



南开大学  
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

密码学实验

---

## 保密通信协议设计

---

学号：2013018

姓名：许健

年级：2020 级

专业：信息安全

2023 年 1 月 8 日

## 目录

|                  |    |
|------------------|----|
| 一、 实验内容          | 1  |
| 二、 实验要求          | 1  |
| 三、 保密通讯协议        | 1  |
| (一) 建立会话密钥       | 1  |
| (二) 使用共享密钥通讯     | 2  |
| (三) 协议安全性分析      | 2  |
| 四、 代码实现          | 2  |
| (一) A、B 建立连接     | 3  |
| (二) 交换 RSA 公钥    | 4  |
| (三) 建立会话密钥       | 4  |
| 1. 随机数生成         | 6  |
| 2. RSA 加解密       | 7  |
| (四) 保密通讯         | 7  |
| 1. 消息发送          | 7  |
| 2. 消息接收          | 8  |
| 3. AES-CBC 模式加解密 | 8  |
| (五) 更换 AES 密钥    | 9  |
| 五、 程序演示          | 10 |

## 一、 实验内容

设计一个保密通信的协议，具体内容为：利用 RSA 公钥密码算法，为双方分配一个 AES 算法的会话密钥，然后利用 AES 加密算法和分配的会话密钥，加解密传送的信息。

假设条件：假设通讯双方为 A 和 B，并假设发方拥有自己的 RSA 公钥  $PK_A$  和私钥  $SK_A$ ，同时收方拥有自己的 RSA 公钥  $PK_B$  和私钥  $SK_B$ ，同时收发双方已经通过某种方式知道了对方的公钥。

## 二、 实验要求

1. 作业需要先设计出保密通讯协议的过程和具体步骤；
2. 分别编写 A, B 两个用户端的程序，写清楚两个程序分别要完成的功能，并能够在两个程序间进行通讯；
3. 对 AES 加密的保密信息，要求采用 CBC 模式进行加密；

## 三、 保密通讯协议

### (一) 建立会话密钥

保密通讯协议设计的难点在于如何建立会话密钥，由于实验假设收发双方已经通过某种方式知道了对方的公钥，因此我们不必考虑主动攻击的情况。一个简单的思路如下：

1. A 向 B 发送  $ID_A$  表明自己的身份
2. 然后 B 发送用 A 的公钥  $ID_A$  加密的会话密钥  $K_s$
3. A 由  $D_{SK_A}[E_{PK_A}[K_s]]$  恢复会话密钥，只有 A 可以解读  $K_s$

在这种情况下仅有 A、B 知道此共享密钥，现在 A、B 双方可以通过此共享密钥进行保密通信。但是这种方式存在弊端，A 的身份  $ID_A$  通过明文传输，缺乏认证，并且 A、B 建立会话密钥的过程缺乏保密性，无法避免假冒情况。

因此保密通信协议需要在此基础上进行改进，消息传递过程中使用对方的公钥加密以及增加随机数认证机制，保障了通信过程的保密性和认证性。具体过程如下：

1. A 用 B 的公开钥加密 A 的身份  $ID_A$  和一个一次性随机数  $N_1$  后发往 B, 其中  $N_1$  用于唯一地标识这一业务。
2. B 用 A 的公开钥  $PK_A$  加密 A 的一次性随机数  $N_1$  和 B 新产生的一次性随机数  $N_2$  后发往 A。因为只有 B 能解读 (1) 中的加密消息，所以 B 发来的消息中  $N_1$  的存在可使 A 相信对方的确是 B。
3. A 用 B 的公钥  $PK_A$  对  $N_1$  加密后返回给 B, 以使 B 相信对方的确是 A。
4. A 选取一个会话密钥  $K_s$ ，然后将  $M = E_{PK_B}[E_{SK_A}[K_s]]$  发给 B, 其中用 B 的公开钥加密是为保证只有 B 能解读加密结果，用 A 的秘密钥加密是保证该加密结果只有 A 能发送。
5. B 以  $D_{PK_A}[D_{SK_B}[M]]$  恢复会话密钥。

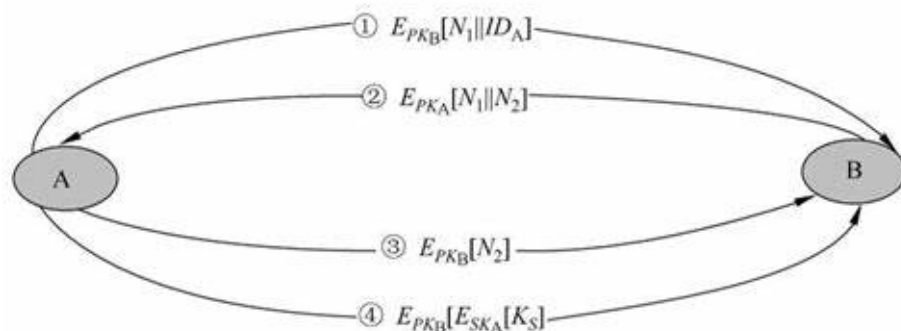


图 1: 保密通讯协议

## (二) 使用共享密钥通讯

A、B 共享会话密钥由两部分组成：第一部分是 AES 密钥，第二部分是 CBC 模式下 AES 的初始化向量 IV，该部分使用 AES 加密，需要收方获取 AES 密钥后，再进行一次 AES 解密。之后 A、B 双方的通信不再依赖 RSA 私公钥。

为了保证密钥的安全性，密钥不被泄露，A、B 通讯时，每隔一段时间/每发 n 条消息/每发 n 个字节进行一次 AES 密钥更换，更换时需要再次用到 RSA 公私钥，重复共享密钥建立过程。

## (三) 协议安全性分析

在使用 RSA 公私钥建立公钥时，RSA 安全性依赖于大整数分解，当使用 1024 位以上 RSA 时，安全性尚可以得到保障；使用 AES 密钥通讯时，AES 算法的安全性由它的密钥长度决定，我们可以采用 192 或者 256bit 长的密钥。通讯过程中，都是采用对方的公钥加密发送信息，保证只有对方可以解读加密结果，并且使用随机数 N 唯一标识，保障通讯的时效性以及提供认证性。当一个 AES 密钥使用一定次数后就会更换，只要确保 RSA 公钥私钥不泄露即可。

# 四、 代码实现

实验中用到的模块主要包括 BigInt 大整数类、RSA 公钥加密、AES 对称加密、Socket 通讯，其中 A、B 建立连接、交换共享密钥、保密通讯等均在 Socket 类中定义了方法实现，下面一一介绍。

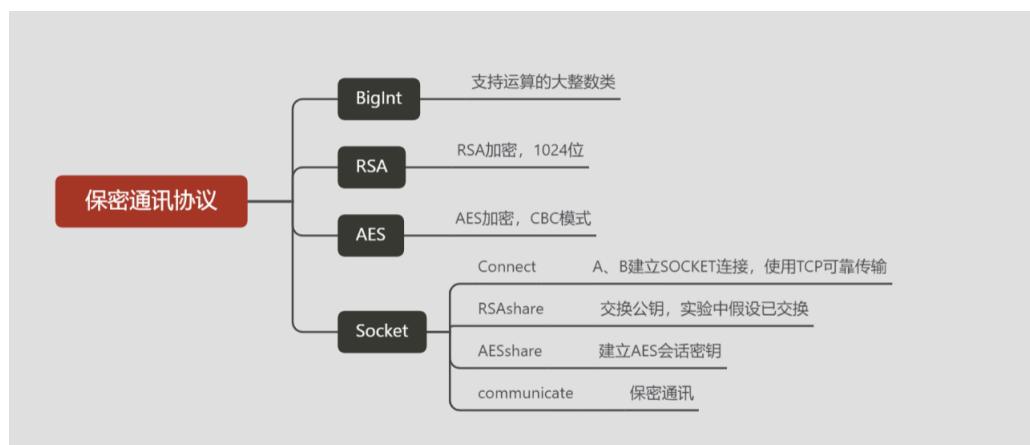


图 2: 主要模块

### (一) A、B 建立连接

A、B 建立 TCP 连接，保障可靠传输机制，保密通信主要是在应用层实现，通过加密确保信息不被利用，建立 TCP 连接属于计算机网络的知识，底层已经实现，我们直接使用库函数，流程图如图3所示。

建立 TCP 连接

```

1 //UserA: Client端
2 WSASStartup(MAKEWORD(2, 2), &wsaData);
3 ClientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
4 .....
5 //身份ID用于表示，这里使用字节序列标识
6 cout << "请输入身份ID:";
7 cin.getline(ID, 8);
8 cout << "等待连接..." << endl;
9 if (connect(ClientSocket, (SOCKADDR*)&ServerAddr, sizeof(ServerAddr)) < 0) {
10     cout << "连接失败" << endl;
11     return -1;
12 }
13 cout << "与UserB连接成功" << endl;
14
15 //UserB: Server端
16 WSASStartup(MAKEWORD(2, 2), &wsaData);
17 ServerSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
18 .....
19 bind(ServerSocket, (SOCKADDR*)&ServerAddr, sizeof(ServerAddr));
20 cout << "Start listening..." << endl;
21 int l = listen(ServerSocket, 1);
22 cout << "Starting accepting..." << endl;
23 ClientSocket = accept(ServerSocket, (SOCKADDR*)&ClientAddr, &ClientAddrLen);
24 cout << "与UserA建立连接" << endl;
25 .....

```

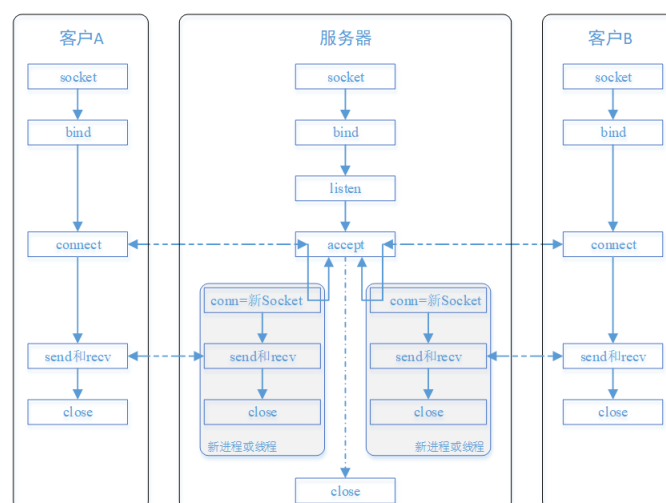


图 3: 建立 TCP 连接流程图

## (二) 交换 RSA 公钥

实验中已经假设双方互知对方公钥, 因此 `RSAShare()` 只完成简单的公钥  $(e, N)$  交换,

RSA 公钥交换

```

1 //RSA公钥交换
2 void RSAShare() {
3     char sendBuf[500];
4     char recvBuf[500];
5     string tmp;
6     //交换N
7     memset(sendBuf, 0, sizeof(sendBuf));
8     memcpy(sendBuf, BigIntToString(UserA.N).c_str(), BigIntToString(UserA.N).
9         size());
10    send(ClientSocket, sendBuf, sizeof(sendBuf), 0);
11    memset(recvBuf, 0, sizeof(recvBuf));
12    recv(ClientSocket, recvBuf, sizeof(recvBuf), 0);
13    tmp = recvBuf;
14    UserB.N = tmp;
15    //交换e
16    memset(sendBuf, 0, sizeof(sendBuf));
17    memcpy(sendBuf, BigIntToString(UserA.e).c_str(), BigIntToString(UserA.e).
18        size());
19    send(ClientSocket, sendBuf, sizeof(sendBuf), 0);
20    memset(recvBuf, 0, sizeof(recvBuf));
21    recv(ClientSocket, recvBuf, sizeof(recvBuf), 0);
22    tmp = recvBuf;
23    UserB.e = tmp;
24    //输出UserA的公钥
25    cout << "UserB.e: " << UserB.e << endl;
26    cout << "UserB.N: " << UserB.N << endl;
27 }

```

## (三) 建立会话密钥

`AESshare()` 函数用于建立 AES 共享会话密钥, 可以大致分为四个步骤:

1. A 发送身份 ID(用户输入的字符序列)、随机数  $N_1$  给 B, 使用 B 的公钥加密
2. B 发送随机数  $N_1$ 、 $N_2$  给 A, 使用 A 的公钥加密
3. A 发送  $N_2$  给 B, 使用 B 的公钥加密
4. A 创建 AES 共享密钥、初始化向量 IV, 使用 AES 加密 IV 打包成  $K_s$ , 发送给 B。A 使用 A 的私钥和 B 的公钥加密两次, 只有 B 才可以解读加密信息。

以上四步完成, 则 UserA、UserB 的共享密钥建立完毕。

---

```

1 //发送 ID 和随机数 n1
2 //产生随机数 n1

```

```

3  string n1 = createRandom();
4  //RSA 加密并发送
5  string tmp = RSA_Encrypt_SingleString(UserB, n1 + ID);
6  send(ClientSocket, tmp.c_str(), tmp.size(), 0);
7  cout << " 发送给 UserB 的随机数 n1: " << n1 << endl;
8  cout << " 发送给 UserB 的 ID: " << ID << endl;
9
10 //接收 n1 和 n2
11 int len = recv(ClientSocket, recvBuf, sizeof(recvBuf), 0);
12 for (int i = 0; i < len; i++) {
13     tmp += recvBuf[i];
14 }
15 //使用 A 的私钥解密
16 string text = RSA_Decrypt_SingleString(UserA, tmp);
17 string n_1 = text.substr(0, 16);
18 string n_2 = text.substr(16, 16);
19 cout << " 接收到 UserB 的随机数 n1: " << n_1 << endl;
20 cout << " 接收到 UserB 的随机数 n2: " << n_2 << endl;
21 //进行随机数比对
22 if (n1 == n_1) {
23     cout << " 经过对比, UserB 传过来的 n1 正是我们发送的随机数 n1" << endl;
24 } else {
25     cout << " 经过对比, UserB 传过来的 n1 与我们发送的随机数 n1 不一致" << endl;
26 }
27
28 //发送 n2
29 //使用 B 的公钥加密并发送
30 tmp = RSA_Encrypt_SingleString(UserB, n_2);
31 send(ClientSocket, tmp.c_str(), tmp.size(), 0);
32 cout << " 发送给 UserB 的随机数 n2: " << n_2 << endl;
33
34 //发送 AES 密钥、初始化 IV
35 //AES 加密 IV
36 AES_Encryption(AES_IV, cipher_IV, AES_Key2);
37 for (int i = 0; i < 16; i++) {
38     tmp = tmp + hex[(cipher_IV[i] >> 4)] + hex[(cipher_IV[i] & 0xf)];
39 }
40 //使用 A 的私钥加密
41 string tmp1 = RSA_Encrypt_SingleString_nocode(UserA, tmp, true);
42 //使用 B 的公钥加密
43 string tmp2 = RSA_Encrypt_SingleString_nocode(UserB, tmp1);
44 send(ClientSocket, tmp2.c_str(), tmp2.size(), 0);
45 //存储加密前的 IV
46 for (int i = 0; i < 16; i++) {

```

```

47     temp = temp + hex[(AES_IV[i] >> 4)] + hex[(AES_IV[i] & 0xf)];
48 }
49 cout << " 发送给 UserB 的 AES 密钥: " << tmp.substr(0, 32) << endl;
50 cout << " 发送给 UserB 的 AES 加密前的 IV: " << temp << endl;
51 cout << " 发送给 UserB 的 AES 加密后的 IV: " << tmp.substr(32, 32) << endl;

```

### 1. 随机数生成

A、B 通讯过程中需要用到随机数，关于随机数的生成我提供两种策略，第一种是使用 C++ 提供的 rand() 函数结合随机种子拼接成一个 64/128/256 位的随机数，用于一般小规模情况，但是安全性弱一点。另一种是使用 RSA 产生器生成伪随机序列，保存起来可以不停更换，本地实验我们采用了 64 位的随机数，也可以升级至更高位数，上述两种方法均有提供。

RSA 产生器使用幂形式的迭代公式：

$$X_{n+1} = (X_n)^d \bmod m, \quad n = 1, 2, \dots$$

其中  $(d, m)$  是参数， $X_0(0 \leq X_0 < m)$  是种子。

#### 随机数产生

```

1 // 创建一个n位随机数，确保首位不为1
2 string createRandom(int n = 64) {
3     n = n / 4;
4     unsigned char hex_table[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A',
5                                     'B', 'C', 'D', 'E', 'F' };
6     ostringstream oss;
7     int k;
8     do {
9         k = rand() % 16;
10    } while (k == 0);
11    oss << hex_table[k];
12    for (size_t i = 1; i < n; ++i) {
13        oss << hex_table[rand() % 16];
14    }
15    string str(oss.str());
16    return str;
17 }
18 // 伪随机数生成：RSA产生器
19 BigInt createRandomRSA() {
20     Rsa rsa;
21     init(rsa, 256);
22     // 获取伪随机数序列的X[index]号元素
23     // 这里使用rand()函数初始化index、X0
24     int index = rand();
25     BigInt X0 = rand();
26     BigInt X1;
27     for (int i = 0; i < index; i++) {
28         X1 = rsa.decryption(X0);

```



```

29     X0 = X1;
30 }
31 return X1;
32 }

```

## 2. RSA 加解密

为了使得 RSA 加密算法可以对字符串操作，我们需要将输入格式化，封装了函数留待后用，因为 RSA 的输入位数有限制，所以单次 RSA 加密字符串长度需要限制。同时对于 RSA 中不可用的字符还需要进行编码解码，完善接口，格式化输入输出等，在此给出函数定义。

```

1 //需要对字符编码，包含任意字符，编码从 00 开始增加
2 string RSA_Encrypt_SingleString(Rsa& rsa, string m, bool reverse = false);
3 string RSA_Decrypt_SingleString(Rsa& rsa, string c, bool reverse = false);
4 char code[38] = {
5     '0','1','2','3','4','5','6','7','8','9',
6     'A','B','C','D','E','F','G','H','I','J',
7     'K','L','M','N','O','P','Q','R','S','T',
8     'U','V','W','X','Y','Z',' ','*'
9 };
10
11 //不需要对字符编码，字符串中只包含 0-9、a-z、A-Z，十六进制字符串
12 //RSA 加密，reverse 控制公钥/私钥加密
13 string RSA_Encrypt_SingleString_nocode(Rsa& rsa, string m, bool reverse = false);
14 //RSA 解密，reverse 控制公钥/私钥解密
15 string RSA_Decrypt_SingleString_nocode(Rsa& rsa, string c, bool reverse = false);

```

## (四) 保密通讯

### 1. 消息发送

在主函数中实现消息发送，对于输入的字符串，需要使用 AES 算法加密再发送，加密模式采用 CBC 模式。

#### 消息发送

```

1 void communicate() {
2     HANDLE hThread_recv = CreateThread(NULL, NULL, &handler_Recv, LPVOID(
3         ClientSocket), 0, NULL);
4     string sendBuf;
5     while (1) {
6         getline(cin, sendBuf);
7         //AES-CBC Encryption
8         string tmp = AES_Encryption_CBC(sendBuf, AESKey, AESIV);
9         send(ClientSocket, tmp.c_str(), tmp.size(), 0);
10        cout << "发送给UserA/B: " << sendBuf << endl;
11    }
12 }

```

```
11 }
```

## 2. 消息接收

创建一个单独的线程，对于接收到的消息进行解密再输出，解密依然采用 AES 的 CBC 模式。

### 消息接收线程

```
1 DWORD WINAPI handler_Recv(LPVOID lparam) {
2     SOCKET recvSocket = (SOCKET)(LPVOID)lparam;
3     char recvBuf[500];
4     while (1) {
5         memset(recvBuf, 0, sizeof(recvBuf));
6         int len = recv(recvSocket, recvBuf, sizeof(recvBuf), 0);
7         string tmp = recvBuf;
8         //AES-CBC Decryption
9         string result = AES_Decryption_CBC(tmp, AESKey, AESIV);
10        cout << "UserB发送消息: " << result << endl;
11    }
12    closesocket(recvSocket);
13    return 0;
14 }
```

## 3. AES-CBC 模式加解密

CBC 模式的加密分如下步骤：

1. 首先将数据按照 16 个字节一组进行分组得到  $D_1D_2\cdots D_n$ （若数据不是 16 的整数倍，用指定的 PADDING 数据补位）
2. 第一组数据  $D_1$  与初始化向量 IV 异或后的结果进行 AES 加密得到第一组密文  $C_1$ （初始化向量 I 为全零）
3. 第二组数据  $D_2$  与第一组的加密结果  $C_1$  异或以后的结果进行 DES 加密，得到第二组密文  $C_2$ 。之后的数据以此类推，得到  $C_n$
4. 按顺序连为  $C_1C_2C_3\cdots C_n$  即为加密结果。解密过程反向执行即可。

```
1 for (int i = 0; i < size / 16; i++) {
2     memcpy(plain, m.substr(i * 16, 16).c_str(), 16);
3     for (int j = 0; j < 16; j++)
4         plain[j] ^= iv[j];
5     memset(cipher, 0, 16);
6     AES_Encryption(plain, cipher, Key);
7     for (int j = 0; j < 16; j++)
8         result += cipher[j];
9     memcpy(iv, cipher, 16);
10 }
```

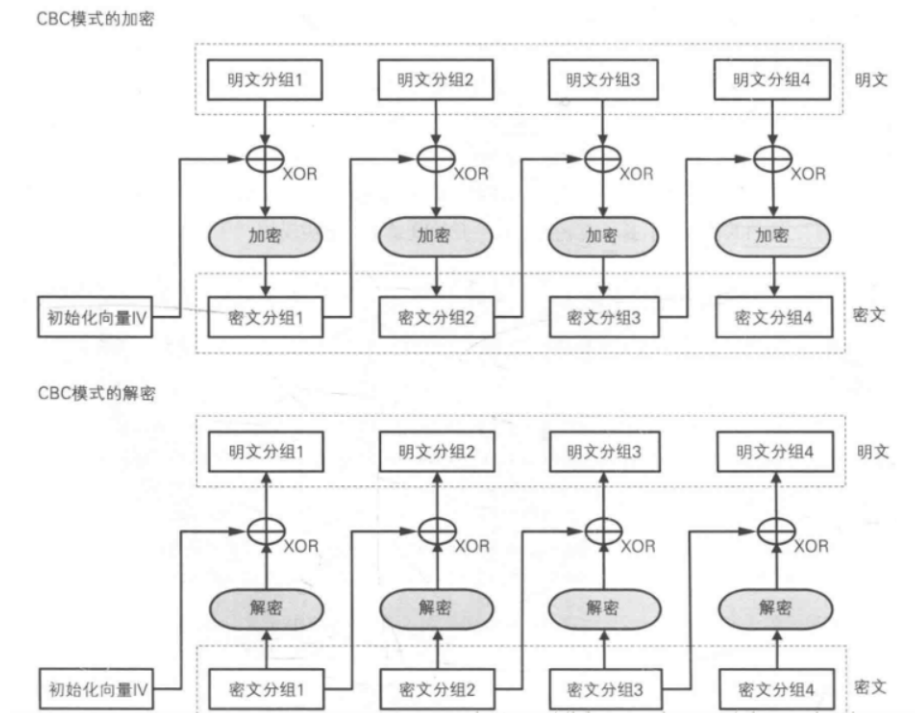


图 4: CBC 模式

### (五) 更换 AES 密钥

为了更换 AES 密钥, 本次实验定义了一个线程函数, 该函数定期检查是否到了更改用于加密的 AES 密钥的时间。它通过使用更改标志和开始时间来执行此操作, 只要该标志设置为真, 该时间就会重置。该线程还检查自开始时间以来经过的时间是否大于定义的最长时间 `MAX_TIME`。如果是, 它会将更改标志设置为真并重置开始时间。

#### 更换 AES 密钥

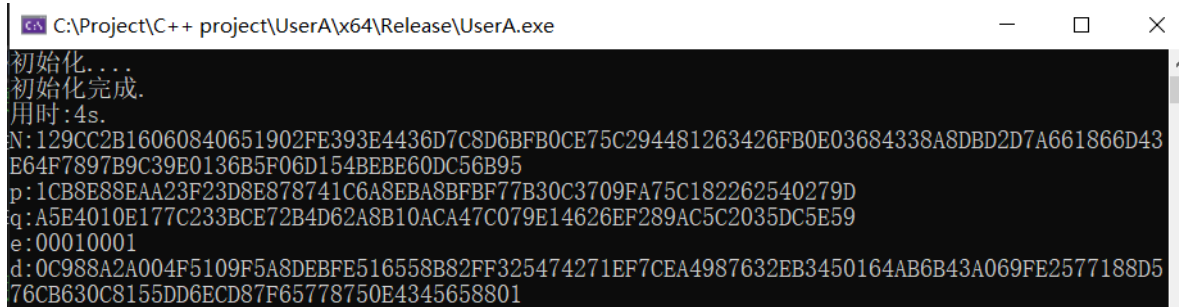
```

1  bool change = false;
2  #define MAX_TIME 5000000
3  //timer线程函数, 隔一段时间更换AES密钥
4  //更换指标也可以替换成A、B发送消息次数
5  DWORD WINAPI timer(LPVOID lparam) {
6      clock_t start = clock();
7      while (true) {
8          if (change) {//是否需要更换AES密钥
9              start = clock();
10             change = false;
11         }
12         if (clock() - start > MAX_TIME) {//密钥使用超过一定时间, 不安全需要更换
13             change = true;
14             start = clock();
15         }
16     }
17 }

```

## 五、 程序演示

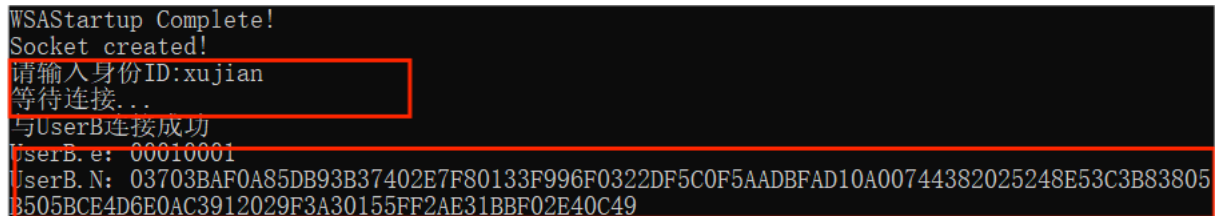
程序开始运行，需要初始化 RSA 参数。



```
C:\Project\C++ project\UserA\x64\Release\UserA.exe
初始化...
初始化完成.
用时:4s.
N:129CC2B16060840651902FE393E4436D7C8D6BFB0CE75C294481263426FB0E03684338A8DBD2D7A661866D43
E64F7897B9C39E0136B5F06D154BEBE60DC56B95
p:1CB8E88EAA23F23D8E878741C6A8EBA8BFBF77B30C3709FA75C182262540279D
q:A5E4010E177C233BCE72B4D62A8B10ACA47C079E14626EF289AC5C2035DC5E59
e:00010001
d:0C988A2A004F5109F5A8DEBFE516558B82FF325474271EF7CEA4987632EB3450164AB6B43A069FE2577188D5
76CB630C8155DD6ECD87F65778750E4345658801
```

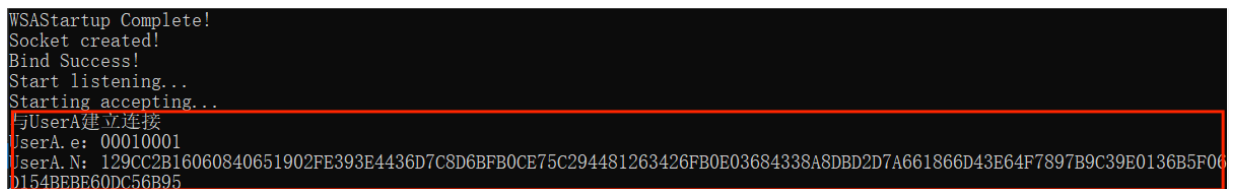
图 5: 初始化 RSA 参数

之后 UserA(Client) 与 UserB(Server) 要建立连接，并交换 RSA 参数。



```
WSAStartup Complete!
Socket created!
请输入身份ID:xujian
等待连接...
与UserB连接成功
UserB. e: 00010001
UserB. N: 03703BAF0A85DB93B37402E7F80133F996F0322DF5C0F5AADBFAAD10A00744382025248E53C3B83805
B505BCE4D6E0AC3912029F3A30155FF2AE31BBF02E40C49
```

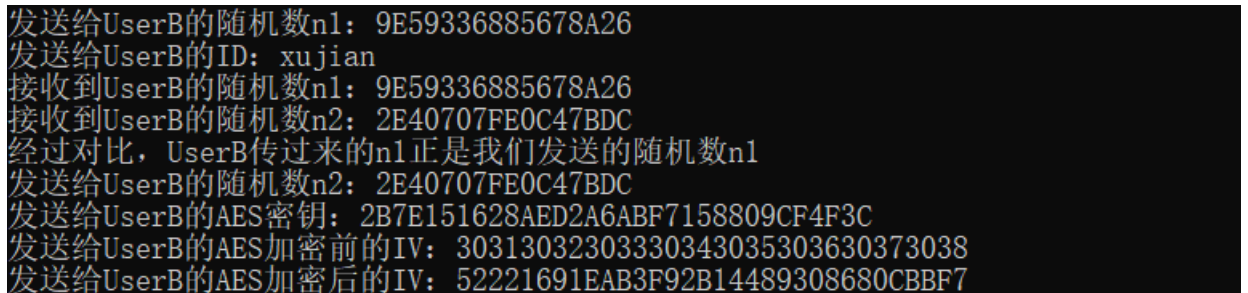
图 6: UserA-Client



```
WSAStartup Complete!
Socket created!
Bind Success!
Start listening...
Starting accepting...
与UserA建立连接
UserA. e: 00010001
UserA. N: 129CC2B16060840651902FE393E4436D7C8D6BFB0CE75C294481263426FB0E03684338A8DBD2D7A661866D43E64F7897B9C39E0136B5F06
D154BEBE60DC56B95
```

图 7: UserB-Server

A、B 建立共享密钥是最重要的一步，日志如图8、9所示，RSA 加解密没有问题，A、B 互相发送的随机数都得到确认，A 的身份 ID 在 B 方成功收到。最后 A 将 AES 密钥、IV 发送给 B，B 收到后解密 IV，与 A 加密前的一致。



```
发送给UserB的随机数n1: 9E59336885678A26
发送给UserB的ID: xujian
接收到UserB的随机数n1: 9E59336885678A26
接收到UserB的随机数n2: 2E40707FE0C47BDC
经过对比，UserB传过来的n1正是我们发送的随机数n1
发送给UserB的随机数n2: 2E40707FE0C47BDC
发送给UserB的AES密钥: 2B7E151628AED2A6ABF7158809CF4F3C
发送给UserB的AES加密前的IV: 30313032303330343035303630373038
发送给UserB的AES加密后的IV: 52221691EAB3F92B14489308680CBBF7
```

图 8: UserA 建立共享密钥日志

```

接收到UserA的随机数n1: 9E59336885678A26
接收到UserA的ID: XUJIAN
发送给UserA的随机数n1: 9E59336885678A26
发送给UserA的随机数n2: 2E40707FE0C47BDC
解密后的数据
02140400070007151400120407111312
接收到UserA的随机数n2: 2E40707FE0C47BDC
经过对比, UserA传过来的n2正是我们发送的随机数n2
UserA发送的AES密钥: 2B7E151628AED2A6ABF7158809CF4F3C
UserA发送的AES加密后的IV: 52221691EAB3F92B14489308680CBBF7
UserA发送的AES解密后的IV: 30313032303330343035303630373038

```

图 9: UserB 建立共享密钥日志

查看 A、B 通讯日志, 无论是 A 发送给 B 的加密数据, 还是 B 发送给的加密数据, 都被正常解密输出, 如图10、11所示。

```

xujian wants to test the program
发送给UserB: xujian wants to test the program
UserB发送消息: the program run successfully
ok
发送给UserB: ok
lalala lalala
发送给UserB: lalala lalala
UserB发送消息: test over
hascuch chus cusc us
发送给UserB: hascuch chus cusc us
hauschsacu hcuas c
发送给UserB: hauschsacu hcuas c
UserB发送消息: xhuashx aux asxu

```

图 10: UserA 保密通讯日志

```

UserA发送消息: xujian wants to test the program
the program run successfully
发送给UserA: the program run successfully
UserA发送消息: ok
UserA发送消息: lalala lalala
test over
发送给UserA: test over
UserA发送消息: hascuch chus cusc us
UserA发送消息: hauschsacu hcuas c
xhuashx aux asxu
发送给UserA: xhuashx aux asxu

```

图 11: UserB 保密通讯日志

使用 wireshark 捕获通讯的数据包, Data 部分使用的是密文传输, 这样就确保消息被截获后仍然不泄露, 保障了通信的保密性。