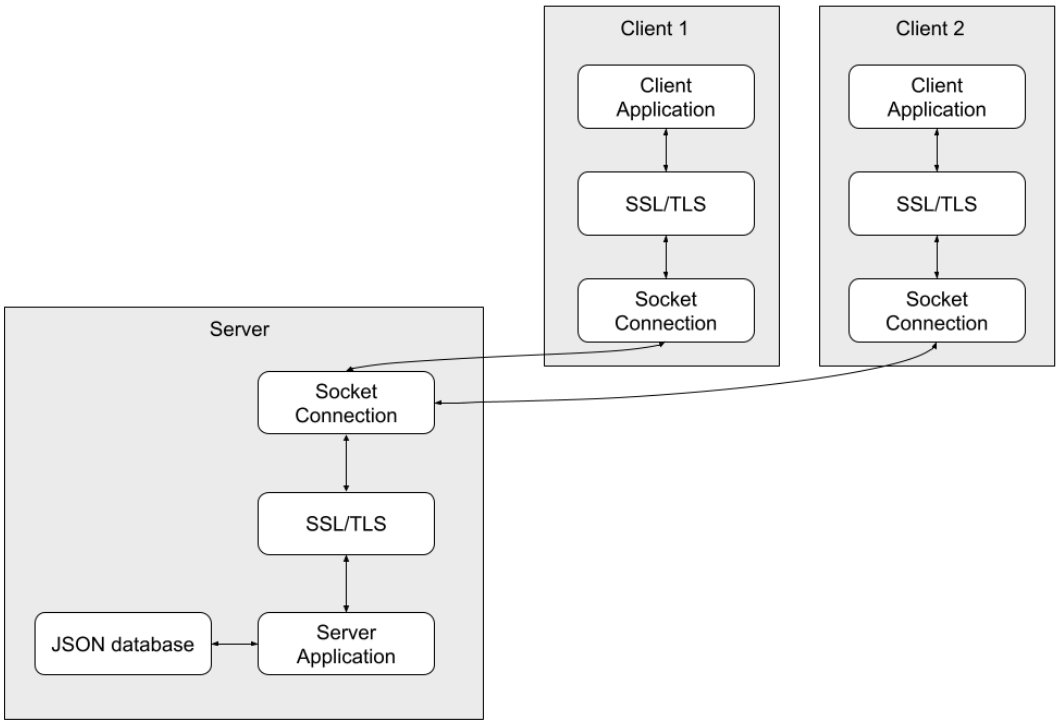
Introduction

This report outlines the development and architecture of a secure chat system designed for encrypted communication between clients and server. The system employs SSL/TLS protocols to safeguard data, uses authentication mechanisms for user verification, and demonstrates its effectiveness through network monitoring tools, Wireshark.

Architecture Description

The system follows a client-server architecture, where a central server manages multiple clients connecting through secure sockets.



The server-side consists of a Server Application responsible for managing all server operations. This includes authentication, where it oversees user registration by writing to a JSON database and login verification by reading from it. When a client attempts to connect, the server saves the client's socket and waits for successful authentication. Once authenticated, clients can send messages to the server, which then relays these messages to all other connected clients through their respective sockets. The SSL/TLS Layer ensures that all communications are securely encrypted. Additionally, the server relies on a JSON database to store usernames and passwords, while socket connections facilitate encrypted communication between the server and clients.

On the client side, the Client Application manages the logic for sending and receiving messages as well as user interactions. After a connection is established and authentication is complete, the Thread Management system comes into play, with one thread dedicated to handling user inputs and sending messages, while another thread is responsible for receiving messages from the server. The SSL/TLS Layer handles secure, encrypted communication with the server through sockets, maintaining confidentiality throughout the interaction.

Encryption Description

The chat system uses SSL/TLS with AES128-SHA between the server and clients.

1. SSL/TLS Protocol: The Secure Socket Layer (SSL) or Transport Layer Security (TLS) establishes an encrypted link between the client and the server. The content is set to ssl.PROTOCOL_TLSv1_2, which is a widely supported protocol for secure communication.
2. AES-128-SHA Cipher: The encryption algorithm chosen is Advanced Encryption Standard with a 128-bit key (AES128-SHA), which provides a balance between security and performance. AES-128 is a symmetric key encryption algorithm known for its efficiency and security.

Server-Side Encryption Code Explanation

```
# Configure SSL context
self.context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
self.context.load_cert_chain(certfile="cert.pem", keyfile="cert.pem")
self.context.load_verify_locations('cert.pem')
self.context.set_ciphers('AES128-SHA')
```

1. The first line sets up the SSL context with the TLS version 1.2 protocol for establishing secure connections.
2. The second line loads a certificate and its corresponding private key from the file named “cert.pem”.
3. Next line set up the certificate verification location using the same certificate.
4. The last line specifies that the server will use AES-128 for encrypting data and SHA for message integrity checks. This provides a balance of security and performance.

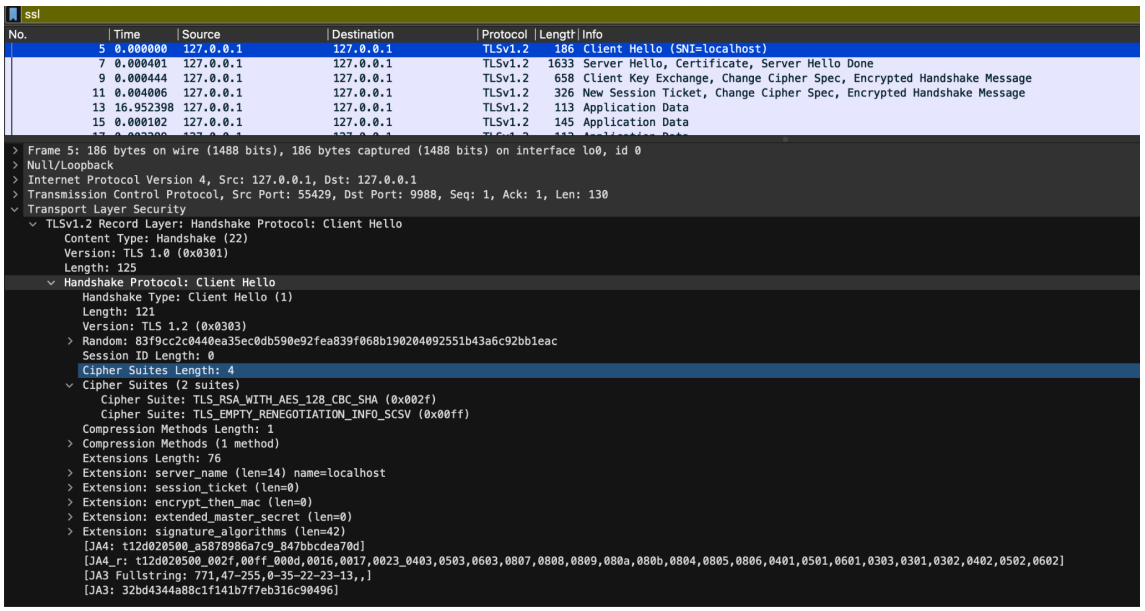
Client-Side Encryption Code Explanation

```
# Set up an SSL context with a specific protocol and cipher
self.context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
self.context.set_ciphers('AES128-SHA')
```

1. Similar to the server, the client creates an SSL context using the TLS 1.2 protocol for secure communication between the client and server.
2. The last line select the cipher suite to match the server’s configuration (AES-128 with SHA)

Clarification of Wireshark Results

I used Wireshark to capture the network traffic and confirm the encryption of the messages between clients and server. The captured data shows that all communication between the server and clients is encrypted.



1. TLS Handshake Initialisation (Frame 5)
   The client initialises a TLS handshake with a “Client Hello” message, indicating support for TLSv1.2. The packet includes the Server Name Indication (SNI) extension with the hostname set to localhost.

2. Server Response (Frame 7)
   The server responds to the clients “Hello” message with its own “Server Hello” message, which includes a server certificate. The server then completes its part of the handshake with a “Server Hello Done” message.

3. Client Key Exchange and Cipher Spec (Frame 9)
   The client sends its part of the key exchange information and indicates a change in the cipher specification, informing the server that all subsequent communications will be encrypted.

4. Server Confirmation (Frame 11)
   The server acknowledges the client’s key exchange and confirms the cipher spec change. A new session ticket is issued to the client for session resumption, and the server sends an encrypted message to complete the handshake.

5. Encrypted Data Transfer (Frame 13)
   After completing the handshake, the client and server exchange encrypted data. The application data packets observed in the subsequent frames indicate ongoing secure communication between the client and server.

Summary

This secure chat system successfully achieves the goal of providing an encrypted communication platform using SSL/TLS with the AES128-SHA cipher. It includes user authentication to prevent unauthorised access, and the use of Wireshark confirms that all messages are securely transmitted. This report demonstrates the architecture, encryption method, and verification process used to ensure data security.