

Basic Info:

Environment: Linux, Python 3

Library: socket, threading, ssl, hashlib, json, pyCrypto

IP, PORT: localhost, 8082(server)

Source Code: <https://github.com/StevChen/SecureMessageChannel>

Implementation:

1. SSL/TLS/login mechanism

Using the openssl to generate the self-signed certificate(public key) and private key pair. And then using the module python3 built in module to wrap the socket into ssl.

- Server has its certificate and private key pair, while client has server certificate in its local file.
- See design diagram below and appendix for detail implementation and screenshot

2. User authentication and username and password management

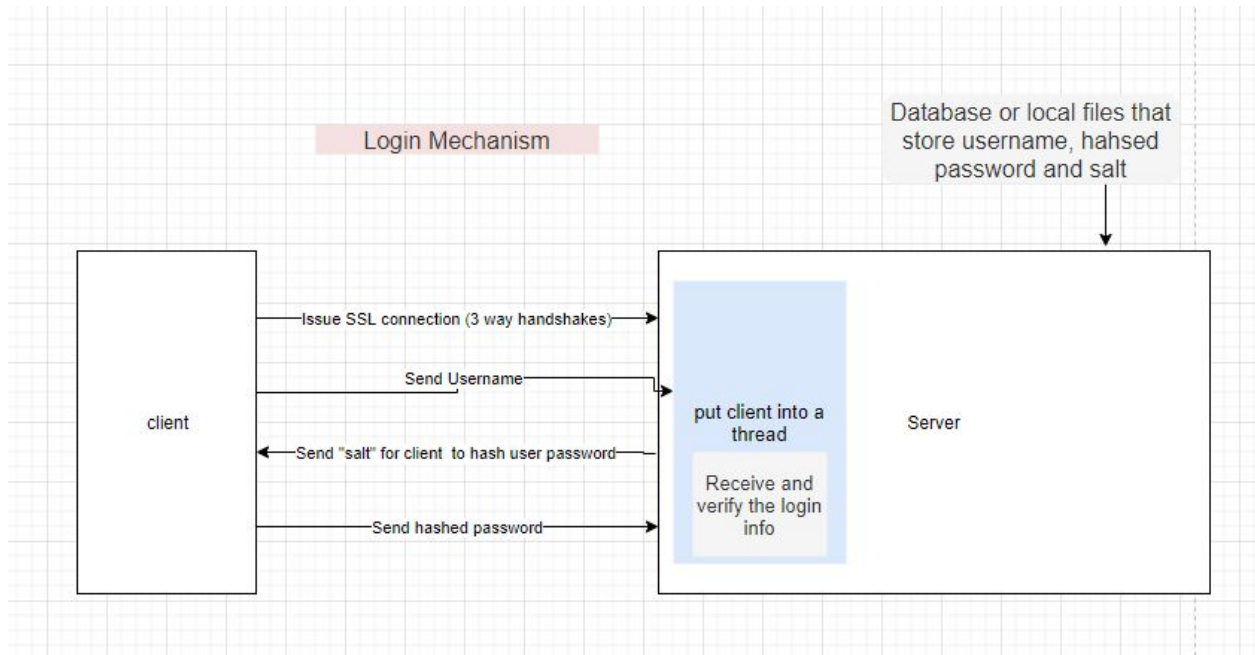
- The user profile, username and user password is saved in the local json file in json format
- There are two groups of user, admin and regular user. They have different accessibility base on their user group. (eg, admin can see all the connected user; regular user can see friend list and chat with friend)

3. Communication module

- There are two clients. Sender and Receiver, the receiver start the wait for chat connection first, and then sender send the chat request.
 - In first round, they do a key exchange. Send send his public key with chat request and then receiver return a packet with receiver's public key to the sender
 - In second round, sender type in the message and send the package with ciphertext. Receiver receive the packet and print out the packet as project request (see appendix for screen shot)

4. Deal with offline client

- See appendix



Login mechanism:

- Client login with username and server send the corresponding salt
- Client input password and server verify the hashed password

```

{
  "0": {
    "username": "admin",
    "password": "c91e58179c6bd209fefe66e3d14364ba7b3513c846b14e4b3b4bfde4d255cf80",
    "salt": "Xqb5XbnCjNU154z",
    "group": "admin",
    "metadata": {
      "friends": [
        ""
      ],
      "group": "admin"
    }
  },
  "1": {
    "username": "user1",
    "password": "17ee9cbc8771f5e42299d5e3409139dbd8271bab97afda26e0c1fcd5a888c26",
    "salt": "ZAT5SbynJnGdcZFy",
    "group": "regular",
    "metadata": {
      "friends": [
        "user2"
      ],
      "group": "regular"
    }
  },
  "2": {
    "username": "user2",
    "password": "f17ebd74274d2932fbc0699d1b276384cf74330cc6eb6c32edb6ebd6a61ec0ef",
    "salt": "HCUHZA2h6jnBnsA1",
    "group": "regular",
    "metadata": {
      "friends": [
        "user1"
      ],
      "group": "regular"
    }
  }
}
  
```

* User profile stored as json format in .json file

```

Waiting for client.....
connection established for client: ('127.0.0.1', 61285)
strat a new thread to handle client requests
Waiting for client.....
{'username': 'user1', 'password': '17ee9cbc8771f5e42299d5e3409139dbd
8271bab97afda26e0c1fcd5a888c26', 'salt': 'ZAt5SbynJnGDCZFy', 'group
': 'regular', 'metadata': {'friends': ['user2'], 'group': 'regular'}}
}
ZAt5SbynJnGDCZFy
user: user1 connected
waiting for password...
user: user1 login successfully
[]

Connection Success
Username: user1
Received Salt: ZAt5SbynJnGDCZFy
Password:
login success
Thank you for using the secure chat
Please pick one of the options below, eg. '0' for exit
[1] Show friend list
[2] Chat with friend
[3] wait for chat connection
[0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request:

```

User login

SSL/TLS:

```

#create context
context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
#context.verify_mode = ssl.CERT_REQUIRED
context.load_cert_chain(certfile=server_cert, keyfile=server_key)
#context.load_verify_locations(cafile=client_certs)

#create socket
bindsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
bindsocket.bind((listen_addr, listen_port))
bindsocket.listen(5)

```

Server side ssl creation

- Server has self-signed certificate(server_cert) and server_key

```

def init_client(self):
    # create ssl context
    self.context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH, cafile=self.server_cert)
    # self.context.load_cert_chain(certfile=self.client_cert, keyfile=self.client_key)

    # create socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sslsock = self.context.wrap_socket(sock, server_side=False, server_hostname=self.server_hostname)

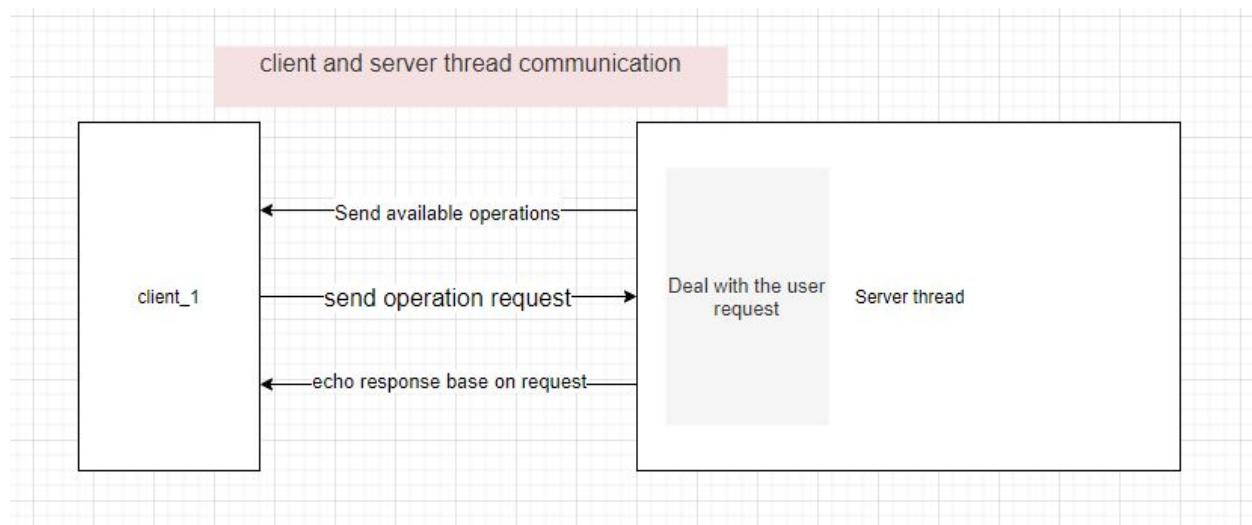
    try:
        self.sslsock.connect((self.host_ip, self.port))
    except Exception as e:
        print("connection fail: ", e, " client exit.")
        sys.exit()

```

Client side ssl creation

- Where server_cert is the certificate save in local at client machine.

Client/Server Communication:



```
hub/securemessagechannel/client_1$ python3
client.py
Connection Success
Username: user2
Password:
login success
Thank you for using the secure chat
Pleas pick one of the options below, eg. '0' for exit
[1] Show friend list
[2] Chat with friend
[3] wait for chat connection
[0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request: █
```

Login as regular user

```

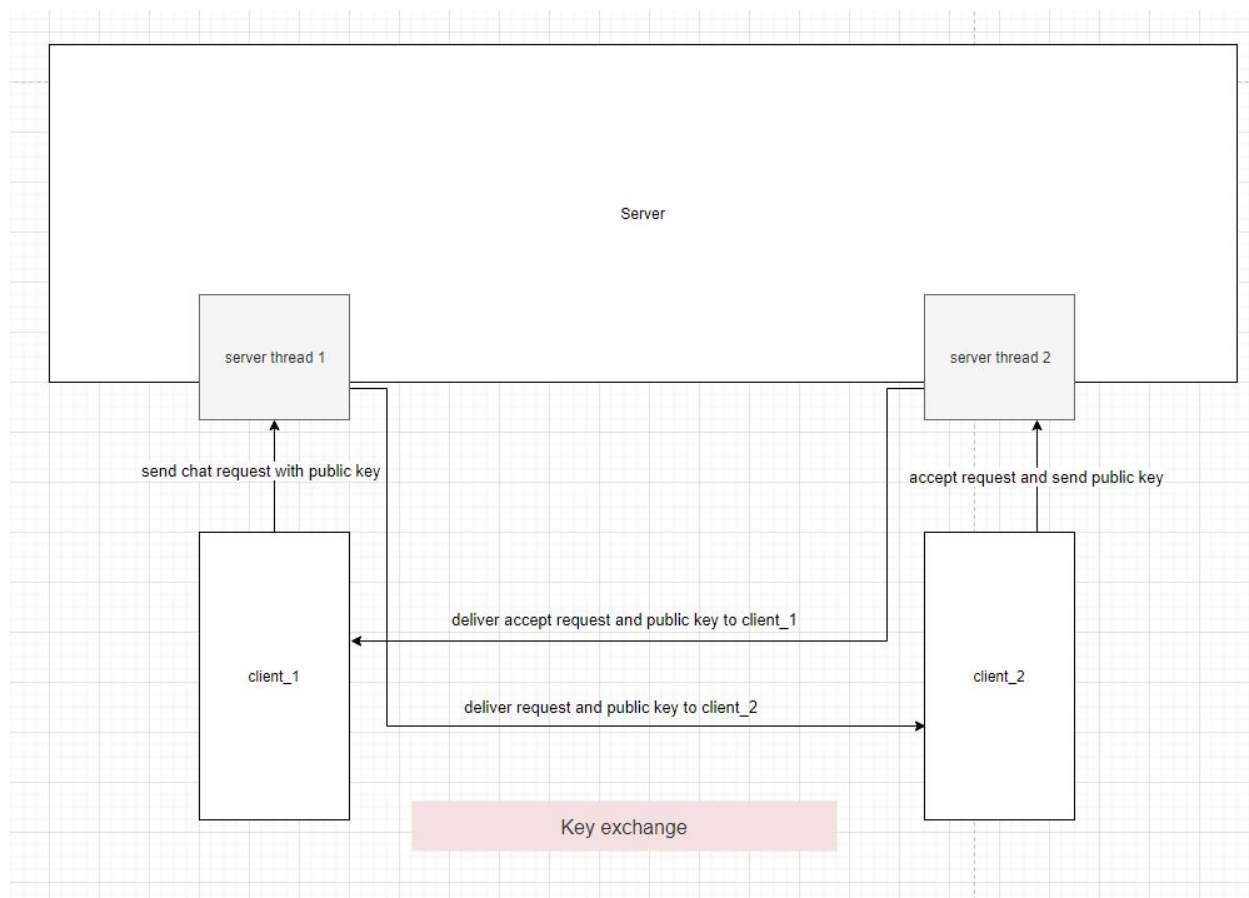
Connection Success
Username: admin
Password:
login success
Welcome admin
Please pick one of the options below, eg. '0' for exit
[1] Show connected users
[2] add new user
[0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request: 1
["user1", "admin"]
Enter your request: 0
Closing connection

```

Login as admin user

- User has different accessibility base on their user group, add new user is not implemented

Key exchange and communication module:



- Client communication following the same idea above, client can send encrypted message to the server thread and server thread will be responsible to deliver the encrypted message to the target client_user.

```

Pleas pick one of the options below, eg. '0' for exit
[1] Show friend list
[2] Chat with friend
[3] wait for chat connection
[0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request: 3
waiting for chat connection
waiting for message
received message
Message: b'A Plan to Turn New York Into a Capital of Cybersecurity'
friend public key(session key):
-----BEGIN PUBLIC KEY-----
MIIBTjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvp4/C9JB0bQGkVf941bb
serXfz5ZtXZTmPTpR6gV7q2KDx8j/hy0zXMkmKCN9i+1RYM1pFls+2JECtsMdk6
UeRGBF/WAm9APsmqLDHTHSPY6fM0VCM7guE4IGOjdmn7BtzKnNM+VDqMmzfcP7Qa
SV03ih5knSCXCBnNMb+5k9fd/cTmuXyn2lg7u4C3tVke1uRXKfZbUudcV/glo7T7
9gzvAYeMmu8r81KBKJWdaeikBPMd3wN+qQNM0MDvxfkHzqMfgtaffYpMpQGMAKJ
iqUXUOh2PR2/QpXJCD6IFwYrYlNBtp6F2uaR6glLZRzm8uyB3A5ny1kHRxA9+hz
mQIDAQAB
-----END PUBLIC KEY-----
ciphertext: 71fef58a941af6885979746b8d506d38d0a4c13db71410cf2304e91dbddcde0a27aa4601316420fb652644920
f31ef2904e6e5e05ed33147c8a3eb5d38db4e84ac617e85249e15855846c58b8853315fe6d7e3fdf6bca67f0a7fd1eb48e3b99
0abe5db81cda17bf61e6eb281b396a4f99a13a97bba37bcf3e5ae1a5c76ba4e62e1fcbdb15b1979bafd3177c672d66ef1dfae1
16bcd6babcb92634397b580ec44ae765da34b0a403a629e05264c3895db3696bb1e9529062f0e14eb23df6ba52f5953b9db20c
06c7c56bb6d98d1d26b1d5373c4beb82dcf2a93b0d118af57fca768ba9ce196a10ad06aea3dc0ebf0da09b38e2ae9ed349f3de
c1fd4d49851d157

```

1. Receiver choose option 3 to wait for chat connection
2. Receiver receiver sender's public key and then, waiting for message
3. And display the result as project requested, the order above, decrypted message, session key and ciphertext

```

Connection Success
Username: user1
Password:
login success
Thank you for using the secure chat
Pleas pick one of the options below, eg. '0' for exit
[1] Show friend list
[2] Chat with friend
[3] wait for chat connection
[0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request: 2
['user2']
Who would you like to chat?user2
Enter your message: A Plan to Turn New York Into a Capital of Cybersecurity
message packet {"id": "2", "user": "user2", "msg": "787e6c8ebb36b08590571b70dffa6fb7fa600fff3f415611ba9e89f488f71c7aaaa
8c1d1207245e8a8351a5f543ebda8d4be54b2953c3a1e1bc5824d08d1fe5c782fc4294e991f7d76ff67a1977057d74537c8004cd3c318c72cd5b333
a25b4d83face15fdf570fe4fee7a480b15d89b604218f06534f20700b795ae2d4f90533c3da09abb3d1940d5f8c8379de26cb17adcf19f401225c0b
5ee613a268e785f51a9bfbdb38eb92d55c1619bee66d1f7a7e0eed86d52e599f81cd880244542ab23ee2673ea124c947f47fbd218258739117fc58a0
e4537bd0144cf5a88f67910b13929dd6227455f3b9e6e9d34aeee021a3aa8bc82c14d5a0dc343da312deb274"}
sent message

```

1. Sender choose friend to chat
2. Sender received receiver's public key
3. Sender input the message and send cipher text

Screenshot with key exchange:

```
Please enter your request (1, 2, 3...without brackets)
Enter your request: 2
['user2']
Who would you like to chat?user2
send my key {"id": "2", "user": "user2", "key": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAg9iXh9hW+23pJqY0Dm6xx\nQwPUL91k
dBM438kXgvZYYk2F5QIGZvzRYE/h9yy4/PdN9DnD9w36K6VWEAajFP7P\n+nW0sH6RkOHZt+4SutW
53JiPB5a0n+X9GEZaF/JvqG92pEASVeq/61b1EJVa+u2DA\nK0vxRzKeieNVYed+ADnM/ky6EDP4
dVda7+Av4vcs2RSA3aG2ad0EISqIWGcfnT6Q\nyQ10nHipErIFwGicJCeKJqKdlm3nkBfIW10zle
V3T/7I+vaInkfRm8o4a0UJRwYM\nSIsoyiU2e9h/ogEBI+mkCsHYn0CC2c1J2HuLeCrKA+7h5oAB
Uj8oJUj+Or/8Z+sq\nkwIDAQAB\n-----END PUBLIC KEY-----"}
received friend key -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxn+Rhs6dDAhJb7WUaJ+n
rxKDBB0MdcRMBuL8d0Xp71fie7Gzd9+J1W7Q+jZhhrOH6pMB1i9y+Ly0e7r0FY1
E5mkkcCjAWHuJrpKewaLtdzX+cWatnDhY0MITKSeh/640geJe8/EFmAYNic4afKL
KVXTY/8woH2j7QRT3tRbkgFQEYZcqnchs+849wf6sqvKuo2DGR1QQgw5fNTUsTi4
2VDj13uY4cUpm1khD0wie58ZqumNExFuydXatm8Isx80EC9mkezUy6gPwqKPymUZ
/cvxLkAsmgckd16LEzLJYrvrKoofQNEH+H0UYr6k5LqE81I6S6TIKInUGAlfMH68
WwIDAQAB
-----END PUBLIC KEY-----
Enter your message: 
```

Sender above, receiver below

```
Please enter your request (1, 2, 3...without brackets)
Enter your request: 3
waiting for chat connection
Got friend key -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAg9iXh9hW+23pJqY0Dm6xx
QwPUL91kdBM438kXgvZYYk2F5QIGZvzRYE/h9yy4/PdN9DnD9w36K6VWEAajFP7P
+W0sH6RkOHZt+4SutW53JiPB5a0n+X9GEZaF/JvqG92pEASVeq/61b1EJVa+u2DA
K0vxRzKeieNVYed+ADnM/ky6EDP4dVda7+Av4vcs2RSA3aG2ad0EISqIWGcfnT6Q
yQ10nHipErIFwGicJCeKJqKdlm3nkBfIW10zleV3T/7I+vaInkfRm8o4a0UJRwYM
SIsoyiU2e9h/ogEBI+mkCsHYn0CC2c1J2HuLeCrKA+7h5oABUj8oJUj+Or/8Z+sq
kwIDAQAB
-----END PUBLIC KEY-----
send my key {'id': '3', 'user': 'user1', 'key': '-----BEGIN PUBLIC KEY---
--\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxn+Rhs6dDAhJb7WUaJ+n\nrxKD
BB0MdcRMBuL8d0Xp71fie7Gzd9+J1W7Q+jZhhrOH6pMB1i9y+Ly0e7r0FY1\nE5mkkcCjAWHu
jrpKewaLtdzX+cWatnDhY0MITKSeh/640geJe8/EFmAYNic4afKL\nKVXTY/8woH2j7QRT3tRb
kgFQEYZcqnchs+849wf6sqvKuo2DGR1QQgw5fNTUsTi4\n2VDj13uY4cUpm1khD0wie58ZqumN
ExFuydXatm8Isx80EC9mkezUy6gPwqKPymUZ\n/cvxLkAsmgckd16LEzLJYrvrKoofQNEH+H0U
Yr6k5LqE81I6S6TIKInUGAlfMH68\nWwIDAQAB\n-----END PUBLIC KEY-----'}
waiting for message
```

Key pair generation:

```
# private and public key generation
private_key = RSA.generate(2048)
public_key = private_key.publickey()
session_key = public_key
```

Deal with offline client:

```
.py
Connection Success
Username: user1
Password:
login success
Thank you for using the secure chat
Please pick one of the options below, eg. '0' for exit
[1] Show friend list
[2] Chat with friend
[3] wait for chat connection
[0] Disconnect from the server
Please enter your request (1, 2, 3...without brackets)
Enter your request: 2
['user2']
Who would you like to chat?user2
user is not connected
Enter your request: █
```

1. If user is not on the connected user list, send the message user is not online