Assignment A02: Secure Chat Implementation Report

Student Roll Number	22i-1281
Student Full Name	Aisha Siddiqa
Date	2025-11-17
Assignment	A02 - Secure Chat Demonstration

1. Introduction and Project Goals

This report details the design and implementation of a secure client-server chat application developed for Assignment A02. The primary objective was to build a secure, practical cryptographic system from first principles, demonstrating a comprehensive understanding of core security principles by rigorously achieving **CIANR** for all user communications:

- **Confidentiality:** Ensuring data is unreadable to unauthorized third parties.
- **Integrity:** Ensuring data has not been altered in transit.
- **Authenticity:** Verifying the identity of the communicating parties.
- **Non-Repudiation:** Providing undeniable, cryptographic proof of the conversation's content and participants.

2. System Architecture and Cryptographic Primitives

The application follows a modular client-server model, utilizing standard and robust cryptographic primitives to establish trust and secure communication channels without relying on high-level security libraries like TLS/SSL.

2.1 Cryptographic Stack

The system employs a hybrid cryptographic approach, with modules dedicated to each primitive:

Function	Primitive Used	Module	Role in Protocol
Authentication	RSA & X.509 Certificates	app/crypto/pki.py	Validating identity and establishing a chain of trust via a trusted CA.

Key Agreement	Diffie-Hellman (DH)	app/crypto/dh.py	Establishing a temporary session key for **Forward Secrecy**.
Symmetric Encryption	AES-128 in CBC Mode	app/crypto/aes.py	Ensuring **Confidentiality** of message contents and credentials.
Integrity & Signing	RSA with SHA-256 Digest	app/crypto/sign.py	Generating and verifying digital signatures for **Integrity, Authenticity**, and **Non-Repudiation**.

2.2 Modular Architecture

The core logic is divided into the following key packages:

Module	**Description**
app/client.py & app/server.py	Main entry points managing user interaction, connection handling, and protocol orchestration.
app/common/utils.py	Provides general utilities such as reliable network reception (recv_exact) and base64 encoding.
app/crypto/	Contains dedicated classes for PKI validation, DH key exchange, AES operations, and digital signing.
app/storage/transcript.py	Responsible for maintaining the append-only log of all session metadata and generating the final, signed **Session Receipt**.

3. Secure Protocol Implementation (4 Phases)

The communication protocol is executed in four distinct, serialized phases:

3.1. Phase 1: Control Plane (Authentication & Negotiation)

This phase establishes a trusted, temporary channel for secure login credential exchange:

1. **Mutual Authentication:** Client and server exchange X.509 certificates and use pki.validate_certificate to confirm issuance by the trusted CA.
2. **Ephemeral Key Exchange (Initial):** A temporary AES key is established using Diffie-Hellman, only to secure the transit of login credentials.
3. **Secure Credential Transit:** The client sends credentials encrypted with the temporary key.

3.2. Phase 2: Key Agreement (Perfect Forward Secrecy)

Immediately after successful login verification, the Phase 1 temporary key is discarded.

1. **New Session Key:** A second, entirely new Diffie-Hellman exchange is performed to generate a fresh, long-term **chat session key**. This step ensures **Perfect Forward Secrecy (PFS)**, meaning compromise of the long-term RSA key will not affect the confidentiality of this session's messages.

3.3. Phase 3: Data Plane (Encrypted & Signed Messaging)

All subsequent chat messages are protected with multiple layers of security to achieve confidentiality, integrity, and authenticity:

1. **Confidentiality:** The message is encrypted using the unique chat session key via aes.encrypt.
2. **Integrity & Authenticity:** A digital signature is generated over a digest of the sequence number, timestamp, and ciphertext using the sender's private RSA key. This signature is verified upon receipt.
3. **Freshness:** Each message includes a strictly increasing sequence number (seqno). The receiver validates this number, rejecting out-of-order or repeated values to prevent **replay attacks**.

3.4. Phase 4: Tear Down (Session Non-Repudiation)

When the session concludes, an undeniable audit trail is generated:

1. **Transcript Generation:** Both parties maintain an identical, in-memory, append-only transcript log of all message metadata.
2. **Session Receipt:** Each party computes a SHA-256 hash of their final transcript and signs this hash with their private RSA key, generating a **Session Receipt**. These receipts are exchanged and validated, providing mutual, cryptographically-bound proof of the entire

conversation for **Non-Repudiation**.

4. PKI Management and Certificate Evidence

The system's trust model is built on a custom CA. Server and client certificates are issued by this CA.

The public details of the server's certificate confirm correct issuance by the trusted root:

Field	Value
Issuer	CN=SecureChat Root CA
Subject	CN=localhost
Validity	Nov 9 13:45:57 2025 GMT to Nov 9 13:45:57 2026 GMT
Key Size	2048 bit RSA

The full inspection of the certificate confirms the correct use of sha256WithRSAEncryption for the signature algorithm.

5. Conclusion

The implementation successfully achieves the CIANR security goals by composing robust cryptographic primitives into a 4-phase application-layer protocol. The system ensures confidentiality, integrity, and authenticity in real-time messaging, culminating in a powerful non-repudiation mechanism via the signed session receipt. Test evidence is submitted separately to confirm the resilience of all implemented security features.