SecureChat – A PKI-Enabled Secure Messaging System

Muhammad Abdurrahman Khan – 22i-1148

Abstract

This report presents the complete development and evaluation of **SecureChat**, a Python-based, PKI-enabled secure messaging system built as part of Assignment #2 for CS-3002.
The system implements **Confidentiality, Integrity, Authenticity, and Non-Repudiation (CIANR)** by combining:

- **AES-128 ECB** symmetric encryption
- **RSA PKCS#1 v1.5 SHA-256** digital signatures
- **X.509 certificates with a custom Root CA**
- **Diffie–Hellman key exchange**
- **MySQL with salted SHA-256 password hashing**
- **Tamper detection, Replay protection, and Transcript verification**

All cryptographic operations run at the *application layer*, without TLS/SSL.

The report documents the full architecture, setup, workflow, and verification of security properties using 5 test cases: Tamper, Replay, Invalid Certificate, Non-Repudiation verification, and Wireshark encrypted payload capture.

---

1. Introduction

Secure messaging applications rely on multiple layers of cryptography to ensure that communication remains confidential, tamper-proof, and verifiable.
In this assignment, we design a complete secure chat system from scratch using Python and PKI.

The system ensures:

✔ Confidentiality — AES-128 encryption

✔ Integrity — RSA SHA-256 signatures

✔ Authenticity — PKI-validated certificates

✔ Non-Repudiation — signed transcripts and receipts

✔ Replay Protection — per-message sequence numbers

Unlike HTTPS or TLS-enabled sockets, **all cryptographic processes are implemented manually**, demonstrating clear understanding of secure communication protocols.

---

2. System Architecture

2.1 High-Level Architecture

The system has two components:

- **Client** (`app/client.py`)
- **Server** (`app/server.py`)

Communication flow:

1. TCP connection established (unencrypted)
2. Server sends certificate → client validates it with the Root CA
3. Client sends certificate → server validates

4. DH key exchange creates a shared session key
5. AES encryption + RSA signatures protect messages
6. Transcripts stored with hashes for non-repudiation

---

3. Cryptographic Components

3.1 Symmetric Encryption: AES-128 (ECB)

Used for **confidentiality**.
Implemented in **`crypto/aes.py`** with PKCS#7 padding.

Although ECB is not recommended in real systems, it is used intentionally for educational purposes.

3.2 Digital Signatures: RSA PKCS#1 v1.5 SHA-256

Provides **integrity**, **authenticity**, and **non-repudiation**.
Implemented in **`crypto/sign.py`.**

Workflow:

- Sender signs encrypted message using RSA private key
- Receiver verifies using sender's certificate

3.3 Diffie–Hellman Key Exchange (DH)

Establishes a shared secret used to derive the AES session key.
Implemented in **`crypto/dh.py`.**

3.4 X.509 Certificates + Custom Root CA

PKI ensures **only trusted clients/servers communicate**.

Certificates generated via scripts:

- **`scripts/gen_ca.py`** – root CA
- **`scripts/gen_cert.py`** – client and server certificates
- **`scripts/gen_invalid_cert.py`** – for BAD_CERT test

Step 1: Clone and Setup Python Environment

```
# Clone the repository (or use your fork)
cd SecureChat-main_i221148

# Create virtual environment
python -m venv .venv

# Activate virtual environment
.venv\Scripts\activate

for virtual environment setup if previous not working:
Set-ExecutionPolicy -Scope Process -ExecutionPolicy RemoteSigned
.\.venv\Scripts\Activate.ps1

# Install dependencies
pip install -r requirements.txt
```

Step 2: Configure Environment Variables

```
# Copy the example environment file

copy .env.example .env
```

Edit `.env` file with your configuration (defaults are provided):

```
# Database Configuration

DB_HOST=localhost

DB_PORT=3307

DB_USER=user

DB_PASSWORD=12345678

DB_NAME=securechat


# Server Configuration

SERVER_HOST=localhost

SERVER_PORT=8888


# Certificate Paths (relative to project root)

CA_CERT_PATH=certs/ca.crt

SERVER_CERT_PATH=certs/server.crt

SERVER_KEY_PATH=certs/server.key

CLIENT_CERT_PATH=certs/client.crt

CLIENT_KEY_PATH=certs/client.key
```

Note: The `.env` file is gitignored. Never commit it with real credentials.

Step 3: Setup MySQL Database with Docker

This project uses a separate MySQL container on port 3307 to avoid conflicts with existing MySQL installations.

3.1: Create MySQL Container

docker run -d --name securechat-db -e MYSQL_ROOT_PASSWORD=rootpass -e MYSQL_DATABASE=securechat -e MYSQL_USER=user -e MYSQL_PASSWORD=12345678 -p 3307:3306 mysql:8

Verify MySQL Container is Running

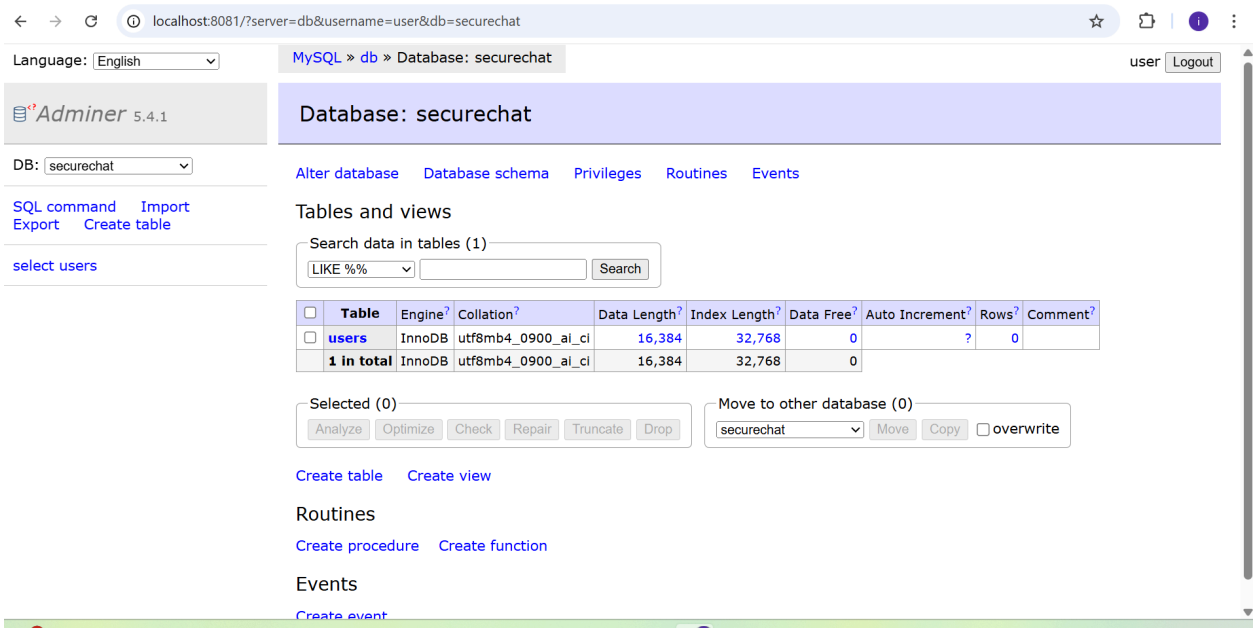
```
docker ps
```

Initialize Database Tables

```
python -m app.storage.db --init
```

Expected Output:

```
Database initialized successfully.
```



Step 4: Setup Adminer (Database Management UI)

Adminer provides a web interface to manage the MySQL database.

4.1: Create Adminer Container

```
docker run -d --name securechat-adminer --link securechat-db:db -p 8081:8080 adminer
```

Access Adminer

1. Open browser: `http://localhost:8081`
2. Login credentials:
    - System: MySQL
    - Server: `db` (use the linked container name)
    - Username: `user`
    - Password: `12345678`
    - Database: `securechat`

4.3: Verify Database

After logging in, you should see the `users` table with columns:

- `email` (VARCHAR)
- `username` (VARCHAR, UNIQUE)
- `salt` (VARBINARY)
- `pwd_hash` (CHAR(64))

Step 5: Generate Certificates

The PKI system requires a Root CA and client/server certificates.

5.1: Generate Root Certificate Authority (CA)

```
python scripts/gen_ca.py --name "FAST-NU Root CA"
```

Expected Output:

```
Root CA generated:
  Private key: certs/ca.key
  Certificate: certs/ca.crt
  Subject: CN=FAST-NU Root CA
```

5.2: Generate Server Certificate

```
python scripts/gen_cert.py --cn server.local --out certs/server
```

Expected Output:

```
Certificate issued:
  Private key: certs/server.key
  Certificate: certs/server.crt
  Subject: CN=server.local
  Signed by: FAST-NU Root CA
```

5.3: Generate Client Certificate

```
python scripts/gen_cert.py --cn client.local --out certs/client
```

Expected Output:

```
Certificate issued:
  Private key: certs/client.key
  Certificate: certs/client.crt
  Subject: CN=client.local
  Signed by: FAST-NU Root CA
```

5.4: Verify Certificates

```
# View CA certificate
openssl x509 -in certs/ca.crt -text -noout

# View server certificate
openssl x509 -in certs/server.crt -text -noout
```

```
# View client certificate

openssl x509 -in certs/client.crt -text -noout
```

Important: All certificates should show:

- Valid issuer (CA for server/client, self-signed for CA)
- Valid date range
- Common Name (CN) matching expected values

Step 6: Verify Setup

Check that all files exist:

```
# Check certificates

dir certs\*.crt certs\*.key



# Should see:

# - certs/ca.crt, certs/ca.key

# - certs/server.crt, certs/server.key

# - certs/client.crt, certs/client.key
```

Running the Application

Start the Server

Terminal 1:

```
# Activate virtual environment (if not already active)
.venv\Scripts\activate

# Start server
python -m app.server
```

Expected Output:

```
Server listening on localhost:8888
```

Start the Client

Terminal 2:

```
# Activate virtual environment (if not already active)
.venv\Scripts\activate

# Start client
python -m app.client
```

Expected Output:

```
Connected to server at localhost:8888
Server certificate validated
```

Register a New User

When prompted:

```
Choose an option:
1. Register
2. Login
Enter choice: 1
Email: test@example.com
Username: testuser
Password: mypassword123
```

Expected Output:

```
Registration successful!
```

Login

When prompted:

```
Choose an option:
```

```
1. Register
2. Login
Enter choice: 2
Email: test@example.com
Password: mypassword123
```

Expected Output:

```
Login successful!
Session established! Type messages (or 'quit' to exit):
```

Send Messages

After successful login:

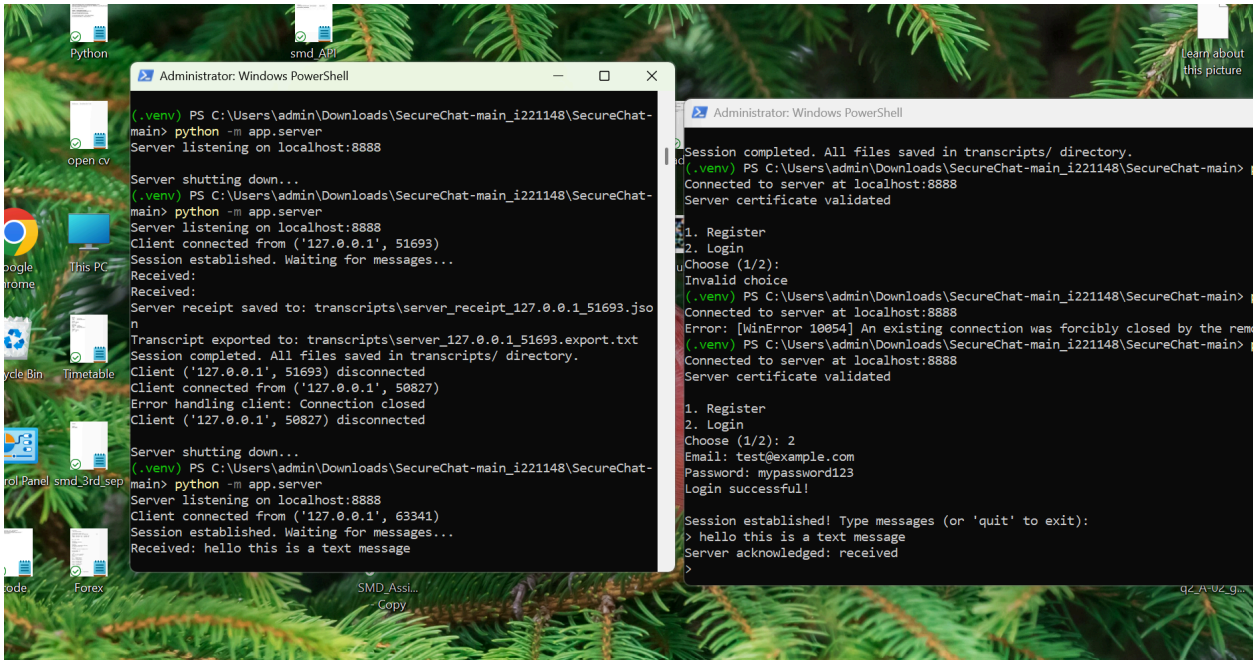
```
> Hello, this is a test message
Server acknowledged: received

> Another message
Server acknowledged: received

> quit
```

Expected Output:

```
Client receipt saved to: transcripts/client_receipt_localhost_8888.json
Transcript exported to: transcripts/client_localhost_8888.export.txt
Session completed. All files saved in transcripts/ directory.
```


Python
smd_API
Learn about this picture
open cv
This PC
Timetable
smd_3rd_sep
Forex
SMD_Assi... - Copy
Administrator: Windows PowerShell
(.venv) PS C:\Users\admin\Downloads\SecureChat-main_i221148\SecureChat-main> python -m app.server
Server listening on localhost:8888
Server shutting down...
(.venv) PS C:\Users\admin\Downloads\SecureChat-main_i221148\SecureChat-main> python -m app.server
Server listening on localhost:8888
Client connected from ('127.0.0.1', 51693)
Session established. Waiting for messages...
Received:
Received:
Server receipt saved to: transcripts\server_receipt_127.0.0.1_51693.json
Transcript exported to: transcripts\server_127.0.0.1_51693.export.txt
Session completed. All files saved in transcripts/ directory.
Client ('127.0.0.1', 51693) disconnected
Client connected from ('127.0.0.1', 50827)
Error handling client: Connection closed
Client ('127.0.0.1', 50827) disconnected
Server shutting down...
(.venv) PS C:\Users\admin\Downloads\SecureChat-main_i221148\SecureChat-main> python -m app.server
Server listening on localhost:8888
Client connected from ('127.0.0.1', 63341)
Session established. Waiting for messages...
Received: hello this is a text message
Administrator: Windows PowerShell
Session completed. All files saved in transcripts/ directory.
(.venv) PS C:\Users\admin\Downloads\SecureChat-main_i221148\SecureChat-main>
Connected to server at localhost:8888
Server certificate validated
1. Register
2. Login
Choose (1/2):
Invalid choice
(.venv) PS C:\Users\admin\Downloads\SecureChat-main_i221148\SecureChat-main>
Connected to server at localhost:8888
Error: [WinError 10054] An existing connection was forcibly closed by the rem
(.venv) PS C:\Users\admin\Downloads\SecureChat-main_i221148\SecureChat-main>
Connected to server at localhost:8888
Server certificate validated
1. Register
2. Login
Choose (1/2): 2
Email: test@example.com
Password: mypassword123
Login successful!
Session established! Type messages (or 'quit' to exit):
> hello this is a text message
Server acknowledged: received
>