**Reflective Commentary - Secure Overlay Chat Protocol (SOCP)**

**Developed by Group 12 for the University of Adelaide Security Programming course, 2025**

**Introduction: Context and Purpose**

This report reflects on Group 12’s experience developing the Secure Overlay Chat Protocol (SOCP v1.3) for the Advanced Secure Programming assignment. It focuses on the design and implementation of a secure, standards-compliant version, alongside a deliberately backdoored version for controlled peer review. The commentary critically examines protocol decisions, cryptographic integrity, key management, testing, interoperability, and lessons learned from feedback, while addressing ethical considerations, assumptions, and challenges encountered during the creation and evaluation of vulnerabilities.

**Reflection on the Standardised Protocol**

The Secure Overlay Chat Protocol (SOCP) represents a robust, end-to-end encrypted communication framework designed for decentralized networks. Its **n-to-n server mesh topology** ensures that each server maintains knowledge of its local users while relying on a network-wide directory for routing messages securely between servers. Every payload is encrypted using **RSA-4096 with OAEP for encryption and PSS for signatures**, guaranteeing confidentiality and integrity without relying on symmetric key distribution.

SOCP leverages WebSocket (RFC 6455) as the mandatory transport layer, providing real-time, bidirectional communication for both server-to-server and user-to-server messaging. Initially, our group considered TCP due to its straightforward, low-level connections and minimal overhead, reflected in early votes. However, during implementation we realized TCP alone would complicate browser compatibility, asynchronous messaging, and reliable propagation of public channel broadcasts. The voting ultimately favoured WebSocket, recognizing that standardizing on them simplifies client implementation, ensures interoperability, and better supports RSA-4096 encrypted payloads, content signatures, and fan-out of public channel messages. This highlights the tension between low-level control and practical deployment, showing how browser support, persistent connections, and secure broadcast channels decisively shape protocol adoption.

Implementing SOCP gives teams hands-on experience with distributed network coordination, end-to-end encryption, and secure message routing, highlighting the challenges of building a scalable, interoperable chat system. Its design enforces strong security RSA-4096 encryption, signed messages, and transport-level verification, while WebSocket enable persistent, browser-compatible bidirectional messaging, and gossip-based presence updates support fault-tolerant delivery.

Trade-offs include higher computational overhead from RSA-only encryption, limited flexibility due to the mandatory public channel, and added complexity in managing per-user keys and broadcast fan-out. Multi-server routing and gossip updates may also introduce latency in larger networks. Overall, SOCP provides a rigorous, security-first framework that balances standardization, reliability, and real-world implementation challenges.

**Backdoor Design and Exploitation**

* Describe what vulnerability your team intentionally introduced
* Explain your rationale — to illustrate how minor logic flaws (e.g., misplaced validation) can subvert strong cryptography.
* Reflect on how hard it was to detect: What made it plausible and subtle?
* Show the exploit conceptually (in appendix).
* Discuss ethics: how intentional vulnerabilities reveal the fragility of developer assumptions.

**Evaluation of Received Feedback**

* **Backdoors:** Check if other groups detected any intentionally left debug or test backdoors.
* **Code Issues:** Note if they identified security flaws, input validation problems, or logic errors.
* **Usefulness:** Assess whether the feedback was clear, actionable, and helped improve your project.
* **Overall Reflection:** Summarize key insights and any gaps in their review.

**Feedback Provided to Other Groups**

The following feedback was provided by Sahar Hassan Alzahrani to groups 29, 100, and 42 on their secure chat system projects. Feedback for Group 29’s chat system highlighted strengths in design and modern library usage, while identifying vulnerabilities such as hardcoded keys, insecure cookies, unsafe logging, and unencrypted WebSocket. A key challenge in this review was that I had not previously worked with some of these modern libraries, and the project included multiple components, including a frontend, which required additional effort to understand and correctly apply static and dynamic analysis tools. For Group 100’s Secure Overlay Chat System, the review emphasized robust thread-safe design, structured message headers, and session-specific keys, yet critical issues were identified, including unauthenticated debug backdoors, weak MD5-derived IVs, binding to all network interfaces, broad exception handling, and input-validation deficiencies. A primary challenge in this case was interpreting complex runtime behaviours and reproducing crashes safely, which I addressed through focused fuzzing harnesses, manual code tracing, and embedding reproducible examples. Transitioning to Group 42, their Rust-based Secure Chat Protocol demonstrated strong type safety, structured identifier abstractions, and resilient error handling under fuzzed input. Static and dynamic analyses revealed timing side-channels in RSA signing, insecure WebSocket usage, unvalidated FromStr parsing, and weak key management, underscoring the importance of rigorous input validation, secure transport channels, and robust cryptographic hygiene. The challenge in reviewing this group was navigating Rust-specific idioms and cryptographic abstractions while maintaining clarity in recommendations. The full, detailed reviews and supporting evidence are provided in the Appendix.

* **Identify:** State your name and the group reviewed.
* **Overview:** Brief summary of project purpose and focus areas.
* **Strengths:** Highlight well-implemented design or security features.
* **Weaknesses:** Identify vulnerabilities or issues; mention tools used (static/dynamic/manual).
* **Challenges:** Note any difficulties you faced and how you addressed them.

**Reflection on AI Use**

Critically evaluate your use of AI (e.g., ChatGPT) for documentation, debugging, and explanation:

* Strengths: code organization, error interpretation, concept clarification.
* Limitations: lack of context for dynamic runtime issues, occasional hallucinations about function names or parameters.
* Reflect ethically: how AI accelerated understanding but required human verification for cryptographic correctness.
* End with insight: AI is a valuable teaching assistant, not an oracle — it reinforces critical reading of generated code.

**Reflection on testing**

We conducted unit tests to verify the correctness of core functions and integration tests to ensure proper client-server messaging and system behavior. Security testing ......... Interoperability testing......... These tests helped identify minor issues, which were resolved to maintain compatibility and functional integrity. Detailed test scripts, results, and documentation are provided in the GitHub repository and Appendix, giving a clear record of our testing methodology and outcomes.

**Group Contributions**

* **Debasish Saha Pranta:** introducer YAML, server-server, backdoor implementation …
* **Samin Yeasar Seaum:**  file transfer, cryptographic, backdoor implementation … …
* **Abidul Kabir:** server-client protocol flow, socket handling, database persistence…
* **Sahar Alzahrani:** Built the initial client-server framework with WebSocket messaging and RSA security; added documentation, testing, secure-version validation, and organized the project.
* **Mahrin Mahia:** backdoor design and testing…
* **Maria Hasan Logno:** test case integration, interoperability experiments…

**Conclusion**

* Summarize the overall growth: from implementing a secure system to realizing how human error undermines security.
* Emphasize learning outcomes: protocol discipline, ethical coding, cryptographic transparency.
* Connect to the *cybersecurity skill shortage*: real-world demand for developers who understand both the defensive and offensive sides of secure software.

**Group 12:**

* 1. Debasish Saha Pranta (a1963099, debasishsaha.pranta@student.adelaide.edu.au)
* 2. Samin Yeasar Seaum (a1976022, saminyeasar.seaum@student.adelaide.edu.au)
* 3. Abidul Kabir (a1974976, abidul.kabir@student.adelaide.edu.au)
* 4. Sahar Alzahrani (a1938372, sahar.alzahrani@student.adelaide.edu.au)
* 5. Mahrin Mahia (a1957342, mahrin.mahia@student.adelaide.edu.au)
* 6. Maria Hasan Logno (a1975478, mariahasan.logno@student.adelaide.edu.au)

**Appendix**