南开大学

Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

密码学 (1091) 大作业程序说明文档

基于可信第三方的 RSA,AES 群聊保密通信协议

苗发生

年级：2020 级

专业：信息安全 & 法学双学位

指导教师：古力

2023 年 1 月 8 日

目录

一、 实验说明

(一) 程序运行环境

运行 Windows 操作系统的 PC 机，Visual studio2019，以及能够开发网络编程的环境，比如配置 Npcap 工具包，文件包含目录，库文件目录，连接器以及预处理器的配置，具体配置截图如下：

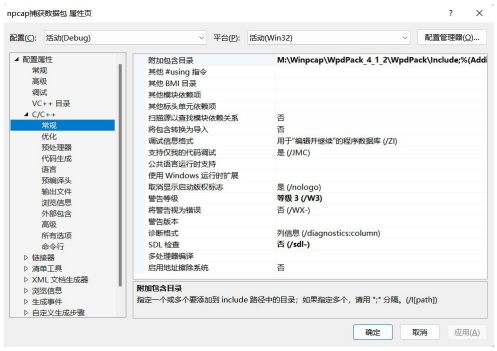

图 1: 添加包含文件目录

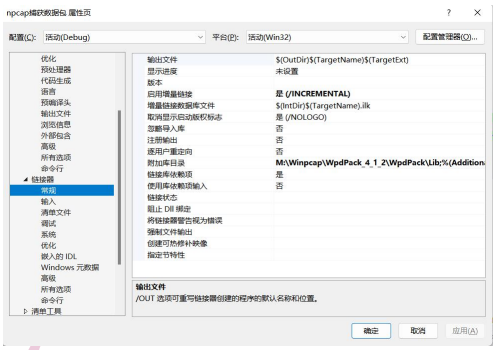

图 2: 添加库文件目录

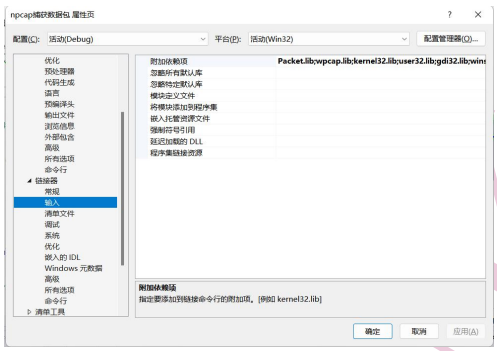

图 3: 添加链接时使用的库文件

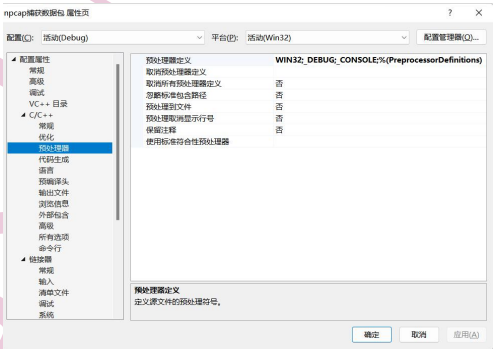

图 4: 预处理器配置

(二) 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双方的公钥。(大作业文档中的假设说明)

本次协议是实现基于 RSA,AES 的群聊保密通信，为了避免不同客户端利用自己的密钥去解密其他人收到的加密数据包，在本项目中，我们不需要客户端知道对方的 AES 密钥（即不同客户端持有不同的 AES 密钥且无法获知对方的 AES 密钥），进一步降低了假设条件（但需要引入可信第三方，可信第三方负责管理和分配所有客户端的密钥，假设可信第三方是安全的）。

(三) 大作业要求

设计一个保密通信的协议，具体要求为：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。

具体要求如下：

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

4) 大作业的提交方式同实验报告的提交，也就是说既要提交程序实现的说明文档，也要提交源代码和可执行程序

本项目在完成实验要求的所有细节后，对协议和项目进行了充分的扩展，将 A,B 两个客户端的通信扩展成基于可信第三方的群聊通信（也完成可以满足两个客户端的单聊，即实验要求），通信中还包括用户自己选择的可信凭证（依此来判断加密通信信息是否被篡改），并且密钥的分配和管理集中在可信第三方，每个客户端只需要从 KDC（密钥分配中心）获取自己的 AES 加解密密钥即可，无需知道其他客户端的密钥，这也进一步提高了群聊中的保密通信安全。

二、 必备基础知识

(一) RSA 密钥分配算法

1. RSA 算法介绍

RSA 算法是目前理论和实际应用中最为成熟的和完善的公钥密码体制。RSA 用来解决对称密码的密钥分发问题（在本项目中，可信第三方通过 RSA 算法来进行密钥分配）。还可以用来进行数字签名来保证信息的否定与抵赖，利用数字签名较容易发现攻击者对信息的非法篡改以保证信息的完整性（本项目中通过引入通信凭证并借助 RSA 和 AES 算法来确保信息来源正确以及传输过程中未被篡改）。RSA 的安全性依赖于大整数的因子分解的困难性，为了满足信息安全强度的需求，密钥的位数都比较多 (1024 位甚至更高)，导致幂模运算的运算量极大，成为提高 RSA 算法加解密速度的瓶颈。

在本实验中，为了确保密钥分配的效率，我们选择 256 位的 RSA 密钥，如果想要进一步提高安全性，只需要简单修改源代码中的参数即可！

2. RSA 算法原理

序列密码和分组密码算法都要求通信双方通过交换密钥实现使用同一个密钥，这在密钥的管理、发布和安全性方面存在很多问题，而公钥密码算法解决了这个问题。

公钥密码算法是指一个加密系统的加密密钥和解密密钥是不同的，或者说不能用其中一个推导出另一个。在公钥密码算法的两个密钥中，一个是用于加密的密钥，它是可以公开的，称为公钥；另一个是用于解密的密钥，是保密的，称为私钥。公钥密码算法解决了对称密码体制中密钥管理的难题，并提供了对信息发送人的身份进行验证的手段，是现代密码学最重要的发明。

RSA 密码体制是目前为止最成功的公钥密码算法，它是在 1977 年由 Rivest 、Shamir 和 Adleman 提出的第一个比较完善的公钥密码算法。它的安全性是建立在“大数分解和素性检测”这个数论难题的基础上，即将两个大素数相乘在计算上容易实现，而将该乘积分解为两个大素数因子的计算量相当大。虽然它的安全性还未能得到理论证明，但经过 40 多年的密码分析和攻击，迄今仍然被实践证明是安全的。

3. RSA 算法具体细节

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，则

$$\varphi(n) = (p-1)(q-1)$$

其中 $\varphi(n)$ 为欧拉函数，随机选取正整数 e，使其满足 (e,$\varphi(n)$)=1, 即 e 和 $\varphi(n)$ 互素，则将（n，e）作为公钥

2) **私钥:**

求出正整数 d，使其满足

$$e \times d \equiv 1 mod(\varphi(n))$$

则将（n，d）作为私钥

3) **加密算法:**

对于明文 m，由 $c \equiv m^e mod(n)$, 得到密文 c

4) **解密算法:**

对于密文 c，由 $m \equiv c^d mod(n)$, 得到密文 m

如果攻击者获得了 n 、e 和密文 c，为了破解密文必须计算出私钥 d，为此需要先分解 n。当 n 的长度为 1024 比特时，在目前还是安全的，但从因式分解技术的发展来看，1024 比特并不能保证长期的安全性。为了保证安全性，要求在一般的商业应用中使用 1024 比特的长度，在更高级别的使用场合，要求使用 2048 比特长度。

RSA 整体运行示意图如下:

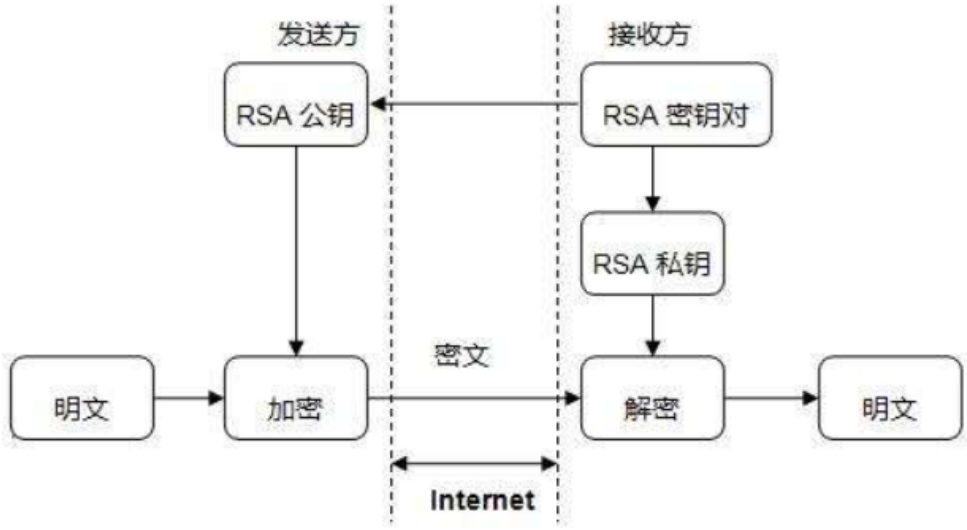

图 5: RSA 运行示意图

4. 生成大素数

大素数的生成主要包括线性同余生成大随机奇数以及 Miller-Rabin 测试，流程图如下图所示:

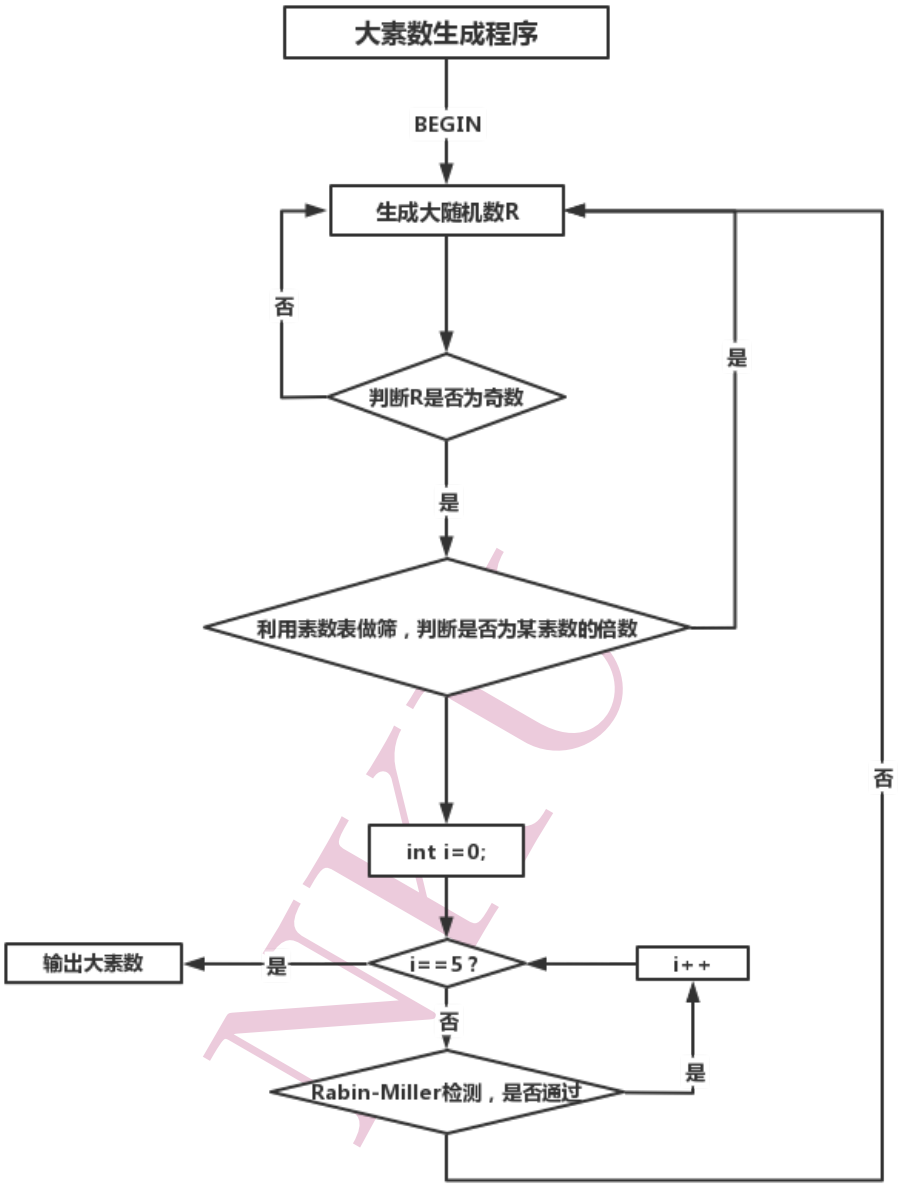

图 6: 生成大素数流程图

(二) AES 加密解密算法

1. AES 算法简介

AES 是高级加密标准（英语：Advanced Encryption Standard）的缩写，在密码学中又称 Rijndael 加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的 DES，已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院（NIST）于 2001 年 11 月 26 日发布于 FIPS PUB 197，并在 2002 年 5 月 26 日成为有效的标准。2006 年，高级加密标准已然成为对称密钥加密中最流行的算法之一

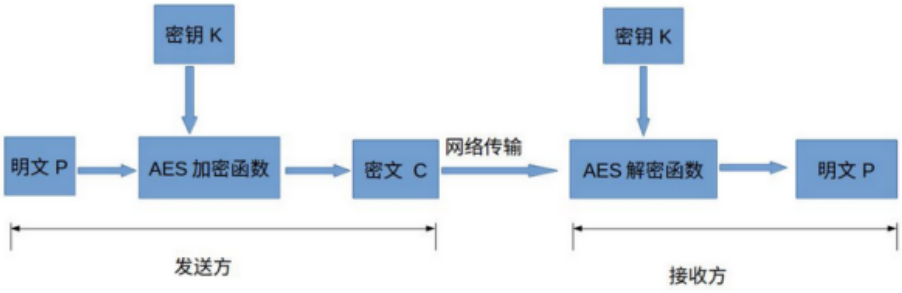

图 7: 使用 AES 的流程

2. AES 算法原理

AES 算法本质上是一种对称分组密码体制，采用代替/置换网络，每轮由三层组成：线性混合层确保多轮之上的高度扩散，非线性层由 16 个 S 盒并置起到混淆的作用，密钥加密层将子密钥异或到中间状态。Rijndael 是一个迭代分组密码，其分组长度和密钥长度都是可变的，只是为了满足 AES 的要求才限定处理的分组大小为 128 位，而密钥长度为 128 位、192 位或 256 位，相应的迭代轮数 N，为 10 轮、12 轮、14 轮。AES 汇聚了安全性能、效率、可实现性、灵活性等优点。最大的优点是可以给出算法的最佳差分特征的概率，并分析算法抵抗差分密码分析及线性密码分析的能力。

加密的主要过程包括：对明文状态的一次密钥加，$N_r - 1$ 轮轮加密和末尾轮轮加密，最后得到密文。其中 $N_r - 1$ 轮轮加密每一轮有四个部件，包括字节代换部件 ByteSub、行移位变换 ShiftRow、列混合变换 NixColumn 和一个密钥加 AddRoundKey 部件，末尾轮加密和前面轮加密类似，只是少了一个列混合变换 NixColumn 部件。解密过程与加密过程类似。

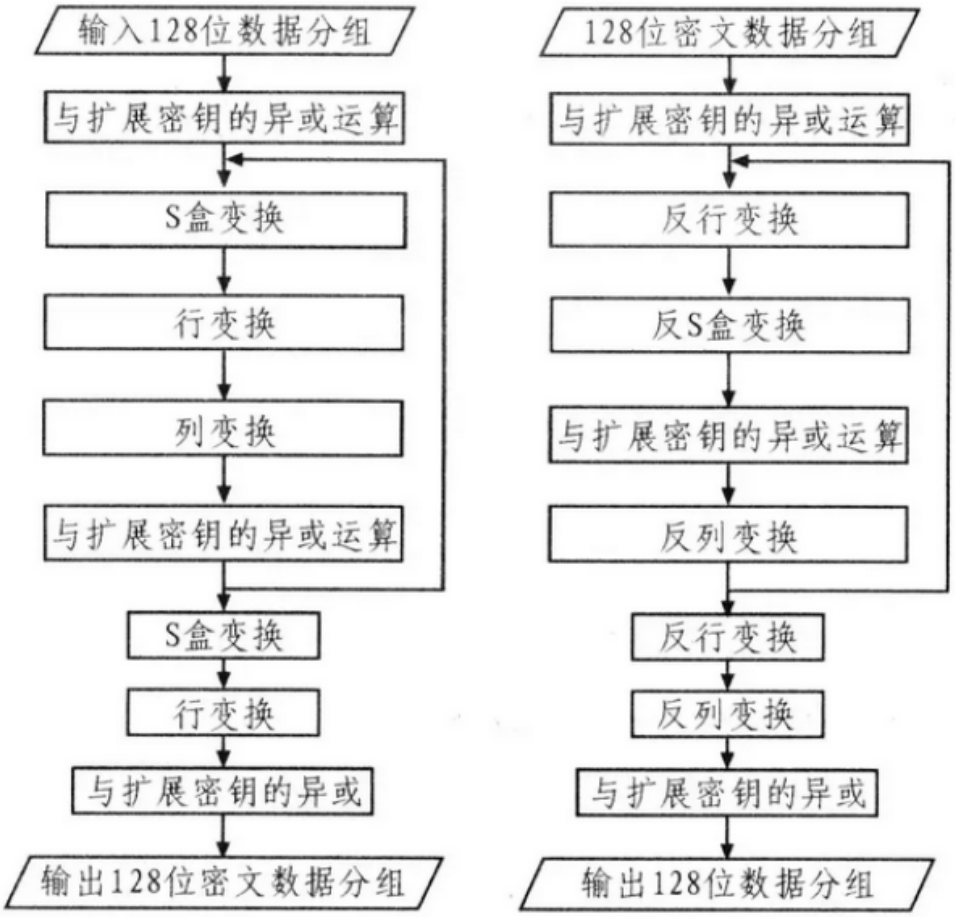

图 8: AES 算法加密和解密

3. AES 算法的 CBC 模式

CBC 模式的全称是 Cipher Block Chaining 模式（密文分组链接模式），之所以叫这个名字，是因为密文分组像链条一样相互连接在一起。在 CBC 模式中，首先将明文分组与前一个密文分组进行 XOR 运算，然后再进行加密。

CBC 模式加密过程如下：

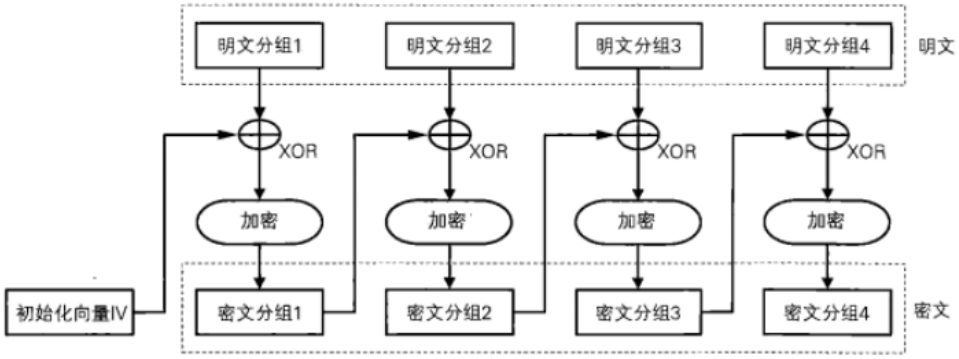

图 9: CBC 模式加密

CBC 模式解密与加密类似，以密文分组作为输入，对第一个密文分组直接解密，后边的密

文分组解密后需要和前一个密文分组进行 XOR 操作

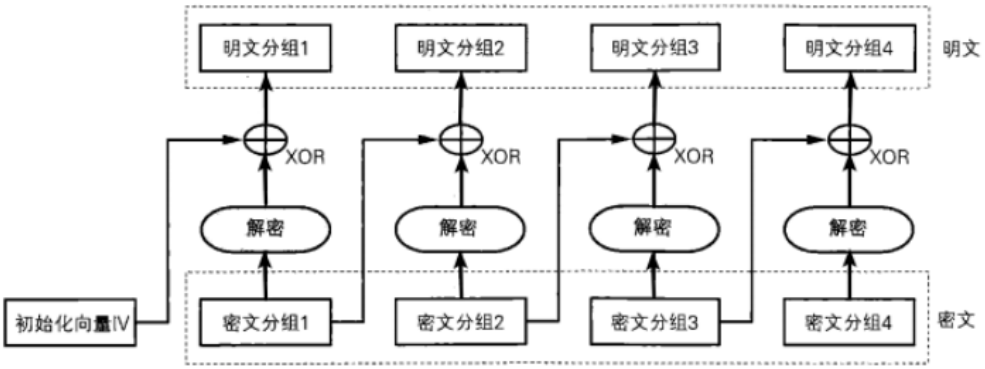

图 10: CBC 模式解密

在本次实验中，AES 算法的加解密我们均采用 CBC 模式！

4. AES 算法具体细节

1) 字节替换 ByteSub 部件

字节代换是非线形变换，独立地对状态的每个字节进行。代换表（即 S-盒）是可逆的，由以下两个变换的合成得到：

A) 首先，将字节看作 $GF(2^8)$ 上的元素，映射到自己的乘法逆元，‘00’映射到自己

B) 其次，对字节做如下的（GF(2) 上的，可逆的）仿射变换：

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

图 11: 仿射变换

字节代换整个过程的示意图如下所示：

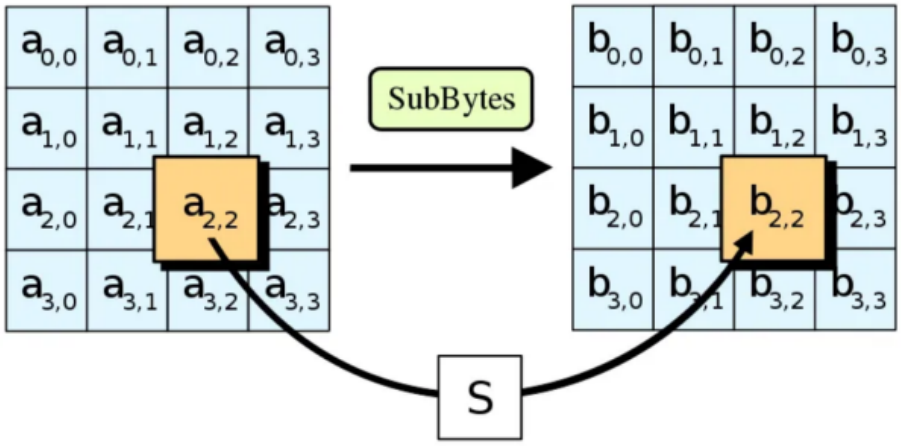

图 12: 字节替换

注意：该部件的逆运算部件就是先对自己做一个逆仿射变换，然后映射到自己的乘法逆元上

2) **行移位变换 ShiftRow 部件**

行移位是将状态阵列的各行进行循环移位，不同状态行的位移量不同。第 0 行不移动，第 1 行循环左移 C_1 个字节，第 2 行循环左移 C_2 个字节，第 3 行循环左移 C_3 个字节。位移量 C_1、C_2、C_3 的取值与 N_b 有关，对应于不同长度的位移量如下表所示:

N_b	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

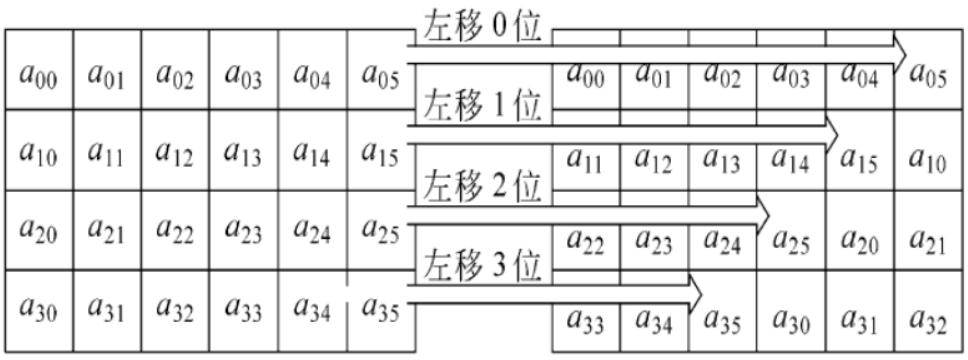

图 13: 行移位示意图

ShiftRow 的逆变换是对状态阵列的后 3 列分别以位移量 N_b-C_1、N_b-C_2、N_b-C_3 进行循环移位，使得第 i 行第 j 列的字节移位到

$$(j + N_b - C_i) mod N_b$$

3) **列混合变换 NixColumn 部件**

在列混合变换中，将状态阵列的每个列视为系数为 $GF(2^8)$ 上的多项式，再与一个固定的多项式 c(x) 进行模 x^4+1 乘法。当然要求 c(x) 是模 x^4+1 可逆的多项式，否则列混合变换就是不可逆的，因而会使不同的输入分组对应的输出分组可能相同。Rijndael 的设计者给出的 c(x) 为（系数用十六进制数表示）：

$$c(x) = 03x^3 + 01x^2 + 01x + 02$$

c(x) 是与 x^4+1 互素的，因此是模 x^4+1 可逆的。列混合运算也可写为矩阵乘法。设 b(x)= c(x)a(x)，则

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

图 14: 列混合中的矩阵乘法

这个运算需要做 GF(2^8) 上的乘法，但由于所乘的因子是 3 个固定的元素 02、03、01，所以这些乘法运算也是比较简单的。列混合运算的整体示意图如下：

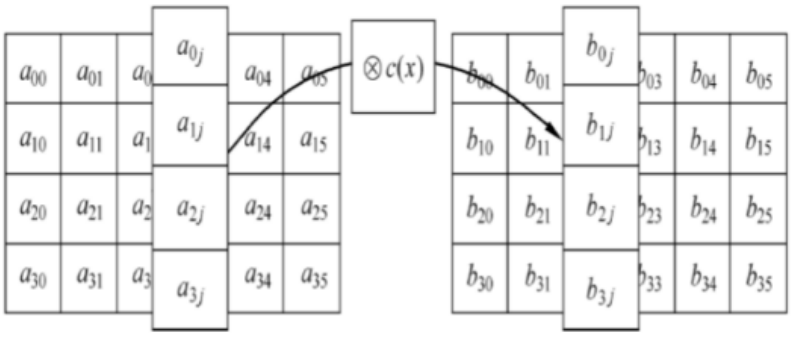

图 15: 列混合运算示意图

注意：列混合运算的逆运算是类似的，即每列都用一个特定的多项式 d(x) 相乘，d(x) 满足如下

$$03x^3 + 01x^2 + 01x + 02 \otimes d(x) = 01$$

由此可得

$$d(x) = 0Bx^3 + 0Dx^2 + 09x + 0E$$

4) **密钥加 AddRoundKey 部件**

密钥加是将轮密钥简单地与状态进行逐比特异或。轮密钥由种子密钥通过密钥编排算法得到，轮密钥长度等于分组长度 N_b。密钥加运算的逆运算是其自身。

密钥加运算示意图如下：

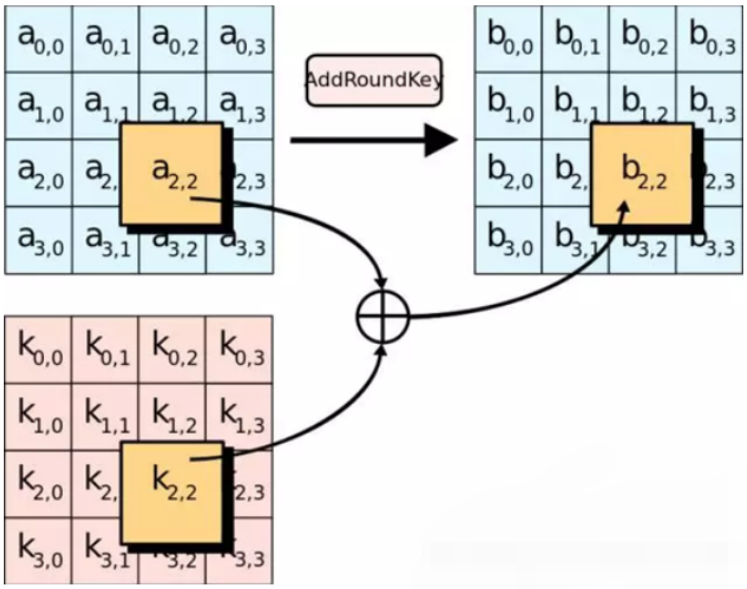

图 16: 密钥加运算示意图

5) **密钥编排**

密钥编排指从种子密钥得到轮密钥的过程，它由密钥扩展和轮密钥选取两部分组成。其基本原则如下：

(1) 轮密钥的字数（4 比特 32 位的数）等于分组长度乘以轮数加 1；(2) 种子密钥被扩展成为扩展密钥；(3) 轮密钥从扩展密钥中取，其中第 1 轮轮密钥取扩展密钥的前 N_b 个字，第 2 轮轮密钥取接下来的 N_b 个字，如此下去。

A) 轮密钥的字数（4 比特 32 位的数）等于分组长度乘以轮数加 1

B) 种子密钥被扩展成为扩展密钥

C) 轮密钥从扩展密钥中取，其中第 1 轮轮密钥取扩展密钥的前 N_b 个字，第 2 轮轮密钥取接下来的 N_b 个字，如此下去

6) **解密过程**

AES 算法的解密过程和加密过程是相似的，也是先经过一个密钥加，然后进行 $N_r - 1$ 轮轮解密和末尾轮轮解密，最后得到明文。和加密不同的是 $N_r - 1$ 轮轮解密每一轮四个部件都需要用到它们的逆运算部件，包括字节代换部件的逆运算、行移位变换的逆变换、逆列混合变换和一个密钥加部件，末尾轮加密和前面轮加密类似，只是少了一个逆列混合变换部件。

在解密的时候，还要注意轮密钥和加密密钥的区别，设加密算法的初始密钥加、第 1 轮、第 2 轮、…、第 Nr 轮的子密钥依次为 k(0), k(1), k(2), …, k(N_r-1), k(N_r) 则解密算法的初始密钥加、第 1 轮、第 2 轮、…、第 Nr 轮的子密钥依次为 k(N_r), InvMixColumn (k(N_r-1)), InvMixColumn (k(N_r-2)), …, InvMixColumn (k(1)), k(0)。

(三) TCP 网络传输协议

传输控制协议（TCP，Transmission Control Protocol）是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的 RFC 793 [1] 定义。TCP 旨在适应支持多网络应用的分层协议层次结构。连接到不同但互连的计算机通信网络的主计算机中的成对进程之间依靠 TCP 提供可靠的通信服务。TCP 假设它可以从较低级别的协议获得简单的，可能不可靠的数据报服务。原则上，TCP 应该能够在从硬线连接到分组交换或电路交换网络的各种通信系统之上操作。

本次设计的通信保密协议是在 TCP 的基础上，用 RSA,AES 等密码技术提供安全保障，进而开创性的实现“基于可信第三方的 RSA,AES 群聊保密通信协议”

三、 协议设计

(一) 密钥分配协议设计

1. 密钥分配协议设计

本次密钥分配协议引入可信第三方作为密钥分配中心（Key Distribution Center,KDC），每当用户接入群聊后，向 KDC 发送密钥 Request，并附带唯一通信凭证 N（防止密钥传输的过程中遭遇中间人的劫持或篡改），KDC 接到请求后，随机生成一个 128 位的大整数作为该用户的密钥，并将 N+1(N')，Request 以及大整数通过 RSA 算法进行加密，加密后发送给该用户，该用户接收到密钥后，首先进行 RSA 解密，判断接收到的 N' 是否等于 N+1，若等于，则说明没有遭到中间人伪造攻击，将该密钥保留起来用作保密通信过程中 AES 加密和解密的密钥；若不等于，则意味着遭到中间人攻击或含有密钥的数据包在传输过程中损毁，此时重新请求 KDC，直到得到的密钥可用为止。（请求次数超过 5 次则意味着该用户存在被监视的风险，直接中断请求）。

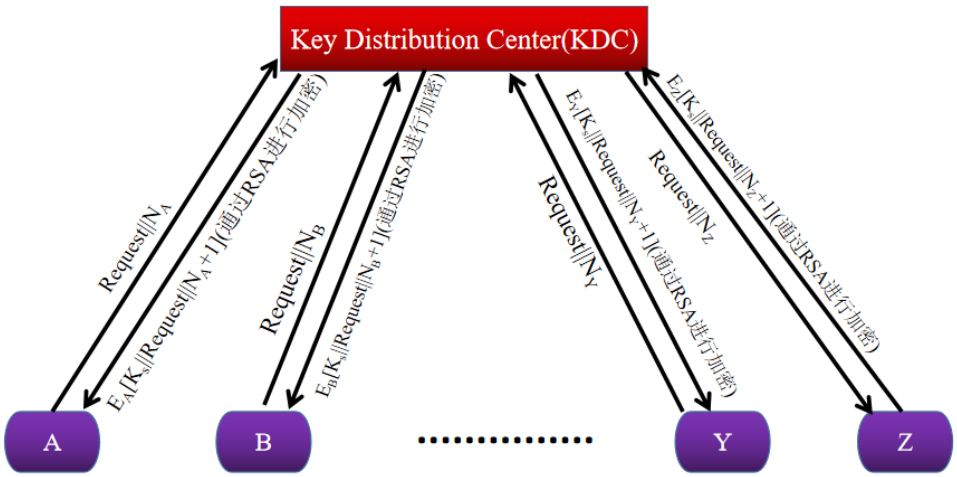

图 17: 密钥分配协议

2. 亮点及安全性分析

A) 由于密钥在请求和传递的过程中，包含有唯一凭证，通过该凭证可以判断所得到的密钥来源是否正确以及是否被监听和篡改，有效避免中间人伪造攻击以及监听

B) 该协议打破传统 AES 协议中，通信双方享有相同的加密和解密密钥；本协议中，KDC 向不同用户分配的密钥均不相同，且为完全随机生成，避免不同用户之间互相窃取对方的保密通信消息，比传统的基于 AES 加密算法的保密通信协议更安全

(二) 保密通信协议设计

为了防止不同用户之间的窃听，我们为不同用户分配了不同的 AES 加解密密钥，不同用户持有不同的加解密密钥，在此基础上实现保密通信是本次协议设计的最大难点。

为了实现持有不同密钥的用户之间可以实现通信，我们借助于可信第三方。可信第三方持有所有用户的密钥，我们发送的消息通过 AES 加密后先发送给可信第三方，可信第三方选择该用户持有的 AES 密钥对其进行解密，解密后再利用信息接收方的 AES 密钥（通过遍历密钥表得到）进行加密，加密后发送给信息的接收方，这样便实现了持有不同密钥用户之间的保密通信。

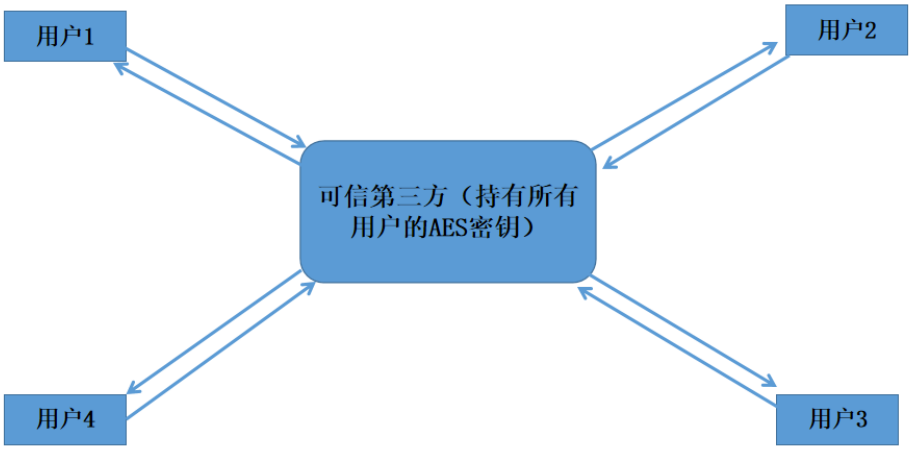

图 18: 保密通信协议

(三) 密钥管理

设计群聊模式的保密通信协议另一个难度一个比较大的挑战是密钥管理，和单聊模式不同，群聊模式涉及用户较多，不同用户之间若持有相同的 AES 加解密密钥，极容易导致信息的泄露（例如，A 用户用自己的密钥去解密 B 用户得到的密文信息）。本协议的密钥管理集中在可信第三方，可信第三方维护一个密钥表，该表存有所有用户不同的 AES 加解密密钥，每一个用户在可信第三方处的体现为一个单独的线程，该线程中也存有当前用户的密钥，不同线程互不冲突确保了不同用户之间的信息不会泄露；当对通信信息进行保密处理时，用对应用户线程中的密钥对其信息进行加密；若要进行群发（即聊天室模式），则在加密的时候，需要在密钥表中找到所有信息接收方的密钥，分别用其密钥进行加密，然后发送给对应用户。

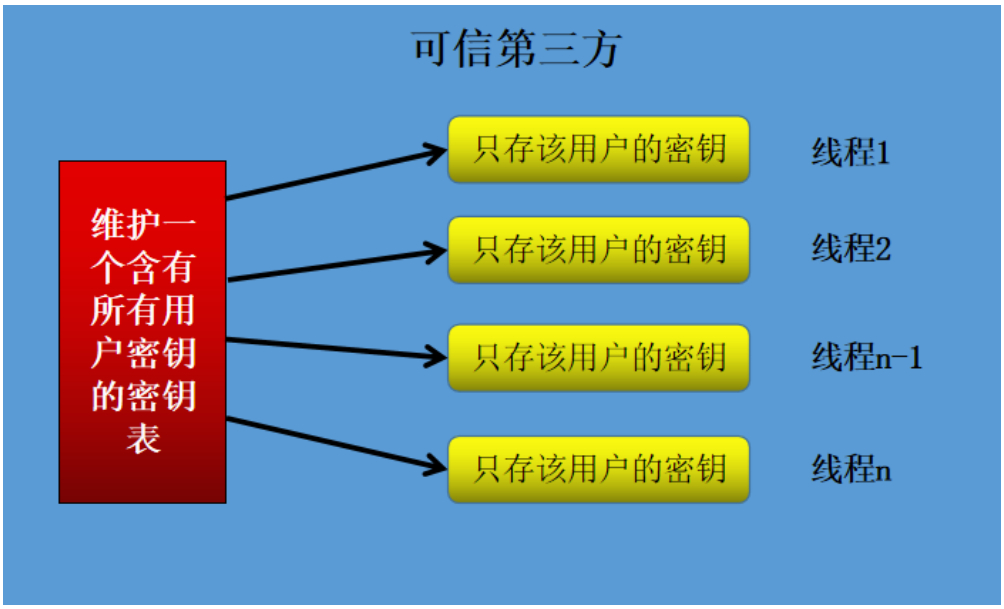

图 19: 密钥管理示意图

（四） 协议的具体步骤

1. 协议的执行步骤

本次协议是在 TCP 传输协议的基础上，借用可信第三方，利用 RSA 和 AES 密码技术实现的群聊保密通信协议，协议的具体步骤如下：

1) 启动可信第三方（负责密钥分配，密钥管理，以及保密信息的处理和转发）

2) 启动客户端 A,B,C,D,E（该保密通信协议目前支持同时启动 1000 个客户端的群聊模式），每个客户端的步骤均相同，下面以客户端 A 为例子的进行协议步骤的具体解释

3) 客户端 A 随机选取一个 0-128 的数字，作为保密通信凭证，并附带“rand”字段，向可信第三方告知请求密钥

4) 可信第三方解析到“rand”字段后，随机生成一个 128 位的大整数作为客户端 A 的密钥

5) 可信第三方将客户端 A 的保密通信凭证 N 进行加 1，即 N’，并利用客户端 A 的 RSA 公钥，对 N’，随机生成的 128 位大整数，以及客户端 A 的请求字段合成一个整体后进行加密，加密后传输给客户端 A

6) 客户端 A 收到可信第三方发来的含有 AES 密钥的密文后，利用自己的 RSA 私钥进行解密，解密后判断 N’是否等于 N+1，若不等于，则意味着密钥在传输的过程中遭到窃取或篡改，此时重新请求可信第三方（重复 3-6 步骤），直到收到可信密钥为止，若请求次数超过次，则意味着客户端 A 的通信始终受到监听和篡改，通信结束

7) 若保密通信凭证正确，客户端 A 将收到的密钥进行保存，此时客户端 A 的密钥分配成功！所有客户端成功接入群聊模式后，进行可信密钥的分配，接下来的步骤为通信保密阶段。

8) 客户端 A 将要发送的信息，利用自己从密钥分配中心（KDC，也是可信第三方）得到的密钥，通过 AES 的 CBC 模式进行加密，加密后将加密信息发送给可信第三方（若要发送的信息超过一个数据包的长度（1024Byte），则分包进行发送）

9) 可信第三方掌握所有人的密钥。在收到客户端 A 发来的加密数据包后，利用客户端 A 的 AES 密钥对数据包进行解密

10) 可信第三方识别客户端 A 要发送的目标，在密钥表中查找发送目标的 AES 密钥（不同的客户端在可信第三方处有不同的索引）

11) 可信第三方利用目标客户端的 AES 密钥，对数据包进行加密，加密后传输给目标客户端（数据的接收方）

12) 如果客户端 A 要发送的目标客户端并非一个（即群聊模式），则可信第三方重复（9-11 步骤），实现对所有目标客户端的通信保密传输

13) 由于每个客户端是双线程实现，客户端 A 不用等待收到信息后再进行传输，在传输完成第一条信息后，可继续进行保密通信其他信息

14) 如果客户端 A 想退出保密通信聊天室，则输入“Exit”，可信第三方收到“Exit”信息后，销毁客户端 A 的密钥和其他有关信息（在可信第三方处的体现为一个单独的线程，关闭掉该线程）

协议的流程图如下:

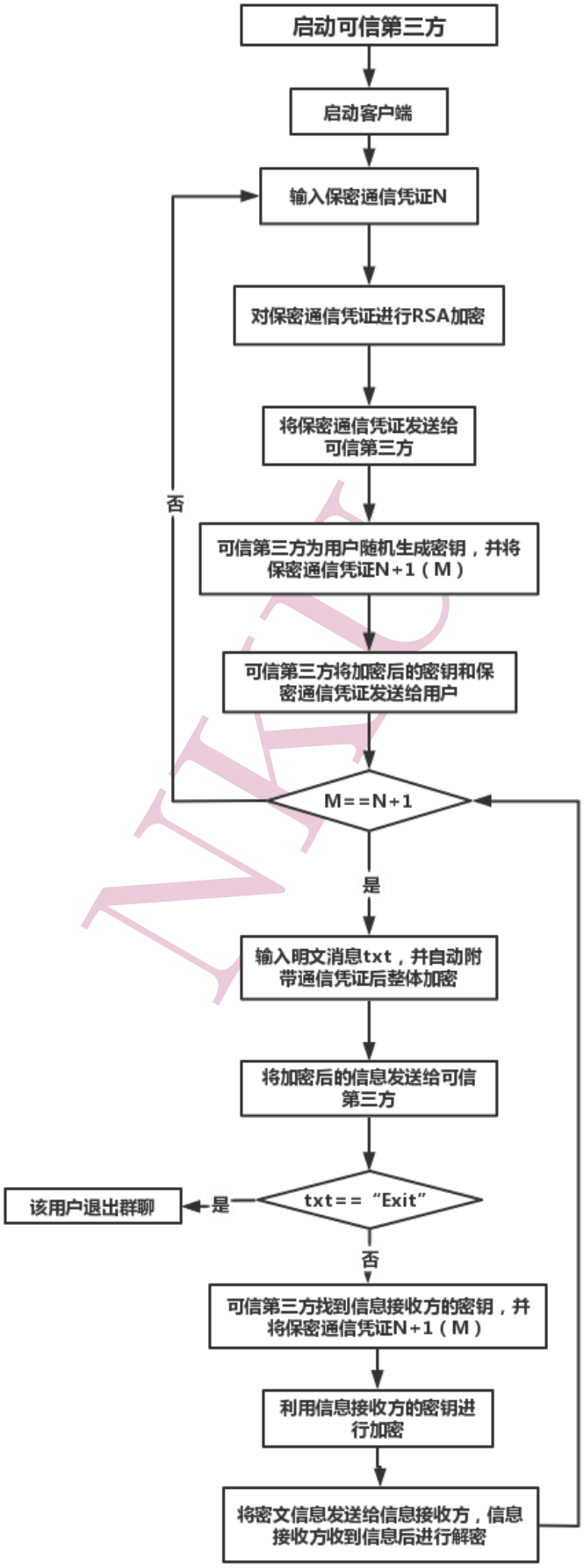

图 20: 协议的整体流程图 (简化版，但可以体现协议执行的整个过程)

2. 程序的使用步骤

当然，协议的步骤过于复杂是为了保证通信的绝对安全，本次群聊模式的保密通信可以抵抗中间人的劫持和伪造攻击，并且由于不同的客户端持有不同的 AES 密钥，也可以避免社会学攻击（例如，可避免客户端 A 利用自己的密钥去解密客户端 B 收到的加密数据包），但是，使用步骤却非常简单。

使用步骤如下:

1) 启用可信第三方（在客户端启动之前必须先启动可信第三方，否则无法为客户端分配密钥）

2) 启动客户端 A,B,C,D,E（最多为 1000 个），每个客户端的使用步骤均相同

3) 客户端启动后，首先需要输入保密通信凭证（0-128 之间的任意数字），此时即向可信第三方请求密钥

4) 在成功获得密钥后，即弹出“保密通信密钥成功，通信信道确定安全，下面开始进行通信交流!”的字段

5) 输入信息进行保密通信

6) 得到其他客户端发来的信息

7) 交替进行步骤 5-6（由于客户端的实现为双线程，故发送消息和接受消息互不干扰）

8) 想要退出保密通信时，输入“Exit”告知可信第三方，可信第三方也会将退出信息告知给其他客户端，至此，通信结束

（五） 不同部件要完成的功能

1) **可信第三方:**

A) 在客户端发出 Request 请求后，进行密钥的分发，将随机生成的密钥利用 RSA 加密后发送给客户端

B) 负责密钥的管理，维护一个存有所有用户密钥的密钥表，并将每个客户端单独开一个线程，存储该客户端的所有信息

C) 负责保密通信信息的处理，当收到客户端传来的信息后，首先进行解密，并根据所要发送的客户端（即信息的接收方，可能不止一个），在密钥表中查找对应密钥，并对收到的信息进行 AES 加密后发送

D) 进行信息的识别，当收到“Request”字段后，进行密钥的分配；当收到“Exit”字段后，删除该客户端对应的密钥以及其他所有信息（即删除该客户端对应的线程，序列号，以及密钥表中存储的该客户端的 AES 密钥等信息）

E) 负责日志的记录（时间戳，密钥分配信息，密钥管理信息，日志时间，群聊总人数，以及转发保密通信信息）

2) **客户端:**

A) 随机选择保密通信凭证（0-128）, 该通信凭证的作用是及时发现保密通信信息传输过程是否遭到中间人的篡改和伪造，使得通信更安全

B) 保密通信信息的发送

C) 保密通信信息的接受，接收信息后首先需要判断通信凭证是否遭到篡改

D) 日志的记录（只记录该客户端上的行为，包括时间戳，发送和接收到的信息以及通信状态是否安全）

四、 关键代码展示

由于 AES 和 RSA 加解密过程在之前的密码学实验中均已经完成并在实验报告中进行了详细的解释和说明，故本项目关键代码的展示中省去 AES 和 RSA 加解密细节的展示，只展示实现本协议的核心代码

(一) 时间戳

为了对保密通信进行日志的记录，需要在输出的时候，体现出具体时间，我们将时间戳封装为 timestamp() 函数，需要时间戳的地方调用该函数即可，时间戳的代码实现如下：

时间戳

```
char* timestamp()
{
    time_t timep;
    time(&timep);
    char tmp[256];
    strftime(tmp, sizeof(tmp), "%H:%M:%S", localtime(&timep));
    return tmp;
}
```

(二) 网络配置

我们借用可信第三方，在 TCP 协议的基础上，实现我们的保密通信协议，再进行信息交互之前，我们需要配置 wVersionRequested,socket,sockaddr_in 等，具体的配置信息如下，以客户端为例（可信第三方网络和客户端基本相同）：

配置网络

```
WORD wVersionRequested = MAKEWORD(1, 1);
WSADATA wsaData;
int error;  //WSAStartup成功返回0，否则为错误代码
error = WSAStartup(wVersionRequested, &wsaData);
if (error != 0)
{
    cout << timestamp() << "  WSAStartup Error:" << WSAGetLastError() << endl
        ;
    return 0;
}
// 1. 创建流式套接字
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s == INVALID_SOCKET)
```

```
{
    cout << timestamp() << "  Socket error:" << WSAGetLastError() << endl;
    return 0;
}

//2.链接服务器
sockaddr_in   addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(8888);
addr.sin_addr.s_addr = inet_addr("127.100.100.100");

int len = sizeof(sockaddr_in);
if (connect(s, (SOCKADDR*)&addr, len) == SOCKET_ERROR)
{
    cout << timestamp() << "  connect  error: " << GetLastError() << endl;
    return 0;
}
```

(三) 多线程

为了使得项目更完整，体验更佳，我们在多处使用多线程。在可信第三方，我们为每一个客户端单独开启一个线程，使得不同线程间互不干扰，且便于密钥的管理；在客户端，发送信息和接收信息分开，两个线程使得不存在阻塞现象，可以实现随时信息的发送以及随时信息的接受。

1. 可信第三方多线程

可信第三方为每一个客户端分配一个线程，线程数量会随着客户端的接入而不断增加，下面为可信第三方为客户端分配线程的实现：

可信第三方为每一个客户端单独开启一个线程

```
while (true)
{
    sockaddr_in addrClient;
    len = sizeof(sockaddr_in);
    // 4.接受成功返回与client通讯的Socket
    SOCKET c = accept(SS, (SOCKADDR*)&addrClient, &len);
    if (c != INVALID_SOCKET)
    {
        // 创建线程，并且传入与client通讯的套接字
        total[ans] = c;
        ans++;//记录当前用户的总数量
        HANDLE hThread = CreateThread(NULL, 0, ThreadFun, (LPVOID)c, 0, NULL)
            ;
        CloseHandle(hThread); // 关闭对线程的引用
    }
}

DWORD WINAPI ThreadFun(LPVOID lpThreadParameter)
```

```
{
    //一个线程对应一个客户端，存储该客户端的所有信息以及负责该客户端的保密通信和密钥分发
}
```

2. 客户端双线程

客户端的 main 函数本身为一个线程，我们将该线程用作发送信息的线程，并在 main 函数中新创建一个线程，用作处理该客户端的信息接收（即接受线程）

客户端双线程

```

//在main函数中创建接收消息的线程，并将套接字传递给接受信息的线程
CreateThread(NULL, 0, &ThreadRecv, &s, 0, NULL);

DWORD WINAPI ThreadRecv(LPVOID IpParameter)//接收消息的线程
{
    SOCKET cliSock = s;//获取客户端的SOCKET参数

    bool flag = false;
    while (true)
     {
       //.......此处为处理接收的信息，对消息的解密等
     }
    return 0;
}
```

(四) RSA 加解密

RSA 的加解密主要用于密钥传输阶段，可信第三方掌握客户端的 RSA 公钥，客户端掌握自己的私钥和公钥。RSA 加解密过程中，使用的参数如下：

用于 RSA 加解密的参数

```
Big_Int p("B0FA72FBB98B313503F335F2A0898DFB"), q("
    FD3D2105D2DD83B6B17F90F049694643");
Big_Int one(1);
Big_Int n = p * q, phi = (p - one) * (q - one);
//n.Print();
Big_Int e(10001);
Big_Int d = e.Inverse(phi);
Big_Int another_n("66
    F2BD4FB36D3466BC87D9D46C59698884BD256AD5F876ECC54E9B1D14E05F99");
```

1. RSA 加密

由于只有客户端掌握 RSA 算法的私钥，故只有客户端可以进行解密（也就是说只有可信第三方可以进行加密）。因此，RSA 的加密主要发生在可信第三方，可信第三方对于随机产生的 AES 密钥进行 RSA 加密后传输，RSA 加密的代码如下：

RSA 加密

```
cout << endl;
//key.Print();
Big_Int key0 = key.Exp(e, n);//使用对方公钥加密
cout << timestamp() << "    " << "利用RSA对密钥进行加密后传输,加密结果为: ";
key0.Print();
//cout << endl << "jiami" << endl;
strcat(sendDataa, t2.data());
```

2. RSA 解密

由于只有客户端掌握 RSA 算法的私钥，故只有客户端可以进行解密。因此，RSA 的解密发生在在客户端，客户端用来解密 KDC（密钥分配中心）发来的 AES 密钥，RSA 解密代码如下:

RSA 解密

```
Big_Int AES_key_Encryby_RSA(tk);
cout << timestamp() << "  利用RSA加密后的密钥:";
AES_key_Encryby_RSA.Print();
//cout << "AAAAAAAAAAAAAAAAAAAAAAAA" << endl;
Big_Int b_key = AES_key_Encryby_RSA.Exp(d, n);
cout << timestamp() << "  得到的密钥Hex形式为: ";
for (int i = 0; i < b_key.output.length(); i += 2) {
    string bit = b_key.output.substr(i, 2);
    aes_key[i / 2] = hexToInt(bit);
    cout <<bit << " ";
}
```

（五） AES 加解密

在本项目中，AES 算法主要用于对传输信息和通信凭证的加解密，采用的是 CBC 模式，利用 CBC 模式加解密的初始化向量如下:

初始化向量

```
array<int, 16> InitVector = { 0,1,0,1,0,1,0,1,0,1,1,0,0,1,0,1 };
```

该初始向量为随机生成，进一步保证了通信内容的安全性

1. AES 加密

AES 加密阶段主要发生在客户端发送消息（包括通信凭证和传输的消息）之前，以及可信第三方进行信息转发的时候，对明文消息进行 AES 的 CBC 模式加密，加密的核心代码如下:

AES 的 CBC 加密

```
array<int, 16> temp;
string CBC_group;
for (int index = 0; index <= data.length() / 16; index++) {
    //每一轮均需要先清空上一轮的密文
    CBC_group = "";
```

```
    for (int j = 0; j < 16; j++) {
        if (index * 16 + j < data.length())
            temp[j] = data[index * 16 + j] ^ CBC_vector[j];
        else
            temp[j] = 0 ^ CBC_vector[j];
    }
    keySchedual = Gen_subkeys(aes_key);
    array<array<int, 4>, 4> pTextBlock = Input_tran(temp);
    array<array<int, 4>, 4> cTextBlock = Encryption(pTextBlock, keySchedual);
    CBC_vector = Output_tran(cTextBlock);
    for (int k = 0; k < 16; k++) {
        CBC_group += intToHex(CBC_vector[k]);
    }
    strcat(sendData, CBC_group.data());
}
```

2. AES 解密

AES 的解密过程和 AES 的加密过程类似，具体区别在 AES 算法介绍部分呈现。AES 的解密主要出现在传输过程中，接受密文消息的部分，发生在客户端的接收线程以及可信第三方处理客户端传来信息的地方，对密文信息进行 AES 的 CBC 模式解密，解密的核心代码如下:

AES 的 CBC 解密

```
cout << timestamp() << "  " << "[收到的密文信息]  " << zhen << endl;
array<int, 16> temp;
for (int i = 0; i < zhen.length() / 32; i++) {
    string CBC_group = zhen.substr(i * 32, 32);
    for (int j = 0; j < 32; j += 2) {
        string bit = CBC_group.substr(j, 2);
        temp[j / 2] = hexToInt(bit);
    }
    keySchedual = Gen_subkeys(aes_key);
    array<array<int, 4>, 4> cTextBlock = Input_tran(temp);
    array<array<int, 4>, 4> pTextBlock = Decryption(cTextBlock, keySchedual);
    array<int, 16> output = Output_tran(pTextBlock);
    for (int k = 0; k < 16; k++) {
        output[k] = output[k] ^ CBC_vector[k];
        result += output[k];
    }
    CBC_vector = temp;
}
//每次加密完后, 重新初始化用于CBC解密的向量
CBC_vector = InitVector;
string strr = tt.substr(0, 20);
cout << timestamp() << "  " << "[对应的明文信息]  ";
for (int i = 0; i < 20; i++)
{
    if (strr[i] != '0')
```

```
        cout << strr[i];
    else
        break;
}
cout << result << endl;
```

（六） 密钥生成

当客户端向可信第三方请求分配 AES 密钥后，可信第三方随机生成客户端的 AES 密钥，生成密钥，打印密钥（十进制形式，十六进制形式，以及通过 RSA 加密后的形式）的代码如下：

密钥生成

```
Big_Int key(0);
key.GeneRandom(128);
//cout << "AAAAA" << endl;
cout << timestamp() << "    自动生成的密钥为: ";
for (int i = 0; i < key.output.length(); i += 2) {
    string bit = key.output.substr(i, 2);
    cout << bit << " ";
    aes_key_own[i / 2] = hexToInt(bit);
    aes_key[ans - 1][i / 2] = hexToInt(bit);
}
cout << endl;
cout << timestamp() << "    密钥十进制表示为: ";
for (int i = 0; i < key.output.length(); i += 2) {
    string bit = key.output.substr(i, 2);
    cout << aes_key_own[i / 2] << " ";
}
cout << endl;
//key.Print();
Big_Int key0 = key.Exp(e, n);//使用对方公钥加密
cout << timestamp() << "    " << "利用RSA对密钥进行加密后传输,加密结果为: ";
key0.Print();
//cout << endl << "jiami" << endl;
strcat(sendDataa, t2.data());
```

（七） 判断是否遭遇篡改或伪造攻击

在该协议中，我们通过用户自己选择的通信凭证来判断接收到的密文信息是否正确，在可信第三方收到数据包进行转发之前，将用户的通信凭证 N+1，用户收到信息后，首先判断通信凭证是否为预期的凭证，如果是，则说明未遭遇篡改和窃听，否则则意味着遭遇篡改，此时丢弃数据包，核心代码如下：

通过通信凭证判断是否遭遇篡改或伪造攻击

```
char t1 = t[3];
if (t1 - 1 != sN1[0]) {
    cout << timestamp()<<"  通信凭证匹配错误! " << endl;
```

```
}
else
{
    cout << timestamp()<<"  通信凭证正确匹配!" << endl;
}
string skey = t.substr(4, t.length());
```

（八） 群聊转发

群聊转发是本次保密通信协议实现的难点，为了确保用户不能利用自己的 AES 密钥去解密去解密其他用户的密文信息，我们给每个用户分配的 AES 密钥不同，而 AES 算法中，加密和解密密钥是相同的。故我们在群聊转发的时候，可信第三方在密钥表中查找信息接收方的密钥，并利用接收信息方的密钥为可信第三方得到的明文信息（此明文信息为可信第三方在接收到密文消息后，利用发送发的 AES 密钥对密文进行解密而得到）进行加密，之后再分别进行转发，实现代码如下：

保密通信协议群聊转发

```
for (int k = 0; k < total[k]; k++)
{
    //群聊转发不需要转发给信息发送方，当匹配到信息发送方的线程后，直接跳过
        continue
    if (total[k] == c)
        continue;
    string gogo = "00000000000000000000";
    string tempp = name;
    tempp += " Say:";

    char sendDataaa[512];
    for (int i = 0; i < 20; i++)
    {
        if (i < tempp.size())
        {
            sendDataaa[i] = tempp[i];
        }
        else
        {
            sendDataaa[i] = '0';
        }
    }

    string data = resultt;
    string tempppppp = name;
    tempppppp+=" Say:";
    for (int i = tempppppp.length(); i < 20; i++)
    {
        tempppppp += "0";
    }
```

```
    if (data == "")
        continue;
    array<int, 16> temp;
    string CBC_group;
    for (int i = 0; i <= data.length() / 16; i++) {
        CBC_group = "";
        for (int j = 0; j < 16; j++) {
            if (i * 16 + j < data.length())
                temp[j] = data[i * 16 + j] ^ buff[j];
            else
                temp[j] = 0 ^ buff[j];
        }
        keySchedual = Gen_subkeys(aes_key[k]);
        array<array<int, 4>, 4> pTextBlock = Input_tran(temp);
        array<array<int, 4>, 4> cTextBlock = Encryption(pTextBlock,
            keySchedual);
        buff = Output_tran(cTextBlock);
        for (int k = 0; k < 16; k++) {
            CBC_group+= intToHex(buff[k]);
        }
        /*strcat(sendDataaa, CBC_group.data());*/
        tempppppp += round.data();
    }
    const char* ppp = tempppppp.c_str();
    //cout << "加密后的结果: " << tempppppp << endl;
    sendRes=send(total[k], ppp, 512, 0);
    buff = InitVector;
    }
}
```

（九） 程序退出

客户端可以在任意时刻选择退出，当客户端输入“Exit”字段并发送给可信第三方，可信第三方通过 AES 解密后，若识别到“Exit”字段，则删除该客户端的所有信息（AES 密钥，名字等），并告知其他用户，“该客户端已经离开群聊!”，客户端在确认可信第三方收到自己的退出请求后，主动退出，代码如下：

程序退出

```
if(ste.substr(0,4) == "Exit")//判断客户端是否退出
{
    string str = "Exit: ";
    str += name;
    str += " leave the Chatting room!";
    const char* p = str.c_str();
    for (int i = 0; i < total[i]; i++)
    {
        if (total[i] != c)
        {
```

```
            send(total[i], p, strlen(p), 0);
        }
    }
    cout << timestamp() << "  " << str << endl;
    return 0;
}
```

（十） 格式转换

为了能够适配密码学实验中实现的 AES 和 RSA 算法，我们需要进行适当的格式转换。结合密码学实验中已经实现的 AES 和 RSA 算法，我们需要实现 16 进制字符串转换为对应的十进制数字以及将十进制数字转换为对应的十六进制字符串，代码如下：

十六进制字符串转为十进制数字

```
int hexToInt(string hex_str) {
    int r = 0;
    for (int i = 0; i < hex_str.length(); i++) {
        r *= 16;
        if (hex_str[i] >= '0' && hex_str[i] <= '9') {
            r += hex_str[i] - 48;
        }
        else if (hex_str[i] >= 'A' && hex_str[i] <= 'Z') {
            r += hex_str[i] - 55;
        }
        else
            r += hex_str[i] - 87;

    }
    return r;
}
```

十进制数字转换为十六进制字符串

```
string intToHex(int dec_int) {
    string r;
    while (dec_int) {
        int t = dec_int % 16;
        if (t < 10)
            r = char(t + 48) + r;
        else
            r = char(t + 55) + r;
        dec_int /= 16;
    }
    while (r.length() < 2) {
        r = '0' + r;
    }
    return r;
}
```

五、 实验结果演示

(一) 程序启动

程序启动包括可信第三方的启动以及客户端的启动，本协议最多支持同时启动 1000 个客户端，可以充分满足实验和现实需求；另外，需要注意的是，由于密钥分配中心承担密钥分配的任务，故启动客户端前需要启动密钥分配中心（即可信第三方）。

1. 可信第三方

可信第三方启动后，会持续监听是否有客户端的接入（利用 while（teur）循环实现），成功启动后的截图如下：

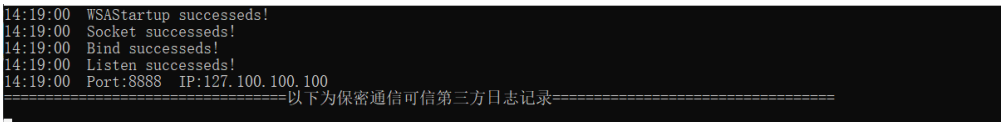

图 21: 启动可信第三方

2. 客户端

客户端启动的数量任意（大于等于 2 个小于 1000 个），按照需求启动即可，启动客户端后，首先需要输入用户名，再输入通信保密凭证，紧接着会向可信第三方请求密钥分配，可信第三方随机生成密钥后，利用客户端的 RSA 公钥进行加密，客户端收到加密的密钥后，利用自己的 RSA 私钥进行解密得到分配的密钥成功启动后的截图如下（本次演示为启动四个客户端）：

图 22: 启动客户端 1

图 23: 启动客户端 2

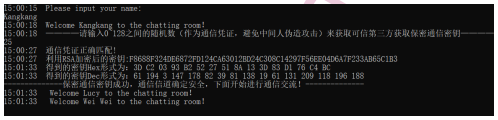

图 24: 启动客户端 3

图 25: 启动客户端 4

可以发现，再本协议中，可信第三方为不同的客户端分配的 AES 密钥均不相同！另外，客户端收到可信第三方分配的密钥后，利用 RSA 私钥解密的时间大约为 90s。(此部分时间需等待)

(二) 密钥分配

密钥分配的任务由密钥分配中心（即可信第三方）承担，当客户端向可信第三方请求密钥后，密钥分配中心随机生成 128 位的大整数当作客户端的密钥，利用对应的 RSA 公钥进行加密后发送给客户端，同时，密钥分配中心还会记录中当前聊天室中的用户数量，下面为可信第三方日志记录的截图：

图 26: 可信第三方密钥分配

可以看出，该日志记录中还包含时间戳，记录为每个客户端分配密钥的具体时间。

(三) 保密通信

1. 可信第三方

所有客户端的保密通信均需要发送给可信第三方，可信第三方将密文信息解密后，再用信息接收方的 AES 密钥对信息进行加密，加密后发送给信息接收方，可信第三方中记录中所有信息发送和接受的时间以及信息的明文和密文，下面为可信第三方的保密通信截图:

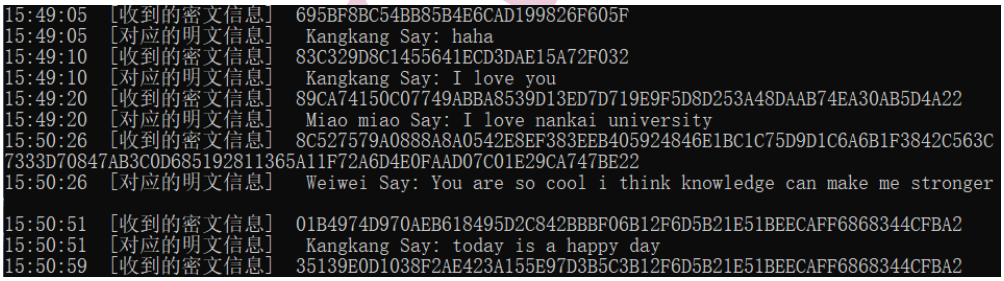

图 27: 可信第三方保密通信信息处理

可以看到，可信第三方接受所有客户端的信息，并对信息进行针对性的转发（转发给信息发送方要发送的对象）；并且不同长度的明文信息对应的密文长度不同，明文长度以 16 位为界进行分组，对每一组采用 AES 的 CBC 加密模式进行加密，最后一组长度不够 16 位补 0

2. 客户端

客户端可以进行明文信息的发送，以及接受发送给自己的密文信息，并利用自己的 AES 私钥进行解密，下面为客户端（以其中两个为例）在保密通信中的截图:

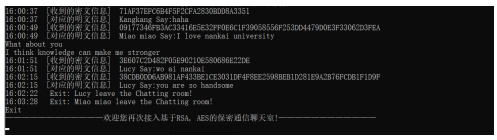

图 28: 客户端 1 保密通信截图

图 29: 客户端 2 保密通信截图

（四） 程序退出

在该项目中，客户端可以随时选择退出，退出时输入“Exit”，当可信第三方收到该字段后，客户端退出，可信第三方销毁该客户端的所有信息（保密通信密钥，客户端的名字等），并进行日志的记录，记录该客户端什么时间退出的程序，并告知其他客户端，该客户端已经退出，不要再向其发送消息，可信第三方的截图如下：

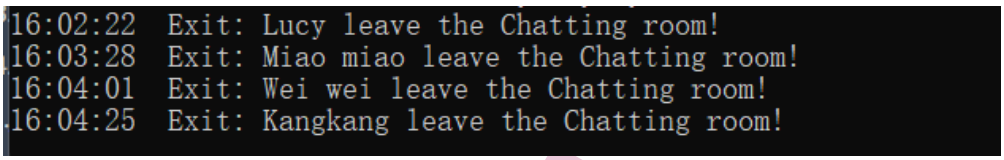

图 30: 可信第三方记录客户端退出

客户端输入“Exit”后，在得到可信第三方的回复后，停留 1 秒后，自动退出，下面为客户端退出程序的截图：

图 31: 客户端退出

六、 技术难点与解决方案

1) **协议如何抵抗中间人攻击?**

用户在启动服务器后，首先会自己选择一个通信凭证。用户每次输入要发送的消息后，会自动带上通信凭证，并利用 AES 算法将所要发送的消息和通信凭证一起进行加密，可信第三方在接收到密文信息后，会将通信凭证加 1，之后再进行消息的转发，接收方收到消息后，会判断该通信凭证是否为预期通信凭证，若是预期通信凭证，则证明信息在传递的过程中未遭到篡改；若不是预期的凭证，则意味着数据包在传递的过程中遭遇中间人的篡改或伪造。通过增添并验证通信凭证，可以成功抵抗中间人攻击。

2) **如何保证用户无法用自己的 AES 私钥解密其他用户的密文数据?**

和普通的单聊模式相比，本协议支持群聊，即存在多个用户。如果不同用户持有相同的密钥，则可以实现利用自己的密钥解密其他用户的密文数据；本协议中的子协议（密钥分配协议）确保不同的用户持有不同的密钥，不同用户之间胡不知晓对方的密钥，故可以保证用户无法用自己的密钥去解密其他用户的密文数据，使得通信更安全。

3) **如何实现持有不同 AES 密钥的用户之间的互相通信?**

在成功解决问题 2 后，又引入一个新的问题，如何实现持有不同 AES 密钥的用户之间的互相通信? 本协议通过引入可信第三方，来实现持有不同 AES 密钥的用户之间的互相通信。可信第三方持有所有用户的密钥，用户不管将消息发给谁，均需要发送给可信第三

方，再由可信第三方进行转发给目标用户。可信第三方收到用户发来的密文信息后，先在密钥表中查找该用户的密钥，之后利用该用户的密钥进行解密得到明文，再将该明文利用信息接收发的密钥（在密钥表中查找）进行加密，加密后发送给信息接收方。本协议通过引入可信第三方，来实现持有不同密钥的用户之间的互相通信。

4) **当用户数量过多时，如何实现密钥管理?**

由于该协议支持群聊模式，不同的用户持有不同的 AES 加解密密钥，当用户数量过多时，密钥管理又成为一个新的问题。可信第三方通过给每一个用户单独开启一个线程，该线程中存有该用户的所有信息，包括该用户的 AES 密钥；可信第三方还会维护一个密钥表，该密钥表存有所有用户的密钥信息，当用户接入后，为用户分配密钥，当用户离开后，从密钥表中删去该用户的密钥，并删除该用户的所有其他信息（即直接 kill 掉该用户所占有的线程）。

5) **如何进行密钥分配?**

本次密钥分配引入可信第三方作为密钥分配中心（Key Distribution Center,KDC）, 每当用户接入群聊后，向 KDC 发送密钥 Request，并附带唯一通信凭证 N（此标志符防止密钥传输的过程中遭遇中间人的劫持或篡改），KDC 接到请求后，随机生成一个 128 位的大整数作为该用户的密钥，并将 N+1(N'), Request 以及密钥通过 RSA 算法进行加密，加密后发送给该用户，该用户接收到密钥后，首先进行 RSA 解密，判断接收到的 N' 是否等于 N+1, 若等于，则说明没有遭到中间人伪造攻击，将该密钥保留起来用作保密通信过程中 AES 加密和解密密钥; 若不等于，则意味着遭到中间人攻击或含有密钥的数据包在传输过程中损毁，此时重新请求 KDC，直到得到的密钥可用为止。（请求次数超过 5 次则意味着该用户存在被监视的风险，直接中断请求）。更具体的密钥分配信息见上文密钥分配协议设计部分。

七、 参考文献

[1] TCP 标准文档，https://www.rfc-editor.org/rfc/rfc793

[2] 杨波. 现代密码学（第四版）[M]. 北京：清华大学出版社，2003.

[3] Schneier B 著. 吴世忠，祝世雄，张文证等译. 应用密码学协议、算法与 C 源程序. 北京：机械工业出版社，2000.

[4] Adams C, Lloyd S 著。冯登国等译. 公开密钥基础设施 ~ 概念、标准和实施. 北京：人民邮电出版社，2001.

[5] 赵战生，冯登国，戴英侠，荆继武编著. 信息安全技术浅谈. 北京：科学出版社，1995 年.

[6] 王育民, 刘建伟. 通信网的安全—-理论与技术 [M]. 西安: 西安电子科技大学出版社, 1999.

[7] 胡予濮，张玉清，肖国镇. 对称密码学 [M]. 北京：机械工业出版社，2002.

[8] 朱文余，孙琦. 计算机密码应用基础 [M]. 北京：科学出版社，2000.

[9] 杨波. 网络安全理论与应用 [M]. 北京：电子工业出版社，2002.

[10] Arto Salomaa. Public-Key Cryptography, Second Edition[M]. Springer-Verlag,1996.

[11] Space communications protocol specification(SCPS)-Security Protocal(SCPS-SP). Blue Book. MAY 1999.

[12] Housley R,Fond W, Polk T etal. Internet X.509 Public Key In frastructure Certificate and CRL Profile. RFC2459,PKIX Working Group ,1999

[13] Song Y Yan. Number Theory for Computing[M], Second Edition[M]. Springer-Verlag,2002.

[14] Joan Danmen, Vincent Rijmen. AES Proposal: Rijndael[M]. AES algorithm submisson, September 3,1999,AES home page: http://www.nist.gov/aes

[15] O Goldreich. Foundation of Crryptography: Basic Tools[M]. Cambridge University Press, Cambridge,2004.