Peer Review Report

Reviewer: Abidul Kabir
ID: a1974976
Date: 16 October 2025 (Revised 18 October)

Summary

The system shows a strong understanding of secure communication design and key management, with structured modularity and working encryption/signing logic.

However, both manual inspection and automated scanning (Bandit) reveal several reliability and code-quality concerns — especially excessive use of silent try/except/pass and minor database query construction risks.

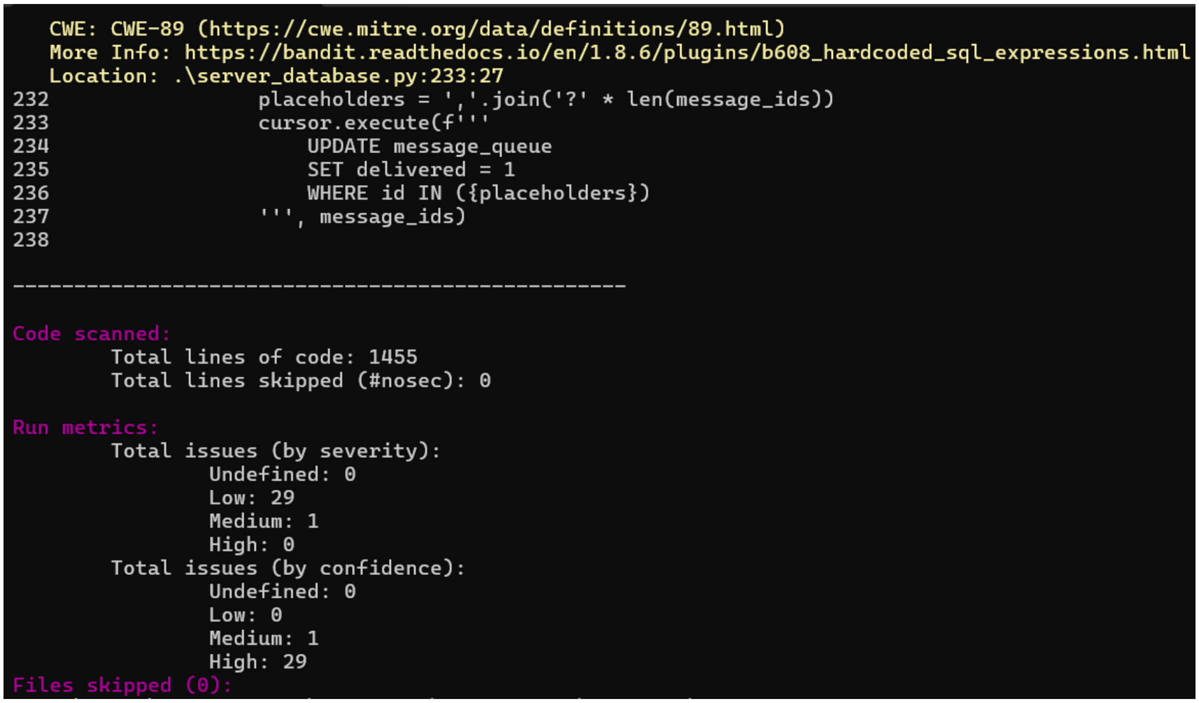
Tools and Testing Approach

Method	Tools / Process
Static Security Analysis	bandit -r .
Manual Code Review	Reviewed client.py, node.py, server_database.py, crypto.py, config.py
Functional Testing	Local runtime tests of node startup and message exchange
Code Quality	Pylint

High-Level Findings

1. [Critical] Weak or Hardcoded Cryptographic Parameters.
2. [High] Unsecured Transport (no TLS).
3. [High] No Authentication or Key Validation Handshake.
4. [Medium] Incomplete Input Validation
5. [Medium] SQL Query Composition Risk.

Bandit Result



```
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b608_hardcoded_sql_expressions.html
   Location: .\server_database.py:233:27
232                 placeholders = ','.join('?' * len(message_ids))
233                 cursor.execute(f'''
234                     UPDATE message_queue
235                     SET delivered = 1
236                     WHERE id IN ({placeholders})
237                 ''', message_ids)
238

--------------------------------------------------

Code scanned:
        Total lines of code: 1455
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 29
                Medium: 1
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 1
                High: 29
Files skipped (0):
```

Finding Details and Impacts

Severity	Finding	Description	Impact
Critical	**Weak or Hardcoded Cryptographic Parameters**	RSA key generation uses a configurable bit length; some scripts allow 1024-bit fallback or omit explicit enforcement.	If user input or environment variables can set key length, the system becomes vulnerable to brute-force or factorization attacks.
High	**Unsecured Transport (no TLS)**	The system communicates using unencrypted WebSocket (ws://) or raw TCP connections instead of wss://.	Without TLS, attackers on the same network can intercept or manipulate messages, exposing user metadata and keys.
High	**No Authentication or Key Validation Handshake**	The protocol allows peers to announce public keys without verifying their ownership through a challenge–response or signed proof.	Enables impersonation and replay attacks—an attacker could register a key for another user and receive messages intended for them.
Medium	**Incomplete Input Validation**	Incoming JSON frames and user data are parsed without field or schema validation, relying solely on json.loads() success.	Malformed or oversized payloads could cause runtime exceptions or denial-of-service crashes during message handling.
Medium	**SQL Query Composition Risk.**	SQL queries use f-strings to insert variables (Bandit finding B608). Parameterized execution is not consistently applied.	Potential for SQL injection if user input reaches the query layer. Even if currently internal, this is a maintainability and safety risk.

Recommendations

1. Enforce RSA keys ≥2048 bits; remove weak key options and validate key strength on import.
2. Use wss:// or TLS-secured connections; enable certificate validation to prevent MITM attacks.
3. Implement challenge–response verification to confirm key ownership and prevent impersonation.
4. Add JSON schema or field validation and limit message size to prevent malformed input or DoS.
5. Use parameterized SQL queries instead of f-strings to eliminate injection risk.
6. Replace silent try/except/pass with logging; add docstrings and structured error handling.

Code Quality Review (Pylint Summary)

Overall Score: 7.11 / 10

Strengths

- Code executes successfully with no syntax errors or fatal issues.
- The system is modular with clear separation between **client**, **node**, and **database** components.
- Implements key cryptographic and networking logic aligning with SOCP's architecture.
- Readable code flow with correct use of Python async and socket handling.

Issues Identified

- **Missing documentation:** Many modules, classes, and methods lack docstrings (C0114, C0115, C0116).
- **Excessive line length and whitespace:** Numerous long lines and trailing whitespace warnings.
- **Broad exception handling:** Frequent use of except Exception: (W0718) reduces error visibility and can hide vulnerabilities.
- **Complexity warnings:** Too many branches/statements in functions (R0912, R0915), making logic harder to maintain.
- **Redundant imports and unused variables:** Reimported or unused modules (W0404, W0611).
- **Duplicate code blocks:** Repeated cryptographic helper functions across files (R0801).
- **No docstring or type hints for database methods:** Makes schema handling and debugging harder.

Functional / SOCP Gaps

1. Server bootstrap process – No dynamic handshake (SERVER_HELLO_JOIN, SERVER_WELCOME, SERVER_ANNOUNCE).
2. Presence cleanup – No broadcast of USER_REMOVE when clients disconnect.
3. Duplicate-message suppression – Missing seen_ids cache to prevent routing loops.
4. Heartbeat and timeout checks – Absent HEARTBEAT frames and 45 s timeout policy.
5. File transfer feature – Missing FILE_START, FILE_CHUNK, FILE_END handling.
6. Standard ACK / ERROR responses – Not implemented
7. Direct message verification – End-to-end signature verification missing.