**Reviewer:** Debasish Saha Pranta(a1963099)
**Reviewed for**: Group 38

**Date:** 19 October 2025

## 1. General Feedback

The project demonstrates strong architectural ambition and thoughtful use of security concepts. The team has clearly designed a multi-component system with an Introducer, peer servers, and Python clients communicating via signed and encrypted WebSocket messages. The documentation is detailed and professional, showing that the group understands both distributed coordination and cryptographic protection.

**Positive Observations**

- The architecture follows a layered security approach: introducer registration, RSA-based authentication, and signed payloads.
- Clear modular design (Introducer + Server + Client) makes it easier to reason about trust boundaries.
- Use of Spring Boot and REST/WebSocket interfaces is well-structured and cleanly separated.
- The README explains setup, keys, and expected data flow, showing good understanding of secure protocol design.

**Testing & Execution Findings**

- The Introducer service builds and runs successfully, which confirms that basic Spring Boot dependencies and application configuration work as intended.
- However, the **server module fails to compile** due to missing Lombok-generated getters/setters (getType(), getHost(), setPayload() etc.). Because of this, I could not start the server or connect any Python clients, preventing dynamic testing of message exchange and encryption.
- Several build-time warnings show **duplicate Maven plugin and dependency declarations**, which can disable annotation processing and lead to inconsistent builds.

**Potential Vulnerabilities (based on code inspection)**

1. **Weak input validation** – Several DTOs (e.g., CreateUserRequest, ProtocolMessage) do not appear to sanitize input fields such as usernames or message payloads. This could allow injection or denial-of-service vectors if untrusted

data is processed directly.

2. **Key management exposure** – Key generation scripts (dev-keygen.sh) mentioned in the README were missing. Without secure key provisioning, there is a risk of developers manually reusing or exposing test keys.

3. **Lack of authentication on WebSocket endpoints** – The Introducer appears to accept JOIN or ANNOUNCE messages without clear signature verification. If not enforced, this can allow impersonation of nodes.

4. **Information disclosure via logs** – Debug statements print peer IDs, hosts, and possibly user data to console logs. In production, this could leak sensitive metadata.

5. **Error-handling gaps** – Missing null checks or unchecked exceptions (e.g., in message relay handlers) could allow remote peers to crash the server through malformed packets.

**Suggestions for Improvement**

- Fix Lombok configuration and ensure annotation processing runs correctly so the server compiles.

- Apply strict validation and sanitization for all incoming JSON payloads.

- Verify digital signatures before accepting introducer or peer announcements.

- Remove or mask sensitive fields (IDs, IPs, keys) in log output.

- Provide sandbox or Docker deployment scripts to allow safe testing in isolated environments.

- Include automated unit and integration tests for encryption, authentication, and message integrity.

**Overall Assessment**

Although the system could not be fully executed due to build errors, the design and documentation indicate a solid understanding of secure protocol design principles. The architectural clarity, modular separation, and use of modern frameworks are commendable. Once the build and runtime issues are addressed, this project has the potential to be a robust, secure chat system.