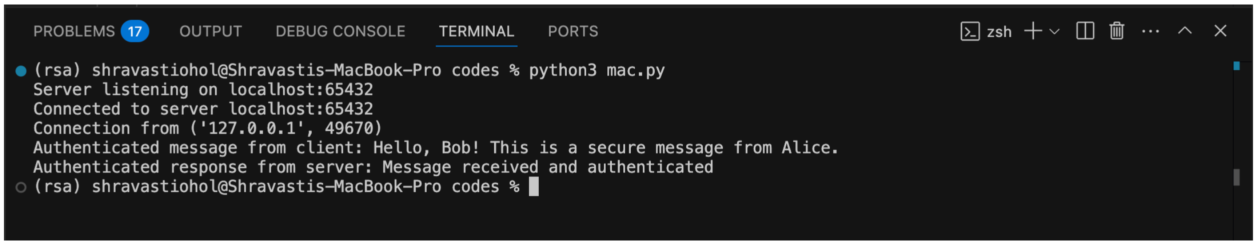
MAC and HMAC implementation

Code

```
import hashlib
import hmac
import socket
import threading
import base64

class SecureMessenger:
    def __init__(self, shared_secret_key):
        """
        Initialize the secure messenger with a shared secret key.

        :param shared_secret_key: Bytes representing the shared secret key
        """
        self.shared_secret_key = shared_secret_key

    def generate_hmac(self, message):
        """
        Generate HMAC for a given message using SHA-256.

        :param message: Message to authenticate
        :return: Base64 encoded HMAC
        """
        # Convert message to bytes if it's a string
        if isinstance(message, str):
            message = message.encode('utf-8')

        # Generate HMAC using SHA-256
        hmac_digest = hmac.new(
            key=self.shared_secret_key,
            msg=message,
            digestmod=hashlib.sha256
        ).digest()

        # Return base64 encoded HMAC for easy transmission
        return base64.b64encode(hmac_digest).decode('utf-8')

    def verify_hmac(self, message, received_hmac):
        """
        Verify the authenticity of a message by comparing HMACs.

        :param message: Original message
        :param received_hmac: HMAC received with the message
        :return: Boolean indicating message authenticity
        """
        # Convert message to bytes if it's a string
        if isinstance(message, str):
            message = message.encode('utf-8')

        # Decode the received HMAC
        received_hmac_bytes = base64.b64decode(received_hmac)

        # Compute the expected HMAC
        expected_hmac_bytes = hmac.new(
            key=self.shared_secret_key,
            msg=message,
            digestmod=hashlib.sha256
        ).digest()

        # Compare the HMACs
        return hmac.compare_digest(received_hmac_bytes, expected_hmac_bytes)

class Server:
    def __init__(self, host, port, shared_secret_key):
        """
        Initialize the server with networking and secure messaging capabilities.
        :param host: Server host address
        :param port: Server port number
        :param shared_secret_key: Shared secret key for HMAC
        """
        self.host = host
        self.port = port
        self.secure_messenger = SecureMessenger(shared_secret_key)
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind((self.host, self.port))
        self.server_socket.listen(1)

    def start(self):
        """
        Start the server and listen for incoming connections.
        """
        print(f"Server listening on {self.host}:{self.port}")

        # Accept client connection
        client_socket, address = self.server_socket.accept()
        print(f"Connection from {address}")

        try:
            while True:
                # Receive message from client
                received_data = client_socket.recv(1024).decode('utf-8')
                if not received_data:
                    break

                # Split received data into message and HMAC
                message, client_hmac = received_data.split('|HMAC|')

                # Verify the message's authenticity
                if self.secure_messenger.verify_hmac(message, client_hmac):
                    print(f"Authenticated message from client: {message}")

                    # Prepare response
                response = "Message received and authenticated"
                response_hmac = self.secure_messenger.generate_hmac(response)

                # Send response with HMAC
                client_socket.send(f"{response}|HMAC|{response_hmac}".encode('u
            else:
                print("Message authentication failed!")
                client_socket.send("Authentication failed".encode('utf-8'))

        except Exception as e:
            print(f"Error: {e}")
        finally:
            client_socket.close()

class Client:
    def __init__(self, host, port, shared_secret_key):
        """
        Initialize the client with networking and secure messaging capabilities.

        :param host: Server host address
        :param port: Server port number
        :param shared_secret_key: Shared secret key for HMAC
        """
        self.host = host
        self.port = port
        self.secure_messenger = SecureMessenger(shared_secret_key)
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def start(self):
        """
        Start the client and connect to the server.
        """
        self.client_socket.connect((self.host, self.port))
        print(f"Connected to server {self.host}:{self.port}")

        try:
            # Send a secure message to the server
            message = "Hello, Bob! This is a secure message from Alice."
            message_hmac = self.secure_messenger.generate_hmac(message)

            # Send message with HMAC
            self.client_socket.send(f"{message}|HMAC|{message_hmac}".encode('ut

            # Receive and verify server's response
            server_response = self.client_socket.recv(1024).decode('utf-8')

            if '|HMAC|' in server_response:
                response, server_hmac = server_response.split('|HMAC|')

                # Verify server's response
                if self.secure_messenger.verify_hmac(response, server_hmac):
                    print(f"Authenticated response from server: {response}")
                else:
                    print("Server response authentication failed!")
            else:
                print("Received response without HMAC")

        except Exception as e:
            print(f"Error: {e}")
        finally:
            self.client_socket.close()

def main():
    # Shared secret key (must be the same for both client and server)
    shared_secret_key = b'our_super_secret_key_123!'

    # Server and client configurations
    HOST = 'localhost'
    PORT = 65432

    # Demonstration of server and client
    def run_server():
        server = Server(HOST, PORT, shared_secret_key)
        server.start()

    def run_client():
        # Add a small delay to ensure server is ready
        import time
        time.sleep(1)
        client = Client(HOST, PORT, shared_secret_key)
        client.start()

    # Create threads for server and client
    server_thread = threading.Thread(target=run_server)
    client_thread = threading.Thread(target=run_client)

    # Start threads
    server_thread.start()
    client_thread.start()

    # Wait for threads to complete
    server_thread.join()
    client_thread.join()

if __name__ == "__main__":
    main()
```

Output :



```
(rsa) shravastiohol@Shravastis-MacBook-Pro codes % python3 mac.py
Server listening on localhost:65432
Connected to server localhost:65432
Connection from ('127.0.0.1', 49670)
Authenticated message from client: Hello, Bob! This is a secure message from Alice.
Authenticated response from server: Message received and authenticated
(rsa) shravastiohol@Shravastis-MacBook-Pro codes %
```