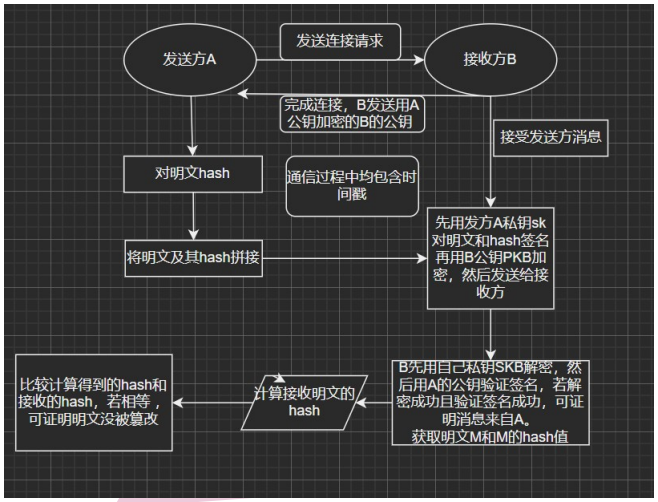

图 1: 保密通信的整体流程

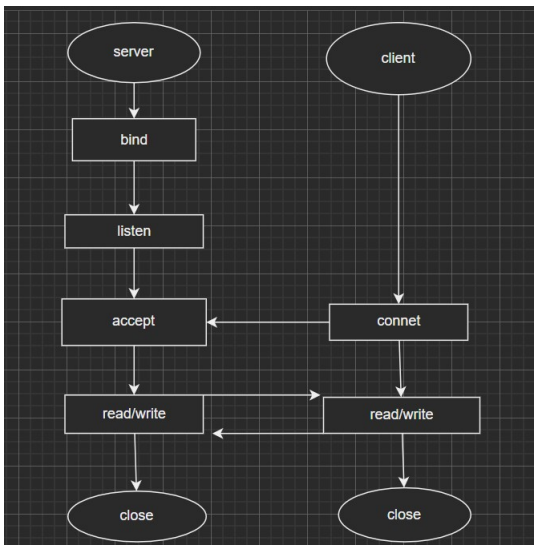

图 2: 客户端和服务器通信过程

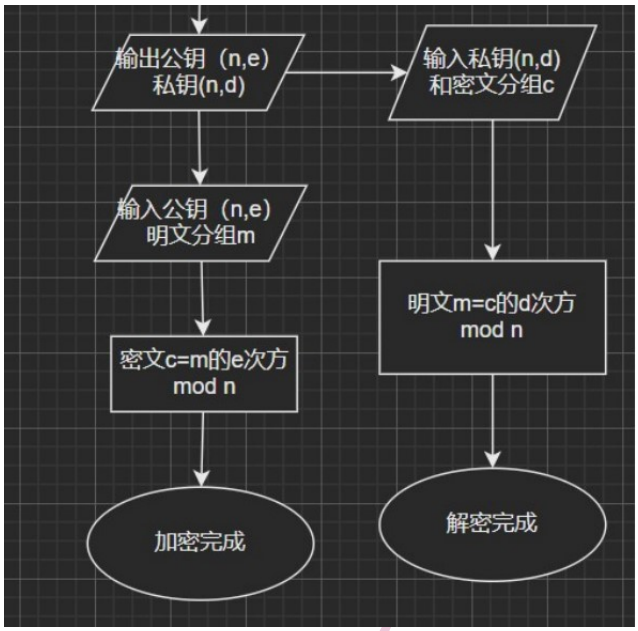

图 3: RSA 加解密过程

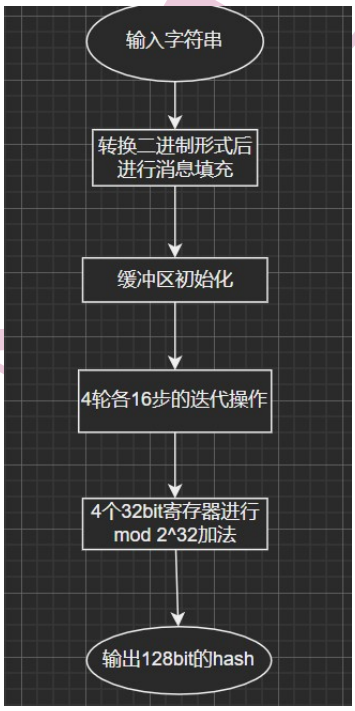

图 4: MD5 计算 hash 过程

五、 代码说明

发方用户 A 文件清单：

保密通信客户端发方用户 A.exe: 用户 A 程序（发方），主要有：RSA 密钥生成（含随机大数生成、Rabin-miller 检测等），TCP 客户端通信功能（如请求连接、文件读写等），RSA 加解

密，MD5hash 计算，时间戳等功能。

md5.h:md5 的 hash 计算;socket 初始化.h：TCP socket 套接字初始化结构和参数；

大数定义类.h: 定义 RSA 大数类的相关操作；素性检测.h: 随机大数生成，rabin-miller 检测的定义，两者完成 RSA 的基本功能。

保密通信客户端发方用户 A.cpp: 完成发方用户 A 各项功能的 C++ 代码

收方用户 B 文件清单：

保密通信服务器收方用户 B.exe: 用户 A 程序（收方），主要有：RSA 密钥生成（含随机大数生成、Rabin-miller 检测等）TCP 服务器端通信功能（如建立连接、文件读写等），RSA 加解密，MD5hash 计算，时间戳等功能。

md5.h:md5 的 hash 计算;socket 初始化.h：TCP socket 套接字初始化结构和参数；

大数定义类.h: 定义 RSA 大数类的相关操作；素性检测.h: 随机大数生成，rabin-miller 检测的定义，两者完成 RSA 的基本功能。

保密通信服务器收方用户 B.cpp: 完成收方用户 B 功能的 C++ 代码。

1.exe 文件中每次启动会生成新的 A，B 的公私钥，且 B 中已知 A 的公钥

2.RSA 参与运算和输入输出时需要大数（密钥、明文等）的形式，目前大数部分的功能支持整型和大数类型的数对大数进行赋值操作。因此如果需要传输带有非数字字符的消息，程序将转换为对应 ASCII 码后进行输入，接收消息解密后也将 ASCII 码转换为对应的字符后恢复消息。

3. 注意 bind 的端口号，不要使用已经被占用的；

4. 注意程序运行次序，先运行作为服务器端收方 B，密钥生成结束后提示“等待客户端连接”后，再运行作为客户端的发方 A 进行密钥生成、请求连接等操作；

5. 此外，有时第一次打开或者短时间内多次关闭再启动等情况，可能出现服务器端发送加密的 PKB 失败导致的无法通信的问题，应该是进程冲突或通信端口被占用的问题，或者 Rabin 选取次数过少，不满足素数条件导致的计算错误，还可能是上一次关闭程序后进程资源没有完全释放等的问题，一般可以通过关闭两个程序后，按说明的次序重新运行程序的方式来解决。

六、 实验环境

windows10+visual studio2022

收发方程序已在 win10 平台上通过多次测试，按照代码说明部分操作均可正常完成功能。

七、 设计思路

1.socket 建立通信：

利用 TCP socket 套接字建立连接，主要过程如流程图中所示，服务器端（用户 B）用 bind 函数绑定到本地地址，然后监听；用户端（用户 A）发起 connect 请求后，用户 B 接受并建立连接，之后双方可进行消息交互.

2.MD5 算法：

如流程图和实验原理处，这里注意对明文消息做好填充, 分组处理，4 轮共 64 次迭代过程中使用的逻辑函数、移位方式有所不同等要点即可。

3.RSA 算法：

（1）设计一个大整数类, 生成需要的大素数在类中完成加减乘除取模和模幂等 RSA 算数运算和逻辑判断（>,<,=）等基本运算功能

（2）连续多次使用 rand() 生成若干个较小的数，再将小数进行左移位后拼接在一起，构成一个随机生成的大数

（3）在随机生成的大数中取奇数，然后进行多轮 RabinMiller 检测。若多次检测成功就可以认为大概率是一个素数

（4）加密/解密：见实验原理部分。

4. 防范恶意行为的安全设置：

针对可能存在的中间人攻击和窃听、伪造，实验中增加了 B 的公私钥对，用于对 A 发送消息时的加密保护；

针对可能存在的重放攻击，实验中增加了时间戳，用以确保消息的新鲜性。

八、 核心代码

socket 的初始化在对应.h 文件中，这里绑定端口为 4567，默认本机地址 127.0.0.1, 时间戳用 ctime 来实现

通信服务器

```
// 创建socket
    SOCKET sListen = ::socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    // 填充sockaddr_in结构
    sockaddr_in sin;
    sin.sin_family = AF_INET;
    sin.sin_port = htons(4567);
    sin.sin_addr.S_un.S_addr = INADDR_ANY;

    // bind本地地址
    if (::bind(sListen, (LPSOCKADDR)&sin, sizeof(sin)) == SOCKET_ERROR)
    {
        printf("Failed bind() /n");
        return 0;
    }

    // 监听有无连接
    if (::listen(sListen, 2) == SOCKET_ERROR)
    {
        printf("Failed listen() /n");
        return 0;
    }
    cout << "等待客户端连接" << endl;
    // 接受客户的连接请求
    sockaddr_in remoteAddr;
    int nAddrLen = sizeof(remoteAddr);
    SOCKET sClient = 0;
    char sendmessage[] = "";//向A发送数据
    while (sClient == 0)
    {
        // 接受一个新连接
        sClient = ::accept(sListen, (SOCKADDR*)&remoteAddr, &nAddrLen);
        if (sClient == INVALID_SOCKET)
        {
            printf("Failed accept()");
            time_t t;
            time(&t);
            printf("日期和时间%s", ctime(&t));
        }


        printf("接受到一个连接: %s /r/n", inet_ntoa(remoteAddr.sin_addr));
        time_t t;
        time(&t);
        printf("连接建立的时间%s", ctime(&t));
        continue;
    }
    //发送
    ::send(sClient, sendmessage, strlen(sendmessage), 0);
```

```
49
50
51
52          // 从客户端接收数据
53          char recvbuff[1024];
54          string asss;
55          int nRecv = ::recv(sClient, recvbuff, 1024, 0);
56          recvbuff[nRecv] = 0;
57      // 关闭同客户端的连接
58      ::closesocket(sClient);
59
60      // 关闭监听套节字
61      ::closesocket(sListen);
```

通信客户端

```
1  // 创建套节字
2      SOCKET s = ::socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
3      if (s == INVALID_SOCKET)
4      {
5          printf(" Failed socket() /n");
6          return 0;
7      }
8
9
10     // 填写远程地址信息
11     sockaddr_in servAddr;
12     servAddr.sin_family = AF_INET;
13     servAddr.sin_port = htons(4567);
14
15     servAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");//设置本地主机ip
16
17     if (::connect(s, (sockaddr*)&servAddr, sizeof(servAddr)) == -1)
18     {
19         printf(" Failed connect() /n");
20         time_t t;
21         time(&t);
22         printf("建立连接的时间%s", ctime(&t));//显示对应的年月日和时间秒
23         return 0;
24     }
25
26     //进行保密通信
27
28     char recvbuff[1024];//接收数据的缓冲区
29
30     char sendtext[2048];//发送数据缓冲区
31
32     //while (TRUE)
33     {
```

```
        //从用户B端接收被PKA加密的B的公钥PKB
        int nRecv = ::recv(s, recvbuff, 1024, 0);
        if (nRecv > 0)
        {
            recvbuff[nRecv] = 0;//修改缓冲区

            //要改!!
            printf("接收到PKA加密的PKB: %s/n", recvbuff);
            time_t t;
            time(&t);
            printf("消息发送的日期和时间 %s", ctime(&t));
        }
    //发送数据
    ::send(s, sendtext, strlen(sendtext), 0);

    }

    // 关闭套节字
    ::closesocket(s);
```

md5 涉及的移位表，逻辑函数等在 md5.h 中保存，这里只列出 md5 运算的核心逻辑

MD5 实现

```
string md5(string m)//对明文m进行hash
{
        int mlength = m.length();//明文长度
        int mgroup = ((mlength + 8) / 64) + 1;//明文分组个数

        unsigned int* mbit = new unsigned int[mgroup * 16];
        memset(mbit, 0, sizeof(unsigned int) * mgroup * 16);//内存初始化

        for (int i = 0; i < mlength; i++)
        {
                mbit[i / 4] |= m[i] << ((i % 4) * 8);
        }
        mbit[mlength >> 2] |= 0x80 << ((mlength % 4) * 8);

        mbit[mgroup * 16 - 2] = mlength * 8;

        //分组处理消息
        unsigned int a, b, c, d;
        for (int i = 0; i < mgroup; i++)
        {
                a = A;
                b = B;
                c = C;
                d = D;
                unsigned int x1;
```

```
                int x2;

                for (int j = 0; j < 64; j++)//4个16步迭代操作
                {
                        if (j < 16)
                        {
                                x1 = F(b, c, d);//逻辑函数依次为FGHI
                                x2 = j;//对字进行置换
                        }
                        else if (j >= 16 && j < 32)
                        {
                                x1 = G(b, c, d);
                                x2 = (1 + 5 * j) % 16;
                        }
                        else if (j >= 32 && j < 48)
                        {
                                x1 = H(b, c, d);
                                x2 = (5 + 3 * j) % 16;
                        }
                        else if (j >= 48 && j < 64)
                        {
                                x1 = I(b, c, d);
                                x2 = (7 * j) % 16;
                        }
                        //压缩函数的迭代
                        unsigned temp = d;
                        d = c;
                        c = b;
                        b = b + shift(a + x1 + mbit[i * 16 + x2] + T[j],
                            Lmove[j]);
                        a = temp;
                }
                A = a + A;
                B = b + B;
                C = c + C;
                D = d + D;
        }
        return inttohex(A) + inttohex(B) + inttohex(C) + inttohex(D);//将hash
            用16进制形式输出
}
```

RSA 的实现部分在对应 RSA 实验部分有较为详细的说明，在对应大数生成.h, 素性检测.h 中包含了需要的表，大数的相关运算逻辑，随机大数生成的方式，rabin-miller 检测的实现等功能，并有对应的注释。

随机大数生成

```

```

```
main.cpp中: srand((unsigned int)time(NULL));//用时间来更换不同的随机种子
//改为在程序开始处调用一次即可

void bignumber::Random()
{
        for (int i = 0; i < 16; i++)//64/4=16
                //RAND()最多0X7FFF的数，需要3次调用
                data[i] = (rand() << 17) + (rand() << 2) + rand() % 4;
}

//产生一个可能是素数的大数,
//此数为奇数,且不能被小于2000的素数整除时，进行rabin检测
void makeprime(bignumber& n)
{
        int i = 0;
        bignumber test;
        const int length = sizeof(prime_table) / sizeof(int);
        while (i != length)
        {
                n.Random();
                while (!n.IsOdd())
                        n.Random();

                i = 0;
                for (; i < length; i++)
                {
                        test = prime_table[i];
                        if ((n % test) == 0)
                                break;
                }
        }
}
```

如果一个大素数能够通过多次连续的 rabin-miller 检测，我们可以认为大概率上是一个素数。

rabin-miller 素性检测

```

//对大奇数n进行RabinMiller检测
bool RabinMiller(const bignumber& n)//课本P94算法描述
{
        bignumber x, y, z;
        unsigned int p, q;
        x = n - 1;
        p = 0;

        while (!x.IsOdd())//n-1的二进制描述
        {
                p++;
```

```
13                  x >> 1;
14          }
15
16          bool flag1 = true;
17          while (flag1)
18          {
19                  y.Random();//随机生成一个y来作为检测数
20             //y应该小于N的整数
21                  if (y < n)
22                          flag1 = false;
23
24          }
25          y = pow_mod(y, x, n);
26
27          if ((!(y == 1)) && (!(y == (n - 1))))
28          {
29                  q = 1;
30                  while ((n <= p - 1) && (!(y == (n - 1))))
31                  {
32                          y = (y * y) % n;
33                          if (y == 1)
34                                  return false;
35                          q++;
36                  }
37                  if (!(y == (n - 1)))
38                          return false;
39          }
40          return true;
41  }
42
43  //多次测试
44  bignumber testnumber()
45  {
46
47          bignumber n;
48          int j = 0;
49          int m;
50          cout << "对生成的数进行几次rabinmiller" << endl;
51          cin >> m;
52
53          while (j < m)//每个大数10次rabinmiller通过，可基本认为是素数了。
54          {
55                  cout << "随机生成大数: " << endl;
56                  makeprime(n);
57                  n.show();
58                  for (; j < m; j++)
59                  {
60
```

```
                        if (!RabinMiller(n))
                        {
                                cout << "在"<<m<<"轮以内测试失败" << endl;
                                break;
                        }
                        cout << "通过第"<<(j+1)<<"轮RabinMiller测试" << endl;
                }
        }
        return n;
}
```

拓展欧几里得算法求乘法模逆

```
//用扩展欧几里德算法求乘法模逆
bignumber Egcd(const bignumber& a, const bignumber& b, bignumber& x,
    bignumber& y)
{
        bignumber t, d;
        //如果一个操作数为零则无法进行除法运算
        if (b == 0)
        {
                x = 1;
                y = 0;
                return a;
        }
        d = Egcd(b, a % b, x, y);
        t = x;
        x = y;
        y = t - ((a / b) * y);
        return d;
}
```

下面依次为如何应用 RSA 进行保密通信进行代码实现。

RSA 密钥生成

```
//密钥生成

    srand((unsigned int)time(NULL));//用时间来更换不同的随机种子
    //如果只在random函数中调用，计算速度会变慢很多，因为每次调用rand重新使用
        新种子
    //改为在程序开始处调用一次即可
    cout << "生成大素数P" << endl;
    bignumber x;

    //产生大素数
    bignumber p = testnumber();
    p.show();
    cout << endl;
```

```
    for (int i = 63; i >= 0; i--)//从低位向高位输出
        cout << p.data[i];
    cout << endl;

    cout << "生成大素数Q" << endl;

    //产生大素数
    bignumber q = testnumber();

    //16进制形式显示
    q.show();
    cout << endl;

    cout << "n = p * q" << endl;
    bignumber n = p * q;

    cout << "n= " << endl;
    //16进制形式显示
    n.show();

    cout << endl;
    cout << "任意生成互素的e和乘法逆元d " << endl;

    //n的欧拉函数n1
    bignumber n1 = (p - 1) * (q - 1);

    //e为公开钥
    bignumber e;

    //d为秘密钥，即e模n1的乘法逆元
    bignumber d;

    //存储n1模e的乘法逆元
    bignumber temp1;
    bignumber temp2;

    while (1)
    {
        //随机生成互素的e
        e.Random();
        while (!(gcd(e, n1) == 1))
        {
            e.Random();//不满足就重新生成
        }

        //用扩展欧几里德算法求乘法逆元
        temp2 = Egcd(e, n1, d, temp1);

```

```
        //e*d模t结果为1时，d为e的乘法逆元
        temp2 = (e * d) % n1;
        if (temp2 == 1)
            break;

        //否则重新生成e
    }


    cout << "公钥e= " << endl;
    //16进制形式显示
    e.show();

    cout << endl;

    cout << "私钥d= " << endl;
    //16进制形式显示
    d.show();
    cout << endl;
```

A,B 间进行密钥交互和保密通信实现过程。

B 用 PKA 的 eAnA 加密 eB 拼接 nB

```
//发送加密PKA的PKB（eB|nB）
        cout << "密文C=m^e mod n" << endl;
        bignumber c1 = pow_mod(PKB, eA, nA);
        cout << "加密后的PKB为：" << endl;
        //16进制形式显示
        c1.show();
        string c1str = c1.savestring();

        cout << endl;

        for (int i = 0; i < c1str.length(); i++)
        {
            sendmessage[i] = c1str[i];//转换完的字符串逐个字符进行赋值操作
        }
```

A 用 SKA 解密得到 PKB

```
        string x1 = recvbuff;//将接收的消息转换为字符串
        x = bignumber(x1);//把字符串转换为大数
        bignumber m1 = pow_mod(x1, d, n);
        cout << "解密完成，获取PKB=(eB|nB)" << endl;
        string PKB = string(m1);//把大数转换为string，拆分得到eB,nB
        int line = PKB.find('|');//从0开始的下标位置

        int len = PKB.length();
```

```
        //substr从0开始，（开始位置，读取个数）
        string e1 = PKB.substr(0, line);//eB部分,从开头向后数line个，line+1个对应下标line“|”

        string n1 = PKB.substr(line + 1, len-1);//nB部分,从|位置的下一个开始向后读到结束

        bignumber eB = bignumber(e1);
        bignumber nB = bignumber(n1);//将截取的e1,n1转换为大数形式参与运算
```

A 用 SKA 对明文拼接 hash 签名

```

        //对明文进行hash
        cout << "请输入需要加密传输的明文" << endl;
        string m;//输入明文
        cin >> m;
        cout << "明文是：" << m << endl;
        string hashm = md5(m);
        cout << "明文hash为" << hashm << endl;

        if (flag) {//如果输入内容有除了数字外的字符，需转化
        int z1[1000];
        for (int i = 0; i < m.length(); i++)
        {
            if (m[i] < '0' || m[i]>'9')//如果不是数字
            {
                z1[i] = m[i];//将字符常量赋值给整型常量，字符可以转换为对应ascii码的形式
            }
        }}
        //将明文和明文hash拼接，中间用|隔开
        string oneline = "|";
        string ass = m + oneline + hashm;
        cout << "明文和hash拼接后为" << ass << endl;
        //用skA(d, n)签名
        // ass转换为大数y
        //cout << "m=c^d mod n" << endl;
        y1 = ass;
        y = bignumber(y1);
        bignumber mx99 = pow_mod(y, d, n);
        //用获取的PKB对签名内容加密
        //加密
        cout << "用PKB加密明文所得密文C=m^e mod n" << endl;
        bignumber c = pow_mod(z, eB, nB);
        //16进制形式显示
        cout << "实际传输的加密内容为：" << endl;
        c.show();
        string cstr = c.savestring();//长度为256
```

B 用 SKB 解密得到签名并用 PKA 验签

```
        bignumber m1 = pow_mod(c2, dB, nB);
        cout << "解密完成" << endl;


        //用已知的PKA验证签名
        bignumber m2 = pow_mod(m1, eA, nA);
        cout << "PKA验证签名通过" << endl;
        //明文|hash
```

获取明文和 hash

```

        int line = asss.find('|');//从0开始的下标位置

        //int len = s.length();
        //substr从0开始，（开始位置，读取个数）
        string s1 = asss.substr(0, line);//明文部分,从开头向后数line个，line+1个对应下标line“|”
        if (flag) {//如果消息中有除了数字之外的字符
            int z1[1000];
            for (int i = 0; i < s1.length(); i++)
            {
                if (s1[i] < '0' || s1[i]>'9')
                {
                    s1[i] = z1[i];//将int对应位置ASCII数字赋值给char,将直接转换为对应的字符
                }
            }
        }
        string s2 = asss.substr(line + 1, 32);//hash部分,从|位置的下一个开始向后读到结束，32位

        cout << "解密得到的明文为：" << endl;
        cout << s1 << endl;
        cout << "解密得到的明文hash为" << endl;
        cout << s2 << endl;
```

计算 hash，看消息是否被篡改

```
//计算明文hash
        string mhash = md5(s1);
        cout << "将明文计算hash" << endl;
        cout << mhash << endl;
        //判断hash是否相等s2
        for (int i = 0; i < 32; i++)
        {
```

```
8             if (mhash[i] != s2[i])
9             {
10                cout << "hash值不同！收到的明文已经被篡改！" << endl;
11                break;
12            }
13        }
14        cout << "如果hash相等，说明明文没有被篡改，可认为消息满足完整性" <<
              endl;
```

九、 改进与优化

对 RSA 密钥生成阶段中，如果只使用 rand(), 会发现每次生成的密钥是不变的，这无法满足生成多组公私钥对的需求。

针对这个问题，采用 srand 来随机化选取随机种子的方式来解决。

但是，如果直接在随机大数生成处添加 srand()，则会出现运算速度慢的问题，导致生成密钥的耗时很长。我认为应该是因为如果直接放在随机大数生成处，则会导致每次调用 rand()，都会重新生成一次随机数种子，然后再 rand

对这个问题，我选择将 srand() 放在 main() 下，这样每次只需要调用一次 srand，每次运行程序只需生成一次随机数种子，这样既可以保证每次使用密钥生成时可以生成不同的公私钥对，又可以提高程序的效率和减少开销。

十、 实验结果

首先启动服务器端收方用户 B，生成 nB,eB,dB;

再启动服务器端发方用户 A，生成 nA,eA,dA

大数生成和 rabin-miller 检测

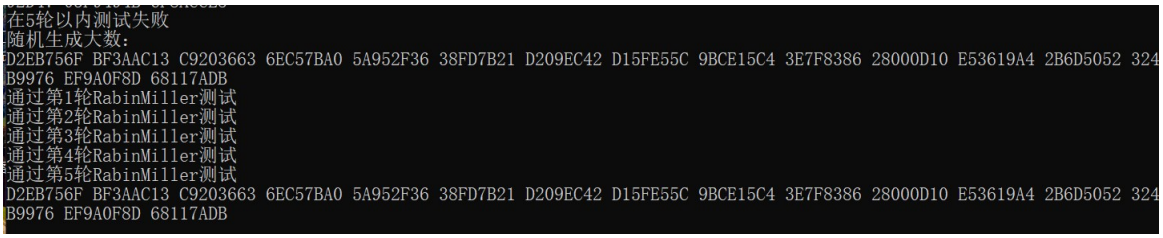

图 5: 大数和 rabin 过程示例

生成公钥（e,n）和私钥（d,n）

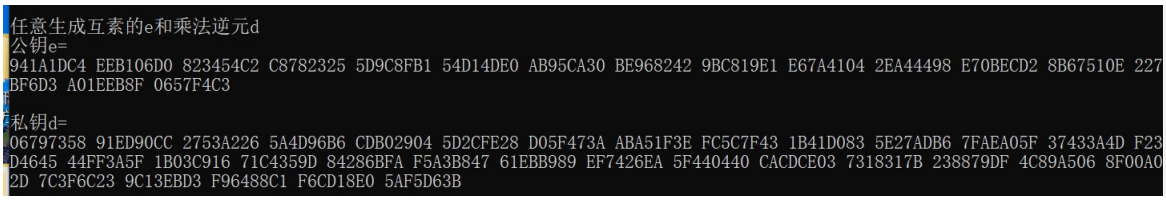

图 6: 密钥生成示例

服务器端生成密钥结束后等待用户端连接，用户端生成密钥结束后自动请求连接，建立连接完成会提示，并打印建立连接的时间

由于 A 的公钥已经由 B 以某种方式知晓了，下面 B 向 A 以 PKA 加密方式共享给 A 公钥 PKB

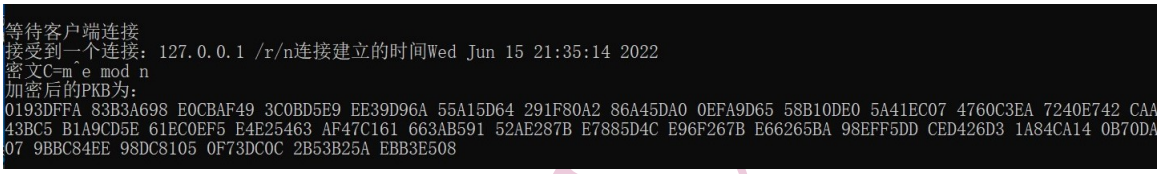

图 7: 建立连接，B 把 PKA 加密后的 PKB 发给 A

A 建立连接完成后，等待服务器端用户 B 发来的用 PKA 加密的 PKB，接收到消息时打印时间戳。

A 用 skA(dA,nA) 解密后，得到 PKB，可用于后续的保密通信。

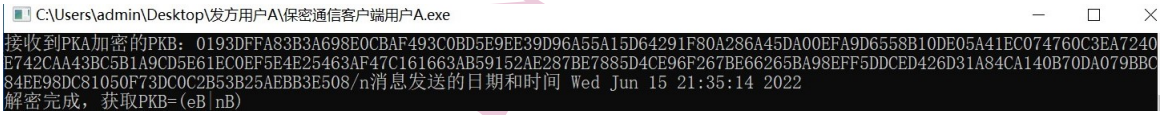

图 8: A 建立连接并解密 pkB

发方 A 程序处等待输入明文 123；接收到明文后，计算明文 hash，将明文和 hash 进行拼接，中间用 | 隔开，然后用 skA 对拼接的消息进行签名，最后再用 PKB 对签名加密，得到最后向收方 B 传输的密文消息。

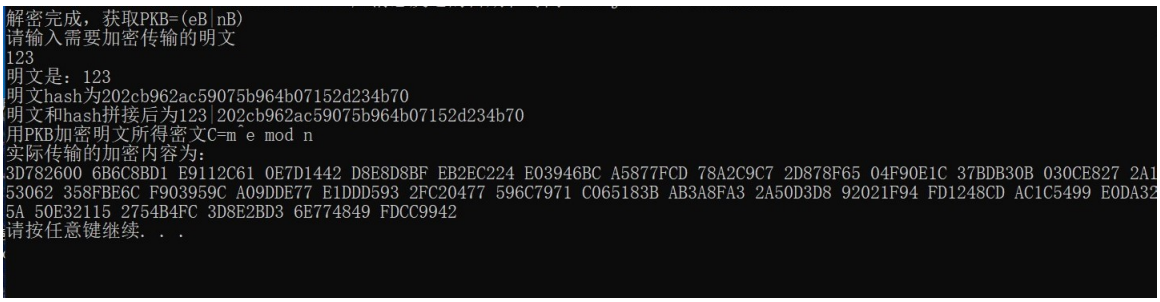

图 9: A 输入明文，计算 hash，签名，并用 PKB 加密

实际上，A 能够用 SKA 正确解密出来 pkB, 说明 A 拥有 A 的签名 skA，通过签名的不可伪造、不可抵赖等特性，B 已经就可以确信对方是 A 了。

B 接受 A 发来的消息，然后用 SKB 解密得到签名，再用已知的 PKA 验证签名。

图 10: B 接受消息

通过 A 能够正确获得 B 的公钥，且 A 发来签名消息可以成功认证，可以证明 A 确实拥有私钥 SKA，证明了消息是由 A 发来的，保证了消息来源真实性。

解密和验证签名通过后，B 获取到发方 A 发来的明文 123 和 123 的 hash，然后计算明文 123 的 hash。

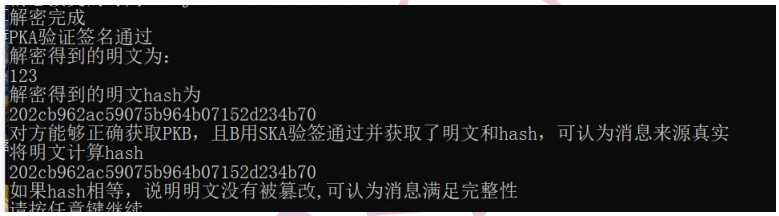

图 11: B 对明文 hash

对比接收的明文 hash，发现两者完全一致，说明消息没有被篡改, 确保消息完整性。

至此，A、B 之间的保密通信就完成了。

上面的例子是以数字为例的，这是由于大数计算过程直接对大数运算比较方便。实际上，通过把字符转换为 ASCII 码的形式，把 ASCII 码对应序号当作赋值的数字输入，然后接收解密后再逐个对应恢复出对应字符，可以支持更丰富的输入输出

下面传输含有字母数字特殊符号的明文内容，用来说明保密通信中的可传输消息的普适性。

发方输入 LGC@123NKU#1919

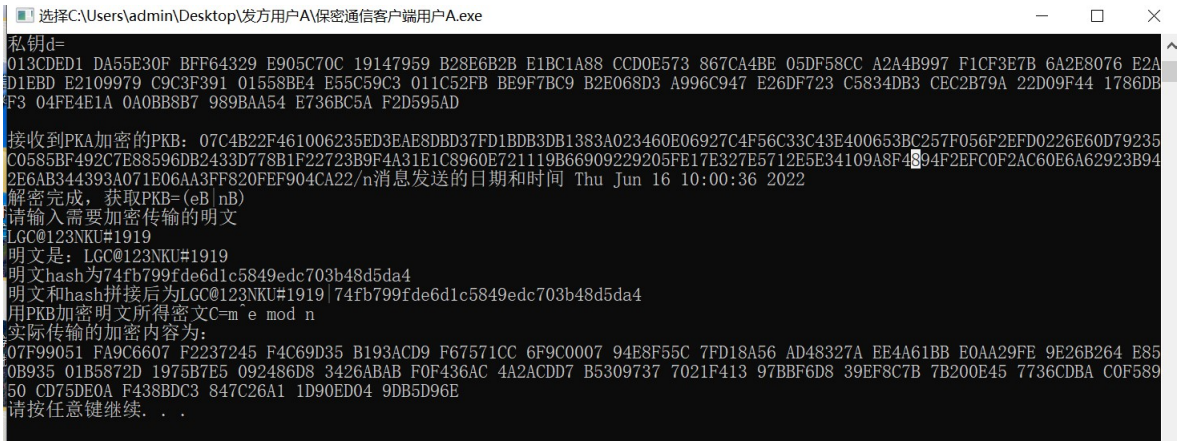

图 12: 发方 A

收方最终获取的明文也是 LGC@123NKU#1919，说明了可以传输字母数字和特殊字符。

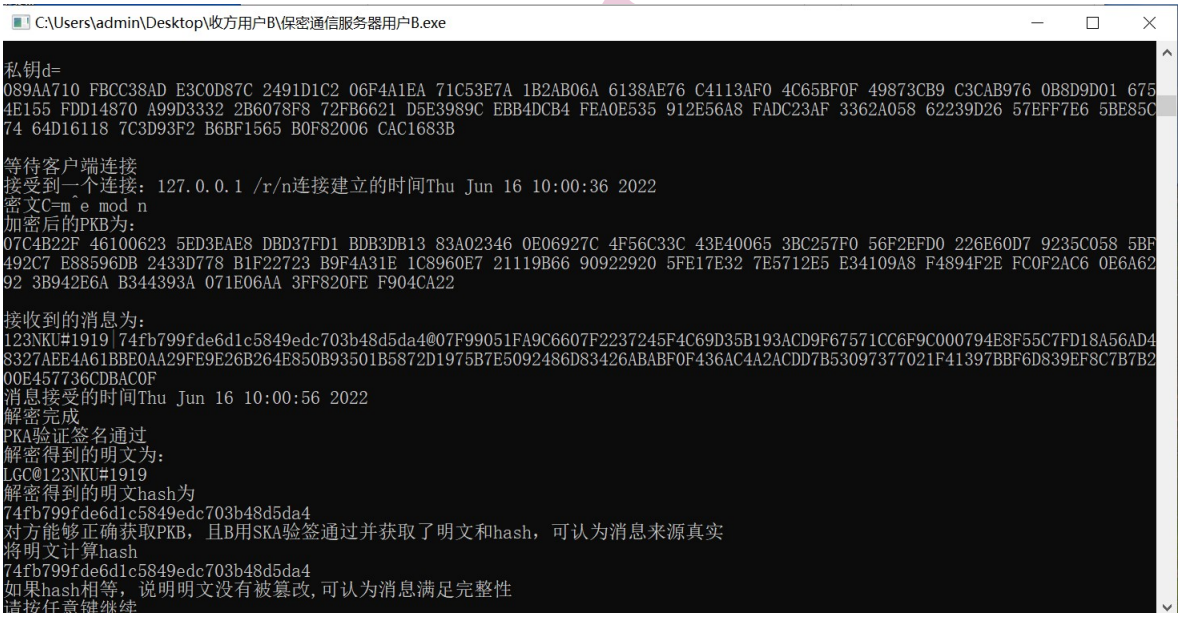

图 13: 收方 B

十一、 总结和收获

1. 引入 B 的公私钥对，并结合时间戳的信息，在实验要求的基础上，本次实验还实现了 TCP 的双向通信，并保证了消息的新鲜性，有效地防范了中间人攻击和重放攻击。

2. 每次进行会话时使用了不同的随机数种子，通信过程可使用新的公私钥对，可以有效解决密钥管理时容易泄露旧密钥的问题。

3. 对 RSA,MD5 的相关知识和应用有了更加深入的了解和认识，并且和 C/S 通信结合进行保密通信的协议设计和代码实现有了更加深入的体会和收获