This Project was undertaken as a part of Sasken Summer Internship Program-2025.

C. V. Raman Global University, Bhubaneswar

Odisha-752054

A CASE STUDY REPORT
ON

SECURE CHAT SYSTEM USING MUTUAL TLS

SUBMITTED BY(Team-1):

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 today's digital era, secure communication is essential to safeguard users against threats such as data breaches, identity theft, and unauthorized surveillance. This project introduces the design and implementation of a **Secure Chat System** built with **C++17** and **Mutual TLS (mTLS)**. The system enables real-time, encrypted, and authenticated communication between multiple clients and a central server. By integrating **OpenSSL**, both the server and clients authenticate each other's digital certificates, effectively minimizing the risk of man-in-the-middle attacks.

The server manages multiple client connections concurrently using multithreading and broadcasts messages only to authenticated participants. On the client side, the server's identity is verified before establishing an encrypted communication channel. To further enhance protection, the system incorporates an additional manual password verification step.

This project not only deepens the understanding of secure communication protocols but also highlights the practical application of system-level programming and cryptography. The solution is designed to be scalable and extensible, with potential future improvements including a **graphical user interface (GUI)**, **dynamic authentication mechanisms**, and **secure file sharing**.

Overall, it demonstrates a robust prototype for secure enterprise communication systems.

REQUIREMENT

Before implementing the Secure Chat System with Mutual TLS (mTLS), it was essential to define both **functional** and **non-functional requirements** to guide development and ensure the system achieved its objectives.

Functional Requirements

- Establishing real-time communication between clients and the server.
- Enforcing **mutual certificate-based authentication** using mTLS.
- Ensuring **secure message transmission** via encrypted channels.
- Broadcasting messages to all authenticated clients.

Non-Functional Requirements

The non-functional requirements focused on system performance, reliability, and environment setup:

- **Programming Language:** C++17 was chosen for its advanced features and efficiency in system-level programming.
- **Operating Environment:** Linux was selected to leverage its strong networking stack and command-line utilities.
- **Cryptographic Library:** OpenSSL was used to implement TLS, manage digital certificates, and handle encryption.
- **Build System:** A Makefile automated the build process for consistency and ease of maintenance.

Performance & Quality Goals

- Low-latency communication with support for multiple concurrent clients.
- Strong adherence to security protocols.
- Portability, scalability, and maintainability as core design principles.

DEBUGGING TOOLS

Debugging was an essential part of the development process for the **Secure Chat System using Mutual TLS (mTLS)**. Since the system integrates cryptography, networking, and multithreading, careful debugging was required to ensure correct operation, secure communication, and efficient performance. The main focus areas included memory management, socket operations, TLS handshakes, and certificate validation.

GDB (GNU Debugger) played a central role in analyzing program execution. It was used to step through the code, set breakpoints, and inspect runtime variables to identify logic errors and threading issues. It was particularly useful in diagnosing segmentation faults and synchronization problems when handling multiple concurrent clients.
Valgrind was extensively applied to detect hidden memory-related issues. It helped locate memory leaks, buffer overflows, and invalid memory accesses — common pitfalls in C++ system-level programming. Eliminating these problems improved both the stability and efficiency of the system.

For debugging cryptographic operations, **OpenSSL command-line utilities** were leveraged. These tools allowed verification of certificate chains, simulation of TLS handshakes, and confirmation that the correct encryption algorithms and protocols were being applied. This step was crucial in ensuring mutual authentication between clients and the server worked as intended.

Together, these debugging efforts ensured that the Secure Chat System was reliable, secure, and able to perform consistently under different network scenarios. By identifying and resolving low-level issues early, the project maintained both functional correctness and the high security standards required for a secure communication system.

SYSTEM DESIGN

The design of the Secure Chat System is based on a reliable client–server architecture that enables secure, real-time communication using Mutual TLS (mTLS). The server functions as the central hub, responsible for managing multiple client connections, authenticating them with digital certificates, and broadcasting messages across the network in a secure manner.

Each client initiates a TLS handshake when connecting to the server. During this process, both the client and the server exchange and validate digital certificates issued by a trusted Certificate Authority (CA). This two-way verification ensures that only authorized participants gain access, effectively preventing impersonation and unauthorized intrusion.

The system is deployed in a Linux environment, which provides powerful networking support through socket programming. OpenSSL is integrated to handle all cryptographic operations, including certificate validation and TLS session management. To handle concurrent communication, the server uses multithreading, allowing multiple authenticated clients to exchange messages simultaneously without performance bottlenecks.

The codebase is designed in a modular structure, with separate components for the server, client, and build process. A Makefile is included to automate compilation and linking, ensuring portability and ease of maintenance. To further strengthen security, the system introduces a manual password verification step alongside certificate-based authentication, adding another layer of protection.
Overall, the design emphasizes security, scalability, and modularity. It provides a strong foundation for understanding secure communication protocols while serving as a practical prototype for enterprise-grade secure messaging solutions.

TESTING

Testing the Secure Chat System with Mutual TLS (mTLS) required more than verifying its basic functionality. The process focused on validating the system's security, reliability, and stability under different conditions to ensure it performed as expected in real-world scenarios. The OpenSSL command-line utilities were used extensively to generate, inspect, and validate the digital certificates employed in the mTLS process. They also helped simulate different authentication scenarios, such as expired or invalid certificates, and ensured the system correctly rejected unauthorized connections.

To monitor live communication, Wireshark was employed for packet-level inspection. It confirmed that all data exchanges were encrypted and that TLS handshakes between clients and the server were completed successfully, verifying the confidentiality and integrity of communication.

In addition to tool-based checks, manual testing scenarios were created to simulate both valid and invalid client behaviors. These included incorrect password attempts, unauthorized certificates, and multiple concurrent client connections. Such testing ensured that the system could handle errors gracefully while maintaining strict security enforcement.

Through these methods, the Secure Chat System was verified to meet its intended goals of secure authentication, encrypted communication, and reliable multi-client operation.

IMPLEMENTATION

The implementation of the **Secure Chat System with Mutual TLS (mTLS)** brought together secure communication principles, cryptographic protocols, and system-level programming into a working application. Developed in **C++17**, the project leveraged modern language constructs for better code organization, maintainability, and performance.

Core Components

- **Server (secure_chat_server.cpp)**: Acts as the central hub, handling multiple client connections using multithreading. It validates each client's certificate during the TLS handshake and manages broadcasting of messages to all authenticated participants.
- **Client (secure_chat_client.cpp)**: Initiates a secure connection to the server, verifies the server's certificate, and enables encrypted chat functionality.
- **Build Configuration (Makefile)**: Automates compilation, linking, and dependency handling, ensuring consistency across Linux systems.

Integration of OpenSSL

The **OpenSSL library** was central to the project, enabling TLS session management, certificate handling, and cryptographic operations. The system relied on certificates issued by a trusted **Certificate Authority (CA)** to achieve **mutual authentication**. During connection setup, the client and server exchanged certificates, which were validated before establishing a secure channel.

Authentication & Security

- **mTLS Authentication:** Ensured that only clients with valid certificates, signed by the CA, could access the system.
- **Secondary Password Verification:** Added a manual authentication step beyond certificates, strengthening access control.
- **TLS Tunnel:** Once authentication was complete, all communication occurred within an encrypted TLS channel, ensuring confidentiality and integrity of data.

Concurrency & Communication

The server used **multithreading** to handle concurrent client sessions efficiently. This allowed multiple clients to exchange messages in real time without interruptions. The broadcast mechanism ensured that messages from any authenticated client were securely delivered to all other connected clients.

Error Handling & Stability

Special attention was given to error handling, particularly for cases involving:

- Invalid or expired certificates.
- Failed password attempts.
- Dropped or unstable client connections.
  These conditions were handled gracefully to maintain the stability of the system.

Outcome

This phase translated theoretical security models into a **fully functional system**. The implementation demonstrated **real-time encrypted communication, robust authentication mechanisms, scalability through multithreading, and maintainability through modular design and automation**. The system serves as both an **educational tool** and a **prototype for enterprise-grade secure messaging platforms**.

LEARNING OUTCOMES

- **Mutual TLS (mTLS) Implementation:** Acquired hands-on experience in setting up certificate-based authentication, where both client and server validate each other's identities. This reinforced knowledge of how mTLS is applied in secure APIs, enterprise solutions, and cloud platforms.

- **OpenSSL Usage:** Learned to generate, configure, and validate digital certificates using OpenSSL, including working with Certificate Authorities (CA), managing key pairs, and establishing secure TLS sockets in C++.

- **Linux System Programming:** Enhanced understanding of Linux-based networking concepts such as sockets, threading, and process management. Gained practical experience with Makefiles, shell scripting, and command-line utilities, strengthening system-level programming skills.

- **Thread Management:** Implemented a multithreaded server capable of supporting multiple secure clients simultaneously, gaining insights into thread creation, synchronization, and shared resource handling in C++.

- **Debugging Secure Applications:** Improved debugging skills in a security context using GDB for runtime analysis, Valgrind for memory safety, and Wireshark for monitoring TLS traffic. These tools ensured that communication was encrypted and free from memory-related issues.

- **Project Planning & Documentation:** Understood the importance of well-defined requirements, modular architecture, and systematic documentation throughout implementation and testing—critical practices for real-world software development and collaboration.

SUMMARY

The Secure Chat System with Mutual TLS (mTLS) demonstrates a practical and complete implementation of secure communication principles using modern system-level programming techniques. By integrating C++17, OpenSSL, and Linux-based tools, the project successfully delivers a real-time chat application where every participant is authenticated and all communication remains encrypted.

At the core of the system lies Mutual TLS (mTLS), which requires both the client and server to present and validate certificates signed by a trusted Certificate Authority (CA). This mechanism guarantees confidentiality and ensures bi-directional authentication, effectively preventing man-in-the-middle attacks and blocking unauthorized access.

The server, designed with multithreading, efficiently manages multiple client connections and securely broadcasts messages to authenticated users. The client application validates the server's identity, establishes a TLS-encrypted session, and exchanges messages through the secure tunnel. A secondary manual password verification further strengthens access control.

Development and debugging were carried out in a Linux environment, using GDB, Valgrind, Wireshark, and OpenSSL to verify correctness, security, and performance.

Overall, this project illustrates how theoretical concepts from cybersecurity and network programming can be transformed into a working real-world solution. It provides a strong foundation for future improvements such as a graphical user interface (GUI), advanced authentication mechanisms, and secure file transfer capabilities.

CONCLUSION

The development of the Secure Chat System with Mutual TLS (mTLS) represents a successful application of secure communication protocols, system-level programming, and cryptographic techniques. Its primary achievement is the creation of a mutually authenticated and encrypted channel between clients and the server, ensuring both data confidentiality and identity verification.

By integrating OpenSSL into a C++17 application, the project demonstrates the practical use of secure sockets and certificate-based trust models. The implementation covers the complete mTLS workflow—including certificate generation, TLS handshakes, and encrypted message exchange. Security is further reinforced by introducing an additional manual password verification step, which makes unauthorized access significantly harder.

The server employs multithreading to manage multiple client connections efficiently, maintaining both performance and security. With its modular and scalable design, the system is well-prepared for future enhancements.

Overall, the project achieves its intended functional and security objectives while providing valuable hands-on experience in secure coding, Linux networking, and debugging practices. It also establishes a strong foundation for extending the system with features such as secure file sharing, graphical interfaces, and multi-factor authentication, making it relevant for both academic study and practical industry applications.