keytool -genkeypair -alias mykey -keyalg RSA -keystore mykeystore.jks -keysize 2048

keytool -export -alias mykey -file mycert.cer -keystore mykeystore.jks

keytool -import -alias server-cert -file mycert.cer -keystore clienttruststore.jks

1. keytool -genkeypair -alias serverkey -keyalg RSA -keysize 2048 -validity 365 -keystore serverkeystore.jks -storepass password -keypass password -dname "CN=localhost, OU=MyOrg, O=MyCompany, L=MyCity, S=MyState, C=US"

2. keytool -exportcert -alias serverkey -keystore serverkeystore.jks -file servercert.cer -storepass password

3. keytool -importcert -alias servercert -file servercert.cer -keystore clienttruststore.jks -storepass truststorepassword -noprompt

a. SecureServer.java

import java.io.\*;

import java.security.KeyStore;

import javax.net.ssl.\*;

public class SecureServer {

public static void main(String[] args) throws Exception {

// Load the keystore

KeyStore keyStore = KeyStore.getInstance("JKS");

String keystorePath = "/home/sharat/Desktop/serverkeystore.jks";

try (FileInputStream keyStoreFis = new FileInputStream(keystorePath)) {

keyStore.load(keyStoreFis, "password".toCharArray());

}

// Initialize the KeyManagerFactory with the keystore

KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());

kmf.init(keyStore, "password".toCharArray());

// Initialize the SSLContext with the KeyManagers from the KeyManagerFactory

SSLContext sslContext = SSLContext.getInstance("TLS");

sslContext.init(kmf.getKeyManagers(), null, null);

// Create SSL server socket factory from SSLContext

SSLServerSocketFactory sslServerSocketFactory = sslContext.getServerSocketFactory();

// Create SSL server socket

int serverPort = 8443;

SSLServerSocket serverSocket = (SSLServerSocket) sslServerSocketFactory.createServerSocket(serverPort);

// Enable client authentication if needed

serverSocket.setNeedClientAuth(false);

System.out.println("Server started. Listening on port " + serverPort + "...");

// Server loop

while (true) {

// Accept client connection

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

// Handle client communication in a separate thread or process

new Thread(new ClientHandler(clientSocket)).start();

}

}

// Example client handler thread

static class ClientHandler extends Thread {

private SSLSocket clientSocket;

public ClientHandler(SSLSocket clientSocket) {

this.clientSocket = clientSocket;

}

@Override

public void run() {

try {

// Read from and write to the client socket as needed

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));

// Example: Echo server functionality

String line;

while ((line = reader.readLine()) != null) {

System.out.println("Received: " + line);

writer.write(line + "\n");

writer.flush();

}

// Close client socket

clientSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

b. SecureClient.java

import javax.net.ssl.\*;

import java.io.\*;

import java.security.KeyStore;

public class SecureClient {

public static void main(String[] args) throws Exception {

// Load the truststore

KeyStore trustStore = KeyStore.getInstance("JKS");

String keystorePath = "/home/sharat/Desktop/clienttruststore.jks";

try (FileInputStream trustStoreFis = new FileInputStream(keystorePath)) {

trustStore.load(trustStoreFis, "truststorepassword".toCharArray());

}

// Initialize the TrustManagerFactory with the truststore

TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());

tmf.init(trustStore);

// Initialize the SSLContext with the trust managers

SSLContext sslContext = SSLContext.getInstance("TLS");

sslContext.init(null, tmf.getTrustManagers(), null);

// Create an SSLSocketFactory from the SSLContext

SSLSocketFactory factory = sslContext.getSocketFactory();

// Create an SSLSocket connected to the specified host and port

SSLSocket socket = (SSLSocket) factory.createSocket("localhost", 8443);

// Initialize output and input streams for communication

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

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

// Send a message to the server

out.println("Hello, secure server!");

// Read and print the server's response

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

// Close the streams and the socket

out.close();

in.close();

socket.close();

}

}

c. SecureClientEventHandler.java

import javax.net.ssl.\*;

import java.io.\*;

import java.security.KeyStore;

public class SecureClient {

public static void main(String[] args) throws Exception {

// Load the truststore

KeyStore trustStore = KeyStore.getInstance("JKS");

String keystorePath = "/home/sharat/Desktop/clienttruststore.jks";

try (FileInputStream trustStoreFis = new FileInputStream(keystorePath)) {

trustStore.load(trustStoreFis, "truststorepassword".toCharArray());

}

// Initialize the TrustManagerFactory with the truststore

TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());

tmf.init(trustStore);

// Initialize the SSLContext with the trust managers

SSLContext sslContext = SSLContext.getInstance("TLS");

sslContext.init(null, tmf.getTrustManagers(), null);

// Create an SSLSocketFactory from the SSLContext

SSLSocketFactory factory = sslContext.getSocketFactory();

// Create an SSLSocket connected to the specified host and port

SSLSocket socket = (SSLSocket) factory.createSocket("localhost", 8443);

// Add a handshake completed listener to handle events after SSL handshake

socket.addHandshakeCompletedListener(event -> {

System.out.println("Handshake completed with: " + event.getSession().getPeerHost());

});

// Initialize output and input streams for communication

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

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

// Send a message to the server

out.println("Hello, secure server!");

// Read and print the server's response

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

// Close the streams and the socket

out.close();

in.close();

socket.close();

}

}

d. SecureclientSession.java

import javax.net.ssl.\*;

import java.io.\*;

import java.security.KeyStore;

public class SecureClient {

public static void main(String[] args) throws Exception {

// Load the truststore

KeyStore trustStore = KeyStore.getInstance("JKS");

String keystorePath = "/home/sharat/Desktop/clienttruststore.jks";

try (FileInputStream trustStoreFis = new FileInputStream(keystorePath)) {

trustStore.load(trustStoreFis, "truststorepassword".toCharArray());

}

// Initialize the TrustManagerFactory with the truststore

TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());

tmf.init(trustStore);

// Initialize the SSLContext with the trust managers

SSLContext sslContext = SSLContext.getInstance("TLS");

sslContext.init(null, tmf.getTrustManagers(), null);

// Create an SSLSocketFactory from the SSLContext

SSLSocketFactory factory = sslContext.getSocketFactory();

// Create an SSLSocket connected to the specified host and port

SSLSocket socket = (SSLSocket) factory.createSocket("localhost", 8443);

// Retrieve the SSLSession associated with the socket

SSLSession session = socket.getSession();

// Display session details

System.out.println("SSLSession Details:");

System.out.println("Cipher suite: " + session.getCipherSuite());

System.out.println("Protocol: " + session.getProtocol());

System.out.println("Session ID: " + session.getId());

System.out.println("Peer Host: " + session.getPeerHost());

System.out.println("Peer Port: " + session.getPeerPort());

// Initialize output and input streams for communication

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

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

// Send a message to the server

out.println("Hello, secure server!");

// Read and print the server's response

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

// Close the streams and the socket

out.close();

in.close();

socket.close();

}

}