**Key Features:**

1. **AES Encryption**:
   * The system uses the **AES block cipher** (Advanced Encryption Standard) with a 256-bit key to encrypt and decrypt the messages exchanged between the client and server.
   * **AES operates on 16-byte blocks**, meaning that any data sent over the network must be processed in 16-byte chunks.
2. **SHA-256 Hashing**:
   * The **SHA-256 hashing algorithm** is used to ensure message integrity. Before sending the message, the client computes the hash of the message and includes it with the message itself.
   * The server computes the hash of the received message and compares it with the hash sent by the client. If they match, it means the message has not been tampered with during transmission.
3. **Socket Communication**:
   * The system uses **TCP/IP sockets** for communication. The client connects to the server using a specific IP address and port.
   * Once the server accepts a connection, the client sends encrypted messages, and the server decrypts them using the AES key.
4. **Multithreading**:
   * **Threading** is used to handle both receiving and sending messages simultaneously. The client and server both run a separate thread to listen for incoming messages while still allowing the user to input and send new messages.
5. **Pre-shared Key**:
   * Both the client and the server must share a secret **AES key**. This key is used to encrypt and decrypt messages. The key is entered by the user when the server and client are started.

**Code Breakdown**

**1. Server Implementation (server.py)**

* **Socket Setup**: The server creates a socket and binds it to a specified host and port. It then listens for incoming client connections. The accept() method is called to accept a connection from a client.

python

Sao chép mã

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.bind((HOST, PORT))

s.listen(1)

conn, addr = s.accept()

* **AES Encryption**: The server expects to receive messages that are encrypted using the AES algorithm. Upon establishing a connection, the server prompts the user to enter a pre-shared AES key, which is hashed using SHA-256 to create a 256-bit key.

python

Sao chép mã

key = str(input('[+] AES Pre-Shared-Key for the Connection : '))

hashed = hashlib.sha256(key.encode()).digest()

aes = pyaes.AES(hashed)

* **Message Decryption and Integrity Check**: When the server receives a message, it decrypts the data using AES. After decryption, the server computes the hash of the received message and compares it to the hash included in the message to verify its integrity.

python

Sao chép mã

data\_recv = json.loads(mess)

verify\_and\_display(data\_recv)

If the hashes match, the server displays the message with a "check" symbol (☑); otherwise, it displays a "cross" symbol (☒).

* **Multithreading**: The server uses a thread to listen for incoming data continuously, without blocking the main thread. This allows the server to handle multiple tasks concurrently.

python

Sao chép mã

Listening\_Thread = myThread(1)

Listening\_Thread.daemon = True

Listening\_Thread.start()

**2. Client Implementation (client.py)**

* **Socket Setup**: The client connects to the server by specifying the IP address and port. After establishing the connection, the client prompts the user to enter the pre-shared AES key.

python

Sao chép mã

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.connect((HOST, PORT))

* **Message Encryption**: Before sending a message, the client encrypts it using AES and computes the SHA-256 hash of the message. Both the message and the hash are then sent to the server.

python

Sao chép mã

sending\_data = str(input(""))

mess\_hash = hashlib.sha256(str(sending\_data).encode('utf-8')).hexdigest()

send\_data = {"timestamp": timestamp, "message": sending\_data, "hash": mess\_hash}

The message is then processed into 16-byte chunks and encrypted.

* **Multithreading**: Similar to the server, the client also runs a listening thread to continuously receive messages from the server while still allowing the user to input and send messages.

python

Sao chép mã

Listening\_Thread = myThread(1)

Listening\_Thread.daemon = True

Listening\_Thread.start()

* **Sending Encrypted Data**: After encrypting the message, the client sends the ciphertext over the network to the server.

python

Sao chép mã

enc\_bytes = []

for byte in sending\_bytes:

ciphertext = aes.encrypt(byte)

enc\_bytes += bytes(ciphertext)

s.send(bytes(enc\_bytes))

**3. Threading for Concurrent Communication:**

Both the server and client create a thread to handle incoming messages concurrently. The main thread is responsible for sending messages, while the listening thread is responsible for receiving and processing messages.

python

Sao chép mã

class myThread(threading.Thread):

def \_\_init\_\_(self, id):

threading.Thread.\_\_init\_\_(self)

self.threadID = id

def stop(self):

self.is\_alive = False

def run(self):

# Threaded listening logic

**How the System Works:**

1. **Server Side**:
   * The server waits for a client to connect. Once connected, it listens for incoming messages, decrypts them, verifies the message's hash, and displays the message if the hash matches.
   * The server can also send encrypted messages back to the client.
2. **Client Side**:
   * The client connects to the server, sends messages after encrypting them, and receives messages sent by the server. The client encrypts each message using AES and ensures the integrity of the message by including its hash.
   * The client listens for responses from the server while allowing the user to input messages.

**Security Considerations:**

* **Key Exchange**: The AES key used for encryption is pre-shared, which means it must be securely communicated between the client and server before starting communication. In real-world applications, you would typically use a more secure method for key exchange, such as Diffie-Hellman.
* **Hashing**: SHA-256 is used to ensure message integrity, preventing tampering with the message during transmission. However, this does not protect against replay attacks. To mitigate such attacks, timestamps or nonces could be included in the messages.
* **Error Handling**: The code assumes ideal conditions for communication. In practice, you would want to implement more robust error handling (e.g., handling broken connections or malformed data).