**设计报告目录**

[一、课程设计目的 4](#_bookmark0)

[二、需求分析 4](#_bookmark1)

* 1. [背景和目标 4](#_bookmark2)
  2. [功能需求 5](#_bookmark3)
  3. [性能需求 5](#_bookmark4)
  4. [界面需求 6](#_bookmark5)
  5. [其它约束条件 6](#_bookmark6)

[三、详细设计 6](#_bookmark7)

* 1. [系统结构设计 6](#_bookmark8)
  2. [模块设计 7](#_bookmark9)
  3. [重要数据结构设计 10](#_bookmark10)
  4. [程序函数清单 13](#_bookmark11)

[四、系统设计难点与亮点： 24](#_bookmark12)

[五、设计成果： 28](#_bookmark13)

[六、设计心得： 33](#_bookmark14)

**设计报告**

# 一、课程设计目的

大型程序设计实践课程设计的目的是通过应用所学的理论知识和技术技能来完成一个较大规模的软件项目，以实践和巩固在软件开发过程中的实际操作能力。课程要求小组合作，在指导教师的指导下，完成一个具有一定复杂度和规模的软件项目。以下是大型程序设计实践课程设计的主要目的：

学习实践软件开发流程：在整个软件开发周期中经历需求分析、系统设计、编码实现、测试调试等各个阶段，并且理解每个阶段的重要性和相互之间的关系。

提高团队协作能力：课程需要与小组成员合作，共同完成项目，这涉及到任务分配、沟通协调、合作解决问题等方面的能力培养。

掌握学习软件工程原则：要求学习并应用软件工程的基本原则和最佳实践，如模块化设计、代码规范、版本控制、软件测试等，以确保项目的质量和可维护性。

学会解决实际问题：通过选择一个实际的问题或需求，并进行软件开发来解决它，可以锻炼问题解决能力和创新思维。

综合运用知识和技能：大型程序设计实践要求综合运用所学的多个学科领域的知识和技能，如编程语言、数据库、网络编程等，从而加深对这些知识的理解和应用能力。

总之，大型程序设计实践课程设计旨在培养软件开发实践能力，在真实项目中进行有效的团队合作，并将所学的理论知识和技术应用于实际的软件开发。

# 二、需求分析

## 背景和目标

随着互联网的普及和发展，人们对于在线沟通和交流的需求越来越高，一个安全可靠的即时通讯系统应运而生。即时通讯系统，简称IM，是一种能够实现实时文本、语音、视频、文件等多媒体信息交换的网络应用。IM系统可以提高人们的沟通效率和质量，增强社交关系和团队协作，满足个人和企业的多样化需求。

本项目旨在开发一个功能完善高效的安全的即时通讯系统，为用户提供一个高效、便捷、安全的在线沟通和协作平台。

## 功能需求

### 客户端功能：

1. **用户注册功能：**用户在注册界面输入用户名、邮箱以及密码，与服务器通信添加用户完成注册。
2. **用户登录功能：**用户输入用户名和密码，与服务器完成认证登录
3. **通讯录管理功能**：包含添加好友以及删除好友，与服务器通信维护个人通讯录
4. **即时通讯功能：**实现与在线好友的P2P通信功能
5. **端到端透明加解密功能**：在P2P通信过程中，采用非对称密码体制制协商加密秘钥，用对称密码算法进行加解密，实现端到端的透明加解密
6. **信息隐藏功能：**实现以图片为载体，透明嵌入提取文本信息的功能
7. **语音聊天的功能：**实现能够发送语音消息的功能，并进行透明加解密
8. **呈现个人信息的功能：**客户端登录成功之后，本地存储个人信息并进行呈现
9. **人机交互功能：**实现美观简洁的GUI界面，方便用户使用和操作

### 服务器功能：

1. **用户注册功能：**与用户通信，在数据库中添加用户信息，完成注册
2. **用户验证功能：**验证用户名和密码，检验输入的信息是否正确，若正确，则登录成功
3. **提供公钥功能：**用户在登录时，将自己的公钥发给服务器储存。当一个用户在发送文件时，请求该用户的公钥并返回
4. **维护在线状态功能：**用户登录成功后，记录保存用户的在线状态、IP地址、端口号
5. **用户在线列表发放：**负责向用户发送其好友的在线列表，包括好友的用户名、IP地址、端口号
6. **实现管理员管理功能：**实现一个管理员管理的页面，对用户信息进行添加与修改。

## 性能需求

本系统的主要性能需求如下：

1. **响应时间**：系统在正常网络条件下，消息发送和接收的响应时间不超过1秒
2. **吞吐量：**系统应保证在高峰时段，能够同时支持多位在线用户，并能够处理较大交互量的消息
3. **可用性**：系统应保证在正常运行条件下，能够实现99.99%的可用性，将出现故障的可能性尽可能降低
4. **可扩展性：**系统应保证在用户量增长或功能变化的情况下，能够灵活地进行水平或垂直扩展，满足不同的需求

## 界面需求

本系统主要界面需求如下：

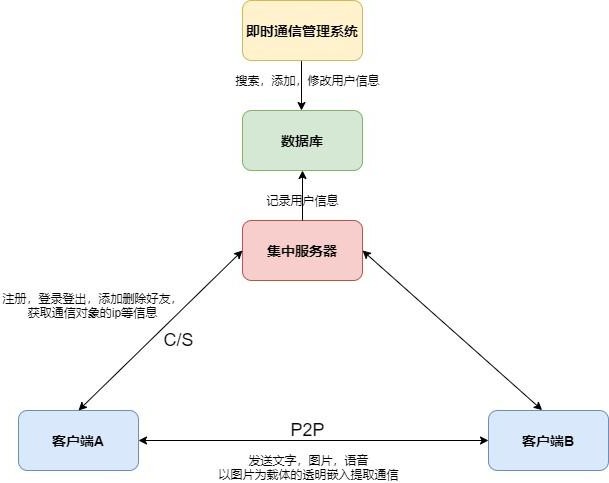
1. **界面风格：**系统采用简洁、清晰、美观的界面风格，符合大多数用户的审美习惯和操作习惯
2. **界面布局：**系统应合理地安排界面元素的位置和大小，做到合理、简洁、美观，避免过于拥挤或者空旷
3. **界面交互：**系统应提供明确的提示信息和反馈信息，支持多种交互方式，提高用户的操作便利性和使用满意度

## 其它约束条件

1. 项目在开发过程中应统一使用python语言，在pycharm平台上进行开发，并且使用共享文档统一接口参量属性以及命名
2. 项目开发应该充分考虑用户对象以及使用场景
3. 项目开发过程中应该尊重知识产权，遵守相关的法律法规

# 三、详细设计

## 系统结构设计



**图1：系统结构图**

利用C/S和P2P的混合架构实现即时通讯， 采用对称密码、非对称密码、信息隐藏等实现消息的

秘密传输。

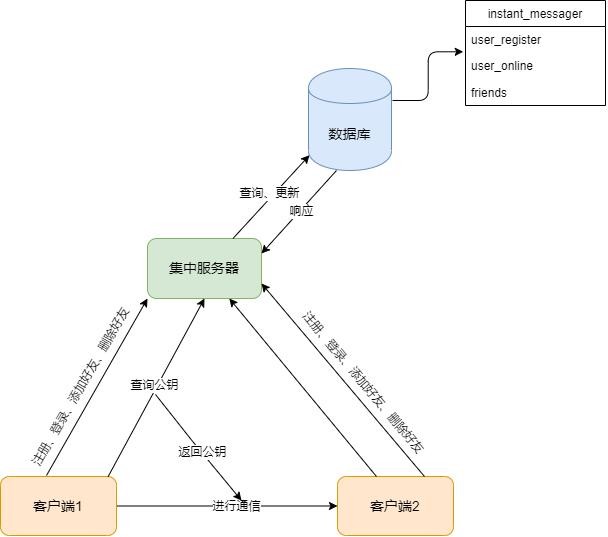
**集中服务器：**用户注册、登录登出、通讯录管理、呈现个人信息

**客户端：**美观的可视化 GUI 界面，即时通信，端到端透明加解密以传输文字、图片，语音、以图片为载体的透明嵌入提取。

**即时通信管理系统：**管理员可以添加用户，更新用户信息，查找用户。

## 模块设计

### 集中服务器：



**图2：集中服务器结构图**

**用户注册：**与用户进行通信，完成用户注册，并进行必要的验证。

**用户验证：**对用户提供的用户名和密码进行验证，确保用户身份的合法性和准确性。

**用户公钥提供：**当用户需要发送文件时，服务器与用户进行通信，并向用户提供其好友的公钥。这样用户可以使用好友的公钥进行文件加密，确保文件传输的安全性。

**用户在线状态维护：**当用户在线时，服务器负责记录和保存用户的在线状态、IP 地址和端口号等信息。这样其他用户可以查询到好友的在线状态，并与其进行通信或传输文件。

**用户在线列表发放：**服务器负责向用户发送其好友的在线列表，包括好友的用户名、IP 地址和端口号等信息。这样用户可以获取好友的在线状态，并根据需要进行通信或文件传输。通过以上功能，集中服务器能够向用户提供必要的信息和服务来满足用户的需求，从而实现用户之间的通信和

文件传输的安全与便利。

**即时通信管理系统：**实现一个管理员管理的页面，对用户信息进行添加与修改。

### 客户端：



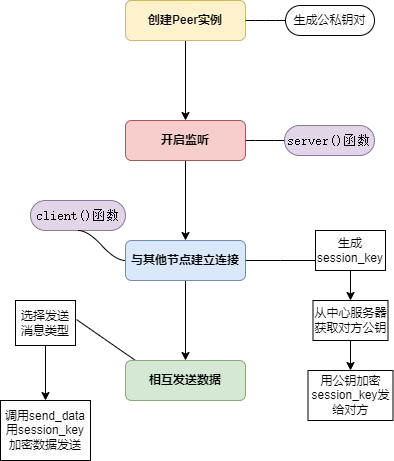
**图3：客户端结构图**

**可视化前端：**通过创建登录注册页面、主页面、个人信息页面、聊天页面、加好友页面、图片加密页面等实现人机交互。

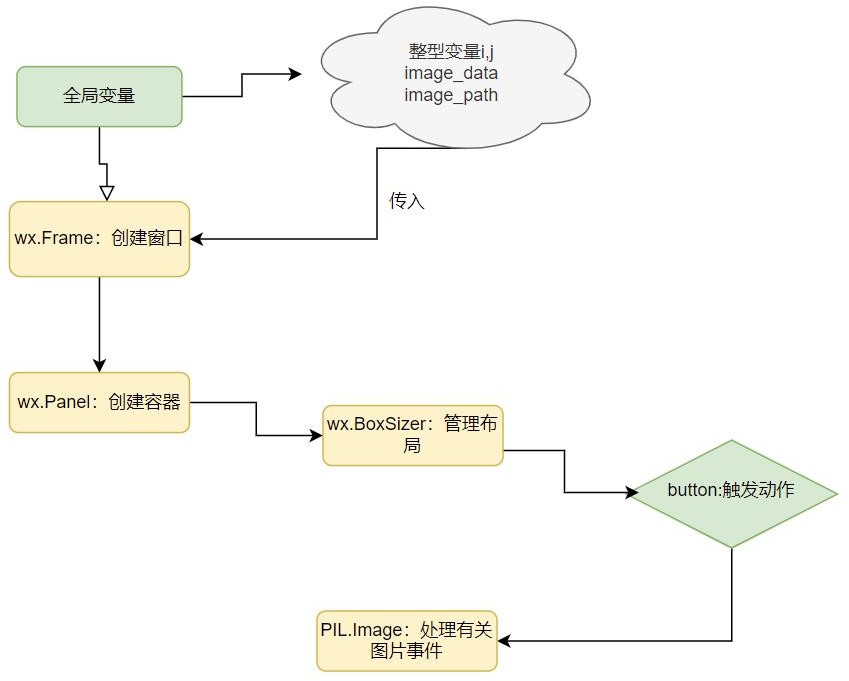
**语音聊天透明加解密：**实现音频输入和输出、发送和接收音频数据等功能；让用户通过麦克风录制音频，并保存为文件，然后将其发送给接收方。接收方接收其他用户发送过来的音频之后，将其保存为文件，并播放出来。

**实现 P2P 连接：**不存在客户端与服务器这样的严格区分，同时每个节点又充当着客户端和服务器。各个节点之间是平等的，只要接入网络，任意节点都能够将消息通知给网络中的每个节点。

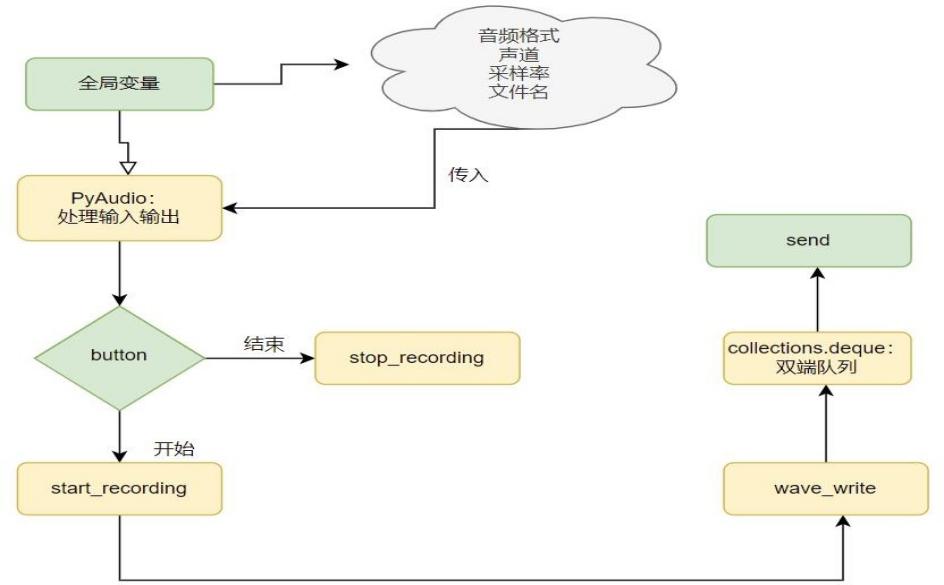
**以图片为载体的透明嵌入提取功能：**图片上传并输入需要隐藏的文本消息，然后将文本消息转换为二进制，并且使用 LSB 算法对图片进行处理，信息隐藏在图片中发送给接收方。接收方收到后，首先将图片保存在本地，然后根据 LSB 算法原理，将秘密信息从图片中提取出来，转换为文本形式显示出来。



**图4：P2P流程图**



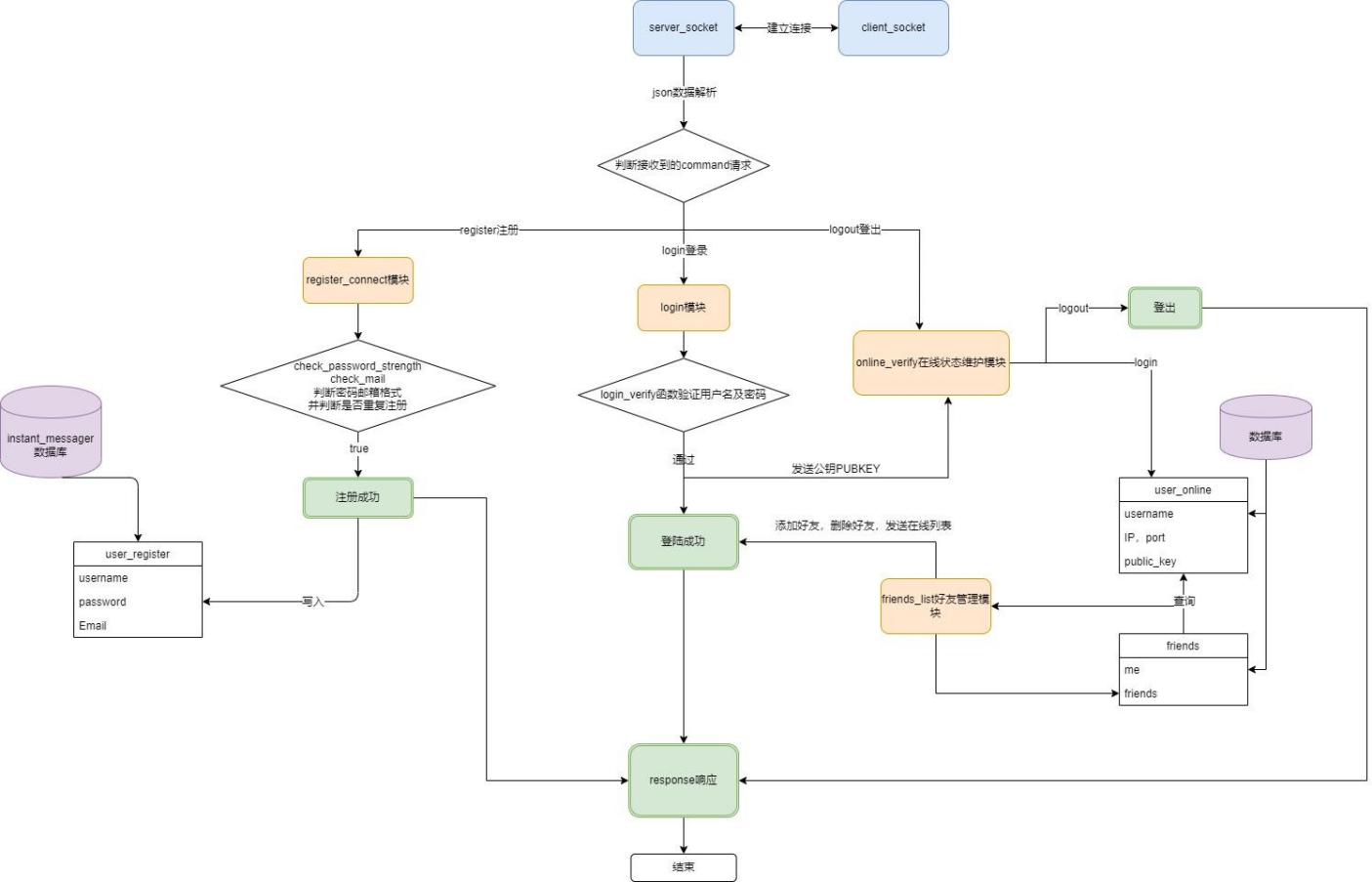
**图5：信息隐藏流程图**



**图6：音频透明传输流程图**

## 重要数据结构设计

### 集中服务器：



**图7：集中服务器流程图**

0

**User\_Register 表结构：**用于存储用户注册信息的数据库表，包含以下字段：

* username: 用户名，作为主键
* password: 密码
* email: 邮箱

**User\_Online 表结构：**用于存储用户在线状态信息的数据库表，包含以下字段：

* username: 用户名，作为主键
* ip: 用户的 IP 地址
* port: 用户的端口号
* online\_status: 在线状态（例如，1 表示在线，0 表示离线）
* public\_key: 用户的公钥

**Socket 对象：**用于与客户端建立连接的套接字对象，包含以下信息：

* host: 主机地址
* port: 监听的端口号

**friends 表结构：**用于记录好友关系列表，包含以下信息：

* me: 个人用户名
* friend: 好友用户名

### 客户端：

##### P2P 通信

class Peer:

peers = [] # 存储已知的节点地址 port = 5001 # 设置默认端口 key\_pairs = {} # 存储自己的公私钥对

session\_keys = {} #存储所有连接的会话密钥

**# 初始化**

def init (self, logger,username): self.username = username #获取自己用户名 self.log = logger

self.host = socket.gethostbyname(socket.gethostname()) #获取自己 IP

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) self.sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1) self.conn\_pool = {} # 连接池

self.key\_pairs = KeyCreate.generate\_key\_pairs() #生成公私钥

self.peersock = {} #存储和对方节点的 socket

**# 服务器端**

def server(self): #开启服务器监听，等待其他节点连接

def message\_handle(self, conn, addr): #接收 socket 中的接收数据

def process\_data(self, conn, buffer, addr): #根据发送不同类型的数据来尽情处理 def server\_close(self): #关闭服务器监听

def remove\_client(self, client\_ID): #从连接池中将socket 删除

**# 客户端**

def client(self, host, port, username): #发起 socket 连接，交换公钥协商会话密钥

def send\_data(self, conn, cmd, \*\*kv): #根据不同请求发送数据 def socket\_close(self, conn): #关闭连接

##### CS 通信

class Client\_Socket:

**# 初始化**

def init (self, logger,username): self.flag = 0 #设置接收标志位 self.PUBKEY\_get = 0 #获取对方节点公钥 self.username = username

self.log = logger

self.host = socket.gethostbyname(socket.gethostname()) self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) self.sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

**# 客户端 socket 方法**

def server\_connect(self,host,port): #与中心服务器发起连接 def send\_request(self, cmd, msg): #根据不同命令封装待 def receive(self): #接收中心服务器反馈的数据

def publickey\_get(self, username): #获取对方节点公钥

##### 信息隐藏

**# ImageHide 算法**

def select\_image(event): #实现选择本地图片文件的功能

def toasc(strr): #实现将字符串转换为 ASC||码的列表，并返回该列表 def bin\_to\_text(bin\_str): #将二进制转换为文本形式

def enc\_file(event): #用于将用户输入的文本写入到一个文件中 def plus(str): #将二进制字符串补齐为 8 位，方便 LSB 算法的处理

def get\_key(strr): #逐个字节将要隐藏的文件内容转换为二进制，并拼接起来 def mod(x, y): #定义一个模运算的功能，返回两个数相除的余数

def LSB(event):#实现 LSB 算法，进行信息隐藏，并保存修改后的图片

def un\_LSB(event):#实现从图片中提取隐藏的信息的功能，并输出消息内容 def send\_ready(event,peer:Peer):#实现图片发送的功能

def hp\_process(peer:Peer):#实现创建 GUI 面板的功能，绑定事件和函数

## 程序函数清单

**集中服务器 ：**代码在Minichat\_Server文件夹中

##### server\_socket.py

* + - ` init (self, logger)`：构造方法，初始化服务器套接字对象和日志记录器
    - `start(self)`：启动服务器，监听客户端连接。
    - `handle\_connection(self, client\_socket)`：处理客户端连接的方法。
    - `process\_request(self, data)`：解析接收到的 JSON 数据。
    - `response\_command(self, data, timer\_running=True)`：根据客户端请求的命令进行响应。

##### 算法描述：

①如果`command`为 `'login'`：

* + - * 提取`data`中的用户名、密码、公钥、IP 和端口信息。
      * 调用`login\_verify(username, password)`函数进行用户名和密码的验证。
      * 如果验证失败，返回响应数据`{"response": "Verify Failed!"}`。
      * 如果验证成功，调用`online\_no\_repeat(username)`函数检查用户是否已经在线。
      * 如果用户已经在线，返回响应数据`{"response": "Login Repeatedly!"}`。
      * 如果用户不在线，调用`online\_check(username, password, ip, port, public\_key)`函数将用户信息插入数据库，并定时发送在线好友列表和个人信息（用户名、邮箱）。
      * 返 回 响 应 数 据 `{"response": "Verify Successfully and Got Public Key!", "about\_me":

me\_json}`。

②如果`command`为 `'add user'`：

* + - * 提取`data`中的自己的用户名和要添加的好友的用户名。
      * 调用`add\_friends(me\_username, friend\_username)`函数尝试添加好友。
      * 如果成功添加好友，返回响应数据`{"response": "Add User Successfully!"}`。
      * 如果已经是好友关系，返回响应数据`{"response": "You have been friends!"}`。

③如果`command`为 `'delete user'`：

* + - * 提取`data`中的自己的用户名和要删除的好友的用户名。
      * 调用`del\_friends(me\_username, friend\_username)`函数尝试删除好友。
      * 如果成功删除好友，返回响应数据`{"response": "Delete User Successfully!"}`。
      * 如果没有好友关系，返回响应数据`{"response": "Don't delete the same person!"}`。

④ 如果`command`为 `'logout'`：

* + - * 提取`data`中的用户名。
      * 调用`logout(username)`函数将用户从在线列表中移除。
      * 返回响应数据`{"response": "Logout Successfully!"}`。

⑤如果`command`为 `'PUBKEY\_GET'`：

* + - * 提取`data`中的用户名。
      * 调用`get\_target\_user\_public\_key(username)`函数获取目标用户的公钥。
      * 返回响应数据`{"response": "PUBKEY", "PUBKEY": publickey}`。

⑥如果`command`为 `'select friends list'`：

* + - * 提取`data`中的用户名。
      * 调用`online\_no\_repeat(username)`函数检查用户是否在线。
      * 如果用户不在线，进入定时循环，并调用`send\_online\_friends\_list(username)`函数定时发送在线好友列表。
      * 在定时循环中，每隔一段时间调用`send\_online\_friends\_list(username)`函数获取在线好友

列表，并返回响应数据`{"response": "friends list", "online\_friends\_list": friends\_json}`。

##### friends\_list.py

-`create\_friends()`: 创建好友表。

* `add\_friends(me, friend)`: 添加好友关系。

-`del\_friends(me, friend)`: 删除好友关系。

-`about\_me(username)`: 读取个人信息。

* `query\_friends(me)`: 查询好友列表并返回在线好友列表。

-`send\_online\_friends\_list(username)`: 发送在线好友列表给指定用户。

##### 算法描述：

-建立数据库连接：通过 `pymysql.connect()` 函数与数据库服务器建立连接。

-创建游标对象：通过连接对象的 `cursor()` 方法创建游标对象，用于执行 SQL 查询和操作数据库。

-创建好友表：通过执行 `CREATE TABLE IF NOT EXISTS` 语句创建名为 `friends` 的表，用于存储好友关系。

-添加好友：实现 `add\_friends(me, friend)` 函数，用于向好友表中插入好友关系。首先，查询好友表检查是否已存在相同的好友关系。如果存在，则返回 `False`；否则，执行插入操作，将好友关系添加到表中，并返回 `True`。

-删除好友：实现 `del\_friends(me, friend)` 函数，用于从好友表中删除好友关系。首先，查询好友表检查是否存在该好友关系。如果不存在，则返回 `False`；否则，执行删除操作，将好友关系从表中删除，并返回 `True`。

-读取个人信息：实现 `about\_me(username)` 函数，用于从用户注册表中查询个人信息。通过执行 SQL 查询语句获取用户名和邮箱，并将查询结果转换为字典格式。最后，将字典转换为 JSON字符串并返回。

-查询好友：实现 `query\_friends(me)` 函数，用于查询指定用户的好友列表。首先，初始化好友表。然后，执行 SQL 查询语句获取指定用户的好友列表。对于每个好友，查询其在 线 信息，并将在线信息组织成一个列表。最后，关闭游标和数据库连接，并返回在线好友列表。

-发送在线好友列表：实现 `send\_online\_friends\_list(username)` 函数，用于发送指定用户的在线好友列表。在一个循环中，调用 `query\_friends(username)` 获取在线好友列表，并将列表转换为

JSON 字符串。然后，通过 `return` 返回在线好友列表的 JSON 字符串。

-关闭游标和数据库连接：在合适的位置，通过调用 `cursor.close()` 和 `cnx.close()` 方法关闭游标和数据库连接，释放资源。

##### online\_verify.py

* `online\_check(username, password, ip, port, public\_key)`: 在线状态维护，检查用户登录验证通过后，将用户信息插入到 `user\_online` 表中，表示用户在线状态。

-`online\_no\_repeat(username)`: 检查用户是否已经在线，如果用户已经在线，则无法重复登录。

* `logout(username)`: 用户登出操作，将用户从 `user\_online` 表中标记为下线状态。

##### 算法描述：

-数据库连接：通过 `pymysql.connect()` 函数与数据库服务器建立连接，并创建游标对象。

-创建表：通过执行 `CREATE TABLE IF NOT EXISTS` 语句创建名为 `user\_online` 的表，用于存储在线用户的信息。表中包括用户名、IP 地址、端口号、在线状态、公钥等字段。

-在线状态维护：实现 `online\_check(username, password, ip, port, public\_key)` 函数，用于维护用户的在线状态。首先，通过调用 `login.login\_verify(username, password)` 函数验证用户的登录信息。然后，检查用户是否重复登录，如果重复登录则返回 False。接着，执行插入操作将用户的在线信息插入到 `user\_online` 表中。如果插入成功，则返回 True；否则，返回 False。

-检查用户是否已经在线：实现 `online\_no\_repeat(username)` 函数，用于检查指定用户是否已经在线。通过执行 SQL 查询语句检索 `user\_online` 表，如果存在该用户的记录，则表示用户已经在线，返回 False；否则，返回 True。

-登出：实现 `logout(username)` 函数，用于用户的登出操作。通过执行 SQL 删除语句将用户从 `user\_online` 表中删除。如果删除成功，则返回 True；否则，返回 False。

##### login.py:

* `login\_verify(username, password)`: 登录验证函数，用于验证用户名和密码是否匹配。该函数连接数据库，执行查询语句，根据查询结果判断登录是否成功。

##### 算法描述：

-首先，建立与数据库的连接。通过 pymysql.connect() 函数来连接数据库服务器。需要提

供数据库的主机名（host）、用户名（user）、密码（password）、数据库名（database）等信息。

-创建游标对象。通过连接对象的 cursor() 方法创建游标对象，用于执行 SQL 查询和操作数据库。

-定义 SQL 查询语句。在这个例子中，SQL 查询语句是 select \* from user\_register where username = %s，其中 %s 是一个占位符，将在后续的执行中用实际的用户名进行替换。

-执行 SQL 查询。使用游标对象的 execute() 方法执行 SQL 查询语句，并传入用户名作为参数。执行后，可以通过游标对象的方法获取查询结果。

-判断验证结果。通过 fetchone() 方法获取查询结果中的第一行数据，如果结果为 None，说明用户名不存在；如果查询到的密码与用户提供的密码不一致，说明密码不正确；如果用户名和密码都匹配，说明验证成功。

-关闭游标和数据库连接。在完成查询操作后，需要显式地关闭游标和数据库连接，释 放资源。

-根据验证结果返回相应的布尔值。如果验证成功，返回 True；否则，返回 False。

##### public\_key.py:

-`get\_target\_user\_public\_key(username)`: 连接数据库，获取指定用户名的用户的公钥。该函数连接数据库，执行查询语句，返回查询结果中的公钥。

##### 算法流程：

-连接数据库：通过 `pymysql.connect()` 函数与数据库服务器建立连接，并创建游标对象。

-获取目标用户的公钥：实现 `get\_target\_user\_public\_key(username)` 函数，用于获取目标用户的公钥。通过执行 SQL 查询语句检索 `user\_online` 表，获取指定用户名的公钥。如果查询结果非空，则返回公钥；否则，返回 None。

-关闭游标和数据库连接：在函数中关闭游标对象和数据库连接，释放资源。

##### register.py:

-`check\_password\_strength(password)`: 密码格式检验函数，检查密码是否符合要求。该函数接受一个密码作为输入，验证密码是否满足以下要求：长度至少为 8 个字符，包含大写字母、小写字母、数字和特殊字符（@#$%^&+=）。如果密码符合要求，返回 True；否则返回 False。

-`check\_mail(email)`: 邮箱格式检验函数，检查邮箱地址是否符合要求。该函数接受一个邮箱地址作

为输入，使用正则表达式验证邮箱地址的格式是否正确。如果邮箱格式正确，返回 True；否则返回 False。

* `write\_to\_table(username, password, email)`: 将数据写入数据库函数，将注册信息写入数据库。该函数接受用户名、密码和邮箱作为输入，首先验证密码和邮箱的格式是否正确，然后连接数据库并执行相应的操作。如果用户名已存在，则返回 "Repeated User!"；如果插入操作成功，则返回 "Register Successfully!"；如果插入操作失败，则返回 "Register Failed!"。

##### 算法流程：

-密码格式检验：使用正则表达式和条件判断，依次检查密码的长度、大写字母、小写字母、数字和特殊字符的要求。

-邮箱格式检验：使用正则表达式匹配邮箱的格式，判断是否符合要求。

-将数据写入数据库：建立数据库连接，创建`User\_Register`表（如果不存在），进行密码和邮箱格式验证，检查用户名是否已存在，执行插入操作将用户名、密码和邮箱写入数据库，提交事务并获取插入结果。

##### register.connect.py:

-`connect()`: 监听客户端函数，创建一个 Socket 对象并绑定主机地址和端口号，然后开始监听端口。循环监听端口，接收客户端的连接并接收发送过来的数据。根据接收到的数据内容，提取出命令和相关参数， 并根据命令调用相应的函数进行处理。在注册命令（'register'） 的情况下， 调用

`register.write\_to\_table()` 函数将数据写入数据库，并将处理结果发送回客户端。

##### 算法流程：

-创建一个 Socket 对象并绑定主机地址和端口号。

-开始监听端口，等待客户端的连接。

-循环监听端口：

1. 接收客户端的连接。
2. 接收客户端发送的数据。
3. 根据接收到的数据进行处理：
   * 如果是注册命令，提取用户名、密码和邮箱信息。
   * 调用`register.write\_to\_table`函数将用户信息写入数据库，并获取注册结果。
   * 将注册结果发送给客户端。

-关闭连接。

#### **客户端：**代码在Minichat\_Client文件夹中

##### P2P.py

def p2p\_process(username):

log\_fmt = '%(asctime)s - %(levelname)s - %(message)s' logging.basicConfig(level=logging.DEBUG, format=log\_fmt)

peer = Peer(logger=logging.getLogger(), username=username) #实例化节点 # 启动服务器线程

server\_thread = Thread(target=peer.server) server\_thread.setDaemon(True) server\_thread.start()

return peer

**实现功能：**与GUI登录间绑定，当登录成功后，节点自启动**参数说明：**username(本节点用户名）

##### KeyCreate.py

def generate\_key\_pairs():

return {"public\_key": public\_pem, "private\_key": private\_pem}

**实现功能：**随机生成节点公私钥对，保存在字典中 (RSA算法)**参数说明：**public\_key(公钥）、private\_key(私钥）

def encrypt\_session\_key(public\_pem, session\_key):

return c\_session\_key

**实现功能：**使用对方公钥加密随机生成的会话密钥

**参数说明：**public\_pem(对方公钥）、session\_key(会话密钥）**算法流程：**

def decrypt\_session\_key(private\_pem, c\_session\_key):

return d\_session\_key

**实现功能：**用自己私钥解密对方发过来的密文获得会话密钥

**参数说明：**private\_key(私钥）、c\_session\_key(加密后的会话密钥)

def generate\_session\_key():

return session\_key

**实现功能：**随机生成会话密钥

def encrypt\_msg(session\_key,msg):

return c\_msg\_str

**实现功能：**用会话密钥加密明文数据

**参数说明：**session\_key(会话密钥）、msg(明文消息)

def decryrt\_msg(session\_key,c\_msg\_str):

return d\_msg

**实现功能：**用会话密钥解密密文消息

**参数说明：**session\_key(会话密钥）、c\_msg\_str(密文消息)

##### VoiceChat.py

def audio\_input(peer:Peer): root = tk.Tk()# 创建 GUI 界面

def start\_recording(): # 开始录音函数 def stop\_recording(): # 结束录音函数 def on\_closing(): # 关闭窗口函数

# 添加开始录音按钮

start\_button = tk.Button(root, text="开始录音", command=start\_recording) start\_button.pack()

# 添加停止录音按钮

stop\_button = tk.Button(root, text="停止录音", command=stop\_recording) stop\_button.pack()

# 关联窗口关闭事件与退出主循环的函数

root.protocol("WM\_DELETE\_WINDOW", on\_closing)

root.mainloop() # 运行 GUI 主循环

# 关闭音频输入流 stream.stop\_stream() stream.close() audio.terminate()

# 存入音频到本地

with open(WAVE\_OUTPUT\_FILENAME, 'rb') as file: audio\_data = file.read()

send\_audio(peer,audio\_data) return True

**实现功能：**调用该函数开始录音并调用send\_audio函数将音频发出去**参数说明：**peer(Peer类实例）

def send\_audio(peer:Peer,audio\_data):

...

peer.send\_data(conn,'SEND\_AUDIO',data=audio\_data) return True

**实现功能：**通过调用Peer的send\_data方法将音频数据加密发送至对方节点**参数说明：**peer(Peer类实例）、audio\_data(音频数据)

def audio\_output(audio\_data): return output\_filepath

**实现功能：**接收到音频数据后输出**参数说明：**audio\_data(音频数据))

**信息隐藏算法**

**ImageSent 文件**

def select\_image(event): return

#实现选择本地图片文件的功能

参数： image\_path：保存选择图片文件的路径

file\_dialog：创建文件对话框对象 wx.Image：创建图片对象

static\_bitmap：将图片转换为位图对象并显示

def send\_ready(event,peer:Peer): return image\_data

#实现图片发送的功能

参数： open：打开 image\_path 指定的图片文件，并读取二进制数据

image\_data：保存二进制数据

def pp\_process(peer:Peer): return

#实现创建 GUI 面板的功能，绑定事件和函数参数：peer：传入 peer 参数，进行通信

Conn：与通信主机连接的 socket 对象

Host：通信主机的地址 Button：将事件和函数进行绑定

**算法描述：**

导入 wx, PIL, io, P2P 等模块，创建 GUI 页面，将按钮与事件绑定，用户点击发送图片按钮之后，调用 pp\_Process 函数，弹出 GUI 页面，点击选择图片，用户选择本地图片文件之后，点击发送，即可以将图片发送给通信对方，并将其保存在本地。

**ImageHide 文件**

def select\_image(event): return

#实现选择本地图片文件的功能

参数： image\_path：保存选择图片文件的路径 file\_dialog：创建文件对话框对象 wx.Image：创建图片对象

static\_bitmap：将图片转换为位图对象并显示

def toasc(strr): return int(strr, 2)

#实现将字符串转换为 ASC||码的列表，并返回该列表参数：str：传入的字符串

def bin\_to\_text(bin\_str): return bin\_text

#将二进制转换为文本形式

参数：bin\_str：传入的二进制形式 bin\_int：保存十进制形式 bin\_bytes：保存字节形式 bin\_text：保存文本形式

def enc\_file(event):

return

#用于将用户输入的文本写入到一个文件中

参数：content：保存文本框输入的文本

Open：打开文本文件

Write：将文本内容写入到指定文件中

def plus(str):

return str.zfill(8)

#将二进制字符串补齐为 8 位，方便 LSB 算法的处理

参数：str：传入的参数

def get\_key(strr): return str

# 逐个字节将要隐藏的文件内容转换为二进制，并拼接起来

参数：str：传入的文本内容

Plus：使用 plus 函数进行拼接

def mod(x, y): return x % y

#定义一个模运算的功能，返回两个数相除的余数

def LSB(event):

return

#实现 LSB 算法，进行信息隐藏，并保存修改后的图片参数：image\_data：用来保存图片的二进制数据

image\_path：用来保存图片路径

key：保存需要隐藏的信息

算法实现： # 分别将每个像素点的 RGB 值余 2，去掉最低位的值

# 从需要隐藏的信息中取出一位，转换为整型

# 两值相加，将信息隐藏起来

def un\_LSB(event):

return

#实现从图片中提取隐藏的信息的功能，并输出消息内容

参数：le 文件长度

new ：信息提取出后所存放的文件 tiqu ：提取信息保存的路径 width：图片的宽

height：图片的高

def send\_ready(event,peer:Peer):

return image\_data #实现图片发送的功能

参数： open：打开 image\_path 指定的图片文件，并读取二进制数据

image\_data：保存二进制数据

def hp\_process(peer:Peer): return

#实现创建 GUI 面板的功能，绑定事件和函数参数：peer：传入 peer 参数，进行通信

Conn：与通信主机连接的 socket 对象

Host：通信主机的地址 Button：将事件和函数进行绑定

**算法描述：**

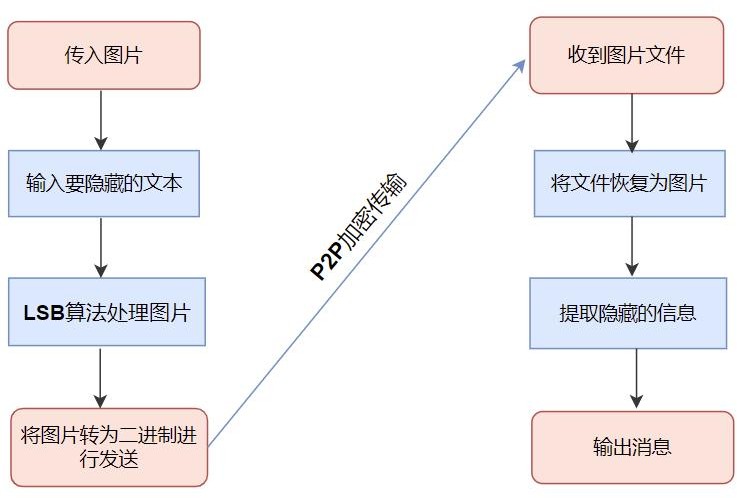
导入 wx, PIL, io, P2P 等模块，创建 GUI 页面，将按钮与事件绑定，用户点击信息隐藏按钮之后，调用 pp\_Process 函数，弹出 GUI 页面。用户在文本框中输入需要隐藏的信息，将输入内容写入指定文本文件；点击选择图片，选择本地图片文件之后，点击信息隐藏，点击发送，即可以将承载消息的图片发送给通信对方，对方收到后将图片保存在本地，点击提取消息按钮进行消息提取。

# 四、系统设计难点与亮点：

**难点一：**实现以图片为载体的透明嵌入提取功能

该程序使用wxPython库的图形用户界面，实现图片的上传、信息的隐藏和提取的功能。它主要包含以下几个部分：首先进行本地图片的上传，输入需要隐藏的文本消息，然后将文本消息转换为

二进制，并且使用LSB算法对图片进行处理，将信息隐藏在图片中发送给接收方。接收方收到后，首先将图片保存在本地，然后根据LSB算法原理，将秘密信息从图片中提取出来，转换为文本形式显示。



**图8：图片为载体的透明嵌入提取功能**

**难点二：**P2P通信程序设计：

因为p2p通信的原理就是没有固定的客户端和服务器之分，也就是说节点可能既充当客户端的职责也当服务器的职责。所以在设计过程中需要仔细设置每一个socket的设置与通信。最后，我们在每组节点之间建立两个socket，即向对方发送数据用一个socket，接收数据用一个socket，这样可以很好解决客户端与服务器端协调问题。

同时在接收图片以及音频数据时，因为转换成二进制字符过长，导致无法接收方一次接收，所以要分次接收，全接收完毕后进行拼接。但因为服务器端无法判断哪些分组是要拼接的，故我们在发送的消息中添加 b"end\_marker" 作为一条消息的结尾标识符，服务器端通过判定是否有该标识符来决定是否先暂时缓存下来等待拼接。消息处理函数如下：

def message\_handle(self, conn, addr):

conn.sendall("connect server successfully!".encode(encoding='utf8')) client\_ID = None

buffer = b"" *# 创建一个空的字节串作为缓冲区*

while True: try:

data = conn.recv(4096) *# 根据需要调整缓冲区大小*

if not data:

break

buffer += data *# 将接收到的数据添加到缓冲区*

*# 判断是否存在结束标记*

if b"end\_marker" in buffer:

*# 提取完整的数据，不包括结束标记*

complete\_data = buffer[:buffer.index(b"end\_marker")]

*# 将完整消息进行处理*

self.process\_data(conn, complete\_data, addr)

*# 清空缓冲区，保留可能存在于之后的数据*

buffer = buffer[buffer.index(b"end\_marker") + len(b"end\_marker"):]

except ConnectionResetError: self.log.error('Connection reset by peer %s', addr) break

except Exception as e: self.log.exception(e) break

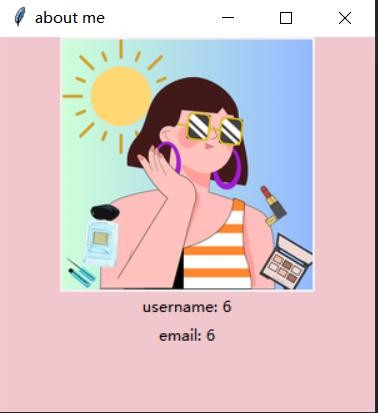
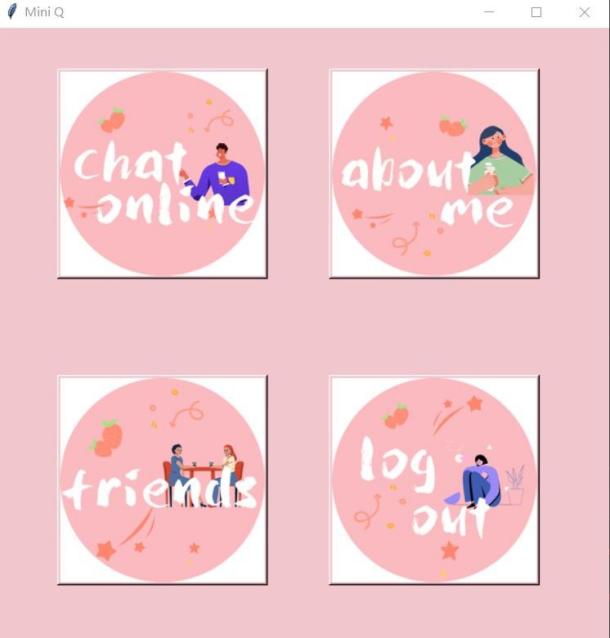
self.remove\_client(client\_ID)

**亮点一：**即时通信管理系统：实现一个管理员管理的页面，对用户信息进行添加与修改。



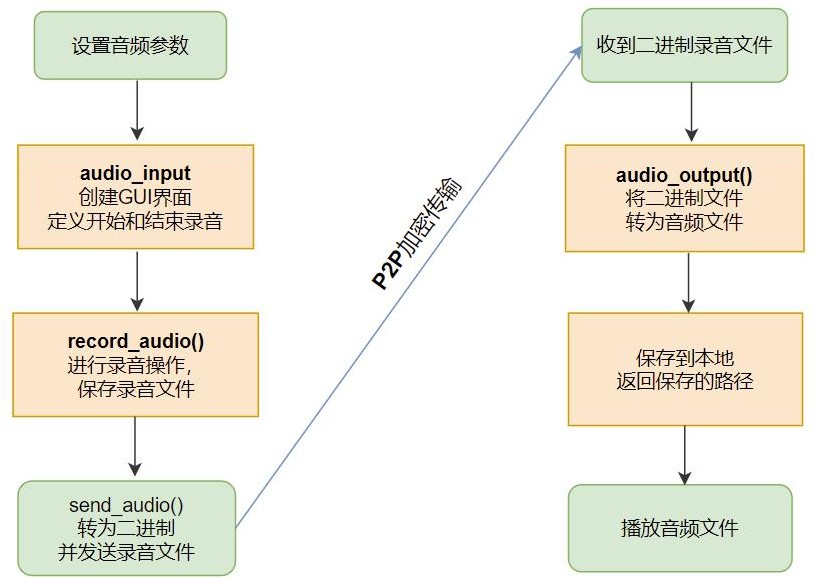
**图9：即时通信管理系统**

**亮点二：**GUI界面美化：在实现人机交互的基础上，美化页面，添加了个人设计。



**图10：GUI界面设计**

**亮点三：**语音聊天透明加解密：该程序使用pyaudio和wave库来处理音频数据，实现音频输入和输出、发送和接收音频数据等功能；用tkinter库来创建GUI界面，实现与其他用户进行通信。首先设置音频参数，让用户通过麦克风录制音频，并保存为文件，然后将其发送给接收方。接收方接收其他用户发送过来的音频之后，将其保存为文件，并播放出来。

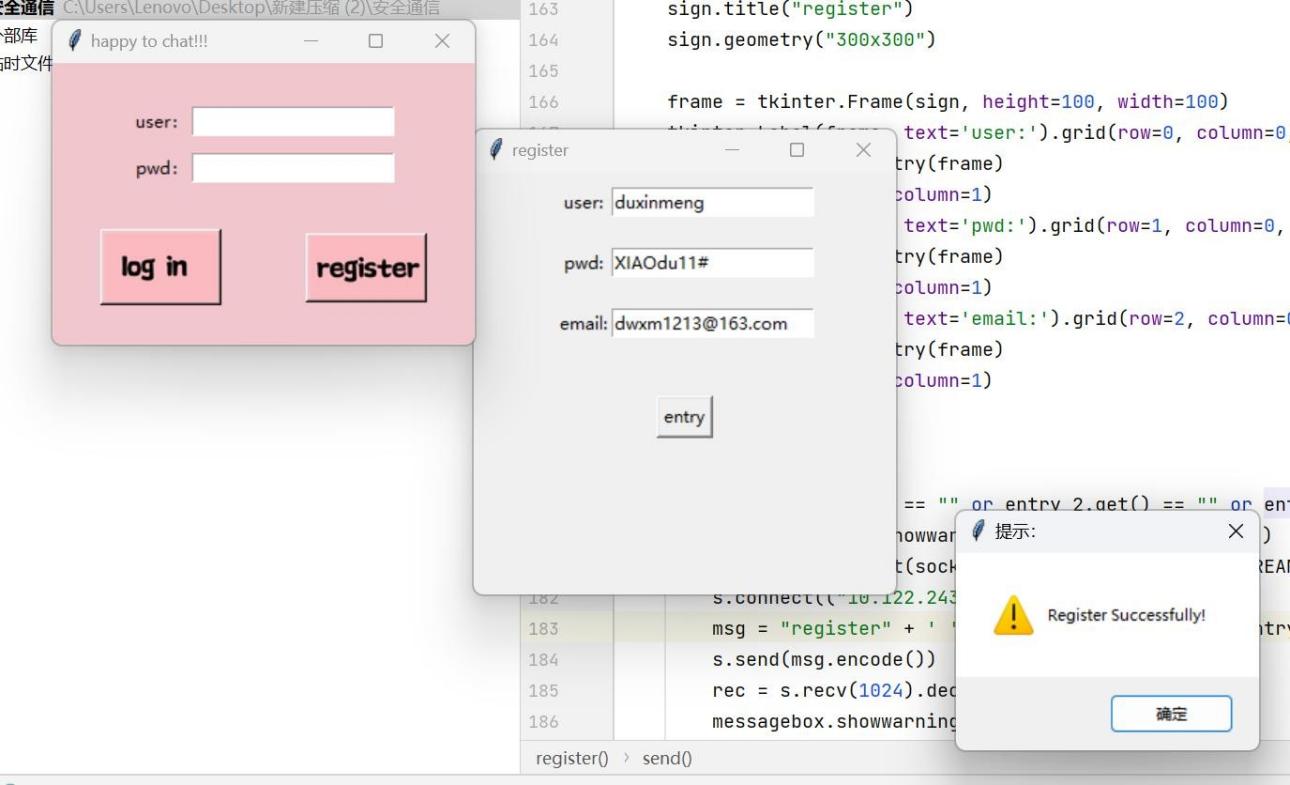


**图9：语音聊天透明加解密**

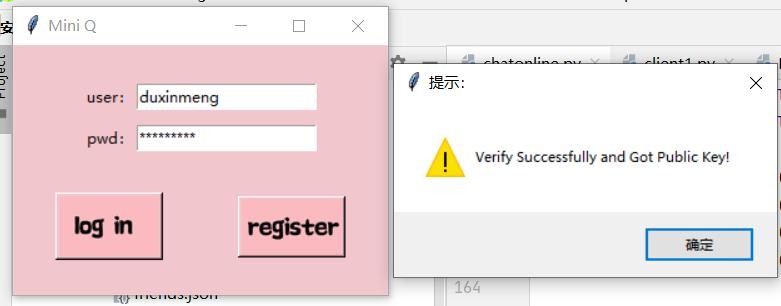
# 五、设计成果：

实现了实验要求的全部基本功能，利用C/S和P2P的混合架构实现即时通讯，采用对称密码、非对称密码、信息隐藏等实现消息的秘密传输。并在实现基本功能的基础上实现了额外功能，如语音聊天、用户管理系统等。

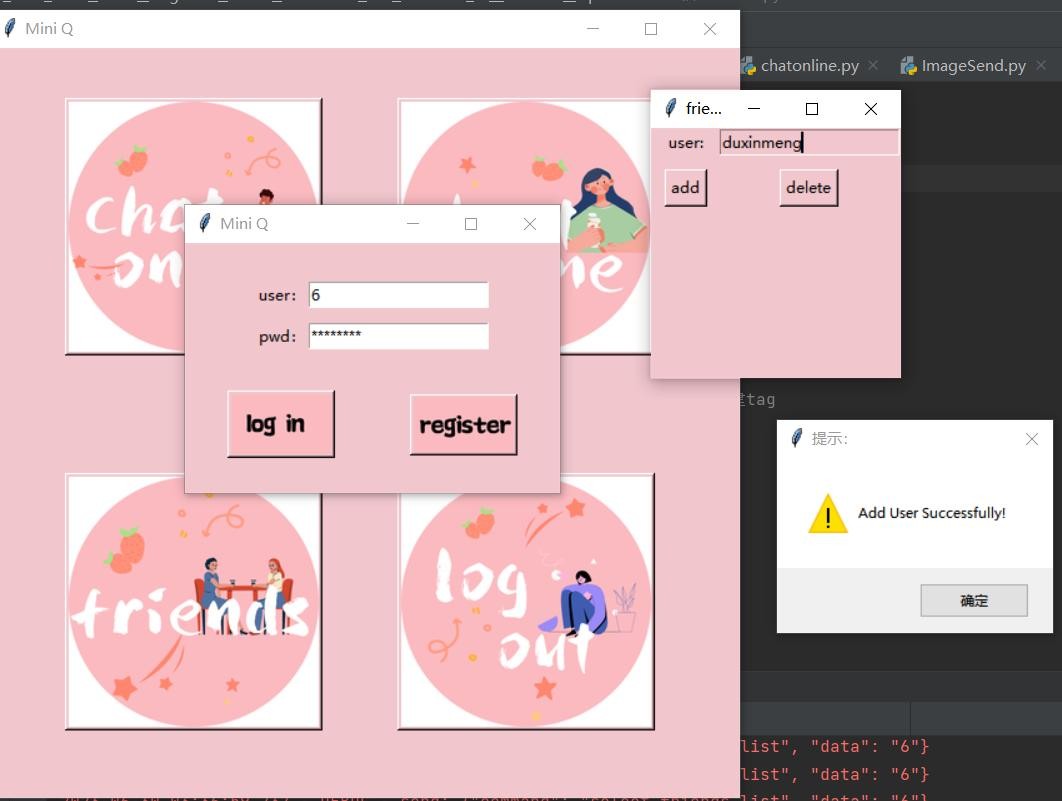
1. **用户注册，密码限制**



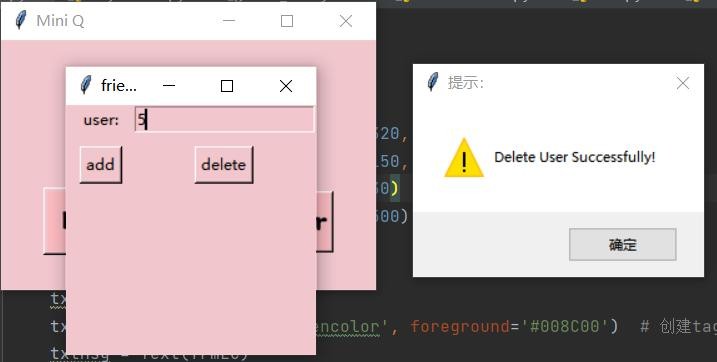
1. **登录**



1. **添加删除好友**

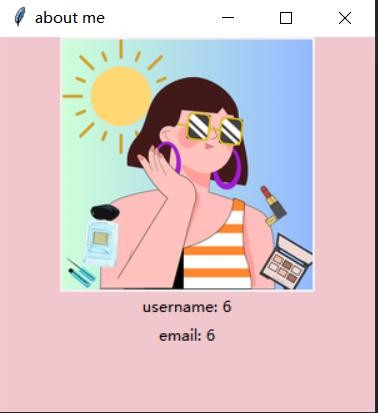


**添加好友**



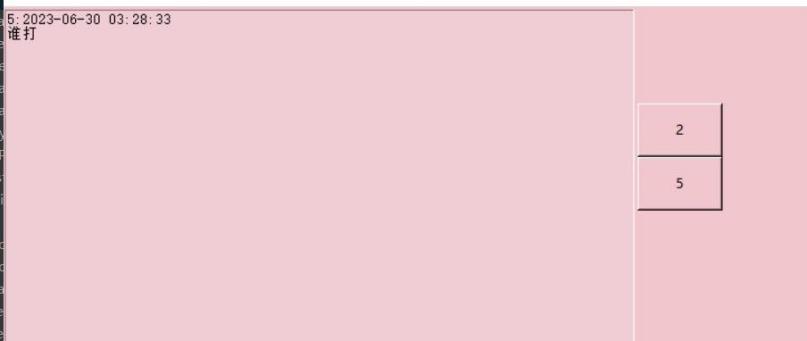
**删除好友**

1. **个人信息界面**
2. **通讯** 发送文字：



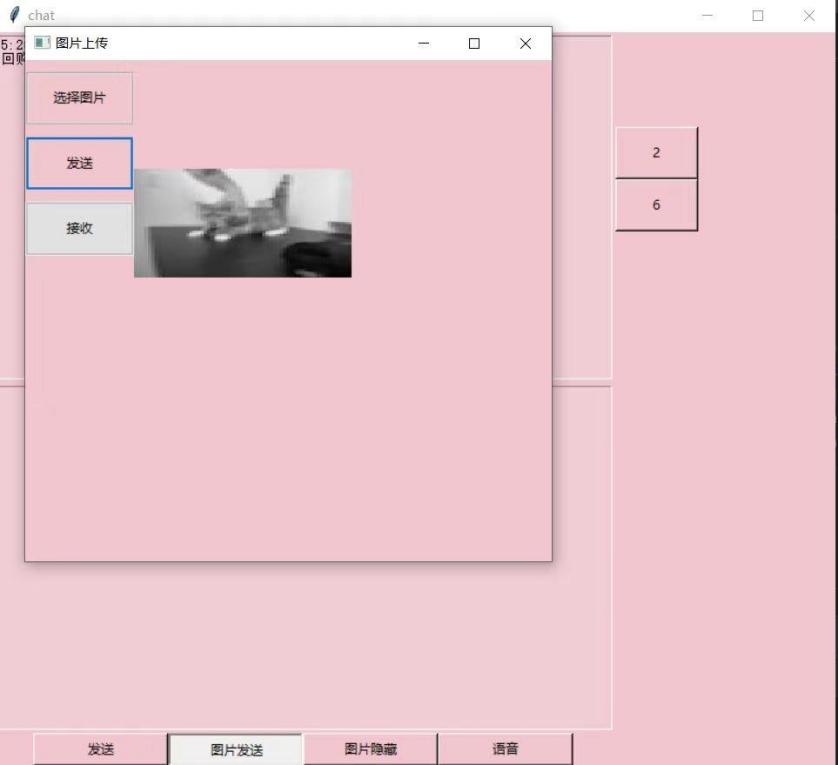


**发送方**

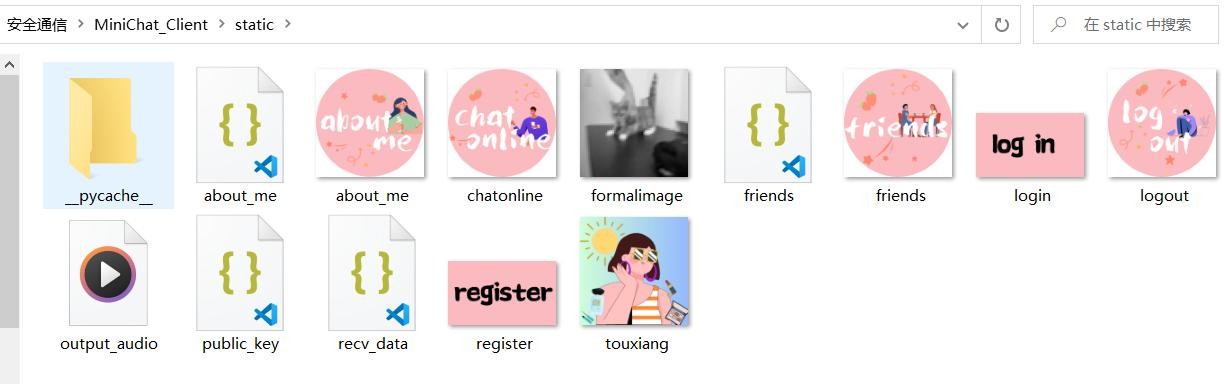


**接收方**

发送图片：



**发送方**



**接收方**

发送语音：

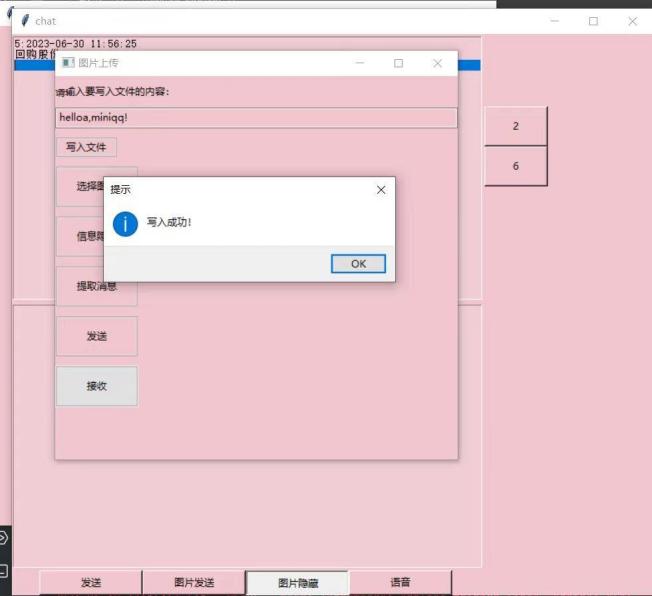
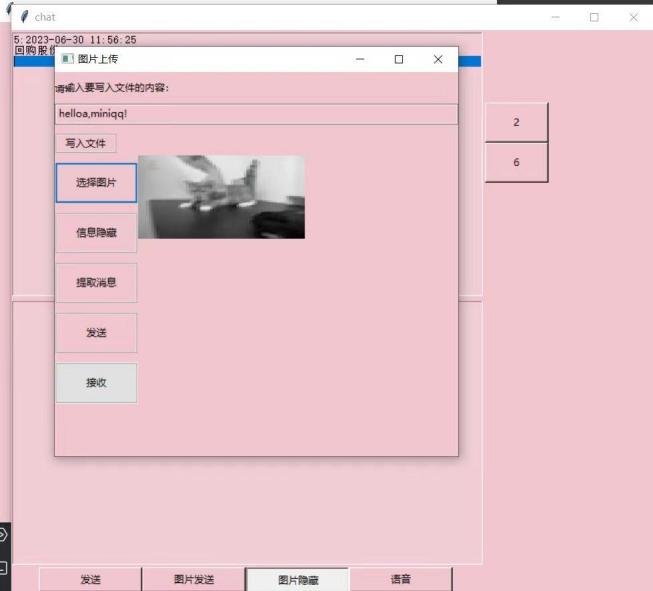


**录音成功发送**

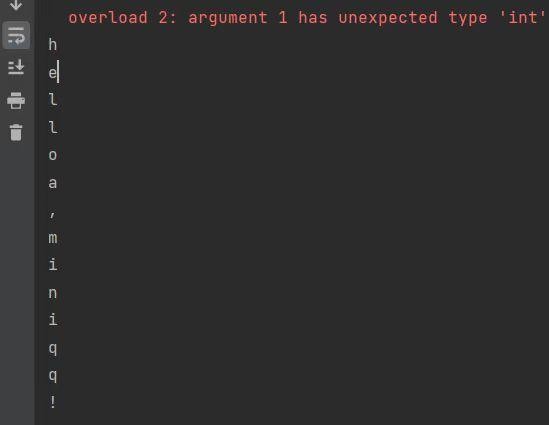


**录音成功接收**

以图片为载体的透明嵌入提取：

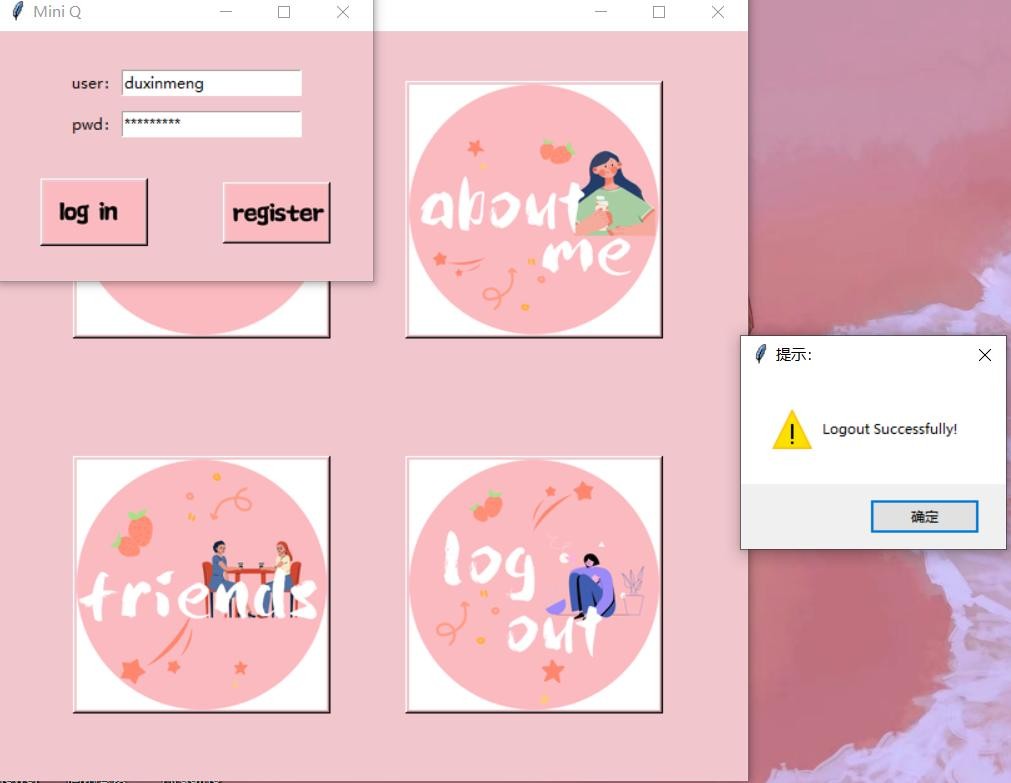


**发送方**



**接收方**

1. **登出**



**成功登出**

1. **即时通信管理系统**



**即时通信管理系统**

# 六、设计心得：

设计与实现的重要性：本次实验涉及到一个较为复杂的系统，需要设计和实现多个功能模块。在开始实施之前，进行充分的设计和规划是非常重要的。我们在第一天通过仔细思考和规划，更清楚地

理解了本系统的需求和功能，并讨论出代码的结构和模块之间的关系。这为我们进一步的设计打下良好基础。

安全性的重要性：在本次实验中，涉及到用户的注册信息、登录凭据以及通信内容的加密等安全性需求。安全性是一个不可忽视的因素，需要在系统设计和实现中充分考虑。我们采用合适的加密算法、安全存储密码、身份验证机制等措施是确保系统安全的关键。同时，对于可能的漏洞和攻击进行风险评估和安全测试，以保护用户的隐私和系统的完整性。

用户体验的关注：作为一个面向用户的系统，良好的用户体验是至关重要的。通过本次实验，我们认识到用户界面的设计、操作的便利性、明确的提示信息等因素对于用户的满意度和系统的可用性起到重要作用。因此，此次设计我们特别关注了界面的美观简洁性，合理的布局和反馈机制，以提供良好的用户体验。

性能优化与可扩展性：本次实验中，系统需要具备快速的响应时间和高吞吐量，同时具备可扩展性，能够应对用户量增长和功能变化的情况。在实践中，我们对代码的性能不断进行优化，如选择合适的数据结构和算法，避免性能瓶颈。同时，我们也思考了系统的扩展性，采用模块化设计和合理的架构，以方便新增功能和适应需求变化。

团队合作与沟通：本次实验是一个较大规模的项目，需要多人协作完成。在团队合作过程中，良好的沟通和协调十分重要。及时的信息共享、明确的任务分工以及协作工具的有效使用，能够提高团队的效率和协同能力，大家不同的思维有利于程序的完善。