

**物联网安全系统**

**设计文档初稿**

|  |  |  |  |
| --- | --- | --- | --- |
| **组长：** | 魏子杰 | **指导老师：** | 姚宏 |
| **组员：** | 李迎光 | **职 称：** | 教授 |
| **组员：** | 胡国煜 | **院 系：** | 计算机 |
| **组员：** | 王琛 | **专 业：** | 网络工程 |

2019 年 4 月

**文档变更记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **变更（+/-）说明** | **日期** | **批阅人** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1. 引言 1](#_Toc25219)

[§1.1引言 1](#_Toc26867)

[1.1.1 背景 1](#_Toc765)

[1.1.2 编写目的和范围 1](#_Toc22021)

[1.1.3 涉及加密算法与协议及使用表 1](#_Toc15849)

[1.1.4 参考资料 1](#_Toc22007)

[1.1.5 使用的文字处理和绘图工具 2](#_Toc1681)

[2. 设计概述 2](#_Toc4773)

[§2.1任务和目标 2](#_Toc15409)

[2.1.1 需求概述 2](#_Toc8169)

[2.1.2 运行环境概述 2](#_Toc13981)

[2.1.3 源代码管理工具 2](#_Toc17435)

[2.1.4 设计工具 2](#_Toc8510)

[3. 系统需求分析 3](#_Toc18865)

[§3.1详细需求分析 3](#_Toc24246)

[3.1.1 功能需求分析 3](#_Toc19128)

[3.1.2 性能需求分析 3](#_Toc17137)

[3.1.3 资源需求分析 3](#_Toc11580)

[3.1.4 系统运行环境及限制条件分析 3](#_Toc7787)

[3.1.5 Log文件 3](#_Toc32734)

[4. Kerberos认证过程 4](#_Toc11503)

[5. 设计图示 6](#_Toc8233)

[6. 报文标准 10](#_Toc19239)

[7. 接口设计 12](#_Toc8283)

[§7.1 聊天类 12](#_Toc12709)

[§7.2 加密类 13](#_Toc20677)

[§7.3 Kerberos认证过程类 14](#_Toc10585)

[8. 设计和开发规范 16](#_Toc12533)

[§8.1 数据库设计规范 16](#_Toc3210)

[§8.2 小组成员编码规范 17](#_Toc28474)

# 

# 引言

## §1.1引言

### 背景

在互联网以爆炸的形势发展后，网络安全成为了一个不得不考虑的问题，我们传递的密码可能在网络上被复制，传递的消息可能在网络上被篡改，甚至我们的身份还可能因重放而被仿冒，所以寻求一个相对安全的物联网系统显得至关重要。

### 编写目的和范围

本详细设计说明书编写的目的是说明程序模块的设计考虑，包括程序描述、输入/输出、算法和流程逻辑等，为软件编程和系统维护提供基础。

### 涉及加密算法与协议及使用表

表 1-1

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **术语或缩略语** | **说明** | **使用方式** |
| 1 | DES | DataEncryptionStandard,数据加密标准 | Kerberos认证过程，聊天的消息传递过程 |
| 2 | RSA | Ron Rivest，Adi Shamir，Leonard Adleman一起提出的非对称加密算法 | 用户向AS注册时的证书验证 |
| 3 | Kerberos认证协议 | 一种计算机网络授权协议，用来在非安全网络中，对个人通信以安全的手段进行身份认证 | 用户请求访问服务的验证过程 |

### 参考资料

表 1-2

|  |  |  |
| --- | --- | --- |
| 资料名称 | 作者 | 文件编号、版本 |
|  |  |  |

### 使用的文字处理和绘图工具

文字处理软件：Office、WPS

绘图工具：Visio

# 设计概述

## §2.1任务和目标

### 需求概述

在自定义应用的基础上，实现数据传输安全化、kerberose认证体系、证书颁发。

用已经实现的DES、RSA进行加密、签名。

主要功能：实现一个IM即时聊天软件，能够实现用户之间的在线对话，对话内容经过加密传送到对方，由对方解密呈现给用户。 Client通过访问AS认证服务器用来认证Client的身份，获取票据许可票据ticket，然后访问TGS票据许可服务器获取服务许可票据、这个过程中会返回会话密钥，通过服务许可票据访问服务器Server，获取IM软件的通信服务，通过会话密钥加密聊天信息实现Client与Server聊天会话。

证书用来认证服务器Server是否是真正的Server，由服务认证阶段AS返回给Client。Client通过对证书的认证来判断是否需要继续访问该Server。

输入、输出：聊天信息。

性能需求：要求支持并发访问服务器、加密解密时间要快、安全性高、可靠性好。

### 运行环境概述

1. 操作系统：Windows 10
2. 数据库：mysql 8
3. 服务器：Tomcat 8

### 源代码管理工具

源代码管理工具选用git

### 设计工具

1. 数据流程图显示整个程序运行后的数据走向
2. Java开发工具：IntelliJ IDEA
3. 数据库：workbench

# 系统需求分析

## §3.1详细需求分析

### 功能需求分析

1. 各个用户能够注册自己的账号
2. 各个用户在忘记密码后能够通过密码找回功能对密码进行找回
3. 各个用户在登陆上系统后能够与在线用户在聊天室进行消息的传递
4. 所有的信息传递均需要进行加密

### 性能需求分析

1. 用户登陆系统时间不宜过长，用户体验要比较良好
2. 聊天过程中消息的传递速度要与现有的聊天软件，如QQ，微信基本一致

### 资源需求分析

1. 对带宽的要求不宜过高
2. 对内存的占用不宜过高

### 系统运行环境及限制条件分析

1. 四台主机操作系统均为windows10，一台为AS，一台为TGS，一台为Server，最后一台为Client，其中任意的主机均可充当Client身份。
2. 限制条件，局域网条件，不可连接外网，仅允许局域网内的接入服务的用户可相互发送信息交流。

### Log文件

1. 出错信息
2. 注册信息出错

信息1：AS响应超时。

信息2：插入数据库出错。

信息3：证书认证失败。

1. Kerberos认证样例

信息1：AS（TGS，V）响应超时。

信息2：AS（TGS，V）认证失败。

1. 服务过程出错

信息1：消息响应超时。

1. 正常访问信息
2. 注册信息样例

信息：XXX用户进行注册。 时间：XX月XX日XX时XX分XX秒。

1. Kerberos认证样例

信息1：XXX用户获取AS票据服务票据。 时间：XX月XX日XX时XX分XX秒。

信息2：XXX用户获取TGS访问服务票据。 时间：XX月XX日XX时XX分XX秒。

信息3：XXX用户获取Server认证。 时间：XX月XX日XX时XX分XX秒。

1. 聊天室登入退出样例

信息1：XXX用户接入聊天室。 时间：XX月XX日XX时XX分XX秒。

信息2：XXX用户退出聊天室。 时间：XX月XX日XX时XX分XX秒。

# Kerberos认证过程

1. 服务认证交换：获得票据许可票据
2. C → AS : IDc || IDtgs || TS1
3. AS → C : EKc[Kc,tgs || IDtgs || TS2 || Lifetime2 || Tickettgs]

Tickettgs = EKtgs[Kc,tgs || IDc || ADc || IDtgs || TS2 || Lifetime2]

1. 服务器授权票据交换：获得服务许可票据
2. C → TGS : IDv || Tickettgs || Authenticatorc
3. TGS → C : EKc,tgs[Kc,v || IDv || TS4 || Ticketv]

Tickettgs = EKtgs[Kc,tgs || IDc || ADc || IDtgs || TS2 || Lifetime2]

Ticketv = EKv[Kc,v || IDc || ADc || IDv || TS4 || Lifetime4]

Authenticatorc = EKc,tgs[IDc || ADc || TS3]

1. 客户/服务器认证交换：获得服务
2. C → V : Ticketv || Authenticatorc
3. V → C : EKc,v[TS5 + 1]

Ticketv = EKv[Kc,v || IDc || ADc || IDv || TS4 || Lifetime4]

Authenticatorc = EKc,tgs[IDc || ADc || TS5]

表 5-3 Kerberos认证协议成分说明

|  |  |  |
| --- | --- | --- |
| **序号** | **成分** | **说明性定义** |
| 1 | C | 客户端（用户） |
| 2 | AS | 认证服务器 |
| 3 | TGS | 票据授权服务器 |
| 4 | V | 服务器（应用程序） |
| （1）C → AS | | |
| 5 | IDc | Client端的用户标识 |
| 6 | IDtgs | 告诉AS用户请求访问TGS |
| 7 | TS1 | 让AS验证client端的时钟是与AS的时钟同步的 |
| （2）AS → C | | |
| 8 | EKc | 基于用户口令的加密，使得AS和client可以验证口令，  并保护(2) |
| 9 | Kc,tgs | session key的副本，由AS产生，client可用于在AS与client之间信息的安全交换，而不必共用一个永久的key |
| 10 | IDtgs | 确认这个ticket是为TGS制作的 |
| 11 | TS2 | 告诉client该ticket签发的时间 |
| 12 | Lifetime2 | 告诉client该ticket的有效期 |
| 13 | Tickettgs | client用来访问TGS的ticket |
| （3）C → TGS | | |
| 14 | IDv | 告诉TGS用户要访问服务器V |
| 15 | Tickettgs | 向TGS证实该用户已被AS认证 |
| 16 | Authenticatorc | 由client生成，用于验证ticket |
| （4）TGS → C | | |
| 17 | EKc,tgs | 用仅由C和TGS共享的密钥Kc,tgs保护(4) |
| 18 | Kc,v | session key的副本，由TGS生成，供client和server之间信息的安全交换，而无须共用一个永久密钥 |
| 19 | IDv | 确认该ticket是为server V签发的 |
| 20 | TS4 | 告诉client该ticket签发的时间 |
| 21 | TicketV | client用以访问服务器V的ticket |
| 22 | Tickettgs | 可重用，从而用户不必重新输入口令 |
| 23 | EKtgs | ticket用只有AS和TGS才知道的密钥加密，以预防篡改 |
| 24 | Kc,tgs | TGS 可用的session key副本，用于解密authenticator,从而认证ticket； |
| 25 | IDc | 指明该ticket的正确主人 |
| 26 |  |  |
| （5）C → V | | |
| 27 | Ticketv | 向服务器证实该用户已被AS认证 |
| 28 | Authenticatorc | 由客户生成，用于验证ticket有效 |
| （6）V → C | | |
| 29 | Ekc,v | 使C确认报文来自V |
| 30 | TS5＋1 | 使 C确信这不是报文重放 |
| 31 | TicketV | client用以访问服务器V的ticket |
| 32 | EKv | 用只有AS和TGS才知道的密钥加密的票据，以预防篡改 |
| 33 | Kc,v | 用户的会话密钥副本 |
| 34 | IDc | 票据的合法用户 |
| 35 | ADc | 防止非法使用 |
| 36 | IDv | 使服务器确信解密正确 |
|  |  |  |
|  |  |  |

# 设计图示

## §5.1设备部署图&流程图&时序图

1. 设备部署图



图 5-1

1. 流程图



图 5-2

1. 时序图



图 5-3

## §5.2状态机示意图（并发）

### Kerberos认证示例



图 5-4



图 5-5



图 5-6



图 5-7



图 5-8



图 5-9

### 访问Server示例



图 5-10



图 5-11



图 5-12



图 5-13



图 5-14



图 5-15



图 5-16

## §5.3 UI界面&函数调用

1. Kerberos登陆认证



图 5-17

1. 认证调用函数：

public String client\_to\_as(String ID\_c，String ID\_tgs，Date TS\_1);

public String[] as\_parse\_client(String message);

public void send\_message(String message);

public String client\_to\_tgs(String ID\_v，String Ticket\_tgs，String Authenticator\_c);

public String[] tgs\_parse\_client(String message);

public String tgs\_to\_client(String K\_c\_v，String ID\_v，TimeStamp TS\_4，String Ticket\_v);

public String[] client\_parse\_tgs(String message);

public String client\_to\_v(String Ticket\_v，String Authenticator\_c);

public String[] v\_parse\_client(String message);

public String v\_to\_client(String K\_c\_tgs，String ID\_tgs，TimeStamp TS\_2，String Lifetime\_2,String Ticket\_tgs);

public String[] client\_parse\_v(String message);

public String encrypt(BigInteger m, BigInteger[] pubkey)

public String decrypt(BigInteger c, BigInteger[] selfkey)

public String sign(BigInteger hm, BigInteger[] selfkey)

public String verify(BigInteger s, BigInteger[] pubkey)

1. 注册调用函数

public String register(String ID\_c，String ID\_key);

public void add(String user\_id,String user\_password);

public bool find(String user\_id,String user\_password);

public String get\_Certification();

public boolean verify\_Certification();

1. 输入本次聊天昵称，显示Kerberos认证

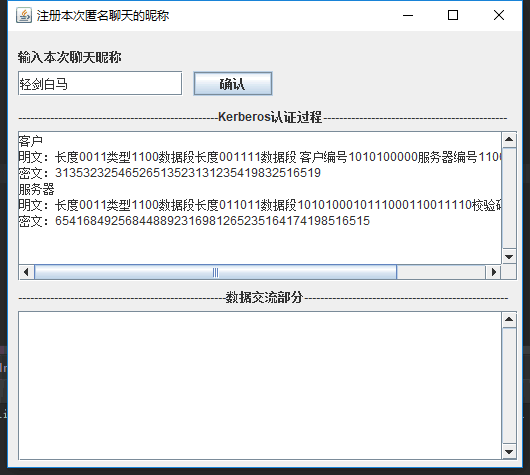


图 5-18

public void init();

public void deal(Socket socket);

public void send\_message(String message);

public String encrypt(BigInteger m, BigInteger[] pubkey)

public String decrypt(BigInteger c, BigInteger[] selfkey)

public String sign(BigInteger hm, BigInteger[] selfkey)

public String verify(BigInteger s, BigInteger[] pubkey)

public Bitset string\_to\_bitset(String text);

public String bitset\_to\_string(Bitset bitset);

public String[] split(String message);

public String[] split(String text, int n);

1. 聊天过程

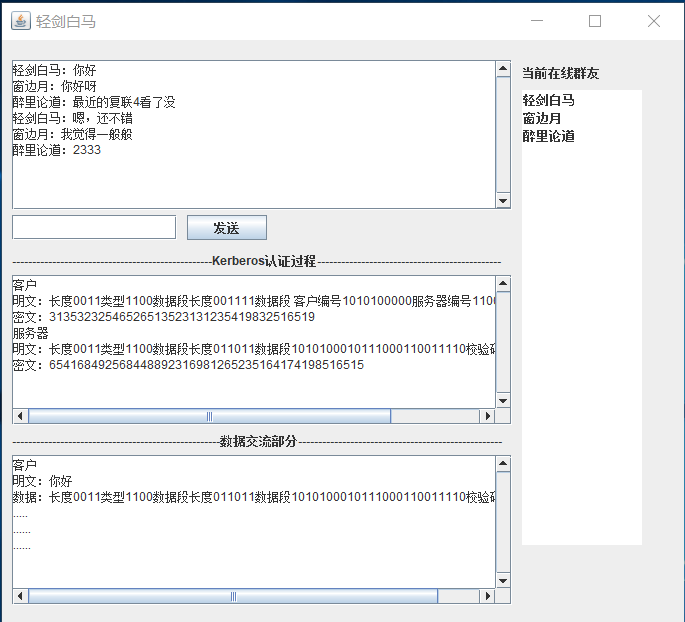


图 5-19

public int get\_online\_num();

public void broadcast(String message);

public void send\_message(String message);

public String encrypt(BitSet M, Int index);

public Bitset string\_to\_bitset(String text);

public String bitset\_to\_string(Bitset bitset);

public String[] split(String message);

public String[] split(String text, int n);

public String[] encrypt(BigInteger[] m, BigInteger[] pubkey)

public String[] decrypt(BigInteger[] c, BigInteger[] selfkey)

public String sign(BigInteger hm, BigInteger[] selfkey)

public String verify(BigInteger s, BigInteger[] pubkey)

# 报文标准（String类型）

1. 服务认证交换

C → AS

|  |  |  |
| --- | --- | --- |
| IDc  (4 字节) | IDtgs (2 字节) | TS1 (13 字节) |

AS → C

|  |  |  |  |
| --- | --- | --- | --- |
| Kc,tgs  (7字节)  (56 位 2进制) | IDtgs (2 字节) | | TS2 (13 字节) |
| Lifetime2  (13 字节) | | Tickettgs() | |

Tickettgs 经过K\_tgs加密

|  |  |  |
| --- | --- | --- |
| Kc,tgs (56 位 2进制) | IDc (4 字节) | ADc (12 字节) |
| IDtgs (2 字节) | TS2 (13 字节) | Lifetime2 (13 字节) |

AS → C出错码

|  |
| --- |
| Error (4字节) |

表 6-1

|  |  |
| --- | --- |
| Error | 出错信息 |
| 0000 | 用户ID不存在数据库中 |
| 0001 | 用户口令与数据库不匹配 |

1. 服务器授权票据交换

C-->TGS

|  |  |  |
| --- | --- | --- |
| IDv (2 字节) | Tickettgs | Authenticatorc |

TGS-->C

|  |  |  |
| --- | --- | --- |
| Kc,v (56 位 2进制) | IDv (2 字节) | TS4 (13 字节) |
| Ticketv | | |

Tickettgs ：56位DES密钥 EKtgs 加密

|  |  |  |
| --- | --- | --- |
| Kc,tgs (56 位 2进制) | IDc (4 字节) | ADc (12 字节) |
| IDtgs (2 字节) | TS2 (13 字节) | Lifetime2 (13 字节) |

Authenticatorc：56位DES密钥 EKc,tgs 加密

|  |  |  |
| --- | --- | --- |
| IDtgs (2 字节) | ADc (12 字节) | TS3 (13 字节) |

TicketV ：56位DES密钥 EKv 加密

|  |  |  |
| --- | --- | --- |
| Kc,v (56 位 2进制) | IDc (4 字节) | ADc (12 字节) |
| IDv (2 字节) | TS4 (13 字节) | Lifetime4 (13 字节) |

1. 客户/服务器认证交换

C-->V

|  |  |  |
| --- | --- | --- |
| IDv (2 字节) | TicketV | Authenticatorc |

V-->C（使用）Kc,v来进行加密

|  |
| --- |
| TS5+1 |

Authenticatorc：56位DES密钥 EKc,tgs 加密 加密前29字节

|  |  |  |
| --- | --- | --- |
| IDc (4 字节) | ADc (12 字节) | TS5 (13 字节) |
| N（用户公钥） | | E（用户公钥） |

TicketV ：56位DES密钥Kc,v加密 加密前51字节

|  |  |  |
| --- | --- | --- |
| Kc,v (56 位 2进制 7字节) | IDc (4 字节) | ADc (12 字节) |
| IDv (2 字节) | TS4 (13 字节) | Lifetime4 (13 字节) |

1. 注册

C-->AS

|  |
| --- |
| Request (4 字节) |

Request默认为0000

AS-->C

|  |
| --- |
| Certification |

Certification

|  |  |  |
| --- | --- | --- |
| IDas  (2 字节) | PKas ( 字节) | Sig ( 字节) |

C-->AS

|  |  |
| --- | --- |
| 用户ID (4 字节) | 用户口令(6-10字节) |

Request默认为0000，IDc从0001一直到9999

出错AS-->C

|  |
| --- |
| Error (4字节) |

表 6-2

|  |  |
| --- | --- |
| Error | 出错信息 |
| 0002 | 用户ID已存在数据库中 |

1. 聊天室

C-->V：使用56位对称sessionKey进行加密

|  |  |
| --- | --- |
| ST (4 字节) | Message位数（8 字节） |
| Message (N 字节) | Sig（Hash（Message）） (M 字节) |

ST：1000（退出），此时后面都不需要添加

ST：1001（消息发送）

V-->C：使用56位对称sessionKey进行加密

|  |  |  |  |
| --- | --- | --- | --- |
| ST (4 字节) | ClientID | | Message位数（8 字节） |
| Message (N 字节) | | Sig（Hash（Message）） (M 字节) | |

ST：为1000时，后面接退出的用户名

ST：为1001时，后面接续与C-->V一致

ST：为1002时，后面接进入聊天室的用户名

# 接口设计

## §7.1 聊天类

**ChatRoomServer接口**

/\*\*

\*开启socket监听，等待处理用户连接

\*/

public void init()

/\*\*

\*处理用户连接

\* @socket 开启线程服务用户

\*/

public void deal(Socket socket)

/\*\*

\*返回用户数量

\*/

public int get\_online\_num()

/\*\*

\*将接收到的一条用户消息发送给所有在线用户

\*/

public void broadcast(String message)

**ChatRoomClient接口**

/\*\*

\*开启socket，连接服务器

\*/

public void init()

/\*\*

\*向聊天室发送信息mes

\*/

public void send\_message(String mes)

**RegisterClient接口**

/\*\*

\*向AS发送注册请求

\* @request 请求字符串

\*/

public void register\_request(String request)

/\*\*

\*验证AS回传的证书签名

\* @certification AS回传的证书

\*/

public bool verify\_certification(String certification)

/\*

\* @id 客户向AS发送的注册ID

\* @command 客户向AS发送的注册口令

\* @PK AS回传的证书中的公钥

\*向AS发送经过AS回传的证书中的公钥进行加密的ID和command

\*/

public void send\_id\_command(String id,String command,String PK)

## §7.2 加密类

**RSA接口**

/\*\*

\* 通过此接口把String类型数据转换为BigInteger类型数据,n是要把text以n byte大小分割再转换为BigInteger[]

\*/

public BigInteger[] string\_to\_big(String text,int n);

/\*\*

\* 通过此接口把BigInteger类型数据转换为String类型数据。

\*/

public String big\_to\_string(BigInteger[] bigInteger);

/\*\*

\* 把String数据分割为小于等于n的String[]。

\*/

public String[] split(String text,int n);

/\*\*

\* 加密

\* @param m 被加密的信息转化成为大整数m

\* @param pubkey 公钥

\*/

public String[] encrypt(BigInteger[] m,BigInteger[] pubkey)

/\*\*

\* 解密

\* @param c

\* @param selfkey 私钥

\*/

public String[] decrypt(BigInteger[] c,BigInteger[] selfkey)

/\*\*

\* 签名

\* @param hm 消息经过hash后的值

\* @param selfkey 私钥

\*/

public BigInteger sign(BigInteger hm,BigInteger[] selfkey)

/\*\*

\* 认证

\* @param s 签名

\* @param pubkey 公钥

\*/

public BigInteger verify(BigInteger s,BigInteger[] pubkey)

**DES接口**

/\*\*

\* 通过此接口把String[]类型数据转换为二进制的BitSet[]类型数据。

\*/

public Bitset[] strings\_to\_bitsets(String[] message\_array);

/\*\*

\* 通过此接口把二进制的BitSet[]类型数据转换为String类型数据。

\*/

public String bitsets\_to\_string(Bitset[] bitset);

/\*\*

\* 把String数据分割为小于等于64bit的String[]。

\*/

public String[] split(String message);

/\*\*

\* M 原文/密文 对应 0/1

\* 加密、解密方法， 加密时候，M是原文，index = 0 .解密时，M是密文，index = 1.

\*/

public String encrypt(BitSet M,Int index);

## §7.3 Kerberos认证过程类

**Client-->AS**：

/\*\*

\* Client 向AS申请 获取服务许可票据

\*/

public String client\_to\_as(String ID\_c,String ID\_tgs,Date TS\_1);

/\*\*

\* AS 解析 从Client发过来的请求报文

\* 把Message解析到Stirng[], String数组元素为：ID\_c || ID\_tgs || TS\_1

\*/

public String[] as\_parse\_client(String message);

**AS-->Client**：

/\*\*

\* AS向Client发送认证票据报文

\*/

public String as\_to\_client(String K\_c\_tgs,String ID\_tgs,Date TS\_2,String Lifetime\_2,String Ticket\_tgs);

/\*\*

\* 解析AS发给Client的报文message

\* 把Message解析到Stirng[], String数组元素为：K\_c\_tgs || ID\_tgs || TS\_2 || Lifetime\_2 || Ticket\_tgs

\*/

public String[] client\_parse\_as(String message);

**Client-->TGS**：

/\*\*

\* Client 向TGS申请 获取服务许可票据

\*/

public String client\_to\_tgs(String ID\_v,String Ticket\_tgs,String Authenticator\_c);

/\*\*

\* TGS 解析 从Client发过来的请求报文

\* 把Message解析到Stirng[], String数组元素为：ID\_v || Ticket\_tgs || Authenticator\_c

\*/

public String[] tgs\_parse\_client(String message);

**TGS-->Client**：

/\*\*

\* TGS 向Client发送服务许可票据报文

\*/

public String tgs\_to\_client(String K\_c\_v,String ID\_v,Date TS\_4,String Ticket\_v);

/\*\*

\* Client 从TGS返回的报文中解析出服务许可票据

\* 把Message解析到Stirng[], String数组元素为：K\_c\_v || IDv || TS\_4 || Ticket\_v

\*/

public String[] client\_parse\_tgs(String message);

**Client-->Server V**：

/\*\*

\* Client 向Server V 获取服务

\*/

public String client\_to\_v(String Ticket\_v,String Authenticator\_c);

/\*\*

\* Server V 解析 从Client发过来的请求报文

\* 把Message解析到Stirng[], String数组元素为：Ticket\_v || Authenticator\_c

\*/

public String[] v\_parse\_client(String message);

**Server V-->Client**：

/\*\*

\* Server V 向Client发送相互认证报文

\*/

public String v\_to\_client(String K\_c\_v,Date TS\_5);

/\*\*

\* Client 从Server V返回的报文中解析出相互认证报文

\* 把Message解析到Stirng[], String数组元素为：K\_c\_v || TS\_5

\*/

public String[] client\_parse\_v(String message);

**生成票据和认证消息**

/\*\*

\* 生成票据许可票据Ticket\_tgs

\*/

public String get\_Ticket\_tgs(String K\_tgs,String K\_c\_tgs,String ID\_c,String AD\_c,String ID\_tgs,String TS\_2,Date Lifetime\_2);

/\*\*

\* 生成服务许可票据Ticket\_v

\*/

public String get\_Ticket\_v(String K\_v,String K\_c\_v,String ID\_c,String AD\_c,String ID\_v,String TS\_4,Date Lifetime\_4);

/\*\*

\* Client经过AS生成认证消息AuthenticatorC

\*/

public String get\_Authenticator\_c(String K\_c\_tgs,String ID\_c,String AD\_c,Date TS\_3);

/\*\*

\* Client经过TGS生成认证消息AuthenticatorC 这个函数不用写，直接调用上面函数参数传入TS\_5即可

\*/

~~public String get\_Authenticator\_c(String K\_c\_v,String ID\_c,String AD\_c,Date TS\_5);~~

**生成证书&判断证书**

/\*\*

\* 生成证书Certification字符串

\*/

public String get\_Certification();

/\*\*

\* 判定证书Certification可靠性

\*/

public boolean verify\_Certification();

## §7.4 DB类

/\*\*

\* 插入数据库

\*/

public void add(String user\_id,String user\_password);

/\*\*

\* 查找数据库

\*/

public bool find(String user\_id,String user\_password);

## §7.5 Socket类

/\*\*

\* 服务器开启服务

\*/

public void server\_start();

/\*\*

\* 客户端开启

\*/

public void client\_start(String message);

/\*\*

\* 发送数据

\*/

public void send\_message(String message);

# 设计和开发规范

## §8.1 数据库设计规范

1. 数据库命名规范

采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成; 命名简洁明确(长度不能超过30个字符); 例如：user, stat, log, 也可以safe\_user, safe\_stat, safe\_log给数据库加个前缀; 除非是备份数据库可以加0-9的自然数：user\_db\_20161000000;

1. 数据库表名命名规范

采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成; 命名简洁明确,多个单词用下划线'\_'分隔; 例如：user\_login, user\_profile, user\_detail, user\_role, user\_role\_relation, user\_role\_right, user\_role\_right\_relation 表前缀'user\_'可以有效的把相同关系的表显示在一起;

1. 数据库表字段名命名规范

采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成; 命名简洁明确,多个单词用下划线'\_'分隔; 例如：user\_login表字段 user\_id, user\_name, pass\_word, eamil, tickit, status, mobile, add\_time; 每个表中必须有自增主键,add\_time(默认系统时间) 表与表之间的相关联字段名称要求尽可能的相同;

1. 数据库表字段类型规范

用尽量少的存储空间来存数一个字段的数据; 例如：能使用int就不要使用varchar、char,能用varchar(16)就不要使用varchar(256); IP地址最好使用int类型; 固定长度的类型最好使用char,例如：邮编; 能使用tinyint就不要使用smallint,int; 最好给每个字段一个默认值,最好不能为null;

1. 数据库表索引规范

命名简洁明确,例如：user\_login表user\_name字段的索引应为user\_name\_index唯一索引; 为每个表创建一个主键索引; 为每个表创建合理的索引; 建立复合索引请慎重;

## §8.2 小组成员编码规范

这里只做简单阐述，如有变动，会及时通知整个小组成员。

1. 不要给类名加前缀。使用UpperCamelCase风格。
2. 类名少用缩写,不要使用下划线字符 (\_)。
3. 不允许任何魔法值（未经定义的常量）直接出现在代码中。

例如：int []a = new int[4];

这样定义就是错误的，因为这里的4到底代表什么？直接看上去大大降低了代码的可读性。正确的定义应该如下：

static final int USER\_TABLE\_MAX = 16;

int []a = new int[PRICE\_TABLE\_MAX];

1. 如果大括号为空，则简洁地写成{}即可，不需要换行；如果是非空代码块则：
2. 左大括号前不换行
3. 做大括号后换行
4. 右大括号前换行
5. 右大括号后还有else等代码则不换行表示终止的右大括号后必须换行

其它注意事项：

1. 除使用工具直接生成的类外，其它所有类的功能要有注释，即使只是一个数据对象；非常见功能的API必须要有功能描述的注释。独立的要被应用其他功能所调用的模块的接口和公共API的注释要完备，即包括功能说明、参数和返回值说明。其它情况的API的注释尽量完备，但不强制要求。
2. 一次性的流打开后必须关闭和释放。一般地，在流打开后，都有一个try catch语句，务必加上finally块释放流资源。
3. 即使IF，While是单语句，也要使用“{ }”来划分程序块。
4. 一些常识性代码，不需要写没必要的注释。

# 附：小组分工

表 8-1

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **人员** | **任务一** | **任务2** |
| 1 | 魏子杰 | Kerberos认证过程socket线程编写 | C-->AS与AS-->C，请求票据访问票据过程收发数据包函数。 |
| 2 | 李迎光 | UI界面设计 | C-->TGS与TGS-->C，请求服务访问票据过程收发数据包函数。 |
| 3 | 胡国煜 | DES & RSA加密接口编写 | C-->V与V-->C，服务认证过程收发数据包函数 |
| 4 | 王琛 | 数据库访问设计与数据库设计 | 证书认证过程的收发数据包的函数 |