SecureChat Project Report
Part 1: Technical Report

Ibraheem Farrukh
Student ID: 22i-1111

December 4, 2025

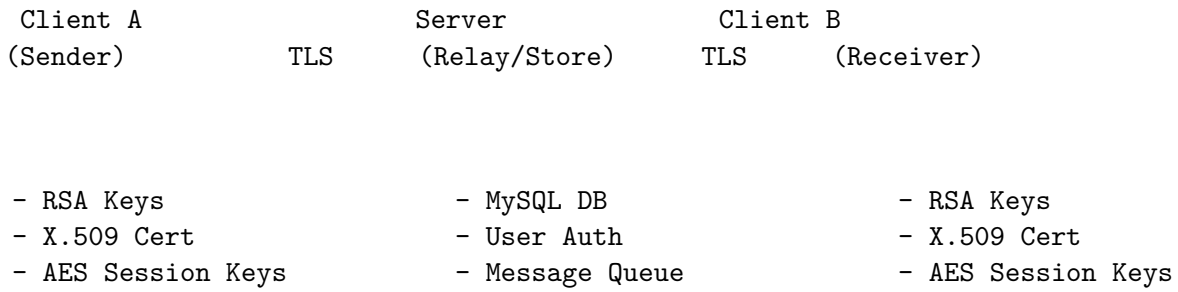
Contents

1 Project Overview

SecureChat is a secure messaging application implementing end-to-end encrypted communication between clients via a central server. It demonstrates cryptographic security principles including PKI, hybrid encryption, digital signatures, and protection against common attacks.

2 Architecture

The system utilizes a client-server model where the server acts as a relay and storage mechanism, but cannot read message content due to end-to-end encryption.



3 Core Components

Component	Location	Purpose
PKI Module	`pki.py`	Certificate generation, validation, CA management
Encryption	`app/crypto/encryption.py`	AES-256-GCM symmetric encryption
Signatures	`app/crypto/signatures.py`	RSA-PSS digital signatures
Key Exchange	`app/crypto/key_exchange.py`	Hybrid RSA+AES key exchange
Server	`server.py`	Message routing, authentication, storage
Client	`client.py`	User interface, message handling
Database	`db.py`	MySQL persistence layer

Table 1: System Component Breakdown

4 Security Mechanisms

4.1 Public Key Infrastructure (PKI)

- **Certificate Authority (CA):** Self-signed root certificate signs all client/server certs.
- **X.509 Certificates:** RSA-2048 keys with SHA-256 signatures.
- **Certificate Chain Validation:** Clients verify server cert; server verifies client certs.

4.2 Hybrid Encryption (Confidentiality)

The encryption flow ensures message confidentiality:

1. Generate random AES-256 session key.
2. Encrypt message with AES-256-GCM (authenticated encryption).
3. Encrypt AES key with recipient's RSA public key.
4. Send: `[RSA-encrypted-key] + [AES-encrypted-message] + [GCM-tag]`

4.3 Digital Signatures (Integrity & Authentication)

The signing flow ensures non-repudiation and integrity:

1. Hash message content with SHA-256.
2. Sign hash with sender's RSA private key (PSS padding).
3. Attach signature to message.
4. Recipient verifies using sender's public key.

4.4 Anti-Replay Protection

- **Nonces:** Unique random values per message.
- **Timestamps:** Messages expire after configured window.
- **Sequence Numbers:** Detect out-of-order/duplicate messages.

5 Message Protocol

The following JSON structure defines the payload exchanged between clients:

```
{
  "type": "message",
  "from": "alice",
  "to": "bob",
  "timestamp": 1733356800,
  "nonce": "a1b2c3d4e5f6...",
  "sequence": 42,
  "encrypted_key": "base64...",
  "ciphertext": "base64...",
  "signature": "base64...",
  "iv": "base64...",
  "tag": "base64..."
}
```

6 Database Schema

Table	Purpose
`users`	User credentials (hashed passwords), public keys
`messages`	Encrypted message storage with metadata
`sessions`	Active session tracking
`nonces`	Used nonces for replay prevention

Table 2: MySQL Database Schema