

# 网络空间安全学院（密码学院）

2023-2024 学年度第 2 学期

J00717 密码工程课程设计 课程论文

论文题目： 安全通信协议设计与分析

学生学号及姓名： **20213006839 甄五四**

学生所在班级： 信息安全(密码学方向)理科实验  
班 **2021**

论文提交时间： **2024 年 6 月 15 日**

授课教师： 孙敬张

教师评阅：

成绩： \_\_\_\_\_

教师签名： \_\_\_\_\_

## 摘要

安全通信协议在保护信息传输过程中起着重要作用。本文旨在设计、实现和分析一种安全通信协议，以确保通信的机密性、完整性和身份认证。设计的协议涵盖了通信双方的身份认证、密钥协商和消息传输过程，充分考虑了机密性、完整性和抗重放攻击等安全属性。本文所设计安全通信协议基于 RSA、SHA256、3DES 以及 Diffie-Hellman 实现，基于 RSA 实现数字签名、Diffie-Hellman 实现密钥交换，3DES 对通信数据加密。其中还增加了时间戳、SHA256，随机数等，确保通信协议的安全性。既可以实现方便高校通信，也可以实现通信数据的完整性、保密性。最后使用 Python 实现了该设计的安全通信协议，确保协议的正确性和可行性。同时，本文使用密码学工具 WireShark，对设计的协议进行了安全性分析，检测潜在的漏洞和攻击。本文还比较了设计的协议与现有的安全通信协议 TLS/SSL 之间的异同，并讨论其优缺点。最后在实验环境中演示了设计的协议，并测试其性能和安全性，记录和分析了实验结果，包括协议通信的延迟、安全性等指标。

**关键字：**安全协议、身份认证、密钥协商、RSA、Diffie-Hellman

## Abstract

The safety communication protocol plays a crucial role in safeguarding the information transmission process. This article aims to design, implement, and analyze a secure communication protocol to ensure confidentiality, integrity, and identity authentication in communication. The designed protocol covers identity authentication of both communicating parties, key negotiation, and message transmission processes, fully considering security attributes such as confidentiality, integrity, and resistance against replay attacks. The designed secure communication protocol is based on RSA, SHA256, 3DES, and Diffie-Hellman implementation, utilizing RSA for digital signatures, Diffie-Hellman for key exchange, and 3DES for encrypting communication data. It also incorporates elements like timestamps, SHA256, random numbers, etc., to ensure the security of the communication protocol. It can facilitate convenient communication for universities while ensuring data integrity and confidentiality. The protocol has been implemented using Python to ensure its correctness and feasibility. Furthermore, cryptographic tools like Wireshark were used to perform security analysis on the designed protocol, detecting potential vulnerabilities and attacks. A comparison between the designed protocol and the existing security communication protocol TLS/SSL is also provided, discussing their respective advantages and disadvantages. Finally, the protocol was demonstrated in an experimental environment, testing its performance and security. The experiment results, including communication latency and security metrics, were recorded and analyzed.

**Keywords:** Security Protocol; Identity Authentication; Key Negotiation; RSA; Diffie-Hellman

# 目录

1	前言 .....	5
2	安全协议基础理论 .....	5
2.1	安全协议的密码学基础 .....	5
2.2	密码体制 .....	6
2.2.1	对称密码体制 .....	6
2.2.2	公钥密码体制 .....	7
2.3	数字签名 .....	9
2.4	消息认证和 Hash 函数 .....	10
3	系统设计 .....	10
4	系统实现 .....	13
4.1	RSA 公私钥生成 .....	13
4.2	RSA 的签名验签和加解密 .....	14
4.3	通信中基本函数 .....	15
4.4	客户端通信 .....	17
4.5	服务器端通信 .....	18
5	测试结果 .....	19
5.1	运行 .....	19
5.2	安全性分析 .....	21
6	系统特色及与现有的安全通信协议比较 .....	23
6.1	系统特色 .....	23
6.2	与现有的安全通信协议比较 .....	24
6.2.1	TSL/SSL 介绍 .....	24
6.2.2	本课设设计协议与 TLS/SSL 异同点 .....	24
7	结论 .....	25
	致谢 .....	26
	参考文献 .....	27

# 1 前言

在当今数字时代，保障信息在网络传输过程中的安全性已成为一项至关重要的任务。随着互联网的普及，越来越多的敏感信息在网络上传输，如金融数据、个人隐私信息和商业机密等。这些信息在传输过程中面临着诸多安全威胁，包括窃听、数据篡改、身份伪造以及中间人攻击等。因此，设计一种高效、安全的通信协议，以确保信息的保密性、完整性和真实性，对于保护用户隐私和维护数据安全具有重要的意义。传统的安全通信协议，如 SSL/TLS，尽管在大多数情况下能够提供良好的安全保障，但其实现过程复杂且资源消耗较大，对于某些轻量级应用场景而言并不适用。为了满足这些应用场景的需求，本文提出了一种基于 RSA、DES 和 Diffie-Hellman 算法的轻量级双向认证协议。该协议不仅能够实现通信双方的身份认证，还能够通过 Diffie-Hellman 密钥交换算法生成安全的用于 3DES 对称加密的密钥，以及通过 SHA256 计算 Hash 值保证通信内容的机密性和完整性。此外，协议中加入了时间戳机制，有效防止重放攻击，提升了整体安全性。

本文的结构安排如下：首先，介绍 RSA、DES 和 Diffie-Hellman 等算法的基本原理及其在安全通信中的应用。接着，详细描述协议的设计，包括各个阶段的具体步骤和安全措施。随后，通过实际实现和实验验证，评估协议的性能和安全性。最后，总结本文的工作，并提出未来可能的改进方向。通过本文的研究，旨在为轻量级应用场景下的安全通信提供一种高效、可靠的解决方案，保障数据在网络传输过程中的安全性和完整性。希望本课设的工作能够为实际应用中的数据的安全传输提供有力的技术支持和理论依据。

## 2 安全协议基础理论

### 2.1 安全协议的密码学基础

安全协议运行在计算机通讯网或分布式系统中，为安全通信提供了基石。所谓协议就是多个参与者为了完成某项特定的任务而采取的一系列步骤。第一，协议自始至终是有序的过程，每一个步骤必须依次执行。第二，协议至少需要两

个参与者。第三，通过执行协议必须能够完成某项任务。安全协议运行在计算机通信网或分布式系统中，运用密码算法与协议逻辑实现密钥分配、身份认证、信息保密和电子交易安全等目的。安全协议的定义是：两方或多方参与者为获得某种特定的安全目标而执行的一个分布算法，此分布算法是一个确定的动作序列。这些动作主要发送和接收使用密码功能（如加密、签名、Hash 函数等）构造的消息。安全协议可用于保障计算机网络信息系统中秘密信息的安全传递与处理，用来保证网络用户能够安全、方便、透明地使用系统中的密码资源。本课设的研究中涉及到数字签名、认证协议、密钥分配等内容，下面就本文涉及的内容讨论如下。

2.2 密码体制

密码体制是安全协议的基础，密码方法可以隐藏和保护需要保密的消息。加密与解密过程如图 2-1 所示。

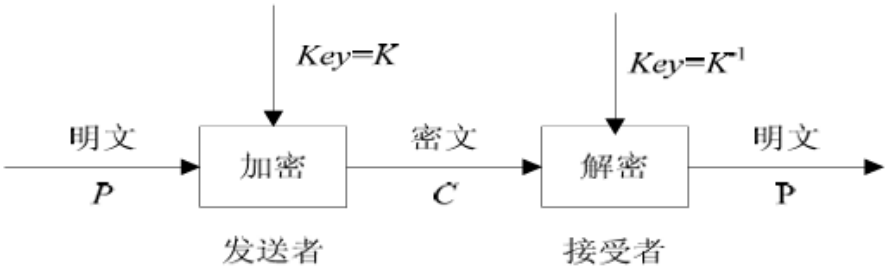


图 2.1 加密与解密模型

2.2.1 对称密码体制

对称密码体制是加密和解密使用相同密钥的密码系统。对数据进行加密的对称密钥体制如图 2-2 所示。

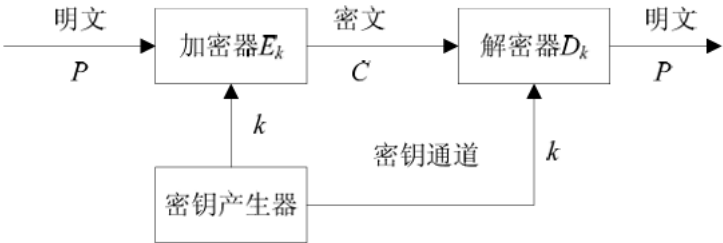


图 2.2 对称密钥密码体制

对称密钥密码体制的优点包括加解密速度快和安全强度较高，因此广泛用于数据加密。然而密钥交换困难：在进行安全通信前，需要以安全方式交换密钥，对于相距较远的用户，这可能非常困难甚至无法实现。本课设首先通过身份认证以及密钥交换来共享对称密钥，最后通过 3DES 对传输数据进行加密。3DES 指的是密钥长度为 24 个字节，该算法的加解密过程分别是对明文/密文数据进行三次 DES 加密或解密，得到相应的密文或明文。假设  $E_k$  和  $D_k$  分别表示 DES 的加密和解密函数，P 表示明文，C 表示密文，那么加解密的公式如下：加密： $C = E_{k_3}(D_{k_2}(E_{k_1}(M)))$ ，即对明文数据进行，加密—>解密—>加密的过程，最后得到密文数据；解密： $M = D_{k_1}(E_{k_2}(D_{k_3}(C)))$ ，即对密文数据进行，解密—>加密—>解密的过程，最后得到明文数据；其中：K1 表示 3DES 中第一个 8 字节密钥，K2 表示第二个 8 字节密钥，K3 表示第三个 8 字节密钥，K1、K2、K3 决定了算法的安全性，若三个密钥互不相同，本质上就相当于用一个长为 168 位的密钥进行加密。多年来，它在对付强力攻击时是比较安全的。

### 2.2.2 公钥密码体制

公钥密码体制，也被称为非对称密码体制，是一种加密通信的方法，由 Whitfield Diffie 和 Martin Hellman 在 1976 年首次提出。与对称密码体制不同，公钥密码体制使用一对密钥，即公钥和私钥，来进行加密和解密。公钥可以安全地共享给任何人，而私钥则必须保密。这种体制的安全性依赖于某些数学难题，如大整数分解或离散对数问题，这些难题在计算上非常困难，从而保护了私钥的安全。在公钥密码体制中，发送者使用接收方的公钥来加密信息，而接收者则使用自己的私钥来解密信息。这种方法使得密钥管理更加简单，特别是在多用户环境中，因为每个用户只有一对密钥，而不是与每个通信方都有一组共享密钥。这意味着对于 n 个用户，总共只需要 n 对密钥，而不是对称密码体制所需的  $n(n-1)/2$  个密钥。然而，公钥密码体制的加解密过程通常比对称密码体制慢，因为它们涉及更复杂的数学运算。因此，在实际应用中，公钥密码体制通常用于加密小量的数据，如用于安全传输的对称密钥，或者用于数字签名以验证数据的完整性。

和来源。对于大量数据的加密，通常会使用更快捷的对称密码体制，并结合公钥密码体制来安全地分发对称密钥。

### (1) Diffie-Hellman 方案

Diffie-Hellman 方案是第一个公钥密码系统，直至今天仍在广泛使用。该方案可供两个用户间生成共享密钥。Diffie-Hellman 方案基于离散对数问题，即给定  $n, g$  和素数  $p$ ，试图得到  $k$ ，满足  $n = g^k \bmod p$ 。当  $p$  为大素数时，这个问题是可解的，但随着  $p$  增大计算难度呈指数级增长。在 Diffie-Hellman 方案中，所有的用户共享一个公共模  $p$  和一个公共  $g$ 。 $g$  不等于 0、1 和  $p-1$ 。每个用户选择一个私钥  $K-1$ ，并且计算对应的公钥  $K$ 。当两个用户需要通信时，每个加密者使用自己的私钥加密对方的公钥，然后把计算结果作为共享密钥  $S$ 。因为用户共享一个密钥  $S$ ，所以 Diffie-Hellman 方案是对称密钥交换协议的实例。基于解离散对数问题是计算不可行的假设，从公钥计算出对应的私钥也是计算不可行的。在实际应用中，应该取得很大（数百比特）以满足离散对数问题的条件假设。

### (2) RSA 方案

RSA 算法是一种非对称加密算法，由 Ron Rivest、Adi Shamir 和 Leonard Adleman 在 1977 年提出。它基于数论中的模运算和大的素数因子分解的困难性。RSA 算法可以用于加密消息以及数字签名的生成和验证。

RSA 算法的关键步骤如下：

选择两个大的随机素数  $p$  和  $q$ 。计算  $n = pq$ ，其中  $n$  是模数。计算欧拉函数  $\Phi(n) = (p-1)(q-1)$ 。选择一个与  $\Phi(n)$  互质的整数  $e$  作为公钥指数，通常选择一个较小的数，如 65537。计算私钥指数  $d$ ，使得  $e * d \equiv 1 \pmod{\Phi(n)}$ ，即  $d$  是  $e$  关于模  $\Phi(n)$  的乘法逆元。公钥是  $(e, n)$ ，私钥是  $(d, n)$ 。

加密过程：明文  $m$  是一个整数，密文  $c$  计算为  $c \equiv m^e \pmod{n}$ 。

解密过程：密文  $c$  进行解密。明文  $m$  恢复为  $m \equiv c^d \pmod{n}$ 。

RSA 算法还可以用于数字签名：签名者使用自己的私钥  $(d, n)$  对消息的哈希值进行加密，得到签名。验证者使用签名者的公钥  $(e, n)$  对签名进行解密，并将解密后的哈希值与原始消息的哈希值进行比较。如果两个哈希值相同，则签名有效。RSA 算法的一个重要特性是它的公钥和私钥可以互换使用，这意味着可以用



私钥进行加密，公钥进行解密，或者用公钥进行加密，私钥进行解密。这种特性使得 RSA 可以用于加密和数字签名的双重目的。在实际应用中，为了同时提供机密性和认证，通常会使用一种称为数字信封的技术。发送者首先使用对称密钥加密消息，然后使用接收者的公钥加密对称密钥。接着，发送者使用自己的私钥对消息的哈希值进行签名。接收者首先使用发送者的公钥验证签名，然后使用自己的私钥解密对称密钥，最后使用解密的对称密钥解密消息。

使用公钥系统进行消息来源不可否认的技术称为数字签名。由于只有私钥的持有者能够生成有效的签名，因此一旦消息被签名，其他人无法伪造。这提供了消息来源的不可否认性。

## 2.3 数字签名

数字签名是网络通信和网络安全的一种非常特殊的密码认证形式，用于验证消息的来源和完整性，确保信息的不可否认性。它在网络通信和网络安全中扮演着关键角色，特别是在身份认证、数据完整性保护和不可否认性等方面。数字签名在密钥分配、认证协议和电子商务系统中尤为重要。数字签名的种类繁多，包括通用数字签名、仲裁数字签名、不可否认签名、代理签名、盲签名、公平盲签名、群签名、一次性签名、门限签名等。这些不同类型的数字签名适用于各种不同的应用场景。数字签名的实现依赖于公开密钥算法，它生成一段独特的数字串，这段数字串与特定的消息和签名者的私钥相关联，他人无法伪造。一个数字签名算法通常包括两个部分：秘密的签名算法和公开的验证算法。签名者使用私钥和签名算法对消息进行签名，而任何人都可以使用签名者的公钥和验证算法来验证签名的真实性。验证算法会根据给定的签名输出“真”或“假”，以判断签名是否有效。数字签名方案的安全性基于一些数学难题，包括：大整数的因子分解问题，如 RSA 和 Rabin 体制；有限域上的离散对数问题，这是目前最广泛应用的问题，包括 ElGamal 方案、DSA 方案等；椭圆曲线问题，这是一个研究热点，与离散对数问题相对应，它提供了更快的计算速度和更高的实用性，但在安全参数的选择和某些问题的研究上较为复杂。在实际应用中，数字签名的实现可能基于不同的体制，而 RSA 体制是其中之一。RSA 体制因其良好的安全性和广泛的应用而成为数字签名的常用选择。本课设主要基于 RSA 体制来实现数字签名。

## 2.4 消息认证和 Hash 函数

Hash 函数是一种将任意长度的消息映射为某一固定长度的消息摘要的函数，其目的是产生文件、消息或其他数据块等需认证数据的“指纹”。又称为哈希函数、杂凑函数、散列函数、杂凑算法、消息摘要算法。Hash 函数应用很广泛，可应用于数字签名中，它能够提高数字签名的速度、能够破坏数字签名算法的某种数学结构（比如同态结构）、能够将签名变换和加密变换分开来，用对称密码体制实现保密，而用公钥密码体制实现数字签名。Hash 函数还可用于消息的完整性检测、消息的非否认检测等方面。除此之外，Hash 函数还可用于安全协议的设计、分析和应用等方面。典型的 Hash 函数包括 MD2、MD4、SHA256、SHA-1、HMAC、Rabin 等。本课设选择 SHA256 来实现 Hash 计算。

## 3 系统设计

安全通信协议通信总体流程如图 3.1，接下来详细介绍各部分设计。

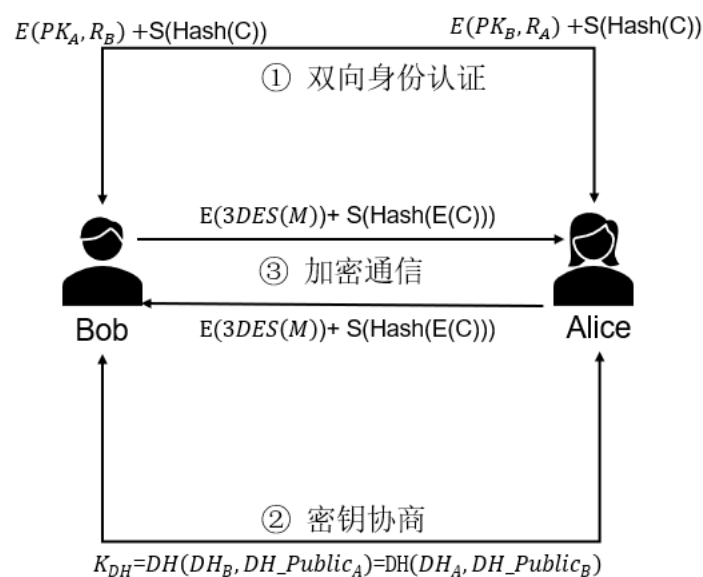


图 3.1 安全通信协议

一、首先进行系统初始化，生成服务器和客户端所需公私钥，建立连接：

1. RSA 密钥生成：服务器和客户端各自生成一对 RSA 公私钥对。将各自的公钥通过安全渠道传递给对方，存储为 .pem 文件。

2. 服务器启动：输入服务器 BobServer 的 IP 地址和聊天端口号，监听端口

输入，等待客户端连接。

3. 客户端连接：客户端输入服务器端 IP 地址和端口号，连接到服务器。

二、接下来就是通信双方进行双向身份认证：

客户端认证：

1. 生成一个随机数 $R_A$ 。
2. 用服务器的 RSA 公钥 $PK_B$ 加密 $R_A$ ： $C_A = RSA\_Encrypt(PK_B, R_A)$ 。
3. 计算  $C_A$  的 SHA256 值： $SHA256\_C_A = SHA256(C_A)$ 。
4. 用 自 己 的 RSA 私 钥  $SK_A$  对  $SHA256\_C_A$  签 名： $Sig_A = RSA\_Sign(SK_A, SHA256\_C_A)$ 。
5. 生成时间戳 $T_A$ 。
6. 将 $T_A$ 、 $Sig_A$  和 $C_A$ 发送给服务器。

服务器端认证：

1. 生成一个随机数 $R_B$ 。
2. 用服务器的 RSA 公钥 $PK_A$ 加密 $R_B$ ： $C_B = RSA\_Encrypt(PK_A, R_B)$ 。
3. 计算  $C_B$  的 SHA256 值： $SHA256\_C_B = SHA256(C_B)$ 。
4. 用 自 己 的 RSA 私 钥  $SK_B$  对  $SHA256\_C_B$  签 名： $Sig_B = RSA\_Sign(SK_B, SHA256\_C_B)$ 。
5. 生成时间戳 $T_B$ 。
6. 将 $T_B$ 、 $Sig_B$  和 $C_B$ 发送给服务器。

服务器验证客户端：

1. 接收到 $T_A$ 、 $Sig_A$  和 $C_A$ 后，校验时间戳 $T_A$ 是否有效。
2. 用客户端公钥 $PK_A$ 验签： $SHA256\_C_A = RSA\_Verify(PK_A, Sig_A)$ 。
3. 计算 $C_A$ 的 SHA256 值并比对： $SHA256\_C_A == SHA256\_C_A'$ 。
4. 如果一致，则客户端合法，否则关闭连接。

客户端验证服务器：

1. 接收到  $T_B$ 、 $Sig_B$  和 $C_B$ 后，校验时间戳 $T_B$ 是否有效。
2. 用服务器公钥 $PK_B$ 验签： $SHA256\_C_B = RSA\_Verify(PK_B, Sig_B)$ 。
3. 计算 $C_B$ 的 SHA256 值并比对： $SHA256\_C_B == SHA256\_C_B'$ 。
4. 如果一致，则服务器合法，否则关闭连接。

三、身份验证合法后，进行 Diffie-Hellman 密钥协商：

客户端发送 DH 参数：

1. 客户端生成 DH 参数组  $DH_{Group}$ 。
2. 生成时间戳  $T_{DH}$ 。
3. 将  $T_{DH}$  和  $C_{DH}$  发送给服务器。

服务器接收 DH 参数并生成 DH 密钥：

1. 校验时间戳  $T_{DH}$  是否有效。
2. 得到  $DH_{Group}$ 。
3. 生成服务器端的 DH 密钥对  $(DH_B, DH\_Private_B, DH\_Public_B)$ ， $DH_B$  是根据整数  $DH_{Group}$  自动选择素数和本原根并生成 DHE 对象。

客户端发送 DH 公钥：客户端生成 DH 密钥对  $(DH_A, DH\_Private_A, DH\_Public_A)$ ，这里仅输出 DH 密钥对的前 20 位。将  $DH\_Public_A$  发送给服务器， $DH_A$  是根据整  $DH_{Group}$  自动选择素数和本原根并生成 DHE 对象。

服务器计算共享密钥并发送 DH 公钥：服务器接收  $DH\_Public_A$  后，计算共享密钥  $K_{DH} = DH\_FinalKey(DH_B, DH\_Public_A)$ 。将  $DH\_Public_B$  发送给客户端。

客户端计算共享密钥：客户端接收  $DH\_Public_B$  后，计算共享密钥  $K_{DH} = DH\_FinalKey(DH_A, DH\_Public_B)$ 。

四、3DES 加密通信：

建立 3DES 密钥：双方分别取共享密钥  $K_{DH}$  的前 24 位作为 3DES 密钥  $3DES_{Key} = K_{DH}[0:24]$ 。

通信过程：

发送线程：

1. 客户端输入消息  $M$
2. 用  $3DES_{Key}$  加密消息：  $3DES\_M = 3DES\_Encrypt(3DES_{Key}, M)$ 。
3. 用服务器的 RSA 公钥  $PK_B$  加密  $3DES\_M$ ：  $C_M = RSA\_Encrypt(PK_B, 3DES\_M)$ 。
4. 计算  $C_M$  的 SHA256 值：  $SHA256\_C_M = SHA256(C_M)$ 。

5. 用自己的 RSA 私钥  $SK_A$  对  $SHA256_C_M$  签名： $Sig_M = RSA\_Sign(SK_A, SHA256_C_M)$ 。
6. 生成时间戳  $T_M$ 。
7. 将  $T_M$ 、 $Sig_M$  和  $C_M$  发送给服务器。

接收线程：

1. 服务器接收到  $T_M$ 、 $Sig_M$  和  $C_M$ 。
2. 校验时间戳  $T_M$  是否有效。
3. 用客户端的 RSA 公钥  $PK_A$  验签： $SHA256_C_M' = RSA\_Verify(PK_A, Sig_M)$ 。
4. 计算  $C_M$  的 SHA256 值并比对： $SHA256_C_M == SHA256_C_M'$ 。
5. 用服务器的 RSA 私钥  $SK_B$  解密  $C_M$  得到  $3DES_M$ 。
6. 用  $3DES_{Key}$  解密  $3DES_M$  得到明文  $M$ 。

终止通信：

1. 客户端发送 quit 作为结束标志。
2. 服务器接收到 quit 后，结束接收线程。
3. 服务器发送 quit 作为结束标志。
4. 客户端接收到 quit 后，结束发送线程。

## 4 系统实现

### 4.1 RSA 公私钥生成

本系统通过调用 Crypto 中 RSA 函数来实现 RSA 公私钥的生成，并保存为 pem 格式

```
# coding:gbk
from Crypto import Random
from Crypto.PublicKey import RSA
def generate_and_store_keys(identity):
    random_generator = Random.new().read
    # 随机数生成器
    # 生成用于 RSA 加密的公私钥对
    rsa_encryption = RSA.generate(2048, random_generator)
    private_pem_encryption = rsa_encryption.exportKey()
    public_pem_encryption = rsa_encryption.publickey().exportKey()
```

```

with open(f'RSA_Private{identity}.pem', 'wb') as f:
    f.write(private_pem_encryption)
with open(f'RSA_Public{identity}.pem', 'wb') as f:
    f.write(public_pem_encryption)
# 生成 Alice 的公私钥对
generate_and_store_keys('Alice')
# 生成 Bob 的公私钥对
generate_and_store_keys('Bob')

```

## 4.2 RSA 的签名验签和加解密

调用 Crypto 库中 RSA 的 SV 和 ED 操作来实现签名，验证签名，加密和解密，在签名的时候对数据的 SHA256 码进行签名，可以加快签名验签速度，还可以对数据进行完整性检验。

签名函数 `rsa_sign()`

```

def rsa_sign(data, privatePemPath):
    try:
        # 读取私钥文件
        with open(privatePemPath, 'rb') as private_key_file:
            pri_key = RSA.import_key(private_key_file.read()) # 导入私钥
            signer = pss.new(pri_key) # 创建 PSS 签名对象
            hash_obj = SHA256.new(data.encode('utf-8'))
            signature = base64.b64encode(signer.sign(hash_obj))
            return signature
    except Exception as e:
        print('签名失败:', e)
        return None

```

签名验证函数 `ras_verify()`

```

def rsa_verify(signature, data, publicPemPath):
    try:
        # 读取公钥文件
        with open(publicPemPath, 'rb') as public_key_file:
            pub_key = RSA.import_key(public_key_file.read())
            hash_obj = SHA256.new(data.encode('utf-8'))
            verifier = pss.new(pub_key)
            verifier.verify(hash_obj, base64.b64decode(signature))
            return True
    except (ValueError, TypeError) as e:
        print('验签失败:', e)
        return False

```

RSA 加密函数 `ras_encrypt()`

```

def rsa_encrypt(publicPemPath, RSA_DecryptText):
    with open(publicPemPath, 'r') as f:
        key = f.read()

```

```

        rsakey = RSA.import_key(key)
        cipher = ED_PKCS1_v1_5.new(rsakey)
        RSA_EncrptText =
base64.b64encode(cipher.encrypt(RSA_DecryptText.encode('utf-8')))
    return RSA_EncrptText
RSA 解密函数 ras_decrypt ()
def rsa_decrypt(privatePemPath, RSA_EncrptText):
    with open(publicPemPath, 'r') as f:
        key = f.read() # 读取公钥文件内容
        rsakey = RSA.import_key(key) # 导入公钥
        cipher = ED_PKCS1_v1_5.new(rsakey)
        RSA_EncrptText =
base64.b64encode(cipher.encrypt(RSA_DecryptText.encode('utf-8')))
    return RSA_EncrptText

```

### 4.3 通信中基本函数

发送消息函数 `sendTo ()`

```

def sendTo(theSocket, message, hint=None):
    nonce = generate_nonce()
    timestamp = timeStamp()
    print(f"发送时间戳: {timestamp}")
    message_with_nonce = f"{message}-{nonce}-{timestamp}"
    hash_value =
hashlib.sha256(message_with_nonce.encode()).hexdigest()
    message_to_send = f"{message_with_nonce}-{hash_value}"
    theSocket.send(message_to_send.encode('utf-8'))
    if hint is not None:
        print(hint)

```

接收消息函数 `readFrom ()`

```

def readFrom(theSocket, hint=None):
    message_received = theSocket.recv(1024).decode('utf-8')
    if hint is not None:
        print(hint)
    parts = message_received.split('-')
    if len(parts) < 4:
        print("消息格式不正确！")
        return None
    message = '-'.join(parts[:-3])
    nonce = parts[-3]
    timestamp = parts[-2]
    received_hash = parts[-1]
    print(f"接收到的消息: {message}")
    print(f"接收到的时间戳: {timestamp}")

```

```

    if not timestamp:
        print("时间戳不存在或格式错误！")
        return None
    if nonce in used_nonces:
        print("Nonce 已被使用过，可能是重放攻击！")
        return None
    hash_value = hashlib.sha256(f"{message}-{nonce}-{timestamp}".encode()).hexdigest()
    if hash_value != received_hash:
        print("消息摘要不匹配，可能是被篡改的消息！")
        return None
    used_nonces[nonce] = True
    return message

```

发送随机数等校验信息进行身份认证函数 `sendLegalInfoTo()`

```

def sendLegalInfoTo(theSocket, privatePemPath, publicPemPath):
    # 向对方发送随机数等校验信息进行合法性认证
    LegalMessage = str(random.randint(2**31, 2**32))
    RSA_EncryptSignatureTo(theSocket, LegalMessage, privatePemPath,
publicPemPath)

```

对对方的随机数等信息进行合法性校验函数 `readLegalInfoFrom()`

```

def readLegalInfoFrom(theSocket, publicPemPath):
    legal = RSA_VerifyFrom(theSocket, publicPemPath)
    # 收到对方时间戳，收到对方签名后 RSA_VerifyFrom 用对方的公钥解签，得到签名中的
    SHA256 值，对一同发来的 RSA 密文（校验信息）作 SHA256，如果两个 SHA256 值相等则验
    证对方为合法
    return legal

```

发送数据包函数 `communicatePackageSender()`

```

def communicatePackageSender(theSocket, TDES_Key, privatePemPath,
publicPemPath, TDES_DecryptText):
    # 对 str 型的原始明文 TDES_DecryptText 进行一系列操作后发包
    prpcryptObject = prpcrypt()
    TDES_EncryptText = TDES_Encrypt(prpcryptObject, TDES_DecryptText,
TDES_Key)
    RSA_EncryptSignatureTo(theSocket, TDES_EncryptText,
privatePemPath, publicPemPath)
    # 对 TDES 密文用对方公钥加密后签名，连同校验信息一并发送给服务器

```

接收数据包函数 `communicatePackageReceiver()`

```

def communicatePackageReceiver(theSocket, TDES_Key, privatePemPath,
publicPemPath):
    # 对签名进行校验，无误后再进行 TDES 解密
    legalClient, RSA_DecryptText = RSA_VerifyFrom(theSocket,
publicPemPath)
    RSA_DecryptText = RSA_DecryptText.replace("b'",
''.replace("'", ''))

```



```

    if legalClient:
        TDES_EncryptText = str(RSA_SVED.rsa_decrypt(privatePemPath,
RSA_DecryptText))
        TDES_EncryptText = str(TDES_EncryptText).replace("b'",
'').replace("'", '')
        prpcryptBob = prpcrypt()
        TDES_DecryptText = TDES_Decrypt(prpcryptBob,
TDES_EncryptText, TDES_Key)
        return TDES_DecryptText

```

## 4.4 客户端通信

客户端首先根据服务器端 IP 地址和端口号建立套接字通信，通过身份验证校验后开始进行密钥协商，首先随机生成一个整数 DH\_Group 需要是 pyDHE 库中的组，典型的组有 14、15、16 等，这里取 15，DH\_Group=15 所代表的具体参数为 Modulus (p): 2048 位素数、Generator (g): 2，生成 DH 算法的公私钥对，将 DH\_Group，DH 公钥发送给服务器，接收服务器端公钥，取前 20 位作为服务器端公钥，根据 DH 算法生成共享密钥取前 24 位作为共享密钥，之后开启通信线程，收发信息。

```

Fc.sendLegalInfoTo(serverSocket, privatePemPath, publicPemPath)
# 向服务器发送自己的合法性校验信息
legalServer = Fc.readLegalInfoFrom(serverSocket, publicPemPath)
# 校验服务器的合法性
if (legalServer):
    # 校验对方身份合法时开始协商 DH 密钥以及后续通信
    DH_Group = 15
    # 生成 DH 算法公私钥对的参数
    DH_AliceClient, DH_PrivateAlice, DH_PublicAlice =
DH.DH_Original(DH_Group)
    # 生成 DHE 对象及 DH 公私钥
    Fc.sendTo(serverSocket, DH_Group)
    # 1.发送 DH_Group 给服务器
    Fc.sendTo(serverSocket, DH_PublicAlice, "已向服务器发送客户端 DH 公钥")
    # 2.发送客户端 DH 公钥给服务器
    DH_PublicBob = Fc.readFrom(serverSocket, '已接收到服务器 DH 公钥')
    # 3.接收服务器 DH 公钥
    print('服务器 DH 公钥', str(DH_PublicBob)[0:20])
    # DH 密钥交换协议得到的密钥
    DH_FinalKey = DH.DH_FinalKeyGenerator(DH_AliceClient,
int(DH_PublicBob))
    # 生成共享 DH 密钥
    print('开始通信...')
    print()
    TDES_Key = str(DH_FinalKey)[0:24]

```

```

# 使用协商出的共享 DH 密钥的前 24 位作 TDES 密钥
# 多线程实现边监听边发送
clientSending = threading.Thread(target=Fc.sendingThread,
args=(serverSocket, TDES_Key, privatePemPath, publicPemPath),
name='clientSendingThread')
clientReceiving = threading.Thread(target=Fc.receivingThread,
args=(serverSocket, TDES_Key, privatePemPath, publicPemPath),
name='clientReceivingThread')
clientSending.start()
clientReceiving.start()
clientSending.join()
clientReceiving.join()

```

## 4.5 服务器端通信

服务器端启动后，接收客户端的验证消息，并发送自己的身份验证信息。验证成功开启密钥交换，接收客户端的 DH\_Group，DH 公钥，取前 20 位作为客户端公钥，发送服务器端公钥，根据 DH 算法生成共享密钥取前 24 位作为共享密钥，之后开启通信线程，收发信息。

```

legalClient = Fc.readLegalInfoFrom(clientSocket, publicPemPath)
# 校验客户端的合法性
Fc.sendLegalInfoTo(clientSocket, privatePemPath, publicPemPath)
# 向客户端发送自己的合法性校验信息
if(legalClient):
    # 开始协商 DH 密钥以及后续通信
    DH_Group = Fc.readFrom(clientSocket)
    # 1.收到 DH_Group
    print('客户端使用的 DH_Group 为: ', DH_Group)
    DH_PublicAlice = Fc.readFrom(clientSocket, "已接收到客户端 DH 公钥")
    # 服务器从客户端套接字接收其公钥 DH_PublicAlice
    print("客户端 DH 公钥: ", DH_PublicAlice[0:20])
    DH_BobServer, DH_PrivateBob, DH_PublicBob =
DH.DH_Original(int(DH_Group))
    # 服务器使用相同的 int 型 DH_Group 生成 DHE 对象及 DH 公私钥
    Fc.sendTo(clientSocket, DH_PublicBob, "已向客户端发送服务器 DH 公钥")
    # 均合法时开始通信
    DH_FinalKey = DH.DH_FinalKeyGenerator(DH_BobServer,
int(DH_PublicAlice))
    # 服务器生成共享 DH 密钥
    TDES_Key = str(DH_FinalKey)[0:24]
    # 使用协商出的共享 DH 密钥的前 24 位作 TDES 密钥
    print('开始通信...')
    print()
    # 多线程实现边监听边发送
    serverSending = threading.Thread(target=Fc.sendingThread,

```

```

args=(clientSocket, TDES_Key, privatePemPath, publicPemPath),
name='serverSendingThread')
    serverReceiving = threading.Thread(target=Fc.receivingThread,
args=(clientSocket, TDES_Key, privatePemPath, publicPemPath),
name='serverSendingThread')
    serverSending.start()
    serverReceiving.start()
    serverSending.join()
    serverReceiving.join()

```

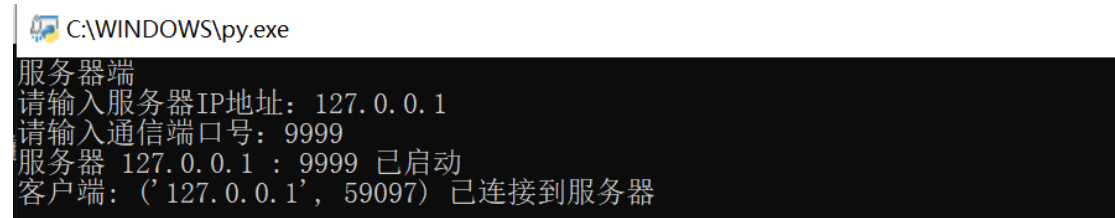
## 5 测试结果

### 5.1 运行

本课程设计在 windows 10,python 3.12.2 环境下开发完成，为了实验方便本文使用 127.0.0.1 本地回环地址来测试系统可行性。

首先输入服务器 IP 地址和通信端口号建立连接，下面测试直接在终端运行.exe 文件。

服务器端：



```

C:\WINDOWS\py.exe
服务器端
请输入服务器IP地址: 127.0.0.1
请输入通信端口号: 9999
服务器 127.0.0.1 : 9999 已启动
客户端: ('127.0.0.1', 59097) 已连接到服务器

```

客户端：



```

选择 C:\WINDOWS\py.exe
客户端
请输入服务器IP地址: 127.0.0.1
请输入通信端口号: 9999
正在尝试与服务器 127.0.0.1 : 9999 建立连接...
连接成功

```

之后客户端发送时间戳签名校验信息，以及接收服务器端数据包进行身份验证，。

```
正在用对方RSA公钥加密，加密前为： 3783795375
正在用本机RSA私钥解密XhAsh值签名
发送时间戳：2024年06月18日 10:50:28
已发送校验信息：b' bWRLRuqzRa7j8tV00nSvr+ome3o/OK7dgsrM0at4a0caF7yCzxCaFsCmpunYwVp17h0+meQ3H3emhE41UTv4i7a8+z8QqcYunqla0peu0rIS1t1tVEFRmPz/RsWsfKxzvszaxlxBSCAEakN3eDe0Yckw4W1lqERtLbAB2Zz+sDbhS2k2DFk9TG7ZE2bQjxYqn1+ZVds7AzY10hJr5d0XaoFEj7QEaSXtONCKq0jnQRs+G98Fz0pW0uXG2MhXU0/W4RANzG01SzoGkpkOfk18LbEYSotpl4PUjWmyiV1535n4mt0t18Ap1R7jBp5DsRtWYumhEbbg0B06e8YKSg==
发送时间戳：2024年06月18日 10:50:28
已发送RSA签名：b' WQoTtrR+rwXcLmWbEp7wTAIriSqMpy1M6L59Dz1P3q+ZtwwhUncM3yblLb0k0aUH7jdozTy+NHSrTkV9Zm50U5ghn2HNLFWX8Fc3TWEVY1xD7xsRxBad970pC9YH11UV2Z6XoB6Qj2c31k5h1Vpq4c1+1zGpQvP8Yq/fF4hIhJfMbcVjgtSV5sIPhDb1Zn/MDNutNKW3UwH5/OA4KvJbRk0BCRz0HY4QMThkteR/annCvaxc3R5R05Cbw1Hqc/Osu8tgbQo20VpvZG41GpcktZP8jCfSAvhKvVPV6+OzcWkTfvY2BkAgLgHPVvxgJOIkysbCW8GE4RRSIDhR2R64w==

已收到校验信息
接收到的消息：b' bHA92Ds0cfocxjo9fV/a/qwq6LTUOH3C18ReugSA7ZaNQoeEADmapqvmEpdn6nTPJmL12j0U05zkHZbUUS9cDNTRY7KDNHws3mwH+R9UHZMMs+sgP/qMDTgLBbNFWsGndRP40VdxFwp1RjA6/WnPZJhMkqHbXlv7tECshT1HGAAodHtb7KKc12xvcMBoKuC1rpbAGAhM1zJ6iIf4h15qW54ypGTh1bkH1+btddhZJmdNcBJ3MOLZQoeYVWovbT2qbD8nRONShEwCqdrwvieJP7WxmrZRo+FovFFH1ahWGM/PceC41DVB0pNuXNtisvbnH1JuYiDtD5sk1Q==
接收到的时间戳：2024年06月18日 10:50:28
校验信息为：b' bHA92Ds0cfocxjo9fV/a/qwq6LTUOH3C18ReugSA7ZaNQoeEADmapqvmEpdn6nTPJmL12j0U05zkHZbUUS9cDNTRY7KDNHws3mwH+R9UHZMMs+sgP/qMDTgLBbNFWsGndRP40VdxFwp1RjA6/WnPZJhMkqHbXlv7tECshT1HGAAodHtb7KKc12xvcMBoKuC1rpbAGAhM1zJ6iIf4h15qW54ypGTh1bkH1+btddhZJmdNcBJ3MOLZQoeYVWovbT2qbD8nRONShEwCqdrwvieJP7WxmrZRo+FovFFH1ahWGM/PceC41DVB0pNuXNtisvbnH1JuYiDtD5sk1Q==
已收到对方的RSA签名
接收到的消息：b' RnUhjPzCegFcmKXBO0oSa0phZRRdnGTcobvRF8SEc92SP2cC3r++m79mTROBL46SpJzHmVwIwSsz08YT1wWh0fERIE4oiLhegxwJTFzjP0sRZQAUCu83davuT34QaxcETx2ydQJJuU58Rkb1+xdm0tgSAhYp0AGHhPjYQVPS04sIY1sQIGNfWPa+FKNGVa00+v19bSeIuungWib+G0xaMekprn/WkV1P3aeazVmNu6mJaTqSPfKk1TAGsCz/OcxL5/CH01BV1v3a5nD+PQ4GDzaygKi1B7Kmf3crZceGem196KXbW4jQ/csv+amK+AlC1gmZm8C1bJ1Yyez0A==
接收到的时间戳：2024年06月18日 10:50:28
签名：b' RnUhjPzCegFcmKXBO0oSa0phZRRdnGTcobvRF8SEc92SP2cC3r++m79mTROBL46SpJzHmVwIwSsz08YT1wWh0fERIE4oiLhegxwJTFzjP0sRZQAUCu83davuT34QaxcETx2ydQJJuU58Rkb1+xdm0tgSAhYp0AGHhPjYQVPS04sIY1sQIGNfWPa+FKNGVa00+v19bSeIuungWib+G0xaMekprn/WkV1P3aeazVmNu6mJaTqSPfKk1TAGsCz/OcxL5/CH01BV1v3a5nD+PQ4GDzaygKi1B7Kmf3crZceGem196KXbW4jQ/csv+amK+AlC1gmZm8C1bJ1Yyez0A==
对方身份是否合法： True
```

服务器端收到客户端信息后开始校验，并发送自己的签名校验信息。

```
已收到校验信息
接收到的消息：b' bWRLRuqzRa7j8tV00nSvr+ome3o/OK7dgsrM0at4a0caF7yCzxCaFsCmpunYwVp17h0+meQ3H3emhE41UTv4i7a8+z8QqcYunqla0peu0rIS1t1tVEFRmPz/RsWsfKxzvszaxlxBSCAEakN3eDe0Yckw4W1lqERtLbAB2Zz+sDbhS2k2DFk9TG7ZE2bQjxYqn1+ZVds7AzY10hJr5d0XaoFEj7QEaSXtONCKq0jnQRs+G98Fz0pW0uXG2MhXU0/W4RANzG01SzoGkpkOfk18LbEYSotpl4PUjWmyiV1535n4mt0t18Ap1R7jBp5DsRtWYumhEbbg0B06e8YKSg==
接收到的时间戳：2024年06月18日 10:50:28
校验信息为：b' bWRLRuqzRa7j8tV00nSvr+ome3o/OK7dgsrM0at4a0caF7yCzxCaFsCmpunYwVp17h0+meQ3H3emhE41UTv4i7a8+z8QqcYunqla0peu0rIS1t1tVEFRmPz/RsWsfKxzvszaxlxBSCAEakN3eDe0Yckw4W1lqERtLbAB2Zz+sDbhS2k2DFk9TG7ZE2bQjxYqn1+ZVds7AzY10hJr5d0XaoFEj7QEaSXtONCKq0jnQRs+G98Fz0pW0uXG2MhXU0/W4RANzG01SzoGkpkOfk18LbEYSotpl4PUjWmyiV1535n4mt0t18Ap1R7jBp5DsRtWYumhEbbg0B06e8YKSg==
已收到对方的RSA签名
接收到的消息：b' WQoTtrR+rwXcLmWbEp7wTAIriSqMpy1M6L59Dz1P3q+ZtwwhUncM3yblLb0k0aUH7jdozTy+NHSrTkV9Zm50U5ghn2HNLFWX8Fc3TWEVY1xD7xsRxBad970pC9YH11UV2Z6XoB6Qj2c31k5h1Vpq4c1+1zGpQvP8Yq/fF4hIhJfMbcVjgtSV5sIPhDb1Zn/MDNutNKW3UwH5/OA4KvJbRk0BCRz0HY4QMThkteR/annCvaxc3R5R05Cbw1Hqc/Osu8tgbQo20VpvZG41GpcktZP8jCfSAvhKvVPV6+OzcWkTfvY2BkAgLgHPVvxgJOIkysbCW8GE4RRSIDhR2R64w==
接收到的时间戳：2024年06月18日 10:50:28
签名：b' WQoTtrR+rwXcLmWbEp7wTAIriSqMpy1M6L59Dz1P3q+ZtwwhUncM3yblLb0k0aUH7jdozTy+NHSrTkV9Zm50U5ghn2HNLFWX8Fc3TWEVY1xD7xsRxBad970pC9YH11UV2Z6XoB6Qj2c31k5h1Vpq4c1+1zGpQvP8Yq/fF4hIhJfMbcVjgtSV5sIPhDb1Zn/MDNutNKW3UwH5/OA4KvJbRk0BCRz0HY4QMThkteR/annCvaxc3R5R05Cbw1Hqc/Osu8tgbQo20VpvZG41GpcktZP8jCfSAvhKvVPV6+OzcWkTfvY2BkAgLgHPVvxgJOIkysbCW8GE4RRSIDhR2R64w==
对方身份是否合法： True

正在用对方RSA公钥加密，加密前为： 3271249876
正在用本机RSA私钥解密XhAsh值签名
发送时间戳：2024年06月18日 10:50:28
已发送校验信息：b' bHA92Ds0cfocxjo9fV/a/qwq6LTUOH3C18ReugSA7ZaNQoeEADmapqvmEpdn6nTPJmL12j0U05zkHZbUUS9cDNTRY7KDNHws3mwH+R9UHZMMs+sgP/qMDTgLBbNFWsGndRP40VdxFwp1RjA6/WnPZJhMkqHbXlv7tECshT1HGAAodHtb7KKc12xvcMBoKuC1rpbAGAhM1zJ6iIf4h15qW54ypGTh1bkH1+btddhZJmdNcBJ3MOLZQoeYVWovbT2qbD8nRONShEwCqdrwvieJP7WxmrZRo+FovFFH1ahWGM/PceC41DVB0pNuXNtisvbnH1JuYiDtD5sk1Q==
发送时间戳：2024年06月18日 10:50:28
已发送RSA签名：b' RnUhjPzCegFcmKXBO0oSa0phZRRdnGTcobvRF8SEc92SP2cC3r++m79mTROBL46SpJzHmVwIwSsz08YT1wWh0fERIE4oiLhegxwJTFzjP0sRZQAUCu83davuT34QaxcETx2ydQJJuU58Rkb1+xdm0tgSAhYp0AGHhPjYQVPS04sIY1sQIGNfWPa+FKNGVa00+v19bSeIuungWib+G0xaMekprn/WkV1P3aeazVmNu6mJaTqSPfKk1TAGsCz/OcxL5/CH01BV1v3a5nD+PQ4GDzaygKi1B7Kmf3crZceGem196KXbW4jQ/csv+amK+AlC1gmZm8C1bJ1Yyez0A==
```

身份验证通过后进行密钥协商，首先客户端生成 DH 公私钥,接收服务器端公钥后生成共享密钥，取前 24 位作为 3DES 加密密钥

```
正在生成本机DH公私钥...
本机DH公钥： 11471718009523110344
本机DH私钥： 5192408523314181425
发送时间戳：2024年06月18日 10:50:28
已向服务器发送客户端DH公钥
已收到服务器端DH公钥
接收到的消息：23401747638329057005018571720286087883595430567618146964673509372333967286592427622782208367619468904851481811971182142662708050181868875097677425202425191761576222832411936260255711738995011025545848838720131666458526551673272738452115027183247568250858343563824120714378709791382330859146917502841003667237122974492501146960740949224314003862463372932461836285895613458719061157790181942751356279693862199961422210577489604922058670098013484870036011984672611042962431720518903901652549800853653089678112835418927570587599110913727000316268589336367169884788568308199283989245035621638136783781152718442933212190422422466847615202297370046320856717016245430975007118749364700955718269493587853964780030742034221753569681887300754849088634508735930788269167321564049074764598917004583428737285346685631310256751325228937241673298088693546446003257472807897788272886127238127932450444148200712550791495558921507803996
接收到的时间戳：2024年06月18日 10:50:28
服务器DH公钥 23401747638329057005
本机计算的DH共享密钥： 24547285491212490216
```

服务器端收到 DH\_Group 和对方公钥后，生成自己的公私钥，计算 DH 共享密钥，取前 24 位作为对称加密密钥。

```
接收到的消息：15
接收到的时间戳：2024年06月18日 10:50:28
客户端使用的DH_Group为： 15
已收到客户端DH公钥
接收到的消息：11471718009523110344550144762359443471162633571572461980863528896714465233771827339557807169540877686265699635943950130212009248244417627460812474894644729150815748699216264921943547587430708205149044948735941717540001992604992949776099071002302714800186198203221202614290676830705562923047434865204475913247362246561688749487584791066907523840548537468165475694894909344605062902690117708064117640546741689260283608629399005923297821878828926011470681741612216223950761127916235864153354373398919142181562195632561055252360771853371588399906189103161921743172981651979965679314358513521688461310012303055237071855140030965214816875499230829352084874859285267374493670839925864629728893995916659026250749725376710760183460373106549953204355778875598609226093223552200347759938437090630799360126007663192082533057624346885119951433174397629456865499395366324508009933257072727133434949293469148981955731432789911162611
接收到的时间戳：2024年06月18日 10:50:28
客户端DH公钥： 11471718009523110344
正在生成本机DH公私钥...
本机DH公钥： 23401747638329057005
本机DH私钥： 51589135345767269314
发送时间戳：2024年06月18日 10:50:28
已向客户端发送服务器DH公钥
本机计算的DH共享密钥： 24547285491212490216
```

之后可以开始相互通信，quit 则退出。



客户端:

```
输入英文或输入quit结束发送: 已收到校验信息
接收到的消息: b' QZ064XGMmOHTI1hpSoEeqgSpf9LMvwtKQbTrVQi jUuoGHEV jY3CjYAx4W1nPaZq1xCEUNFma8mSDLx jFIVpagde07hwCxHSWo6soM5MuW4naBFV+HMLWt6XHCUYv7Ff/kCWPft3qvcU1
Uux1G1W/szWHGtI+KtEo1iG0LnVv37AysgUNLk1t7ZSUK+DD13SuQ7d4iL6NN2fKAL1pF74m/bPRcOWMotu5Gw+1SYh8J0xVGemd5CIV+Fs1tW/W/6StWbp1b0GwX6Z46yWqC07UNR10a52cEDebN4fEbvJweV
Yma4snTLdN0zk4p2ZQvEZxmRR0/A2y jTdEaxeBVQA6BA==
接收到的时间戳: 2024年06月18日 10:50:50
校验信息为: b' QZ064XGMmOHTI1hpSoEeqgSpf9LMvwtKQbTrVQi jUuoGHEV jY3CjYAx4W1nPaZq1xCEUNFma8mSDLx jFIVpagde07hwCxHSWo6soM5MuW4naBFV+HMLWt6XHCUYv7Ff/kCWPft3qvcU1U
ux1G1W/szWHGtI+KtEo1iG0LnVv37AysgUNLk1t7ZSUK+DD13SuQ7d4iL6NN2fKAL1pF74m/bPRcOWMotu5Gw+1SYh8J0xVGemd5CIV+Fs1tW/W/6StWbp1b0GwX6Z46yWqC07UNR10a52cEDebN4fEbvJweV
Yma4snTLdN0zk4p2ZQvEZxmRR0/A2y jTdEaxeBVQA6BA==
已收到对方的RSA签名
接收到的消息: b' dqkxo+/FrqCMU+oLe/nI/mgZ0myJtEoIcxx2m1WFz jivPb7hqi5EAA6hJBSyBoHj76Uf2 jsdppcdj2wakQ8nbV82WSMBFAkpDvZ7IbIeE6yHP5KN21oe+H6C1r1AYzvwUJDRBp0u0aM2a
ubU2EkpQXfFbbd+J90SwTt5+80uok2Qo+UCrP18nhKw1a71t+TaOKKYG7GD1tqExV9cH3BgapU5wm4Vh/zRbdu0F2s0Uz10khhcDu4WmsTiaxSAArmKH8126/zJyqlke2X5fwhcCnfrgPvC0eerx1kt9qS
jFB17BKXL4mtkTD4F90L4V18R/TuVlqogBHza0e8RLCw==
接收到的时间戳: 2024年06月18日 10:50:50
签名信息为: b' dqkxo+/FrqCMU+oLe/nI/mgZ0myJtEoIcxx2m1WFz jivPb7hqi5EAA6hJBSyBoHj76Uf2 jsdppcdj2wakQ8nbV82WSMBFAkpDvZ7IbIeE6yHP5KN21oe+H6C1r1AYzvwUJDRBp0u0aM2a
ubU2EkpQXfFbbd+J90SwTt5+80uok2Qo+UCrP18nhKw1a71t+TaOKKYG7GD1tqExV9cH3BgapU5wm4Vh/zRbdu0F2s0Uz10khhcDu4WmsTiaxSAArmKH8126/zJyqlke2X5fwhcCnfrgPvC0eerx1kt9qSjFB17
BKXL4mtkTD4F90L4V18R/TuVlqogBHza0e8RLCw==
对方身份是否合法: True

3DES密文为: Dicjrr4BkWo=
解密得到3DES明文为: hi,alice

hello,bob
生成3DES密文为: EFHCxM6pP/Mzp0RYw9zVAA==

正在用对方RSA公钥加密,加密前为: EFHCxM6pP/Mzp0RYw9zVAA==
正在用本机RSA私钥对密文Hash值签名
发送时间戳: 2024年06月18日 10:53:48
已发送校验信息: b' DxH/QpBja6f1NlHKhWw2nOpW6gXLWq/FLjY5XE80r4zwcemUBal8JxstSyTheL+CuZnOfjDR0sJjV/k00Eo2j7NNOA9qww/n/+DeNohFHGefEYfgQ5ysEI46+3ecACY8010D3piHza2
YT05ISTjir/q6U/kqooGkqUE8QPCPoSF21NA/vj6W1uAJReqnSmC+exdNEq32pKyRk0h0T1Lsj+Nkd+FHV1wrpvh2oE48d4Y/iwbMmpKMncHecsJptsKypyNdr6CWOAA3DF4nWMMVrQVhzRAUZI1q2jttst
1a5VqA5Apq785x6zSWASjgYRwal+8qJlulutGfdm9m7J6g==
已发送RSA签名: b' ce1t/xjDR/ghMQcVr+MG75LE6Qv4zh69g4i+63yhxipwHvIqivckvN1qrXvnhAviQ+XZ2DV3zMa4vc2EdVc1KhHJMt+5f4fbGAgw2DSJjRk1FBdfNaBapQC+nys91BHAA/Of+eGxcC
sBmqXcUm026ehZc2ngyn6k0ve23Uly2xYXurYgmVLTtC7B6mkLJoTLXXd+10BBSFoX2Rw3NqdaysQKWCYa6zap35pBYKOFHb6yRMRHlue+QVqV09rPFxwZ0SMKC30CHSFm142Xx1986yC08Eoboe9rS50
C38XzCkrmHU/kD6SvGqijWf4D4CrttUgeblM1MzGk+YNDZQ==

服务器端:
```

```
输入英文或输入quit结束发送:
已收到校验信息
接收到的消息: b' ouSJuDrfG8G6LR4qALxxmt2/ind+IupdEC7Dd7AR3kWPvm4LV6xtr7wA6hr18z0QEmY1/1FKQB8FEqair1pm/hnZDNQApwgTLEs+Z9xW7eNk/Cz0gx5ThjZnltoFntOBs+wBkVe0oiIG
ObMY1v/pB0cHEEmk7b5iUwQTAxqMB/B+dFmFdiritpTS2LSbXw6x45RMhwaT1ItVX/3wNysmERmsV+Oux2sfsVIPVqkZxP/0XKAqqPS1IGrAu9+f1KsY6kelJRR4JuQJsyJ1imgj58wLOBMwuhV/uyTYq0
XW7JVZdhRzep17TtkirhyJ8CImkKYTaVmQpN1NgJYPSA==
接收到的时间戳: 2024年06月18日 10:50:44
校验信息为: b' ouSJuDrfG8G6LR4qALxxmt2/ind+IupdEC7Dd7AR3kWPvm4LV6xtr7wA6hr18z0QEmY1/1FKQB8FEqair1pm/hnZDNQApwgTLEs+Z9xW7eNk/Cz0gx5ThjZnltoFntOBs+wBkVe0oiIG0
bMY1v/pB0cHEEmk7b5iUwQTAxqMB/B+dFmFdiritpTS2LSbXw6x45RMhwaT1ItVX/3wNysmERmsV+Oux2sfsVIPVqkZxP/0XKAqqPS1IGrAu9+f1KsY6kelJRR4JuQJsyJ1imgj58wLOBMwuhV/uyTYq0X
W7JVZdhRzep17TtkirhyJ8CImkKYTaVmQpN1NgJYPSA==
已收到对方的RSA签名
接收到的消息: b' QpIpa1JMj0001t7jz5nsI7vt8+93uokiWqJTw72aXCTokV3/2za6Nuo/4YZ7aDRBGq015aGhU8zb9/f9J93QOf/w1qlbyH1RbhJcXI+ju4BIjDVuukSR/Omik+ZSUCIwoVtM1z38X3y
od/G6arxz2ZLrEFj8/1XtZRjkIq54031TFqQzpproobYSwYSZvTwosoSAFsJk61ENoAhwvsdD065QY11tw05Y+w4YIMpsol1p297sWIDWn81P0195VNeKL4P5Q0i5xbVj0xLPYwtrB0ANHUA574AS1BNuS
0JKJ6J5eGVuvzpLlQnWmda0ZfX8drR15Au1bbbp3J1WA==
接收到的时间戳: 2024年06月18日 10:50:44
签名信息为: b' QpIpa1JMj0001t7jz5nsI7vt8+93uokiWqJTw72aXCTokV3/2za6Nuo/4YZ7aDRBGq015aGhU8zb9/f9J93QOf/w1qlbyH1RbhJcXI+ju4BIjDVuukSR/Omik+ZSUCIwoVtM1z38X3yod/G6
arxz2ZLrEFj8/1XtZRjkIq54031TFqQzpproobYSwYSZvTwosoSAFsJk61ENoAhwvsdD065QY11tw05Y+w4YIMpsol1p297sWIDWn81P0195VNeKL4P5Q0i5xbVj0xLPYwtrB0ANHUA574AS1BNuS0JkKJ6
J5eGVuvzpLlQnWmda0ZfX8drR15Au1bbbp3J1WA==
对方身份是否合法: True

3DES密文为: t2tJwz0+fIwzp0RYw9zVAA==
解密得到3DES明文为: hello,bob

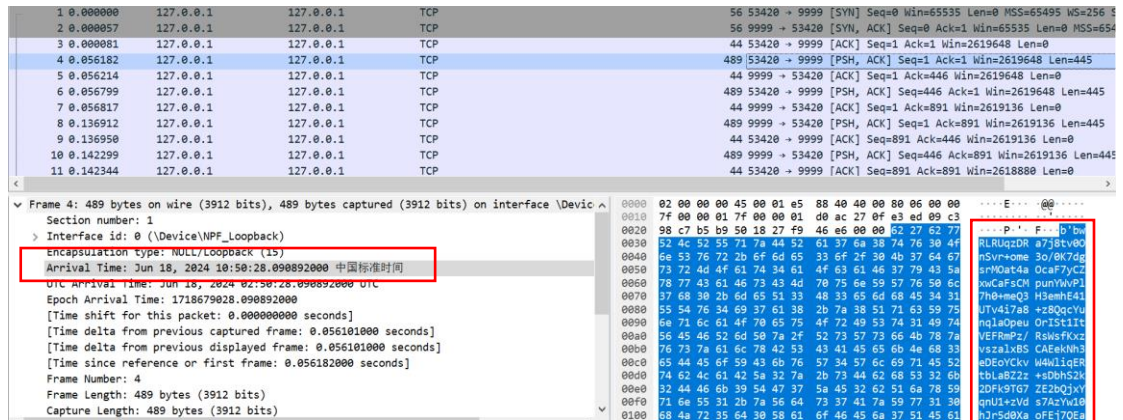
hi,alice
生成3DES密文为: Dicjrr4BkWo=

正在用对方RSA公钥加密,加密前为: Dicjrr4BkWo=
正在用本机RSA私钥对密文Hash值签名
发送时间戳: 2024年06月18日 10:50:50
已发送校验信息: b' QZ064XGMmOHTI1hpSoEeqgSpf9LMvwtKQbTrVQi jUuoGHEV jY3CjYAx4W1nPaZq1xCEUNFma8mSDLx jFIVpagde07hwCxHSWo6soM5MuW4naBFV+HMLWt6XHCUYv7Ff/kCWPft3qvc
U1Uux1G1W/szWHGtI+KtEo1iG0LnVv37AysgUNLk1t7ZSUK+DD13SuQ7d4iL6NN2fKAL1pF74m/bPRcOWMotu5Gw+1SYh8J0xVGemd5CIV+Fs1tW/W/6StWbp1b0GwX6Z46yWqC07UNR10a52cEDebN4fEbvJ
weVYma4snTLdN0zk4p2ZQvEZxmRR0/A2y jTdEaxeBVQA6BA==
接收到的时间戳: 2024年06月18日 10:50:50
已发送RSA签名: b' dqkxo+/FrqCMU+oLe/nI/mgZ0myJtEoIcxx2m1WFz jivPb7hqi5EAA6hJBSyBoHj76Uf2 jsdppcdj2wakQ8nbV82WSMBFAkpDvZ7IbIeE6yHP5KN21oe+H6C1r1AYzvwUJDRBp0u0aM2
aubU2EkpQXfFbbd+J90SwTt5+80uok2Qo+UCrP18nhKw1a71t+TaOKKYG7GD1tqExV9cH3BgapU5wm4Vh/zRbdu0F2s0Uz10khhcDu4WmsTiaxSAArmKH8126/zJyqlke2X5fwhcCnfrgPvC0eerx1kt9q
SjFB17BKXL4mtkTD4F90L4V18R/TuVlqogBHza0e8RLCw==
```

## 5.2 安全性分析

本课设使用密码学协议分析工具 **wireshark** 对通信过程中的包进行抓取,分析数据传输安全性。

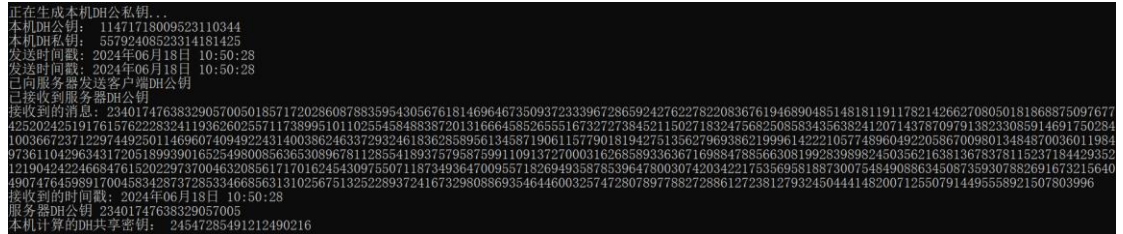
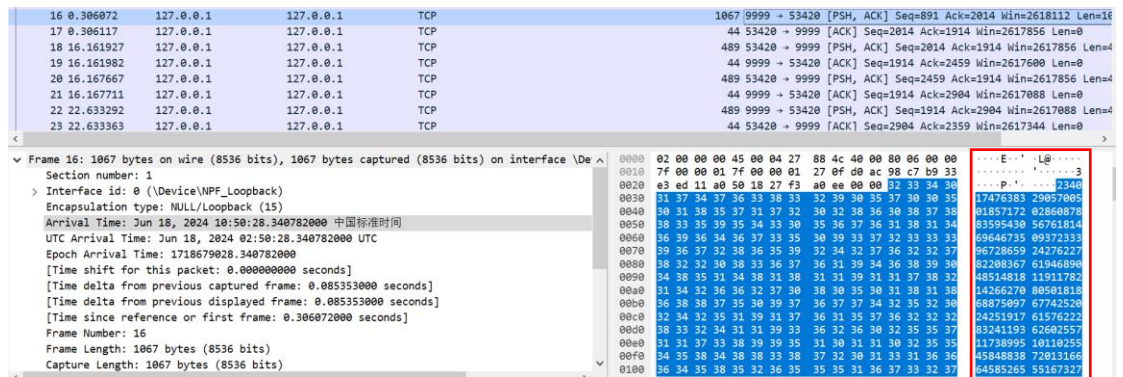
首先我们通过抓包可以看到在 10: 50: 28 客户端发送的时间戳,明文传输,之后生成随机数 3783795375,加密签名后发送给客户端。



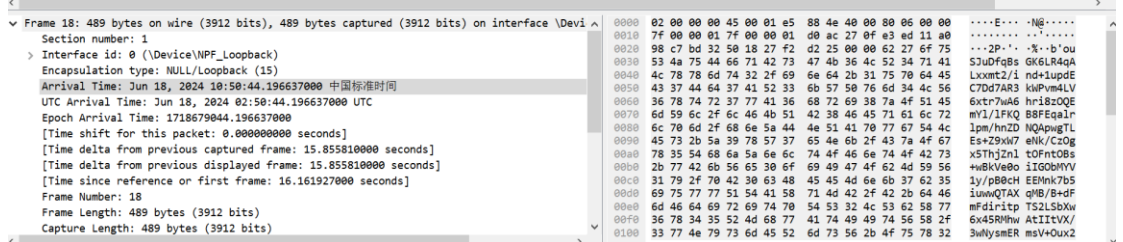
下面是对 3783795375 的密文签名的抓取，以及 RSA 对随机数加密的密文，可以看到传输过程中均以密文传输，有效保证了数据传输的完整性、保密性。时间戳的加入也能降低重放攻击的危害，但注意通信时间的延长影响。



在进行密钥协商的过程中，使用 DH 进行密钥交换时，公钥是明文传输，因为非对称密钥体制的特性，这并不会影响到通信的安全性。



最后就是利用密钥交换的密钥对传输消息进行加密，传输过程中首先明文发送时间戳，之后发送加密密文以及签名值。





No.	Time	Source	Destination	Protocol	Length	Info
16	0.306072	127.0.0.1	127.0.0.1	TCP	1067	9999 → 53420 [PSH, ACK] Seq=891 Ack=2014 Win=2618112 Len=16
17	0.306117	127.0.0.1	127.0.0.1	TCP	44	53420 → 9999 [ACK] Seq=2014 Ack=1914 Win=2617856 Len=0
18	16.161927	127.0.0.1	127.0.0.1	TCP	489	53420 → 9999 [PSH, ACK] Seq=2014 Ack=1914 Win=2617856 Len=4
19	16.161982	127.0.0.1	127.0.0.1	TCP	44	9999 → 53420 [ACK] Seq=1914 Ack=2459 Win=2617600 Len=0
20	16.167667	127.0.0.1	127.0.0.1	TCP	489	53420 → 9999 [PSH, ACK] Seq=2459 Ack=1914 Win=2617856 Len=4
21	16.167711	127.0.0.1	127.0.0.1	TCP	44	9999 → 53420 [ACK] Seq=1914 Ack=2904 Win=2617088 Len=0
22	22.633292	127.0.0.1	127.0.0.1	TCP	489	9999 → 53420 [PSH, ACK] Seq=1914 Ack=2904 Win=2617088 Len=4
23	22.633363	127.0.0.1	127.0.0.1	TCP	44	53420 → 9999 [ACK] Seq=2904 Ack=2359 Win=2617344 Len=0
24	22.637140	127.0.0.1	127.0.0.1	TCP	488	9999 → 53420 [PSH, ACK] Seq=2359 Ack=2904 Win=2617088 Len=4
25	22.637165	127.0.0.1	127.0.0.1	TCP	44	53420 → 9999 [ACK] Seq=2904 Ack=2803 Win=2616832 Len=0
38	200.108326	127.0.0.1	127.0.0.1	TCP	489	53420 → 9999 [PSH, ACK] Seq=2904 Ack=2803 Win=2616832 Len=4
39	200.108358	127.0.0.1	127.0.0.1	TCP	44	9999 → 53420 [ACK] Seq=2803 Ack=3349 Win=2616832 Len=0
40	200.109139	127.0.0.1	127.0.0.1	TCP	488	53420 → 9999 [PSH, ACK] Seq=3349 Ack=2803 Win=2616832 Len=4
41	200.109157	127.0.0.1	127.0.0.1	TCP	44	9999 → 53420 [ACK] Seq=2803 Ack=3793 Win=2616320 Len=0

▼

Frame 20: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits) on interface \Device\NPF\_{...}

Section number: 1

Interface id: 0 (\Device\NPF\_{...})

Encapsulation type: NULL/Loopback (15)

Arrival Time: Jun 18, 2024 10:50:44.202377000 中国标准时间

UTC Arrival Time: Jun 18, 2024 02:50:44.202377000 UTC

Epoch Arrival Time: 1718679044.202377000

[Time shift for this packet: 0.00000000 seconds]

[Time delta from previous captured frame: 0.005685000 seconds]

[Time delta from previous displayed frame: 0.005685000 seconds]

[Time since reference or first frame: 16.167667000 seconds]

Frame Number: 20

Frame Length: 489 bytes (3912 bits)

Capture Length: 489 bytes (3912 bits)

0000 02 00 00 00 45 00 01 e5 88 50 40 00 80 06 00 00 .....Pg.....

0010 7f 00 00 01 7f 00 00 01 d0 ac 27 0f e3 ed 13 5d .....b'Op

0020 98 c7 bd 32 58 18 27 f2 65 b8 00 00 62 27 51 70 ...2p...e...b'Op

0030 54 70 71 69 4a 4d 4a 71 4f 4f 4f 31 74 37 6a 7a Tpq1JHJq 0001t7jz

0040 35 6e 73 49 37 56 74 38 2b 39 33 75 6f 6b 69 77 5ns17Vt8 +93uokiw

0050 71 4a 54 77 56 37 32 61 58 43 54 6f 6b 56 33 2f qTWV72a XCTokV3/

0060 32 7a 61 36 4e 75 6f 2f 34 59 5a 37 61 44 52 42 2za6Nuo/ 4Y27aDRB

0070 47 71 30 31 35 73 47 68 55 38 7a 42 39 2f 66 39 Gg015aSh U8z89/f9

0080 4a 39 33 51 4f 66 2f 77 31 71 6c 62 79 48 6c 52 J93Q0f/w lq1byHlR

0090 62 68 4a 78 58 49 74 6a 75 34 42 49 6a 44 56 75 bhJXIXtj u48IJDVU

00a0 75 6b 38 52 2f 4f 6d 69 6b 2b 5a 53 55 63 49 77 uk8R/Omi k+ZSuIw

00b0 6f 56 74 4d 6c 7a 33 38 58 33 79 6f 64 2f 47 36 oVtM1z38 X3yod/G6

00c0 61 72 7a 78 32 5a 4c 72 45 46 6a 38 2f 31 58 74 arxz2ZLr EF18/1Xt

00d0 5a 52 6a 60 49 71 35 34 30 33 6c 54 46 71 51 67 ZKjK1q54 031Tf4Qg

00e0 61 70 72 72 6f 6f 62 59 53 77 57 59 53 5a 76 54 aprroobY S4WYS2VT

00f0 77 6f 73 6f 53 41 46 73 4a 6b 36 49 45 4e 6f 41 wos0AFs Jk6IEIoA

0100 68 77 76 73 64 44 30 36 35 51 59 54 31 74 77 4f hwsdD06 5QVt1tw0

综上，我们的通信协议具有一定的安全性，可以作为轻量级加密通信使用，通关通信时间戳也可以看出系统的延迟低。

## 6 系统特色及与现有的安全通信协议比较

### 6.1 系统特色

本系统是基于 RSA+SHA256+DH+3DES 的轻量级加密通信协议，所使用的加密方法简单，可操作性强，RSA、SHA256 和 3DES 结合使用在保障了通信安全性的同时，也具有较快的加解密速度，高效性使其更适合即使的通信需要，系统延迟低。系统在通信之前，双向身份认证确保双方的身份都是可信的。通过生成随机数，并对其进行 RSA 加密和签名，然后交换这些加密数据，确保通信双方都能够确认对方的身份。每次认证时都会附带时间戳，防止重放攻击，提高系统的安全性。在双向认证通过后，系统使用 Diffie-Hellman 算法协商共享密钥，确保通信过程中使用的加密密钥是动态生成的：使用共享 DH 密钥生成的 24 位 TripleDES 密钥进行数据加密，保证通信数据的安全性。消息在发送前不仅用 3DES 加密，还用接收方的 RSA 公钥再次加密，并用发送方的 RSA 私钥对加密消息的 SHA256 值签名，双重加密和签名增加了安全性。系统支持多线程并发处理，提升了通信效率和用户体验，发送和接收数据分别由不同的线程处理，确保了实时性和并发处理能力。由于系统是轻量级的，没有涉及 PKI 的复杂使用，公私钥对的生成和分发变得更为简便：RSA 公钥通过安全渠道直接下发，对 PKI 依赖较少，简化了系统的部署和管理。

23

## 6.2 与现有的安全通信协议比较

### 6.2.1 TLS/SSL 介绍

TLS(传输层安全协议)和 SSL(安全套接层)是网络通信中常用的加密协议,用于保护数据在传输过程中的安全性和完整性。它们通过使用对称加密、非对称加密和数字签名等技术,确保通信过程中的数据不被窃取、篡改或伪造。TLS/SSL 在客户端和服务端之间建立安全连接,通过握手过程协商加密算法和密钥,保证通信的机密性。它还提供了身份验证机制,通常通过数字证书来验证服务器和客户端的身份。TLS/SSL 协议不仅提供了数据加密功能,还包括了身份验证、会话密钥生成和交换等安全机制,有效防止中间人攻击、数据泄露和劫持等网络威胁。TLS 是 SSL 的继承者,修复了 SSL 的多个安全漏洞,并改进了性能和安全性。目前,TLS 已成为互联网中广泛使用的安全协议,应用于 HTTPS、电子邮件、安全即时消息和 VPN 等领域,为网络通信提供坚实的安全保障。

### 6.2.2 本课设计协议与 TLS/SSL 异同点

对于现有应用最广泛的 TLS/SSL 协议相比,本课设所设计的协议与其相同点,也有不同之处,本课设所设计的协议也有区别于 TLS/SSL 协议的优点。首先是加密机制:两者都使用了对称加密(如 3DES)和非对称加密(如 RSA)来确保数据传输的安全性。其次是身份认证,两个协议都涉及到了身份认证,本协议使用 RSA 公私钥对进行双向身份验证,确保通信双方的合法性。TLS/SSL 使用证书和公私钥对进行身份验证,通过 CA 签署的证书确保服务器和客户端的合法性。在通信过程中都设计密钥交换,本课设设计的协议使用 Diffie-Hellman (DH) 算法协商共享密钥,确保后续通信的安全性。TLS/SSL 使用多种密钥交换算法(包括 DH 和 ECDHE 等)来生成会话密钥。数据传输过程中都涉及到对通信数据加密,我们的协议使用 3DES 算法进行数据加密,确保数据的保密性。TLS/SSL 使用多种加密算法(如 AES、TripleDES 等)对数据进行加密,确保数据的保密性。协议也都包含完整性保护我们的协议使用 RSA 签名和 SHA256 校验保证数据的完整性,TLS/SSL 使用 HMAC(如 HMAC-SHA256)等技术保证数据的完整性。两个协议也有诸多不同之处,首先在复杂性和功能方面,我们设计的协议相对简单,适合轻量级身份认证和通信加密,没有涉及 PKI(公钥基础设施)和复杂的证书管理。



TLS/SSL 功能全面，支持证书管理、强大的加密算法和认证机制，是互联网安全通信的标准协议。对于证书管理，我们设计的协议不涉及 CA 签署的证书，仅使用预分发的公钥进行认证。而 TLS/SSL 依赖 CA 颁发和管理证书，通过受信任的第三方机构进行身份验证。在抗攻击性方面，我们设计的协议实现了基本的防重放攻击和中间人攻击，但由于没有使用标准的抗攻击技术，安全性较低。而 TLS/SSL 经过多年的验证和改进，具有较强的抗攻击能力，能够防御多种已知的攻击，如重放攻击、中间人攻击和回滚攻击。在灵活性和可扩展性方面，我们设计的协议设计相对固定，不支持灵活的算法选择和协议扩展。TLS/SSL 支持多种加密算法和密钥交换方式，具有很强的灵活性和可扩展性，可以适应不同的安全需求。对于两者的异同，也有各种的优缺点，对于 TLS/SSL 协议，其高度安全，经过多年的验证和改进，能够防御多种已知的攻击，安全性高。而且作为互联网安全通信的标准协议，被广泛应用和认可。灵活性强，支持多种加密算法和密钥交换方式，能够适应不同的安全需求。它的缺点有实现复杂，协议较为复杂，实现和维护成本较高。资源占用高，需要更多的计算资源和存储空间，不适合资源受限的环境。依赖证书管理，需要依赖 CA 签署和管理证书，增加了系统复杂性和管理成本。

我们的协议首先是简单易实现，适合于不需要复杂功能的轻量级应用，便于理解和实现。资源占用低，适合于资源有限的环境，如嵌入式系统和物联网设备。可以根据具体需求快速定制和修改，适应性强。缺点在于安全性较低：由于没有使用标准的抗攻击技术，安全性可能不足以应对复杂的攻击，为了更轻便高效，我们使用的 3DES、SHA256 在安全性方面都有一定风险。缺乏标准化，没有经过广泛验证和认可，可能存在潜在的安全漏洞。证书管理复杂，没有 CA 证书管理机制，公钥分发和管理可能较为繁琐。

总之我们设计的协议在轻量级、简单易实现和资源占用方面具有优势，适合于一些特定的应用场景。然而在安全性、标准化和抗攻击能力方面，TLS/SSL 则更为强大和可靠。因此，在选择协议时需要根据具体的应用场景和安全需求进行权衡和选择。

## 7 结论

在本项目中，我们设计并实现了一个轻量级的身份认证和加密通信协议。通过使用 RSA 算法进行身份认证和 Diffie-Hellman 算法进行密钥协商，结合 3DES 算法进行加密通信，成功实现了安全的客户端与服务器之间的双向通信。我们验证了双方的身份合法性，确保了数据传输的机密性和完整性，并实现了基本的防重放攻击机制。但是整个通信协议还有诸多不安全因素，比如公钥分发，当前通过直接交换 .pem 文件的方式进行公钥分发，安全性较低，容易受到中间人攻击。未来可以考虑引入公钥基础设施（PKI）来提升公钥分发的安全性。对于抗重放攻击虽然实现了基本的防重放攻击机制，但仍需加强时间戳的校验和管理，确保时间戳的唯一性和有效性。以及目前只支持 RSA 和 3DES 算法，未来可以考虑引入更多的加密算法和哈希算法（如 AES、SHA-256），增强协议的安全性和灵活性。在性能优化方面，由于加密和解密操作会带来一定的性能开销，未来可以针对高并发场景进行优化，提升系统的整体性能。未来本安全通信协议还有许多可以改进的展望，例如引入 PKI，通过引入公钥基础设施，提升公钥分发和管理的安全性，进一步增强协议的安全性。增加会话恢复、完全前向保密等高级安全功能，提升通信的可靠性和安全性。考虑与现有的标准协议（如 TLS/SSL）进行兼容，提升协议的标准化程度，便于集成到现有的系统中。进行全面的安全审计和漏洞检测，及时修复潜在的安全隐患，确保协议的安全性。

本项目作为一个轻量级的身份认证和加密的安全通信协议的设计与实现，主要面向课程设计目的。在实际应用中，应充分考虑安全性、性能和兼容性等因素，并结合具体应用场景进行优化和改进。在本项目的开发过程中，我深入了解了现代密码学的基本原理和实际应用，对 RSA、Diffie-Hellman、3DES 等经典算法有了更深刻的认识。同时，我也认识到安全通信协议设计的复杂性和挑战性，特别是在实现双向认证和密钥协商的过程中，需要充分考虑各种潜在的攻击和安全隐患。通过本项目，我不仅提升了编程和调试能力，更加深了对信息安全的理解和重视。这些经验和知识将为我未来在信息安全领域的研究和实践打下坚实的基础。

## 致谢

在此,我要衷心感谢所有对本次课程设计的成功完成做出贡献的人士。首先,我要深深感谢我的导师,他们在整个项目过程中给予了我悉心的指导、鼓励和宝贵的见解。他们的专业知识和支持在塑造这项工作、扩展我在安全通信协议领域的知识方面起到了关键作用。还有网安学院的所有教授我知识的老师们,他们对于网络安全、密码学、安全协议等等方面知识的讲授,让我得以有扎实基础来进行课程设计,直至完成本次课程设计。我还要感谢我的父母家人,他们的帮助和鼓励在我学习生涯、课程设计期间至关重要。此外,我也要感谢我的同学,他们的讨论、反馈和合作努力丰富了协议设计的发展和完善过程。最后,我要感谢开源社区和诸如 Python 和 WireShark 等工具的支持,这些对于协议的实现和评估至关重要。这个项目是一次充满收获的学习经历,我衷心感谢每一位参与其中的人员。

## 参考文献

- [1] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR[J]. Software Concepts and Tools, 1996,17: 93-102.
- [2] G Denker and J Millen. CAPSL Intermediate Language[A]. In Formal Methods and Security Protocols[C],1999.FLOC'99 Workshop.
- [3] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. Designs[J], Codes and Cryptography, 2:107-125, 1992.
- [4] 张兵, 马新新, 秦志光. 轻量级 RFID 双向认证协议设计与分析[J]. 电子科技大学学报, 2013, 42(3): 425-430.
- [5] 徐远涛, 黄继刚, 王宝军, 等. 身份认证系统认证协议的设计与分析[J]. 计算机与信息技术, 2007 (6): 44-47.
- [6] 王惠斌. 安全认证协议的设计与分析[D]. 解放军信息工程大学, 2010.
- [7] 卿斯汉. 安全协议 20 年研究进展[J]. 软件学报, 2003, 14(10): 1740-1752.
- [8] 李金库, 张德运, 张勇. 身份认证机制研究及其安全性分析[J]. 计算机应用研究, 2001, 18(2): 126-128.
- [9] 余海冰, 潘泽宏. Diffie-Hellman 密钥交换技术综述[J]. 科技信息, 2007 (10):

51-52.

[10] 李亚敏, 李小鹏, 吴果. 身份认证的密钥交换算法[J]. 计算机工程, 2006, 32(12): 171-172.