Secure UDP Chat Application

Instructor: Dr. Yusuf Ozturk

Course: CompE 560 (Computer Data Networks)

Student Name: Kashmira Nitin Chavan

Red Id: 132721430

Objective

This project aims to develop a secure and reliable real-time UDP-based chat application that meets graduate-level security and network requirements. It implements hybrid cryptography (RSA + AES), message authentication (HMAC), and reliable transport (UDP with ACK and retransmission).

Design and Architecture

The system consists of a UDP server and multiple GUI-based clients. The server manages key exchange and message rebroadcasting. Each client connects, securely exchanges keys, and uses AES to encrypt chat messages. Clients use HMAC to verify message authenticity. Message IDs are used to support reliable UDP and avoid duplication.

Methodology

The client starts by generating RSA keys and sending the public key to the server. The server responds with an AES key encrypted using RSA. This AES key is used to securely encrypt chat messages. Each message includes a unique ID and is authenticated with HMAC. Clients send back ACKs for every message they receive. If no ACK is received, the message is retransmitted. The chat is operated through a user-friendly **GUI** that allows users to type and view messages in real time.

How To Run:-

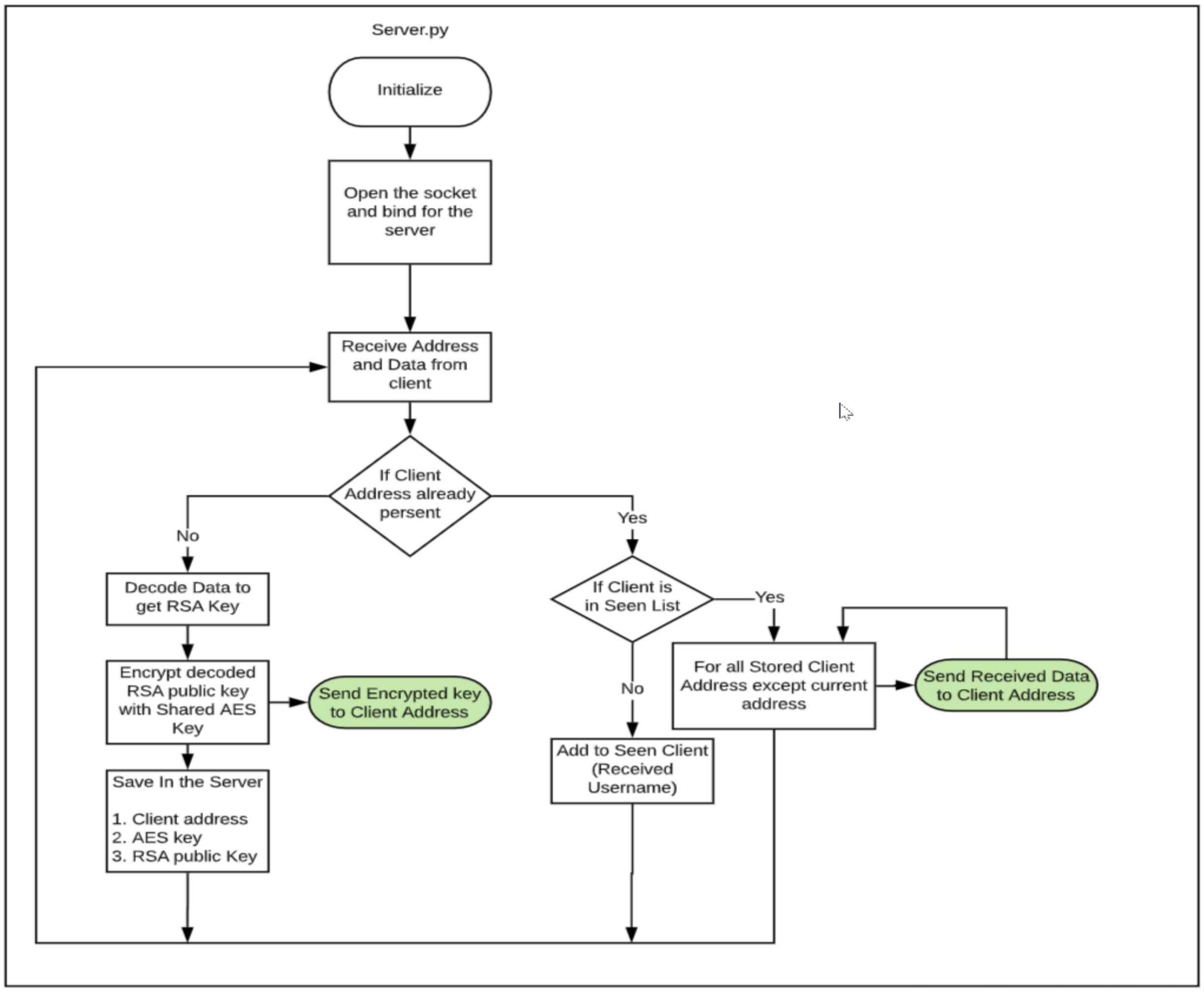
1. Install Dependencies**:** pip install pycryptodome

2. Run the Server: python server.py

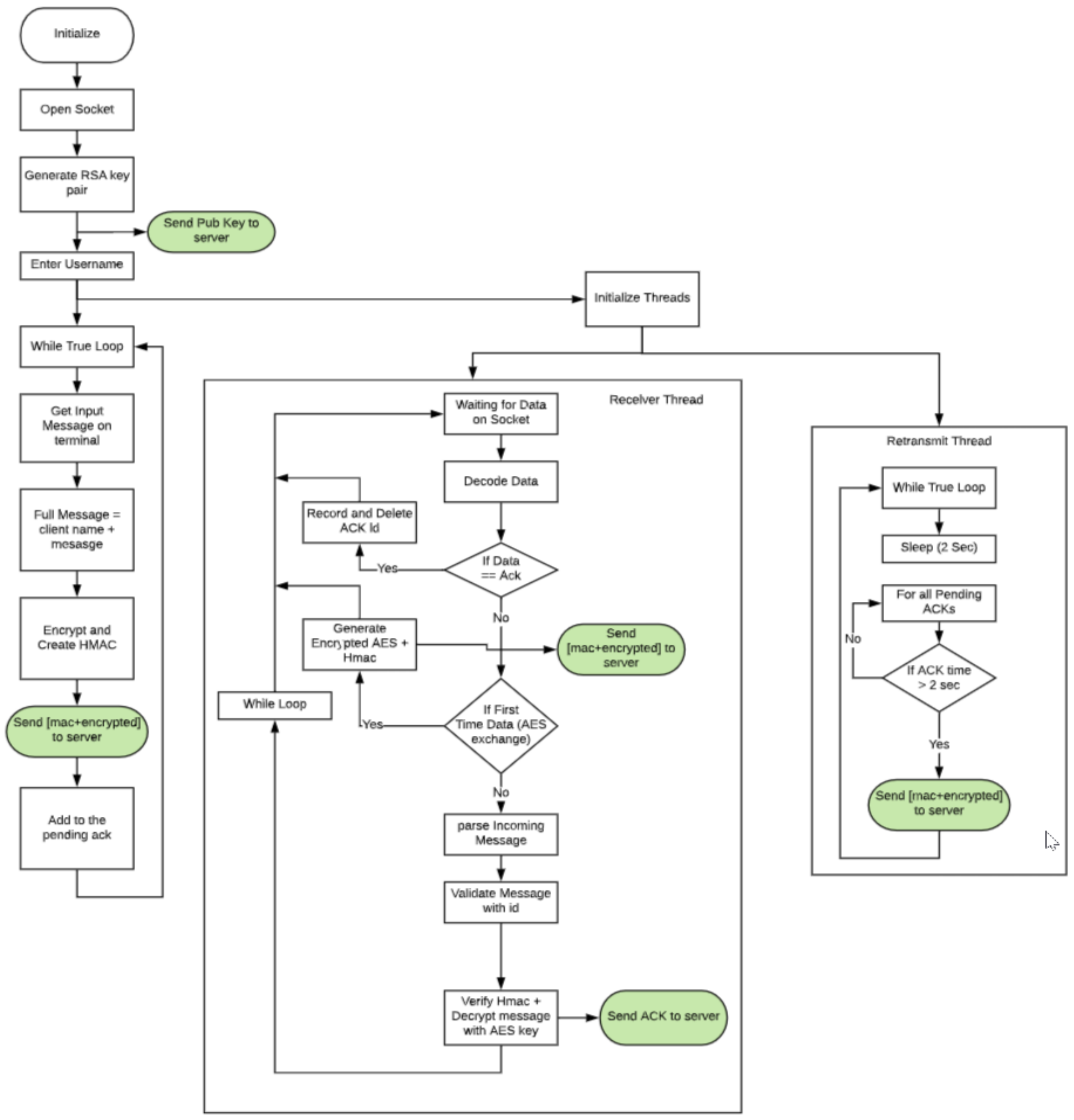
3. Run the Client: python client.py

4. Run the GUI: python client.py --gui

Flow chart:-

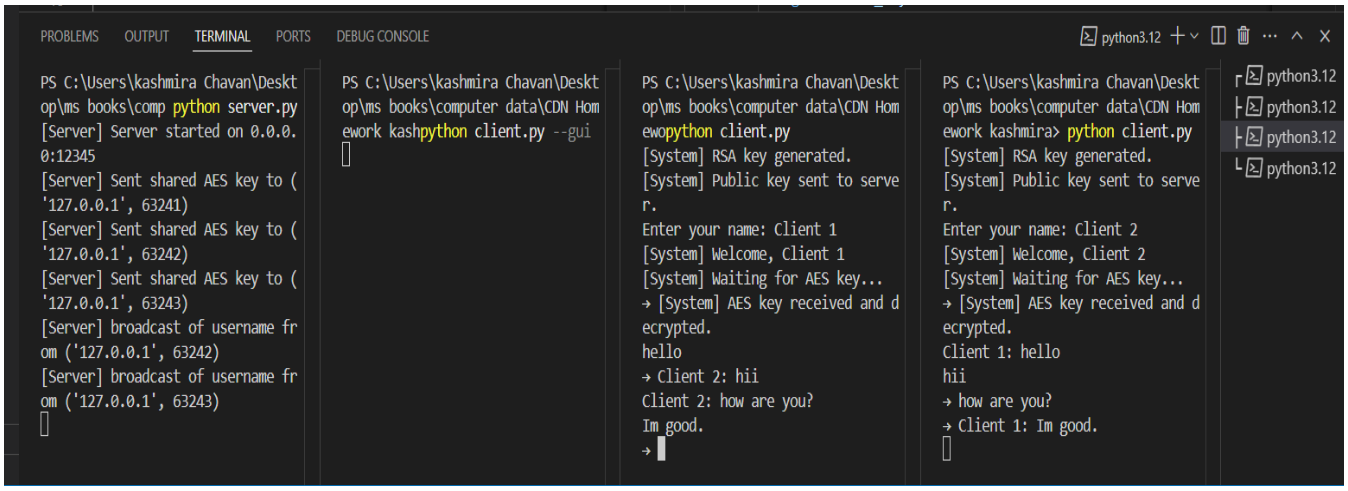

Server.py
Initialize
Open the socket and bind for the server
Receive Address and Data from client
If Client Address already persent
No
Yes
Decode Data to get RSA Key
Encrypt decoded RSA public key with Shared AES Key
Send Encrypted key to Client Address
Save In the Server
1. Client address
2. AES key
3. RSA public Key
If Client is in Seen List
Yes
No
For all Stored Client Address except current address
Send Received Data to Client Address
Add to Seen Client (Received Username)

Client.py (Terminal Mode)



Result:-

The chat application successfully exchanges encrypted and authenticated messages between multiple clients. The server broadcasts messages, and clients confirm receipt using ACKs. Retransmission logic ensures delivery in case of UDP loss. The GUI supports live message updates and user input.

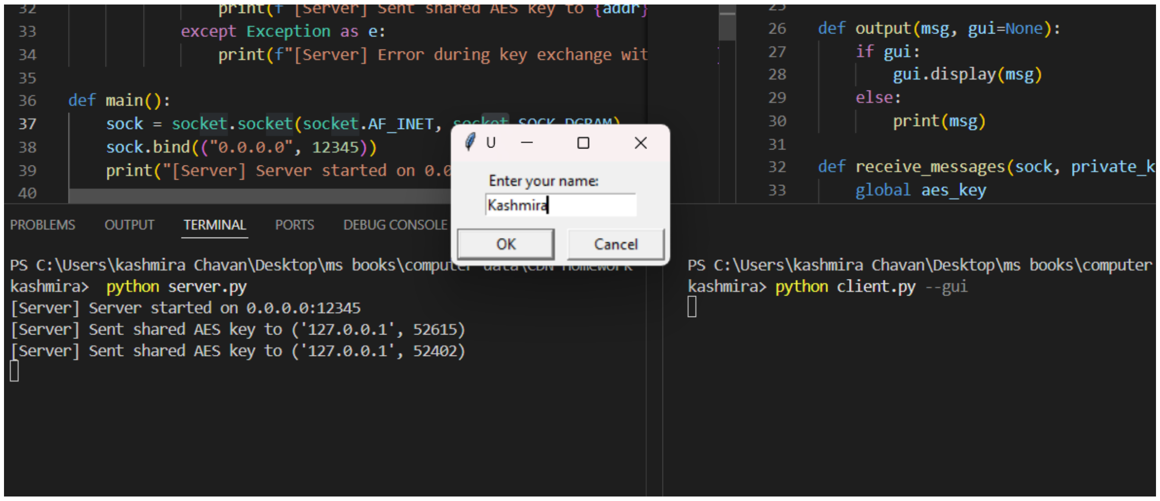


```
PS C:\Users\kashmira Chavan\Deskt
op\ms books\comp python server.py
[Server] Server started on 0.0.0.
0:12345
[Server] Sent shared AES key to (
'127.0.0.1', 63241)
[Server] Sent shared AES key to (
'127.0.0.1', 63242)
[Server] Sent shared AES key to (
'127.0.0.1', 63243)
[Server] broadcast of username fr
om ('127.0.0.1', 63242)
[Server] broadcast of username fr
om ('127.0.0.1', 63243)
```

```
PS C:\Users\kashmira Chavan\Deskt
op\ms books\computer data\CDN Hom
ework kashpython client.py --gui
```

```
PS C:\Users\kashmira Chavan\Deskt
op\ms books\computer data\CDN Hom
ewopython client.py
[System] RSA key generated.
[System] Public key sent to serve
r.
Enter your name: Client 1
[System] Welcome, Client 1
[System] Waiting for AES key...
→ [System] AES key received and d
ecrypted.
hello
→ Client 2: hii
Client 2: how are you?
Im good.
→
```

```
PS C:\Users\kashmira Chavan\Deskt
op\ms books\computer data\CDN Hom
ework kashmira> python client.py
[System] RSA key generated.
[System] Public key sent to serve
r.
Enter your name: Client 2
[System] Welcome, Client 2
[System] Waiting for AES key...
→ [System] AES key received and d
ecrypted.
Client 1: hello
hii
→ how are you?
→ Client 1: Im good.
```



```
            print(f"[Server] Sent shared AES key to {addr}")
        except Exception as e:
            print(f"[Server] Error during key exchange wit

def main():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind(("0.0.0.0", 12345))
    print("[Server] Server started on 0.0
```

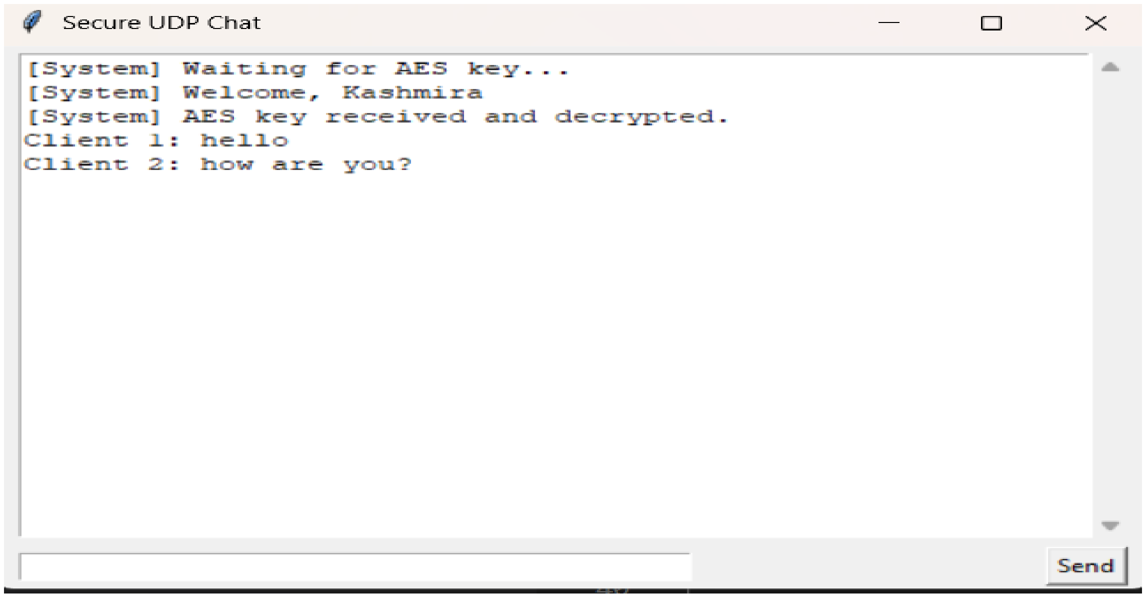
```
def output(msg, gui=None):
    if gui:
        gui.display(msg)
    else:
        print(msg)

def receive_messages(sock, private_k
    global aes_key
```

Enter your name: Kashmira

```
PS C:\Users\kashmira Chavan\Desktop\ms books\computer data\CDN Homework
kashmira>  python server.py
[Server] Server started on 0.0.0.0:12345
[Server] Sent shared AES key to ('127.0.0.1', 52615)
[Server] Sent shared AES key to ('127.0.0.1', 52402)
```

```
PS C:\Users\kashmira Chavan\Desktop\ms books\computer
kashmira> python client.py --gui
```



```
[System] Waiting for AES key...
[System] Welcome, Kashmira
[System] AES key received and decrypted.
Client 1: hello
Client 2: how are you?
```

5. Conclusion

This project demonstrates secure communication over unreliable transport by combining cryptographic techniques with custom reliable delivery logic.