Project Report: Secure Chat System using Mutual TLS (mTLS)

This project was undertaken as part of the
Sasken Summer Internship Program – 2025.

C. V. Raman Global University, Odisha, Bhubaneswar
By Team T1

NAME	Mail id
Anshuman Mahapatra	mahapatraanushuman156@gmail.com
Aradhana Ashajyoti	aradhanaashajyoti@gmail.com
Stuti	stuti0105@gmail.com
Subrat Upadhyay	subrat.upadhayay123@gmail.com

TABLE OF CONTENTS

Abstract

In the digital age, secure communication is critical to protect users from threats such as data breaches, identity theft, and unauthorized surveillance. This project presents the design and implementation of a **Secure Chat System** developed using **C++17** and **Mutual TLS (mTLS)**. The system ensures real-time, encrypted, and authenticated communication between multiple clients and a central server. By leveraging **OpenSSL**, both server and client verify each other's digital identities through certificate-based authentication, significantly reducing the risk of man-in-the-middle attacks.

The chat server handles multiple clients concurrently using multithreading and broadcasts messages to all authenticated participants. The client verifies the server's identity, establishes a secure connection, and communicates over an encrypted channel. In addition to mTLS, a secondary manual password check is implemented to add another layer of security. Development and deployment are done in a Linux environment, utilizing tools like GDB, Valgrind, and Wireshark for debugging and testing.

This project not only strengthens the understanding of secure communication protocols but also demonstrates the practical use of system-level programming and cryptography. The solution is scalable and adaptable, with future enhancements including GUI support, dynamic authentication, and file sharing. It serves as a strong prototype for secure enterprise communication systems.

Design

The design of the Secure Chat System is centered around a robust **client-server architecture** that ensures secure, real-time communication through **Mutual TLS (mTLS)**. The server acts as the central communication hub, managing multiple client connections, authenticating them using digital certificates, and broadcasting messages securely. Each client application initiates a **TLS handshake**, during which both the client and the server exchange and verify certificates signed by a trusted **Certificate Authority (CA)**. This bi-directional verification is crucial to prevent unauthorized access and ensure a trusted communication channel.

The system is designed for deployment on **Linux**, leveraging its native socket programming capabilities. The **OpenSSL** library is integrated for cryptographic operations, including certificate management and TLS session handling. To support concurrency, the server employs **multi-threading**, allowing multiple authenticated clients to communicate simultaneously without interruption.

The code is modular, with distinct components for the client, server, and build configuration (Makefile). Security is further enhanced through a **manual password verification** mechanism in addition to certificate-based authentication. The overall design emphasizes **security, scalability, and modularity**, making it both a reliable educational tool and a practical prototype for enterprise-level secure messaging platforms.

Debugging Tools

Effective debugging is essential when developing secure, system-level applications like the Secure Chat System using Mutual TLS (mTLS). This project required rigorous testing and analysis to ensure secure, reliable, and stable operation under various network conditions. A range of powerful debugging tools was utilized throughout the development lifecycle to diagnose issues related to memory management, socket handling, TLS handshakes, and certificate validation.

GDB (GNU Debugger) was employed to trace program execution, analyze segmentation faults, and inspect runtime variables, especially during complex interactions between client and server threads. **Valgrind** played a key role in detecting memory leaks, buffer overflows, and improper memory usage—common pitfalls in C++ system programming.

For debugging cryptographic operations and verifying digital certificates, **OpenSSL command-line tools** were extensively used. These tools helped simulate handshakes, verify certificate chains, and ensure correct encryption protocols were applied. Additionally, **Wireshark**, a network protocol analyzer, was instrumental in monitoring packet-level TLS communications. It allowed the team to confirm that data transmission was fully encrypted and that the TLS handshake completed correctly.

Collectively, these tools ensured that the system was not only functionally correct but also met stringent security and performance standards expected in secure communication applications.

Requirement

Before the implementation of the Secure Chat System using Mutual TLS (mTLS), it was crucial to identify both **functional** and **non-functional requirements** to guide development and ensure the final product met its intended objectives. The primary **functional requirements** included the ability to establish real-time communication between clients and a server, mutual certificate-based authentication using mTLS, secure message transmission through encrypted channels, and the capability to broadcast messages to all connected clients.

The **non-functional requirements** focused on system behavior, performance, and environment. The project was to be developed in **C++17**, offering advanced language features suited for robust and efficient system programming. It required **Linux** as the operating environment to leverage its powerful networking stack and command-line tools. The **OpenSSL** library was selected for implementing the TLS protocol, managing certificates, and handling cryptographic operations. To automate and manage the build process, a **Makefile** was used.

Performance expectations included low-latency message delivery, support for concurrent client connections, and strong security protocols. Portability, scalability, and maintainability were also prioritized. By clearly defining these requirements at the outset, the project maintained a focused development process and ensured that both security and usability were adequately addressed in the final implementation.

Tools Used for test cases

Testing a secure communication system like the Secure Chat System using Mutual TLS (mTLS) involves not just verifying functionality, but also validating **security, reliability, and resilience** under various conditions. This project employed a combination of **manual testing techniques** and **industry-standard tools** to ensure thorough evaluation of the system's behavior and security posture.

The **OpenSSL command-line utilities** were essential for generating, inspecting, and validating digital certificates used during mutual TLS authentication. These tools enabled simulation of various certificate scenarios such as expired certificates, invalid certificate chains, and mismatched CA signings. To test the system's response to raw network traffic and edge cases, **Netcat (nc)** and **Telnet** were used, helping developers assess socket-level behavior and manual connection attempts.

For inspecting real-time encrypted traffic and TLS handshakes, **Wireshark** proved invaluable. It allowed the team to capture and analyze packet flows, verify the presence of encrypted communication, and confirm proper handshake completion. Additionally, various **manual test scripts and case scenarios** were developed to simulate valid and invalid client behaviors, ensuring robust error handling and authentication enforcement.

These testing tools and strategies collectively verified that the system met its design goals in terms of **security, correctness, and stability**.

Implementation

The implementation of the Secure Chat System using Mutual TLS (mTLS) brought together the concepts of secure communication, system-level programming, and cryptographic protocols into a cohesive, functioning application. Developed using C++17, the system leveraged modern language features to enhance code clarity, maintainability, and performance. The core components included a multithreaded server (secure_chat_server.cpp) and a client application (secure_chat_client.cpp), both built for a Linux environment.

At the heart of the implementation was the OpenSSL library, which enabled the integration of TLS protocols, management of digital certificates, and execution of secure handshakes. During the connection phase, both client and server performed a mutual TLS handshake, exchanging certificates signed by a trusted Certificate Authority (CA). Only after successful verification did the system allow message exchange.

Once the secure session was established, all messages were encrypted and transmitted through the TLS tunnel. The server used threading to handle multiple clients concurrently and broadcast incoming messages to all authenticated clients. A Makefile automated the build process, managing dependencies and ensuring consistent compilation across systems.

This phase translated theoretical security principles into a fully working system, demonstrating not only secure authentication but also real-time, encrypted, and scalable communication between users.

Learning outcomes

- **Mutual TLS Authentication**

Gained hands-on experience in implementing mTLS, where both the client and server validate each other’s identity using digital certificates. This concept is widely used in secure APIs, enterprise applications, and cloud platforms.

- **OpenSSL Integration**

Learned to generate, manage, and validate certificates using OpenSSL. This included working with Certificate Authorities (CA), key pairs, and configuring secure TLS sockets in C++.

- **Linux System Programming**

Strengthened knowledge in Linux networking concepts such as sockets, threading, and command-line tooling. The use of Makefiles, GDB, and shell scripts added to the practical understanding of low-level programming.

- **Thread Management**

Built a multithreaded server capable of handling multiple secure client connections concurrently. This required learning about thread creation, synchronization, and resource sharing in C++.

- **Debugging and Testing in Security Context**

Developed skills in debugging secure programs using GDB, Valgrind, and Wireshark. These tools helped ensure memory safety and confirmed that all messages were encrypted and transmitted securely.

- **Project Planning and Documentation**

Understood the importance of planning architecture, defining clear requirements, and documenting each step of implementation and testing—essential for real-world software development and collaboration.

Summary

The Secure Chat System using Mutual TLS (mTLS) represents a complete and practical implementation of secure communication principles using modern system-level programming techniques. The project successfully integrates C++17, OpenSSL, and Linux-based development tools to create a real-time chat application where each participant is authenticated and all communication is encrypted.

At its core, the project leverages Mutual TLS (mTLS), a security mechanism where both the client and server must present and validate certificates issued by a trusted Certificate Authority (CA). This ensures not only confidentiality but also bi-directional authentication, preventing man-in-the-middle attacks and unauthorized access.

The server, built with support for multithreading, manages concurrent connections efficiently and broadcasts messages securely to all connected clients. The client application validates the server, establishes a secure session, and exchanges messages via a TLS-encrypted tunnel. An added manual password check introduces another layer of access control.

Development and testing were performed on a Linux environment, utilizing tools like GDB, Valgrind, Wireshark, and OpenSSL for debugging, verification, and performance evaluation.

This project demonstrates how theoretical concepts in cybersecurity and network programming can be transformed into a functional, real-world application. It lays a solid foundation for future enhancements, such as GUI support, advanced authentication, and secure file transfers.

CONCLUSION

The development of the **Secure Chat System using Mutual TLS (mTLS)** marks a successful application of secure communication protocols, system-level programming, and cryptographic technologies. The project's core achievement lies in establishing a **mutually authenticated and encrypted communication channel** between clients and a server, ensuring both **data confidentiality** and **identity verification**.

By integrating **OpenSSL** into a C++ application, the project demonstrates how secure sockets and certificate-based trust models can be practically applied. The implementation showcases a complete mTLS workflow—from certificate generation to TLS handshakes and encrypted message exchange. It further strengthens the security model through the addition of a **manual password layer**, making unauthorized access significantly more difficult.

The use of **multi-threading** on the server allows simultaneous handling of multiple client connections without compromising performance or security. The system is designed to be modular, scalable, and ready for future upgrades.

Overall, the project not only meets its functional and security goals but also provides valuable learning in secure coding, Linux networking, and real-world debugging practices. It serves as a strong foundation for building more advanced systems involving **secure file transfers, GUI interfaces, and multi-factor authentication**, making it relevant for both academic learning and industry applications.