



Instituto Politécnico Nacional  
Escuela Superior de Cómputo



Tarea 2

*“Implementación de un token-ring”*

Integrantes:

Jaime Villanueva Héctor Israel

Juárez Espinoza Ulises

Machorro Vences Ricardo Alberto

Equipo 7

Desarrollo de sistemas distribuidos

4CV11

26 de febrero de 2022

## Introducción

A finales de los años 80s, se desarrolló el protocolo HTTP (HyperText Transfer Protocol) por Tim Berners-Lee, y este protocolo de comunicación es utilizado para navegar por las diferentes páginas web. En HTTP toda la información es enviada en texto en claro, haciendo que toda nuestra actividad y datos ingresados estén expuestos a ser monitoreados o usados por gente con intenciones maliciosas. Por ello aparece el protocolo HTTPS que, a diferencia de HTTP, éste maneja la parte de seguridad y lo hace encriptando los datos que se envían entre el cliente y el servidor.

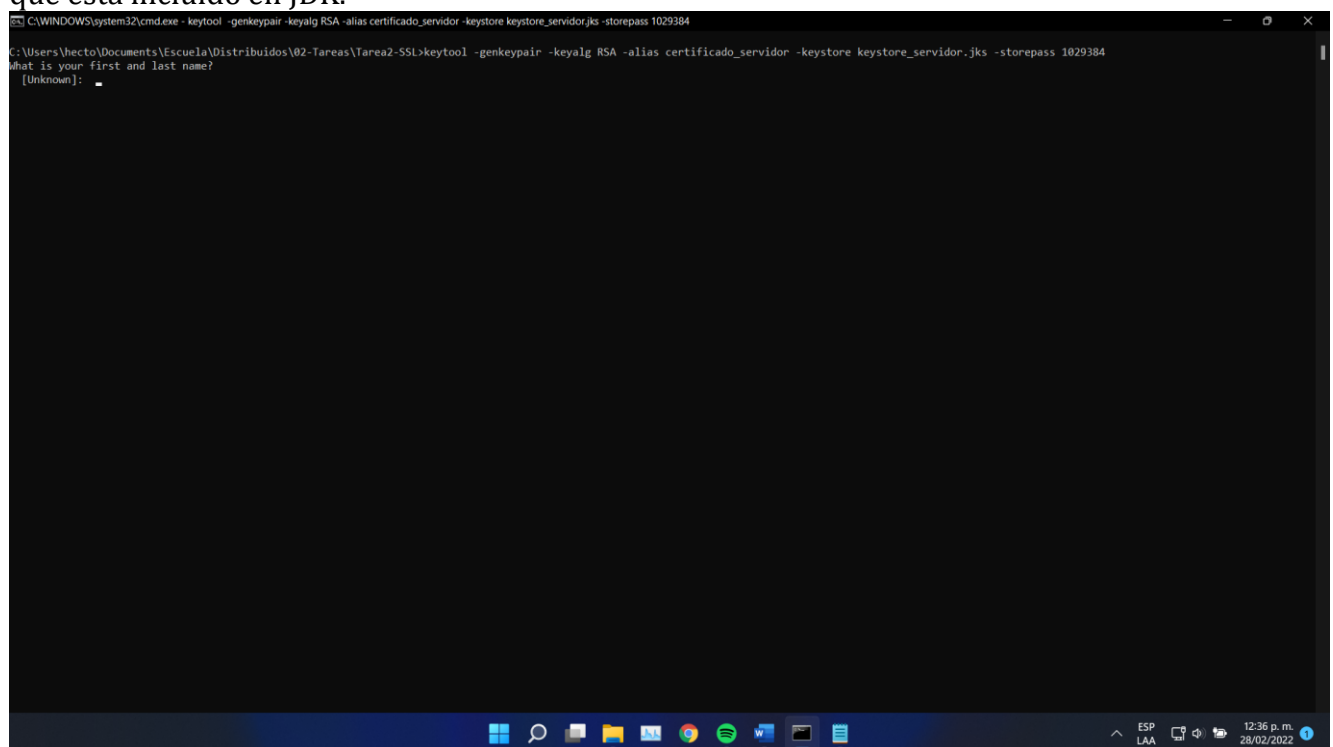
HTTPS incorpora la funcionalidad de seguridad a través del protocolo SSL (Secure Sockets Layer) el cual permite al cliente autenticar la identidad del servidor. Para hacer esta autenticación se ocupa un certificado digital que es un documento electrónico que contiene datos de la identidad, clave pública, firma digital, entre otros de la persona u organización.

Existen dos tipos de certificados para poder identificarnos: el certificado autofirmado que es aquel que el usuario crea y el certificado firmado por una autoridad certificadora (CA).

Para esta práctica, se implementará un token que se enviará de un nodo a otro mediante sockets seguros, en una topología lógica de anillo. Este constará de seis nodos y cada uno de estos será un cliente y un servidor. El cliente deberá hacer re-intentos de conexión e inicialmente el nodo 0 enviará el número 0 (token) al nodo 1. Cuando el nodo n reciba el token lo incrementará, lo mostrará en consola y lo enviará al siguiente nodo. El programa terminará cuando en el nodo 0 el token se mayor o igual a 500.

## Desarrollo

Antes de crear nuestro programa, creamos nuestros certificados utilizando el programa keytool que está incluido en JDK.

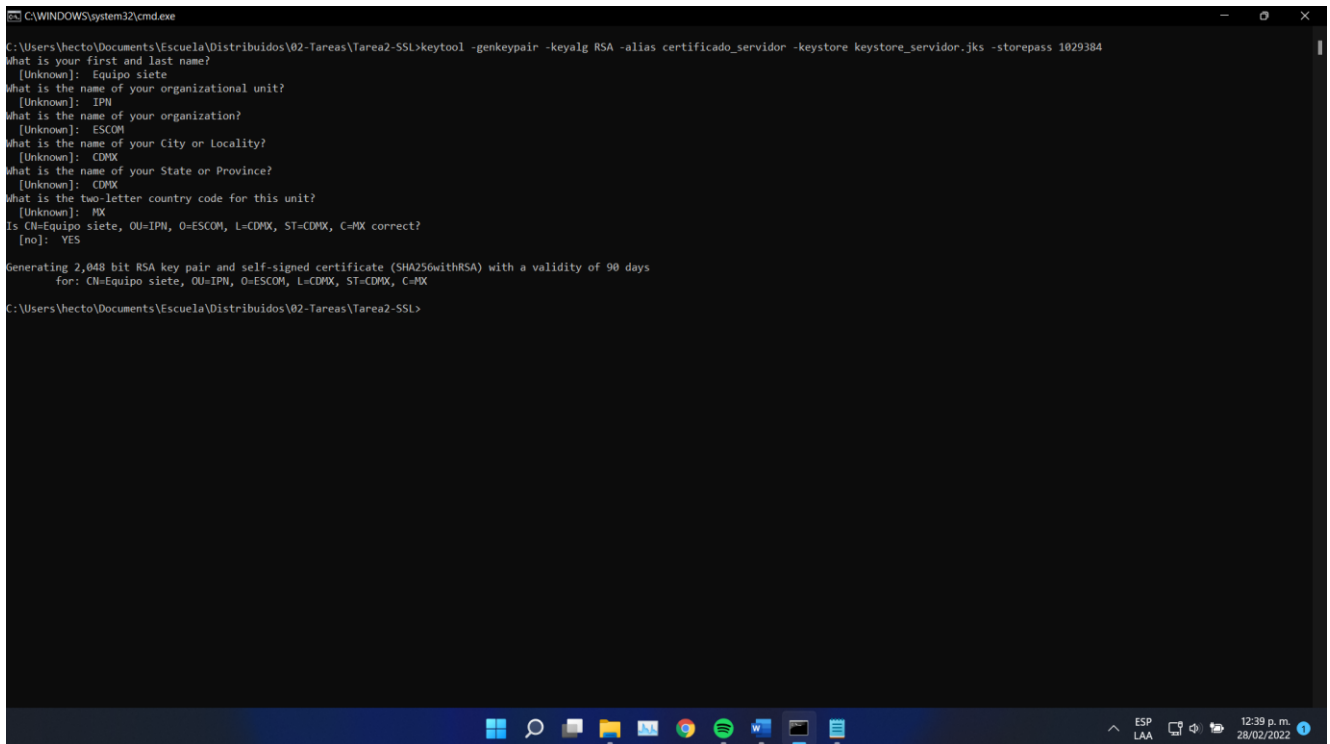


```
C:\WINDOWS\system32\cmd.exe - keytool -genkeypair -keyalg RSA -alias certificado_servidor -keystore keystore_servidor.jks -storepass 1029384
What is your first and last name?
[Unknown]:
```

Ilustración 1 Creación de certificado autofirmado

Como podemos ver en la ilustración 1, se utiliza keytool que es el programa que nos ayudará a crear el certificado, -genkeypair nos va a generar una clave pública (se pone dentro del certificado) y una privada, -keyalg es el algoritmo que se va a utilizar para generar las claves y en este caso será RSA (criptografía asimétrica), -alias es el alias con el que vamos a identificar nuestro certificado, -keystore es el lugar donde se va a almacenar nuestro certificado y -storepass que es la contraseña que va a tener el keystore.

Cuando ejecutemos la instrucción, nos pedirá que capturemos los datos que va a tener el certificado. Podemos darnos cuenta que se crea un archivo llamado keystore\_servidor.jks en el lugar donde se ejecutó el código.



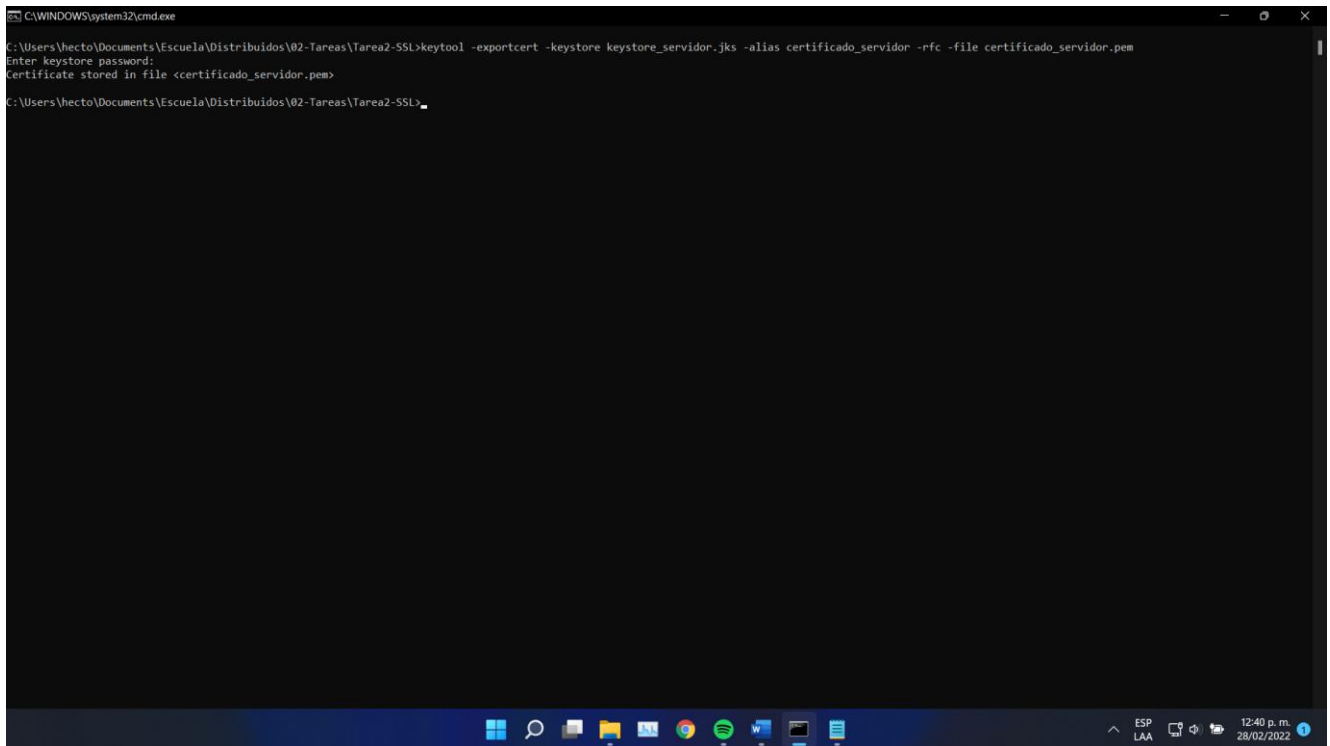
```
C:\WINDOWS\system32\cmd.exe
C:\Users\hecto\Documents\Escuela\Distribuidos\02-Tareas\Tarea2-SSL>keytool -genkeypair -keyalg RSA -alias certificado_servidor -keystore keystore_servidor.jks -storepass 1029384
What is your first and last name?
[Unknown]: Equipo siete
What is the name of your organizational unit?
[Unknown]: IPN
What is the name of your organization?
[Unknown]: ESCOM
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=Equipo siete, OU=IPN, O=ESCOM, 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=Equipo siete, OU=IPN, O=ESCOM, L=CDMX, ST=CDMX, C=MX

C:\Users\hecto\Documents\Escuela\Distribuidos\02-Tareas\Tarea2-SSL>
```

*Ilustración 2 Datos que contendrá el certificado autofirmado*

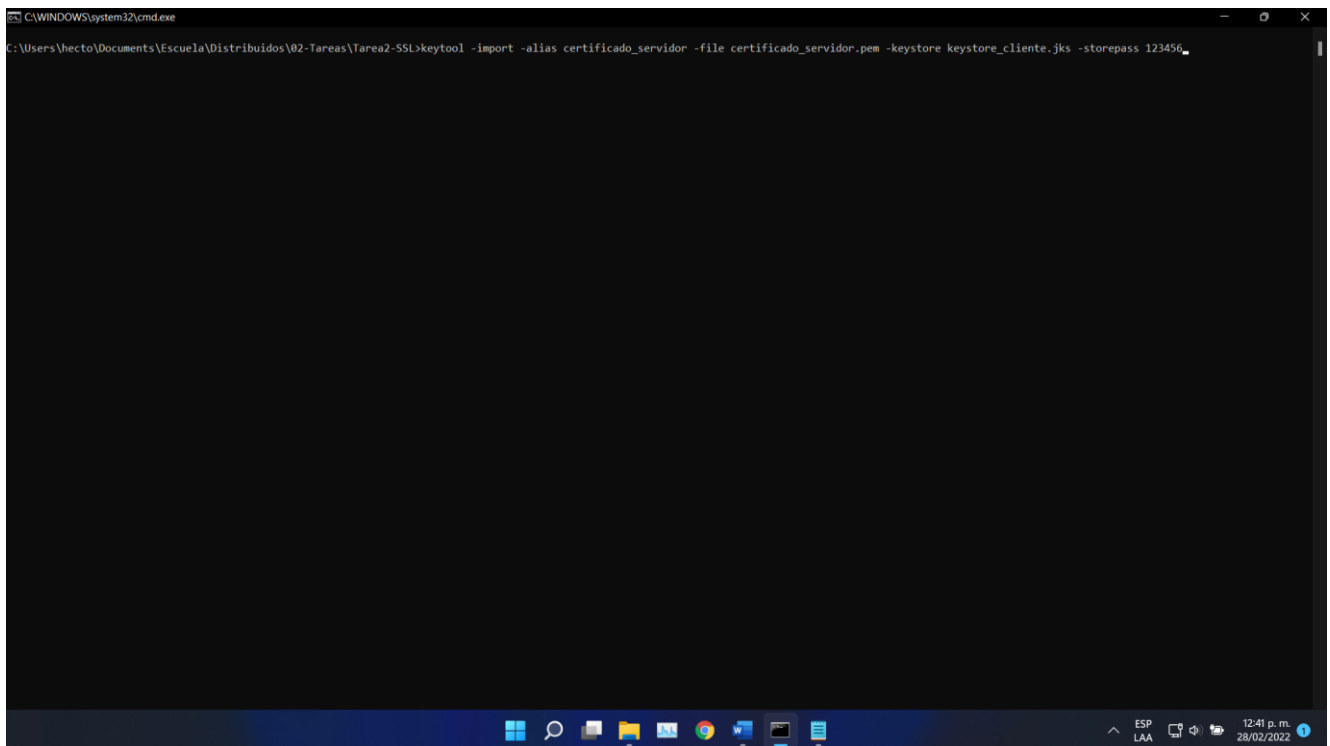
Luego ejecutaremos la instrucción de la ilustración 3 para que podamos obtener el certificado contenido en el keystore. La opción exportcert va a leer del keystore el certificado por el alias y va a generar un archivo de texto que contiene el certificado. El nombre del archivo será certificado\_servidor.pem.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\hecto\Documents\Escuela\Distribuidos\02-Tareas\Tarea2-SSL>keytool -exportcert -keystore keystore_servidor.jks -alias certificado_servidor -rfc -file certificado_servidor.pem
Enter keystore password:
Certificate stored in file <certificado_servidor.pem>
C:\Users\hecto\Documents\Escuela\Distribuidos\02-Tareas\Tarea2-SSL>
```

*Ilustración 3 Se obtiene el certificado*

Por último, se va a crear una keystore que va a ser el que ocupará el cliente y este keystore tendrá el certificado del servidor. Este archivo tendrá el nombre de keystore\_cliente.jks.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\hecto\Documents\Escuela\Distribuidos\02-Tareas\Tarea2-SSL>keytool -import -alias certificado_servidor -file certificado_servidor.pem -keystore keystore_cliente.jks -storepass 123456
```

*Ilustración 4 Creación de keystore para el cliente*

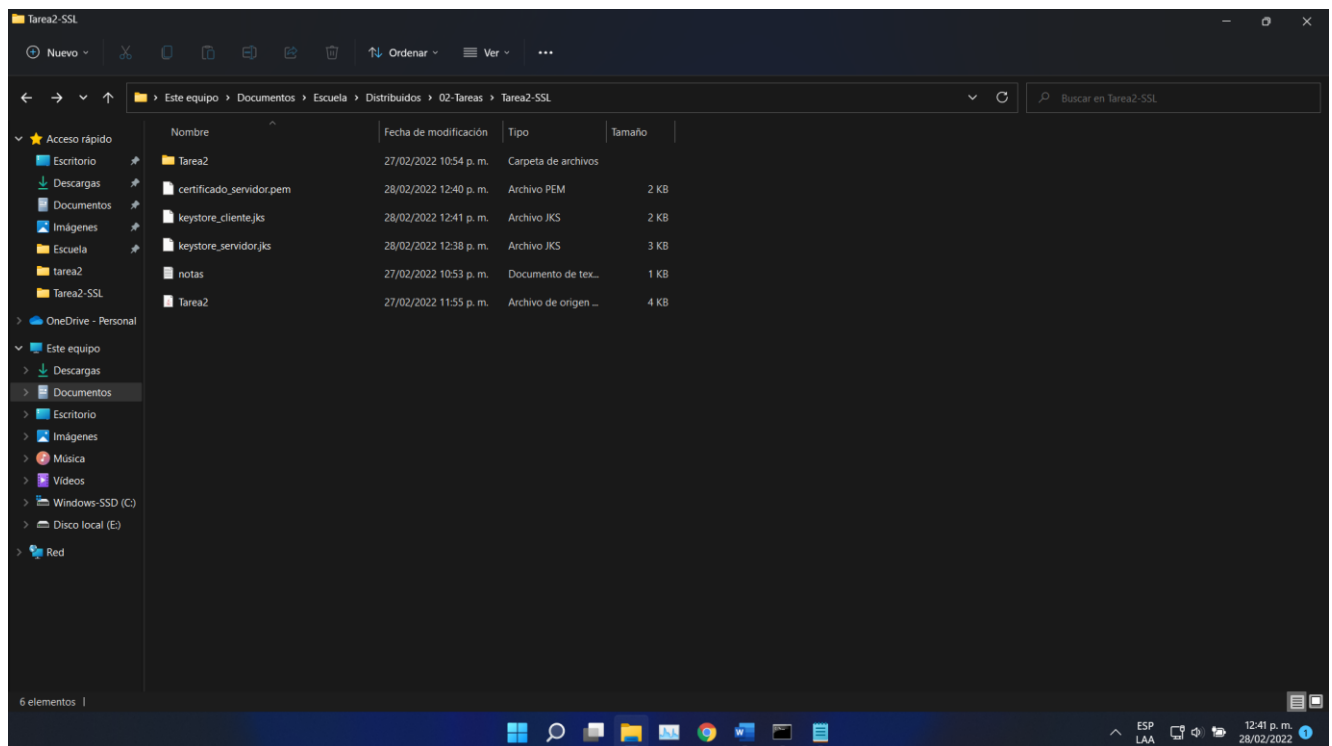
```
C:\WINDOWS\system32\cmd.exe
C:\Users\hecto\Documents\Escuela\Distribuidos\02-Tareas\Tarea2-SSL>keytool -import -alias certificado_servidor -file certificado_servidor.pem -keystore keystore_cliente.jks -storepass 123456
Owner: CN=Equipo siete, OU=IPN, O=ESCOM, L=CDMX, ST=CDMX, C=MX
Issuer: CN=Equipo siete, OU=IPN, O=ESCOM, L=CDMX, ST=CDMX, C=MX
Serial number: 3a2240cd2343e7a5
Valid from: Mon Feb 28 12:38:57 CST 2022 until: Sun May 29 13:38:57 CDT 2022
Certificate fingerprints:
    SHA1: 41:B6:DA:7E:68:25:FE:0A:8B:D8:A3:18:CB:10:E2:48:A4:AA:A5:60
    SHA256: C5:90:28:E6:2D:E5:B5:73:AC:9E:92:64:C7:AA:23:D0:61:D6:1C:12:58:B5:86:48:33:FF:6C:9D:DF:D1:56:53
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: 8C 86 55 D1 1F BF 80 62 29 CA 24 66 28 F2 0F 57 ..U...b).$(.W
0010: EC E1 78 94 ..X.
]
]

Trust this certificate? [no]: yes
Certificate was added to keystore

C:\Users\hecto\Documents\Escuela\Distribuidos\02-Tareas\Tarea2-SSL>
```

*Ilustración 5 Resultado de la creación del keystore del cliente*



*Ilustración 6 Archivos generados*

Con estos archivos podemos crear nuestro programa haciendo uso de sockets seguros.

Para poder correr el programa es necesario pasarle como argumento el número de nodo que se quiere ejecutar que puede ser del 0 al 5. El nodo 0 será el que envíe el primer valor al siguiente nodo.

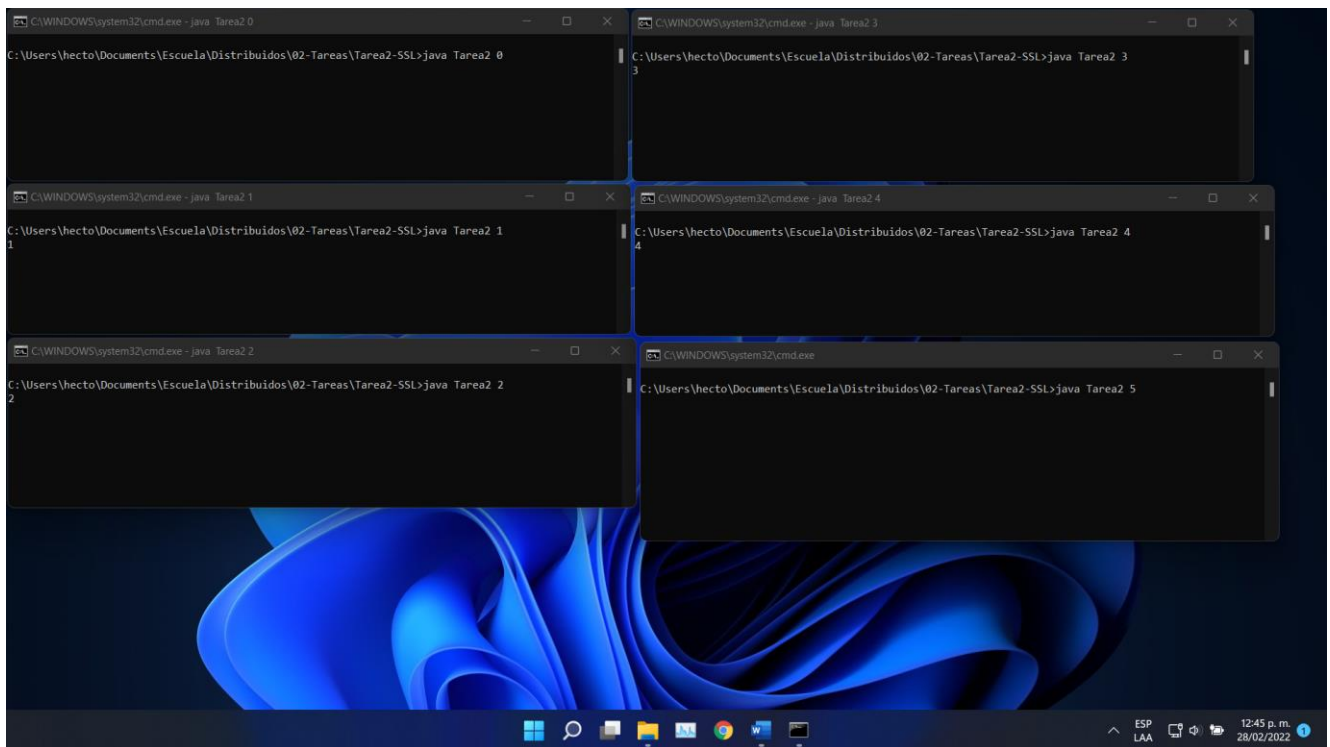
Cada uno de los nodos va a desempeñar el papel de cliente y servidor teniendo cada uno de estos,

su respectivo puerto.

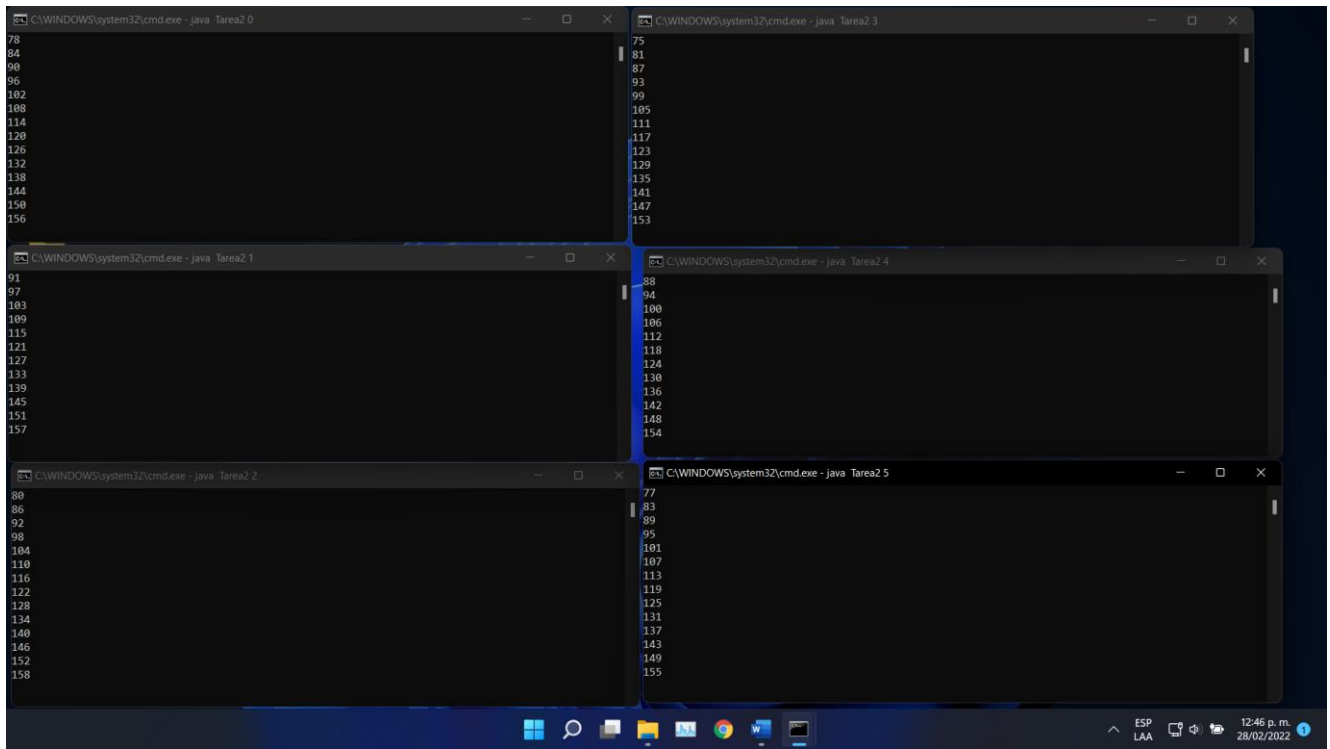
Dentro del programa se indica el keystore del servidor y del cliente con sus respectivas contraseñas para hacer uso del certificado que creamos.

Cuando el nodo esté desempeñando el papel de cliente, primero se crea una instancia de la clase `SSLConnectionFactory`, después creamos un socket con el que podremos comunicarlo con el siguiente nodo y abrimos un stream de salida. Con ese stream vamos a poder enviar un dato al otro nodo y al termino, va a esperar por 50 milisegundos con la intención de que muestra por pantalla el número y se cerrará la conexión.

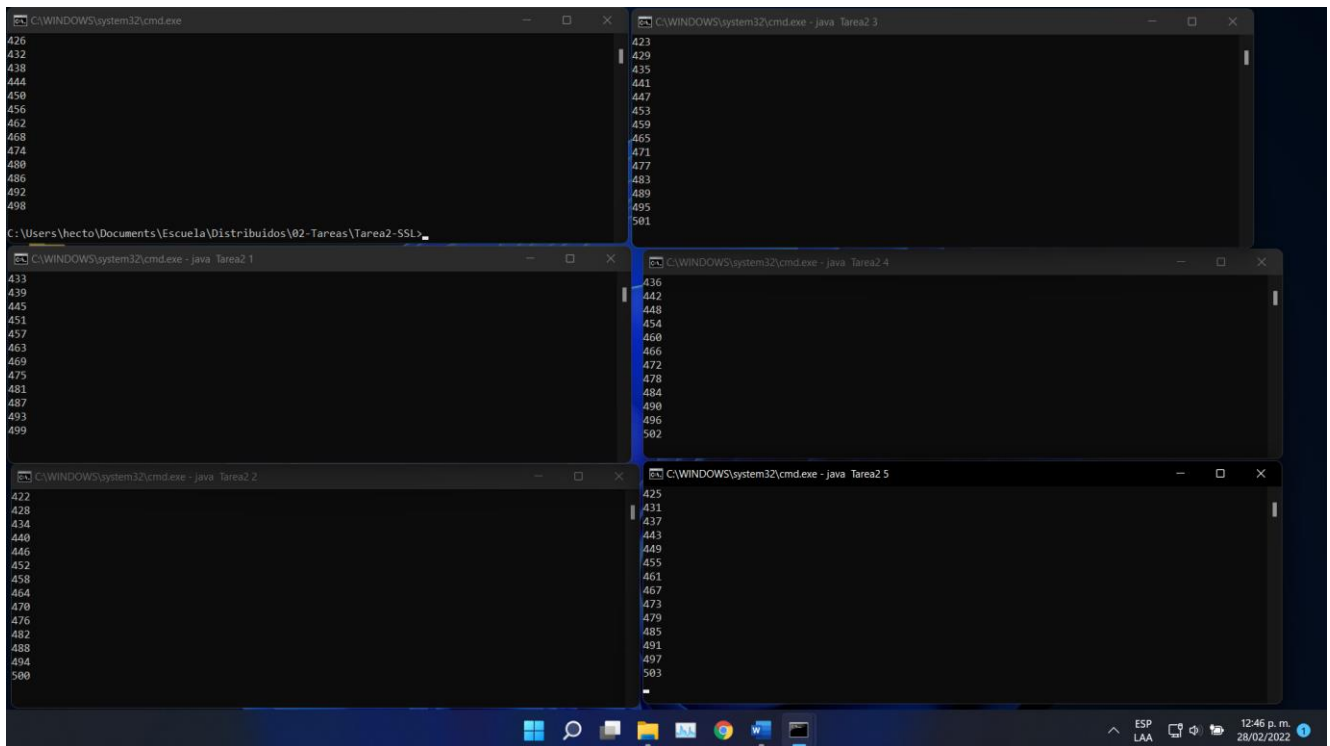
Cuando se esté desempeñando el papel de servidor, se crea una instancia de la clase `SSLServerSocketFactory`, se crea un socket servidor que va a utilizar el método `createServerSocket` y se invoca el método `accept` de la clase `ServerSocket`, el cual regresa un socket cuando se recibe una conexión. Se abre un stream de entrada para recibir el dato enviado por el cliente y cuando se recibe, este se guarda en una variable. Después, se cierra la conexión del servidor, se hace un incremento del dato recibido y el nodo empieza a ejercer ahora el papel de cliente.



*Ilustración 7 Ejecución del programa*



*Ilustración 8 Programa en ejecución*



*Ilustración 9 Fin de ejecución del programa*

Cuando el dato llega al nodo 0, se verifica que no sea mayor o igual a 500, de lo contrario termina su ejecución. Esto lo podemos ver en la imagen 9.

## Conclusiones

### Ricardo Alberto Machorro Vences

*En esta tarea se pudo ver que la comunicación entre elementos que usan sockets no es necesario la sincronización o el uso de hilos, sino que, según los requerimientos y necesidades del programa, aunque se necesite que un elemento funcione tanto como servidor y cliente se puede hacer en el método principal sin tener alguna deficiencia significativa y que en realidad daría una ventaja en lo entendible del código y la facilidad de creación de este. Este programa además me hizo consiente de otros aspectos en los que difieren los sockets normales y los seguros y es la cuestión de tiempo que tardan en la comunicación ya que los seguros por el hecho de autenticación y uso de permisos junto el uso de algoritmos de cifrado y descifrado hace que se tarde más en la comunicación, haciendo que sea más importante la consideración de cierre de conexiones ya que esto puede hacer fallas cuando el cliente o el servidor cierran comunicación antes de tiempo, así que es importante tomar en cuenta el dar un poco de tiempo para dejar que los demás procesos de ambos lados de la comunicación terminen sus procesos. En conclusión, esta tarea mostro como a veces algunas acciones tienen soluciones sencillas y sobre algunas diferencias claves cuando se trabaja con sockets seguros.*

### Héctor Israel Jaime Villanueva

*Pude observar que no es complicada la implementación de los sockets seguros en un programa, pero que tienen una gran importancia en cuestión de seguridad hacia el cliente. También aprendí la facilidad con la que uno puede crear sus propios certificados digitales y la posibilidad de que un nodo pudiera ejercer el papel tanto de cliente como de servidor.*

### Juárez Espinoza Ulises

*En esta tarea se ocuparon varios de los conceptos que hemos venido repasando a lo largo de las clases anteriores, tales como el uso de sockets seguros, para lo cual fue necesario crear un certificado y crear las llaves tanto para el servidor como para el cliente.*

*También se volvió a implementar el uso de un solo programa que fungiera como cliente o servidor dependiendo de los parámetros recibidos al momento de su ejecución. Fue interesante porque en este caso se necesitaban 6 nodos, donde cada uno podía ser cliente y servidor, hacerlo dentro del mismo programa fue algo que debimos de ingeniar para lograrlo, pudimos ver que no es necesario usar instrucciones como la de sincronización. Pudimos solucionar esta demanda mediante el uso de sentencias conocidas en java en este caso switch. Así mismo debido a la demanda del programa fuimos cuidadosos en cerrar las conexiones una vez que el nodo correspondiente había cumplido su función, dejando un pequeño sleep para que tanto cliente y servidor fueran capaces de enviar o recibir los datos de manera correcta.*

## Referencias

Profe Sang. SSL, TLS, HTTPS, HTTP - Explicado Fácilmente. (14 de abril de 2021). Accedido el 27 de febrero de 2022. [Video en línea]. Disponible:  
<https://www.youtube.com/watch?v=6HJAWFenYx8>



"IBM Docs". IBM - Deutschland | IBM. [https://www.ibm.com/docs/es/was-zos/8.5.5?topic=SS7K4U\\_8.5.5/com.ibm.websphere.ihs.doc/ihs/sec\\_overview.html](https://www.ibm.com/docs/es/was-zos/8.5.5?topic=SS7K4U_8.5.5/com.ibm.websphere.ihs.doc/ihs/sec_overview.html) (accedido el 28 de febrero de 2022).

Boxcryptor. "Cifrado AES y RSA". Boxcryptor – No. 1 Cloud Encryption Made in Germany. <https://www.boxcryptor.com/es/encryption/> (accedido el 28 de febrero de 2022).