Secure Messaging Application Project Report

1. Project Description

The Secure Messaging App is a peer-to-peer (P2P) desktop application designed to facilitate confidential communication between users using end-to-end encryption (E2EE). It ensures that messages are securely exchanged over the network and securely stored on the local device. The application is built using **Python** and **PySide6** for the GUI, and implements cryptographic security using **RSA**, **AES-GCM**, **HMAC**, and **Double Ratchet Algorithm** principles.

Main Features:

- Encrypted private messaging
- Login and authentication
- Message history stored securely in a local encrypted database
- Easy-to-use, modern graphical user interface

---

2. Detailed Design and Functional Specifications

2.1 System Architecture

- **Frontend (GUI):** PySide6-based desktop UI
- **Backend (Local Database):** SQLite with AES-encrypted message storage
- **Encryption Layer:**
    - **Key Exchange:** RSA 2048-bit
    - **Session Encryption:** AES-256-GCM (symmetric encryption for message body)
    - **Forward Secrecy:** Double Ratchet Algorithm

  - **Integrity Protection:** HMAC-SHA256

2.2 Functional Specifications

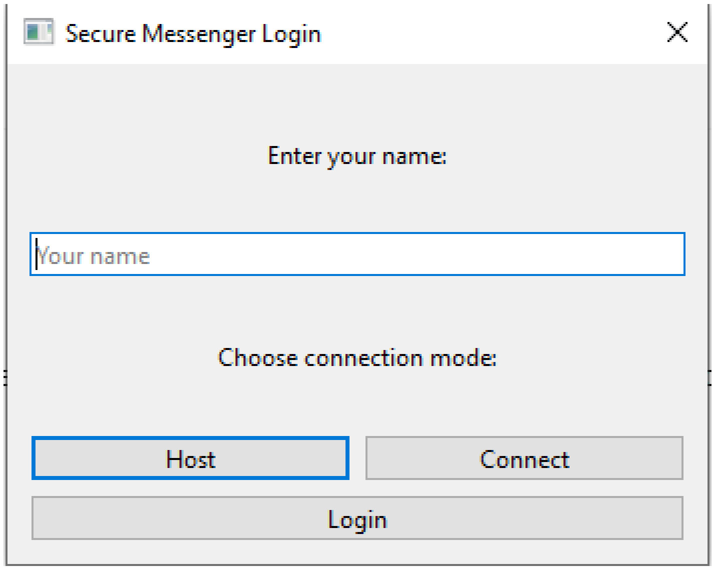
- **User Login:**
  - Simple login with username (no password to focus on P2P encryption)
- **Messaging:**
  - Messages are encrypted before sending.
  - Messages are decrypted after receiving.
- **History:**
  - Stored in a local SQLite database.
  - Messages are encrypted at rest using AES-256.
- **Key Management:**
  - Session keys are refreshed with every message via the Double Ratchet mechanism.

---

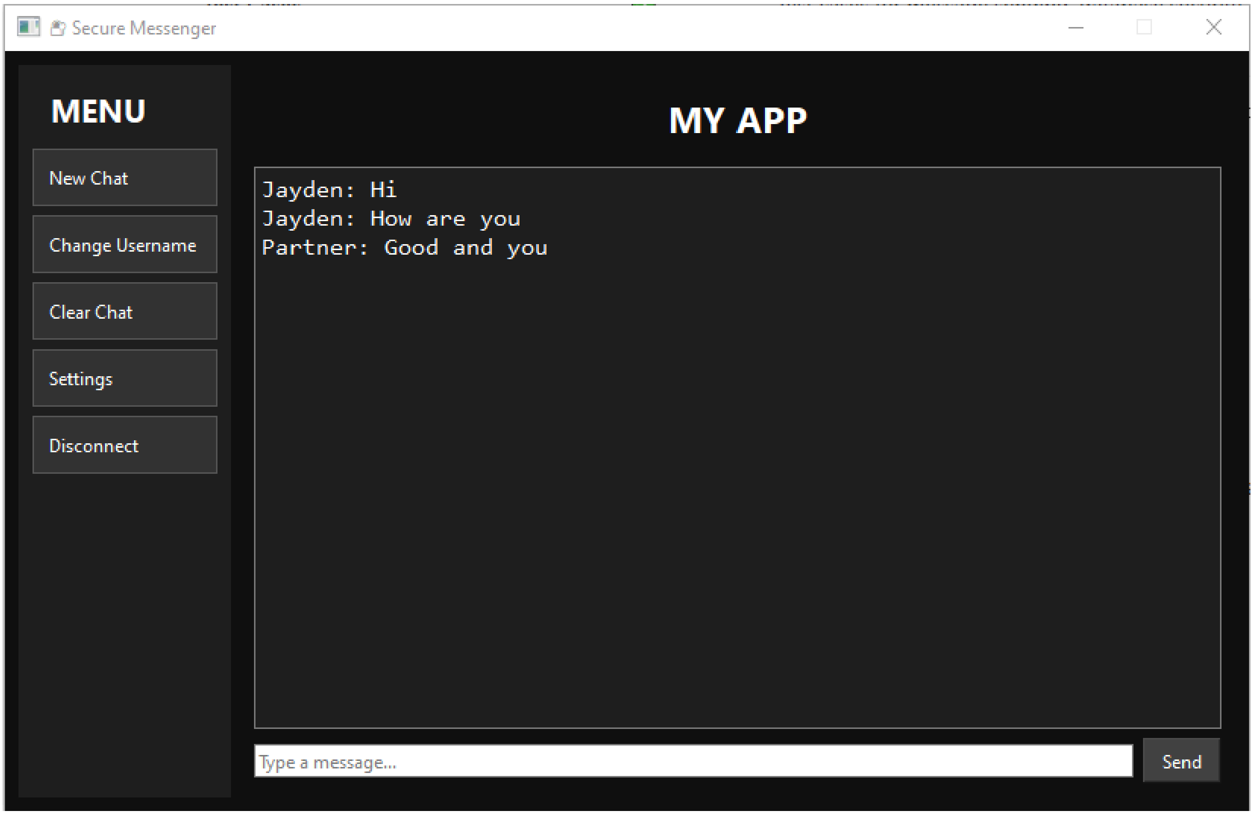
3. Implementation

3.1 Application GUI (Screenshots)

- **Login Screen:** Simple username prompt.



- **Chat Window:** Displays conversation in real-time, includes text box for new messages.



```
Jayden: Hi
Jayden: How are you
Partner: Good and you
```

- **Connection Status:** Displays whether connected to another user.

3.2 Message Encryption

- **RSA (Public/Private Keys):** Exchanged during session setup to securely establish session keys.
- **AES-GCM Encryption:**
  - Messages are encrypted with a unique AES key per session.
  - Nonces are generated per message to ensure uniqueness.
- **Double Ratchet:**
  - Each message rotates the session key.
  - Compromise of one message key does not affect others (forward secrecy).

4. Test Cases

Test Cases	Test Description	Expected Result	Pass/Fail
TC1	Send encrypted message	Message is encrypted and received correctly	
TC2	Database message retrieval	Retrieved messages are decrypted correctly	
TC3	Session key refresh after message	Session key changes after sending each message	
TC4	Unauthorized DB access (simulate)	Messages remain unreadable without AES key	
TC5	Network interception (simulate MITM)	Messages appear as unreadable cipher text	

5. Performance Evaluation

5.1 Communication Channel Security

- **TLS over TCP:** Secures the communication channel.
- **RSA key exchange:** Ensures that session keys are never sent in plaintext.
- **Forward secrecy:** Double Ratchet ensures past/future communications remain secure if keys are compromised.

5.2 Database Storage Security

- **Encryption at Rest:** Messages in the database are unreadable without the AES key.
- **Key Protection:** No private keys or AES keys are stored on disk, only kept in volatile memory.

5.3 Potential Threats Mitigated

Threat	Mitigation Strategy
Man-in-the-middle attacks	TLS + E2EE via RSA key exchange
Database breach	AES-256 encryption at rest
Key compromise	Forward secrecy with Double Ratchet
Message tampering	HMAC integrity verification