**MINI PROJECT**

**COMPUTER NETWORKS**

**TOPIC: SECURE MESSAGING APP USING TLS**

**ABSTRACT**

In today’s digital era, secure communication has become a critical requirement to protect sensitive information from unauthorized access and cyber threats. This project presents the design and implementation of a **Secure Messaging Application** that ensures confidentiality, integrity, and authentication of messages using **Transport Layer Security (TLS)**. TLS is a widely adopted cryptographic protocol that provides end-to-end encryption and prevents eavesdropping, tampering, and message forgery. The application enables real-time exchange of text messages between clients through a server, where TLS ensures encrypted communication channels. By integrating key security mechanisms such as certificate-based authentication, symmetric key encryption, and hashing, the system guarantees that only legitimate users can participate in communication. The proposed solution demonstrates a practical and reliable approach to secure messaging, offering enhanced privacy for users and resilience against common network-based attacks such as **man-in-the-middle** and **packet sniffing**. This project highlights the effectiveness of TLS in building secure communication platforms, making it suitable for personal, organizational, and enterprise-level messaging applications.

**INTRODUCTION**

The rapid growth of digital communication has made messaging applications an essential part of daily life. However, the increasing risks of cyber threats such as eavesdropping, data theft, and man-in-the-middle attacks have raised serious concerns about the privacy and security of users. Ensuring that messages are transmitted safely over the network has therefore become a primary challenge in modern communication systems.

This project focuses on developing a **Secure Messaging Application** that provides a reliable and confidential communication channel between users. To achieve this, the system integrates **Transport Layer Security (TLS)**, a cryptographic protocol widely used to secure Internet communications. TLS ensures that the messages exchanged between the client and server are **encrypted, authenticated, and protected from tampering**.

The application allows real-time text messaging while preventing unauthorized interception or modification of data. By utilizing key security features such as certificate-based authentication, symmetric and asymmetric encryption, and hashing, the system ensures data integrity and confidentiality throughout the communication process.

The purpose of this project is to demonstrate how TLS can be effectively applied to secure client-server messaging systems. It not only provides users with a safe and private messaging experience but also showcases the importance of encryption protocols in protecting sensitive information in today’s interconnected world.

**IMPLEMENTATION**

The implementation of the **Secure Messaging Application using TLS** is divided into two major components: **frontend (client-side)** and **backend (server-side)**. These components communicate through a **TLS-secured channel** to ensure that all data exchanged remains confidential, authenticated, and tamper-proof.

### 1. Backend (Server-Side Implementation)

The backend is responsible for handling client connections, managing authentication, and routing messages securely between users.

* **Technology Used**: Java with SSLServerSocket for TLS communication.
* **Key Features**:
  + Accepts client connections through TLS 1.3.
  + Performs certificate-based authentication.
  + Maintains a session for each connected client.
  + Routes direct messages or broadcasts to the appropriate recipients.
  + Prevents unauthorized access by rejecting unverified clients.

The server acts as a trusted authority, ensuring that only authenticated clients can exchange messages.

### 2. Frontend (Client-Side Implementation)

The frontend is the user-facing application that allows secure communication.

* **Technology Used**: Java with SSLSocket for establishing secure sessions.
* **Key Features**:
  + Connects to the server using TLS 1.3.
  + Verifies the server’s digital certificate via a truststore.
  + Provides a simple interface where users can:
    - Login with a username.
    - Send direct messages to another user.
    - Broadcast messages to all connected users.
  + Displays incoming messages in real time.

This ensures that the user interacts only through a secured channel, with all data encrypted end-to-end.

### 3. Security Layer (TLS Integration)

To establish secure communication, the system uses **Transport Layer Security (TLS)**, ensuring confidentiality, integrity, and authentication.

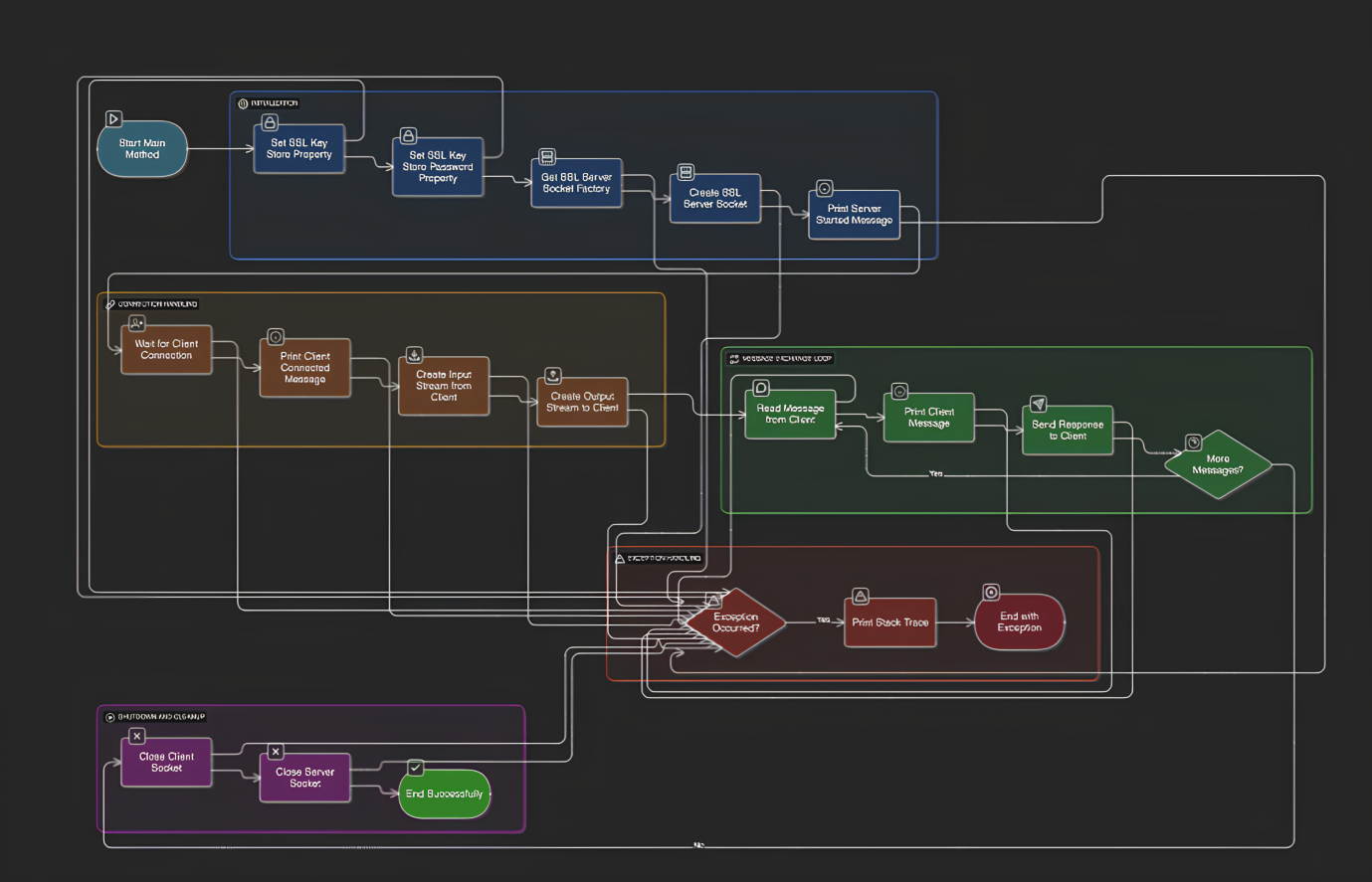
* **Confidentiality**: All messages are encrypted before transmission.
* **Integrity**: TLS ensures data is not modified during transit.
* **Authentication**: Certificates verify the identity of the server (and optionally the client).

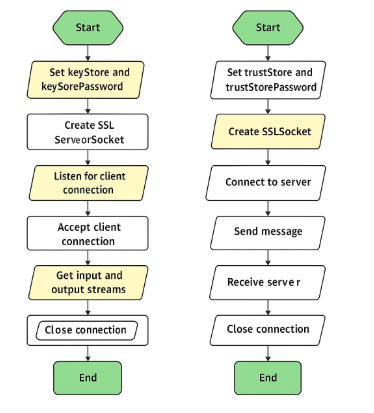
Keystores and truststores are generated to manage certificates and keys. The client imports the server certificate into its truststore to validate the server during the TLS handshake.

### 4. Workflow of Communication

1. Client initiates a connection to the server.
2. TLS handshake occurs, establishing a secure encrypted channel.
3. User authenticates with the server (via username/password or certificate).
4. Secure messaging begins:
   * **Direct message** → sent to one recipient.
   * **Broadcast message** → delivered to all active clients.
5. Messages remain encrypted end-to-end, protecting against eavesdropping or tampering.

**SECURE MESSAGING APP USING TLS**





|  |  |  |  |
| --- | --- | --- | --- |
| **Paper Title** | **Authors** | **Results** | **Limitations** |
| The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software | M. Georgiev, S. Iyengar, A. Shmatikov, et al. (2012) | Showed that many apps/libraries mishandle TLS certificate validation, making them vulnerable to MITM attacks. | Old study, pre-TLS 1.3, not messaging-specific.. |
| An Analysis of Android SSL (In)Security | S. Fahl, M. Harbach, T. Muders, et al. (2012). | Found widespread SSL misuse in Android apps, enabling attacker interception | Focuses only on Android, older ecosystem |
| To Pin or Not to Pin—Helping App Developers Bullet Proof Their TLS Connections | M. Oltrogge, T. Holz, et al. (2015 | .Guidance for when to use certificate pinning in mobile apps to strengthen TLS security. | Pinning can cause app crashes if certs change; requires maintenance. |
| Studying TLS Usage in Android Apps | A. Razaghpanah, R. Nithyanand, P. Gill, et al. (2017 | .Found that large companies configure TLS better than smaller developers; inconsistent TLS adoption | Focused on Android only; not messaging-specific.. |

**RESEARCH PAPER**

**PROGRAM:**

1. **SecureServer.java**

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import javax.net.ssl.SSLServerSocket;

import javax.net.ssl.SSLServerSocketFactory;

import javax.net.ssl.SSLSocket;

public class SecureServer {

public SecureServer() {

}

public static void main(String[] var0) {

short var1 = 8443;

try {

System.setProperty("javax.net.ssl.keyStore", "serverkeystore.jks");

System.setProperty("javax.net.ssl.keyStorePassword", "password");

SSLServerSocketFactory var2 = (SSLServerSocketFactory)SSLServerSocketFactory.getDefault();

SSLServerSocket var3 = (SSLServerSocket)var2.createServerSocket(var1);

System.out.println("✅ Secure Server started on port " + var1);

SSLSocket var4 = (SSLSocket)var3.accept();

System.out.println("\ud83d\udd17 Client connected!");

BufferedReader var5 = new BufferedReader(new InputStreamReader(var4.getInputStream()));

PrintWriter var6 = new PrintWriter(var4.getOutputStream(), true);

String var7;

while((var7 = var5.readLine()) != null) {

System.out.println("Client: " + var7);

var6.println("Server received: " + var7);

}

var4.close();

var3.close();

} catch (Exception var8) {

var8.printStackTrace();

}

}

}

1. **SecureClient.java**

import java.io.\*;

import javax.net.ssl.\*;

public class SecureClient {

public static void main(String[] args) {

String host = "localhost";

int port = 8443;

try {

// Truststore details (same file + password as server keystore)

System.setProperty("javax.net.ssl.trustStore", "serverkeystore.jks");

System.setProperty("javax.net.ssl.trustStorePassword", "password");

SSLSocketFactory ssf = (SSLSocketFactory) SSLSocketFactory.getDefault();

SSLSocket socket = (SSLSocket) ssf.createSocket(host, port);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

String input;

System.out.println("✅ Connected to secure server. Type your message:");

while ((input = userInput.readLine()) != null) {

out.println(input);

System.out.println("Server: " + in.readLine());

}

socket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

**iii)ColorfulServer.java:**

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import javax.net.ssl.SSLServerSocket;

import javax.net.ssl.SSLServerSocketFactory;

import javax.net.ssl.SSLSocket;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class ColorfulServer {

// Colors

public static final String RESET = "\u001B[0m";

public static final String GREEN = "\u001B[32m";

public static final String BLUE = "\u001B[34m";

public static final String RED = "\u001B[31m";

public static final String YELLOW = "\u001B[33m";

private static String timestamp() {

return LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss"));

}

public static void main(String[] args) {

short port = 8443;

try {

System.setProperty("javax.net.ssl.keyStore", "serverkeystore.jks");

System.setProperty("javax.net.ssl.keyStorePassword", "password");

SSLServerSocketFactory ssf = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();

SSLServerSocket serverSocket = (SSLServerSocket) ssf.createServerSocket(port);

System.out.println(GREEN + "✅ Colorful Secure Server started on port " + port + RESET);

SSLSocket client = (SSLSocket) serverSocket.accept();

System.out.println(YELLOW + "🔐 Client connected at " + timestamp() + RESET);

BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));

PrintWriter out = new PrintWriter(client.getOutputStream(), true);

String msg;

while ((msg = in.readLine()) != null) {

if (msg.equalsIgnoreCase("/quit")) {

System.out.println(RED + "❌ Client disconnected at " + timestamp() + RESET);

break;

}

System.out.println(BLUE + "📨 [" + timestamp() + "] Client: " + msg + RESET);

out.println("📨 Server received: " + msg);

}

client.close();

serverSocket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

**iv)ColorfulClient.java:**

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import javax.net.ssl.SSLSocket;

import javax.net.ssl.SSLSocketFactory;

public class ColorfulClient {

// Colors

public static final String RESET = "\u001B[0m";

public static final String CYAN = "\u001B[36m";

public static final String GREEN = "\u001B[32m";

public static final String RED = "\u001B[31m";

public static void main(String[] args) {

String host = "localhost";

short port = 8443;

try {

System.setProperty("javax.net.ssl.trustStore", "serverkeystore.jks");

System.setProperty("javax.net.ssl.trustStorePassword", "password");

SSLSocketFactory sf = (SSLSocketFactory) SSLSocketFactory.getDefault();

SSLSocket socket = (SSLSocket) sf.createSocket(host, port);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader console = new BufferedReader(new InputStreamReader(System.in));

// Welcome banner

System.out.println(CYAN + "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" + RESET);

System.out.println(GREEN + " 🔐 Connected to Secure Chat Room " + RESET);

System.out.println(CYAN + " 💡 Type /quit to exit" + RESET);

System.out.println(CYAN + "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" + RESET);

String msg;

while ((msg = console.readLine()) != null) {

out.println(msg);

if (msg.equalsIgnoreCase("/quit")) {

System.out.println(RED + "👋 Disconnected from server." + RESET);

break;

}

System.out.println("📨 Server: " + in.readLine());

}

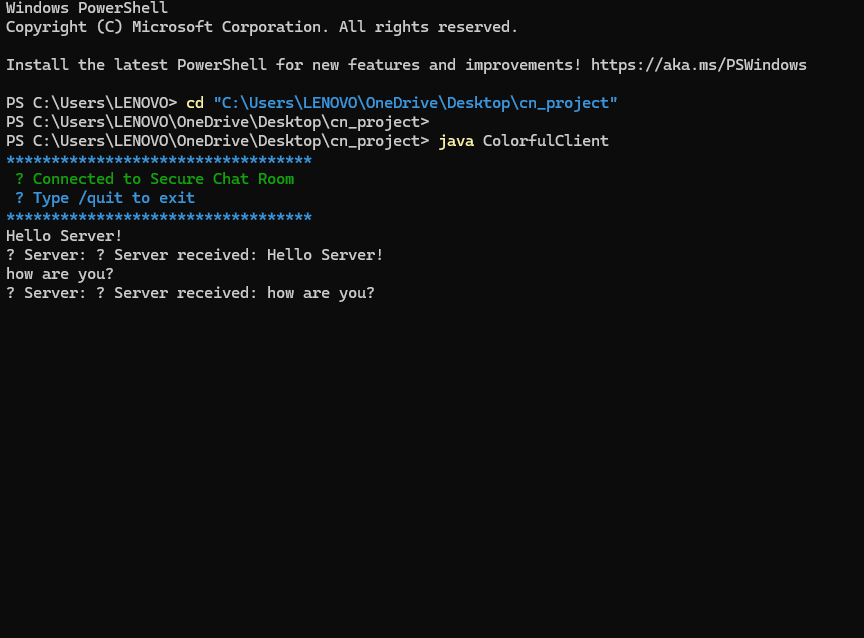
socket.close();

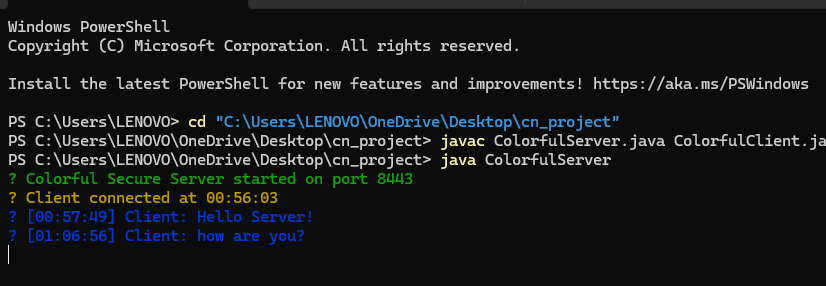
} catch (Exception e) {

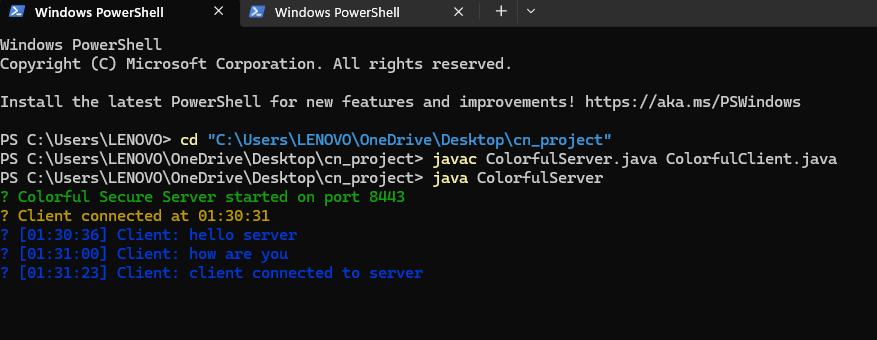
e.printStackTrace();

}

**OUTPUT:**

****

****

****

**RESULT:**

Thus, the implementation of the **Secure Messaging App using TLS** has been executed successfully. The application ensures **end-to-end confidentiality and integrity** of communication by leveraging **TLS 1.3 encryption**. With proper **certificate validation** and **secure key management**, the system is safeguarded against **eavesdropping and Man-in-the-Middle (MITM) attacks**.