COMPE 560 Homework Report: UDP-Based Chat Application

Name: Krishna Gajera
Red Id:132625971

Introduction

For UDP-Based Chat Application, I developed a secure chat application using Python and UDP sockets. This application allows multiple users to send and receive messages through a central server, with all communications protected by strong encryption. As a graduate student, I implemented additional features, including message authentication, a terminal-based user interface, and Error handling for packet loss. This report explains the design and implementation how I met all assignment requirements.

System Design

Components

The application consists of three main files:

- **client_grad.py**: Generates RSA keys, exchanges AES key, encrypts/decrypts messages, handles UI & retries.
- **server_grad.py**: Manages server-side tasks, including key exchange, message decryption, and broadcasting to clients.
- **crypto_utils_grad.py**: Contains encryption and authentication functions for RSA, AES, and HMAC.

How It Works

The system operates as follows:

1. **Client Startup**:
   - Each client generates a 2048-bit RSA key pair (public and private keys).
   - The client sends its public key to the server.
2. **Key Exchange**:
   - The server generates a unique 128-bit AES key for the client.
   - The server encrypts the AES key using the client's RSA public key and sends it back.
   - The client decrypts the AES key using its private key.
3. **Message Communication**:
   - Clients encrypt messages using the AES key and send them to the server.
   - The server decrypts the messages, then re-encrypts and broadcasts them to other clients.
   - Each client decrypts incoming messages using its AES key.
4. **Message Authentication**:
   - Each message includes an HMAC tag to ensure it hasn't been altered and comes from a legitimate sender.
5. **User Interface**:
   - The client uses a terminal-based interface to display and send messages.
6. **Packet Loss Handling**:
   - The client tracks sent messages and resends them if no acknowledgment is received from the server.

Security Features

Hybrid Encryption

To keep communications secure, I used a combination of RSA and AES encryption:

- **RSA (2048-bit)**: RSA is used to securely share the AES key. The client's public key encrypts the AES key, and only the client's private key can decrypt it, ensuring no one else can access the AES key.
- **AES (128-bit)**: AES encrypts all chat messages. It's fast and secure, making it ideal for real-time communication. Each client has a unique AES key shared with the server.

The encryption process uses AES in CBC mode with a random initialization vector (IV) for each message, ensuring that identical messages produce different ciphertexts. Padding is applied to make the message length compatible with AES's block size.

Message Authentication with HMAC

To prevent tampering and verify sender identity, I implemented HMAC (Hash-based Message Authentication Code) using SHA256:

- When a message is encrypted, an HMAC tag is computed over the encrypted data (IV and ciphertext) using the AES key.
- The tag is sent along with the message.
- The receiver recomputes the HMAC tag and compares it with the received tag. If they match, the message is authentic and untampered; otherwise, it's rejected. This ensures that messages are both confidential (via AES) and authentic (via HMAC).

User Interface

The curses UI is split into:

- **Chat window** – Scroll-back log of inbound/outbound messages and system events.
- **Input window** – A persistent prompt (You:) for composing messages.

Status lines are colour-tagged (blue = sent, green = delivered, red = failed) for fast visual feedback.

Error Handling

- **ACK timer:** 2 s
- **Max retries:** 3
- **Failure escalation:** After third timeout, message is dropped, [FAILED] logged locally, server remains unaffected.
- **Logging:**
  - chat.log – client-side lifecycle of every message id
  - server_chat.log – key exchanges, incoming DATA, broadcast events, ACK sends.

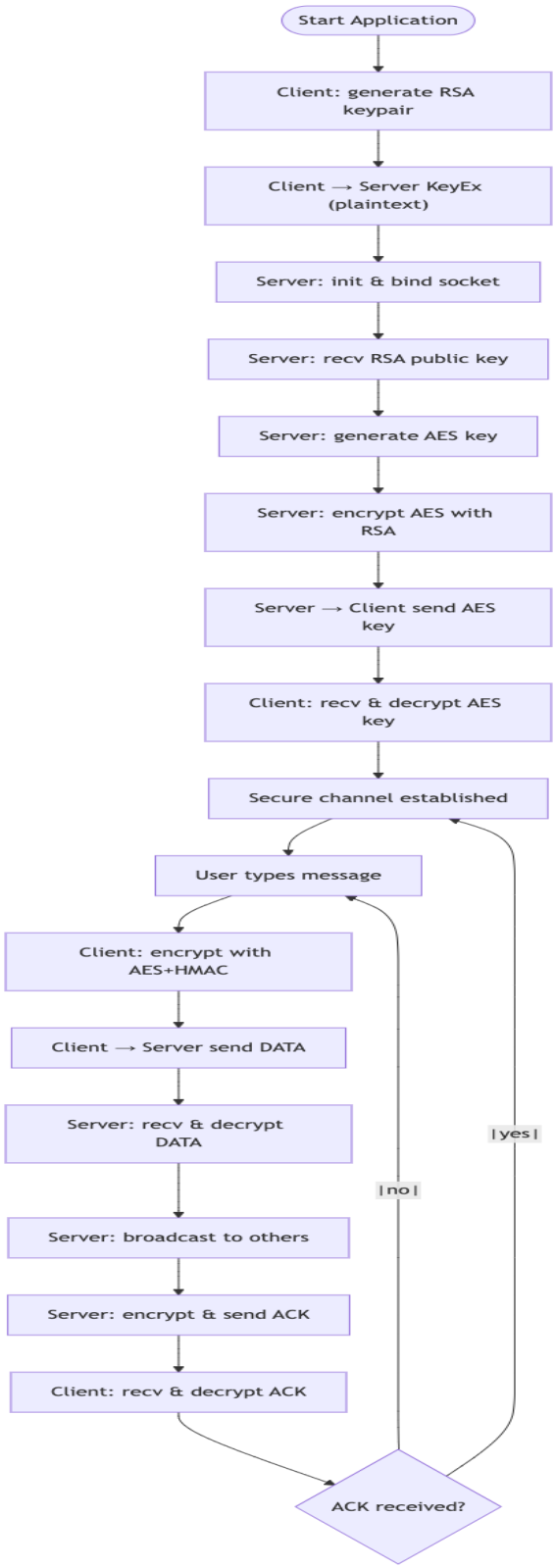
This design meets the graduate-level requirement to "simulate reliable UDP.

Implementation Details

Key Functions

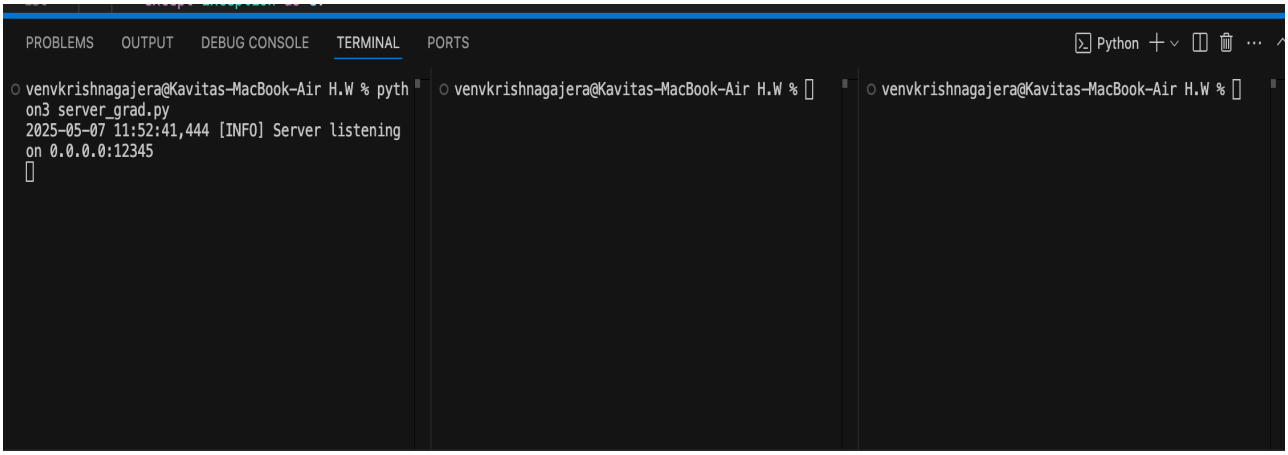
File	Function	Purpose
crypto_utils_grad.py	generate_rsa_keypair()	Generates a 2048-bit RSA key pair for the client.
crypto_utils_grad.py	encrypt_with_rsa()	Encrypts the AES key using the client's RSA public key.
crypto_utils_grad.py	decrypt_with_rsa()	Decrypts the AES key using the client's RSA private key.
crypto_utils_grad.py	encrypt_with_aes()	Encrypts messages with AES, including HMAC for authentication.
crypto_utils_grad.py	decrypt_with_aes()	Decrypts messages with AES, verifying HMAC before decryption.
client_grad.py	receive_messages()	Handles incoming messages, including AES key decryption and chat display.
client_grad.py	ack_monitor()	Monitors pending messages.
server_grad.py	broadcast()	Broadcasts decrypted messages to all clients after re-encryption.

Flow diagram:



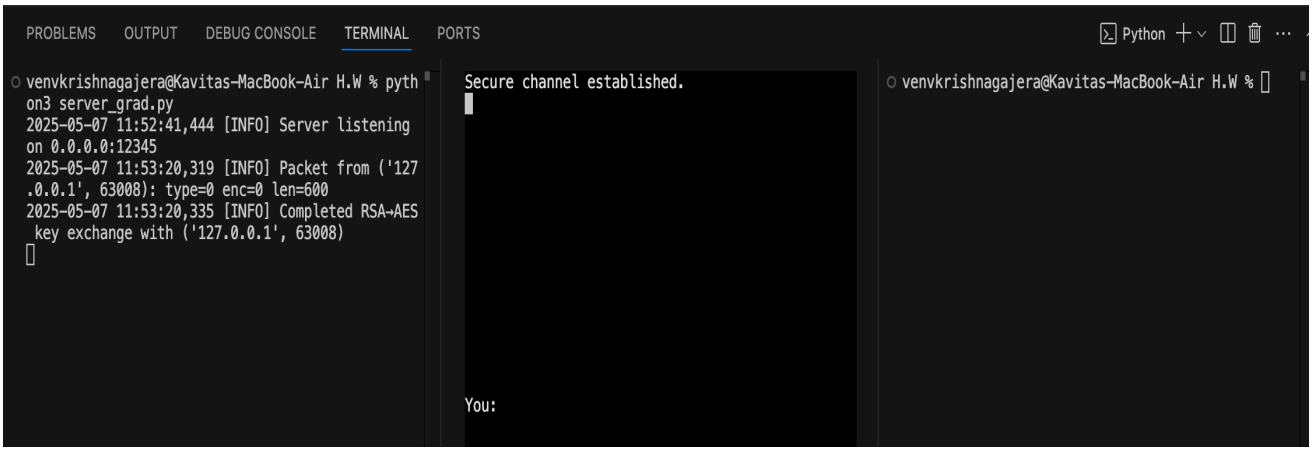
Demonstration

1. ***Server is successfully launched and listening on port 12345:***

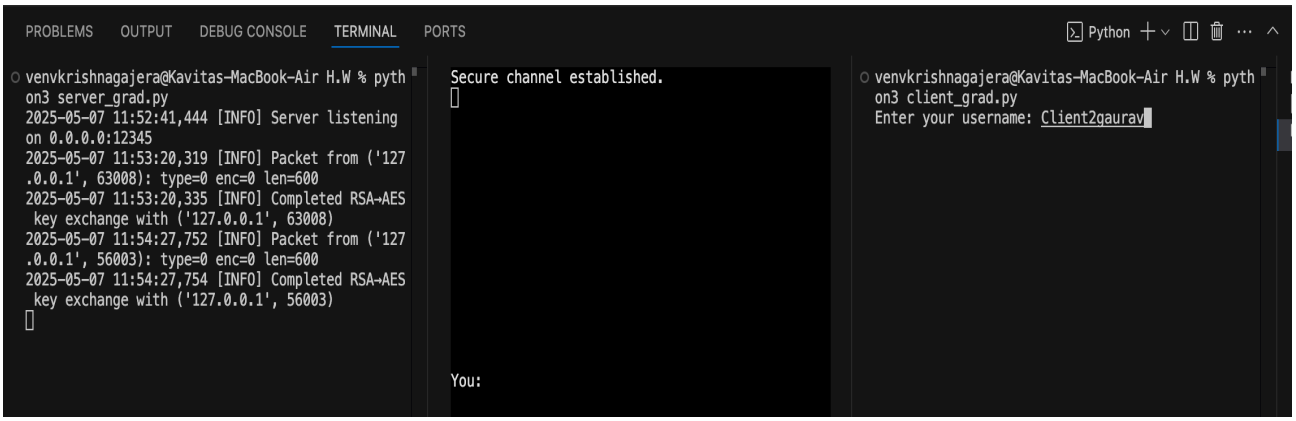


2. ***Client script prompts the user to enter their username:***

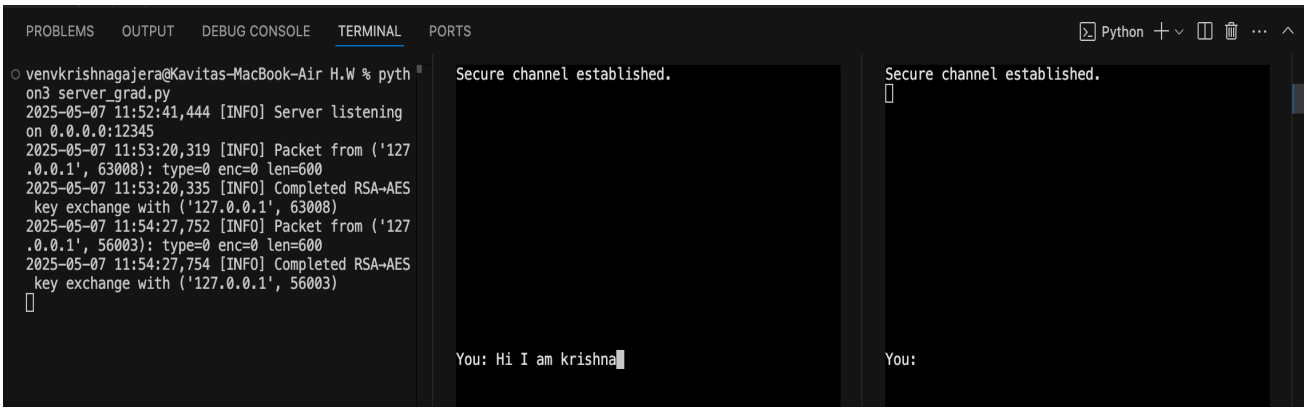
3. ***AES key exchange is completed and secure connection ready***

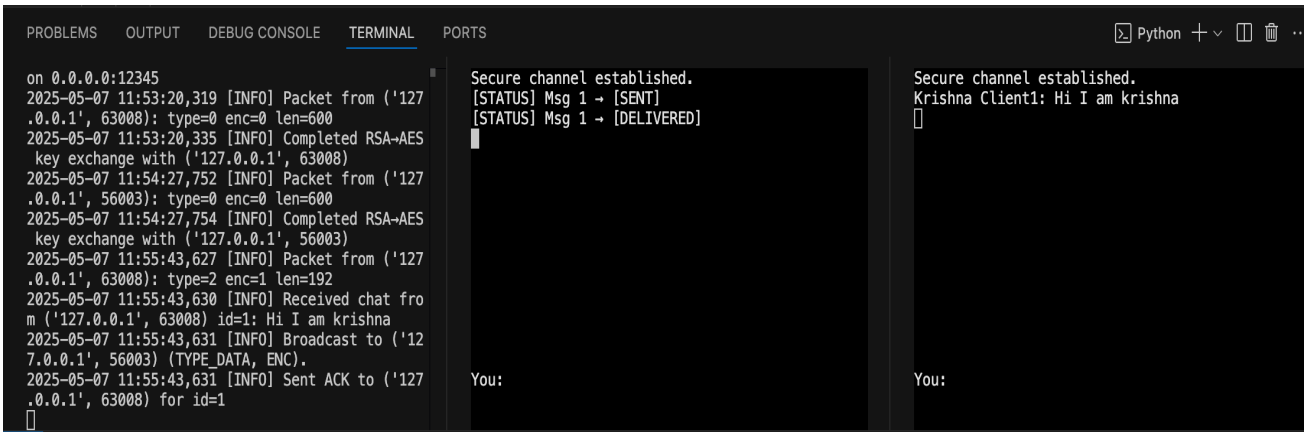


4. ***Multiple Clients Connecting – Another client joins, showing multi-user capability:***

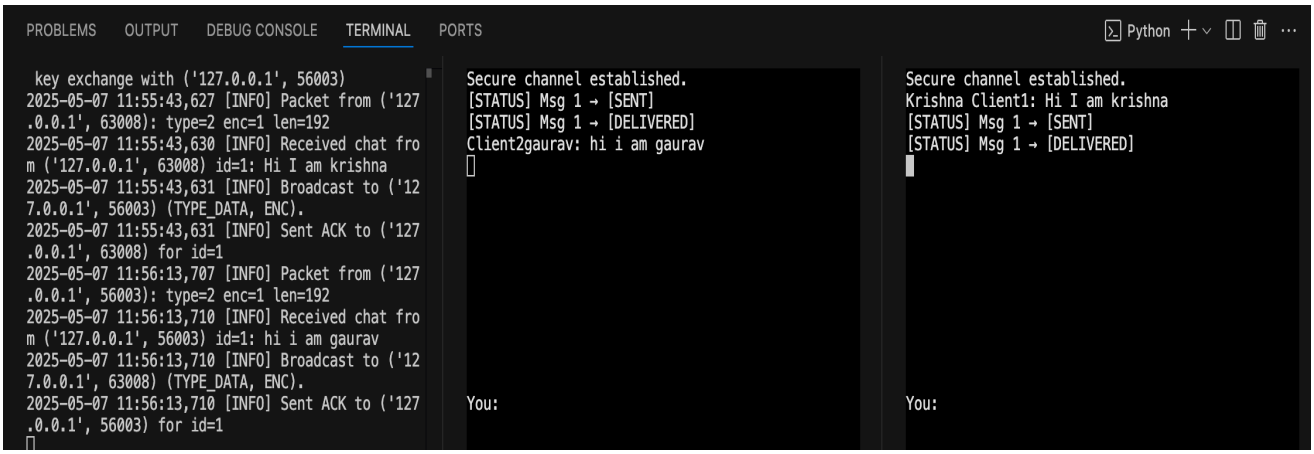


5. ***Message Sent and Delivered – Message from client shows [SENT] and [DELIVERED] status:***

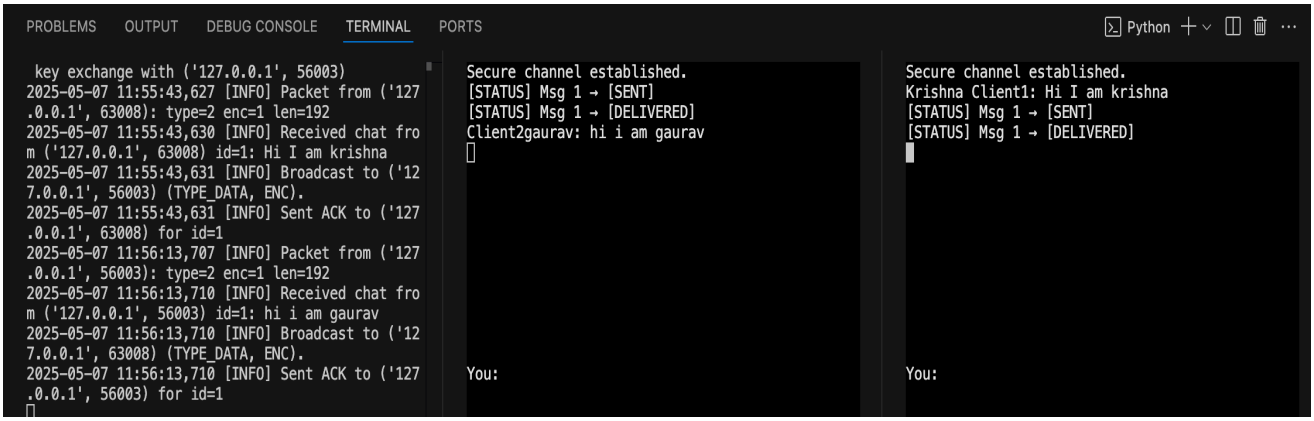




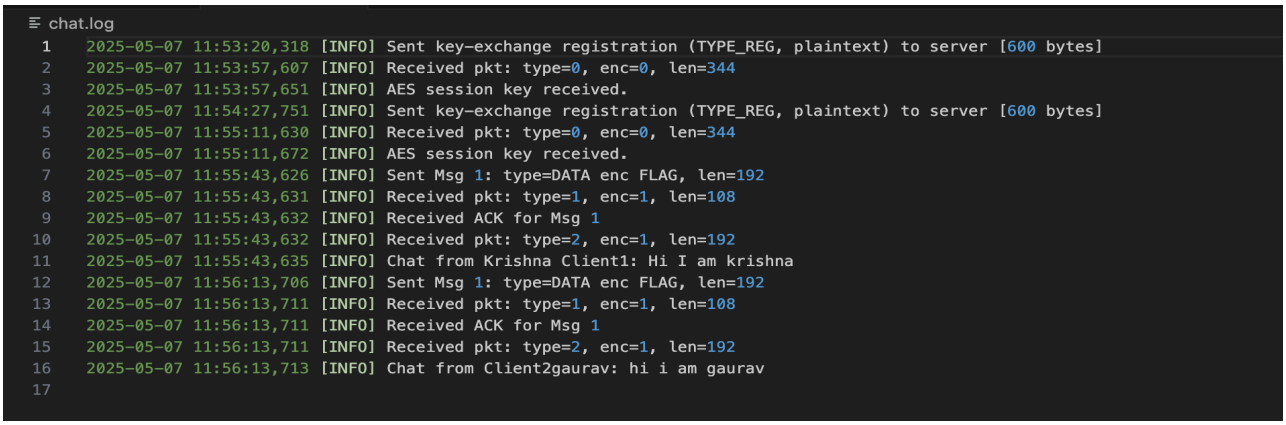
6. ***Client-to-Client Message Delivery – Message received and decrypted by another client:***



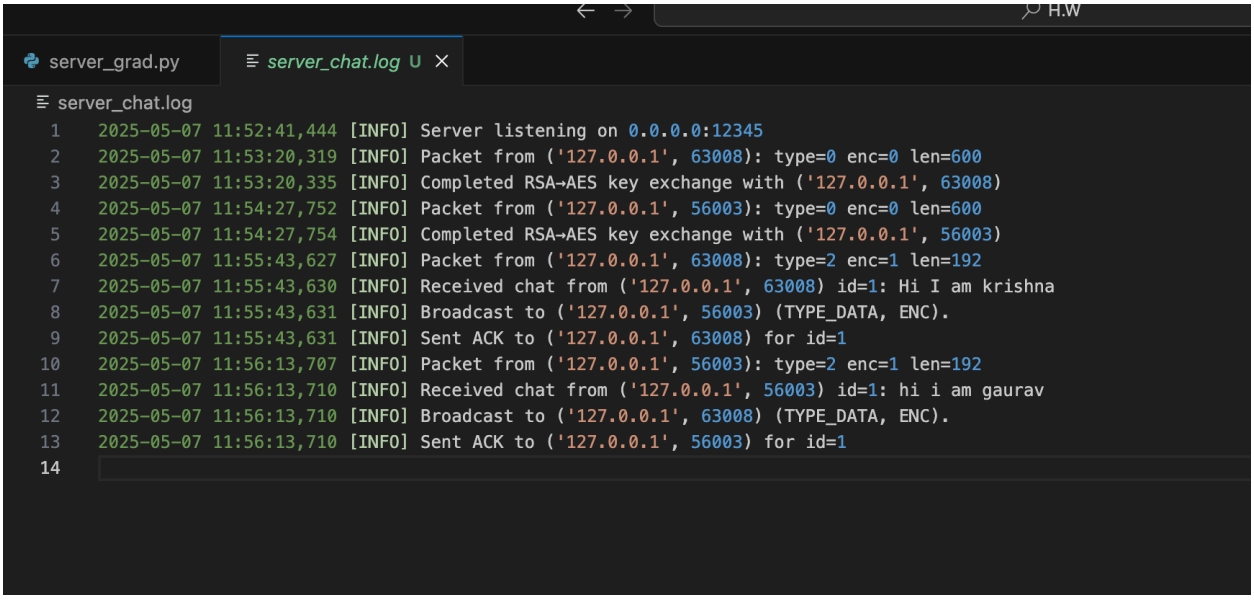
7. ***Two-Way Chat in Action – Messages sent and received between two users:***



8. ***Client log:***



9. ***Server Log:***



Conclusion

This project successfully demonstrates the design and implementation of a secure and interactive UDP-based chat application. It meets all requirements of the assignment, as well as the advanced features expected of graduate students, including message authentication (HMAC), retransmission logic for packet loss recovery, and a terminal-based user interface using curses. By combining RSA for key exchange, AES for encryption, and HMAC for integrity, the system ensures end-to-end message confidentiality and authenticity. The implementation of acknowledgment and retry mechanisms effectively simulates reliable communication over the inherently unreliable UDP protocol. Real-time message status feedback and detailed logging further enhance reliability and user experience.

Through this project, I gained practical experience with secure communication protocols, socket programming and UI development in Python. The system is modular, easy to extend, and demonstrates a solid understanding of both network communication and applied cryptography making it well-suited for educational, experimental, or small-scale secure communication use.