Reviewer: Debasish Saha Pranta(a1963099)
Reviewed for: Group 77

Date: 19 October 2025

1. General Feedback

Your Secure Chat System demonstrates a well-structured and thoughtful approach to implementing encrypted communication in a distributed environment. The design shows a solid understanding of public key cryptography, digital signatures, and secure file transmission.

The codebase is cleanly organized into client, server, and introducer components, and it's evident that you've paid close attention to both modularity and protocol correctness. The choice of RSA-OAEP for encryption and RSASSA-PSS for signing aligns well with secure software engineering practices.

Overall, your submission reflects a technically strong effort with good security awareness and practical attention to implementation detail.

---

2. Positive Aspects

1. **Strong Cryptography Usage**

   The use of RSA-OAEP and PSS signatures demonstrates clear understanding of modern cryptographic standards for confidentiality and integrity.

2. **Readable Message Flow**

   Logs like USER_ADVERTISE, MSG_DIRECT, and USER_FILE_CHUNK make it easy to trace interactions between clients and servers, which is great for debugging and analysis.

3. **File Transfer Design**

   The chunk-based file transfer implementation is efficient and realistic, avoiding large payload bottlenecks.

4. **Modularity**

   Key handling, network management, and protocol logic are neatly separated, making the system maintainable and testable.

5. **Replay Protection**

Timestamp and message ID freshness checks show solid defense-in-depth thinking.

---

3. Security & Vulnerability Analysis

While the system's architecture shows a strong grasp of secure design principles, several potential vulnerabilities and improvement opportunities were noted through static inspection and testing attempts:

1. **Introducer Trust Model – Possible Spoofing Risk**

   The introducer appears to broadcast public keys without strong signature validation of joining servers. If not properly verified, an attacker could impersonate a legitimate server to intercept user metadata or message routes.

2. **Cross-Server Message Integrity**

   Messages relayed across servers might not consistently verify the sender's digital signature before forwarding. This opens potential for **message injection or replay attacks** between federated servers.

3. **Logging of Sensitive Data**

   Console logs sometimes print peer IDs, IPs, and message events (including file names). This can leak identifiable information in production environments. Consider masking or minimizing these details.

4. **Lack of Authentication on Federation Channels**

   The inter-server handshake (introducer_uri join) doesn't show explicit challenge–response or signature validation. Attackers could potentially join the federation with a crafted message and disrupt communication.

5. **DoS via File Transfer Flooding**

   The chunked transfer process does not appear to rate-limit or authenticate sender requests. A malicious peer could flood the network with partial file chunks to exhaust memory or bandwidth.

6. **Timeout Handling & Resilience**

   File transfer timeouts (observed after several chunks) suggest missing ACK or resume logic. An attacker could exploit this to trigger repeated connection resets (denial-of-service behavior).

4. Recommendations for Hardening

- Enforce signature verification for every JOIN, ANNOUNCE, and FORWARD message.
- Sanitize or hash sensitive log data (peer IDs, IPs, filenames).
- Add input validation for file sizes, chunk counts, and payload formats.
- Consider adding HMAC or per-session symmetric encryption for message relay after RSA handshakes.
- Introduce client authentication tokens to prevent unauthorized message initiation.
- Implement rate-limiting and retry mechanisms for file transfers.

5. Overall Comments

Your submission demonstrates impressive understanding of secure distributed system design. The implementation shows strong use of encryption, logical flow, and user-friendly feedback.

However, addressing the introducer authentication gaps and tightening inter-server message validation would significantly improve the system's robustness against spoofing and injection threats.

Once these are refined, the project could serve as a model secure chat architecture combining usability, modularity, and solid cryptographic design.