

《网络安全程序设计》课程

实验报告

专业班级：_____

学 号：_____

学生姓名：_____黄浩岩_____

实验题目：基于 OpenSSL 的安全聊天系统

指导教师：_____梅松_____

成绩评定：_____

1 实验选题

基于 OpenSSL 的安全聊天系统

- Windows 或 Linux 平台均可
- 点到点模式
- 基于 OpenSSL 的安全套接字通信
- 客户端服务端双向认证
- 聊天记录本地加密存储，输入正确口令可查看

2 系统设计

2.1 开发环境

- 操作系统：Ubuntu16.04
- 编程语言：python3
- 集成开发环境：Pycharm

2.2 系统结构设计

2.2.1 系统框图

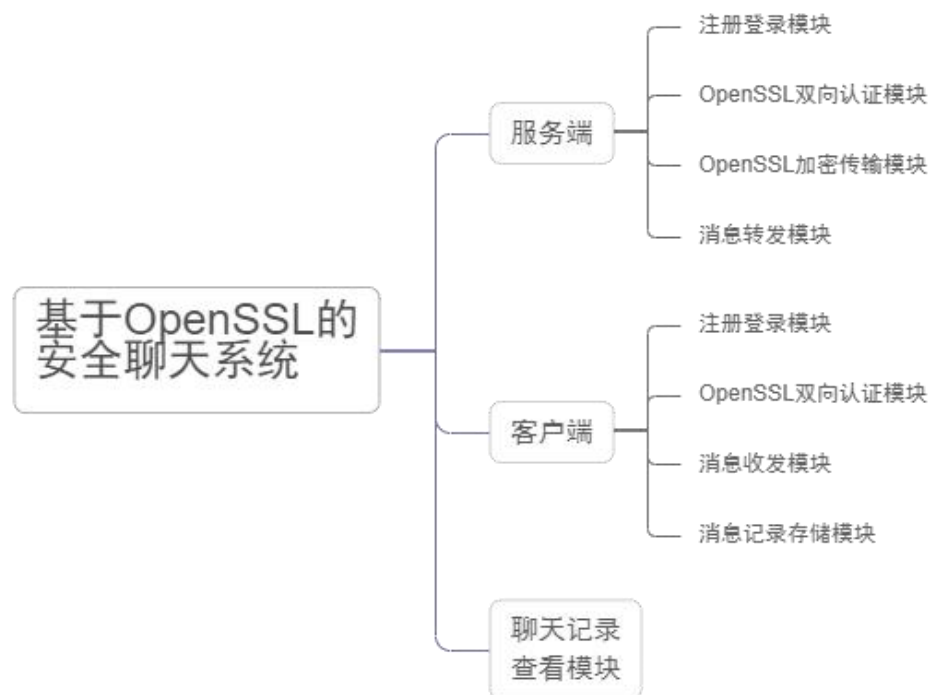


图 2-1 基于 OpenSSL 的安全聊天系统框图

2.2.2 模块描述

服务端：

注册登录模块：对客户端的用户注册数据进行处理，包括对用户名重复检测，对用户名称密码进行加密存储等。同时，基于用户在客户端注册的信息，对用户的名称密码进行验证，验证通过方可进入聊天系统。

OpenSSL 双向认证模块：基于 OpenSSL 进行双向认证。双向认证，即要求服务器和客户端双方都有证书，客户端需要校验服务端，服务端也需要校验客户端，具体过程如下图所示。在服务端，主要将服务端证书发送给客户端，并且对客户端的证书进行验证。

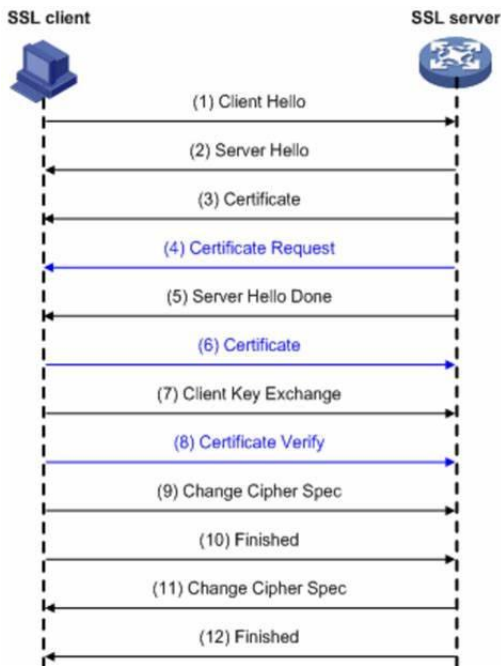


图 2-2 双向认证时序图

OpenSSL 加密传输模块：基于 OpenSSL，对客户端和服务端之间的套接字数据包进行加密传输，这是在双向认证交换密钥的基础上完成的，以保证数据的安全性。

消息转发模块：服务端会对加入聊天系统的用户进行消息的转发，以确保系统中的用户能够接受到彼此发送的消息。

客户端：

注册登录模块：用户通过该模块进行注册和登录，需要与服务端进行验证交互，以确保用户信息的合法，用户需要在客户端输入账号密码后才能进入聊天系统。

OpenSSL 双向认证模块：同服务端，基于 OpenSSL，客户端此处主要对服务端证书进行验证，同时发送自己的证书给服务端。

消息收发模块：客户端进行用户间的消息收发，并基于 OpenSSL 对数据进行加密，发送至服务端。

消息记录存储模块：对聊天的消息记录加密保存至本地。

聊天记录查看模块：

对本地存储的加密聊天记录进行管理，用户需输入密钥后可获得对应用户的本地聊天记录。

2.2.3 系统运行

2.2.3.1 服务端

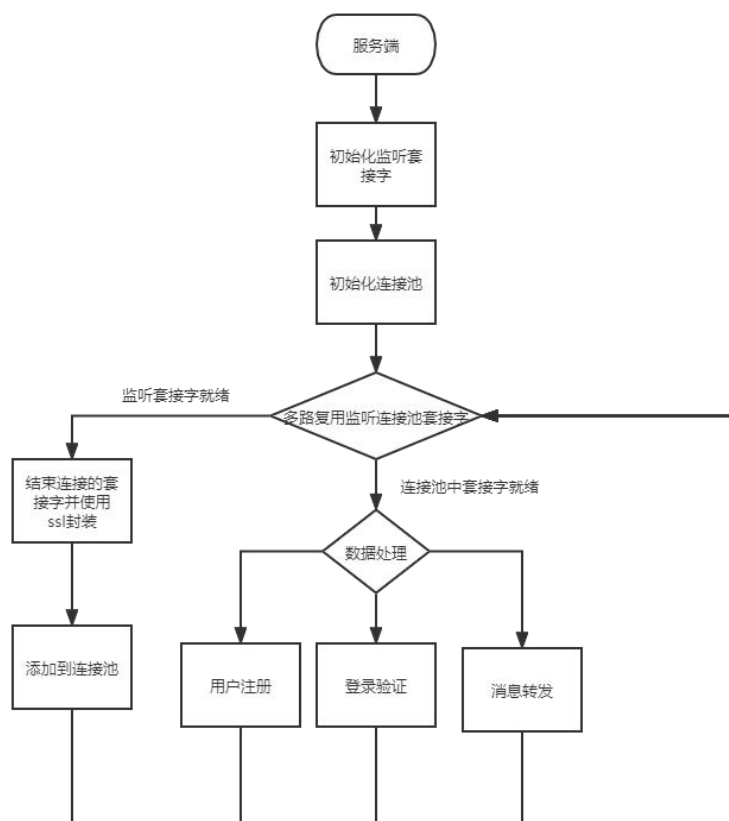


图 2-3 服务端系统运行过程

(1) 服务端启动时首先在一固定端口设置一个监听套接字，用于监听来自客户端的连接

(2) 初始化一个空的连接池，并将监听套接字加入连接池，该连接池用于后续多路复用。

(3) 多路复用监听连接池中的套接字，一直阻塞到套接字就绪，并根据套接字的类型分别处理

(4) 对于监听套接字就绪，则证明有一个客户端与该服务端建立了 TCP 连接，此时接收该套接字并生成新的套接字，然后使用 OpenSSL 对套接字进行封装，完成基于 SSL 的客户端服务端双向认证工作，最后加入到连接池，进行后续的数据通信。

(5) 对于连接池中其它套接字就绪，则该套接字之前已被接收，此时接受的就是具体的聊天系统应用层数据。根据数据的类型，再进行具体的处理：

① 对于注册信息，会对用户的账号进行校验，确保不会重复注册，之后将用户账号和密码进行加密存储，最后告知客户端用户是否注册成功。

② 对于登录信息，会于服务端存储的用户信息进行比对验证，确保密码正确且未重复登录，最后告知客户端用户是否登录成功。

③ 对于其它消息，则是在用户登录成功后在聊天系统中发送的内容，需要将其转发至连接池中其它已登录的用户。

2.2.3.2 客户端

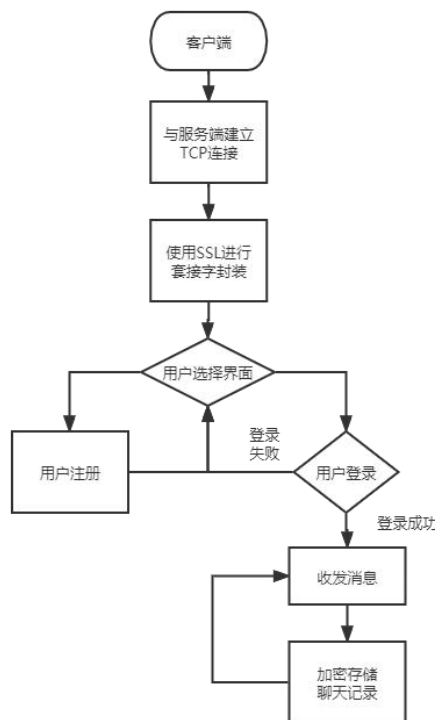


图 2-4 客户端系统运行过程

(1) 客户端首先与选定 IP 地址和端口号的服务端建立 TCP 连接

(2) 然后对建立 TCP 连接的套接字使用 OpenSSL 进行封装，具体而言即进行服务端客户端的双向认证，验证服务端的证书并将自身证书发送给服务端认证，认证通过后，客户端与服务端即建立起了安全套接字。

(3) 建立安全连接后进入用户选择界面，用户可以进行账号注册和用户登录，这两个选项都会与服务端进行账号的验证。

(4) 登录成功后就正式进入聊天室，可以与其他用户进行聊天，能够发送和接收消息。同时对于接收的消息会加密存储到本地。

2.2.3.3 聊天记录查看模块

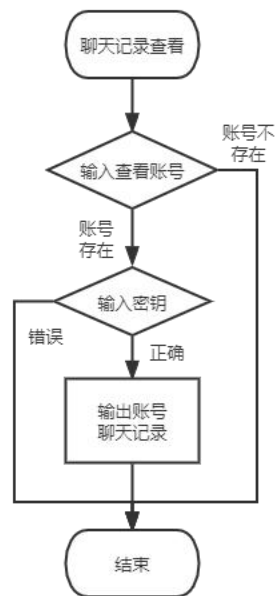


图 2-5 聊天记录查看过程

(1) 聊天记录在本地是分账号进行存储，因此首先要输入查看聊天记录的账号，若账号不存在则直接退出程序。

(2) 输入账号存在后需要输入密钥，以对本地存储的聊天记录进行解密。同样，密码错误则会退出。

(3) 密钥输入正确后会解密输出该账号的聊天记录。

3 详细设计

3.1 主要数据结构、参数定义

系统分为三个程序，一个服务端程序，一个客户端程序，还有一个客户端本地聊天记录查看程序。

在服务端，主要的数据结构包括：用户信息映射 `user_dict`，记录着所有注册的用户账号和加密的密码，用于客户端的登录注册管理；连接池列表 `conn_list`，记录着服务端程序创建的套接字，包括监听套接字以及和客户端建立好的安全套接字；在线用户集合 `active_users`，记录着当前已登录成功的用户，即在聊天系统中能够聊天的用户；连接地址-用户名称映射 `conn_map`，记录着对应套接字远端地址和其账号的对应关系，用于确定用户的离线情况以及数据的转发。参数定义上，主要包括 SSL 相关的服务端证书路径 `SERVER_CERT`、服务端私钥文件路径 `SERVER_KEY_FILE`，根 CA 证书路径 `CA_CERT`，服务端开放的监听端口 `SERVER_PORT`，以及用户信息存储的文件路径 `USER_FILE`。

在客户端，数据结构除了必要的套接字外无其它复杂的数据结构。参数定义

上，包括 SSL 相关的客户端证书路径 CLIENT_CERT、客户端私钥路径 CLIENT_KEY_FILE、根 CA 证书 ca.crt，默认连接的服务端 IP 地址和端口号，以及本地聊天记录的存储文件路径 HISTORY_DIR 和密钥 HISTORY_KEY。

对于聊天记录查看模块，参数定义上包括本地聊天记录的存储文件路径 HISTORY_DIR 和密钥 HISTORY_KEY。

3.2 重要算法说明

系统中重要的算法为利用“IO 多路复用”监听描述符。

在服务端，通过 select 对连接池中的套接字描述符进行多路复用监听，当有描述符就绪时，就对套接字进行处理，对于监听描述符，则进行接收产生新的套接字并使用 SSL 进行封装和双向验证；对于其它描述符，则根据数据的标识分别进行注册、登录和消息转发操作。

在客户端，同样使用 select 进行 IO 多路复用，此处监听的是与服务端建立连接的套接字描述符和控制台标准输入描述符。当套接字描述符就绪时，则证明收到了来自服务端的消息，则根据消息内容进行处理；当控制台标准输入描述符就绪时则将用户输入的数据发送到服务端。

3.3 详细设计

3.3.1 服务端

3.3.1.1 函数列表

序号	函数
1	accept()
2	sign_up(sock, data)
3	sign_in(sock, data)
4	broadcast_data(src_sock, data)

3.3.1.2 函数说明

函数名	accept()
函数说明	接收来自客户端的 TCP 连接，并使用 OpenSSL 进行双向认证和封装
处理流程	接收客户端 TCP 连接→使用 OpenSSL 对套接字进行双向认证和封装 得到安全套接字→将安全套接字加入到连接池

函数名	sign_up(sock, data)
-----	---------------------

函数说明	对客户端注册信息进行处理		
入口参数	参数类型	参 数 名 称	参数说明
	ssl.SSLSocket	sock	与客户端建立的安全套接字
	str	data	收到的客户端数据
处理流程	提取用户名和密码→判断用户名是否注册→对用户信息加密存储→更新用户信息文件→发送客户端注册成功		

函数名	sign_in(sock, data)		
函数说明	对客户端的登录信息进行处理		
入口参数	参数类型	参 数 名 称	参数说明
	ssl.SSLSocket	sock	与客户端建立的安全套接字
	str	data	收到的客户端数据
处理流程	提取用户名密码→判断用户是否注册→判断用户密码是否正确→判断用户是否重复登录→将用户标记为已登录→发送客户端登录成功		

函数名	broadcast_data(src_sock, data)		
函数说明	将收到的数据转发到其它已登录用户的客户端		
入口参数	参数类型	参 数 名 称	参数说明
	ssl.SSLSocket	src_sock	数据来源的安全套接字
	bytes	data	需要转发的数据
处理流程	遍历连接池中的套接字→将数据转发给已登录账号对应的客户端		

3.3.2 客户端

3.3.2.1 函数列表

序号	函数
1	sign_up()
2	sign_in()

3	running_online()
4	save_message()

3.3.2.2 函数说明

函数名	sign_up()		
函数说明	用户注册		
处理流程	输入用户名密码→发送用户名密码到服务端→根据服务端响应判断是否注册成功		

函数名	sign_in()		
函数说明	用户登录		
返回值	类型	取值	取值说明
	bool	True/False	登录成功返回 True，反之为 False
处理流程	输入用户名密码→发送用户名密码到服务端→根据服务端响应判断是否登录成功		

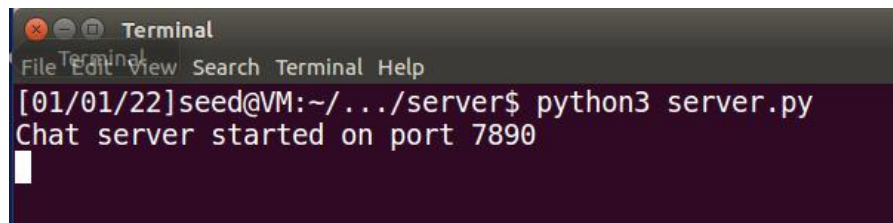
函数名	running_online()		
函数说明	登录后进行数据收发		
处理流程	通过 select 多路复用监听安全套接字和标准输入描述符→套接字就绪则输出数据/标准输入就绪则发送数据到服务端		

函数名	save_message(message)		
函数说明	保存数据到本地		
入口参数	参数类型	参数名称	参数说明
	str	message	需要保存的数据
处理流程	数据 DES 加密→写入聊天记录文件		

4 运行结果

(1) 由于 OpenSSL 双向认证的需要, 首先需要生成根 CA 的整数 `ca.crt`, 服务端的证书 `server.crt` 和私钥 `server.key`, 以及客户端的证书 `client.crt` 和私钥 `client.key`。

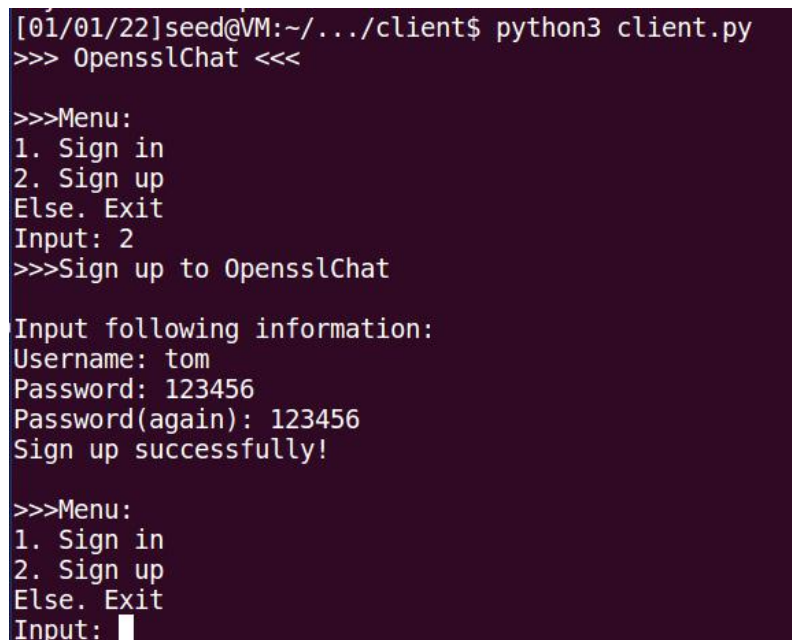
(2) 运行服务端程序, 如图所示:

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is [01/01/22]seed@VM:~/.../server\$. The command python3 server.py has been executed, and the output is Chat server started on port 7890. A cursor is visible on the line following the output.

```
Terminal
File Edit View Search Terminal Help
[01/01/22]seed@VM:~/.../server$ python3 server.py
Chat server started on port 7890
█
```

图 4-1 运行服务端程序

(3) 运行客户端注册账号:

A terminal window showing the execution of the client program. The prompt is [01/01/22]seed@VM:~/.../client\$. The command python3 client.py is entered. The program displays a menu with options 1. Sign in, 2. Sign up, and Else. Exit. The user inputs 2. The program then prompts for Username (tom), Password (123456), and Password (again) (123456). After successful verification, it says "Sign up successfully!". It then shows the menu again, and the user inputs a blank line.

```
[01/01/22]seed@VM:~/.../client$ python3 client.py
>>> OpensslChat <<<

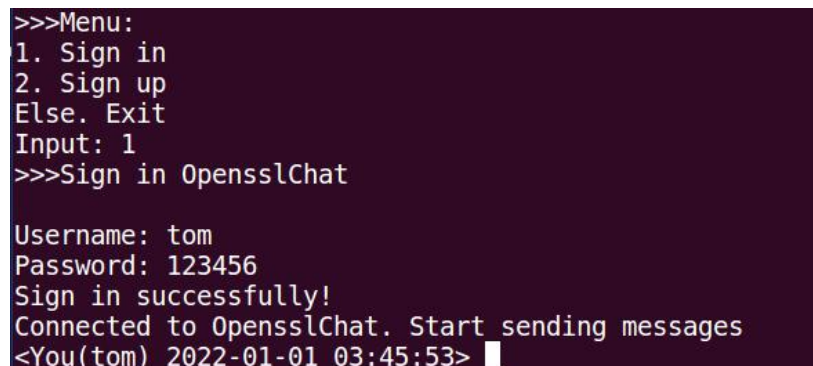
>>>Menu:
1. Sign in
2. Sign up
Else. Exit
Input: 2
>>>Sign up to OpensslChat

Input following information:
Username: tom
Password: 123456
Password(again): 123456
Sign up successfully!

>>>Menu:
1. Sign in
2. Sign up
Else. Exit
Input: █
```

图 4-2 客户端注册账号

(4) 在客户端登录账号:

A terminal window showing the execution of the client program. The prompt is >>>. The program displays the same menu as in Figure 4-2. The user inputs 1. The program then prompts for Username (tom) and Password (123456). After successful verification, it says "Sign in successfully!". It then displays "Connected to OpensslChat. Start sending messages" and a timestamp <You(tom) 2022-01-01 03:45:53>. A cursor is visible on the line following the timestamp.

```
>>>Menu:
1. Sign in
2. Sign up
Else. Exit
Input: 1
>>>Sign in OpensslChat

Username: tom
Password: 123456
Sign in successfully!
Connected to OpensslChat. Start sending messages
<You(tom) 2022-01-01 03:45:53> █
```

图 4-3 客户端登录

(5) 多用户登录聊天:

```
>>>Sign in OpensslChat

Username: tom
Password: 123456
Sign in successfully!
Connected to OpensslChat. Start sending messages
<You(tom) 2022-01-01 03:45:53> $('jack' entered the chat room.
<You(tom) 2022-01-01 03:47:38> nihao
<You(tom) 2022-01-01 03:47:59>
<jack 2022-01-01 03:48:01> hi
<You(tom) 2022-01-01 03:48:01> nice
<You(tom) 2022-01-01 03:48:09> !!!
<You(tom) 2022-01-01 03:48:13>
<jack 2022-01-01 03:48:15> good
<You(tom) 2022-01-01 03:48:15> 

Terminal
File Edit View Search Terminal Help

>>>Sign in OpensslChat

Username: jack
Password: 123456
Sign in successfully!
Connected to OpensslChat. Start sending messages
<You(jack) 2022-01-01 03:47:39>
<tom 2022-01-01 03:47:59> nihao
<You(jack) 2022-01-01 03:47:59> hi
<You(jack) 2022-01-01 03:48:01>
<tom 2022-01-01 03:48:09> nice
<You(jack) 2022-01-01 03:48:09>
<tom 2022-01-01 03:48:13> !!!
<You(jack) 2022-01-01 03:48:13> good
<You(jack) 2022-01-01 03:48:15>
```

图 4-4 多用户登录客户端聊天

```
[01/01/22]seed@VM:~/.../server$ python3 server.py
Chat server started on port 7890
User:'tom'('127.0.0.1', 59114) is online

User:'jack'('127.0.0.1', 59116) is online

User:'tom'(('127.0.0.1', 59114)) is offline

User:'jack'(('127.0.0.1', 59116)) is offline
```

图 4-5 服务端输出信息

(6) 输入用户名和密钥查看本地聊天记录:

```
[01/01/22]seed@VM:~/.../client$ python3 history.py
>>> OpensslChat History <<<
Please input the username: tom
Please input the key(8 bytes): 12345678

Chat History:
$'jack' entered the chat room.

<tom 2022-01-01 03:47:59> nihao

<jack 2022-01-01 03:48:01> hi

<tom 2022-01-01 03:48:09> nice

<tom 2022-01-01 03:48:13> !!!

<jack 2022-01-01 03:48:15> good

[01/01/22]seed@VM:~/.../client$
```

图 4-6 查看本地聊天记录

5 实验与课程总结

在本次网络安全程序设计中，我完成了一个基于 OpenSSL 的安全聊天系统，系统实现了点到点模式、基于 OpenSSL 的安全套接字通信、客户端服务端双向认证、聊天记录本地加密存储输入口令查看，以及账号密码登录等功能。

本次课程设计使用 Python3 实现该系统，因为 Python3 具有丰富的系统库，对于 OpenSSL 的安全套接字通信，Python 中有专门的 ssl 供开发者使用，其中提供了丰富而简洁的接口，让双向认证，套接字加密等操作得以更加便捷高效的处理和开发。通过本次课程设计，我也对 OpenSSL 的编程以及相关的加密通信、双向认证等流程有了更加清晰的认识和理解，巩固了课内所学的知识，同时在编程能力和系统设计能力方面都有了一定的提升，对 Python 相关的 API 的使用也更加数量，收获颇丰。

6 源代码清单

6.1 server.py

```
1.  import hashlib
2.  import select
3.  import signal
4.  import socket
5.  import ssl
6.  import sys
7.
8.  SERVER_PORT = 7890
9.  SERVER_CERT = 'cert/server.crt'
10. SERVER_KEY_FILE = 'cert/server.key'
11. CA_CERT = 'cert/ca.crt'
12. CERT_PASSWORD = '123456'
13. USERS_FILE = 'data/users.ini'
14.
15. conn_list = [] # 连接池
16. active_users = set() # 已在线用户
17. conn_map = {} # 连接地址-用户映射
18.
19.
20. def broadcast_data(src_sock: ssl.SSLSocket, data: bytes):
21.     """数据广播到其它客户端"""
22.     # global conn_list
23.     for sock in conn_list:
24.         if sock != server_socket and sock != src_sock \
25.             and sock.getpeername() in conn_map:
26.             try:
27.                 sock.write(data)
28.             except Exception as e:
29.                 print(e.args)
30.                 sock.close()
31.                 conn_list.remove(sock)
32.
33.
34. def load_users():
35.     user_dict = {}
36.     with open(USERS_FILE, 'r') as file:
37.         for line in file.readlines():
38.             user = line.split('$')
39.             name = user[0].strip()
```

```

40.         pwd = user[1].strip()
41.         user_dict[name] = pwd
42.         return user_dict
43.
44.
45. def accept():
46.     """接收来自客户端的 TCP 连接，并使用 OpenSSL 进行双向认证和封装"""
47.     # 返回一个新套接字 sockfd, 以及另一端套接字绑定的地址 addr(hostaddr,post)
48.     sockfd, _ = server_socket.accept()
49.     ssl_sock = context.wrap_socket(sockfd, server_side=True)
50.     conn_list.append(ssl_sock)
51.
52.
53. def sign_up(sock: ssl.SSLSocket, data: str):
54.     info = data.split('\n\n')
55.     username = info[0]
56.     if username in user_dict:
57.         sock.write('$FAI$User already exists.'.encode())
58.     else:
59.         pwd = hashlib.sha256(info[1].encode()).hexdigest()
60.         user_dict[username] = pwd
61.         with open(USERS_FILE, 'a') as file:
62.             file.write(username + '$$' + pwd + '\n')
63.         sock.write('$SUC$'.encode())
64.
65.
66. def sign_in(sock: ssl.SSLSocket, data: str):
67.     info = data.split('\n\n')
68.     username = info[0]
69.     password = info[1].encode()
70.     if username not in user_dict:
71.         sock.write('$FAI$User does not exist.'.encode())
72.     elif hashlib.sha256(password).hexdigest() != user_dict[username]:
73.         sock.write('$FAI$The password is incorrect.'.encode())
74.     elif username in active_users:
75.         sock.write('$FAI$User has logged in.'.encode())
76.     else:
77.         active_users.add(username)
78.         addr = sock.getpeername()
79.         conn_map[addr] = username
80.         print("User:'%s'%s is online\n" % (username, addr))
81.         sock.write('$SUC$'.encode())
82.         msg = "\n$MSG$'%s' entered the chat room.\n" % username
83.         broadcast_data(sock, msg.encode())

```

```

84.
85.
86. def sign_out(sock: ssl.SSLSocket):
87.     if sock.getpeername() in conn_map:
88.         user = conn_map[sock.getpeername()]
89.         msg = "$MSG$User:'%s' is offline.\n" % user
90.         broadcast_data(sock, msg.encode())
91.         print("User:'%s'(%s) is offline\n" % (user, sock.getpeername()))
92.         active_users.remove(user)
93.     sock.close()
94.     conn_list.remove(sock)
95.
96.
97. def exit_prog(signum, frame):
98.     conn_list.remove(server_socket)
99.     server_socket.close()
100.    for conn in conn_list:
101.        conn.write('$END$Server is down.\n'.encode())
102.        conn.close()
103.    print('\nChat server is down.')
104.    sys.exit()
105.
106.
107. if __name__ == "__main__":
108.     # 创建默认 SSL 上下文
109.     context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
110.     context.load_cert_chain(certfile=SERVER_CERT, keyfile=SERVER_KEY_FILE,
111.                             password=CERT_PASSWORD)
112.     context.load_verify_locations(cafile=CA_CERT)
113.     user_dict = load_users()
114.     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
115.     # SOL_SOCKET: 套接字级别设置
116.     # SO_REUSEADDR: 地址复用
117.     server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
118.     server_socket.bind(('', SERVER_PORT))
119.     # 监听连接, 参数为最大未 accept 的连接数
120.     server_socket.listen(2)
121.     conn_list.append(server_socket)
122.     print("Chat server started on port " + str(SERVER_PORT))
123.     signal.signal(signal.SIGINT, exit_prog)
124.
125.     while True:
126.         # 多路复用返回可读连接列表
127.         read_sockets, _, _ = select.select(conn_list, [], [])

```



```

128.         for sock in read_sockets:
129.             if sock == server_socket:
130.                 accept()
131.             else:
132.                 # addr, port = conn_map[sock]
133.                 data = sock.read(1024)
134.                 if data:
135.                     data = data.decode()
136.                     flag = data[:5]
137.                     msg = data[5:]
138.                     if flag == '$SUP$':
139.                         sign_up(sock, msg)
140.                     elif flag == '$SIN$':
141.                         sign_in(sock, msg)
142.                     else:
143.                         # 转发消息给其它用户
144.                         broadcast_data(sock, data.encode())
145.                 else:
146.                     sign_out(sock)

```

6.2 client.py

```

1.  import binascii
2.  import os.path
3.  import select
4.  import socket
5.  import ssl
6.  import sys
7.  import time
8.  from time import sleep
9.
10. from Crypto.Cipher import DES
11.
12. CA_CERT = 'cert/ca.crt'
13. CLIENT_CERT = 'cert/client.crt'
14. CLIENT_KEY_FILE = 'cert/client.key'
15. CERT_PASSWORD = '123456'
16. SERVER_HOSTNAME = 'hhyserver.com'
17. HISTORY_KEY = b'12345678'
18. HISTORY_DIR = 'data/history/'
19. SERVER_PORT = 7890
20.
21.
22. def prompt():
23.     sys.stdout.write('<You(%s) %s> ' % (user,

```



```

24.                                     time.strftime('%Y-%m-%d %H:%M:%S',
25.                                                         time.localtime(time.time()))))
26.     sys.stdout.flush()
27.
28.
29.     def save_message(message: str):
30.         """保存消息到聊天记录"""
31.         message = message + (8 - len(message) % 8) * ' ' # 八字节对齐
32.         ciphertext = des_obj.encrypt(message.encode())
33.         pass_hex = binascii.b2a_hex(ciphertext)
34.         with open(HISTORY_DIR + user + '.bin', 'ab') as file:
35.             file.write(pass_hex)
36.
37.
38.     def add_label(msg: str):
39.         return '\n<' + user + ' ' + \
40.             time.strftime('%Y-%m-%d %H:%M:%S',
41.                             time.localtime(time.time())) + '> ' + msg
42.
43.
44.     def retry(msg: str) -> bool:
45.         ipt = input(msg + ' [y/Else]')
46.         if ipt == 'y':
47.             return True
48.         return False
49.
50.
51.     def sign_in() -> bool:
52.         print('>>>Sign in OpensslChat\n')
53.         while True:
54.             username = input('Username: ')
55.             password = input('Password: ')
56.             data = '$SIN$' + username + '\n\n' + password
57.             ssl_sock.send(data.encode())
58.             recv = ssl_sock.recv(1024).decode()
59.             if recv == '$SUC$':
60.                 global user
61.                 user = username
62.                 print('Sign in successfully!')
63.                 sleep(1)
64.                 return True
65.             else:
66.                 print('Sign in failed! Error: %s' % recv[5:])
67.                 if not retry('Sign in again?'):

```

```

68.             return False
69.
70.
71. def sign_up():
72.     print('>>>Sign up to OpensslChat\n')
73.     while True:
74.         print("Input following information:")
75.         username = input('Username: ')
76.         password = input('Password: ')
77.         pwd = input('Password(again): ')
78.         if pwd != password:
79.             print('The passwords entered in the two times are inconsistent!')
80.             if not retry('Sign up again?'):
81.                 return
82.         data = '$SUP$' + username + '\n\n' + password
83.         ssl_sock.send(data.encode())
84.         recv = ssl_sock.recv(1024).decode()
85.         if recv == '$SUC$':
86.             print('Sign up successfully!')
87.             sleep(1)
88.             return
89.         else:
90.             print('Sign up failed! Error: %s' % recv[5:])
91.             if not retry('Sign up again?'):
92.                 return
93.
94.
95. def running_online():
96.     print('Connected to OpensslChat. Start sending messages')
97.     prompt()
98.     while True:
99.         read_sockets, _, _ = select.select([sys.stdin, ssl_sock], [], [])
100.        for sock in read_sockets:
101.            if sock == ssl_sock:
102.                data = sock.recv(1024)
103.                if not data:
104.                    print('\nDisconnected from chat server')
105.                    sock.close()
106.                    return
107.                else:
108.                    data = data.decode()[5:]
109.                    save_message(data)
110.                    sys.stdout.write(data)
111.                    prompt()

```

```
112.         else:
113.             msg = sys.stdin.readline()
114.             if msg == '!q\n':
115.                 return
116.             msg = add_label(msg)
117.             save_message(msg)
118.             ssl_sock.send('$MSG$'.encode() + msg.encode())
119.             prompt()
120.
121.
122. def exit_prog():
123.     print('OpensslChat has exited.')
124.     ssl_sock.close()
125.     sys.exit()
126.
127.
128. if __name__ == "__main__":
129.     hostaddr = '127.0.0.1'
130.     port = SERVER_PORT
131.
132.     argc = len(sys.argv)
133.     if argc > 3:
134.         print('Usage: python client.py [server_ip] [server_port]')
135.         sys.exit()
136.     if argc > 1:
137.         hostaddr = sys.argv[1]
138.     if argc > 2:
139.         port = int(sys.argv[2])
140.
141.     if not os.path.exists(HISTORY_DIR):
142.         os.mkdir(HISTORY_DIR)
143.
144.     des_obj = DES.new(HISTORY_KEY, DES.MODE_ECB)
145.     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
146.     context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
147.     context.verify_mode = ssl.CERT_REQUIRED
148.     context.load_cert_chain(certfile=CLIENT_CERT, keyfile=CLIENT_KEY_FILE,
149.                             password=CERT_PASSWORD)
150.     context.load_verify_locations(CA_CERT)
151.     ssl_sock = context.wrap_socket(sock, server_hostname=SERVER_HOSTNAME)
152.     ssl_sock.settimeout(2)
153.
154.     try:
155.         ssl_sock.connect((hostaddr, port))
```

```

156.     except Exception as e:
157.         print(e.args)
158.         print('Unable to connect!')
159.         sys.exit()
160.
161.     user = ''
162.     print('>>> OpensslChat <<<')
163.     while True:
164.         print('\n>>>Menu:\n1. Sign in\n2. Sign up\nElse. Exit')
165.         ipt = input('Input: ')
166.         if ipt == '1':
167.             if sign_in():
168.                 running_online()
169.                 exit_prog()
170.             elif ipt == '2':
171.                 sign_up()
172.             else:
173.                 exit_prog()

```

6.3 history.py

```

1.     import os
2.     import sys
3.     from binascii import a2b_hex
4.
5.     from Crypto.Cipher import DES
6.
7.     MAX_TRY_TIMES = 4
8.     HISTORY_KEY = b'12345678'
9.     HISTORY_DIR = 'data/history/'
10.
11.
12.     print('>>> OpensslChat History <<<')
13.     users = []
14.     for file in os.listdir(HISTORY_DIR):
15.         if not os.path.isdir(file):
16.             users.append(file.rstrip('.bin'))
17.     if not users:
18.         print('No chat history.')
19.         sys.exit()
20.
21.     username = input('Please input the username: ')
22.     if username not in users:
23.         print('No chat history of this user.')
24.         sys.exit()

```

```
25.
26. try_times = 0
27. while try_times < MAX_TRY_TIMES:
28.     k = input('Please input the key(8 bytes): ')
29.     try_times += 1
30.     if k.encode() == HISTORY_KEY:
31.         file = open(HISTORY_DIR + username + '.bin', 'rb')
32.         try:
33.             text = file.read()
34.             finally:
35.                 file.close()
36.
37.         des_obj = DES.new(HISTORY_KEY, DES.MODE_ECB)
38.         ciphertext = a2b_hex(text)
39.         plaintext = des_obj.decrypt(ciphertext)
40.         print('\nChat History:')
41.         print(plaintext.decode())
42.         break
43.     else:
44.         print("Wrong Password! You only have %s trying times!" % (MAX_TRY_TIMES - try_
            times))
45.     result = input("Input to try again, or Ctrl+C to quit")
```