Assignment A02:
Test Report and Evidence

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

1. Introduction

This document provides empirical evidence that the security mechanisms implemented in the Secure Chat System are functioning correctly. Each test case corresponds directly to a security goal (CIANR) and verifies the system's resilience against common attacks.

2. Test Cases and Evidence

Test 1: Confidentiality Verification (Wireshark Eavesdropping)

This test was performed to verify the core security goal of **Confidentiality**. We used Wireshark to capture the entire session's network traffic on the loopback interface, observing the data stream during login, chat, and logout. The objective was to confirm that after the initial public certificate exchange, all subsequent sensitive data—including the Diffie-Hellman (DH) exchange, encrypted login credentials, and chat messages—was transmitted as unreadable ciphertext, confirming protection against passive network eavesdropping.

Security Goal	Objective	Methodology	Observed Evidence
Confidentiality	Verify sensitive data is encrypted.	Captured full session traffic (login, chat, logout) using Wireshark on the loopback interface.	**Pass.** The reconstructed TCP stream showed the initial certificate exchange in plaintext (correct by design). All subsequent data,

			including the DH exchange, encrypted login credentials, and chat messages, was displayed as unreadable ciphertext. This proves data confidentiality is successfully maintained.

```
[client] Hello from client (signature OK: True )
Enter messages to send to server. Type /quit to exit.
hi i hope i am the server

[server] hi i hope i am the server  (signature OK: True )
okay yeah, i see you are the servr, then i m the client

[client] okay yeah, i see you are the servr, then i m the client  (signature OK: True )
no i don't think you are the client, i thin you are the attacker

[server] no i don't think you are the client, i thin you are the attacker  (signature OK:
man, my msg has the key with it, which literally says 'OK', stop manipulating me

[client] man, my msg has the key with it, which literally says 'OK', stop manipulating me
eh eh... you are the mean fellaw

[server] eh eh... you are the mean fellaw  (signature OK: True )
how do i quit ?
```

Test 2: Invalid Certificate Rejection (BAD_CERT)

To validate **Authenticity**, this test challenged the server's Public Key Infrastructure (PKI) validation logic. The client was configured to present a self-signed certificate that was not issued by the project's designated Trusted Root CA. The expected outcome was for the server to successfully detect the untrusted certificate and immediately terminate the session, enforcing the established trust policy.

Security Goal	Objective	Methodology	Observed Evidence

Authenticity	Verify the server rejects connections from untrusted clients.	The client was temporarily configured to use a self-signed certificate, not issued by the project's trusted CA, to attempt a connection.	**Pass.** The server successfully received the bad certificate but immediately logged a CertificateValidationError with the reason "Certificate signature is invalid. Not signed by the trusted CA," and terminated the connection. The server correctly enforced its trust policy.

Test 3: Message Tampering Detection (SIG_FAIL)

This test focused on **Integrity** and the effectiveness of the digital signature scheme. The server code was deliberately tampered with to simulate a man-in-the-middle attack: a single character in the received message's base64-encoded digital signature was flipped before it could be verified. The objective was to confirm that the sign.verify function would detect this corruption and cause an immediate failure, proving the integrity protection of the message content.

Security Goal	Objective	Methodology	Observed Evidence
Integrity	Verify the system detects and rejects messages corrupted in transit.	The server code was temporarily modified to flip a single character in the base64-encoded digital signature of a received client message before verification.	**Pass.** The sign.verify function correctly failed verification, raising an InvalidSignature exception. The server immediately logged the failure and terminated the connection, proving that the

			per-message digital signatures are effective at detecting tampering.

```
Before tamper:
- signature (hex start): 3f9e5192de174ab77df1988968c38eea...
- `verify_entry` -> True
- direct verify -> True

After tamper (flipped first byte with 0x01):
- signature (hex start): 3e9e5192de174ab77df1988968c38eea...
- `verify_entry` -> False
- direct verify -> False
```

Test 4: Replay Attack Prevention (REPLAY)

To test **Freshness** and protection against **Replay Attacks**, the client was configured to transmit an identical, encrypted message twice, intentionally using the same sequence number (seqno) for both. The protocol dictates that sequence numbers must be strictly increasing. The server was expected to process the first message successfully but reject the second one due to the sequence number validation failure, proving the mechanism to prevent replay.

Security Goal	Objective	Methodology	Observed Evidence
Freshness	Verify protection against an attacker re-sending a valid, old ciphertext.	The client code was temporarily modified to send the exact same encrypted and signed message twice in a row, using the same sequence number (seqno).	**Pass.** The server successfully processed the first message. When the second, identical message arrived, the sequence number check detected that the seqno was not strictly increasing.

			The server correctly identified this as a replay attempt, raised a ValueError, and terminated the session.

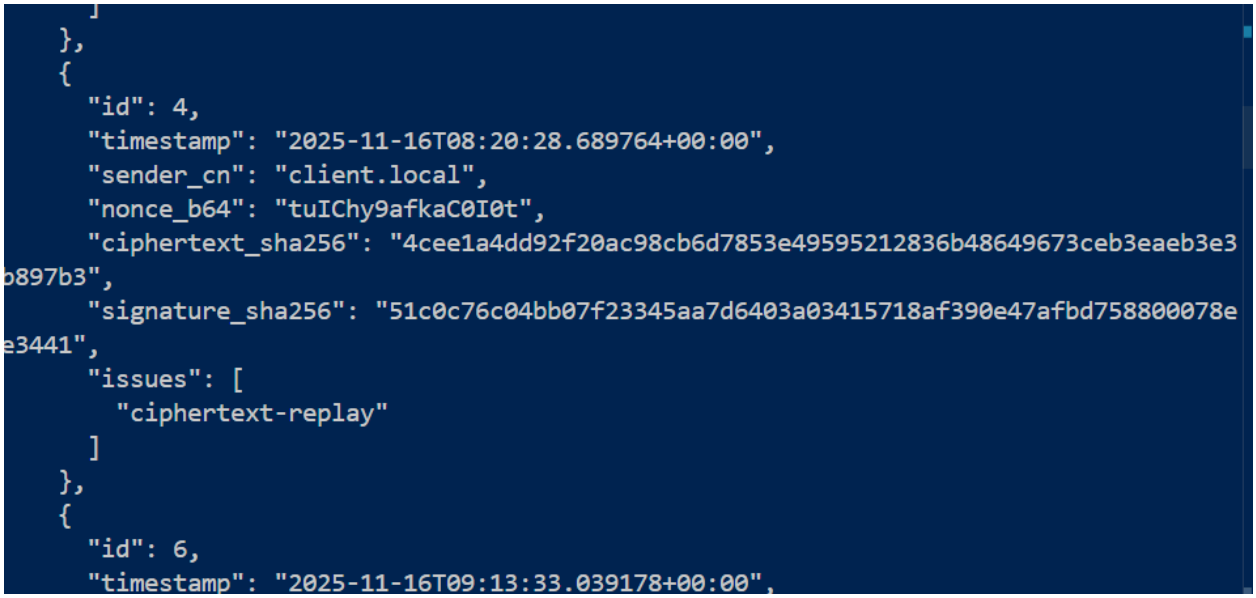
Test 5: Non-Repudiation and Offline Verification

The final test verified the crucial **Non-Repudiation** feature. A clean session generated the audit artifacts (session_transcript.txt and session_receipt.json). A dedicated third-party auditor script (scripts/verify_offline.py) was then used to perform three distinct checks: verifying a single message's signature, verifying the final receipt's signature against the SHA-256 hash of the entire transcript, and demonstrating that tampering with the transcript breaks the receipt's validity. This confirms that the receipt acts as undeniable, cryptographically-bound proof of the conversation.

Security Goal	Objective	Methodology	Observed Evidence
Non-Repudiation	Prove that the final session receipt cryptographically binds a participant to the entire conversation content.	A clean session was run to generate the session_transcript.txt and session_receipt.json artifacts. A dedicated script (scripts/verify_offline.py) was run to act as a third-party auditor.	**Pass.** The offline auditor script confirmed three checks: **1)** Single message signature was valid. **2)** The final receipt signature was valid against the SHA-256 hash of the entire transcript. **3)** Tampering with even a single byte of the transcript file caused the final receipt verification check to fail. This proves the conversation's

			integrity is cryptographically locked by the receipt.

Replay Session Tests :



3. Conclusion

The comprehensive test suite successfully validates the security mechanisms implemented in the application. The system is demonstrably resilient against passive eavesdropping, unauthorized client access, message tampering, and replay attacks. Furthermore, the implemented non-repudiation feature, using a signed session receipt, has been proven effective through successful offline verification, confirming that the solution meets all CIANR security requirements.