**Your Name:**

**CMPSC222 Module 10: SSL Socket Programming (20 pts)**

This assignment covers communication using Java Secure Socket Extension (JSSE), focusing on understanding keys, certificates, TrustStore, and KeyStore. Certificates establish trust between parties, with KeyStore storing private keys and certificates for server identity verification, and TrustStore storing trusted certificates for server identity verification by clients. Utilize Java keytool, a JDK-provided utility, for generating key pairs, keystores, and certificates.

**Checking Environment Variable**

To run Java applications and execute Java-related commands from the command line on Windows, it's essential to set the JDK bin folder in the system's PATH environment variable. Typically, Java is installed in 'C:\Program Files\Java.' You can verify its presence there. Use 'java -version' to check if the JDK is recognized and correctly configured. If configured properly, it should display information about your Java version. You can view the contents of PATH, including the JDK bin folder path, using 'echo %path%'. Ensure that the JDK bin folder is included in PATH to avoid encountering errors when running programs.

A computer screen with white text

Description automatically generated

Open File Explorer and navigate to the JDK bin folder. Find “keytool.exe” within the JDK bin folder.

A screenshot of a computer

Description automatically generated

JDK installation is verified, environment variables are set correctly, and the presence of the keytool utility for SSL key creation is confirmed. We're simplifying our setup by using a single PC for both server and client programs, eliminating the need for multiple computers.

**Creating a keystore**

Create a new folder (e.g., yson for myself) on your desktop, then generate keys within it. Open the command prompt.

A screenshot of a computer

Description automatically generated

1. **Create the Keystore and generate a certificate, a syntax and example are shown below:**

# Syntax

keytool -genkey -keystore <keyname> -alias <alias> -keyalg RSA -keysize 2048 -validity 10000

# Example:

keytool -genkey -v -keystore **server.store** -alias **server** -keyalg RSA -keysize 2048 -validity 10000

This command generates a new RSA private key with an alias ‘server’ and stores it in the file ‘server.store’. The key has a size of 2048 bits and a validity period of 10000 days. After issuing the command, you will be prompted to enter a 6-digit keystore password (e.g., 123456 for myself).

A computer screen with white text

Description automatically generated

Please note that the password for accessing the keystore file will be used in the next few steps.

1. **Export a certificate from a Keystore:**

Once you have created a private key in a Java keystore file, you can export that private key to a certificate file using the Java ‘keytool export’ command.

The certificate is stored in the file that you specified.

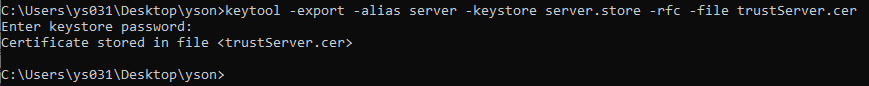
# Syntax

keytool -export -alias <alias> -keystore server -rfc -file <cerFilename>.cer

# example

# keytool -export -alias **server** -keystore **server.store** -rfc -file **trustServer.cer**

When executed, the command prompts for the keystore password, reads from the specified keystore file ('server.store'), looks for the alias 'server', and exports the public key to a new file named 'trustServer.cer' in RFC format.



1. **Import the certificate into a TrustStore:**

To work with Java’s public and private keys, you may import a certificate into your public key keystore using the Java keytool command. For example, if you have a certificate file named 'trustServer.cer' with an alias 'trustServe' and password 'changeit', you can import it into the 'cacerts.store' keystore with the following command:

# Syntax, “changeit” is a default password

keytool -importcert -alias <alias> -file <cerFileName>.cer -keystore <cacerts path> -storepass changeit

Ensure to replace ‘<cacerts\_path>’ with the path to your 'cacerts.store' file. Here are examples:

keytool -importcert -alias **trustServe** -file **trustServer.cer** -keystore “**C:\Program Files\Jana\jdk-19\jre\lib\security\cacerts.store”** –storepass **changeit**

where, the ‘changeit’ password is a default option.

# Import the certificate on your current folder

keytool -importcert -alias **trustServe** -file **trustServer.cer** -keystore **cacerts.store** –storepass changeit

A computer screen with white text

Description automatically generated

**Practice:**

This is a client-server program designed to capitalize all characters sent from the client, following outlined procedures for both the server and the client.

**Server side:**

1. Load both truststore and keystore using System.setProperty() method:

System.setProperty("javax.net.ssl.keyStore","c:\\Users\\ys031\\Desktop\\yson\\server.store");

System.setProperty("javax.net.ssl.keyStorePassword","123456"); //My password is “123456”

System.setProperty("javax.net.debug","ssl");

1. Create a server socket.

SSLServerSocketFactory sslserversocketfactory = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();

SSLServerSocket sslserversocket = (SSLServerSocket) sslserversocketfactory.createServerSocket(9999);

1. Wait for a client connection.

SSLSocket sslsocket = (SSLSocket) sslserversocket.accept();

1. Process client requests: once a client connection is established, obtain input and output streams from the socket to facilitate communication. Read data from the input stream, process it, and send a response back through the output stream. Repeat this process in a loop to handle multiple requests.

BufferedReader inFromClient = new BufferedReader(new InputStreamReader(sslSocket.getInputStream()));

BufferedWriter outToClient = new BufferedWriter(new OutputStreamWriter(sslSocket.getOutputStream()));

String line = null;

while ((line = inFromClient.readLine()) != null)

{

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

String capitalizedSentence = line.toUpperCase();

outToClient.write(capitalizedSentence);

outToClient.newLine();

outToClient.flush();

}

1. Close input/output streams and sockets after communication.

inFromClient.close();

outToClient.close();

sslSocket.close();

sslserversocket.close();

Compile and run this server-side program.

**Client Side:**

1. Load both truststore and keystore using System.setProperty() method:

System.setProperty("javax.net.ssl.keyStore","c:\\Users\\ys031\\Desktop\\yson\\server.store");

System.setProperty("javax.net.ssl.keyStorePassword","123456"); //My password is “123456”

System.setProperty("javax.net.debug","ssl");

1. Create a new socket.

SSLSocketFactory sslsocketfactory = (SSLSocketFactory) SSLSocketFactory.getDefault();

SSLSocket sslSocket = (SSLSocket) sslsocketfactory.createSocket("localhost", 9999);

1. Send request to server and receive response from server.

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

BufferedReader inFromServer = new BufferedReader(new InputStreamReader(sslSocket.getInputStream()));

BufferedWriter outToServer = new BufferedWriter(new OutputStreamWriter(sslSocket.getOutputStream()));

String string = null;

while ((string = inFromUser.readLine()) != null)

{

outToServer.write(string + '\n');

outToServer.flush();

String modifiedSentence = inFromServer.readLine(); //Read line from server

System.out.println("\nReceived from server: " + modifiedSentence);

}

1. Close resources and sockets

inFromUser.close();

inFromServer.close();

outToServer.close();

sslSocket.close();

Ensure to set up your socketFactory instance with the correct keystore path and password using System.setProperty.

Compile and run this client-side program.

Type ‘hello world’ and see all characters capitalized in the output.

**Your Turn: Online Calculator**

* Develop an online calculator capable of basic arithmetic operations: addition, subtraction, multiplication, and division.
* Modify both server and client programs for the client to request arithmetic computations from the server.
* Upon receiving a request from the client, such as "10 + 20", the server computes and returns the result to the client.
* Implement the following interface for the calculator service:

public interface CalculatorService {

public double add(double firstValue, double secondValue);

public double sub(double firstValue, double secondValue);

public double div(double firstValue, double secondValue);

public double mul(double firstValue, double secondValue);

}

* Sample output: user input underlined.

Online Calculator

-----------------------

1. Addition
2. Subtraction
3. Division
4. Multiplication
5. Exit

Choice: 1

Enter First Number: 3

Enter Second Number: 5

2.0 + 4.0 = 6.0

**Grading Rubric:**

* (5 pts) Implement Interface
* (5 pts) Implement client program.
* (5 pts) Implement server program.
* (5 pts) Handling a menu.

**What to submit:**

* Compress all projects (both server and client) and save it as M10-yourfirstname.zip.
* Submit the zip file on Canvas.
* Keep a backup copy of the files.