



南开大学
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

密码学大作业报告

基于身份认证和多模式的 AES 保密通信协议

年级：2020 级

专业：信息安全

指导教师：古力

2023 年 1 月 12 日

摘要

本次密码学大作业设计了一个高仿真的保密通信协议，全过程模拟真实用户访问状态，可以加密传输任意类型任意大小文件，自主选则 CBC、CFB 加密模式，采用具有保密性和认证性的密钥分配协议，利用 RSA 公钥密码算法，进行身份认证后为双方分配一个 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，加解密传送的文件并保存在用户指定路径。

本次大作业主要分为三个大模块：通信连接模块、密钥分配模块和 AES 加解密模块，后续协议设计、代码实现也都会围绕这三个模块为主线进行详细讲解。

关键字：保密通信 AES RSA 多模式 身份认证

目录

一、 功能概述	1
(一) 基本功能	1
(二) 附加功能	1
二、 协议设计	1
(一) 通信连接协议	1
(二) 密钥分配协议	1
1. RSA 公钥私钥生成	2
2. 身份认证	2
3. AES 会话密钥加密与分配	2
(三) AES 加解密协议	3
1. CBC 模式	3
2. CFB 模式	4
3. AES 加密	5
4. AES 解密	6
三、 代码实现	6
(一) 通信连接模块	6
(二) 密钥分配模块	8
1. RSA 公钥私钥生成	8
2. 身份认证	13
3. AES 会话密钥加密与分配	15
(三) AES 加解密	17
1. 文件读取	17
2. AES 加密	17
3. CBC 模式	22
4. CFB 模式	26
四、 遇到的困难及解决	28
(一) 类型转换	28
(二) Sleep 相关问题	29
(三) 个人粗心	30
五、 可执行文件运行说明	30

一、 功能概述

(一) 基本功能

- 支持任意类型、任意大小文件加密传输。
- 利用流式 socket 实现网络通信功能。
- 具有保密性和认证性的 AES 会话密钥分配。
- RSA 加密与解密 AES 会话密钥
- AES 算法加密与解密传输文件

(二) 附加功能

1. 对 AES 加密的保密信息，用户可以自主选则 CBC、CFB 两种加密模式。
2. RSA 公钥私钥、用户身份 ID、AES 会话密钥等均与用户进行交互，可选则自主输入或随机生成。
3. 全过程高仿真模拟用户访问状态，考虑多种输入状态与异常情况。
4. 添加身份认证模块，防止主动攻击与被动攻击。

二、 协议设计

(一) 通信连接协议

发送端创建套接字 Socket 并绑定到一个特定的传输层服务，IP 地址类型设定为 AF_INET (IPV-4 32 位)，服务类型为流式 (SOCK_STREAM)，Protocol (协议) 设置为 0 使系统自动选则。通过 bind 绑定本地地址到指定的 Socket 上。绑定完成后开启 listen 使 socket 进入监听状态，监听接收端连接。通过 accept() 接受接收端连接请求，成功建立连接。当发送端结束传输时，发送 exit 给接收端并关闭 socket。

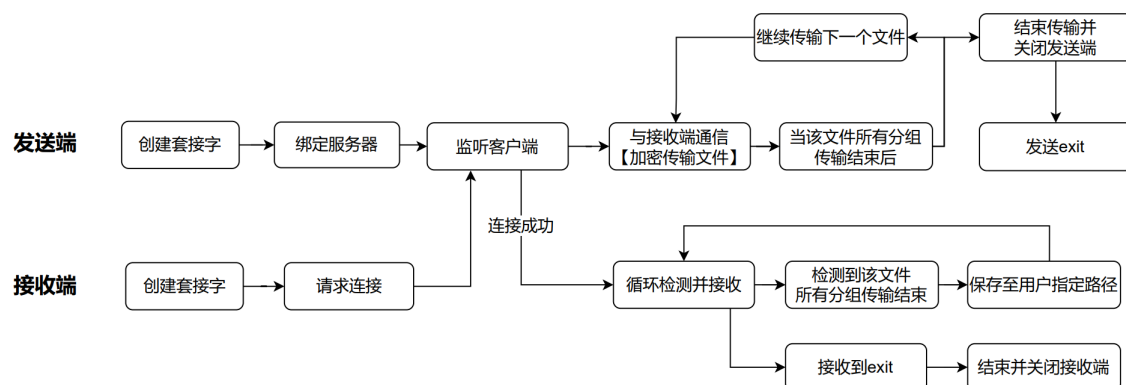


图 1: 通信协议

(二) 密钥分配协议

采用具有保密性和认证性的密钥分配协议，这样既可防止被动攻击，又可防止主动攻击。

1. RSA 公钥私钥生成

虽然大作业说明文档中指出“收发双方已经通过某种方式知道了双方的公钥”，但为了更加模拟真实用户场景，因此添加了 RSA 公钥私钥生成模块，用户可以自主选则随机随机生成或自主输入 RSA 公钥、私钥，如果用户选则随机生成则调用大素数模块，生成 512 位大素数 p 和 q ，计算 $n=p \cdot q$ ，生成与 n 互素的随机数 e 作为公钥，利用扩展欧几里得方法求出 e 模 (n) 的乘法逆元 d 作为私钥。

下图为大素数生成流程图：

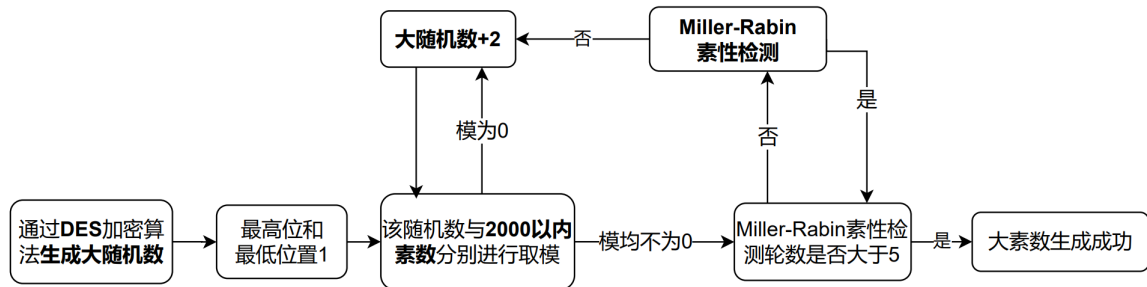


图 2: 生成大素数流程图

下图为生成 RSA 公钥和私钥流程图：

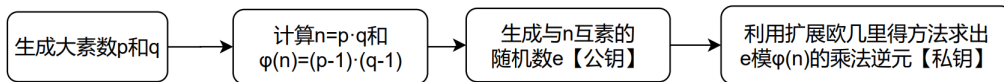


图 3: 生成 RSA 公钥私钥流程图

2. 身份认证

- 发送端用户输入不超过 6 位的身份 ID 并生成一次性随机数 $N1$ ，用接收端的 RSA 公钥加密 ID 和 $N1$ 后发往接收端，其中 $N1$ 用于唯一标识本次文件加密传输。
- 接收端用私钥解密后，随机生成一次性随机数 $N2$ ，并用发送端的公钥加密 $N1$ 和 $N2$ 并发送发送端。
- 发送端用私钥解密后对比收到的 $N1$ 与初始生成的一次性随机数 $N1$ ，如果相等则接收端身份认证通过。此时接收端再用公钥加密收到的 $N2$ 并发送给接收端。
- 接收端用私钥解密后对比 $N2$ 是否与初始生成的一次性随机数 $N2$ 相同，如果相等则发送端身份认证通过。

3. AES 会话密钥加密与分配

(1) 发送端：

- 提示发送端用户输入 128 位密钥（十六进制）
- 使用 RSA 私钥加密，再使用接收端 RSA 公钥加密并发送给接收端
- 此时 largeInt 大整数类的成员变量 `data[]` 存储着 AES 会话密钥加密结果，使用 `to_str()` 函数转换为 `char*` 类型，调用 `send()` 函数发给接收端。

(2) 接收端：

- recv 接收到加密的 AES 会话密钥 (char* 类型)
- 使用 to_arr() 函数，将 char* 类型转换成数组并保存到 largeInt 大整数类的 data[] 成员变量，此时已经满足 RSA 解密所需数据结构类型。
- 使用 RSA 私钥解密，再使用发送端 RSA 公钥解密

(三) AES 加解密协议

1. CBC 模式

(1) CBC 模式加密

- 设置一个明文分组为 128 位，计算循环轮数 N。
- 初始化异或向量 IV。
- 第一个明文分组与初始化向量 IV 异或后的结果进行 AES 加密得到第一组密文 C1，更新异或向量为 C1，发送 C1 至接收端。
- 第二个明文分组与与异或向量 IV 异或后的结果进行 AES 加密得到第二组密文 C2，更新异或向量为 C2，发送 C2 至接收端。
- 之后的明文分组以此类推，循环 N 轮得到 Cn 并发送至接收端。
- 发送“finish”至接收端

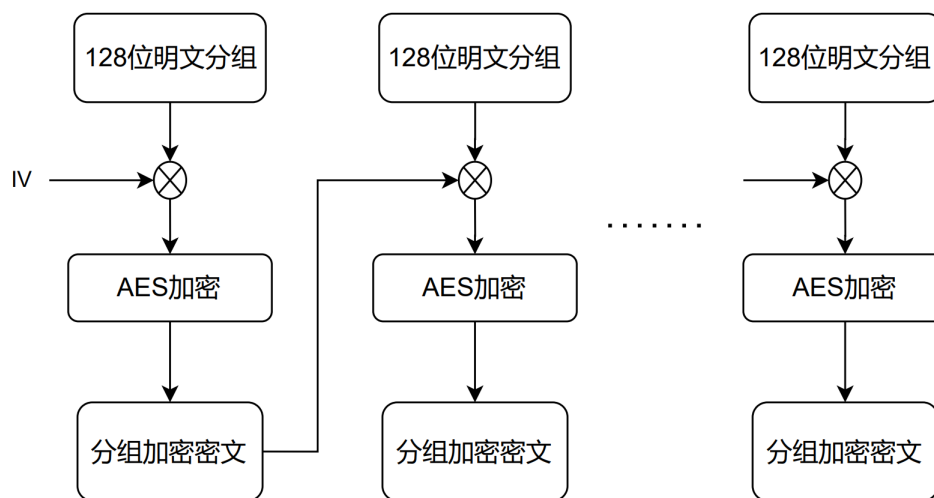


图 4: CBC 模式【加密】

(2) CBC 模式解密

- 接收到第一个密文分组。
- 初始化异或向量 IV（与加密时相同）。

- 第一个密文分组进行 AES 解密后与初始化向量 IV 异或得到第一个明文分组 D1，更新异或向量为 D1。
- 第二个密文分组进行 AES 解密后与初始化向量 IV 异或得到第二个明文分组 D2，更新异或向量为 D2。
- 直到接收到“finish”，此加密文件传输结束。
- 拼接 D1D2...Dn 即为完整明文，保存至用户指定路径。

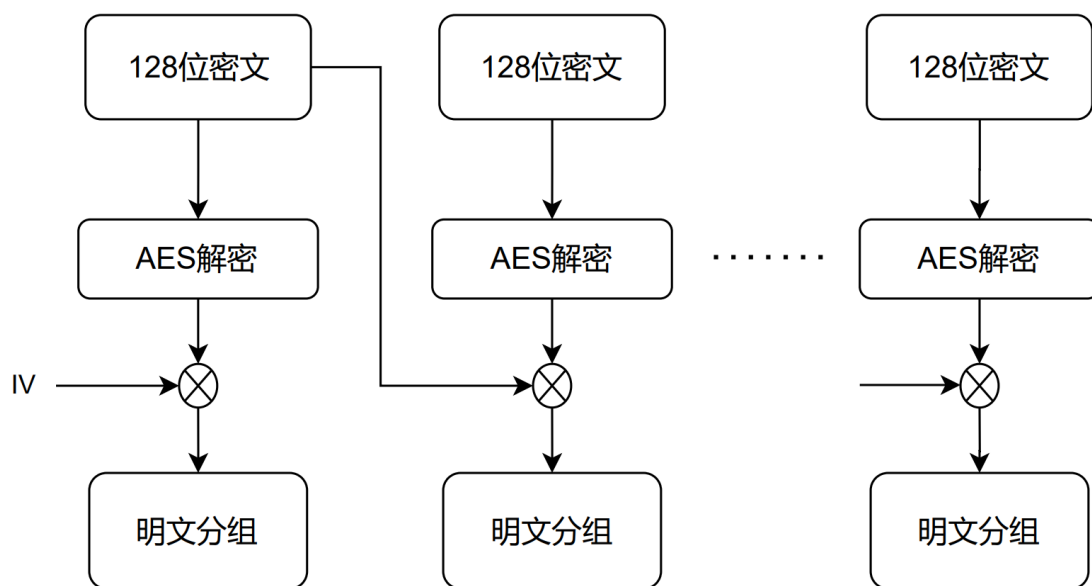


图 5: CBC 模式【解密】

2. CFB 模式

(1) CFB 模式加密

- 设置一个明文分组为 128 位，计算循环轮数 N。
- 初始化向量 IV。
- 第一个明文分组时，初始化向量 IV 通过密钥 key 加密后与明文进行异或得到密文分组 C1，更新向量 IV 为 C1
- 第二个明文分组时，向量 IV 通过密钥 key 加密后与明文进行异或得到密文分组 C2，更新向量 IV 为 C2
- 之后的明文分组以此类推，循环 N 轮得到 Cn 并发送至接收端。
- 发送“finish”至接收端

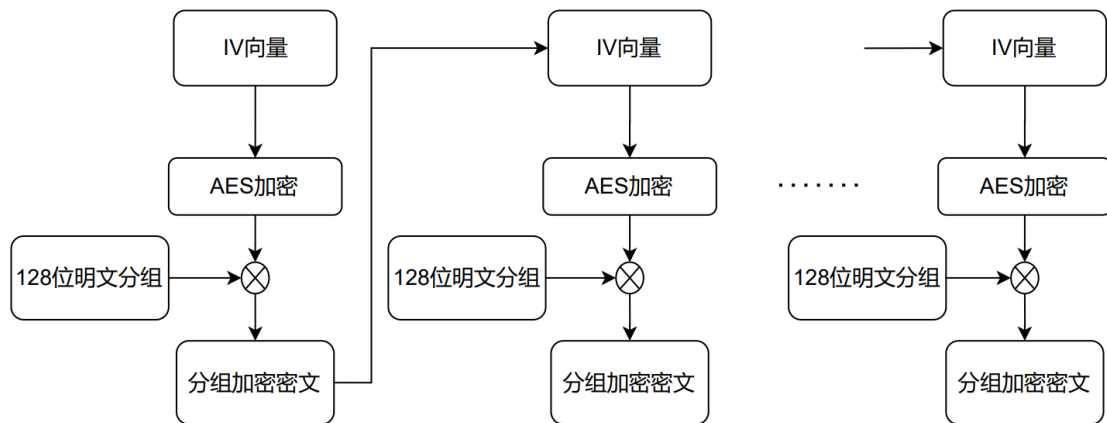


图 6: CFB 模式【加密】

(2) CFB 模式解密

- 接收到第一个密文分组。
- 初始化向量 IV（与加密时相同）。
- 第一个密文分组时，初始化向量 IV 通过密钥加密后，与密文分组 1 进行异或得到明文分组 D1，更新向量为 D1。
- 第二个密文分组时，向量 IV 通过密钥加密后，与密文分组 1 进行异或得到明文分组 D2，更新向量为 D2。
- 直到接收到“finish”，此加密文件传输结束。
- 拼接 D1D2...Dn 即为完整明文，保存至用户指定路径。

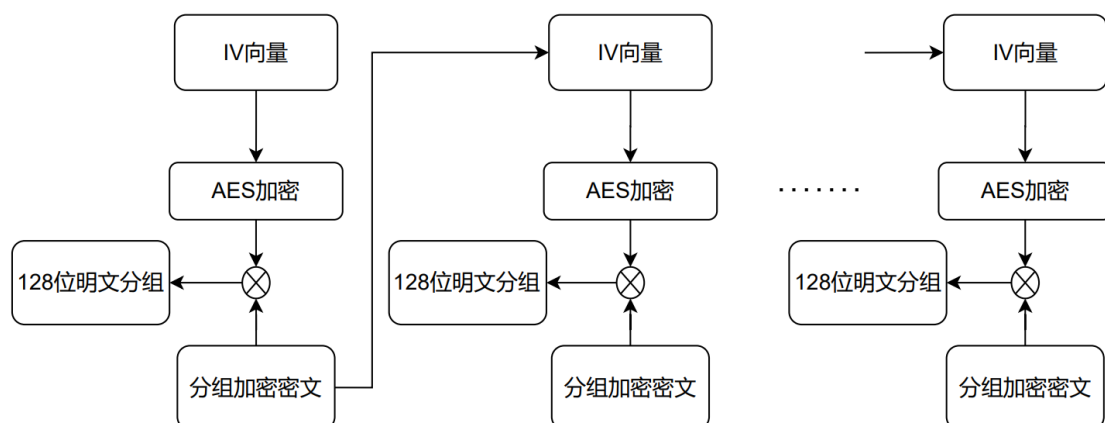


图 7: CBC 模式【解密】

3. AES 加密

- 初始密钥扩展
- 初始轮密钥加

- 字节代换 -> 行移位 -> 列混合 -> 密钥加，循环 10 轮，其中第 10 轮不进行列混合

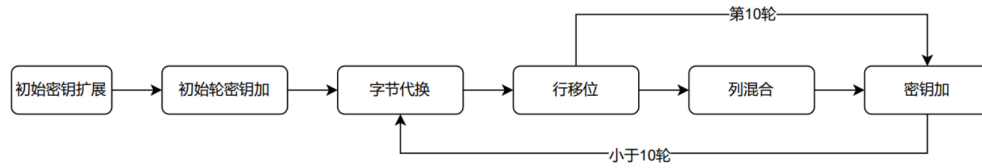


图 8: AES 加密

4. AES 解密

因为此处采取 AES 会话密钥为 128 位，因此解密过程仍为 10 轮，每一轮的操作是加密操作的逆操作。由于 AES 的 4 个轮操作都是可逆的，因此，解密操作的一轮就是顺序执行逆行移位、逆字节代换、轮密钥加和逆列混合。同加密操作类似，最后一轮不执行逆列混合，在第 1 轮解密之前，要执行 1 次密钥加操作。

三、 代码实现

本节对**通信连接**、**密钥分配**、**AES 加解密**三个模块结合代码实现具体阐述，因为通信模块不是本次实验重点因此仅作概括性描述。

(一) 通信连接模块

发送端与接收端通信模块差异性不大（已经在通信模块流程图中体现），因此仅以发送端通信模块为例进行阐述。

- 创建套接字 Socket 并绑定到一个特定的传输层服务，IP 地址类型设定为 AF_INET (IPv-4 32 位)，服务类型为流式 (SOCK_STREAM)，Protocol（协议）设置为 0 使系统自动选则。
- 通过 bind 绑定本地地址到指定的 Socket 上。
- 绑定完成后开启 listen 使 socket 进入监听状态，监听接收端连接。
- 通过 accept() 接受接收端连接请求，成功建立连接。
- 当发送端结束传输时，发送 exit 给接收端并关闭 socket。

通信连接

```

1 WSADATA wsaData;
2 SOCKET ServerSocket;
3 WSStartup(MAKEWORD(2, 2), &wsaData);
4 ServerSocket = socket(AF_INET, SOCK_STREAM, 0);
5 if (ServerSocket == INVALID_SOCKET){
6     cout << "套接字创建失败，请通过:" << WSAGetLastError() << "获取错误详情"
        << endl;
  
```

```

7     return -1;
8 }
9
10 SOCKADDR_IN ServerAddr;
11 USHORT uPort = 2023;
12 ServerAddr.sin_family = AF_INET;
13 ServerAddr.sin_port = htons(uPort);
14 //宏INADDR_ANY转换过来就是0.0.0.0，泛指本机的意思，也就是表示本机的所有IP
15 ServerAddr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
16 if (SOCKET_ERROR == bind(ServerSocket, (SOCKADDR*)&ServerAddr, sizeof(
    ServerAddr))){
17     cout << "bind 失败，请通过" << WSAGetLastError() << "获取错误详情" <<
        endl;
18     closesocket(ServerSocket);
19     return -1;
20 }
21 char serverName[10] = "User_A";
22 cout << "请等待对方连接..." << endl;
23
24 // 绑定完成后开始listen，使socket进入监听状态，监听接收端
25 if (listen(ServerSocket, 1) != 0){
26     cout << "监听失败，请通过" << WSAGetLastError() << "获取错误详情" << endl
        ;
27     closesocket(ServerSocket);
28     WSACleanup();
29     return -1;
30 }
31
32 SOCKET Conn_new_Socket;
33 SOCKADDR_IN Conn_new_Addr;
34 int iClientAddrLen = sizeof(Conn_new_Addr);
35 Conn_new_Socket = accept(ServerSocket, (SOCKADDR*)&Conn_new_Addr, &
    iClientAddrLen);
36 if (Conn_new_Socket == INVALID_SOCKET){
37     cout << "accept接受请求失败，请通过" << WSAGetLastError() << "获取错误详
        情" << endl;
38     closesocket(ServerSocket);
39     WSACleanup();
40     return -1;
41 }

```

如下图所示发送端和接收端成功进行通信：



图 9: 发送端通信连接



图 10: 接收端通信连接

(二) 密钥分配模块

1. RSA 公钥私钥生成

(1) 基于 DES 512 位随机数生成

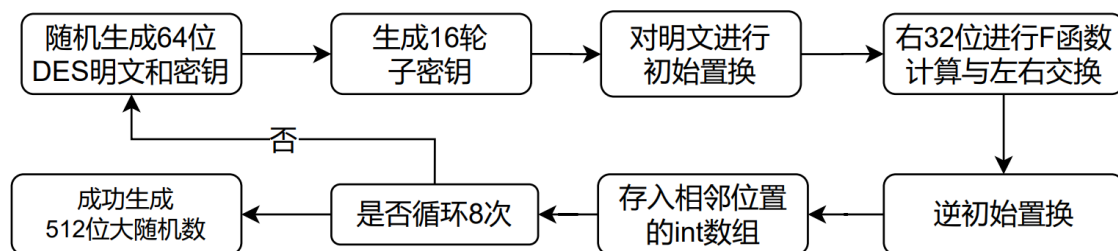


图 11: 基于 DES 大随机数生成

基于 DES 大随机数生成

```

1 void largeInt::Random()
2 {
3     for (int i = 0; i < 8; i++) {
4         data[i] = (rand() << 17) + (rand() << 2) + rand() % 4;
5         if ((i + 1) % 2 == 0) { //每生成两个，就进行一次des
6             bitset<32> m1(data[i]), m2(data[i - 1]);
7             string s = m1.to_string() + m2.to_string();
8             bitset<64> m(s); //明文
9             bitset<32> k1(data[i] + rand()), k2(data[i - 1] +
                rand());
10            s = k1.to_string() + k2.to_string();
11            bitset<64> key(s); //密钥
12            bitset<48> keys[16];
13
14            //生成16轮密钥

```

```

15     generate_keys(key, keys);
16     // 初始置换
17     init_map(m);
18     // 轮结构与左右交换
19     bitset<64> out = rounds(m, keys);
20     // 逆初始置换
21     bitset<64> tem;
22     for (int i = 0; i < 64; i++) {
23         tem[64 - 1 - i] = out[64 - 1 - (
24             ipReverseTable[i] - 1)];
25     }
26     out = tem;
27
28     // 将out赋给data[i] data[i-1]
29     data[i] = out.to_ulong() & 0xffffffff; // 低32位
30     data[i - 1] = (out.to_ulong() >> 32) & 0xffffffff; // 高32位
31 }
32 }
33 }

```

(2) 基于 Rabin-Miller 的大素数检测

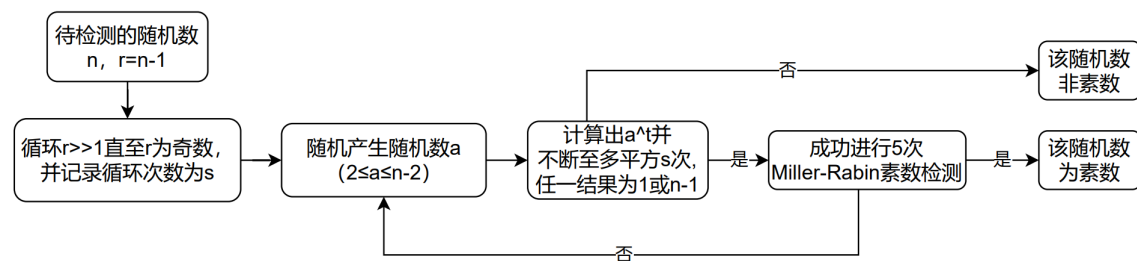


图 12: Rabin-Miller 检测流程图

- 利用 DES 加密算法生成一个 512 位随机数。
- 最高位和最低位置 1
- 将该随机数与 2000 以内所有素数逐个取模，若均不为 0 则执行步骤 4；若某个取模结果为 0，则该随机数 +2 重复执行该步骤。
- 利用 Rabin-Miller 检测法进行检验，一共检测 5 轮。如果 5 轮检验均通过，则成功生成大素数；如果有任意一轮没有通过，则该随机数 +2 并返回步骤 3。

Rabin-Miller 大素数检测

```

1 // 产生一个待测素数,保证此数为奇数,且不能被小于2000的素数整除
2 void Pre_Prime(largeInt& n)
3 {

```

```
4     int i = 0;
5     largeInt divisor;
6     // 2000以内素数
7     const int length = sizeof(prime) / sizeof(int);
8     n.Random();
9     while (!n.IsOdd())
10         n.Random();
11     while (i != length){
12         largeInt TWO(2);
13         n = n + TWO;
14         i = 0;
15         for (; i < length; i++){
16             if ((n % prime[i]) == 0)
17                 break;
18         }
19     }
20     // Rabin-Miller的大素数检测
21     bool RabinMiller(const largeInt& n)
22     {
23         largeInt r, a, y;
24         unsigned int s, j;
25         r = n - 1;
26         s = 0;
27         while (!r.IsOdd()){
28             s++;
29             r >> 1;
30         }
31
32         //随机产生一个小于N-1的检测数a
33         a.RandomSmall();
34         //y = a的r次幂模n
35         y = PowerMode(a, r, n);
36         //检测J=2至J<S轮
37         if ((!(y == 1)) && (!(y == (n - 1)))){
38             j = 1;
39             while ((j <= s - 1) && (!(y == (n - 1)))){
40                 y = (y * y) % n;
41                 if (y == 1)
42                     return false;
43                 j++;
44             }
45             if (!(y == (n - 1)))
46                 return false;
47         }
48         return true;
49     }
50 }
```

如下如所示生成大素数 P 和 Q

```

=====RSA公钥私钥生成=====
现在开始生成您本人的RSA公钥和私钥
请选择: 1 - 随机生成公钥和私钥  2 - 自行输入公钥和私钥 : 1
=====生成第一个素数p=====

正在生成第1个大奇数:
BBFDF222 27A256F4 AA62E631 8FAC02DA A0B89DDE B9E94EDC 36B6CA98 88C6E7D2
30EBDD80 5DF43145 79521D46 9FC0DE9F 667316E7 0F00FBCA 753A02D0 9D73B29D
正在进行第1轮Miller-Rabin测试: ===== 失败

正在生成第2个大奇数:
146DD6BF 29479C31 D1F8DD7 5AA16335 7CA7AB7F 93ED916E E0F8379B 3900EA4C
462BB44E 341D067D 9CF1DE62 B81B97F8 C96B96C5 4E56A54A 46A878BA D37F4DCB
正在进行第1轮Miller-Rabin测试: ===== 失败

正在生成第3个大奇数:
0B0D11EB CCDBEBE9 93782DF9 265795B3 D10D24BE 717F613A 7895449B DB3D69A8
1D04BF80 0B56BF2C 85764EB1 9F5B1757 9032B0FE 12D9D6A6 E6F5ECAB 43BF749
正在进行第1轮Miller-Rabin测试: ===== 失败

正在生成第4个大奇数:
3C1D0ED4 A321E452 236DD8ED CCADEC57 0A535BBE F31BC643 64C84EE8 4BFB4E99
B3C301D7 0871DB48 4A4EBB16 B573FAC8 66114512 024EB8DA C9AAF891 9C70573D
正在进行第1轮Miller-Rabin测试: ===== 失败

正在生成第5个大奇数:
DD3DDA2D 3044A2E3 172DA026 0EBA1CF5 C9D67FE7 2B9538D7 6F6CEB2B 8F120582
90C7D4FC B64E4468 BF9A0315 DFB41C0C 59F5F959 4FD00A1A C1F19059 28B67E6D
正在进行第1轮Miller-Rabin测试: ===== 通过
正在进行第2轮Miller-Rabin测试: ===== 通过
正在进行第3轮Miller-Rabin测试: ===== 通过
正在进行第4轮Miller-Rabin测试: ===== 通过
正在进行第5轮Miller-Rabin测试: ===== 通过

=====素数p生成成功=====
生成的素数p为:
DD3DDA2D 3044A2E3 172DA026 0EBA1CF5 C9D67FE7 2B9538D7 6F6CEB2B 8F120582
90C7D4FC B64E4468 BF9A0315 DFB41C0C 59F5F959 4FD00A1A C1F19059 28B67E6D

```

图 13: 发送端生成 RSA 私钥

```

正在生成第18个大奇数:
FADBB634 F8A99B9E F9746888 A5C0E94C AEBB70C7 932AE8C1 9D151C64 0EF40422
09E796B2 3BE61486 1243F2BA CAAEBA76 7EE305F0 F632CA64 46249F96 C6157F95
正在进行第1轮Miller-Rabin测试: ===== 失败

正在生成第19个大奇数:
B0BFFB8E B41473B1 7F5C7694 A8813F6F 36492887 A87FA899 4DD741BE F6FD2ADA
BF9EFADF D8223F31 274C08F5 52C8B6A6 67D9F36B 1F9F873B CF9E77C6 A81359F1
正在进行第1轮Miller-Rabin测试: ===== 通过
正在进行第2轮Miller-Rabin测试: ===== 通过
正在进行第3轮Miller-Rabin测试: ===== 通过
正在进行第4轮Miller-Rabin测试: ===== 通过
正在进行第5轮Miller-Rabin测试: ===== 通过

=====素数q生成成功=====
生成的素数q为:
B0BFFB8E B41473B1 7F5C7694 A8813F6F 36492887 A87FA899 4DD741BE F6FD2ADA
BF9EFADF D8223F31 274C08F5 52C8B6A6 67D9F36B 1F9F873B CF9E77C6 A81359F1

```

图 14: 发送端生成 RSA 私钥

(4) RSA 公钥私钥生成

利用基于 DES 和 Rabin-Miller 的大素数检测算法生成 512 位大素数 p 和 q, 计算 $n=p \cdot q$, 生成与 n 互素的随机数 e 作为公钥, 利用扩展欧几里得方法求出 e 模 (n) 的乘法逆元 d 作为私钥。

RSA 公钥私钥生成

```
1 srand((unsigned)time(NULL));
```

```

2 // 生成的素数p
3 largeInt P = GeneratePrime();
4 // 生成的素数q
5 largeInt Q = GeneratePrime();
6 // 计算n
7 largeInt n;
8 n = P * Q;
9 // 计算 (n)
10 largeInt fn = (P - 1) * (Q - 1);
11 // 公钥e
12 largeInt e;
13 //d为秘密钥，即e模fn的乘法逆元
14 largeInt d;
15 //y用于参与扩展欧几里得运算，存储t模e的乘法逆元
16 largeInt y;
17
18 // 计算产生互质的数
19 while (1)
20 {
21     //产生与fn互质的e
22     e.Random();
23     while (!(Gcd(e, fn) == 1))
24         e.Random();
25     //用扩展欧几里得算法试图求出e模t的乘法逆元
26     temp = ExtendedGcd(e, fn, d, y);
27     //e*d模t结果为1，说明d确实是e模fn的乘法逆元
28     temp = (e * d) % fn;
29     if (temp == 1)
30         break;
31     //否则重新生成e
32 }

```

如下图所示生成 RSA 公钥和私钥：

```

=====生成n=====
98C0708B D1C0B0CE B9747098 A7A642EC B1551C7C EFB1E3D7 8A8E027E B695D9D5 6062B938 D61F3BD8 F0BE0E7B 1C6ED256 7C4F4760 5443293B 2B9226FD 4F6F2378
25DAEB89 54E97382 57F88206 C5F3640B 4A8B0F79 F23E34F9 E5981C8C 64FADB87 7483E702 0A17AB45 78B15950 76C42AA5 6DF87EC9 96EF2223 451E1EE7 AFD7E99D

=====生成公钥e=====
13D4AA47 12CB4A3B 90551709 1273F10F 650D3A03 08773122 B3DBDD35 77B9C7D1
0FB6C61D B20001E5 BC9368FE 275379D2 23653485 482CC19E C5352348 BB847437

=====生成私钥d=====
460E0BC6 B608677E 3118F9B7 3E9413C8 5AADD8AF 10ABE715 3460DB4A 1F274DEB 912160F9 F0C11838 F24CDD09 8C44B352 DAFD0FDC 16CB6FE5 2C4D8CAC 145D331A
ADD26968 9E952A12 67579340 80EA2866 EF44B603 E6B8AC10 EDF39B42 E54C3784 B3AB6709 69A2CAEA 297F9AF1 541B7B66 03E71B2D 7E5E3CB9 BCBAD6F 6BB8EAC7

```

图 15: 发送端生成 RSA 公钥私钥

```

=====生成n=====
10CE10DA 334E0AC9 4561E267 0F035A4F 89968250 C23AFBA5 EC4CEEA2 FF63D0E4 66893E2F EC6C196C D8DF721 53269446 7F0DA12B 7BEC90E0 9CFEE560 8BE2B892
0C8C6BAC 33017633 C414E49C 7E5FF726 9AC72EBC FA18893F 64080EES 7AAD9103 69A85456 31F83872 1321BDCE 2A1069E6 DA68762E 7A294B79 2B91F967 0F26E28D

=====生成公钥e=====
60E5A870 AE09ABD8 F7767713 8A3EC200 58973577 C3B56B36 F0F49CEC 35D3F4E4
C1D29485 ACD457E3 8DA71FC9 0982C2F7 75206DB1 AC6FFFD6 E1D2ED74 37B76BFD

=====生成私钥d=====
057C04AA 2AC9E467 B6E4FE5A C49C9492 8799583F D20166F1 03D21400 E7BA8096 5D193FDC 80EB6D1E 6BA5E243 3140EED0 1E294B9F 39E1760C 35F28CEE 26650A0F
3942D7BF B573BA92 299EE12D 71A7EB2A ED2AD4A1 D3FD4A99 5579B789 8A2151D5 D78A7470 CAAC4DB4 9FA807D3 79515DA1 1C779966 540C5ACD B7322DF1 18D30815

```

图 16: 接收端生成 RSA 公钥私钥

2. 身份认证

系统提示发送端用户输入不超过 6 位的身份 ID 并生成一次性随机数 N1，用接收端的 RSA 公钥加密 ID 和 N1 后发往接收端，接收端用私钥解密后，随机生成一次性随机数 N2，并用发送端的公钥加密 N1 和 N2 并发送发送端。发送端用私钥解密后对比收到的 N1 与初始生成的一次性随机数 N1，如果相等则提示用户接收端身份认证通过，可以加密发送 AES 会话密钥。此时接收端将收到后解密的 N2 用接收端公钥加密并发给接收端。接收端用私钥解密后对比 N2 是否与初始生成的一次性随机数 N2 相同，如果相等则系统提示用户发送端身份认证通过。

为了避免冗余，此处仅展示发送端身份认证相关代码：

身份认证

```

1  cout << "请输入您的身份ID(不超过6位): ";
2  char id[6];
3  cin >> id;
4  srand((int)time(0));
5  int N1 = rand() % 1000;
6  char N1_str[6];
7  snprintf(N1_str, sizeof(N1_str), "%d", N1);
8  cout << "自动生成一次性随机数IDa: " << N1_str << endl;
9  cout << "发送A公钥加密的一次性随机数N1和身份IDa" << endl;
10 largeInt ID, ID2;
11 arr(ID.data, id)
12 ID2 = Encrypt(key_largeInt, ID, n);
13 ID2.to_str(N1_str)
14 if (SOCKET_ERROR == send(Conn_new_Socket, ID, sizeof(ID), 0)){
15     closesocket(Conn_new_Socket);
16     closesocket(ServerSocket);
17     WSACleanup();
18     return -1;
19 }
20 largeInt N1_, N1__;
21 to_arr(N1_.data, N1_str)
22 N1__ = Encrypt(key_largeInt, N1_, n);
23 N1__.to_str(N1_str)
24 if (SOCKET_ERROR == send(Conn_new_Socket, N1_str, sizeof(N1_str), 0)){
25     closesocket(Conn_new_Socket);
26     closesocket(ServerSocket);
27     WSACleanup();
28     return -1;
29 }

```



```

30 char N1_recv[6] = { 0 };
31 char N2[6] = { 0 };
32 if (SOCKET_ERROR == recv(Conn_new_Socket, N1_recv, sizeof(N1_recv), 0)){
33     closesocket(Conn_new_Socket);
34     closesocket(ServerSocket);
35     WSACleanup();
36     return -1;
37 }
38 if (SOCKET_ERROR == recv(Conn_new_Socket, N2, sizeof(N2), 0)){
39     closesocket(Conn_new_Socket);
40     closesocket(ServerSocket);
41     WSACleanup();
42     return -1;
43 }
44
45 to_arr(N1_.data, N1_recv)
46 N1__ = Decode(key_largeInt, N1_, n);
47 N1_.to_str(N1_recv)
48 arr(ID.data, N2)
49 ID2 = Encrypt(key_largeInt, ID, n);
50 ID2.to_str(N2)
51 if (strcmp(N1_str, N1_recv) == 0)
52     cout << "B传输回的一次性随机数N1经解密与初始生成的N1相同, B身份验证通过"
53     << endl;
54 cout << "使用B的RSA公钥加密收到的一次性随机数N2并再次发送给B" << endl << endl
55 ;
56 if (SOCKET_ERROR == send(Conn_new_Socket, N2, sizeof(N2), 0)){
57     closesocket(Conn_new_Socket);
58     closesocket(ServerSocket);
59     WSACleanup();
60     return -1;
61 }
62 }

```

如下图所示发送端和接收端成功进行身份认证：

```

=====身份认证+AES会话密钥分配【保密性+认证性】=====
请输入您的身份ID(不超过6位): nie
自动生成一次性随机数IDa: 323
发送A公钥加密的一次性随机数N1和身份IDa
B传输回的一次性随机数N1经解密与初始生成的N1相同, B身份验证通过
使用B的RSA公钥加密收到的一次性随机数N2并再次发送给B

```

图 17: 发送端生成 RSA 公钥私钥

```

=====身份认证+AES会话密钥接收【保密性+认证性】=====
接收到【解密后】对方ID为: nie      一次性随机数为: 323
自动生成一次性随机数IDa: 568
发送A公钥加密的一次性随机数N1和N2
A传输回的一次性随机数N2经解密与初始生成的N2相同, A身份验证通过

```

图 18: 接收端生成 RSA 公钥私钥

3. AES 会话密钥加密与分配

(1) 发送端

系统提示发送端用户输入 128 位密钥（十六进制），输入的每 8 个 16 进制数（32 位）保存在 largeInt 大整数类的成员变量 int data[] 数组中的一个元素，使用 RSA 私钥加密，再使用接收端 RSA 公钥加密并发送给接收端，使用 to_str() 函数将 data[] 数组转换为 char* 类型，最后调用 send() 函数发给接收端。

发送端 AES 会话密钥加密与分配

```

1  cout << "请输入AES 128位密钥(十六进制)" << endl;
2  int txt[4][4];
3  int key[4][4];
4  largeInt key_largeInt;
5  for (int i = 0; i < 4; i++)
6      for (int j = 0; j < 4; j++){
7          cin >> (hex) >> key[j][i];
8          key_largeInt.data[i] += key[j][i];
9          if (j < 3)
10             key_largeInt.data[i] *= 16 * 16;
11     }
12 //加密AES会话密钥
13 largeInt c;
14 c = Encrypt(key_largeInt, d, n);
15 c = Encrypt(c, e_B, n_B);
16 //数据类型转换
17 char c_str[512];
18 c.to_str(c_str);
19 // 发送RSA加密后AES密钥
20 if (SOCKET_ERROR == send(Conn_new_Socket, c_str, strlen(c_str), 0)){
21     closesocket(Conn_new_Socket);
22     closesocket(ServerSocket);
23     WSACleanup();
24     return -1;
25 }

```

如下图所示发送端输入 AES 会话密钥样例：00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94，因为 RSA 进行加解密需要使用大整数类（largeInt 类），因此先做数据结构类型转换，再进行 AES 加密后发送给接收端：

```

请输入AES 128位密钥(十六进制)00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94
解输入的AES会话密钥largeInt形式为:
30153594 DA791714
710198AE 00012001
=====加密AES会话密钥=====
RSA加密AES密钥后为:
7619EBB8 E95934EB 23811986 FA28078E 9FCB73DE FD6AC53D A3B2853D F9FC3715 B0E9DAFD AC00B4B9 070A0A6A CA2462C4 CC73F761 6F80455F 6CF429DB 861C9400
893286F6 479C0A49 5AD49B5B FC017609 C248B339 2FC68FB3 2592ECC8 9AB2CC4F CB407607 DED7A7AB 4A4D59F5 94F6D833 322F6ACD CB55A857 994EB4B2 FD93684D

```

图 19: 发送端加密会话密钥

(2) 接收端

调用 `recv` 函数接收到加密的 AES 会话密钥 (`char*` 类型), 使用 `to_arr()` 函数, 将 `char*` 类型转换成数组并保存到 `largeInt` 大整数类的 `data[]` 成员变量, 此时已经满足 RSA 解密所需数据结构类型。先使用接收端 RSA 私钥解密, 再使用发送端 RSA 公钥解密, 最终再将 `data[]` 数组中每个 `int` 值拆分为 4 个单字节, 保存在 `int key[4][4]` 中以备后续 AES 解密。

接收端 AES 会话密钥加密与分配

```

1 // 接收加密的AES会话密钥
2 if (SOCKET_ERROR == recv(ClientSocket, key_str, sizeof(key_str), 0)){
3     closesocket(ClientSocket);
4     WSACleanup();
5     return -1;
6 }
7 // 数据类型转换
8 largeInt key_largeInt;
9 to_arr(key_largeInt.data, key_str);
10 // 解密
11 largeInt key_decode;
12 key_decode = Decode(key_largeInt, d, n);
13 key_decode = Decode(key_decode, e_A, n_A);
14 // 转换为
15 int key[4][4];
16 for (int i = 0; i < 4; i++)
17     for (int j = 3; j >= 0; j--){
18         key[j][i] = key_decode.data[i] % (16 * 16);
19         key_decode.data[i] = key_decode.data[i] / (16 * 16);
20     }

```

如下图所示, 接收端收到加密的 AES 会话密钥后进行解密, 结果与发送端加密前相同, 证明会话密钥加密解密发送成功:

```

=====收到对方发来的加密AES会话密钥=====
7619EBB8 E95934EB 23811986 FA28078E 9FCB73DE FD6AC53D A3B2853D F9FC3715 B0E9DAFD AC00B4B9 070A0A6A CA2462C4 CC73F761 6F80455F 6CF429DB 861C9400
893286F6 479C0A49 5AD49B5B FC017609 C248B339 2FC68FB3 2592ECC8 9AB2CC4F CB407607 DED7A7AB 4A4D59F5 94F6D833 322F6ACD CB55A857 994EB4B2 FD93684D
=====解密AES会话密钥=====
30153594 DA791714
710198AE 00012001
=====已完成密钥分配【保密性+认证性】=====

```

图 20: 接收端解密会话密钥

(三) AES 加解密

1. 文件读取

根据用户输入文件路径进行文件读取，file.seekg(0, ios::end) 设置读指针为文件结尾，以便获取当前位置即为文件大小 size。file.seekg(0, ios::beg); 设置读指针为文件开始，读取所有文件数据至发送缓冲区 sendbuf。

文件读取

```
1 string TEST_FILE;
2 cin >> TEST_FILE;
3 ifstream file(TEST_FILE, ios::binary);
4 file.seekg(0, ios::end);
5 int size = file.tellg();
6 char* buf = new char[size];
7 file.seekg(0, ios::beg);
8 file.read(buf, size);
9 string sendbuf(buf, size);
10 }
```

2. AES 加密

(1) AES 加密整体架构

先进行初始密钥扩展和初始轮密钥加，因为采用的是 128 位 AES 会话密钥，所以需要循环十轮字节代换 -> 行移位 -> 列混合 -> 密钥加，其中第 10 轮不进行列混合。

AES 加密整体架构

```
1 void Encrypt(int in[4][4], int key[4][4])
2 {
3     int type = 1;
4     int subKey[11][4][4];
5     KeyExpansion(key, subKey); // 密钥扩展
6     AddRoundKey(in, subKey[0]); // 轮密钥加
7     for (int i = 1; i <= 10; ++i)
8     {
9         ByteSub(in, type); // 字节代换
10        shiftRow(in, type); // 行移位
11        if (i != 10) // 最后一次计算不需要列混合
12            mixCol(in, type); // 列混合
13        AddRoundKey(in, subKey[i]); // 密钥加
14    }
15 }
```

(2) S 盒产生

此处 S 盒和逆 S 盒是通过数学原理实现的，不是通过静态规定的，流程如下：

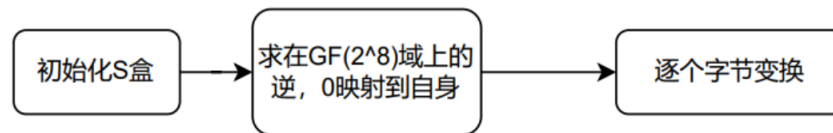


图 21: S 盒产生

S 盒产生

```

1 void s_box_gen(void)
2 {
3     int i, j;
4     int s_box_ary[16][16] = { 0 };
5     // 初始化S盒
6     for (i = 0; i < 0x10; i++)
7         for (j = 0; j < 0x10; j++)
8             s_box_ary[i][j] = ((i << 4) & 0xF0) + (j & (0xF));
9     for (i = 0; i < 0x10; i++){
10        for (j = 0; j < 0x10; j++)
11            if (s_box_ary[i][j] != 0)
12                s_box_ary[i][j] = externEuc(s_box_ary[i][j],
13                    0x11B);
14    }
15    for (i = 0; i < 0x10; i++){
16        for (j = 0; j < 0x10; j++)
17        {
18            s_box_ary[i][j] = byteTransformation(s_box_ary[i][j],
19                0x63);
20            S[i][j] = s_box_ary[i][j];
21        }
22    }
23    // 逐字节变换
24    int byteTransformation(int a, int x)
25    {
26        int tmp[8] = { 0 };
27
28        for (int i = 0; i < 8; i++)
29        {
30            tmp[i] = (((a >> i) & 0x1) ^ ((a >> ((i + 4) % 8)) & 0x1) ^
31                ((a >> ((i + 5) % 8)) & 0x1) ^ ((a >> ((i + 6) % 8)) & 0
32                x1) ^ ((a >> ((i + 7) % 8)) & 0x1) ^ ((x >> i) & 0x1)) <<
33                i;
34        }
35        tmp[0] = tmp[0] + tmp[1] + tmp[2] + tmp[3] + tmp[4] + tmp[5] + tmp[6]
36            + tmp[7];
37        return tmp[0];
  
```

34 }

生成 S 盒如下:

生成S盒:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7c	77	7b	f2	6b	6f	c5	30	1	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
4	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	2	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	b	db
a	e0	32	3a	a	49	6	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	8
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	3	f6	e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	d	bf	e6	42	68	41	99	2d	f	b0	54	bb	16

图 22: S 盒

(2) 密钥扩展

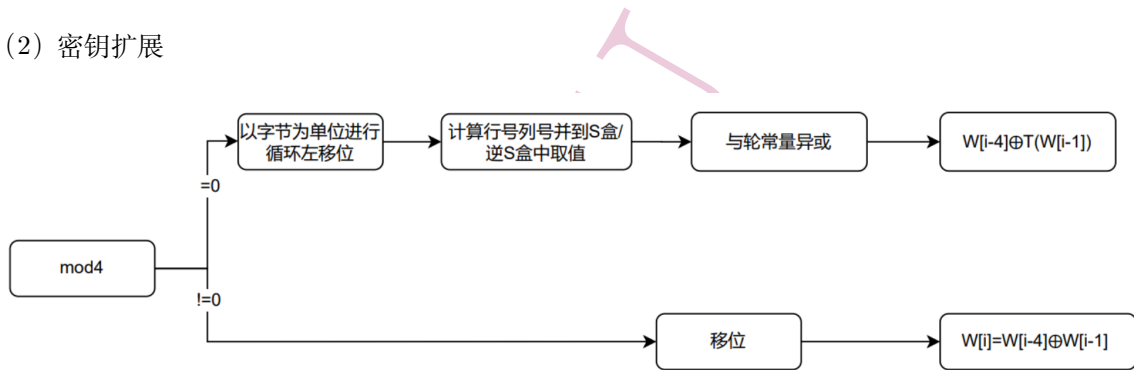


图 23: 密钥扩展

密钥扩展

```

1 void KeyExpansion(int key[4][4], int w[11][4][4])
2 {
3     for (int i = 0; i < 4; ++i)
4         for (int j = 0; j < 4; j++)
5             w[0][i][j] = key[j][i];
6
7     for (int i = 1; i < 11; ++i){
8         for (int j = 0; j < 4; ++j){
9             int temp[4];
10            if (j == 0) {
11                temp[0] = w[i - 1][3][1];
12                temp[1] = w[i - 1][3][2];
13                temp[2] = w[i - 1][3][3];
14                temp[3] = w[i - 1][3][0];
15                for (int k = 0; k < 4; ++k)
16                {
17                    int m = temp[k];

```

```

18         int row = m / 16;
19         int col = m % 16;
20         temp[k] = S[row][col];
21         if (k == 0)
22             temp[k] = temp[k] ^ rC[i - 1];
23     }
24 }
25 else {
26     temp[0] = w[i][j - 1][0];
27     temp[1] = w[i][j - 1][1];
28     temp[2] = w[i][j - 1][2];
29     temp[3] = w[i][j - 1][3];
30 }
31 for (int x = 0; x < 4; x++)
32     w[i][j][x] = w[i - 1][j][x] ^ temp[x];
33 }
34 }
35 }

```

(3) 字节代换

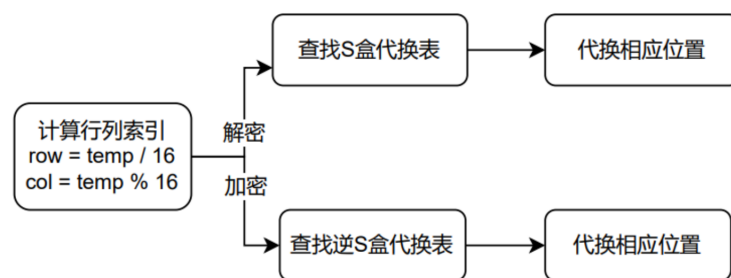


图 24: 字节代换

字节代换

```

1 void ByteSub(int in[4][4], int type)
2 {
3     for (int i = 0; i < 4; i++){
4         for (int j = 0; j < 4; j++){
5             int temp = in[i][j];
6             int row = temp / 16;
7             int col = temp % 16;
8             if (type == 1)
9                 in[i][j] = S[row][col];
10            if (type == 0)
11                in[i][j] = rS[row][col];
12        }
13    }

```

14 }

(4) 行移位

各行进行循环移位

字节代换

```

1 void ShiftRow(int in[4][4], int type) {
2     for (int i = 0; i < 4; i++){
3         for (int j = 0; j < i; j++){
4             if (type == 1){
5                 int temp = in[i][0];
6                 in[i][0] = in[i][1];
7                 in[i][1] = in[i][2];
8                 in[i][2] = in[i][3];
9                 in[i][3] = temp;
10            }
11            else {
12                int temp = in[i][3];
13                in[i][3] = in[i][2];
14                in[i][2] = in[i][1];
15                in[i][1] = in[i][0];
16                in[i][0] = temp;
17            }
18        }
19    }
20 }

```

(5) 列混合

列混合

```

1 void MixColumn(int in[4][4], int type)
2 {
3     for (int i = 0; i < 4; i++){
4         int t0 = in[0][i];
5         int t1 = in[1][i];
6         int t2 = in[2][i];
7         int t3 = in[3][i];
8         if (type == 1){
9             in[0][i] = aesMult(t0, 2) ^ aesMult(t1, 3) ^ t2 ^ t3;
10            in[1][i] = t0 ^ aesMult(t1, 2) ^ aesMult(t2, 3) ^ t3;
11            in[2][i] = t0 ^ t1 ^ aesMult(t2, 2) ^ aesMult(t3, 3);
12            in[3][i] = aesMult(t0, 3) ^ t1 ^ t2 ^ aesMult(t3, 2);
13        }
14        else{
15            in[0][i] = aesMult(t0, 14) ^ aesMult(t1, 11) ^
                aesMult(t2, 13) ^ aesMult(t3, 9);

```



```

16         in[1][i] = aesMult(t0, 9) ^ aesMult(t1, 14) ^ aesMult
17             (t2, 11) ^ aesMult(t3, 13);
18         in[2][i] = aesMult(t0, 13) ^ aesMult(t1, 9) ^ aesMult
19             (t2, 14) ^ aesMult(t3, 11);
20         in[3][i] = aesMult(t0, 11) ^ aesMult(t1, 13) ^
21             aesMult(t2, 9) ^ aesMult(t3, 14);
    }
}

```

(6) 轮密钥加

将轮密钥与状态进行逐比特异或。

轮密钥加

```

1 void AddRoundKey(int in[4][4], int key[4][4])
2 {
3     for (int i = 0; i < 4; ++i)
4         for (int j = 0; j < 4; j++)
5             in[i][j] = in[i][j] ^ key[j][i];
6 }

```

3. CBC 模式

(1) CBC 模式加密

设置一个明文分组为 128 位，计算循环轮数 N 。初始化异或向量 IV ，第一个明文分组与初始化向量 IV 异或后的结果进行 AES 加密得到第一组密文 $C1$ ，更新异或向量为 $C1$ ，发送 $C1$ 至接收端。第二个明文分组与与异或向量 IV 异或后的结果进行 AES 加密得到第二组密文 $C2$ ，更新异或向量为 $C2$ ，发送 $C2$ 至接收端。之后的数据以此类推，循环 N 轮得到 C_n 并发送至接收端，最后发送“finish”至接收端意指此加密文件传输结束。

CBC 模式加密

```

1 int _length = sendbuf.length();
2 bool first_round = true;
3 int rounds = 0;
4 int start = 0;
5 int end = 0;
6 char plaintext[30] = { 0 };
7 unsigned char m_iv[16];
8 unsigned char iv[] = {
9     103,35,148,239,76,213,47,118,255,222,123,176,106,134,98,92 };
10 unsigned char ciphertext[20] = { 0 };
11 unsigned char input[20] = { 0 };
12 memcpy(m_iv, iv, 16);
13 if (_length % 16 == 0)
14     rounds = _length / 16;
15 else

```

```

16     rounds = _length / 16 + 1;
17
18 // 循环发送分组
19 for (int j = 0; j < rounds; j++){
20     start = j * 16;
21     end = j * 16 + 16;
22     if (end > _length)
23         end = _length;
24     memset(plaintext, 0, 16);
25     memcpy(plaintext, &sendbuf[0] + start, end - start);
26     plaintext[end - start] = '\0';
27     // 与向量进行异或
28     for (int i = 0; i < 16; ++i) {
29         if (first_round == true) {
30             input[i] = plaintext[i] ^ m_iv[i];
31         }
32         else {
33             input[i] = plaintext[i] ^ ciphertext[i];
34         }
35     }
36     first_round = false;
37     input[16] = '\0';
38     // 明文分组
39     for (int i = 0; i < 4; i++){
40         for (int j = 0; j < 4; j++){
41             txt[j][i] = input[i * 4 + j];
42         }
43         // AES加密
44         Encrypt_AES(txt, key);
45         // 保存加密结果作为下一轮异或向量
46         for (int i = 0; i < 4; i++){
47             for (int j = 0; j < 4; j++){
48                 ciphertext[i * 4 + j] = txt[j][i];
49             }
50         }
51     }
52 }

```

(2) CBC 模式解密

初始化异或向量 IV（与加密时相同），接收到第一个密文分组后，将第一个明文分组进行 AES 解密后与初始化向量 IV 异或，得到第一个明文分组 D1，更新异或向量为 D1。第二个明文分组进行 AES 解密后与初始化向量 IV 异或得到第二个明文分组 D2，更新异或向量为 D2。直到接收到“finish”，此加密文件传输结束。拼接 D1D2...Dn 即为完整明文，系统提示用户输入该文件保存路径并保存至用户指定位置。

CBC 模式解密

```

1 // 接收密文分组
2 char ciphertext_str[512] = { 0 };
3 if (SOCKET_ERROR == recv(ClientSocket, ciphertext_str, sizeof(ciphertext_str)
4     , 0)){
5     closesocket(ClientSocket);
6 }

```

```

5     WSACleanup();
6     return -1;
7 }
8 // 类型转换
9 largeInt ciphertext_largeInt;
10 to_arr(ciphertext_largeInt.data, ciphertext_str);
11 int ciphertext[4][4];
12 for (int i = 0; i < 4; i++)
13     for (int j = 3; j >= 0; j--)
14     {
15         ciphertext[j][i] = ciphertext_largeInt.data[i] % (16 * 16);
16         ciphertext_largeInt.data[i] = ciphertext_largeInt.data[i] / (16 * 16)
17         ;
18     }
19 for (int i = 0; i < 4; i++)
20     for (int j = 0; j < 4; j++)
21         input[4 * i + j] = ciphertext[j][i];
22 input[16] = '\0';
23 // AES解密
24 Decode_AES(ciphertext, key);
25 memset(buffer, 0, sizeof(buffer));
26 for (int i = 0; i < 4; i++)
27     for (int j = 0; j < 4; j++)
28         buffer[i * 4 + j] = ciphertext[j][i];
29 buffer[16] = '\0';
30 // 异或运算
31 for (int i = 0; i < 16; i++) {
32     if (first_round == true) {
33         plaintext[i] = m_iv[i] ^ buffer[i];
34     }
35     else {
36         plaintext[i] = xor_input[i] ^ buffer[i];
37     }
38 }
39 plaintext[16] = '\0';
40 first_round = false;
41 // 保存加密结果作为下一轮IV向量异或
42 memcpy(xor_input, input, 16);
43 xor_input[16] = '\0';
44 cout << endl;
45 res += string((const char*)plaintext, strlen(plaintext));
46 }

```

(3) 程序运行

CBC 模式下, 如下图所示为附件中 1.txt 测试样例发送端和接收端加解密前后, 每个明文密文分组详细输出, 图 25 中 AES 解密后明文分组与图 24 中红框部分明文分组相同, AES 加

解密与加密通信传输成功：

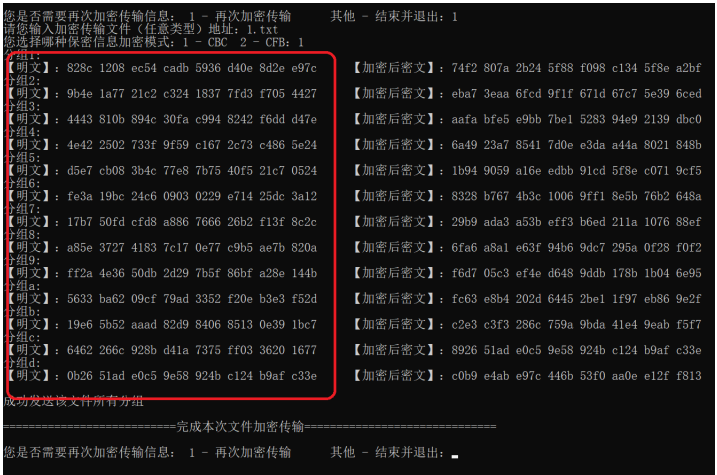


图 25: 发送端各个明文分组 AES 加密前后

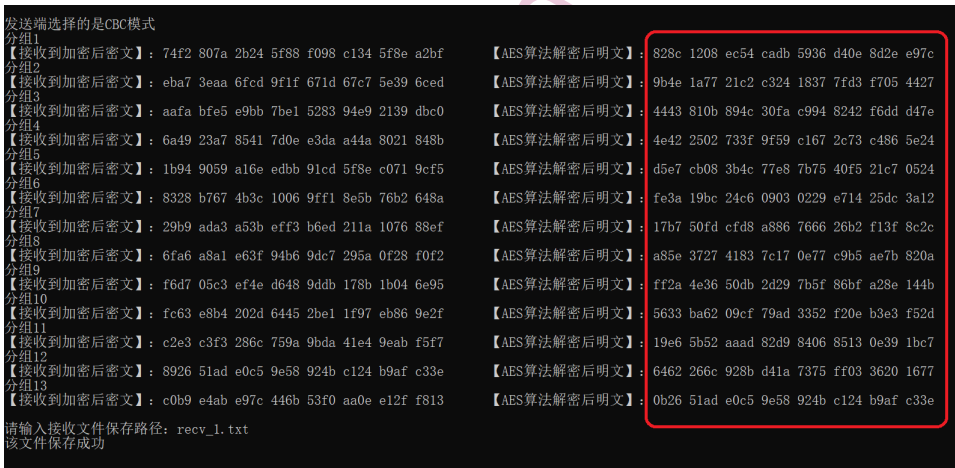


图 26: 接收端各个分组 AES 解密

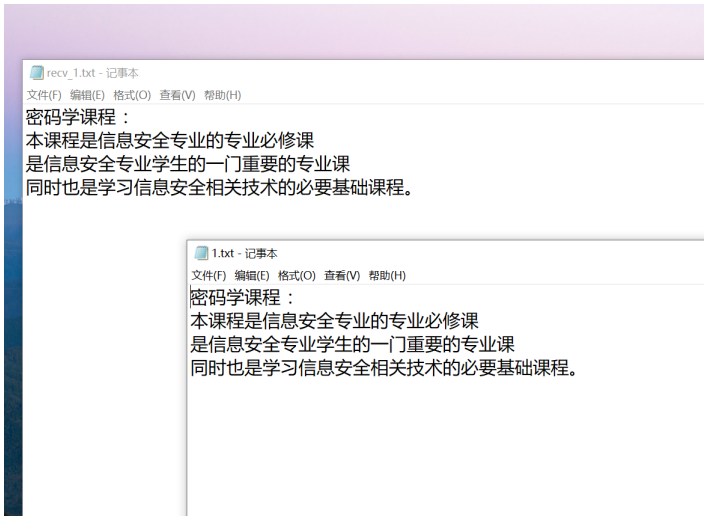


图 27: 传输内容对比

4. CFB 模式

(1) CFB 模式加密

设置一个明文分组为 128 位，计算循环轮数 N。初始化向量 IV，第一个明文分组时，初始化向量 IV 通过密钥 key 加密后与明文进行异或得到密文分组 C1，更新向量 IV 为 C1，循环到第二个明文分组时，向量 IV 通过密钥 key 加密后与明文进行异或得到密文分组 C2，更新向量 IV 为 C2，以此类推循环明文分组 N 轮，最终得到 Cn 并发送至接收端，最后发送“finish”至接收端意指此加密文件传输结束。

CBC 模式加密

```

1 // AES加密
2 int iv_temp[4][4];
3 if (first_round == false) {
4     for (int i = 0; i < 4; i++)
5         for (int j = 0; j < 4; j++)
6             iv_temp[j][i] = cipher[i * 4 + j];
7     Encrypt_AES(iv_temp, key);
8 }
9 else {
10     first_round = false;
11     for (int i = 0; i < 4; i++)
12         for (int j = 0; j < 4; j++)
13             iv_temp[j][i] = m_iv[i * 4 + j];
14     Encrypt_AES(iv_temp, key);
15 }
16 //数据结构类型转换
17 for (int i = 0; i < 4; i++)
18     for (int j = 0; j < 4; j++)
19         input[i * 4 + j] = iv_temp[j][i];
20 // 异或运算
21 for (int i = 0; i < 16; i++)
22     cipher[i] = plaintext[i] ^ input[i];
23 // 保存下一轮向量
24 for (int i = 0; i < 4; i++)
25     for (int j = 0; j < 4; j++)
26         txt[j][i] = cipher[i * 4 + j];

```

(2) CFB 模式解密

初始化异或向量 IV（与加密时相同），接收到第一个密文分组后，第一个密文分组时，初始化向量 IV 通过密钥加密后，与密文分组 1 进行异或得到明文分组 D1，更新向量为 D1。第二个密文分组时，向量 IV 通过密钥加密后，与密文分组 1 进行异或得到明文分组 D2，更新向量为 D2。直到接收到“finish”，此加密文件传输结束。拼接 D1D2...Dn 即为完整明文，系统提示用户输入该文件保存路径并保存至用户指定位置。

CFB 模式解密

```

1 if (first_round == false) {
2     Encrypt_AES(iv_temp, key);

```

```

3 }
4 else {
5     first_round = false;
6     for (int i = 0; i < 4; i++)
7         for (int j = 0; j < 4; j++)
8             iv_temp[j][i] = m_iv[i * 4 + j];
9     Encrypt_AES(iv_temp, key);
10 }
11 for (int i = 0; i < 4; i++)
12     for (int j = 0; j < 4; j++)
13         xor_input[i * 4 + j] = iv_temp[j][i];
14 // 异或
15 for (int i = 0; i < 16; i++)
16     plaintext[i] = input[i] ^ xor_input[i];
17 // 保存下一轮向量
18 for (int i = 0; i < 4; i++)
19     for (int j = 0; j < 4; j++)
20         iv_temp[i][j] = ciphertext[i][j];
21
22 }
23 res += string((const char*)plaintext, strlen(plaintext));

```

(3) 程序运行

CFB 模式下, 如下图所示为附件中 2.txt 测试样例发送端和接收端加解密前后, 每个明文密文分组详细输出, 图 25 中 AES 解密后红框部分与图 24 中红框部分明文分组相同, AES 加解密与加密通信传输成功:

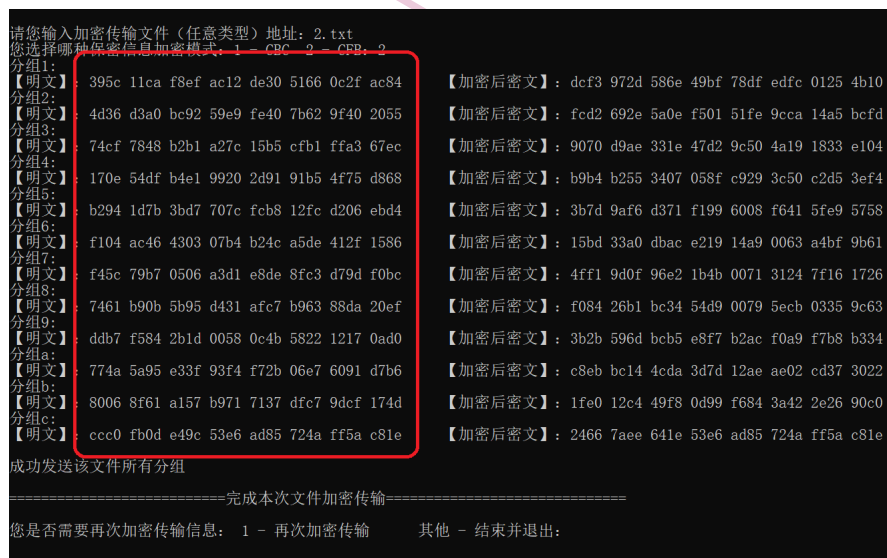


图 28: 发送端各个明文分组 AES 加密前后

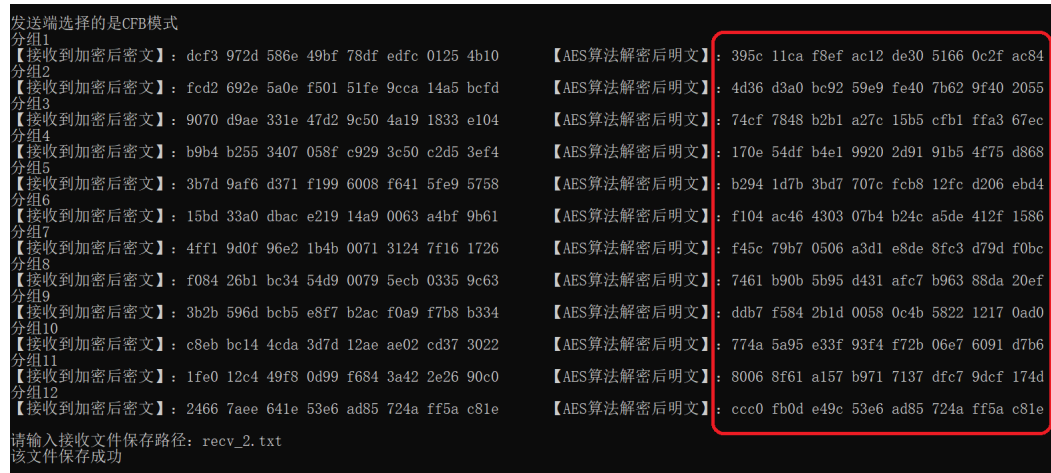


图 29: 接收端各个分组 AES 解密前后



图 30: 传输内容对比

四、 遇到的困难及解决

(一) 类型转换

在 RSA 中我使用的是大整数类 `largeInt`, 通信传输 `send` 和 `recv` 函数用的是 `char*`, AES 加解密用的是 `int` 二维数组存储 128 位 (16 字节) 会话密钥及明文密文。因此我在 `largeInt` 补充 `void to_str(char s[])` 函数将 `largeInt` 类型转为 `char*` 类型, 添加 `to_arr(unsigned int a[], char s[])` 函数, 将 `recv` 函数接收到的 `char*` 类型转换为数组存储。

为避免篇幅冗余以下仅展示 `to_str` 函数代码:

`largeInt` 类型转为 `char*` 类型

```
1 void largeInt::to_str(char s[])
2 {
3     cout << endl;
4     int k = 0;
5     unsigned int temp, result;
6     unsigned int tempAnd = 0xf0000000;
```

```
7
8   for (int i = GetLength() - 1; i >= 0; i--)
9   {
10      temp = data[i];
11      //大数的每一位数字转换成16进制输出
12      for (int j = 0; j < 8; j++)
13      {
14          result = temp & tempAnd;
15          result = (result >> 28);
16          temp = (temp << 4);
17          if (result >= 0 && result <= 9)
18              s[k++] = result + '0';
19          else
20          {
21              switch (result)
22              {
23                  case 10:
24                      s[k++] = 'A';
25                      break;
26                  case 11:
27                      s[k++] = 'B';
28                      break;
29                  case 12:
30                      s[k++] = 'C';
31                      break;
32                  case 13:
33                      s[k++] = 'D';
34                      break;
35                  case 14:
36                      s[k++] = 'E';
37                      break;
38                  case 15:
39                      s[k++] = 'F';
40                      break;
41              }
42          }
43      }
44  }
45  s[k] = '\0';
46 }
```

(二) Sleep 相关问题

发送端发送 AES 加密后的密文分组的时，一开始我在每次循环之间没有加 Sleep 就会导致接收端有漏收的情况，因此需要在每发完一个密文分组之后补充 Sleep 再去加密发送下一个密文分组。

(三) 个人粗心

在 CBC 加密模式中，发送端需要在 AES 加密前对明文进行异或，其中第一次需要用到初始向量，之后均使用前一轮的加密结果作为异或向量，我设置 `bool first_round` 来判断是否为第一次，当我在第一轮异或后并没有将 `first_round` 置为 `False`，当时一直传输不对卡了半天，结果发现是自己粗心遗漏了。

五、 可执行文件运行说明

所有流程图、源代码和可执行文件均在附件中，为了方便您运行可执行文件，特此介绍可执行文件运行相关说明：

- (1) 依次双击 `UserA.exe` 和 `UserB.exe`（按顺序）运行可执行文件。
- (2) 发送端和接收端均提示选则随机生成（选 1）或自行输入公钥或私钥（选 2），如果您选则 1 则自动生成 RSA 公钥和私钥，并且系统会输出大素数 `P` 和 `Q` 产生过程（含 Miller-Rabin 测试）。
- (3) 发送端（`UserA.exe`）系统提示“请输入您的身份 ID（不超过 6 位）”，您可以输入任意不超过 6 位的字符串。
- (3) 系统提示“请输入 AES 128 位密钥（十六进制）”，样例：00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94
- (4) 您可以在发送端（`UserA.exe`）看到加密前后 AES 会话密钥和生成的 S 盒，在接收端看到收到的加密 AES 会话密钥和解密后的 AES 会话密钥【与发送端对比一致】
- (5) 发送端（`UserA.exe`）系统提示“请您输入加密传输文件（任意类型）地址：”，您可以输入您加密文件的路径，已经在本文件夹为您可供测试样例：1.txt 和 2.txt，因此您可以输入 1.txt 或 2.txt 进行直接测试。
- (6) 发送端（`UserA.exe`）系统提示“您选择哪种保密信息加密模式”，您可以输入 1 选则 CBC 模式，也可以输入 2 选则 CFB 模式
- (7) 您可以在发送端（`UserA.exe`）观察到每个明文分组加密前后 16 进制信息，在接收端（`UserB.exe`）观察到每个接收到的密文分组解密后的明文信息【与发送端对比一致】
- (8) 接收端（`UserB.exe`）系统提示“请输入文件保存路径：”，您可以输入文件保存路径及文件名，测试样例：recv_1.txt
- (9) 接收端（`UserB.exe`）系统提示“该文件保存成功”，发送端（`UserA.exe`）系统提示“您是否需要再次加密传输信息：”，您如果选则 1 则发送端和接收端均退出程序，如果选则其他任意摁键则继续发送下一个您即将指定的待加密传输文件。