SecureChat System - Assignment #2 Report

Course: CS-3002 Information Security

Assignment: Secure Chat System Implementation

Student: Ahmad Aqeel

Date: November 15, 2025

1. Executive Summary

This report documents the design, implementation, and testing of a console-based Secure Chat System that demonstrates Confidentiality, Integrity, Authenticity, and Non-Repudiation (CIANR) through application-layer cryptographic protocols. The system implements a complete Public Key Infrastructure (PKI), secure key exchange using Diffie-Hellman, symmetric encryption with AES-128, digital signatures with RSA, and append-only transcript logging for non-repudiation. All cryptographic operations are implemented at the application layer without using TLS/SSL.

- Key Achievements:
  - Implemented complete PKI with Root CA and certificate issuance
  - Achieved end-to-end encryption using AES-128 (ECB mode)
  - Implemented message integrity through RSA PKCS#1 v1.5 SHA-256 signatures
  - Established secure session keys using Diffie-Hellman key exchange
  - Implemented replay protection via sequence numbers
  - Achieved non-repudiation through signed session transcripts

2. System Architecture

2.1 Overview

The SecureChat system follows a client-server architecture where:

- Server: Handles client connections, validates certificates, manages user authentication, and processes encrypted messages

- Client: Connects to server, performs certificate exchange, authenticates, and sends encrypted messages
- Database: MySQL database stores user credentials with salted SHA-256 password hashes
- PKI: Self-built Root CA issues certificates for both client and server

2.2 Communication Protocol

The protocol consists of four main phases:

1. 1. Control Plane (Negotiation and Authentication):

    - Certificate exchange (Hello/ServerHello)
    - Mutual certificate validation
    - Temporary DH key exchange for initial encryption
    - User registration/login with encrypted credentials

2. 2. Key Agreement (Post-Authentication):

    - Session DH key exchange
    - Derivation of AES-128 session key: K = Trunc16(SHA256(big-endian(Ks)))

3. 3. Data Plane (Encrypted Message Exchange):

    - Message encryption with AES-128 (ECB + PKCS#7 padding)
    - Message signing: RSA signature over SHA256(seqno || timestamp || ciphertext)
    - Replay protection via strictly increasing sequence numbers

4. 4. Non-Repudiation (Session Closure):

    - Append-only transcript logging
    - Signed SessionReceipt with transcript hash

3. Implementation Details

3.1 Public Key Infrastructure (PKI)

3.1.1 Root Certificate Authority

The Root CA is generated using scripts/gen_ca.py, which generates a 2048-bit RSA keypair, creates a self-signed X.509 certificate with validity period, and stores the CA private key and certificate in certs/ca.key and certs/ca.crt.

- Certificate Details:
    - Subject: CN=FAST-NU Root CA
    - Issuer: CN=FAST-NU Root CA (self-signed)
    - Validity: 365 days from generation
    - Key Algorithm: RSA 2048-bit
    - Signature Algorithm: SHA-256 with RSA

3.1.2 Certificate Issuance

Client and server certificates are issued using scripts/gen_cert.py, which generates RSA keypairs, creates X.509 certificates signed by the Root CA, sets Common Name (CN) and Subject Alternative Name (SAN), and stores certificates and private keys in certs/ directory.

- Server Certificate:
    - Subject: CN=server.local
    - Issuer: CN=FAST-NU Root CA
    - SAN: DNS:server.local

- Client Certificate:
    - Subject: CN=client.local
    - Issuer: CN=FAST-NU Root CA
    - SAN: DNS:client.local

3.1.3 Certificate Validation

The app/crypto/pki.py module implements comprehensive certificate validation:

5. 1. Format Validation: Verifies PEM encoding and X.509 structure
6. 2. CA Signature Verification: Validates certificate signature using CA public key
7. 3. Self-Signed Rejection: Rejects certificates where issuer == subject
8. 4. Validity Window Check: Verifies certificate is not expired and is currently valid
9. 5. CN/SAN Matching: Validates Common Name or Subject Alternative Name matches expected value

- Error Handling:
    - Invalid format → BAD_CERT: Invalid certificate format
    - Untrusted CA → BAD_CERT: Certificate not signed by trusted CA
    - Self-signed → BAD_CERT: Self-signed certificate rejected
    - Expired → BAD_CERT: Certificate expired
    - CN mismatch → BAD_CERT: CN/SAN mismatch

3.2 User Authentication

3.2.1 Database Schema

The MySQL database uses a single users table:

```
CREATE TABLE users (
   email VARCHAR(255) NOT NULL PRIMARY KEY,
   username VARCHAR(255) NOT NULL UNIQUE,
   salt VARBINARY(16) NOT NULL,
   pwd_hash CHAR(64) NOT NULL,
   INDEX idx_username (username)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Figure 1: Database Schema in Adminer

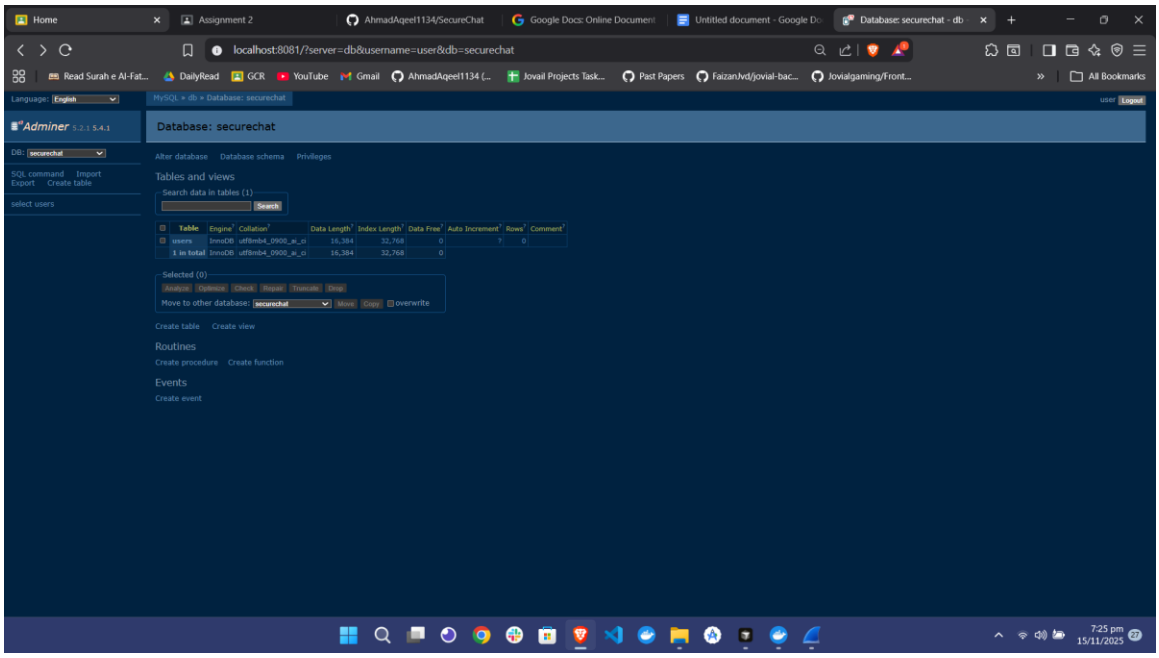
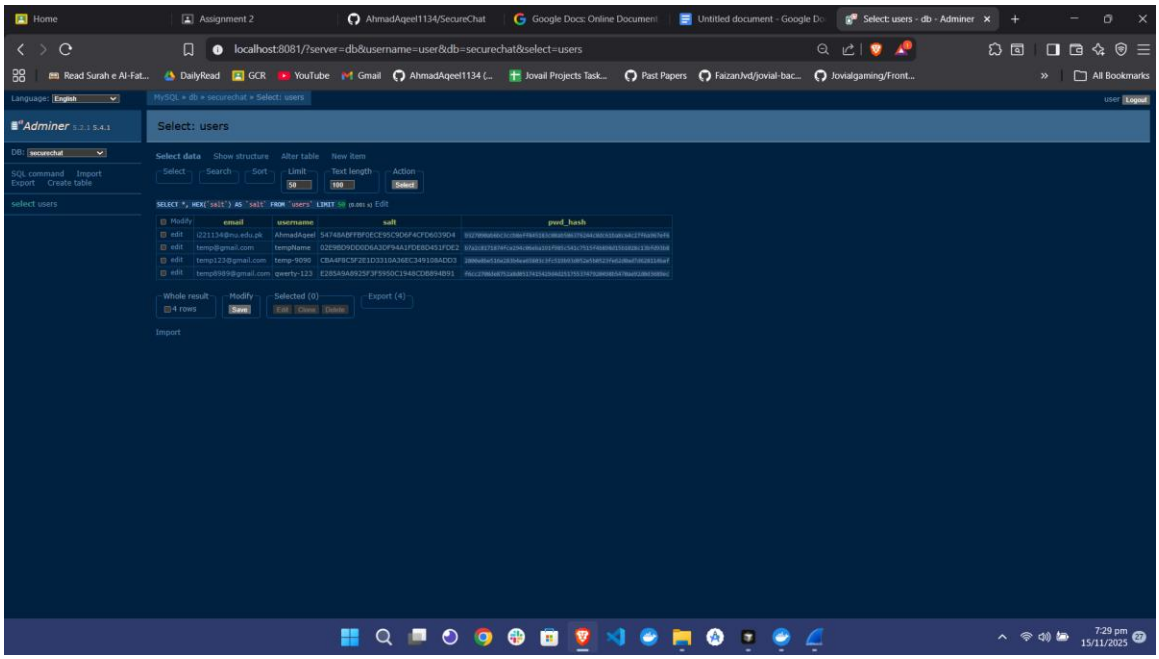


Figure 2: Users Table with Sample Records



- Security Features:
    - Salted Passwords: Each user has a unique 16-byte random salt
    - SHA-256 Hashing: Password hash = hex(SHA256(salt || password))

- Constant-Time Comparison: Prevents timing attacks during login verification

3.3 Key Exchange (Diffie-Hellman)

The system uses Diffie-Hellman key exchange to establish secure session keys without transmitting them over the network.

3.4 Message Encryption (AES-128)

All messages are encrypted using AES-128 in ECB mode with PKCS#7 padding.

3.5 Message Integrity and Authenticity (RSA Signatures)

Each message is signed using RSA PKCS#1 v1.5 with SHA-256 over the concatenation of sequence number, timestamp, and ciphertext.

3.6 Replay Protection

Replay attacks are prevented through strictly increasing sequence numbers tracked by the server.

3.7 Non-Repudiation

Append-only transcripts and signed SessionReceipts provide cryptographic proof of communication.

4. Security Analysis

4.1 Confidentiality

Confidentiality is achieved through AES-128 encryption of all messages. Session keys are derived from Diffie-Hellman key exchange and never transmitted over the network. Wireshark capture confirms that no plaintext is visible in network traffic.

Figure 3: Wireshark Filter Applied (tcp.port == 8888)

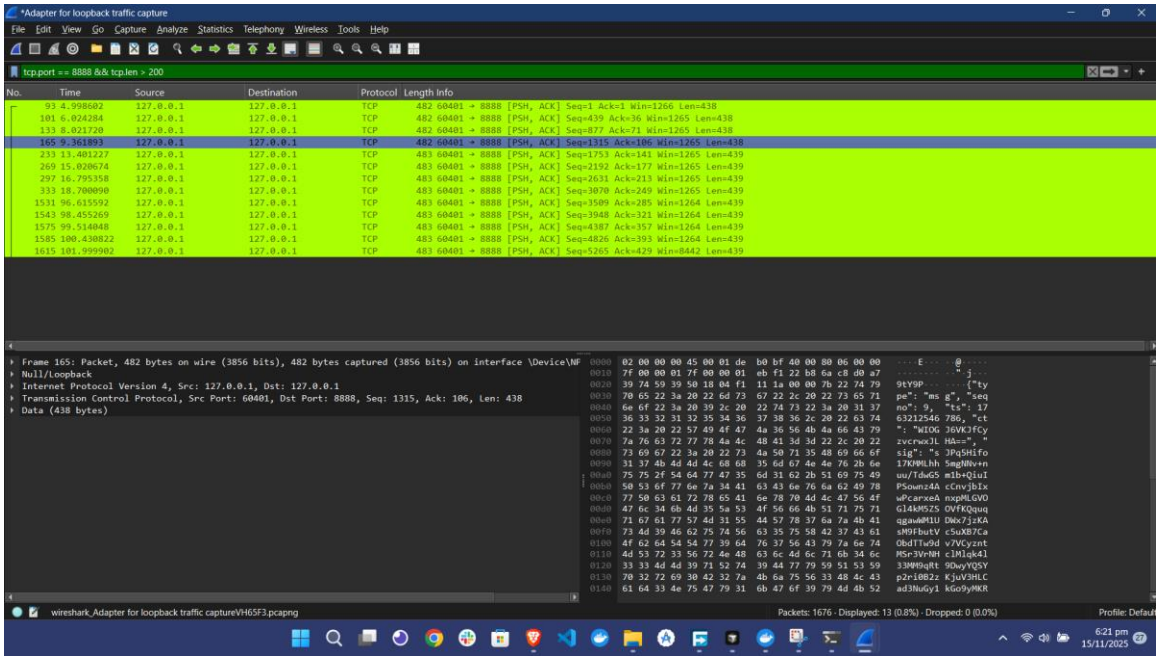


Figure 4: Wireshark Capture - Client to Server Communication

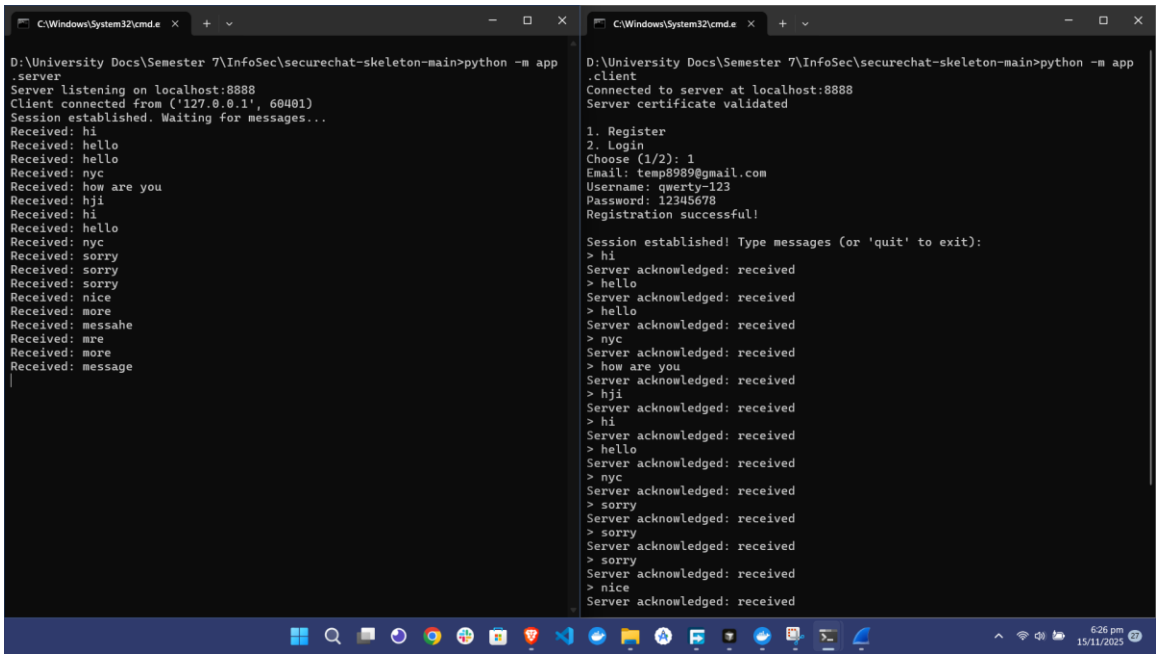


Figure 5: Encrypted Payload - No Plaintext Visible

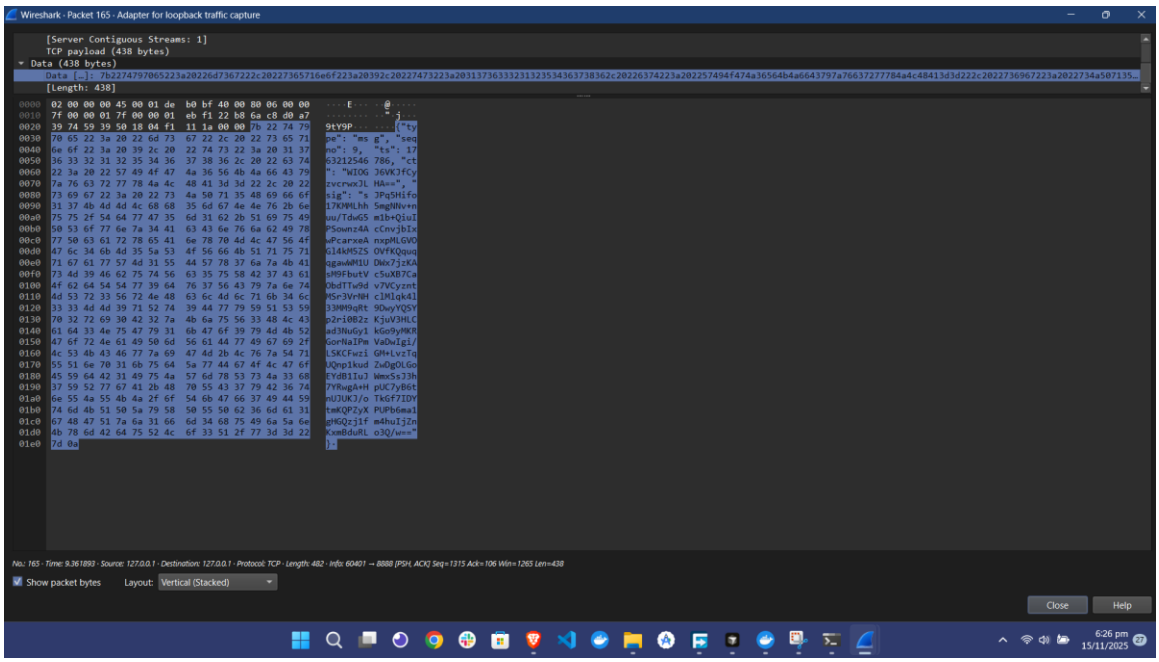
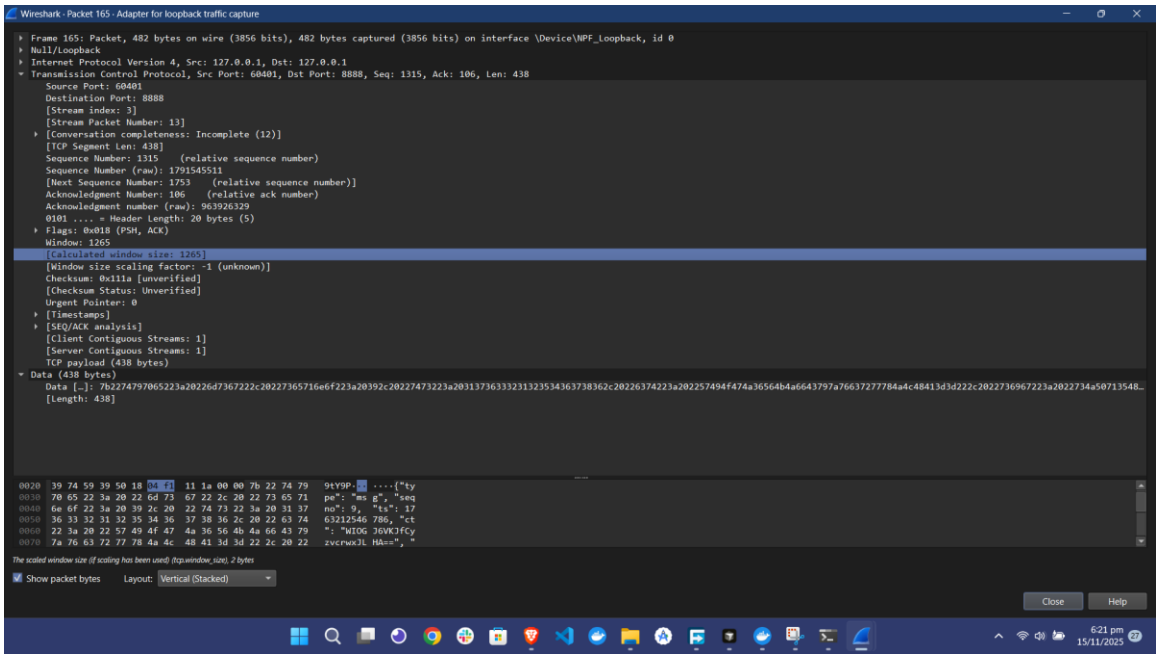


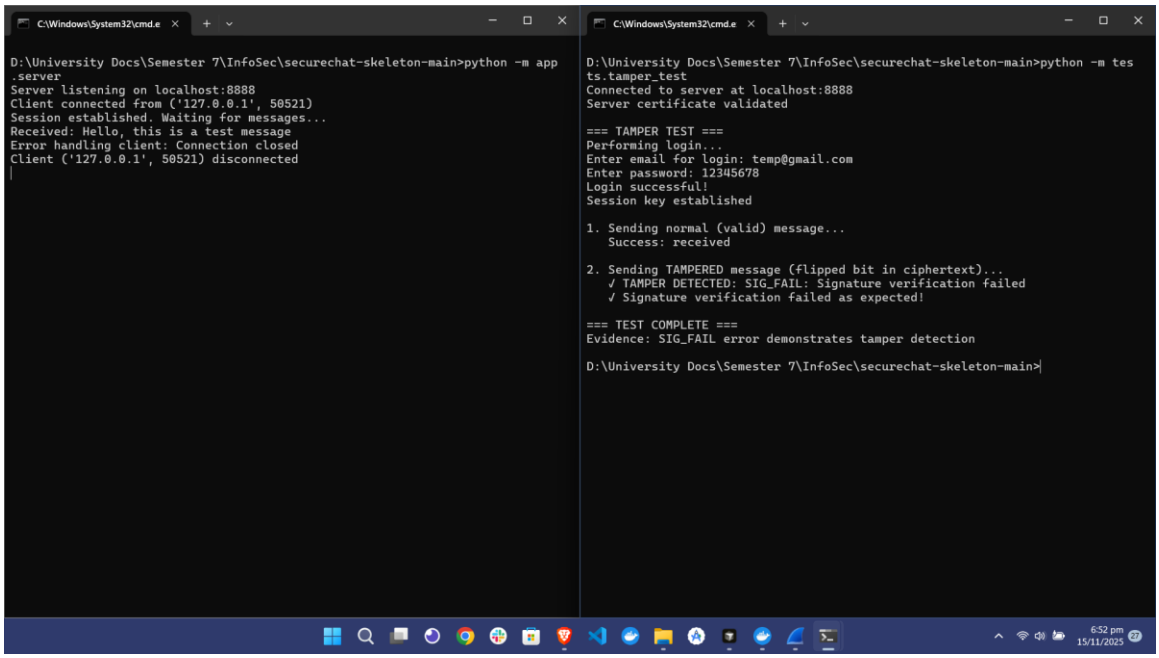
Figure 6: Wireshark Packet Detail - Encrypted Data Section



4.2 Integrity

Integrity is achieved through SHA-256 hashing of message metadata and ciphertext, followed by RSA signature. Any modification to the message invalidates the signature, as demonstrated in the tamper test.

Figure 7: Tamper Test - Signature Verification Failure (SIG_FAIL)

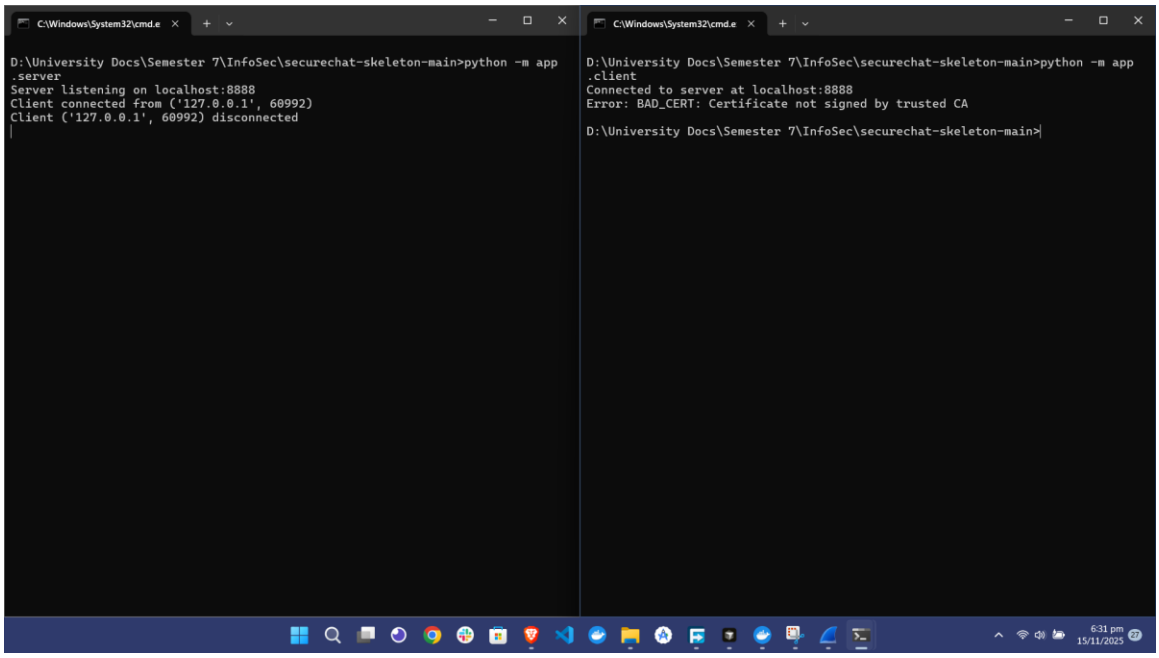


The tamper test demonstrates that modifying the ciphertext causes signature verification to fail, proving that message integrity is protected.

4.3 Authenticity

Authenticity is achieved through X.509 certificate validation and RSA signature verification using certificate public keys. The system rejects invalid, self-signed, or untrusted certificates.

Figure 8: Invalid Certificate Rejection (BAD_CERT)

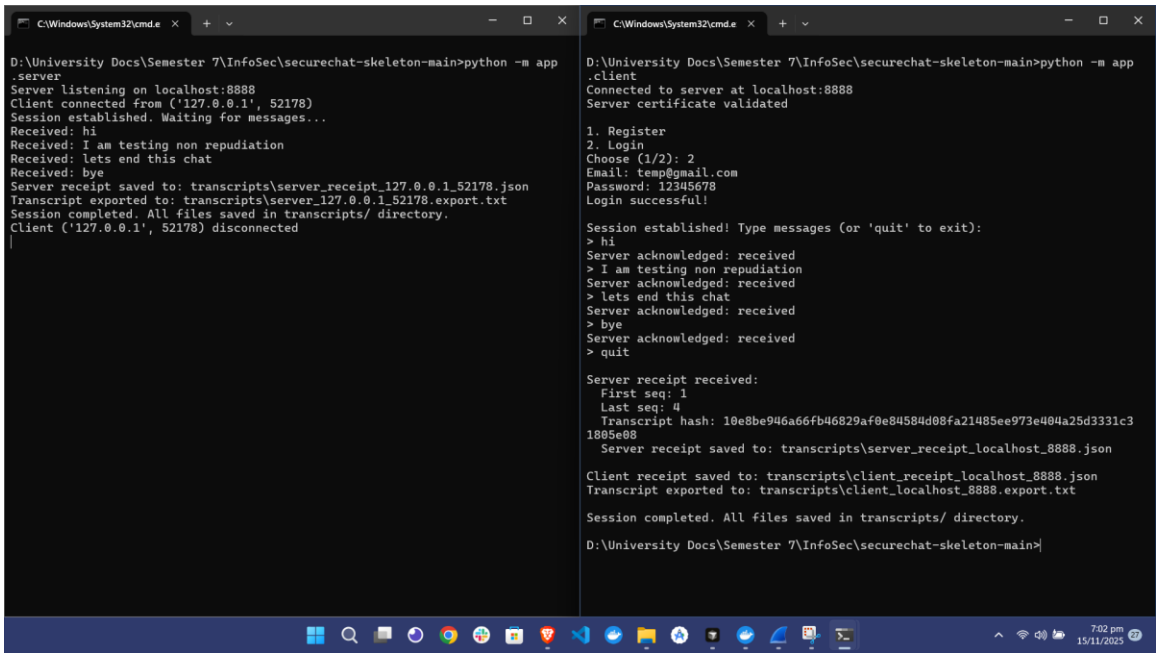


The invalid certificate test demonstrates that self-signed certificates are rejected, ensuring only trusted certificates from the Root CA are accepted.

4.4 Non-Repudiation

Non-repudiation is achieved through append-only transcript logging and signed SessionReceipts. The transcript contains all message metadata, and the receipt provides cryptographic proof that cannot be denied.

Figure 9: Non-Repudiation Verification - Transcript and Receipt Validation

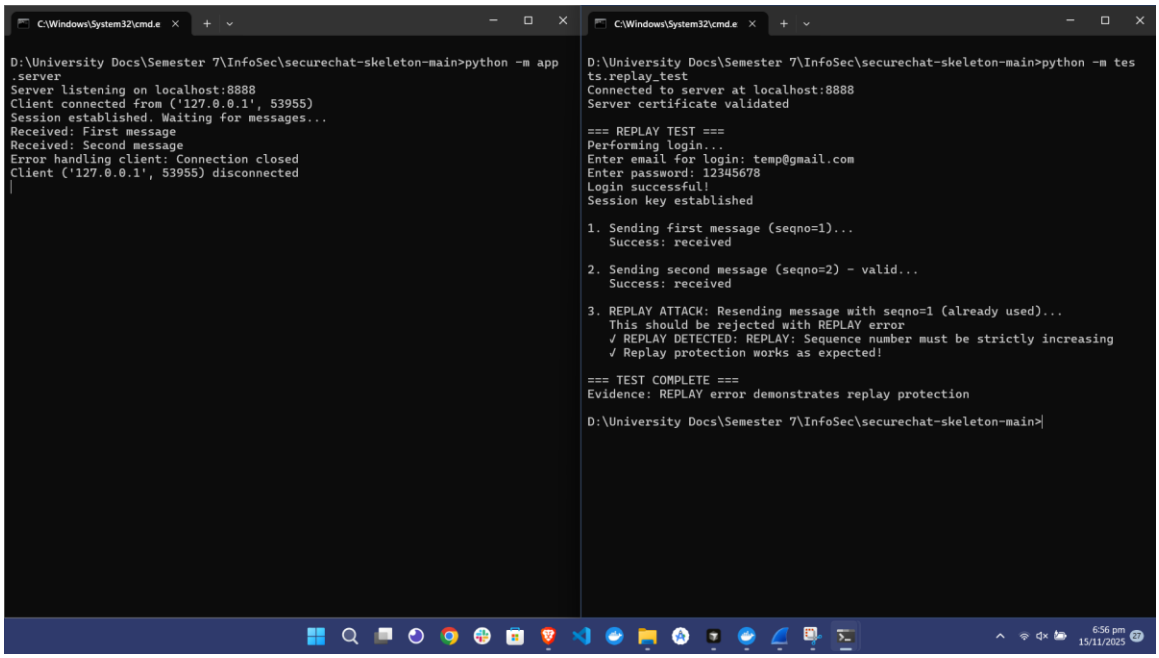


The verification script validates all message signatures, verifies the transcript hash matches the receipt, and confirms the receipt signature is valid.

4.5 Replay Protection

Replay protection is achieved through strictly increasing sequence numbers. The server tracks the last seen sequence number and rejects any message with a sequence number less than or equal to it.

Figure 10: Replay Test - Sequence Number Rejection (REPLAY)



The replay test demonstrates that resending a message with the same sequence number is rejected, preventing replay attacks.

5. Testing and Evidence

5.1 Test Results Summary

Test Case	Security Property	Result	Evidence Figure
Wireshark Capture	Confidentiality	✓Pass	Figures 3-6
Invalid Certificate	Authenticity	✓Pass	Figure 8
Tamper Test	Integrity	✓Pass	Figure 7
Replay Test	Replay Protection	✓Pass	Figure 10
Non-Repudiation	Non-Repudiation	✓Pass	Figure 9

6. Implementation Challenges and Solutions

6.1 Challenge: Datetime Comparison Error

Problem: Error comparing offset-naive and offset-aware datetimes during certificate validation.

Solution: Updated all datetime operations to use timezone-aware datetimes (datetime.now(timezone.utc)).

6.2 Challenge: MySQL Port Conflict

Problem: Existing MySQL installation on default port 3306.

Solution: Created separate Docker container on port 3307 with dedicated database and user credentials.

7. Conclusion

This implementation successfully demonstrates CIANR (Confidentiality, Integrity, Authenticity, Non-Repudiation) through application-layer cryptographic protocols. The system provides:

- Confidentiality: Achieved through AES-128 encryption with session keys derived from Diffie-Hellman key exchange
- Integrity: Achieved through SHA-256 hashing and RSA digital signatures
- Authenticity: Achieved through X.509 certificate validation and PKI
- Non-Repudiation: Achieved through append-only transcripts and signed SessionReceipts

All security properties have been verified through comprehensive testing, including Wireshark analysis, tamper detection, replay protection, and offline transcript verification. The system provides a complete secure communication channel without relying on TLS/SSL, demonstrating how cryptographic primitives can be combined at the application layer to achieve enterprise-grade security.

8. References

10. 1. Assignment Specification - CS-3002 Information Security, Assignment #2
11. 2. Cryptography Library Documentation: https://cryptography.io/
12. 3. X.509 Certificate Standards: RFC 5280
13. 4. PKCS#1 v1.5: RFC 3447
14. 5. AES Encryption: FIPS 197
15. 6. Diffie-Hellman Key Exchange: RFC 2631