Sam - Group 3 Peer Review

This implementation demonstrates a commendable effort in establishing the foundational elements of a secure chat application. Notably, the fingerprint calculation aligns correctly with the protocol's specifications, and essential components such as key generation and basic client-server communication are effectively implemented. These strengths provide a solid base for building a functional and secure system.

However, there are significant deviations from the OLAF/Neighbourhood protocol, particularly in cryptographic implementations and message signing processes. Addressing these discrepancies will be crucial in enhancing the application's security and ensuring full compliance with the established protocol standards. The following sections delve deeper into these aspects, offering detailed feedback and recommendations for improvement.

Reviewed By:

- Samuel Chau (a1799298)

1. Manual Code Review

Architecture and Design

Aspect	Status	Comments
Protocol Implementation Adherence	Not Adhered	The codebase shows significant deviations from the OLAF/Neighbourhood protocol specifications. Key discrepancies include: 1. **Message Signing**: The protocol specifies that messages should be signed using RSA-PSS with SHA-256, and the signature should be the Base64-encoded result of signing the concatenation of the data JSON and the counter. However, the code uses a simple SHA-256 hash of the data concatenated with the counter and then Base64-encodes it, without using the private key for RSA-PSS signing. This does not provide the necessary cryptographic assurance specified by the protocol. 2. **Counter Handling**: The protocol requires that message counters be monotonically increasing integers to prevent replay attacks. While the code includes a counter, there is no implementation for tracking and verifying the counters of incoming messages to ensure they are greater than the previous value, as required by the protocol. 3. **Cryptographic Specifications**: The code does not adhere to the specified cryptographic parameters. For instance, AES keys are generated with a length of 32 bytes (256 bits), whereas the protocol specifies a key length of 16 bytes (128 bits). Additionally, the code uses PyCrypto, which is deprecated, instead of a current cryptographic library. 4. **Message Structures**: Some message formats deviate from the protocol. For example, the `public_chat` message in the

Aspect	Status	Comments
		code includes a `sender` field with the `fingerprint`, but this has been implemented as signed data. 5. Error Handling and Protocol Compliance: The code assumes well-formed inputs and lacks comprehensive validation of incoming messages against the protocol specifications. This can lead to unexpected behavior when messages do not conform exactly to the expected format.

Code Quality

Aspect	Status	Comments
Readability and Organization	Poor Organization	The codebase is disorganized and difficult to navigate. It contains code from previous iterations that are not relevant to the current implementation, leading to confusion. There is a lack of modularization, and functions are not well-separated or documented. The absence of a README or documentation makes it challenging to understand how to set up and run the application. Hardcoded addresses and ports are scattered throughout the code, making it inflexible and hard to configure for different environments. Overall, the code lacks structure and clarity, impeding maintainability and scalability.
Error Handling and Logging	Inadequate	Error handling is insufficient and inconsistent. The application will almost certainly crash when unexpected inputs are provided or when network errors occur, without graceful recovery or informative error messages. Logging is minimal, relying primarily on print statements that are not systematically used throughout the code. This makes debugging and monitoring the application difficult. Critical errors are not properly caught, and exceptions can lead to the application terminating unexpectedly, which poses reliability and security concerns.

Security-specific Checks

Aspect	Status	Comments
Input Validation	Massive Threat Vector	There is minimal input validation across the application. User inputs are not properly sanitized or validated. The code assumes well-formed inputs, which is not safe in a production environment. For example, message parsing relies on searching for substrings within user input to determine the type of message to send (e.g., using `find` on "public:"). This method is dangerous because if a user includes certain keywords in their message unintentionally, it could trigger unintended behaviors, such as broadcasting a private message publicly. This represents a significant security and privacy concern.
Access Control	Needs improvement	The application lacks proper access control mechanisms. Any user can connect to the server without authentication, and there are no checks to prevent

Aspect	Status	Comments
		unauthorized actions. Additionally, multiple clients can connect using the same public key, which should not be permitted, as each client should have a unique key pair to ensure proper identification and secure communication.
Cryptographic Implementations	Needs improvement	The cryptographic implementations in the code do not fully comply with the protocol specifications: 1. Message Signing: The protocol requires RSA-PSS with SHA-256 for digital signatures. The code, however, uses a simple SHA-256 hash of the data concatenated with the counter, without involving the private key for signing. This means that message signatures are not secure, as anyone can compute the SHA-256 hash without possessing the private key. 3. Use of Deprecated Libraries: The code imports `AES` and `get_random_bytes` from the `Crypto` library (PyCrypto), which is deprecated and no longer maintained. This poses security risks due to potential unpatched vulnerabilities. The protocol suggests using current and secure libraries for cryptographic operations. 4. Fingerprint Calculation: The code calculates the fingerprint by hashing the PEM-encoded public key and then Base64-encoding it, which aligns with the protocol's definition. This aspect appears to be correctly implemented. Overall, the incorrect implementation of cryptographic operations undermines the security guarantees provided by the protocol.
Secure Data Storage and Transmission	Insecure	Data transmission is insecure due to the use of hardcoded addresses and ports, which cannot be easily configured for secure channels. The application communicates over unencrypted WebSocket connections (`ws://`), without using TLS (`wss://`), leaving the data susceptible to interception. Additionally, public chats are non-functional, and when attempted, they do not work correctly, indicating issues with data handling and transmission. Private chats are also not fully implemented or are unreliable due to the aforementioned cryptographic issues. The lack of secure communication protocols undermines the confidentiality and integrity of the data exchanged between clients and the server.

2. Static Analysis

The static analysis was performed using Bandit, a security-oriented static analysis tool for Python. Below are the findings from the analysis:

`client.py`

Issue ID	Severity	Confidence	Location	Description	Recommendation	More Info
B413	High	High	client.py:14	The `pyCrypto` library and its module `AES` are deprecated and no longer maintained. Using deprecated libraries can introduce security vulnerabilities.	Replace `pyCrypto` with a maintained library such as `pyca/cryptography` for cryptographic operations.	Link

`server.py`

No issues identified.

3. Dynamic Analysis

Testing Environment Setup

Due to the lack of documentation and absence of a README file, setting up the testing environment was challenging. The codebase contains many files and even a subfolder containing an entire previous iteration, adding to the confusion. This made it extremely difficult to figure out how to run the application and where to start. After contacting group members, I was informed that only `client.py` and `server.py` were needed to run the application. While this information was helpful, such guidance should be clearly provided in documentation or a README file.

Moreover, the hardcoded addresses and ports limit the ability to test the application in a multi-server or multi-client setup. After inspecting the code, I attempted to run the `server.py` and `client.py` files to evaluate the application's functionality.

Functional Testing Observations

Aspect	Observation
Application Setup	The instructions for setting up and running the application are not provided. The reviewer had to contact the group members for guidance, but received insufficient information. Consequently, assumptions were made based on the code to attempt running the application.
Hardcoded Addresses and Ports	The application uses hardcoded addresses and ports (`127.0.0.1:8080`), which restricts testing to a single server and client. This limitation prevents testing of the application's behavior in a distributed environment, which is essential for the OLAF/Neighbourhood protocol that involves multiple servers and clients.
Client Update Handling	Sending a client update manually from the client causes the application to crash. The client and server output the following errors:

Client Output:

```
# Client
Enter message: client update:
Enter message: Task exception was never retrieved
future: <Task finished name='Task-5' coro=<receive_messages() done, defined at
/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/SEP-server_final-1/SEP-
server_final/client.py:167> exception=ConnectionClosedError(Close(code=1011,
reason=''), Close(code=1011, reason=''), True)>
Traceback (most recent call last):
  File "/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/SEP-server_final-1/SEP-
server_final/client.py", line 172, in receive_messages
    response = await websocket.recv()
               ^^^^^^^^^^^^^^^^^^^^^^
  File "/home/sam-chau/Documents/OMesh/Venkata
Vishnubhatla/myenv/lib/python3.12/site-packages/websockets/legacy/protocol.py", line
562, in recv
    await self.ensure_open()
  File "/home/sam-chau/Documents/OMesh/Venkata
Vishnubhatla/myenv/lib/python3.12/site-packages/websockets/legacy/protocol.py", line
938, in ensure_open
    raise self.connection_closed_exc()
websockets.exceptions.ConnectionClosedError: received 1011 (internal error); then
sent 1011 (internal error)
```

Server Output:

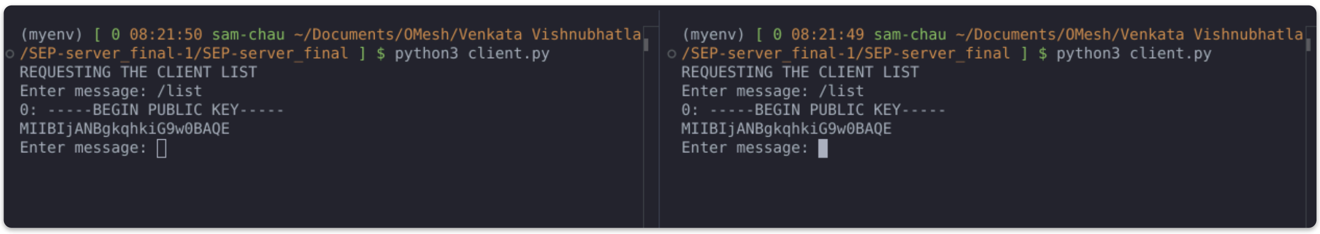
```
# Server
Traceback (most recent call last):
  File "/home/sam-chau/Documents/OMesh/Venkata
Vishnubhatla/myenv/lib/python3.12/site-packages/websockets/legacy/server.py", line
245, in handler
    await self.ws_handler(self)
  File "/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/SEP-server_final-1/SEP-
server_final/server.py", line 325, in websocket_handler
    await handle_message(message, websocket)
  File "/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/SEP-server_final-1/SEP-
server_final/server.py", line 308, in handle_message
    await handle_client_update(data, websocket)
  File "/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/SEP-server_final-1/SEP-
server_final/server.py", line 182, in handle_client_update
    client_keys = data["clients"]
                  ~~~~^^^^^^^^^^^
KeyError: 'clients'
```

Aspect	Observation
Multiple Clients with Same Public Key	The application allows multiple clients to connect to the same server using the same public key. This should not be permitted, as each client should have a unique key pair to ensure proper identification and secure communication. The server output indicates that two users with the same public key are added to the local and global user lists.

Server Output:

```
User added. Total Users in Local List: 2
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
```

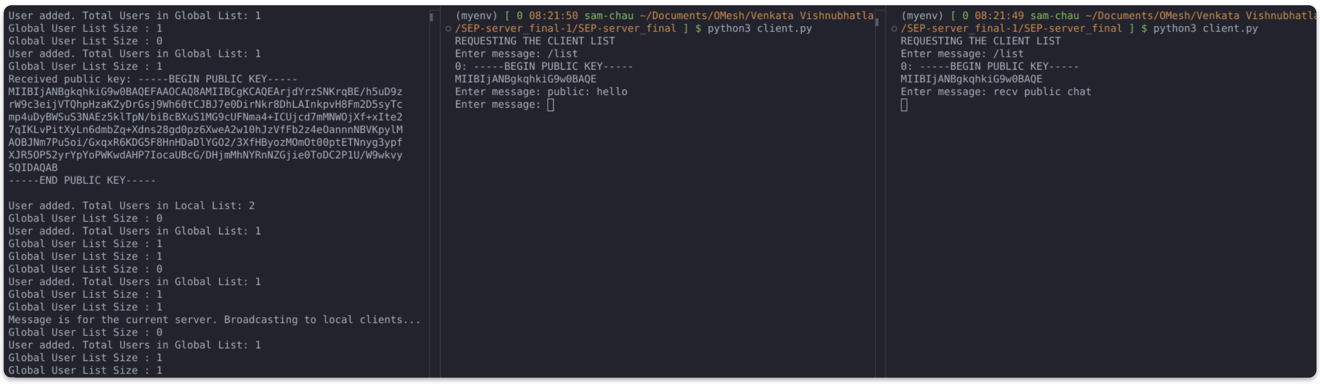
Aspect	Observation
Incorrect Fingerprint Encoding	The fingerprints are not correctly encoded. Instead of being base64-encoded SHA-256 hashes of the public keys, they appear to be excerpts of the PEM keys. This deviates from the protocol specification and can lead to identification issues.
Incomplete Client Update List	The client update list only shows the "key" of one connected client, even when multiple clients are connected. This indicates a problem with the synchronization of client states across the server and clients.



```
(myenv) [ 0 08:21:50 sam-chau ~/Documents/OMesh/Venkata Vishnubhatla
/SEP-server_final-1/SEP-server_final ] $ python3 client.py
REQUESTING THE CLIENT LIST
Enter message: /list
0: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQE
Enter message:
```

```
(myenv) [ 0 08:21:49 sam-chau ~/Documents/OMesh/Venkata Vishnubhatla
/SEP-server_final-1/SEP-server_final ] $ python3 client.py
REQUESTING THE CLIENT LIST
Enter message: /list
0: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQE
Enter message:
```

Aspect	Observation
Public Chats Non-functional	Public chats do not work as intended. Attempting to send a public chat results in errors or no action at all. This feature is crucial for the application's functionality as per the protocol.

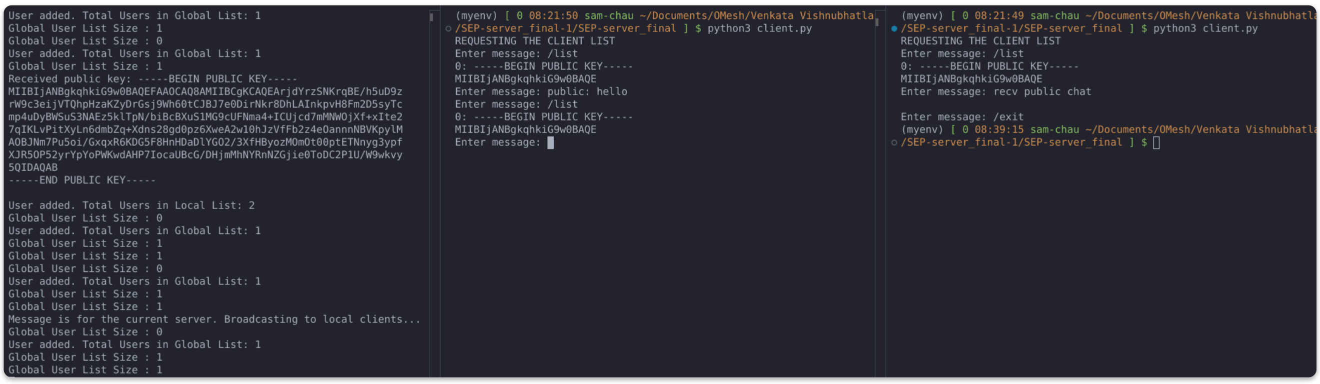


```
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Received public key: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArjdYrzSNKrqBE/h5uD9z
rW9c3eijVTQhpHzaKZyDrGsj9Wh60tCJBJ7e0DirNkr8DhLAInkpvH8Fm2D5syTc
mp4uDyBWSuS3NAEz5klTpN/biBcBXuS1MG9cUFNma4+ICUjcd7mMNWOjXf+xIte2
7qIKLvPitXyLn6dmbZq+Xdns28gd0pz6XweA2w10hJzVfFb2z4eOannnNBVKpylM
AOBJNm7Pu5oi/GxqxR6KDG5F8HnHDaDlYGO2/3XfHByozMOmOt00ptETNnyg3ypf
XJR5OP52yrYpYoPWKwdAHP7IocaUBcG/DHjmMhNYRnNZGjie0ToDC2P1U/W9wkvy
5QIDAQAB
-----END PUBLIC KEY-----

User added. Total Users in Local List: 2
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
Message is for the current server. Broadcasting to local clients...
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
```

```
(myenv) [ 0 08:21:50 sam-chau ~/Documents/OMesh/Venkata Vishnubhatla
/SEP-server_final-1/SEP-server_final ] $ python3 client.py
REQUESTING THE CLIENT LIST
Enter message: /list
0: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQE
Enter message: public: hello
Enter message: 
```

```
(myenv) [ 0 08:21:49 sam-chau ~/Documents/OMesh/Venkata Vishnubhatla
/SEP-server_final-1/SEP-server_final ] $ python3 client.py
REQUESTING THE CLIENT LIST
Enter message: /list
0: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQE
Enter message: recv public chat
```

Aspect	Observation
Exit Command Handling	The `/exit` command works on the client side, closing the client application. However, the server does not recognize the client's disconnection or send out any updates to other clients or servers. This lack of communication about client disconnections can lead to inconsistent client lists and stale state information across the network.

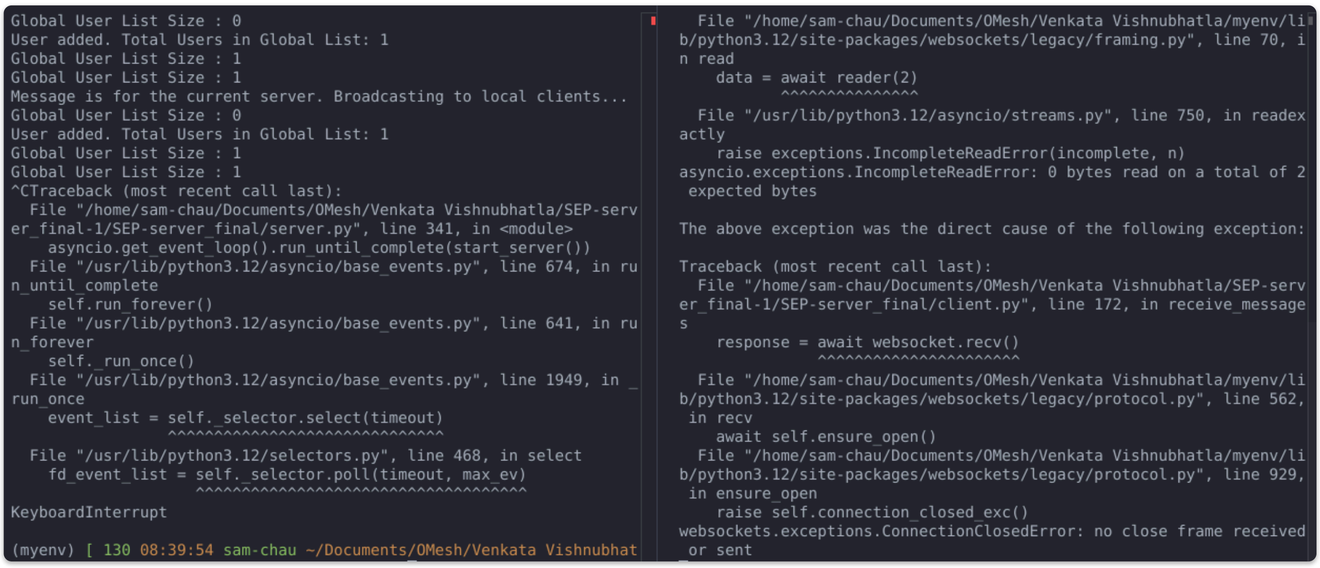


```
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Received public key: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArjdYrzSNKrqBE/h5uD9z
rW9c3eijVTQhpHzaKZyDrGsj9Wh60tCJBJ7e0DirNkr8DhLAInkpvH8Fm2D5syTc
mp4uDyBWSuS3NAEz5klTpN/biBcBXuS1MG9cUFNma4+ICUjcd7mMNWOjXf+xIte2
7qIKLvPitXyLn6dmbZq+Xdns28gd0pz6XweA2w10hJzVfFb2z4eOannnNBVKpylM
AOBJNm7Pu5oi/GxqxR6KDG5F8HnHDaDlYGO2/3XfHByozMOmOt00ptETNnyg3ypf
XJR5OP52yrYpYoPWKwdAHP7IocaUBcG/DHjmMhNYRnNZGjie0ToDC2P1U/W9wkvy
5QIDAQAB
-----END PUBLIC KEY-----

User added. Total Users in Local List: 2
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
Message is for the current server. Broadcasting to local clients...
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
```

```
(myenv) [ 0 08:21:50 sam-chau ~/Documents/OMesh/Venkata Vishnubhatla
/SEP-server_final-1/SEP-server_final ] $ python3 client.py
REQUESTING THE CLIENT LIST
Enter message: /list
0: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQE
Enter message: public: hello
Enter message: /list
0: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQE
Enter message: 
```

```
(myenv) [ 0 08:21:49 sam-chau ~/Documents/OMesh/Venkata Vishnubhatla
/SEP-server_final-1/SEP-server_final ] $ python3 client.py
REQUESTING THE CLIENT LIST
Enter message: /list
0: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQE
Enter message: recv public chat

Enter message: /exit
(myenv) [ 0 08:39:15 sam-chau ~/Documents/OMesh/Venkata Vishnubhatla
/SEP-server_final-1/SEP-server_final ] $ 
```

Aspect	Observation
Error Handling on Shutdown	The application does not handle non-graceful shutdowns properly. If a client or server terminates unexpectedly, the other side does not handle the exception gracefully, leading to unhandled exceptions and potential crashes.



```
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
Message is for the current server. Broadcasting to local clients...
Global User List Size : 0
User added. Total Users in Global List: 1
Global User List Size : 1
Global User List Size : 1
^CTraceback (most recent call last):
  File "/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/SEP-serv
er_final-1/SEP-server_final/server.py", line 341, in <module>
    asyncio.get_event_loop().run_until_complete(start_server())
  File "/usr/lib/python3.12/asyncio/base_events.py", line 674, in ru
n_until_complete
    self.run_forever()
  File "/usr/lib/python3.12/asyncio/base_events.py", line 641, in ru
n_forever
    self._run_once()
  File "/usr/lib/python3.12/asyncio/base_events.py", line 1949, in _
run_once
    event_list = self._selector.select(timeout)
                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3.12/selectors.py", line 468, in select
    fd_event_list = self._selector.poll(timeout, max_ev)
                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
KeyboardInterrupt

(myenv) [ 130 08:39:54 sam-chau ~/Documents/OMesh/Venkata Vishnubhat
```

```
  File "/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/myenv/li
b/python3.12/site-packages/websockets/legacy/framing.py", line 70, i
n read
    data = await reader(2)
           ^^^^^^^^^^^^^^^
  File "/usr/lib/python3.12/asyncio/streams.py", line 750, in readex
actly
    raise exceptions.IncompleteReadError(incomplete, n)
asyncio.exceptions.IncompleteReadError: 0 bytes read on a total of 2
 expected bytes

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/SEP-serv
er_final-1/SEP-server_final/client.py", line 172, in receive_message
s
    response = await websocket.recv()
               ^^^^^^^^^^^^^^^^^^^^^^
  File "/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/myenv/li
b/python3.12/site-packages/websockets/legacy/protocol.py", line 562,
 in recv
    await self.ensure_open()
  File "/home/sam-chau/Documents/OMesh/Venkata Vishnubhatla/myenv/li
b/python3.12/site-packages/websockets/legacy/protocol.py", line 929,
 in ensure_open
    raise self.connection_closed_exc()
websockets.exceptions.ConnectionClosedError: no close frame received
 or sent
```

Aspect	Observation
Message Parsing Vulnerability	The method of parsing commands by searching for substrings (e.g., using `find` within a message string) is dangerous. If a user includes certain keywords (e.g., "public:") in their message, it can trigger unintended behaviors, such as broadcasting a private message publicly. This represents a significant security and privacy concern.

Summary of Dynamic Analysis

The dynamic analysis reveals that the application is largely non-functional and does not adhere to the expected behavior as defined by the OLAF/Neighbourhood protocol. Critical features such as public chats, private / group chats, client updates, and proper handling of client connections are either broken or not implemented correctly. The lack of proper error handling and input validation leads to crashes and inconsistent states, making the application unreliable and insecure for practical use.

4. Backdoor/Vulnerability Assessment

Backdoor	Description	Impact
Message Type Parsing Vulnerability	**Location:** `client.py`, within the `chat` function. The client determines the type of message to send based on the presence of specific substrings anywhere in the user's input (e.g., `"public:"`, `"client update:"`). This parsing method does not enforce that these substrings appear at the start of the message or as standalone commands. As a result, if a user includes these substrings within their message, the client may misinterpret the message type. For example, a private message containing the text `"Let's make this public: not private"` would be interpreted as a public chat due to the presence of `"public:"`, causing the message to be broadcast publicly instead of sent privately. The use of `message.find("public:") != -1` suggests an intentional design choice that can lead to message misclassification.	An attacker can exploit this behavior to trick users into inadvertently broadcasting private messages publicly, leading to the disclosure of sensitive information. By crafting messages that include these substrings, attackers can manipulate the client into performing unintended actions. This undermines user privacy and can cause reputational damage or compromise confidential data. The intentional use of such a parsing method, despite its known risks, suggests a backdoor that enables message interception and unauthorized disclosure.

Additional Observations:

No further intentional backdoors were identified in the code. However, the application contains significant vulnerabilities due to improper implementation of cryptographic functions, lack of input validation, and failure to adhere to protocol specifications. These vulnerabilities severely compromise the security of the application and must be addressed promptly.

6. Results Summary

Strengths

Strength	Description
Fingerprint Calculation	The application correctly calculates the fingerprint of the public key by hashing the PEM-encoded public key and then Base64-encoding it, aligning with the protocol's definition.
Basic Structure in Place	The code includes foundational elements such as key generation, message handling functions, and basic client-server communication setup.

Areas for Improvement

Area for Improvement	Description
Protocol Adherence	The application significantly deviates from the OLAF/Neighbourhood protocol specifications in critical areas such as message signing, counter handling, and message structures.
Cryptographic Implementations	Incorrect use of cryptographic functions, including improper message signing and the use of deprecated libraries, undermines security.
Code Organization and Readability	The code lacks proper organization, modularization, and documentation, making it difficult to understand and maintain.
Input Validation and Error Handling	Minimal input validation and inadequate error handling lead to application crashes and vulnerabilities.

Final Note: A good effort all around to provide a strong foundation, but needs lots of improvement before the final deadline.

7. Recommendations

Security Enhancements

Recommendation	Description
Adhere to Protocol Specifications	Strictly follow the OLAF/Neighbourhood protocol for message signing, counter usage, and message formats to ensure security compliance.
Proper Cryptographic Practices	- Implement RSA-PSS with SHA-256 for message signing using the private key. - Replace deprecated libraries like PyCrypto with modern alternatives such as `cryptography`. - Use correct key sizes and cryptographic parameters as specified by the protocol.
Secure Communication Channels	Use TLS (`wss://`) for WebSocket connections to encrypt data in transit and protect against interception.

Code Quality Improvements

Recommendation	Description
Refactor Codebase	Organize the code into modular, reusable components with clear separation of concerns. Remove obsolete or irrelevant code segments.
Enhance Readability	Add comprehensive comments and documentation to explain code functionality.
Improve Error Handling	Implement robust error handling to catch and manage exceptions gracefully, preventing application crashes.