



Tarea 4 - Transferencia de archivos con sockets seguros

25-03-2023

Pérez Antonio David
Boleta: 2019630589
Grupo: 4CV12
Prof: Carlos Pineda Guerrero

Objetivo.....	2
Desarrollo.....	2
Conversación 1 con Chat GPT.....	2
David Pérez Antonio.....	2
Chat GPT.....	3
David Pérez Antonio.....	6
Chat GPT.....	6
David Pérez Antonio.....	9
Chat GPT.....	9
David Pérez Antonio.....	11
Chat GPT.....	12
David Pérez Antonio.....	13
Chat GPT.....	13
David Pérez Antonio.....	15
Chat GPT.....	17
David Pérez Antonio.....	18
Chat GPT.....	22
David Pérez Antonio.....	22
Chat GPT.....	23
David Pérez Antonio.....	24
Chat GPT.....	24
David Pérez Antonio.....	24
Chat GPT.....	26
David Pérez Antonio.....	28
Chat GPT.....	31
David Pérez Antonio.....	32
Chat GPT.....	32
David Pérez Antonio.....	34
Chat GPT.....	35
Conversación 2 con Chat GPT.....	36
David Pérez Antonio.....	36
Chat GPT.....	39
Creando máquina Virtual en Azure.....	40
Llenando datos principales.....	40
Discos.....	41
Redes.....	41
Revisar y Crear.....	42
Creando.....	42



Abriendo el puerto 8000 que se va a ocupar.....	43
Dentro de Ubuntu.....	43
Descargamos Java 17.....	43
Configuramos un keystore y exportamos el certificado.....	44
Copiamos el certificado a una carpeta local.....	44
Copiamos el archivo fuente al servidor.....	45
Levantamos el servidor.....	45
Importamos el certificado para el Cliente PUT.....	46
Importamos el certificado para el Cliente GET.....	46
Probando el envío de un archivo.....	47
Antes de Enviar.....	50
Después de Enviar.....	50
Probando la recepción de un archivo.....	51
Archivo no existente.....	51
Archivo prueba.txt.....	51
Antes de Recibir.....	52
Después de recibir.....	52
Conclusiones.....	53

Objetivo

Realizar el envío y la recepción de archivos utilizando sockets seguros en java.

Desarrollo

Conversación 1 con Chat GPT

David Pérez Antonio

en Java crea un servidor multithread que utilice sockets, deberá ser capaz de recibir 2 tipos de peticiones GET y PUT

1. la petición GET: el servidor recibirá el tipo de petición GET, después recibirá el nombre del archivo. Si el archivo se pudo leer entonces responderá OK y a continuación enviará el tamaño del archivo, y después el contenido, en caso contrario responderá ERROR

2. la petición PUT: el servidor recibirá el tipo de petición PUT, después recibirá el nombre del archivo, después recibirá el tamaño del archivo, finalmente recibirá el contenido del archivo. Si el archivo se pudo escribir en el disco el servidor responderá OK, en caso contrario ERROR.

para el envío y recepción de datos a través del socket usa la clase Data Input y Output Stream

Chat GPT

Aquí está un ejemplo de cómo crear un servidor multithread en Java que utiliza sockets para recibir peticiones GET y PUT y responder en consecuencia:

java

Copy code

```
import java.io.*;  
import java.net.*;
```

```
public class MultiThreadedServer implements Runnable {
```

```
    Socket clientSocket;
```

```
public MultiThreadedServer(Socket clientSocket) {  
    this.clientSocket = clientSocket;  
}  
  
public static void main(String[] args) throws Exception {  
    ServerSocket serverSocket = new ServerSocket(8000);  
    System.out.println("Server running on port 8000");  
    while (true) {  
        Socket clientSocket = serverSocket.accept();  
        MultiThreadedServer server = new MultiThreadedServer(clientSocket);  
        Thread serverThread = new Thread(server);  
        serverThread.start();  
    }  
}  
  
@Override  
public void run() {  
    try {  
        DataInputStream inFromClient = new  
        DataInputStream(clientSocket.getInputStream());  
        DataOutputStream outToClient = new  
        DataOutputStream(clientSocket.getOutputStream());  
  
        String requestType = inFromClient.readUTF(); // Lee el tipo de petición  
        String fileName = inFromClient.readUTF(); // Lee el nombre del archivo  
  
        if (requestType.equals("GET")) {  
            try {  
                FileInputStream fileInput = new FileInputStream(fileName);  
                outToClient.writeUTF("OK"); // Envía la respuesta OK  
            }  
        }  
    }  
}
```

```
    outToClient.writeLong(fileInput.getChannel().size()); // Envía el tamaño del
archivo

    byte[] buffer = new byte[4096];
    int bytesRead;
    while ((bytesRead = fileInput.read(buffer)) != -1) {
        outToClient.write(buffer, 0, bytesRead); // Envía el contenido del archivo
    }

    fileInput.close();
} catch (FileNotFoundException e) {
    outToClient.writeUTF("ERROR"); // Envía la respuesta ERROR si el archivo no se
puede leer
}
} else if (requestType.equals("PUT")) {
    try {
        FileOutputStream fileOutput = new FileOutputStream(fileName);
        long fileSize = inFromClient.readLong(); // Lee el tamaño del archivo

        byte[] buffer = new byte[4096];
        int bytesRead;
        long totalBytesRead = 0;
        while (totalBytesRead < fileSize && (bytesRead = inFromClient.read(buffer, 0,
(int)Math.min(buffer.length, fileSize - totalBytesRead))) != -1) {
            fileOutput.write(buffer, 0, bytesRead); // Escribe el contenido del archivo en el
disco
            totalBytesRead += bytesRead;
        }

        fileOutput.close();
    }
}
```

```
if (totalBytesRead == fileSize) {  
    outToClient.writeUTF("OK"); // Envía la respuesta OK si se pudo escribir el  
    archivo en el disco  
}  
else {  
    outToClient.writeUTF("ERROR"); // Envía la respuesta ERROR si no se pudo  
    escribir el archivo en el disco  
}  
}  
}  
  
} catch (IOException e) {  
    outToClient.writeUTF("ERROR"); // Envía la respuesta ERROR si no se pudo  
    escribir el archivo en el disco  
}  
}  
}  
  
inFromClient.close();  
outToClient.close();  
clientSocket.close();  
}  
}  
}  
}  
}
```

Este servidor escucha en el puerto 8000 y crea un nuevo hilo para cada cliente que se conecta. Cada hilo maneja una única conexión y se encarga de leer y escribir datos a través del socket. El servidor utiliza las clases DataInputStream y DataOutputStream para enviar y recibir datos en formato binario.

Cuando el servidor recibe una petición GET, intenta abrir el archivo solicitado y enviar su contenido al cliente

David Pérez Antonio

que debo modificar del código anterior para poder usar sockets seguros y definir yo la ruta y contraseña de mi keystore?

Chat GPT

Para usar sockets seguros y definir tu propia ruta y contraseña de keystore, debes hacer los siguientes cambios en el código anterior:

Cambiar la clase Socket por la clase SSLSocket para crear sockets seguros:

java

Copy code

// En lugar de:

```
Socket clientSocket = serverSocket.accept();
```

// Usar:

```
SSLSocketFactory sslSocketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
SSLSocket clientSocket = (SSLSocket) sslSocketFactory.createSocket(serverSocket.accept(),
null, serverSocket.getSoTimeout(), true);
```

Configurar el socket seguro antes de establecer la conexión:

java

Copy code

// Configurar el socket seguro

```
String keyStorePath = "/ruta/a/keystore.jks";
char[] keyStorePassword = "contraseña".toCharArray();
```

```
KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
keyStore.load(new FileInputStream(keyStorePath), keyStorePassword);
```

```
KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
```

```
keyManagerFactory.init(keyStore, keyStorePassword);

TrustManagerFactory trustManagerFactory =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
trustManagerFactory.init(keyStore);

SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(keyManagerFactory.getKeyManagers(),
trustManagerFactory.getTrustManagers(), new SecureRandom());

SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();

// Establecer la conexión
SSLSocket clientSocket = (SSLSocket) sslSocketFactory.createSocket(serverSocket.accept(),
null, serverSocket.getSoTimeout(), true);

Agregar los parámetros necesarios para la configuración del socket seguro:
java
Copy code
// Configurar el socket seguro
String keyStorePath = "/ruta/a/keystore.jks";
char[] keyStorePassword = "contraseña".toCharArray();
char[] keyPassword = "contraseña".toCharArray();

KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
keyStore.load(new FileInputStream(keyStorePath), keyStorePassword);

KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
keyManagerFactory.init(keyStore, keyPassword);
```

```
TrustManagerFactory trustManagerFactory =  
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());  
trustManagerFactory.init(keyStore);  
  
SSLContext sslContext = SSLContext.getInstance("TLS");  
sslContext.init(keyManagerFactory.getKeyManagers(),  
trustManagerFactory.getTrustManagers(), new SecureRandom());  
  
SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();  
  
// Establecer la conexión  
String[] enabledProtocols = {"TLSv1.2"};  
String[] enabledCipherSuites = {"TLS_RSA_WITH_AES_128_CBC_SHA256"};  
clientSocket.setEnabledProtocols(enabledProtocols);  
clientSocket.setEnabledCipherSuites(enabledCipherSuites);  
En este ejemplo, el código establece los protocolos TLSv1.2 y la suite de cifrado  
TLS_RSA_WITH_AES_128_CBC_SHA256 como los habilitados para el socket seguro. Puedes  
modificar estos valores según tus necesidades.
```

Es importante tener en cuenta que tanto el servidor como el cliente deben estar configurados para usar sockets seguros. Para hacerlo, debes seguir un proceso similar al anterior en ambos lados de la conexión.

David Pérez Antonio

Crea un ClientePUT usando sockets, que recibira como parametros la IP del servidor, el puerto y el nombre de un archivo.

El cliente deberá leer el archivo del disco local. Si el archivo pudo learse entonces deberá conectarse al servidor creado previamente usando sockets normales, el cliente enviará el tipo de petición PUT, después enviará el nombre del archivo, después enviará el tamaño del archivo y finalmente enviara el contenido del archivo; si el archivo no pudo ser leido deberá informar del error.

El cliente deberá esperar la respuesta del servidor, si el servidor responde OK el cliente deberá desplegar un mensaje indicando que el archivo fue recibido por el servidor con éxito, de otra manera desplegará un mensaje de error indicando que el servidor no pudo escribir el archivo en el disco local.

Chat GPT

Aquí tienes el código del cliente que cumple con los requerimientos descritos:

java

Copy code

```
import java.io.*;
```

```
import java.net.*;
```

```
public class ClientePUT {
```

```
    public static void main(String[] args) {
```

```
        // Leer los parámetros de entrada
```

```
        String ipServidor = args[0];
```

```
        int puerto = Integer.parseInt(args[1]);
```

```
        String nombreArchivo = args[2];
```

```
        // Leer el archivo del disco local
```

```
        File archivo = new File(nombreArchivo);
```

```
        byte[] archivoBytes = new byte[(int) archivo.length()];
```

```
        try (FileInputStream fis = new FileInputStream(archivo)) {
```

```
            fis.read(archivoBytes);
```

```
        } catch (IOException e) {
```

```
            System.out.println("Error al leer el archivo");
```

```
            return;
```

```
}
```

```
// Conectar con el servidor usando sockets normales
try (Socket socket = new Socket(ipServidor, puerto)) {

    // Enviar la petición PUT y los datos del archivo
    DataOutputStream out = new DataOutputStream(socket.getOutputStream());
    out.writeUTF("PUT");
    out.writeUTF(nombreArchivo);
    out.writeLong(archivoBytes.length);
    out.write(archivoBytes);

    // Esperar la respuesta del servidor
    DataInputStream in = new DataInputStream(socket.getInputStream());
    String respuesta = in.readUTF();

    // Desplegar el mensaje correspondiente
    if (respuesta.equals("OK")) {
        System.out.println("El archivo fue recibido por el servidor con éxito");
    } else {
        System.out.println("El servidor no pudo escribir el archivo en el disco local");
    }

} catch (UnknownHostException e) {
    System.out.println("Host desconocido: " + ipServidor);
} catch (IOException e) {
    System.out.println("Error al conectar con el servidor");
}

}
```



```
}
```

Este cliente se conecta al servidor especificado usando sockets normales. Luego envía la petición PUT y los datos del archivo al servidor, espera la respuesta del servidor y despliega el mensaje correspondiente según la respuesta recibida.

Es importante tener en cuenta que este código no utiliza sockets seguros. Si se requiere seguridad en la comunicación entre el cliente y el servidor, se deben seguir los pasos descritos en la respuesta anterior para configurar los sockets seguros.

David Pérez Antonio

el código anterior esta perfecto, pero podrias modificar el envio del archivo en la linea
`out.write(archivoBytes);`
 para que se envien de modo que coincida con la recepcion del servidor, que es la siguiente
`long fileSize = inFromClient.readLong(); // Lee el tamaño del archivo`

```
byte[] buffer = new byte[4096];
int bytesRead;
long totalBytesRead = 0;
while (totalBytesRead < fileSize && (bytesRead = inFromClient.read(buffer, 0,
    (int) Math.min(buffer.length, fileSize - totalBytesRead))) != -1) {
    fileOutput.write(buffer, 0, bytesRead); // Escribe el contenido del archivo en el
    disco
    totalBytesRead += bytesRead;
}
```

Chat GPT

¡Claro! Lo que podemos hacer es enviar el archivo en bloques de tamaño fijo, en lugar de enviarlo completo de una vez. Así el servidor podrá recibirlo en bloques del mismo tamaño



y escribirlos en el disco. Para ello, podemos modificar la sección del código que envía el archivo de la siguiente manera:

java

Copy code

```
// Enviar la petición PUT y los datos del archivo  
DataOutputStream out = new DataOutputStream(socket.getOutputStream());  
out.writeUTF("PUT");  
out.writeUTF(nombreArchivo);  
out.writeLong(archivoBytes.length);  
int bytesRead = 0;  
while (bytesRead < archivoBytes.length) {  
    int bytesToRead = Math.min(archivoBytes.length - bytesRead, 4096);  
    out.write(archivoBytes, bytesRead, bytesToRead);  
    bytesRead += bytesToRead;  
}
```

En este código, enviamos la petición PUT y los datos del archivo como antes. Sin embargo, luego usamos un bucle para enviar el archivo en bloques de 4096 bytes o menos. La variable bytesRead indica el número de bytes que hemos enviado hasta ahora, y la variable bytesToRead indica el número de bytes que podemos enviar en esta iteración del bucle.

De esta manera, el servidor podrá recibir el archivo en bloques del mismo tamaño y escribirlos en el disco, como se muestra en el código que proporcionaste.

Espero que esto te sea de ayuda. ¡Si tienes alguna otra pregunta o necesitas más ayuda, no dudes en preguntar!

David Pérez Antonio

Desarrollo de sistemas distribuidos

2 / 2

del código anterior que debo cambiar para usar Sockets Seguros y definir yo la ruta y contraseña de mi trustStore?

Chat GPT

Para usar sockets seguros en Java, debes crear un SSLContext y configurarlo con las propiedades de seguridad necesarias. En este caso, debemos usar un truststore para verificar el certificado del servidor. Para especificar la ruta y contraseña del truststore, puedes agregar las siguientes líneas de código:

java

Copy code

```
// Cargar el truststore con el certificado del servidor
String truststorePath = "ruta/al/truststore";
char[] truststorePassword = "contraseña".toCharArray();
KeyStore truststore = KeyStore.getInstance("JKS");
truststore.load(new FileInputStream(truststorePath), truststorePassword);

// Crear un TrustManagerFactory para verificar el certificado del servidor
TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(truststore);

// Crear un SSLContext con los TrustManagers configurados
SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(null, tmf.getTrustManagers(), null);

// Crear un SSLSocketFactory a partir del SSLContext
SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();
```

```
// Crear un socket seguro y conectarse al servidor  
Socket socket = sslSocketFactory.createSocket(servidor, puerto);
```

En este código, estamos cargando el truststore con el certificado del servidor, creando un TrustManagerFactory para verificar el certificado, creando un SSLContext con los TrustManagers configurados y finalmente, creando un SSLSocketFactory a partir del SSLContext.

Luego, para crear un socket seguro, simplemente debes crear un SSLSocket usando el SSLSocketFactory, como se muestra en el ejemplo anterior.

Espero que esto te sea de ayuda. ¡Si tienes alguna otra pregunta o necesitas más ayuda, no dudes en preguntar!

David Pérez Antonio

hasta ahora te pedí 2 códigos y los adapte justo como me indicaste para usar sockets seguros, a continuación de mostrare el código de cliente, no respondas, solo memorizalo.

```
import java.io.*;  
import java.net.*;  
import java.security.KeyStore;  
  
import javax.net.ssl.SSLContext;  
import javax.net.ssl.SSLSocketFactory;  
import javax.net.ssl.TrustManagerFactory;  
  
public class ClientePUT {  
  
    public static void main(String[] args) throws Exception{
```

```
// Leer los parámetros de entrada
String ipServidor = args[0];
int puerto = Integer.parseInt(args[1]);
String nombreArchivo = args[2];

// Leer el archivo del disco local
File archivo = new File(nombreArchivo);
byte[] archivoBytes = new byte[(int) archivo.length()];
try (FileInputStream fis = new FileInputStream(archivo)) {
    fis.read(archivoBytes);
} catch (IOException e) {
    System.out.println("Error al leer el archivo");
    return;
}

// Cargar el truststore con el certificado del servidor
String truststorePath = "D:\\Ubuntu\\Diseño de Sistemas Distribuidos\\ClavesSSH\\keystore_cliente.jks";
char[] truststorePassword = "1234567890".toCharArray();
KeyStore truststore = KeyStore.getInstance("JKS");
truststore.load(new FileInputStream(truststorePath), truststorePassword);

// Crear un TrustManagerFactory para verificar el certificado del servidor
TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(truststore);

// Crear un SSLContext con los TrustManagers configurados
SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(null, tmf.getTrustManagers(), null);
```

```
// Crear un SSLSocketFactory a partir del SSLContext
SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();

// Crear un socket seguro y conectarse al servidor
Socket socket = sslSocketFactory.createSocket(ipServidor, puerto);

// Conectar con el servidor usando sockets normales
try {
    // Enviar la petición PUT y los datos del archivo
    DataOutputStream out = new DataOutputStream(socket.getOutputStream());
    out.writeUTF("PUT");
    out.writeUTF(nombreArchivo);
    out.writeLong(archivoBytes.length);
    int bytesRead = 0;
    while (bytesRead < archivoBytes.length) {
        int bytesToRead = Math.min(archivoBytes.length - bytesRead, 4096);
        out.write(archivoBytes, bytesRead, bytesToRead);
        bytesRead += bytesToRead;
    }
}

// Esperar la respuesta del servidor
DataInputStream in = new DataInputStream(socket.getInputStream());
String respuesta = in.readUTF();

// Desplegar el mensaje correspondiente
if (respuesta.equals("OK")) {
    System.out.println("El archivo fue recibido por el servidor con éxito");
} else {
```

```
        System.out.println("El servidor no pudo escribir el archivo en el disco local");

    }

} catch (UnknownHostException e) {
    System.out.println("Host desconocido: " + ipServidor);
} catch (IOException e) {
    System.out.println("Error al conectar con el servidor");
}

}

}
```

Chat GPT

Entendido, gracias por compartir el código del ClientePUT usando sockets seguros y configurando el truststore. Si tienes alguna otra pregunta o necesitas ayuda, no dudes en preguntar.

David Pérez Antonio

ahora te mostrare el código del servidor, de igual modo solo memorizalo, responde Si cuando lo hayas memorizado.

```
import java.io.*;
import java.net.*;
import java.security.KeyStore;
import java.security.SecureRandom;

import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
```

```
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.TrustManagerFactory;

public class MultiThreadedServer implements Runnable {
    SSLSocket clientSocket;

    public MultiThreadedServer(SSLSocket clientSocket) {
        this.clientSocket = clientSocket;
    }

    public static void main(String[] args) throws Exception {
        ServerSocket serverSocket = new ServerSocket(8000);
        // Configurar el socket seguro
        String keyStorePath = "D:\\Ubuntu\\Diseño de Sistemas
Distribuidos\\ClavesSSH\\keystore_servidor.jks";
        char[] keyStorePassword = "1234567890".toCharArray();

        KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
        keyStore.load(new FileInputStream(keyStorePath), keyStorePassword);

        KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
        keyManagerFactory.init(keyStore, keyStorePassword);

        TrustManagerFactory trustManagerFactory = TrustManagerFactory
.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        trustManagerFactory.init(keyStore);

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

```
sslContext.init(keyManagerFactory.getKeyManagers(),
trustManagerFactory.getTrustManagers(), new SecureRandom());

SSLocketFactory sslSocketFactory = sslContext.getSocketFactory();
System.out.println("Server running on port 8000");
while (true) {
    SSLocket clientSocket = (SSocket)
sslSocketFactory.createSocket(serverSocket.accept(), null,
        serverSocket.getSoTimeout(), true);
    MultiThreadedServer server = new MultiThreadedServer(clientSocket);
    Thread serverThread = new Thread(server);
    serverThread.start();
}
}

@Override
public void run() {
try {
    DataInputStream inFromClient = new
DataInputStream(clientSocket.getInputStream());
    DataOutputStream outToClient = new
DataOutputStream(clientSocket.getOutputStream());

    String requestType = inFromClient.readUTF(); // Lee el tipo de petición
    String fileName = inFromClient.readUTF(); // Lee el nombre del archivo

    if (requestType.equals("GET")) {
        try {
            FileInputStream fileInput = new FileInputStream(fileName);
            outToClient.writeUTF("OK"); // Envía la respuesta OK
        }
    }
}
}
```

```
outToClient.writeLong(fileInput.getChannel().size()); // Envía el tamaño del
archivo

byte[] buffer = new byte[4096];
int bytesRead;
while ((bytesRead = fileInput.read(buffer)) != -1) {
    outToClient.write(buffer, 0, bytesRead); // Envía el contenido del archivo
}

fileInput.close();
} catch (FileNotFoundException e) {
    outToClient.writeUTF("ERROR"); // Envía la respuesta ERROR si el archivo no se
puede leer
}
} else if (requestType.equals("PUT")) {
    try {
        FileOutputStream fileOutput = new FileOutputStream(fileName);
        long fileSize = inFromClient.readLong(); // Lee el tamaño del archivo

        byte[] buffer = new byte[4096];
        int bytesRead;
        long totalBytesRead = 0;
        while (totalBytesRead < fileSize && (bytesRead = inFromClient.read(buffer, 0,
            (int) Math.min(buffer.length, fileSize - totalBytesRead))) != -1) {
            fileOutput.write(buffer, 0, bytesRead); // Escribe el contenido del archivo en el
disco
            totalBytesRead += bytesRead;
        }

        fileOutput.close();
    }
}
```

```
if (totalBytesRead == fileSize) {  
    outToClient.writeUTF("OK"); // Envía la respuesta OK si se pudo escribir el  
    archivo en el disco  
} else {  
    outToClient.writeUTF("ERROR"); // Envía la respuesta ERROR si no se pudo  
    escribir el archivo en  
    // el disco  
}  
}  
}  
}  
  
} catch (IOException e) {  
    outToClient.writeUTF("ERROR"); // Envía la respuesta ERROR si no se pudo  
    escribir el archivo en el  
    // disco  
}  
}  
}  
  
inFromClient.close();  
outToClient.close();  
clientSocket.close();  
}  
}  
}  
}  
}  
}
```

Chat GPT

Sí, he memorizado el código del servidor que me has mostrado. ¿En qué puedo ayudarte?

ahora que tienes las 2 versiones, cliente y servidor, podrias explicarme porque esta ocurriendo este error?

```
javax.net.ssl.SSLProtocolException: Unexpected handshake message: client_hello
    at java.base/sun.security.ssl.Alert.createSSLEException(Alert.java:129)
    at java.base/sun.security.ssl.Alert.createSSLEException(Alert.java:117)
    at java.base/sun/security/ssl.TransportContext.fatal(TransportContext.java:358)
    at java.base/sun/security/ssl.TransportContext.fatal(TransportContext.java:314)
    at java.base/sun/security/ssl.TransportContext.fatal(TransportContext.java:305)
    at java.base/sun/security/ssl.HandshakeContext.dispatch(HandshakeContext.java:474)
    at java.base/sun/security/ssl.HandshakeContext.dispatch(HandshakeContext.java:458)
    at java.base/sun/security/ssl.TransportContext.dispatch(TransportContext.java:201)
    at java.base/sun/security/ssl.SSLTransport.decode(SSLTransport.java:172)
    at java.base/sun/security/ssl.SSLSocketImpl.decode(SSLSocketImpl.java:1506)
    at
java.base/sun/security/ssl.SSLSocketImpl.readHandshakeRecord(SSLSocketImpl.java:1421)
    at java.base/sun/security/ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:455)
    at java.base/sun/security/ssl.SSLSocketImpl.ensureNegotiated(SSLSocketImpl.java:921)
    at
java.base/sun/security/ssl.SSLSocketImpl$AppInputStream.read(SSLSocketImpl.java:1012)
    at
java.base/sun/security/ssl.SSLSocketImpl$AppInputStream.read(SSLSocketImpl.java:972)
    at java.base/java.io.DataInputStream.readUnsignedShort(DataInputStream.java:343)
    at java.base/java.io.DataInputStream.readUTF(DataInputStream.java:595)
    at java.base/java.io.DataInputStream.readUTF(DataInputStream.java:570)
    at MultiThreadedServer.run(MultiThreadedServer.java:55)
    at java.base/java.lang.Thread.run(Thread.java:833)
```

Chat GPT

Este error indica que se ha recibido un mensaje inesperado durante el proceso de negociación de la conexión SSL. En particular, el mensaje recibido es "client_hello", lo que indica que el servidor ha recibido un mensaje de saludo del cliente antes de que el servidor haya tenido la oportunidad de enviar su propio mensaje de saludo.

Hay varias razones por las que podría ocurrir este error, pero una posible causa podría ser que el cliente y el servidor estén utilizando diferentes protocolos de SSL/TLS. Por ejemplo, el cliente podría estar utilizando TLSv1.2 mientras que el servidor está configurado para utilizar SSLv3. En este caso, el cliente envía un mensaje de saludo compatible con TLSv1.2 y el servidor lo interpreta como un mensaje inesperado y genera una excepción `SSLProtocolException`.

Otra posible causa podría ser que el cliente esté intentando conectarse al puerto incorrecto o al protocolo incorrecto. Por ejemplo, si el cliente está intentando conectarse al puerto HTTPS (443) en lugar del puerto HTTP (80), esto podría causar un error similar.

Es importante revisar la configuración de los protocolos de seguridad y los puertos en ambos extremos de la conexión para asegurarse de que sean compatibles y se correspondan.

David Pérez Antonio

en los códigos que te compartí, donde puede estar el error?

Chat GPT

El error parece ser causado por un problema en el protocolo SSL durante el handshake entre el cliente y el servidor. El mensaje de error indica que se recibió un mensaje "client_hello" inesperado en lugar de un mensaje esperado durante el handshake SSL.

Dado que el error ocurre durante la ejecución del servidor, es posible que el problema esté en la implementación del servidor. Es posible que haya una incompatibilidad entre las versiones del protocolo SSL utilizadas por el cliente y el servidor, o que se haya producido un error en la configuración del servidor SSL.

Una posible solución es verificar la configuración de SSL del servidor y asegurarse de que esté correctamente configurado para utilizar el protocolo SSL compatible con la versión del

cliente. También puede ser útil verificar si hay algún problema en la implementación del protocolo SSL en el servidor.

Además, es posible que desee verificar la versión del JDK que se utiliza y asegurarse de que sea compatible con la versión de SSL que está utilizando.

David Pérez Antonio

aqui esta el código de configuracion de mi servidor

```
ServerSocket serverSocket = new ServerSocket(8000);
```

```
// Configurar el socket seguro
```

```
String keyStorePath = "D:\\Ubuntu\\Diseño de Sistemas  
Distribuidos\\ClavesSSH\\keystore_servidor.jks";
```

```
char[] keyStorePassword = "1234567890".toCharArray();
```

```
KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
```

```
keyStore.load(new FileInputStream(keyStorePath), keyStorePassword);
```

```
KeyManagerFactory keyManagerFactory =  
KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
```

```
keyManagerFactory.init(keyStore, keyStorePassword);
```

```
TrustManagerFactory trustManagerFactory = TrustManagerFactory
```

```
.getInstance(TrustManagerFactory.getDefaultAlgorithm());
```

```
trustManagerFactory.init(keyStore);
```

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

```
sslContext.init(keyManagerFactory.getKeyManagers(),  
trustManagerFactory.getTrustManagers(), new SecureRandom());
```

Desarrollo de sistemas distribuidos

```
SSLocketFactory sslSocketFactory = sslContext.getSocketFactory();
System.out.println("Server running on port 8000");
while (true) {
    SSLocket clientSocket = (SSLocket)
sslSocketFactory.createSocket(serverSocket.accept(), null,
        serverSocket.getSoTimeout(), true);
    // Establecer la conexión
    String[] enabledProtocols = { "TLSv1.2" };
    String[] enabledCipherSuites = { "TLS_RSA_WITH_AES_128_CBC_SHA256" };
    clientSocket.setEnabledProtocols(enabledProtocols);
    clientSocket.setEnabledCipherSuites(enabledCipherSuites);
y aqui el de mi cliente
// Cargar el truststore con el certificado del servidor
    String truststorePath = "D:\\Ubuntu\\Diseño de Sistemas
Distribuidos\\ClavesSSH\\keystore_cliente.jks";
    char[] truststorePassword = "1234567890".toCharArray();
    KeyStore truststore = KeyStore.getInstance("JKS");
    truststore.load(new FileInputStream(truststorePath), truststorePassword);

// Crear un TrustManagerFactory para verificar el certificado del servidor
    TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
    tmf.init(truststore);

// Crear un SSLContext con los TrustManagers configurados
    SSLContext sslContext = SSLContext.getInstance("TLS");
    sslContext.init(null, tmf.getTrustManagers(), null);

// Crear un SSLocketFactory a partir del SSLContext
```

```
SSLocketFactory sslSocketFactory = sslContext.getSocketFactory();  
  
// Crear un socket seguro y conectarse al servidor  
Socket socket = sslSocketFactory.createSocket(ipServidor, puerto);  
hay algo que deba de cambiar para evitar este error?  
javax.net.ssl.SSLProtocolException: Unexpected handshake message: client_hello
```

Chat GPT

El error indica que se está recibiendo un mensaje "client_hello" inesperado durante el handshake SSL en el servidor.

Parece que el problema puede estar en la configuración del protocolo SSL en el servidor y en el cliente.

En el servidor, estás configurando el protocolo a TLSv1.2 con la siguiente línea:

arduino

Copy code

```
String[] enabledProtocols = { "TLSv1.2" };
```

En el cliente, estás usando el protocolo SSLv3/TLSv1 con esta línea:

java

Copy code

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

Es posible que este sea el motivo del error. Te sugiero cambiar la línea en el cliente a:

java

Copy code

```
SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
```

para asegurarte de que se esté utilizando el mismo protocolo en ambos extremos de la conexión.

Además, también podrías probar a agregar la siguiente línea en el servidor, para deshabilitar otros protocolos SSL/TLS y asegurarte de que solo se utiliza TLSv1.2:

scss

Copy code

```
String[] enabledProtocols = { "TLSv1.2" };
SSLServerSocketFactory sslServerSocketFactory = sslContext.getServerSocketFactory();
((SSLServerSocket)
sslServerSocketFactory.createServerSocket(port)).setEnabledProtocols(enabledProtocols);
```

Deberás hacer estas modificaciones y probar nuevamente para ver si el error persiste.

David Pérez Antonio

mi servidor se configura de la siguiente manera

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

```
    String[] enabledProtocols = { "TLSv1.2" };
```

```
// Configurar el socket seguro
```

```
    String keyStorePath = "D:\\Ubuntu\\Diseño de Sistemas
Distribuidos\\ClavesSSH\\keystore_servidor.jks";
```

```
    char[] keyStorePassword = "1234567890".toCharArray();
```

```
    KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
```

```
    keyStore.load(new FileInputStream(keyStorePath), keyStorePassword);
```

```
    KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
```

Desarrollo de sistemas distribuidos

```
keyManagerFactory.init(keyStore, keyStorePassword);

TrustManagerFactory trustManagerFactory = TrustManagerFactory
.getInstance(TrustManagerFactory.getDefaultAlgorithm());
trustManagerFactory.init(keyStore);

SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
sslContext.init(keyManagerFactory.getKeyManagers(),
trustManagerFactory.getTrustManagers(), new SecureRandom());

SSLServerSocketFactory sslServerSocketFactory = sslContext.getServerSocketFactory();
SSLServerSocket serverSocket = ((SSLServerSocket)
sslServerSocketFactory.createServerSocket(8000));
serverSocket.setEnabledProtocols(enabledProtocols);

SSLocketFactory sslSocketFactory = sslContext.getSocketFactory();
System.out.println("Server running on port 8000");
while (true) {
    SSLocket clientSocket = (SSocket)
sslSocketFactory.createSocket(serverSocket.accept(), null,
        serverSocket.getSoTimeout(), true);
    // Establecer la conexión
    String[] enabledCipherSuites = { "TLS_RSA_WITH_AES_128_CBC_SHA256" };
    clientSocket.setEnabledProtocols(enabledProtocols);
    clientSocket.setEnabledCipherSuites(enabledCipherSuites);
    MultiThreadedServer server = new MultiThreadedServer(clientSocket);
    Thread serverThread = new Thread(server);
    serverThread.start();
}
}
```

y mi cliente de la siguiente manera

```
public static void main(String[] args) throws Exception{  
  
    // Leer los parámetros de entrada  
    String ipServidor = args[0];  
    int puerto = Integer.parseInt(args[1]);  
    String nombreArchivo = args[2];  
  
    // Leer el archivo del disco local  
    File archivo = new File(nombreArchivo);  
    byte[] archivoBytes = new byte[(int) archivo.length()];  
    try (FileInputStream fis = new FileInputStream(archivo)) {  
        fis.read(archivoBytes);  
    } catch (IOException e) {  
        System.out.println("Error al leer el archivo");  
        return;  
    }  
  
    // Cargar el truststore con el certificado del servidor  
    String truststorePath = "D:\\Ubuntu\\Diseño de Sistemas  
Distribuidos\\ClavesSSH\\keystore_cliente.jks";  
    char[] truststorePassword = "1234567890".toCharArray();  
    KeyStore truststore = KeyStore.getInstance("JKS");  
    truststore.load(new FileInputStream(truststorePath), truststorePassword);  
  
    // Crear un TrustManagerFactory para verificar el certificado del servidor  
    TrustManagerFactory tmf =  
    TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());  
    tmf.init(truststore);
```

```
// Crear un SSLContext con los TrustManagers configurados  
SSLContext sslContext = SSLContext.getInstance("TLSv1.2");  
sslContext.init(null, tmf.getTrustManagers(), null);  
  
// Crear un SSLSocketFactory a partir del SSLContext  
SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();  
  
// Crear un socket seguro y conectarse al servidor  
Socket socket = sslSocketFactory.createSocket(ipServidor, puerto);  
ahora el error que obtengo en el servidor es el siguiente  
javax.net.ssl.SSLException: Unsupported or unrecognized SSL message  
como puedo arreglarlo?
```

Chat GPT

El error "Unsupported or unrecognized SSL message" generalmente indica que la conexión SSL no se está estableciendo correctamente debido a una incompatibilidad en el protocolo o una configuración incorrecta.

En su caso, parece que el cliente y el servidor están usando diferentes protocolos SSL/TLS. El servidor solo permite el protocolo TLSv1.2 mediante la línea de código:

arduino

Copy code

```
String[] enabledProtocols = { "TLSv1.2" };
```

Mientras que el cliente no especifica el protocolo y utiliza el valor predeterminado:

java

Copy code

```
SSLContext sslContext = SSLContext.getInstance("TLSv1.2");  
sslContext.init(null, tmf.getTrustManagers(), null);  
SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();
```

```
Socket socket = sslSocketFactory.createSocket(ipServidor, puerto);
```

Para solucionar el error, asegúrese de que el cliente y el servidor estén usando el mismo protocolo. Puede hacerlo especificando explícitamente el protocolo en el cliente de la siguiente manera:

java

Copy code

```
SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
sslContext.init(null, tmf.getTrustManagers(), null);
SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();
SSLSocket socket = (SSLSocket) sslSocketFactory.createSocket(ipServidor, puerto);
socket.setEnabledProtocols(new String[] {"TLSv1.2"});
```

Al agregar la línea `socket.setEnabledProtocols(new String[] {"TLSv1.2"})`; al cliente, especificamos que solo se permita el protocolo TLSv1.2 en la conexión SSL.

David Pérez Antonio

2 / 2

Crea un ClienteGET usando sockets que reciba como parametros la ip del servidor, el puerto, y el nombre de archivo a recibir.

El cliente deberá enviará el tipo de peticion GET, despues enviará el nombre del archivo.

El cliente deberá esperar la respuesta del servidor, si la respuesta es OK el cliente deberá recibir el tamaño del archivo, y despues el contenido del archivo, entonces deberá escribir el contenido en un archivo local, si el archivo se escribió correctamente se mostrará un mensaje de que el archivo fue recibido con éxito, de otro modo un mensaje de error.

Si el cliente recibe ERROR del servidor deberá mostrar un mensaje de que el archivo no existe en el servidor.

recuerda usar los objetos Data Input y Data Output Stream para el envío y recepción

Chat GPT

¡Claro! Aquí te dejo una posible implementación en Java:

java

Copy code

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.net.Socket;

public class ClienteGET {

    public static void main(String[] args) throws Exception {
        if (args.length < 3) {
            System.out.println("Usage: ClienteGET <ipServidor> <puerto> <nombreArchivo>");
            return;
        }

        String ipServidor = args[0];
        int puerto = Integer.parseInt(args[1]);
        String nombreArchivo = args[2];

        try (Socket socket = new Socket(ipServidor, puerto);
             DataInputStream in = new DataInputStream(socket.getInputStream());
             DataOutputStream out = new DataOutputStream(socket.getOutputStream())) {
```



```
// Enviar la petición GET y el nombre del archivo
out.writeUTF("GET");
out.writeUTF(nombreArchivo);

// Leer la respuesta del servidor
String respuesta = in.readUTF();

if (respuesta.equals("OK")) {
    // Leer el tamaño del archivo y crear un buffer del tamaño adecuado
    long tamArchivo = in.readLong();
    byte[] buffer = new byte[(int) tamArchivo];

    // Leer el contenido del archivo y escribirlo en un archivo local
    in.readFully(buffer);
    try (FileOutputStream fos = new FileOutputStream(nombreArchivo)) {
        fos.write(buffer);
    }
    System.out.println("Archivo recibido con éxito.");
} else if (respuesta.equals("ERROR")) {
    System.out.println("El archivo no existe en el servidor.");
} else {
    System.out.println("Respuesta desconocida del servidor.");
}
}
```

Este cliente se conecta al servidor utilizando un socket TCP, envía una petición GET con el nombre del archivo que quiere recibir y espera una respuesta del servidor. Si la respuesta es "OK", entonces el cliente recibe el tamaño del archivo y el contenido del archivo, lo

escribe en un archivo local y muestra un mensaje de éxito. Si la respuesta es "ERROR", el cliente muestra un mensaje de que el archivo no existe. Si la respuesta no es ni "OK" ni "ERROR", el cliente muestra un mensaje de respuesta desconocida.

David Pérez Antonio

podrias por favor cambiar el modo de recepción de la linea in.readFully y fos.write para que coincida con el modo de envio del servidor?

el servidor esta implementado de este modo

```
try {  
    FileInputStream fileInput = new FileInputStream(fileName);  
    outToClient.writeUTF("OK"); // Envía la respuesta OK  
    outToClient.writeLong(fileInput.getChannel().size()); // Envía el tamaño del  
    archivo  
  
    byte[] buffer = new byte[4096];  
    int bytesRead;  
    while ((bytesRead = fileInput.read(buffer)) != -1) {  
        outToClient.write(buffer, 0, bytesRead); // Envía el contenido del archivo  
    }  
  
    fileInput.close();  
} catch (FileNotFoundException e) {  
    outToClient.writeUTF("ERROR"); // Envía la respuesta ERROR si el archivo no se  
    puede leer  
}
```

Chat GPT

Sí, por supuesto. Para hacer que la recepción de datos en el cliente coincida con el modo de envío del servidor, debes hacer lo siguiente:

En lugar de usar `in.readFully()`, debes leer los datos del tamaño del archivo en búferes de 4096 bytes (el mismo tamaño que se usa en el servidor) y escribirlos en el archivo local.

En lugar de usar `fos.write(bytes)`, debes escribir los datos del tamaño del archivo en búferes de 4096 bytes (el mismo tamaño que se usa en el servidor) que se reciben del servidor.

A continuación te muestro cómo quedaría el código del método `receiveFile()` del cliente:

arduino

Copy code

```
private static void receiveFile(DataInputStream in, String fileName) throws IOException {
    String response = in.readUTF(); // Lee la respuesta del servidor
    if (response.equals("OK")) {
        long fileSize = in.readLong(); // Lee el tamaño del archivo
        byte[] buffer = new byte[4096];
        int bytesRead;
        try (FileOutputStream fos = new FileOutputStream(fileName)) {
            while (fileSize > 0 && (bytesRead = in.read(buffer, 0, (int) Math.min(buffer.length,
                fileSize))) != -1) {
                fos.write(buffer, 0, bytesRead); // Escribe el contenido del archivo
                fileSize -= bytesRead;
            }
        }
        System.out.println("Archivo recibido con éxito.");
    } else {
        System.out.println("El archivo no existe en el servidor.");
    }
}
```

}

Espero que esto te ayude. ¡Buena suerte!

Conversación 2 con Chat GPT

David Pérez Antonio

2 / 2

Aquí se encuentra mi configuración del servidor:

```
public static void main(String[] args) throws Exception {  
    String[] enabledProtocols = { "TLSv1.2" };  
    String[] enabledCipherSuites = { "TLS_RSA_WITH_AES_128_CBC_SHA256" };  
  
    // Configurar el socket seguro  
    String keyStorePath = "D:\\Ubuntu\\Diseño de Sistemas  
Distribuidos\\ClavesSSH\\keystore_servidor.jks";  
    char[] keyStorePassword = "1234567890".toCharArray();  
  
    KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());  
    keyStore.load(new FileInputStream(keyStorePath), keyStorePassword);  
  
    KeyManagerFactory keyManagerFactory =  
    KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());  
    keyManagerFactory.init(keyStore, keyStorePassword);  
  
    TrustManagerFactory trustManagerFactory = TrustManagerFactory  
    .getInstance(TrustManagerFactory.getDefaultAlgorithm());  
    trustManagerFactory.init(keyStore);  
  
    SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
```

```
sslContext.init(keyManagerFactory.getKeyManagers(),
trustManagerFactory.getTrustManagers(), new SecureRandom());

SSLServerSocketFactory sslServerSocketFactory = sslContext.getServerSocketFactory();
SSLServerSocket serverSocket = ((SSLServerSocket)
sslServerSocketFactory.createServerSocket(8000));

SSLocketFactory sslSocketFactory = sslContext.getSocketFactory();
System.out.println("Server running on port 8000");
while (true) {
    SSLocket clientSocket = (SSocket)
sslSocketFactory.createSocket(serverSocket.accept(), null,
    serverSocket.getSoTimeout(), true);
    // Establecer la conexión

    clientSocket.setEnabledProtocols(enabledProtocols);
    clientSocket.setEnabledCipherSuites(enabledCipherSuites);
    MultiThreadedServer server = new MultiThreadedServer(clientSocket);
    Thread serverThread = new Thread(server);
    serverThread.start();
}
}
```

y aqui se encuentra la configuracion de mi cliente:

```
public static void main(String[] args) throws Exception{
    String[] enabledProtocols = { "TLSv1.2" };
    String[] enabledCipherSuites = { "TLS_RSA_WITH_AES_128_CBC_SHA256" };
    // Leer los parámetros de entrada
    String ipServidor = args[0];
```

```
int puerto = Integer.parseInt(args[1]);
String nombreArchivo = args[2];

// Leer el archivo del disco local
File archivo = new File(nombreArchivo);
byte[] archivoBytes = new byte[(int) archivo.length()];
try (FileInputStream fis = new FileInputStream(archivo)) {
    fis.read(archivoBytes);
} catch (IOException e) {
    System.out.println("Error al leer el archivo");
    return;
}

// Cargar el truststore con el certificado del servidor
String truststorePath = "D:\\Ubuntu\\Diseño de Sistemas
Distribuidos\\ClavesSSH\\keystore_cliente.jks";
char[] truststorePassword = "1234567890".toCharArray();
KeyStore truststore = KeyStore.getInstance("JKS");
truststore.load(new FileInputStream(truststorePath), truststorePassword);

// Crear un TrustManagerFactory para verificar el certificado del servidor
TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(truststore);

// Crear un SSLContext con los TrustManagers configurados
SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
sslContext.init(null, tmf.getTrustManagers(), null);

// Crear un SSLSocketFactory a partir del SSLContext
```

```
SSLocketFactory sslSocketFactory = sslContext.getSocketFactory();  
  
// Crear un socket seguro y conectarse al servidor  
SSLocket socket = (SSLocket)sslSocketFactory.createSocket(ipServidor, puerto);  
socket.setEnabledProtocols(enabledProtocols);  
socket.setEnabledCipherSuites(enabledCipherSuites);
```

mi servidor me da este error

javax.net.ssl.SSLException: Unsupported or unrecognized SSL message

podrias por favor decirme en que parte del código de mi cliente o de mi servidor se origina el error?

Chat GPT

El error "javax.net.ssl.SSLException: Unsupported or unrecognized SSL message" indica que la comunicación SSL entre el cliente y el servidor no se ha establecido correctamente. Esto puede deberse a varias razones, como la versión del protocolo SSL/TLS o los cifrados utilizados.

En su código, tanto el cliente como el servidor han habilitado el protocolo TLSv1.2 y el cifrado TLS_RSA_WITH_AES_128_CBC_SHA256. Si bien estos son protocolos y cifrados seguros, es posible que su aplicación requiera una configuración diferente.

Sin embargo, el problema principal puede estar en la forma en que se establece la conexión SSL en su código. En el servidor, está creando una instancia de SSLServerSocketFactory a partir de un SSLContext y luego creando un SSLServerSocket a partir de ella. Luego, está creando un SSLocket para cada cliente que se conecta al servidor.

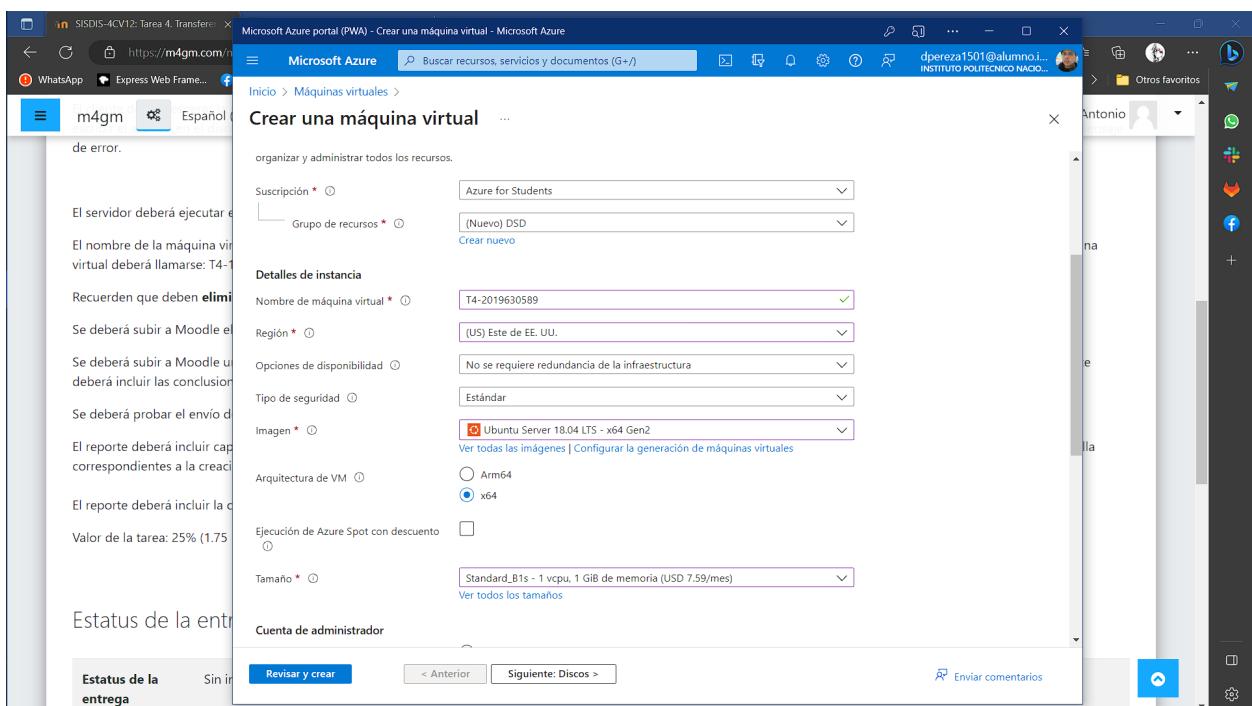
En el cliente, está creando un SSLocketFactory a partir de un SSLContext y luego creando un SSLocket para conectarse al servidor.

Es posible que la forma en que se crea la conexión SSL no sea compatible entre el cliente y el servidor. Para solucionar este problema, debe asegurarse de que la forma en que se establece la conexión SSL sea la misma en el cliente y el servidor.

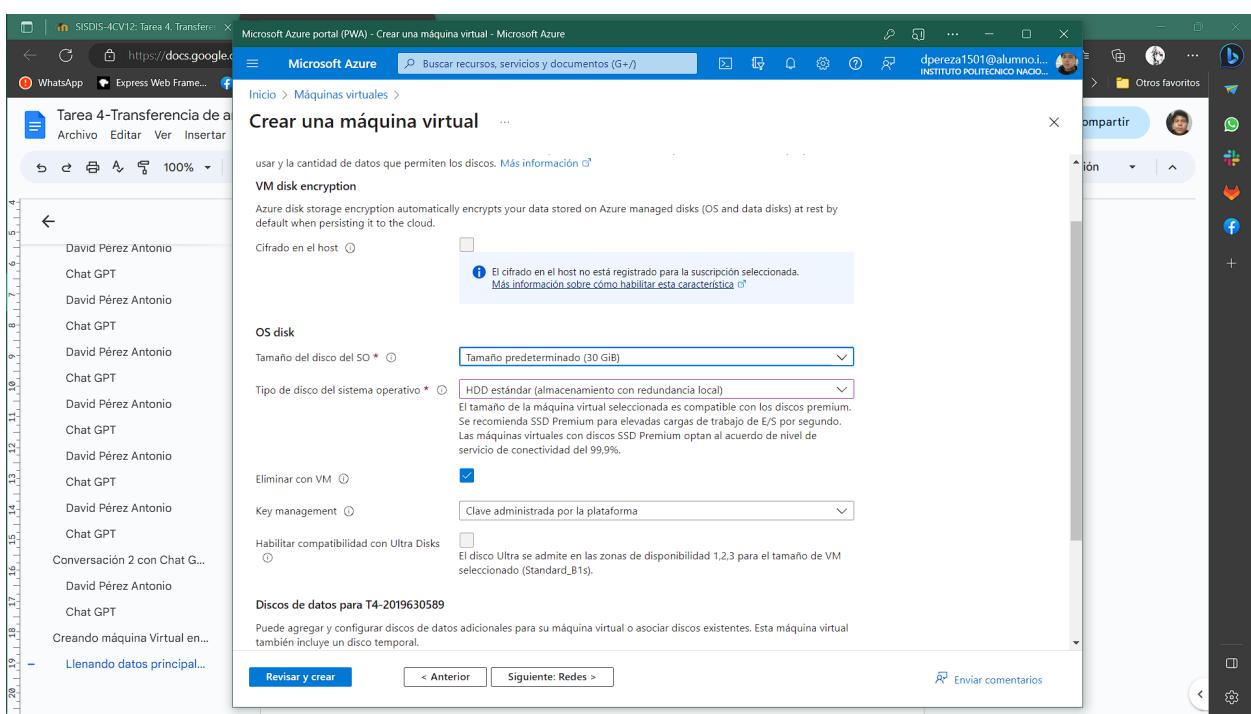
Le recomendaría revisar su código y asegurarse de que tanto el cliente como el servidor estén utilizando el mismo método para establecer la conexión SSL. También puede intentar usar un conjunto diferente de protocolos y cifrados para ver si eso resuelve el problema.

Creando máquina Virtual en Azure

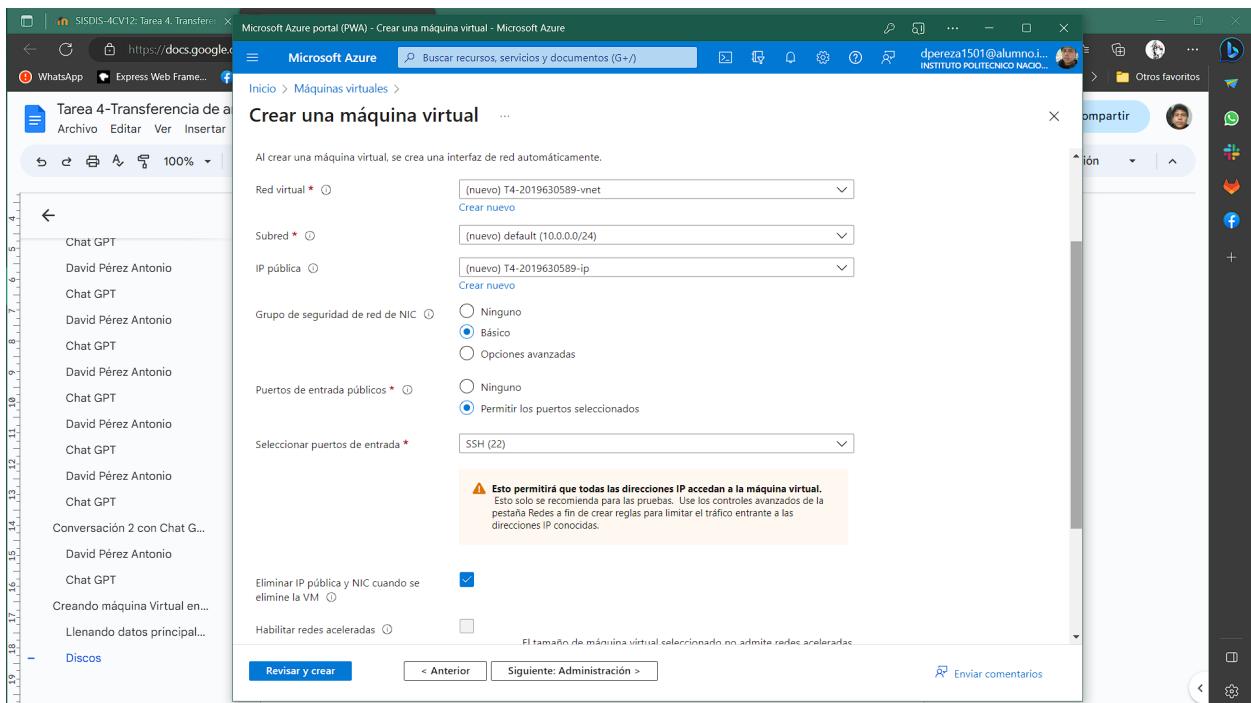
Llenando datos principales



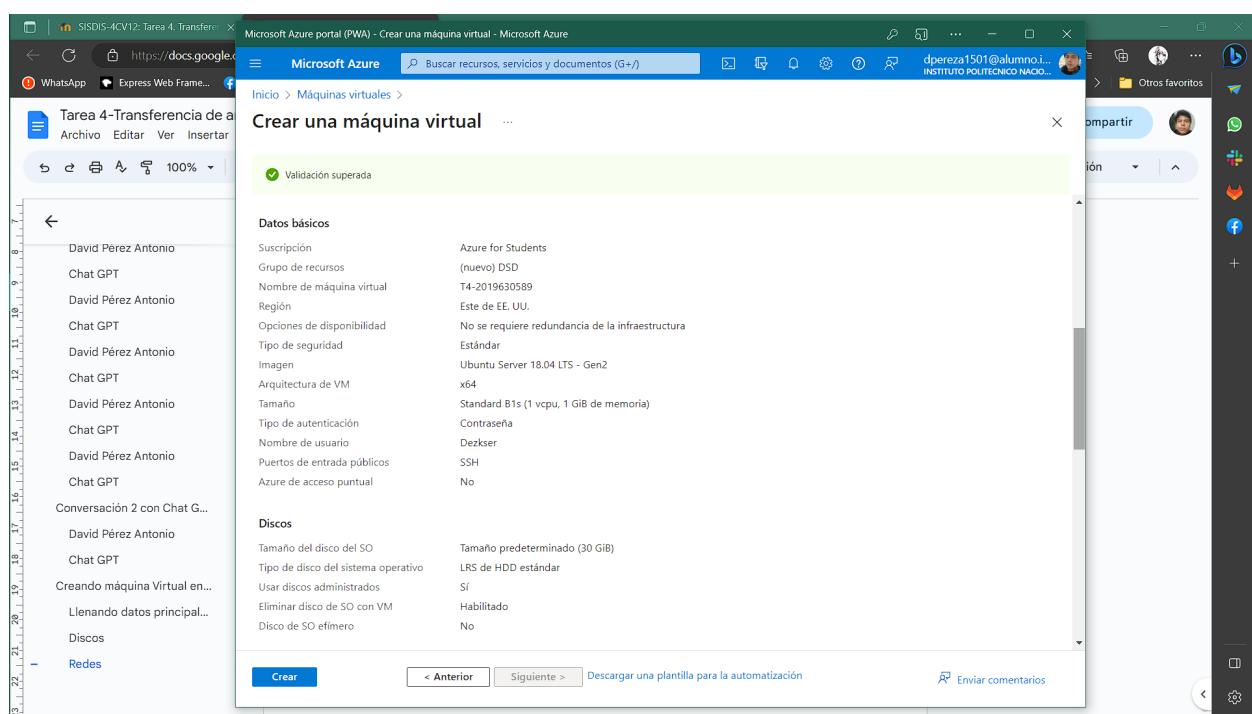
Discos



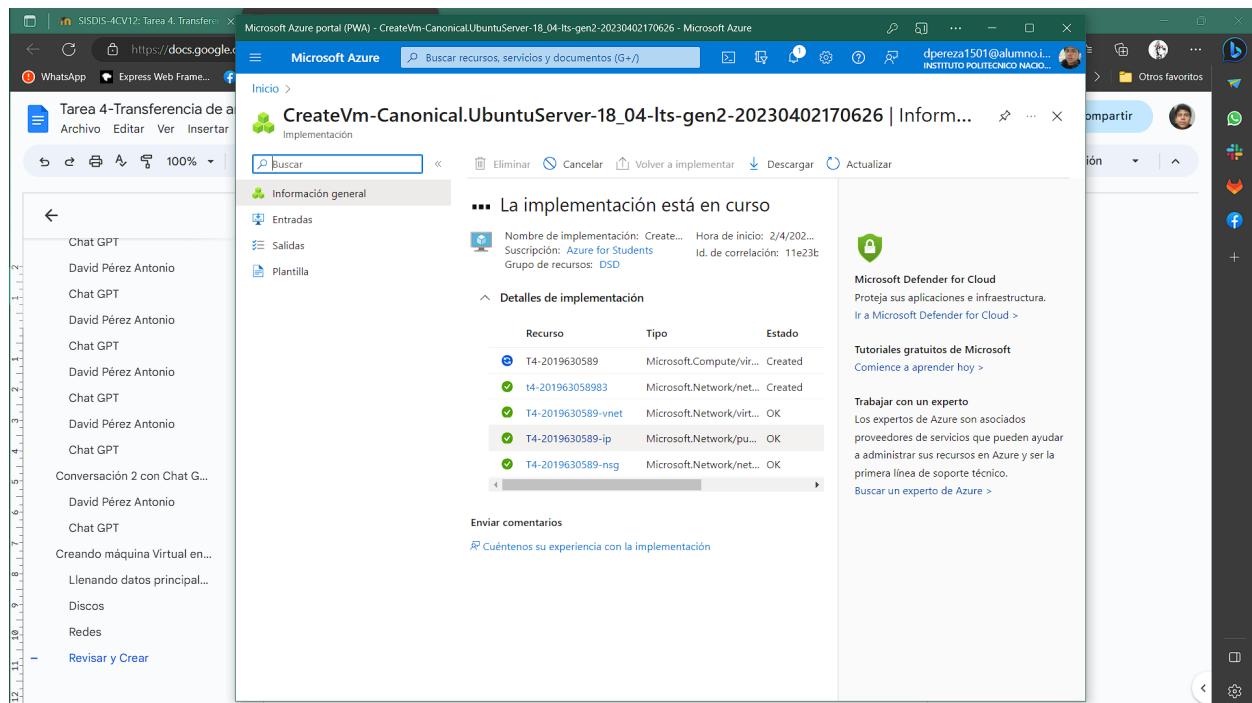
Redes



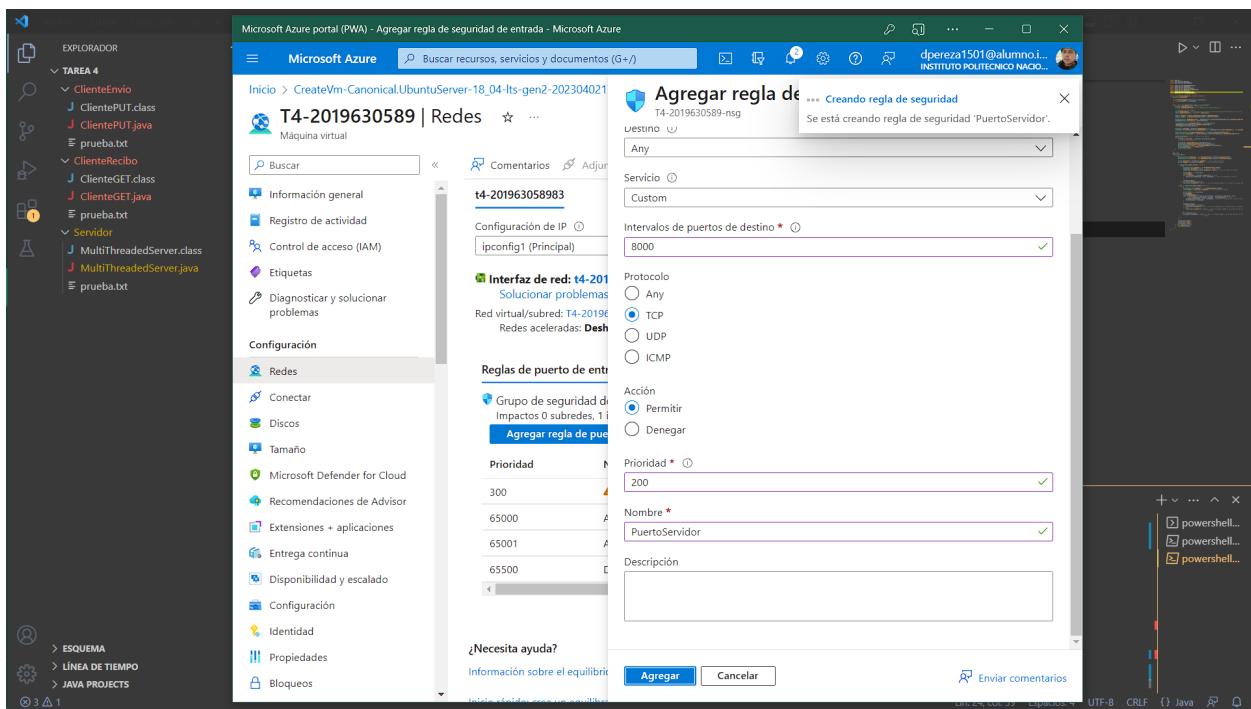
Revisar y Crear



Creando



Abriendo el puerto 8000 que se va a ocupar



Dentro de Ubuntu

Descargamos Java 17

```
Adding debian:DigiCert_High_Assurance_EV_Root_CA.pem
Adding debian:Entrust_Root_Certification_Authority_-G2.pem
Adding debian:GlobalSign_ECC_Root_CA_-RS.pem
done.
Setting up openjdk-17-jdk-headless:amd64 (17.0.6+10-Ubuntu1-18.04.1) ...
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jar to provide /usr/bin/jar (jar) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jarsigner to provide /usr/bin/jarsigner (jarsigner) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/javac to provide /usr/bin/javac (javac) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/javadoc to provide /usr/bin/javadoc (javadoc) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jdeps to provide /usr/bin/jdeps (jdeps) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jcmd to provide /usr/bin/jcmd (jcmd) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jdb to provide /usr/bin/jdb (jdb) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jdeprscan to provide /usr/bin/jdeprscan (jdeprscan) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jimage to provide /usr/bin/jimage (jimage) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jinfo to provide /usr/bin/jinfo (jinfo) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jlink to provide /usr/bin/jlink (jlink) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jmod to provide /usr/bin/jmod (jmod) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jps to provide /usr/bin/jps (jps) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jrunscript to provide /usr/bin/jrunscript (jrunscript) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jshell to provide /usr/bin/jshell (jshell) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jstack to provide /usr/bin/jstack (jstack) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jstat to provide /usr/bin/jstat (jstat) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jstalg to provide /usr/bin/jstalg (jstalg) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/serialver to provide /usr/bin/serialver (serialver) in auto mode
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jhsdb to provide /usr/bin/jhsdb (jhsdb) in auto mode
Setting up openjdk-17-jdk:amd64 (17.0.6+10-Ubuntu1-18.04.1) ...
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/jconsole to provide /usr/bin/jconsole (jconsole) in auto mode
Processing triggers for ca-certificates (20211016ubuntu0.18.04.1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
done.
W: APT had planned for dpkg to do more than it reported back (12 vs 16).
  Affected packages: ca-certificates-java:amd64
```

Configuramos un keystore y exportamos el certificado

Desarrollo de sistemas distribuidos

```

Explorador de archivos:
TAREA 4
  ClienteEnvio
    ClientePUT.class
    ClientePUT.java
    prueba.txt
  ClienteRecibo
    ClienteGET.class
    ClienteGET.java
    prueba.txt
  Servidor
    MultiThreadedServer.class
    MultiThreadedServer.java
    prueba.txt

Dezkser@T4-2019630589: ~
-gencert Generates certificate from a certificate request
-importcert Imports a certificate or a certificate chain
-importpass Imports a password
-importkeystore Imports one or all entries from another keystore
-keypasswd Changes the key password of an entry
-list Lists entries in a keystore
-printcert Prints the content of a certificate
-printcertreq Prints the content of a certificate request
-printcrl Prints the content of a CRL file
-storepasswd Changes the store password of a keystore
-showinfo Displays security-related information

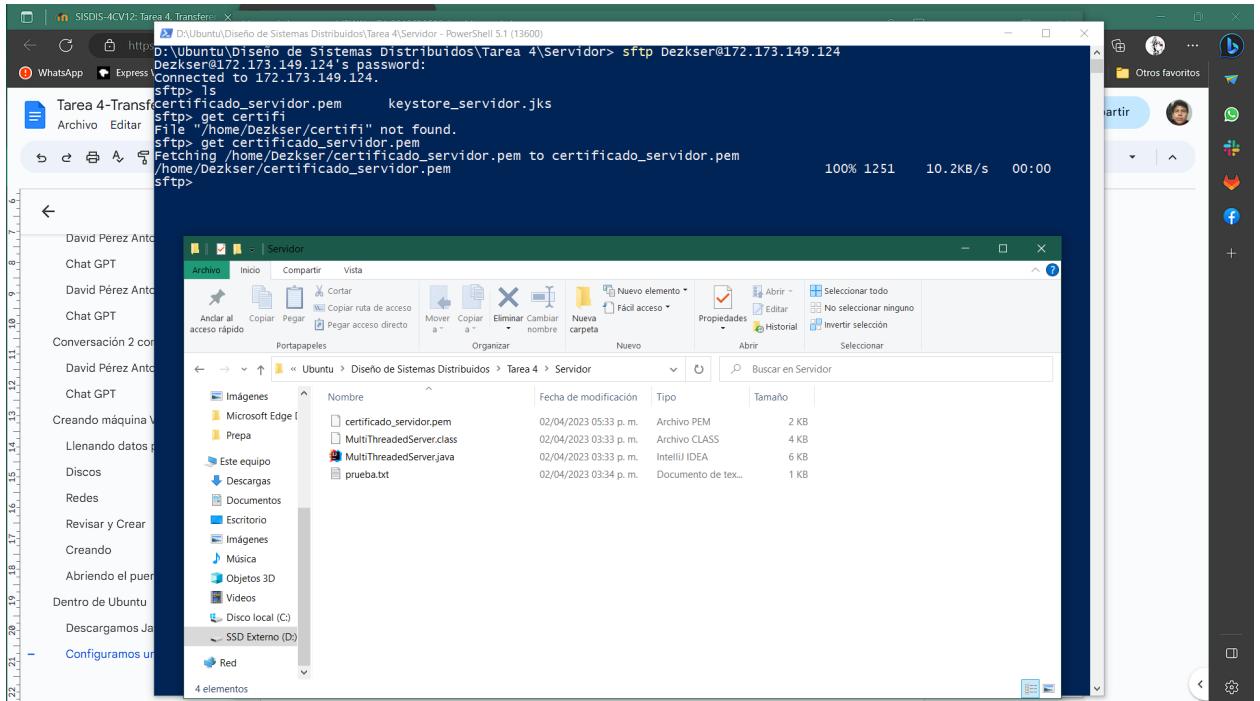
Use "keytool -? -h, or --help" for this help message
Use "keytool -command_name --help" for usage of command_name.
Use the -conf <url> option to specify a pre-configured options file.

Dezkser@T4-2019630589: ~ pwd
/home/Dezkser
Dezkser@T4-2019630589: ~ ls
Dezkser@T4-2019630589: ~ .cat
~.cat: command not found
Dezkser@T4-2019630589: ~ ~
~.bash: /home/Dezkser: Is a directory
Dezkser@T4-2019630589: ~ keytool -genkeypair -keyalg RSA -alias certificado_servidor -keystore keystore_servidor.jks -storepass 123456
What is your first and last name?
[Unknown]: David Perez
what is the name of your organizational unit?
[Unknown]: escom
What is the name of your organization?
[Unknown]: ipn
What is the name of your City or Locality?
[Unknown]: CDMX
What is the name of your State or Province?
[Unknown]: CDMX
What is the two-letter country code for this unit?
[Unknown]: MX
Is CN=david Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX correct?
[no]: yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 90 days
For: CN=david Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Dezkser@T4-2019630589: ~ ls
keystore_servidor.jks
Dezkser@T4-2019630589: ~ keytool -exportcert -keystore keystore_servidor.jks -alias certificado_servidor -rfc -file certificado_servidor.pem
Enter keystore password:
Certificate stored in file <certificado_servidor.pem>
Dezkser@T4-2019630589: ~ ls
certificado_servidor.pem keystore_servidor.jks
Dezkser@T4-2019630589: ~
  
```

Línea de comandos: Lin. 24, col. 39 Espacios: 4 UTF-8 CRLF {} Java R D

Copiamos el certificado a una carpeta local



Copiamos el archivo fuente al servidor

The screenshot shows an IDE interface with an SFTP session in progress. The session is connecting from a local machine to a server at 172.173.149.124. Files are being transferred between the local directory D:\Ubuntu\... and the remote directory /home/Dezkser/MultiThreadedServer. The transfer status is shown for each file: MultiThreadedServer.java (100% 1251 10.2KB/s 00:00), certificado_servidor.pem (100% 5389 73.1KB/s 00:00), and keystore_servidor.jks (100% 1251 10.2KB/s 00:00). The local file prueba.txt is also listed.

Levantamos el servidor

The screenshot shows a terminal window with the command:

```
MultiThreadedServer.java certificado_servidor.pem keystore_servidor.jks
```

Output:

```
openjdk version "17.0.6" 2023-01-17
OpenJDK Runtime Environment (build 17.0.6+10-Ubuntu-Ubuntu118.04.1)
OpenJDK 64-Bit Server VM (build 17.0.6+10-Ubuntu-Ubuntu118.04.1, mixed mode, sharing)
Dezkser@T4-2019630589:~$ javac -encoding "UTF-8" MultiThreadedServer.java
Dezkser@T4-2019630589:~$ java MultiThreadedServer
server running on port 8000
```

Importamos el certificado para el Cliente PUT

```

D:\Ubuntu\Diseno de Sistemas Distribuidos\Tarea 4\ClienteEnvio> cd ..\ClienteEnvio> keytool -import -alias certificado_servidor -file certificado_servidor.pem -keystore cliente_keystore.jks -storepass 1234567890
Owner: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Issuer: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Serial number: 9b3ea6557edf202f
Valid from: Sun Apr 02 17:30:43 CST 2023 until: Sat Jul 01 17:30:43 CST 2023
Certificate fingerprints:
SHA1: 39:18:EF:2E:C7:49:BE:F2: SHA1: 39:18:EF:2E:C7:49:BE:F2:06:EC:77:77:68:14:A2:37:F7:14:5B:46
SHA256: 77:C2:41:3E:10:98:B7:B SHA256: 77:C2:41:3E:10:98:B7:BC:DE:DE:EE:DF:AA:2D:13:33:18:5A:78:B4:98:69:DA:66:8E:69:CB:78:6B:14:98:27
Signature algorithm name: SHA256withRSA Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit Subject Public Key Algorithm: 2048-bit RSA key
Version: 3 Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 25 F0 0F 60 F5 2F B2 CF D9 44 7A 9F 35 EE FC F3 %.../...Dz.5...
0010: FF E9 D3 21 ...
]
]

Trust this certificate? [no]: yes Trust this certificate? [no]: yes
Certificate was added to keystore
D:\Ubuntu\Diseno de Sistemas Distribuidos\Tarea 4\ClienteEnvio>

```

Importamos el certificado para el Cliente GET

```

D:\Ubuntu\Diseno de Sistemas Distribuidos\Tarea 4\ClienteRecibo> cd ..\cliente_recibo> keytool -import -alias certificado_servidor -file certificado_servidor.pem -keystore cliente_keystore.jks -storepass 1234567890
Owner: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Issuer: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Serial number: 9b3ea6557edf202f
Valid from: Sun Apr 02 17:30:43 CST 2023 until: Sat Jul 01 17:30:43 CST 2023
Certificate fingerprints:
SHA1: 39:18:EF:2E:C7:49:BE:F2:06:EC:77:77:68:14:A2:37:F7:14:5B:46
SHA256: 77:C2:41:3E:10:98:B7:BC:DE:DE:EE:DF:AA:2D:13:33:18:5A:78:B4:98:69:DA:66:8E:69:CB:78:6B:14:98:27
Signature algorithm name: SHA256withRSA Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3 Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 25 F0 0F 60 F5 2F B2 CF D9 44 7A 9F 35 EE FC F3 %.../...Dz.5...
0010: FF E9 D3 21 ...
]
]

Trust this certificate? [no]: yes
Certificate was added to keystore
D:\Ubuntu\Diseno de Sistemas Distribuidos\Tarea 4\ClienteRecibo>

```

Probando el envío de un archivo

Para esta parte se probará el envío de un archivo llamado prueba.txt con el contenido siguiente:

Tarea 4. Transferencia de archivos utilizando sockets seguros

Desarrollar los siguientes programas en lenguaje Java utilizando ChatGPT:

1. Servidor multi-thread con sockets seguros, el cual recibirá del cliente dos tipos de peticiones:

1.1 La petición GET seguida del nombre de un archivo (sin directorios). El servidor deberá leer el archivo del disco local, si el archivo se pudo leer el servidor deberá enviar al cliente OK, la longitud del archivo y el contenido del archivo, de otra manera deberá enviar al cliente ERROR.

1.2 La petición PUT seguida del nombre de un archivo (sin directorios), la longitud del archivo y el contenido del archivo. El servidor deberá escribir el archivo en el disco local, si el archivo se pudo escribir en el disco, el servidor enviará al cliente OK de otra manera enviará ERROR.

2. Cliente PUT. Cliente con sockets seguros, el cual deberá recibir como parámetros la IP del servidor, el puerto que escucha el servidor y el nombre del archivo a enviar.

Entonces el cliente leerá el archivo del disco local, si puede leer el archivo se conectará al servidor y enviará una petición PUT, el nombre del archivo a enviar, la longitud del archivo y el contenido del archivo. Si el cliente no puede leer el archivo del disco local desplegará un mensaje de error indicando esta situación.

El cliente deberá esperar la respuesta del servidor, si el servidor responde OK el cliente deberá desplegar un mensaje indicando que el archivo fue recibido por el servidor con éxito, de otra manera desplegará un mensaje de error indicando que el servidor no pudo escribir el archivo en el disco local.

3. Cliente GET. Cliente con sockets seguros, el cual deberá recibir como parámetros la IP del servidor, el puerto que escucha el servidor y el nombre del archivo a recibir.

Entonces el cliente se conectará al servidor y enviará la petición GET y el nombre del archivo.

El cliente deberá esperar la respuesta del servidor, si el servidor responde OK el cliente deberá recibir la longitud del archivo y el contenido del archivo. Entonces el cliente deberá escribir el archivo en el disco local, si puede escribir el archivo deberá desplegar un mensaje indicando que el archivo se recibió con éxito, de otra manera deberá desplegar un mensaje de error.

El servidor deberá ejecutar en una máquina virtual con Ubuntu en Azure.

El nombre de la máquina virtual deberá ser: "T4-" concatenando el número de boleta del alumno o alumna, por ejemplo, si el número de boleta es 12345678, entonces la máquina virtual deberá llamarse: T4-12345678. No se admitirá la tarea si los nodos no se nombran como se indicó anteriormente.

Recuerden que deben eliminar la máquina virtual cuando no la usen, con la finalidad de ahorrar el saldo de sus cuentas de Azure.

Se deberá subir a Moodle el código fuente (en formato texto) de los clientes y el código fuente del servidor.

Se deberá subir a Moodle un reporte en formato PDF incluyendo portada, las capturas de pantalla de la compilación y ejecución de los clientes y el servidor. Así mismo, el reporte deberá incluir las conclusiones.

Se deberá probar el envío de un archivo mediante el cliente PUT y la recepción de un archivo mediante el cliente GET.

El reporte deberá incluir captura de pantalla de cada paso correspondiente a la creación de la máquina virtual. No se admitirá la tarea si no incluye todas las capturas de pantalla correspondientes a la creación de la máquina virtual.

El reporte deberá incluir la conversación sostenida con ChatGPT.

Valor de la tarea: 25% (1.75 puntos de la primera evaluación parcial)

```

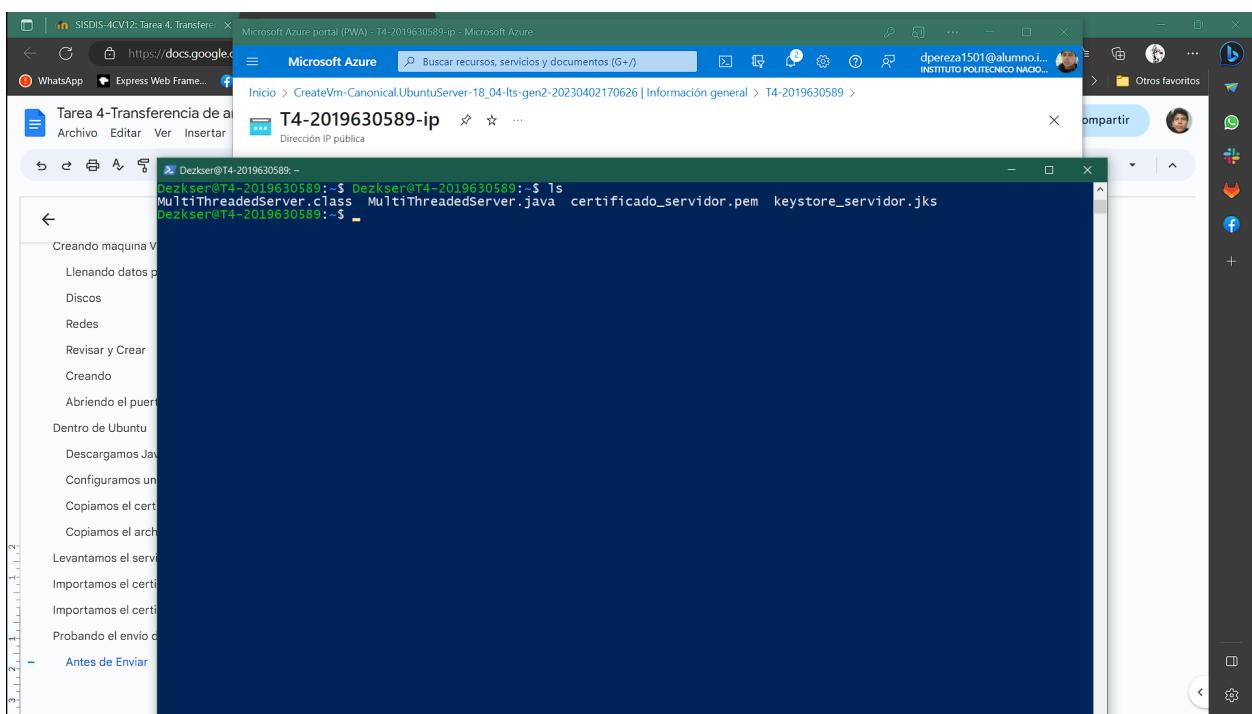
Microsoft Azure portal (PWA) - T4-2019630589-ip - Microsoft Azure
https://docs.google.com/document/d/1JGzXyfjwvDgkVQHdC9WzqNzIwvOOGKt/edit?usp=sharing
dperaza1501@alumno.itesm.mx

Início > Crear > T4-2icado_servidor.pem -keystore keystore_cliente.jks -storepass 1234567890
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteEnvio> keytool -import -alias certificado_servidor -file certificado_servidor.pem -keystore keystore_cliente.jks -storepass 1234567890
Owner: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Issuer: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Serial number: 9b3ea687edf2021
Valid from Sun Jul 01 17:30:43 CST 2023 until: Sat Jul 01 17:30:43 CST 2023
Certificate Fingerprints:
    SHA1: 39:18:EF:2E:C7:49:B1:F2:06:EC:77:77:68:14:A2:37:F7:14:5B:46
    SHA256: 77:C2:41:3E:10:98:B7:BC:DE:DE:EE:DF:AA:2D:13:33:18:5A:78:B4:98:69:DA:66:8E:69:CB:78:6B:14:98:27
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
    SubjectKeyIdentifier [
        KeyIdentifier [
            0000: 25 F0 0F 60 F5 2F B2 CF   D9 44 7A 9F 35 EE FC F3  %.../...Dz.5...
            0010: FF E9 D3 21
        ]
    ]
Propiedades
Bloques

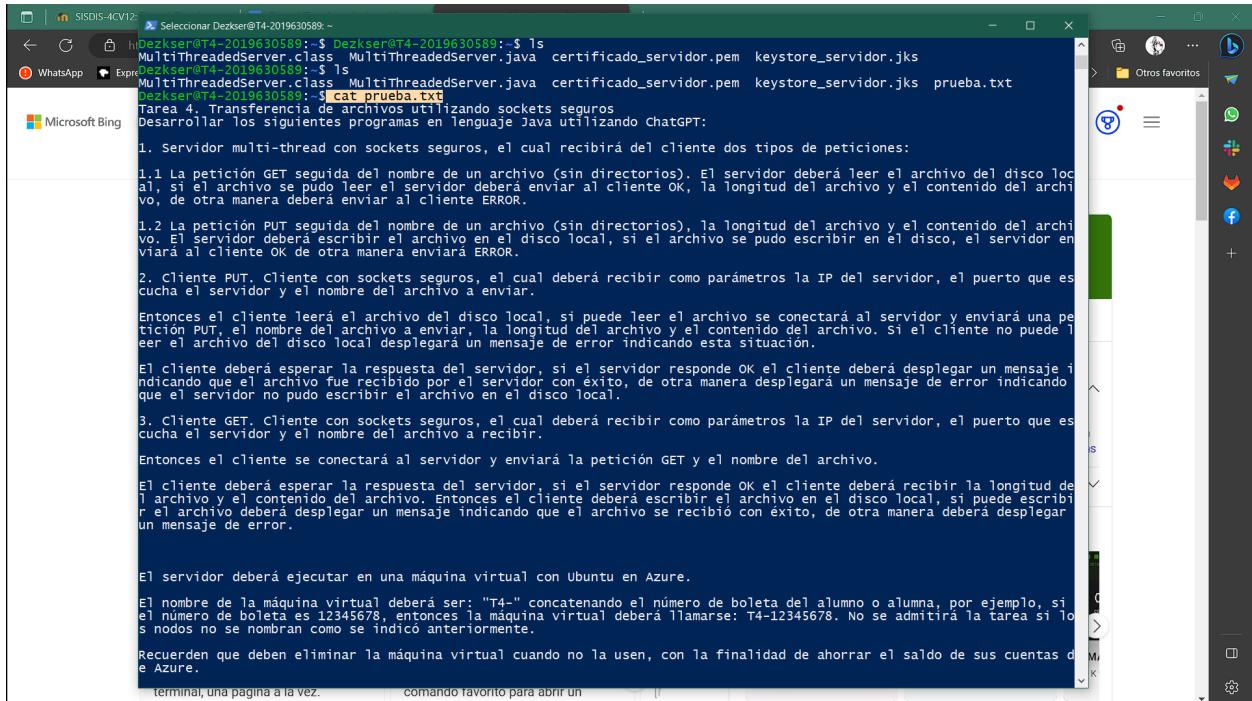
Trust this certificate? [no]: yes
Certificate was added to keystore
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteEnvio> javac -encoding "UTF-8" ClientePUT.java
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteEnvio> java ClientePUT 172.173.149.124 8000 prueba.txt
El archivo fue recibido por el servidor con éxito
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteEnvio>

```

Antes de Enviar



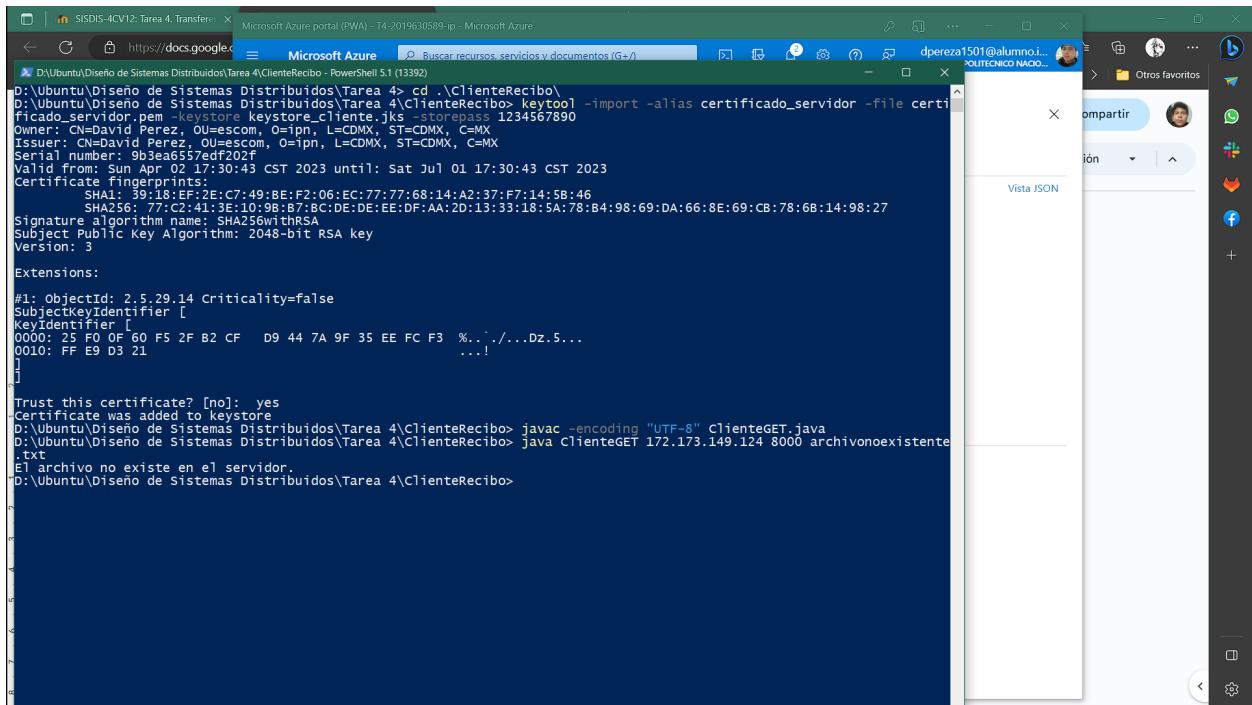
Después de Enviar



Probando la recepción de un archivo

Para este punto vamos a probar primero pidiendo un archivo que no existe en el servidor y después probaremos traer de regreso el archivo que enviamos con el cliente PUT

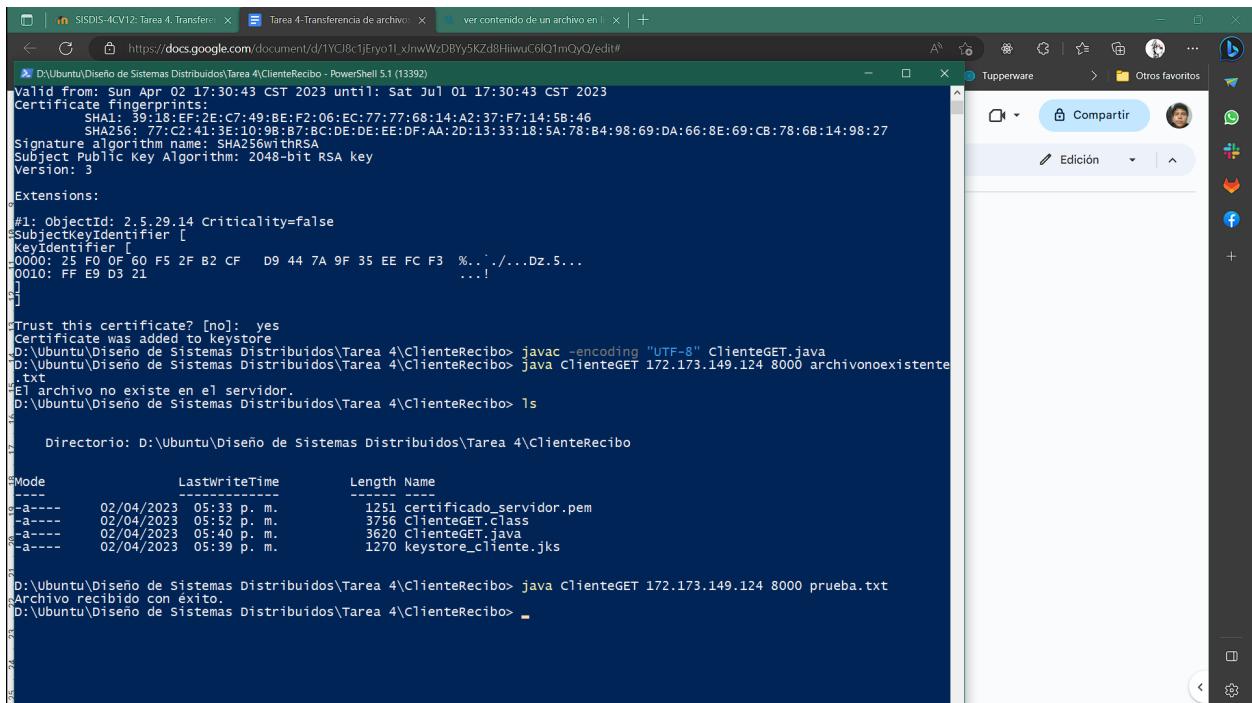
Archivo no existente



```
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> cd ..\Clienterecibo>
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> keytool -import -alias certificado_servidor -file certificado_servidor.pem -keystore cliente_keystore.jks -storepass 1234567890
Owner: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Issuer: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Serial number: 0b3aa557edf202f
Valid from: Sun Apr 02 17:30:43 CST 2023 until: Sat Jul 01 17:30:43 CST 2023
Certificate fingerprints:
    SHA1: 39:18:EF:2E:C7:49:BE:F2:06:EC:77:77:68:14:A2:37:F7:14:5B:46
    SHA256: 77:C2:41:3E:10:98:B7:BC:DE:DE:EE:DF:AA:2D:13:33:18:5A:78:B4:98:69:DA:66:8E:69:CB:78:6B:14:98:27
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
Extensions:
#1. ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 25 F0 0F 60 F5 2F B2 CF   D9 44 7A 9F 35 EE FC F3  %.../...Dz.5...
0010: FF E9 D3 21             ...
]
]

Trust this certificate? [no]: yes
Certificate was added to keystore
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> javac -encoding "UTF-8" ClienteGET.java
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> java ClienteGET 172.173.149.124 8000 archivonoexistente.txt
El archivo no existe en el servidor.
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo>
```

Archivo prueba.txt



```
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> cd ..\Clienterecibo>
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> keytool -import -alias certificado_servidor -file certificado_servidor.pem -keystore cliente_keystore.jks -storepass 1234567890
Owner: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Issuer: CN=David Perez, OU=escom, O=ipn, L=CDMX, ST=CDMX, C=MX
Serial number: 0b3aa557edf202f
Valid from: Sun Apr 02 17:30:43 CST 2023 until: Sat Jul 01 17:30:43 CST 2023
Certificate fingerprints:
    SHA1: 39:18:EF:2E:C7:49:BE:F2:06:EC:77:77:68:14:A2:37:F7:14:5B:46
    SHA256: 77:C2:41:3E:10:98:B7:BC:DE:DE:EE:DF:AA:2D:13:33:18:5A:78:B4:98:69:DA:66:8E:69:CB:78:6B:14:98:27
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
Extensions:
#1. ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 25 F0 0F 60 F5 2F B2 CF   D9 44 7A 9F 35 EE FC F3  %.../...Dz.5...
0010: FF E9 D3 21             ...
]
]

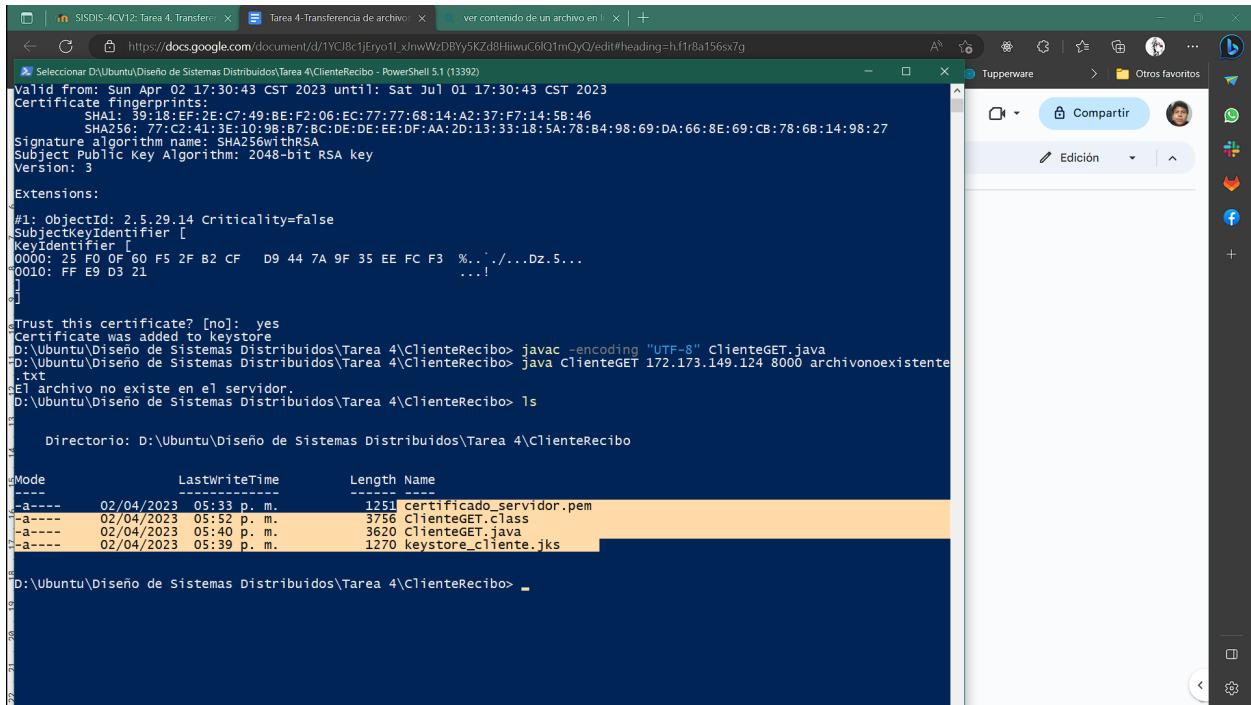
Trust this certificate? [no]: yes
Certificate was added to keystore
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> javac -encoding "UTF-8" ClienteGET.java
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> java ClienteGET 172.173.149.124 8000 archivonoexistente.txt
El archivo no existe en el servidor.
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> ls

Directorio: D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo

Mode                LastWriteTime        Length Name
----                -----        ---- 
-a--- 02/04/2023 05:33 p. m.      1251 certificado_servidor.pem
-a--- 02/04/2023 05:52 p. m.       3756 ClienteGET.class
-a--- 02/04/2023 05:40 p. m.       3620 ClienteGET.java
-a--- 02/04/2023 05:39 p. m.       1270 keystore_cliente.jks

D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo> java ClienteGET 172.173.149.124 8000 prueba.txt
Archivo recibido con éxito.
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\Clienterecibo>
```

Antes de Recibir



```

SISDIS-4CV12: Tarea 4. Transferencia | Tarea 4-Transferencia de archivo | ver contenido de un archivo en | + https://docs.google.com/document/d/1YCl8c1jEryo1LxJnwWzDBYy5KZd8HiiwuC6lQ1mQyQ/edit#heading=h.f18a156sx7g
[...] Seleccionar D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo - PowerShell 5.1 (13392)
Valid from: Sun Apr 02 17:30:43 CST 2023 until: Sat Jul 01 17:30:43 CST 2023
Certificate fingerprints:
SHA1: 39:18:EF:2E:C7:49:BE:F2:06:EC:77:77:68:14:A2:37:F7:14:5B:46
SHA256: 77:02:41:3C:10:98:87:6C:DE:EE:DF:AA:2D:13:33:18:5A:78:B4:98:69:DA:66:8E:69:CB:78:6B:14:98:27
signature algorithm name: SHA256withRSA
subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 25 F0 0F 60 F5 2F B2 CF   D9 44 7A 9F 35 EE FC F3  ... ./...Dz.5...
0010: FF E9 D3 21             ...
]

Trust this certificate? [no]: yes
Certificate was added to keystore
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo> javac -encoding "UTF-8" ClienteGET.java
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo> java ClienteGET 172.173.149.124 8000 archivonoexistente.txt
El archivo no existe en el servidor.
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo> ls

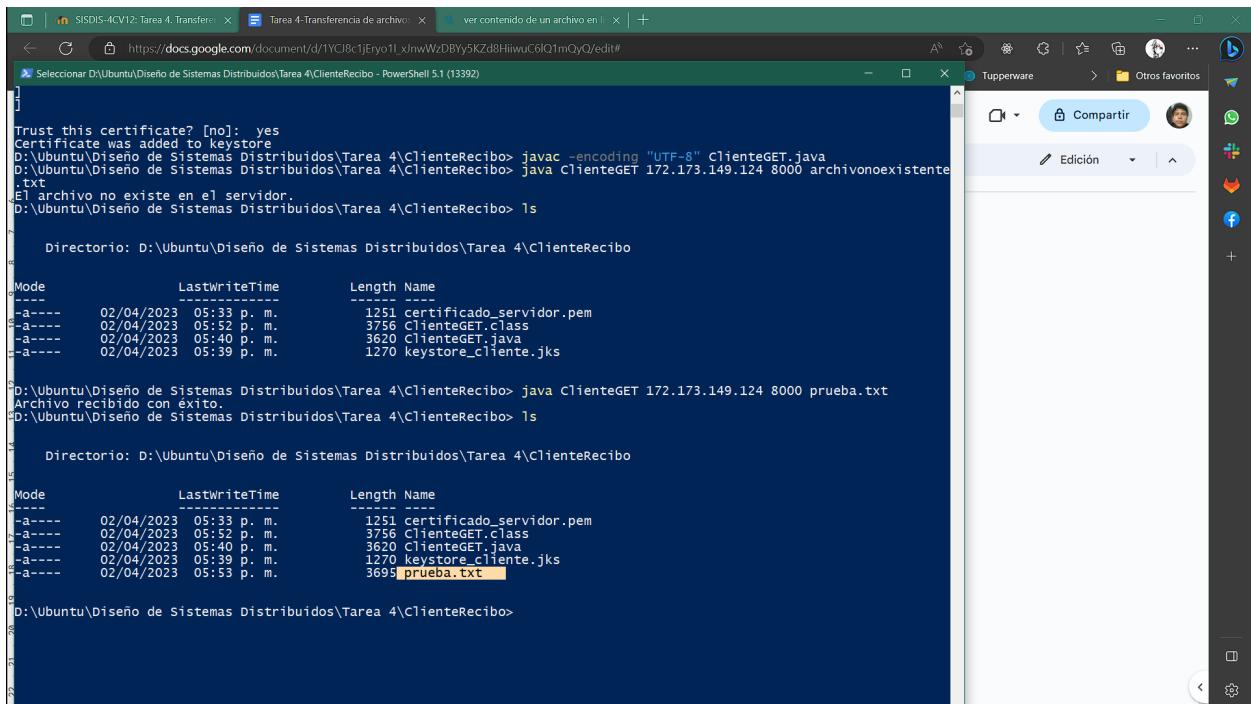
    Directorio: D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo

Mode           LastWriteTime       Length Name
----           -----          ---- 
-a----  02/04/2023 05:33 p. m.    1251 certificado_servidor.pem
-a----  02/04/2023 05:52 p. m.    3756 ClienteGET.class
-a----  02/04/2023 05:40 p. m.    3620 ClienteGET.java
-a----  02/04/2023 05:39 p. m.    1270 keystore_cliente.jks

D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo>

```

Después de recibir



```

SISDIS-4CV12: Tarea 4. Transferencia | Tarea 4-Transferencia de archivo | ver contenido de un archivo en | + https://docs.google.com/document/d/1YCl8c1jEryo1LxJnwWzDBYy5KZd8HiiwuC6lQ1mQyQ/edit#
[...] Seleccionar D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo - PowerShell 5.1 (13392)
Trust this certificate? [no]: yes
Certificate was added to keystore
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo> javac -encoding "UTF-8" ClienteGET.java
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo> java ClienteGET 172.173.149.124 8000 archivonoexistente.txt
El archivo no existe en el servidor.
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo> ls

    Directorio: D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo

Mode           LastWriteTime       Length Name
----           -----          ---- 
-a----  02/04/2023 05:33 p. m.    1251 certificado_servidor.pem
-a----  02/04/2023 05:52 p. m.    3756 ClienteGET.class
-a----  02/04/2023 05:40 p. m.    3620 ClienteGET.java
-a----  02/04/2023 05:39 p. m.    1270 keystore_cliente.jks
-a----  02/04/2023 05:53 p. m.    3695 prueba.txt

D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo> java ClienteGET 172.173.149.124 8000 prueba.txt
Archivo recibido con éxito.
D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo> ls

    Directorio: D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo

Mode           LastWriteTime       Length Name
----           -----          ---- 
-a----  02/04/2023 05:33 p. m.    1251 certificado_servidor.pem
-a----  02/04/2023 05:52 p. m.    3756 ClienteGET.class
-a----  02/04/2023 05:40 p. m.    3620 ClienteGET.java
-a----  02/04/2023 05:39 p. m.    1270 keystore_cliente.jks
-a----  02/04/2023 05:53 p. m.    3695 prueba.txt

D:\Ubuntu\Diseño de Sistemas Distribuidos\Tarea 4\ClienteRecibo>

```

Conclusiones

La práctica presentó un gran reto en el sentido del uso de sockets seguros, pues es muy poco intuitivo su uso, al final después de varios intentos y explicaciones de Chat GPT se logró usar correctamente los sockets seguros, el procedimiento creación de la máquina virtual fue algo ya muy sencillo, al igual que la lógica de programación del envío y recepción del archivo, pues prácticamente fue realizado en su totalidad por el Chat.