## I. Client.py

The `Client.py` script sets up a TCP client that communicates with a server using AES encryption to ensure secure data transfer. Here's a breakdown of its components:

**Imports and Initialization**

- `pyaes`: Used for AES encryption and decryption.
- `socket`: Provides the low-level network interface for communication.
- `threading`: Enables concurrent execution of tasks, allowing the client to handle incoming and outgoing messages simultaneously.
- `hashlib`: Used for hashing messages to verify their integrity.
- `json`: Facilitates the serialization and deserialization of data to and from JSON format.
- `datetime`: Adds timestamps to messages.

**Client Setup**

- The client prompts the user for the server's IP address, port, and an AES pre-shared key.
- It creates a socket, connects to the server, and initializes the AES cipher using a SHA-256 hashed key.

**Functions**

1. `process_bytes`:
    - Breaks down the received byte stream into 16-byte chunks suitable for AES decryption.
2. `process_text`:
    - Converts a string message into 16-byte blocks, padding with ~ if necessary, and then converts each block to a list of byte values.
3. `verify_and_display`:
    - Verifies the integrity of received messages by comparing their SHA-256 hash with the hash received.
    - Displays the message along with a verification tag (☑ for valid, ☒ for invalid).

**Threading and Message Handling**

- `myThread` Class:
  - A custom thread class that continuously listens for messages from the server.
  - It decrypts the incoming messages, verifies their integrity, and prints them.
- Main Loop:
  - The main thread takes user input, hashes it, encrypts it, and sends it to the server.
  - It listens for "quit()" to stop the thread and close the connection.

**Security Measures**

- AES Encryption: Protects the confidentiality of the messages.
- SHA-256 Hashing: Ensures the integrity of the messages by comparing hashes.

## II. Server.py

The `Server.py` script sets up a TCP server that securely communicates with a client using AES encryption. Here's a detailed explanation of its components:

### Imports and Initialization

- `pyaes`: For AES encryption and decryption.
- `socket`: Manages network communication.
- `threading`: Allows the server to handle concurrent tasks, such as listening for client messages and processing inputs.
- `hashlib`: Used for hashing messages to ensure integrity.
- `json`: Handles the conversion of messages to and from JSON format.
- `datetime`: Adds timestamps to messages for logging and verification.

### Server Setup

- The server listens on a default port (`5555`), or a user-specified port if provided as a command-line argument.
- It binds to all network interfaces (`0.0.0.0`) and waits for incoming client connections.

### Key Components

1. Socket Initialization:
    - The server creates a socket, binds it to the host and port, and listens for incoming connections.
    - Upon accepting a connection, it prints the client's address.
2. AES Key Setup:
    - The server prompts the user for a pre-shared AES key, hashes it using SHA-256, and initializes the AES cipher.

### Functions

1. `verify_and_display`:

    - Verifies the integrity of received messages by comparing their SHA-256 hash with the hash received.
    - Displays the message with a verification tag (☑ for valid, ☒ for invalid).
2. `process_bytes`:
    - Processes the received data into 16-byte chunks for decryption.
3. `process_text`:
    - Converts string messages into 16-byte blocks, padding with ~ as needed, and converts each block to byte values.

**Threading and Message Handling**

- `myThread` Class:
    - A custom thread that continuously listens for incoming messages from the client.
    - It decrypts, verifies, and displays these messages.
- Main Loop:
    - The main thread takes user input, hashes and encrypts it, then sends it to the client.
    - It listens for "quit()" to stop the thread and close the connection.

**Security Measures**

- AES Encryption: Ensures the confidentiality of the messages exchanged between the server and the client.
- SHA-256 Hashing: Verifies the integrity of the messages to detect tampering.