学 号 2012201101

密 级

哈尔滨工程大学学士学位论文

安全聊天工具的设计与实现

院（系）名　称：软件工程学院

专　业　名　称：软件工程

学　生　姓　名：孙雪峰

指　导　教　师：史长亭 讲 师

哈尔滨工程大学

2017年6月

# 第1章 绪论

1.1 安全聊天工具研究的背景和意义

2017年5月 12日，一种叫做WannaCry 勒索病毒利用NSA（National Security Agency，美国国家安全局）泄露的危险漏洞“EternalBlue”（永恒之蓝）进行传播。截止2017年5月15日，WannaCry造成了至少有150个国家受到网络攻击，已经影响到金融，能源，医疗等行业，造成严重的危机管理问题。中国部分Windows操作系统用户遭受感染，校园网用户首当其冲，受害严重，大量实验室数据和毕业设计被锁定加密。这个最近广受关注的事件的发生也为我们敲醒了警钟，信息安全隐患这个看起来遥不可及的事情绝不是危言耸听，而是实实在在关系着每一个人的生活。

在当今这个信息爆炸的时代，教育、经济、军事、科技等领域都不可避免的要每天进行大量的数据信息交换。这些领域的信息交互就需要即时通讯工具来完成。然而在国内，我们熟悉的即时通讯工具也就是QQ、微信等工具，但是这些工具的安全问题又有较大的隐患，并不能保证学术研究、商业决策、军事机密、科学研究等等对于安全性要求极高的领域的需求。

所以，本文旨在设计开发一款安全性高、性能稳定的安全聊天工具。满足用户的即时通讯的基础上，通过功能和技术上的设计来保证用户聊天数据的安全性。有了这样一款安全聊天工具，就可以让各个领域的机密数据的即时通讯需求，并且尽可能的降低用户重要信息的泄露的风险，让消息更加安全的传递到用户手中。

1.2 国内外研究现状

1.3 论文的主要研究内容

本文研究的是基于RSA非对称加密算法的安全聊天工具的设计与实现。本系统以winform开发平台为基础，使用SuperSocket开源框架来搭建服务器。并且自定义了基于Socket的应用层通信协议，使用反射和工厂模式来实现通信数据的分发和命令的处理。在客户端自主设计并实现了能实现Socket数据通信的封装、数据解析和数据分发等技术的SuperClient框架。实现了安全模块、通信模块、用户信息管理模块、好友管理模块、系统基本设置模块的功能的开发，完成了一款安全性高，性能稳定的安全聊天工具的设计与实现的相关工作。

1.4 论文的组织结构

本文共分为4章，按照系统设计的主次来划分，具体组织方式如下：

第1章主要讲述了安全聊天工具的研究背景和研究的意义、国内外对于安全即时通信的研究现状以及论文的主要研究内容的总体概述。

第2章从应用层通信协议的设计、数据库的表结构和接口设计、服务端的数据解析和分发以及客户端的数据解析和分发这4个角度来解释整个系统是的所有数据通信线路的设计与实现方式。

第3章着重阐明了本系统采用了哪些安全策略来保证用户聊天信息的安全性的。

第4章则以全局的视角来解释安全聊天工具的各个功能模块是如何实现。还展示了系统的功能测试和安全性能测试的结果。

最后是结论部分，对于整个安全聊天工具的设计与实现的过程予以总结。

# 第2章 系统通信架构的设计

本章主要讲述服务端与客户端、客户端的工作线程与UI线程、服务端与数据库之前的数据通信架构的设计工作。包含通信协议的制定、服务端与客户端的数据解析与分发、线程间通信技术、数据库操作等架构的设计。

2.1 基本架构与开发平台的选择

安全的聊天工具的系统架构是C/S架构并且选择winform开发平台。B/S架构与C/S架构是比较常见的两种系统架构方式。下文针对两种架构的特点以及系统的需求说一说对于基础架构方式的选择。

C/S架构也就是Client与Server模式，需要自己开发服务端和客户端，并且需要自己定义应用层通信协议，然后使用Socket网络通信接口来传递数据。使用C/S架构的特点是服务端运行数据负荷较轻，开发的自由度较大，可以定制特定的应用层协议。另外其实现长连接是十分的简单的。同时其缺点也是十分明显，应用需要针对操作系统来开发，系统的维护和更新的成本也相对较高。对于安全聊天工具来说，windows平台足以应对大多数的应用场景，此外服务端与客户端的长连接以及用户数据的安全性也是强需求，所以C/S架构的缺点也可以接受。

B/S可以理解为C/S架构的具体实现，即使用浏览器作为通用的客户端，基于HTTP应用层通信协议，在服务端也有tomcat、nginx等成熟的程序可以使用。除此之外，B/S架构中只有极少的业务逻辑需要在前端实现，绝大部分的事务逻辑都在服务端来进行，这样就大大降低了客户端电脑的负载，也减轻和降低了系统维护和升级的成本和工作量。

有优点自然也有缺点。第一、在B/S架构中，HTTP协议是基于短连接的，想要实现即时通信这类需要保持长连接的功能交为困难；第二、使用B/S架构会暴露系统的服务器地址，导致系统的安全行出现隐患。对于安全聊天工具的设计与实现中，这个问题是十分致命的。

综上所述，使用C/S架构可以自定义应用层通讯协议，容易实现长连接，安全性更容易保证，C/S架构更适合安全聊天工具的开发。而在开发平台的选择上，C#拥有反射、委托等优秀的特性来支撑数据分发等功能需求，开发文档全面以及SuperSocket这样的优秀开源框架这些特点，更适合本系统的开发。

2.2 服务器与数据库的数据交互设计

2.2.1 数据库设计

安全聊天工具涉及的实体的数目较少，主要有用户、好友这两个实体。本系统中的用户使用用户名、密码、私钥签名来验证身份，用户有昵称、头像、签名等个人信息。此外，用户要保存自己的好友用户名、好友备注等信息。为了聊天数据的进一步的安全，在本系统中不存储聊天记录。综合以上的分析，安全聊天工具设计两张数据表。

1. 用户信息表（user）

用户信息表用来存储用户的个人信息，包括用户名、密码、公钥、昵称、个人签名

等信息。其中用户名作为主键，用户信息表如表2.1所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 字段类型 | 字段描述 | 备注 |
| U\_Username | Varcher(50) | 用户名 | 主键，非空 |
| U\_NickName | Varchar (20) | 名字 | 非空 |
| U\_Public\_Key | Varchar (255) | 公钥 | 非空 |
| U\_Password | Varchar (50) | 登录密码 | 非空 |
| U\_SignaTure | Varchar (100) | 个性签名 | null |
| U\_HeadPic | Varchar (50) | 头像地址 | null |

表2.1 用户信息表

1. 好友信息表（friends）

好友信息表用于存储用户的好友信息，包括用户名、好友用户名、好友备注信息等

信息。其中用户名和好友用户名共同为主键，好友用户名为作为外键，好友备注信息默认为用户昵称，好友信息表如表2.2所示。

表2.2 好友信息表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 字段类型 | 字段描述 | 备注 |
| F\_FID | Int | 好友ID | 主键1 |
| F\_UID | Int | 用户ID | 主键2，外键 |
| F\_NoteName | Varchar (20) | 备注 | 默认为昵称 |

2.2.2 数据库操作接口的规定

安全聊天工具中使用MySQL数据库，通过MySQL Dot Net驱动来连接数据库，并且在服务端系统启动后保持长连接。在服务端系统中通过自定义的MDO对象来操作数据库，使用驱动提供的基础方法封装后生成了高一级的数据库操作接口。接口规定了操作名称、参数列表、返回值、备注等信息，接口的详细信息如表2.3所示。

表2.3 数据库操作接口

|  |  |  |  |
| --- | --- | --- | --- |
| 操作名称 | 参数列表 | 返回值 | 备注 |
| Login | 用户名，密码 | 用户信息 | 失败返回null |
| GetFriendList | 用户名 | 好友信息数组 | 请求用户好友列表 |
| UpdateUser | 用户名、昵称、签名 | 影响行数 | 修改用户个人信息 |
| UpdateNote | 用户名、好友用户名、备注 | 影响行数 | 修改好友备注信息 |
| DeleteFriend | 用户名、好友用户名 | 影响行数 | 删除好友 |
| UpdatePw | 用户名、旧密码、新密码 | 影响行数 | 修改用户密码 |
| SearchFriend | 用户名 | 用户信息 | 查找好友信息 |
| AddFriend | 用户名、好友用户名、备注 | 影响行数 | 添加好友 |

2.3 应用层通信协议设计

2.3.1 起止符通信协议的设计

通信协议主要用于服务端与客户端之间的数据传递。本系统中的协议是指服务端或者客户端将接受到的数据分割成一个个数据包的规范。而对于安全聊天工具来说，传输数据的特点是长度不固定但是一般不会超过100个字符，并且对于数据传输的稳定性和可靠性有很高的要求。

常用的应用层通信协议有结束符协议、固定数量分隔符协议、固定请求大小协议、起止符协议、头部格式固定并且包含内容长度的协议等等。在这些协议中，固定数目分隔符协议适用于不需要区分请求类型的系统。固定请求大小协议适用于每一次请求的数据长度均相同的系统。结束符协议是通过固定结束符来区分每一个数据包，可以应对变长的数据，但是稳定性和可靠性相对较差。头部格式固定并且包含内容长度的协议的数据可靠性极高，不过由于头部长度固定，更适合传递数据较长的场景下。综上所述，我们选择了起止符协议作为整个系统的应用层通信协议。

起止符协议的特点是通过固定的头部和固定的尾部来标识一个数据包，这时可能出现第一个问题是当传递的数据中出现系统规定的结尾标志时，会出现数据解析遗漏和解析错误的现象。为了解决这个问题，我选择了8个字节的头部和8个字节的尾部，这种设计出现重复的概率是多少呢？下面我来进行一下论证。

由于每个字节有8个bit位，也就是说每个字节有256中选择，而8个字节就是256的8次方。换句话说，8个字节的固定头部与固定尾部有1/18446744073709551616的概括会出现与规定的头部或者结尾重复的现象。再假设一共有10000000的用户，每个用户平均每天3000条数据要传输，那么一年中出现重复的概率是1/18446744073709551616\*10000000\*3000\*365，这是一个小于0.01%的数字。这样的一个小概率事件，我们有理由相信是不可能发生的。

通过以上的论证证实，这样的应用层通信协议的设定是符合我们的预期设想的，8个字节的固定起止符的通信协议足以满足我们对于数据解析速度和稳定性的需求。

2.3.2 服务端与客户端之间通信接口的规定

基于上述固定起止符协议，服务端与客户端之间通信接口详细的规定了服务端发往客户端以及客户端发往服务端的数据包中的命令、数据项、数据项分隔符，如表2.1，表2.2所示。命令与数据项之间均是通过“：”来分割，数据项之间均是通过“，”来分割。如果数据项是一个完整的数据实体，就会使用json的类的序列化与反序列化的方法来将实体数据转换完json数组，作为一个整体作为数据项来传递。

表2.3 客户端发送到服务端的数据接口说明

|  |  |  |
| --- | --- | --- |
| 命令 | 数据项 | 说明 |
| Login | 用户名，密码，私钥签名 | 登录 |
| AddFriend | 用户名，好友用户名，好友备注 | 添加好友 |
| DeleteFriend | 用户名，好友用户名 | 删除好友 |
| FriendList | 用户名 | 获取好友列表 |
| ReceiveMsg | 接收者，消息类型，消息 | 转发消息 |
| SearchFriend | 用户名 | 查找好友 |
| UpdateNote | 用户名，朋友的用户名，新备注 | 修改好友备注 |
| UpdatePw | 用户名，旧密码，新密码 | 修改登录密码 |
| UpdateUser | 用户名，昵称，签名 | 修改个人信息 |

表2.4 服务端发送到客户端的数据接口说明

|  |  |  |
| --- | --- | --- |
| 命令 | 数据项 | 说明 |
| FriendList | 好友列表的json串 | 好友列表的json形式 |
| Login | 登录结果 | 登陆者的信息，登录失败返回null |
| ReceiveMsg | 发送者，消息类型，消息 | 接收到好友消息 |
| SearchFriend | 查找到结果的json串 | 好友信息，未找到返回null |
| UpdatePw | 修改密码结果 | 修改密码是否成功的结果 |

2.4 服务端数据解析与分发设计

服务端使用SuperSocket开源框架来搭建。SuperSocket是一个轻量级, 跨平台而且可扩展的Socket 服务器程序框架。基于SuperSocket框架，我定义了起止符协议来支持系统的通信架构和一个起止符协议制作工厂用来在SuperSocket中启动自定义的通信协议。

SuperSocket通过一个我定义的ChatServer类来管理服务端，控制服务端的启动、停止以及管理所有连接Session。每当一个客户端连接到服务端时，一个ChatSession就会被启动，这个Session的实例用来用来发送数据到对应的客户端，接受客户端发送的数据或者关闭当前连接。

当服务端收到数据时，所有来自客户端的按照通信协议规定发送的请求都会被起止符协议解析成我定义的ChatReceiveInfo数据包。这个数据包主要包括Key和Body两部分内容，Key代表本次请求的命令，Body代表本次请求的所携带的数据。SuperSocket通过Dot Net反射机制的查看我定义的命令类的类名并与ChatReceiveInfo数据包中Key字段做比较。当发现相同项时，就会生成对应的命令类的对象，并将数据包传到这个类的ExecuteCommand方法中，最后调用这个方法。此时，数据的解析和分发就已经完成，我根据对于请求的不同，使用传递进来的参数做出相应的业务逻辑处理和判断。到此服务端的数据解析与分发的设计就此完成。通过SuperSocket框架，实现了自定义的数据通信协议的制定以及对应的请求的分发和处理工作，并且可以很方便的完成通信协议的更换以及业务的扩展。

2.5 客户端的SuperClient框架设计

Socket是基于传输层通信协议封装实现的编程接口。在服务端已经开启端口监听的基础，只要是按照固定的IP地址和Port建立起Socket连接，就可以发送和接受数据。但是在安全聊天工具中，需要较为复杂的业务判断，并且数据请求的种类较多，数据分发的目标也不能统一。基于这种情况，我设计了SuperClient框架。

2.5.1 SuperClient框架概述

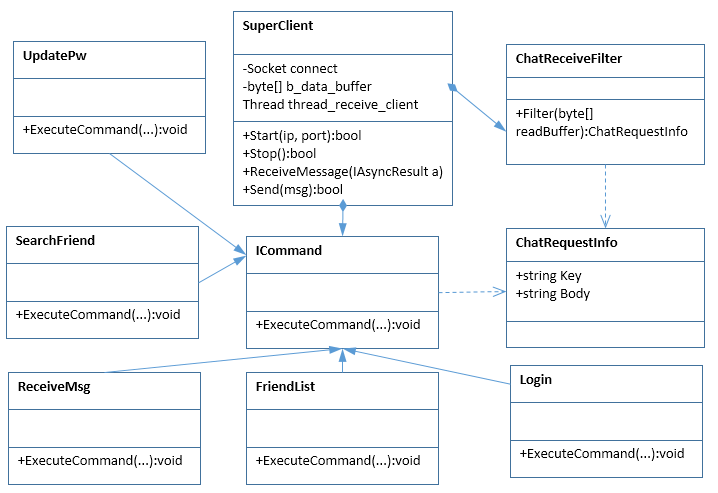
SuperClient框架是基于Socket网络通信编程接口，实现发送客户端请求、解析服务端数据以及数据分发的客户端通信框架。在本框架中，各个功能的控制都在核心类SuperClient中，每一个SuperClient的实例对应一个客户端。使用IP地址与Port作为参数调用对应的方法中就可以建立与服务端的连接，在SuperClient实例中，配置起止符通信协议和Command生成工厂，就可以接收并解析服务端发来的数据包。最后再根据解析出来的命令生成对应的命令的实体，结合数据参数执行对应的业务代码。SuperClient的类图如图2.1所示。

图2.1 SuperClient类图

这里简单的说明一下这个类图，然后在下面的2.4.2和2.4.3中会详细的介绍SuperClient的工作原理。SuperClient类是核心类，客户端的启动、停止、消息的接收和发送都在这个类中操作。ICommand类是所有命令类的父类，其他的命令类都要继承Icommand类并且实现抽象方法ExecuteCommand。ChatReceiveFilter类是负责数据解析，起止符通信协议就在这个类中实现，而解析后的数据包都会被实例化为ChatRequestInfo类。而ChatRequestInfo类作为ExecuteCommand方法的参数来使用。

2.5.2 SuperClient框架的使用方法

SuperClient框架设计的初衷就是让基于Socket的网络通信客户端的开发变得简单。有了它可以让开发者专注于业务逻辑的思考，在设计好应用层的通信协议后就可以轻松的开始业务代码的编写，让Socket客户端的开发变得快速。下面简单的介绍SuperClient的使用方法。

使用SuperClient进行数据通信是很简单的。SuperClient中使用SuperClient类来作为数据通信的核心类，每一个客户端对于一个SuperClient的实体。调用其Start方法和

Stop方法就可以启动和停止通信功能，使用Send方法，就可以实现消息的发送。而接受服务器发送的消息还需要做一点额外的工作。

在SuperClient中内置了起止符协议，在第2章中已经详细介绍了这个协议的特点，基本可以满足常规的开发需求，对于我们的安全聊天工具来说也是非常好的选择。当我们有一类新的数据从服务端发过来的时候，我们只需要编写一个以命令为类名的Command类。这个类需要继承ICommand这个接口并实现其ExecuteCommand方法，在这个方法中就可以通过方法的参数获取到服务端发送的消息。

2.5.3 SuperClient的数据通信原理

熟悉开发框架的基本原理有利于更快的上手使用和开发过程中的bug的排除工作，下面来基于其数据通信的原理为核心，解释一下SuperClient的实现原理。

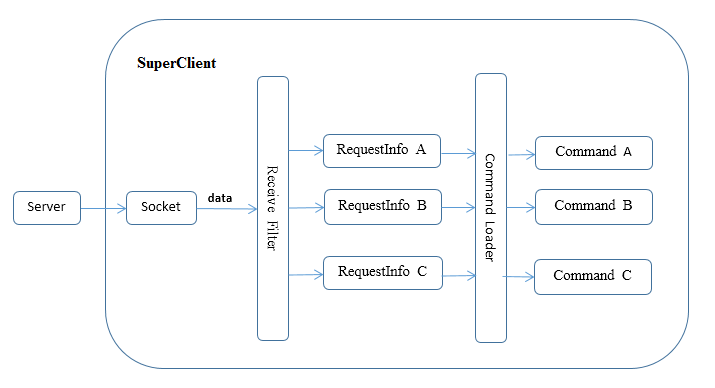
SuperClient是基于Socket网路编程接口实现的数据通信，这是其通信原理的基础。如图2.2所示，当SuperClient与服务器建立连接后，客户端收到的数据都会先缓存到消

图2.2 SuperClient 数据处理模型图

息数组中。框架内置的起止符协议会循环的从缓存中读取数据并从数据找到符合起止符协议规定的一条条完整的请求，每一个请求都会被实例化成ChatRequestInfo类型并返回给SuperClient命令加载器。而SuperClient通过反射的方式查找到哪些公开的类实现了基础的命令接口，进而帮助命令加载器找到与ChatRequestInfo实例的Key相同的类。这时，命令命令加载器就会生成了命令的实例，然后执行其ExecuteCommand方法，通过方法参数将服务端发送过来的数据传递到此命令的ExecuteCommand方法中。综上所述，SuperClient实现了将服务端发送过来的数据解析成规定的形式并且分发到相应的业务处理命令类中。

2.6 多线程通信

SuperClient框架实现了客户端的数据发送、接受、解析与分发的工作，其中数据的分发可以实现不同种类的数据分发到不同的命令类中处理。对于客户端与服务端通信这一类的较为耗时的操作是要在建立新的工作线程来完成的，但是从服务端接收的数据或者消息大多要在主线程中使用。所以，如何解决多线程间的数据通信就是下一个要阐明的问题。

对于安全聊天工具中的线程通信来说，数据流向都是从工作线程流往主线程。此外，在winform开发平台，主线程也同样是UI线程。所以，我们使用Control的Invoke方法通过委托的方式来完成工作线程到主线程的数据通信工作。

首先，要在主线程中建立委托，并且在UI初始化的同时使用UI线程中的数据操作方法来初始化委托。这样当委托被执行的时候，就可以将从服务端接受到的数据传递到UI线程中，在数据操作方法中完成相依的数据操作。然后，还需要在工作线程中调用控件的Invoke方法。当解析后的ChatRequestInfo数据包被分发到相对应的Command类中时，在ExecuteCommand方法中调用所需控件的Invoke方法，并按照参数列表来初始参数，指定UI线程的委托就会被执行。到此，这个线程间数据通信全部完成。

2.7 本章小结

本章主要讲述了安全聊天工具这个系统的全部数据通信结架构的设计工作。本系统中的主要包含服务端与数据库的数据交互，服务端与客户端的数据通信、客户端中还包含工作线程与主线程的线程间的数据通信。此外，本章还包含了系统的数据库设计、应用层通信协议的设计、服务端与客户端通信的命令接口的设计以及客户端通信架构SuperClient的设计。到此，本系统的所有数据传递线路全部打通。

# 第3章 安全性保证策略

作为一款安全聊天工具，安全性能是评估软件好与坏的核心标准。所以在本章着重解释安全聊天工具使用了哪些安全策略来保障用户的通信数据的安全的。本系统的安全性的保障主要是从两个角度来实现，第一、尽最大可能降低用户数据被泄漏的风险；第二、当用户数据被泄露时，也要保证用户隐私安全。

针对以上两点，本系统使用了数据通信加密、用户口令+实体token、阅后即焚功能以及SQL注入防御等方面来保证用户数据的安全性。

3.1 数据加密传输策略

3.1.1 加密算法的选择

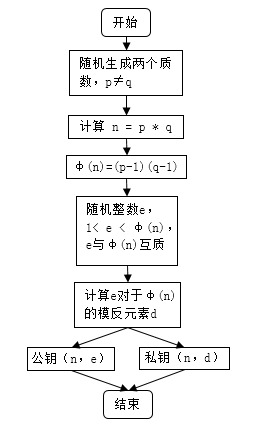
关于加密算法，主要分为对称加密算法和非对称加密算法两大类。对称加密算法中，信息的加密和解密使用的都是同一个密钥。这要一来，数据的发送者的密钥需要使用某种方式来来传递给接受者。这对于安全聊天工具来说，就意味着任何一个消息的接受者都可以获取到发送者的密钥，这对数据的安全性产生了很大的威胁，而非对称加密算法很好的解决了这个问题。

非对称加密算法采用两个相互依赖的密钥，消息发送者使用公钥对消息进行加密，公钥公开，可以在根据需要获取。而私钥保密，不能公开，消息接受者使用自己私钥解密发送给自己的消息。由此可见，非对称加密算法给适合于对安全性要求较高的安全聊天工具。在非对称加密算法中，应用最广泛的就是RSA加密算法，在下一部分将着重解释RSA非对称加密算法的原理和使用在安全聊天工具中的具体应用方法。

3.1.2 RSA加密算法的基本原理

RSA加密算法是目前最有影响力的非对称加密算法，它能够抵抗到目前为止已知的绝大多数密码攻击。RSA算法的核心是基于一个简单的数论事实，那就是两个大质数的乘积的因式分解极为困难[1]。下面我们详细分析一下RSA算法的密钥生成、公钥加密、私钥解密的过程，过程中涉及的数学原理不做解释。

先分析一下密钥的生成过程，如图3.1所示。首选随机选择两个不相等的质数p和q，计算p和q的乘积得到n。然后计算n的欧拉函数φ(n)，φ(n)的值为q-1与p-1的乘积。之后再随机选择一个整数e，条件是1< e < φ(n)，且e与φ(n) 互质。最后在计算e对于φ(n)的模反元素d，密钥生成的过程就完成了。由n和e组成公钥，n和d组成私钥。

图3.1 RSA密钥生成流程图

有了公钥和私钥就可以开始数据的加密和解密任务。假设需要加密的信息m，使用公钥 (n,e) 对m进行加密。这里需要注意，m必须是整数（字符串可以取ascii值或unicode值），且m必须小于n。那么，加密后的数据c ≡ me (mod n)。解密数据的过程类似，使用私钥 (n,d) 来解密，解密的过程为m ≡ cd (mod n)。这样，数据m加密和解密的过程就解释完成。

通过上面的过程，对RSA算法加密的过程有了基本的了解。基于RSA算法的密钥生成、加密解密过程，下面分析一下RSA算法的可靠性。在公钥与私钥的生成过程中一共出现了p、q、n、φ(n)、e、d这6个数字。这六个数字之中，公钥用到了两个（n和e），其余四个数字都是不公开的。其中最关键的是d，因为n和d组成了私钥，一旦d泄漏，就等于私钥泄漏。

现在已知公钥 (n,e) 的条件下，我们按照密钥的生成过程逆向计算私钥 (n,d) 中d数字。ed≡1 (mod φ(n))。只有知道e和φ(n)，才能算出d。e是在公钥中已知的数字，而φ(n)=(p-1)(q-1)。只有知道p和q，才能算出φ(n)。最后，n = pq。只有将n因数分解，才能算出p和q。如果n可以被因数分解，d就可以算出，也就意味着私钥被破解。可是，大整数的因数分解，是一件非常困难的事情。目前，除了暴力破解，还没有发现别的有效方法。在维基百科中有如下记录[2]，"对极大整数做因数分解的难度决定了RSA算法的可靠性。换言之，对一极大整数做因数分解愈困难，RSA算法愈可靠。假如有人找到一种快速因数分解的算法，那么RSA的可靠性就会极度下降。但找到这样的算法的可能性是非常小的。今天只有短的RSA密钥才可能被暴力破解。到2016年为止，世界上还没有任何可靠的攻击RSA算法的方式。只要密钥长度足够长，用RSA加密的信息实际上是不能被解破的。"

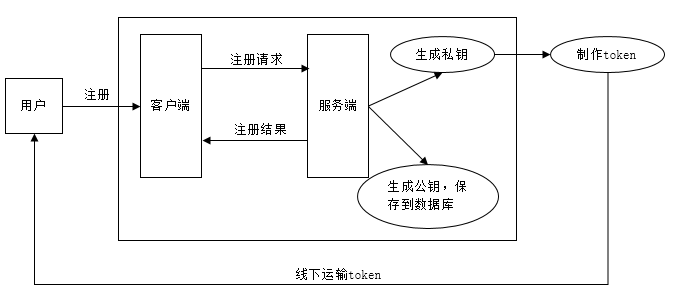
基于以上分析证明，在安全聊天工具中使用RSA加密算法可以保证数据即使被截获的情况也能极大的提升数据的安全性。在私钥没有泄露且使用了足够长的密匙的条件下，用户的数据被破解的可能极低的。

3.1.3 公钥与私钥在系统中的应用

在了解了RSA加密算法的原理之后，下面阐述一下如何在安全聊天工具系统中使用RSA加密来保证用户数据安全。

首先解释密钥的生成的过程。为了避免私钥的泄露，当私钥生成之后，严格禁止通过网络的方式来将私钥交到用户手中。基于这一点要求，有两个方案来生成密钥。第一种方案是在客户端生成密钥。由于公钥是可以公开出去的，所以在客户端生成密钥之后可以通过网络将公钥传递到服务端保存好，待有需要时通过网络获取即可。但是这样的缺点就是用户在更换pc机后无法获取自己的私钥，这样也就无法解密发送给自己的数据。第二种方案是当用户发出注册请求时，在服务器生成密钥如图3.2所示。将公钥直接保存在服务端的数据库中，将私钥直接在服务端通过外部生产环境制作成实体的token，然后在通过现实环境将私钥以实体token的形式交到用户手中。这样的方式虽然会在注册和用户使用时造成一些时间上的延长，但是带来的好处也是巨大的。有了实体token的存在，用户就可以在任意安装了安全聊天工具客户端的pc机上使用自己的帐号来完成即时通讯。更重要的是，用户数据的安全性得到极大的提升。所以，本系统采用第二种方法来生成密钥。

有了密钥再说明一下用户加密和解密数据的过程，这里对RSA加密算法做了一个小的改进，本系统中除了对聊天消息进行数据加密外，还使用数字签名的方法来验证数据是否从预期的发送者发送过来，下面来具体解释一下加密解密的过程。来当用户需要给自己的某一个好友发送消息时，首先使用自己的私钥加密自己的用户名，然后将密文与要发送的数据拼接成一个字符串。而客户端会直接向服务端请求此好友的公钥，使用公钥将私钥加密后的用户名与要发送的数据一起加密，然后打包发送给好友。当好友收

图3.2 密钥生成模型图

到消息后，首先使用自己的私钥将消息解密，然后再向服务端请求消息发送者的公钥来解密消息前一段信息。如果解密出来的用户名与发送者的用户名相同，此消息才是有效消息并传递到主线程；如果不相同则舍弃消息并向用户和服务端报警，提示有恶意的攻击行为发生。到此，安全聊天工具通过数据的加密解密的方式来保证数据安全的过程全部讲述完成。

3.2 身份验证增强策略

比较常见的用户身份验证策略是使用“用户名+用户口令”的方式，用户进行身份验证时，只需要输入自己用户名和密码，由客户端向服务端发出验证请求。然后通过比较用户输入与数据库中数据来返回结果，如果正确用户身份验证成功，允许下一步操作。这样的策略的风险就是当用户的口令泄露时，如果用户不能第一时间发现并更改密码，用户的数据安全就面临这极大的威胁。为了降低这样的风险，本系统在原有的“用户名+用户口令”的身份验证策略的基础上，增加了“签名验证”的策略来进一步提高身份验证的安全系数。

在安全聊天工具的系统中，使用的身份验证的流程如图3.3所示。用户在客户端输入用户名密码，客户端使用本地的私钥加密用户名，然后将用户名、密码、加密后的签名信息一同发送给给服务器，并且请求验证用户身份。服务端监听到客户端的请求后，先根据第一个用户名从数据库中找到用户的公钥，然后使用公钥将签名信息解密。将解密结果与第一个用户名做比较，如果相同再比较用户密码是否正确，最后将验证结果返回给客户端，这就是一个完整的用户身份验证的过程。

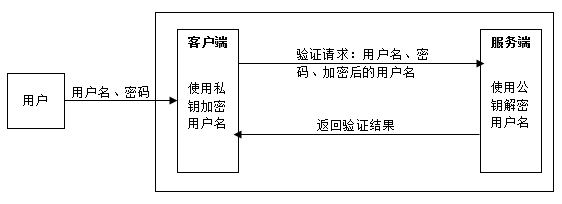
通过“用户名+用户口令+用户签名”的双重保护策略下，即时发生用户口令泄露这样的事件，也有用户签名验证这样的安全策略来保障用户的个人信息的安全。

图3.3 用户身份验证模型图

* 1. SQL注入防御策略

SQL注入是一种常见服务器攻击方式，攻击者在系统的输入域中插入恶意的SQL语句，欺骗服务端进行错误的操作，这样可能导致数据库的数据安全遭到威胁，用户的个人信息有泄漏的风险。对于SQL注入，必须采取相应的防御策略来保证安全聊天工具的服务端的正常运行。

* + 1. SQL注入的基本原理

当我们想要获取数据库中一些特定的数据时，必须要做的就是根据查询数据的特点构造符合我们需要的SQL语句。当SQL语句还需要用户输入的一些信息来辅助查询的时候，这时候就存在一种风险，下面通过一个例子来简单说明这种风险。

假设在系统的数据库有一张User（id， name）表，User表的结构比较简单，只有id和name两个字段。当我们需要查询用户id为1001的用户的name的时候，通过会写出这样的一条SQL语句：SELECT name FROM User WHERE id = ‘1001’，这时候id这个数据1001一般是由用户输入的。在理想的情况下，用户输入id号，我们构造SQL语句，到数据库中执行，然后取回name结果返回给用户，整个流程完成。但是，如果用户输入的id并不是我们预期的数字，而是“1001’ or ‘1’ = 1”的数据，我们的整个SQL语句变成了这样：SELECT name FROM User WHERE id = ‘1001’ or ‘1’ = 1。显而易见的是，不论用户输入的id数字是否在数据库中存在，都会因为or 后面的语句导致WHERE语句恒成立，这样就可以取到数据库表中所有的用户的用户名。这虽然只是一个很简单例子，但是SQL注入攻击的威力可见一斑。如果攻击者在用户输入域中输入更复杂的查询语句，就会导致更严重的数据泄露、数据丢失等严重威胁。所以，我们必须采取相应的策略来防御SQL注入的攻击。在安全聊天工具的开发中，共采用了两种安全策略来防御SQL注入的攻击，具体的防御方式，将在下面的两个部分做详细的解释。

* + 1. 关键词过滤

为了应对SQL注入的攻击，本系统采用的第一种防御策略是关键词过滤。在SQL注入的基本原理的讲解部分中提到，SQL注入都是从攻击者的非法输入开始的。而一般需要通过输入一些关键词、特殊符号等语句来完成完成SQL注入攻击。所以，本文从这个基本点入手，通过关键词过滤的角度来防御SQL注入攻击。

本系统的关键词过滤工作主要集中在客户端进行，在收集用户的输入信息后，首先使用关键词过滤组件将输入中出现的“SELECT、UPDATE、INSERT、DELETE、AND、OR、分号、注释符号”等等在SQL语法中可能使用的关键词以及符号全部删除。将攻击者输入的信息过滤掉特殊的关键词后在代入SQL语句中，这样就从一定程度上降低了被SQL注入攻击的风险。

* + 1. 参数绑定

在关键词过滤的安全策略中，可能会出现关键词遗漏的情况。另外，攻击者还可能将SQL注入的语句转换为16进制的字符输入进系统，这时候关键词过滤的保护策略就会出现失效的风险。为了应对这种逃避关键词过滤的攻击方式，本系统使用了另种更严格的防御策略，参数绑定安全策略。

这是一种通过SQL语句的参数绑定的方式来防御SQL注入的安全策略。在解释SQL的基本原理的部分时，本文提到了一个User表的例子。在那个例子中使用的SQL语句是：SELECT name FROM User WHERE id = ‘1001’。其中预期的参数只有id一个，这条语句的含义是找到用户id是1001的用户的名字。但是，在SQL注入攻击时，语句是：SELECT name FROM User WHERE id = ‘1001’ or ‘1’ = 1。输入的参数并没有全部作为id来使用，而是因为输入的特殊性改变了原本SQL语句的含义。这条语句的含义变成了返回User表中所有用户的名字，条件部分因为输入全部失效了。由此可以得出结论，如果我们可以将攻击者输入的全部内容作为‘id’来执行SQL语句，那么结果就会跟我们预期的保持一致了。按照这样的想法，参数绑定后的的SQL语句变成了下面的样子：SELECT name FROM User WHERE id = ‘(1001’ or ‘1’ = 1’ )’。解释出来就是：找到User表中id是 (1001’ or ‘1’ = 1) 的用户的name。显然这个用户id是不存在的，也就查询不到name的结果集。这样也就成功的防御了SQL注入攻击，这种参数绑定的安全策略是有效的。

* 1. 阅后即焚安全策略

以上都是从技术的层面来实现安全性能保证的，本模块我们从功能的角度来进一步增强系统的安全性。在安全聊天工具中提供了“阅后即焚”的特色功能，来让用户的信息的保密级别再次升级。如果用户通过本系统与好友交流的信息对保密性要求极高，高到最好只让对方看几秒钟就从系统中消息的层次，可以使用“阅后即焚”的安全策略。

阅后即焚功能是指

3.5 本章小结

# 第4章 安全聊天工具的详细设计与实现

4.1 系统功能模块的设计与实现

4.1.1 登录模块

4.1.2 通信模块

4.1.3 用户信息管理模块

4.1.4 好友管理模块

4.1.5 系统基本设置模块

4.2 系统的功能测试与安全测试

4.3 本章小结

参考文献

[0] 罗艺荣. 学生宿舍管理系统的设计与实现[D].厦门大学,2013.

[1] RSA加密算法原理, http://www.ruanyifeng.com/blog/2013/07/rsa\_algorithm\_part\_two. html

[2] 维基百科, RSA加密算法 说明