Práctica B – Curso 2021/2022.

Francisco J. Fernández Masaguer

10/9/2021

En esta práctica se desarrolla una sencilla aplicación segura en red sobre protocolo SSL/TLS. La aplicación se desarrolla usando el paquete Java JSSE para el protocolo SSL y las APIs criptográficas, JCA y JCE. La práctica permitirá al alumno:

- Introducirse al uso de una librería criptográfica moderna (Java JCA/JCE).
- Profundizar en la funcionalidad, prestaciones, utilización y problemática del protocolo SSL/TLS y en su implementación en el paquete Java JSSE.
- Adquirir familiaridad con una herramienta de gestión de claves y certificados (*KeyExplorer*).
- Aprender a manejar certificados usando una clase Java.
- Utilizacion del paquete OpenSSL para gestionar la revocacion de certificados OCSP.
- Formarse en el desarrollo de productos de ingeniería.

1 Descripción del servicio.

1.1 Funcionalidad básica.

Se desarrollará una aplicación cliente-servidor para registrar documentos de forma segura. De acuerdo con la figura 1, en la aplicación habrá dos agentes:

- el cliente de registro (que hace las veces de oficina de registro y propietario del documento) y
- el servidor de registro (que hace las veces de registrador).

En la figura 1, *keystore* y *truststore*[1] son los almacenes de credenciales y confianza, respectivamente, usados por el programa cliente y el programa servidor. Cada participante (cada usuario y el servidor) en el sistema tendrá su propio *keystore* y *truststore:*

- El *keystore* (almacen de credenciales) guarda las credenciales que cada usuario (clientes o servidor) necesita para poder autenticarse correctamente ante los demás.
- El *truststore* (almacen de confianza) guarda los certificados necesarios para poder verificar la autenticación de los demas usuarios.

[1] El tipo de almacen de claves para cliente y servidor debe ser *"jce"*, pues este tipo de almacen permite claves simétricas, a diferencia del *jks*.

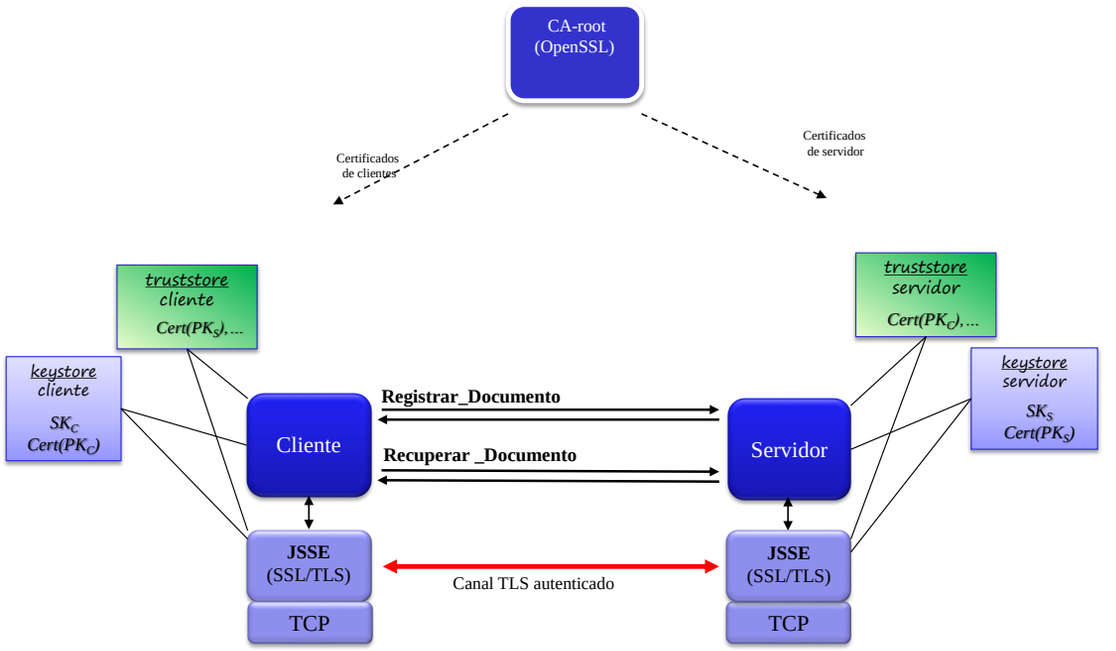

Figure 1: Componentes de la aplicacion

Por ejemplo, cuando el servidor se autentica ante el cliente, el servidor toma sus credenciales del *keystore* del servidor, y el cliente, cuando recibe esas credenciales, usa la informacion del *trutstore* de cliente para verificar que esas credenciales son correctas.

Cada usuario, asi como el servidor, deberá poseer dos certificados:

1. *CertAuth.* Certificado de autenticacion, usado para la autenticacion a nivel de capa TLS/SSL y para el cifrado de documentos a nivel de aplicación.

2. *CertFirma.* Certificado de firma, usado para la validacion, a nivel de aplicación, de las firmas que realice el usuario.

Estos certificados deberan creare o generarse con la misma identidad de usuario (subject name).

De forma general, la aplicacion constará de 3 servicios (El canal SSL usado para estos 3 servicios deberá ser un *canal SSL con autenticación de cliente y de servidor).* :

1. *Registrar_Documento.* Mediante este servicio el cliente o propietario de un documento, enviará el documento cifrado al servidor, por un canal SSL seguro, para que este lo registre y almacene.

2. *Listar_Documentos.* Permitira al usuario conocer la relacion de documentos publicos y sus privados guardados en el registrador.

3. *Recuperar_Documento.*Mediante este servicio, el propietaro podrá, posteriormente, recuperar un documento firmado y almacenado por el registrador.

El detalle concreto de cada servicio es el siguiente:

1. **REGISTRAR_DOCUMENTO.**
   PETICION.

   REGISTRAR_DOCUMENTO.REQUEST *($CertAuth_C$, nombreDoc, tipoConfidencialidad, $E_{PK_S}(K), E_K(documento)$, firmaDoc [,$CertFirma_C$]).*

   En este servicio:

   - $CertAuth_C$ es el certificado de autenticación del cliente (que conlleva su identidad).
   - *nombreDoc* es un nombre, de una longitud maxima de 100 caracteres, para el documento.
   - *tipoConfidencialidad* (con valores PRIVADO y PUBLICO) representa si el documento ha de guardarse por el registrador de forma confidencial o no respectivamente.
   - *documento* es el contenido del fichero (cualquier tipo de fichero) con la información a registrar.
   - $E_{PK_S}(K), E_K(documento)$, es el **cifrado PGP** del documento (cifrado de capa de aplicación). Para realizar el cifrado PGP:
     - Primero se genera una clave simetrica K de forma ALEATORIA con la que se cifra el documento, obteniendose la parte $E_K(documento)$. La clave K deberá generarse de forma aleatoria usando uno de los metodos java incluidos en el JCA/JCE. El cifrado simetrico correspondiente se realizará usando el algoritmo AES con claves de 196 bits y modo CBC.
     - Luego se cifra la clave K por medio de un cifrado de clave publica, usando la clave publica del servidor tomada del certificado $CertAuth_S$ del servidor (previamente almacenado en el trustore del cliente). Esta clave pública debera corresponder a una clave RSA (puesto que el RSA soporta cifrado). Esta es la parte denotada por $E_{PK_S}(K)$.
   - Si *tipoConfidencialidad*=PRIVADO **SI** se realizará el cifrado PGP anterior, pero si *tipoConfidencialidad*=PUBLICO no se realizará, enviándose el documento en claro (sin el cifrado PGP).
   - *firmaDoc*. Sera la firma del propietario sobre el documento, es decir: $Sig_{propietario}(documento)$, realizada con la clave de firma. Estas firmas se deberán poder hacer tanto con claves DSA como con claves RSA.
   - $CertFirma_C$. Es el certificado de clave publica de firma del propietario. Como se ha comentado anteriormente, este certificado debera generarse usando la misma identidad de propietario (*idPropietario*), que la usada por el certificado $CertAuth_C$. Tal y como se indica al ponerlo entre corchetes, el envio de este certificado es opcional. Si no se envia el alumno ha de introducirlo previamente de forma manual en alguno de los almacenes de claves del servidor.

   Tanto el documento como su firma deberan almacenarse temporalmente en el cliente hasta la respuesta del servidor. El cliente enviara el documento al servidor por un canal SSL con una suite que incorpore la autenticacion pero no la confiddencialidad.

   Sobre el documento recibido por el canal SSL, el servidor:

- Validara el Certificado $CertFirma_C$. Si es incorrecto devolverá el mensaje de error "CERTIFICADO DE FIRMA INCORRECTO". La validacion debera incluir la comprobacion de que las identidades de los usuarios de los certificados de autenticacion y firma coincide. Para obtener esta identidad a partir del certificado se puede usar el metodo *X509Certificate.getSubjectDN()*.
- Verificará la firma del documento. Si es incorrecta devolvera el mensaje de error "FIRMA INCORRECTA".
- Si el documento viene con caracter confidencial (*tipoConfidencialidad*=PRIVADO), primero descifrará el documento usando la clave privada correspondiente al $CertAuth_S$. Luego, antes de almacenarse, se cifrara el documento con una clave simétrica solo conocida por el servidor (la misma para todos los documentos recibidos de los clientes). Este cifrado se realizará usando un algoritmo de cifrado simetrico cualquiera distinto al AES, con clave de 196 bis y en modo CFB (elegido, por el grupo, entre los que incorpora el Provider SunJCE). La clave simétrica de cifrado se tomara del *keyStore* del servidor, donde previamente, despues de generada, debera haber sido almacenada.
- Generará un número de registro (*idRegistro*), que sera un número que ira creciendo de forma secuencial con cada nuevo registro y que se usara para identificar el documento en posteriores referencias a él.
- Generará un sello temporal, *selloTemporal,* con la fecha y la hora en que se registra.
- Firmará, usando la clave privada asociada al certificado $CertFirma_S$, el documento (sin cifrar) junto con el número de registro y el sello temporal. Mas exactamente, usando la clave privada de firma del servidor, computará la firma:
  - $SigRD = Sig_S(idRegistro, selloTemporal, idPropietario, documento, firmaDoc)$.
- Almacenará en un fichero el documento (cifrado o no), junto con su firma, número de registro, sello temporal y firma de registrador *SigRD*. (NOTA: nombrar el fichero de acuerdo con la nomenclatura: *idRegistro_idPropietario.sig[.cif]*).

Respuesta.

Si no hay ningun error en el proceso anterior, enviará al cliente la respuesta:

Registrar_documento.Response $(0, idRegistro, selloTemporal, idPropietario, SigRD, CertFirma_S)$.

donde $CertFirma_S$ es el certificado de la clave de firma del servidor.

En caso contrario enviará la respuesta:

- Registrar_documento.Response*(nERROR)*

donde *nERROR* será un número negativo que se usara para diferenciar la causa del error.

El cliente, en caso de respuesta correcta:

- Verificará el certificado $CertFirma_S$. Si es incorrecto presentará por pantalla el mensaje de error "CERTIFICADO DE REGISTRADOR INCORRECTO".
- Verificará la firma del registrador *SigRD*. en el mensaje. Para la verificación de esta firma se usará el *documento* y *firmaDoc* almacenados temporalmente por el usuario
  - Si esta firma es incorrecta presentara por pantalla el mensaje de error "FIRMA INCORRECTA DEL REGISTRADOR".

- Si es correcta:
  - Presentará en pantalla el mensaje "Documento correctamente registrado con el numero" añadiendole el numero de registro.
  - Computará y almacenara *h(documento)*, donde *h* es la funcion hash SHA-384, asociandolo al numero de registro *idRegistro* recibido.
  - Borrará de su equipo el documento enviado y la *firmaDoc*.

2. **LISTAR_DOCUMENTOS**. Con este servicio cada propietario podra visualizar la relacion de documentos públicos y sus propios documentos privados guardados por el servidor de registro.

   PETICION.

   LISTAR_DOCUMENTOS.REQUEST *(Tipo, $CertAuth_C$).*

   donde:

   - $CertAuth_C$ es el certificado de autenticación del cliente (que conlleva su identidad).
     - *Tipo*= "PRIV", para solicitar los privados, y *Tipo* = "PUB" para solicitar los publicos.

   RESPUESTA

   El servidor verificará que el certificado es correcto. Si no lo es, se devolverá un codigo de error de "CERTIFICADO INCORRECTO", y en caso positivo se enviara la respuesta:

   - LISTAR_DOCUMENTOS.RESPONSE *(ListaDocPublicos).*

   ó

   - LISTAR_DOCUMENTOS.RESPONSE *(ListaDocPrivados).*

   donde cada elemento de la lista *ListaDocPublicos y ListaDocPrivados*, constará de *idRegistro*, *id_Propietario, nombreDoc* y *selloTemporal.*

   En el cliente, se deberá presentar por pantalla el contenido de la lista enviada.

3. **RECUPERAR DOCUMENTO.** Mediante este servicio, un propietario podrá recuperar un documento de la base de datos del servidor.

   PETICION.

   RECUPERAR_DOCUMENTO.REQUEST *($CertAuth_C$, idRegistro).*

   donde:

   - $CertAuth_C$ es el certificado de autenticación del cliente (que conlleva la identidad del propietario).

   A la recepcion de la petición, el servidor:

   - Comprobara si existe el documento *idRegistro.*En caso contrario devolvera el error."DOCUMENTO NO EXISTENTE".
   - Accederá a la base de datos y comprobara si el documento es privado o no. Si es PRIVADO comprobará que quien lo solicita es el usuario legítimo (la identidad del usuario se obtendrá del certificado $CertAuth_C$). En caso contrario devolvera el error "ACCESO NO PERMITIDO".
   - Si el documento es privado, procederá con el descifrado del documento, tomando la clave de descifrado de su *keystore* y usando como algoritmo de descifrado el mismo que se uso para cifrarlo.

   RESPUESTA.

Si no hay ningun error en el proceso anterior, enviara al cliente la respuesta:

RECUPERAR_DOCUMENTO.RESPONSE
(0, tipoConfidencialidad, idRegistro, idPropietario, selloTemporal, $E_{PK_C}(K)$, $E_K(documento)$, SigRD, CertFirm
).

donde:

- $E_{PK_C}(K), E_K(documento)$es el cifrado PGP del documento (cifrado de capa de aplicación) realizado por el servidor. Se aplica aqui todo lo comentado en el cifrado similar descrito encima para $E_{PK_S}(K), E_K(documento)$, salvo que ahora se usa la clave publica del cliente tomada del certificado $CertAuth_C$.

En caso contrario enviará la respuesta:

RECUPERAR_DOCUMENTO.RESPONSE*(nERROR)*

donde *nERROR,* (distinto de 0), se usara para diferenciar la causa del error.

A la recepción del documento el propietario:

- Verificara el certificado $CertFirma_S$.
  - Si *tipoConfidencialidad* es PRIVADO, descifrará el documento usando la clave privada correspondiente.
  - Usando la clave publica del certificado $CertFirma_S$ comprobará si la firma SigRD es correcta.
    - Si la verificación es incorrecta sacará por pantalla el mensaje"FALLO DE FIRMA DEL REGISTRADOR".
    - Si la verificación es correcta, computara el hash SHA-512 *h(documento)* sobre el documento recibido y lo contrastará con el que guarda almacenado.
      - Si esta comprobación es correcta archivara el documento recibido y sacará el mensaje "DOCUMENTO RECUPERADO CORRECTAMENTE" junto con el número de registro y el sello temporal.
      - Si es incorrecta sacará el mensaje "DOCUMENTO ALTERADO POR EL REGISTRADOR".

1.2 Verificacion de no revocacion del certificado del servidor.

En este apartado se debera verificar mediante la tecnica OCSP, en el proceso de Handshake TLS, que el certificado del servidor no este revocado. Se implementaran las dos tecnicas OCSP:

1. La opcion de verificacion por el cliente, denominada usualmente *Client-driven OCSP*
2. La opcion de verificacion por el servidor, tambien llamada *OCSP Stapling.*

Para esto:

- Se consultara el apartado *Client-driven OCSP and OCSP Stapling en la* pagina 8-74 del manual *Java Platform Standard Edition, Security Developer's Guide, July 2020,* que describe los parametros del JSSE a configurar tanto del lado del cliente como del lado del servidor.
- Se instalara un servidor OCSP (OCSP-Responder) usando OpenSSL y configurándolo para tal efecto.

- Se debera probar tanto que el certificado este no revocado como que este revocado. Para revocar/desrevocar el certificado del servidor en el OCSP-Responder se usaran los comandos que el OpenSSL dispone a tal efecto.

En las dos figuras de debajo se ilustra cada uno de los dos esquemas de revocacion OCSP

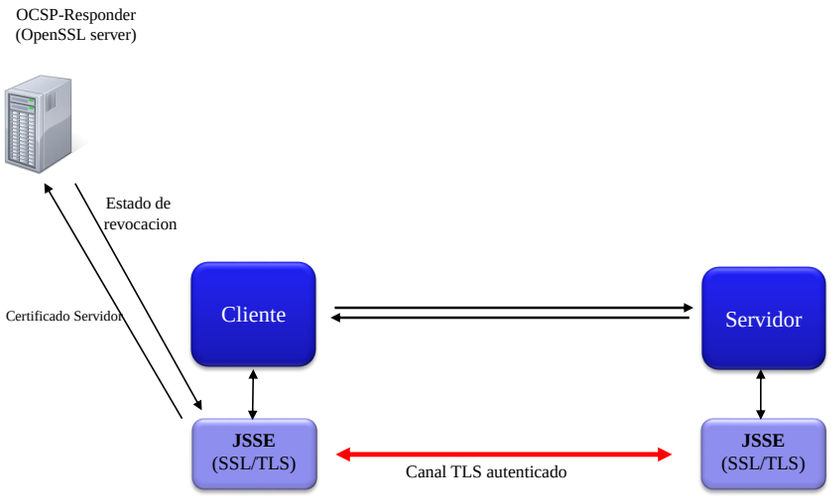

Figure 2: OCSP de Cliente

1.3 Implementación de un nuevo TrustManager de cliente.

Para la validación de las credenciales del servidor se implementará en el cliente un nuevo TrustManager que tendrá la misma funcionalidad que el TrustManager por defecto salvo en caso de fallo de validacion de las credenciales del servidor por, por ejemplo, no existir en el TrustStore de cliente alguno de los certificados necesarios para validar las credenciales del server. En ese caso el nuevo TrustManager deberá funcionar como sigue:

- Informará por pantalla al usuario de que las credenciales del servidor no son confiables, preguntandole si las acepta o no:
  - Si el usuario no acepta las credenciales se abortara la comunicacion.
  - Si el usuario acepta:
    - Se preguntara por pantalla la contraseña para acceder/abrir en escritura el *truststore* del cliente.
    - Se añadirán las credenciales del server (certificado o cadema de certificados) al *truststore* del cliente. Las nuevas credenciales añadidas se deberan tener en cuenta para las siguientes conexiones con el servidor.

Para este apartado se deberán consultar la secciones *"TrustManager Interface"*, *"TrustManagerFactory Class"*, *"X509TrustManager Interface"* y *"X509ExtendedTrustManager Class"*del manual del JSSE.

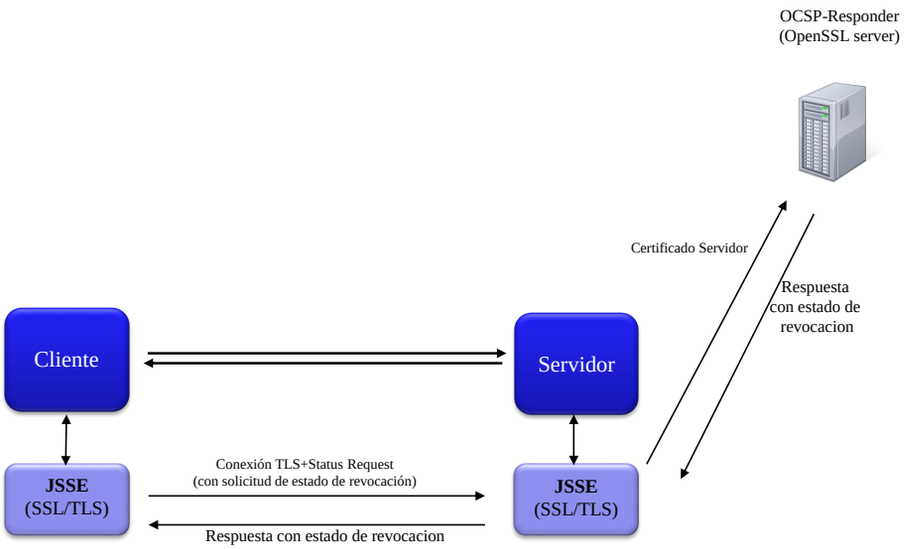

Figure 3: OCSP Stapling (de Servidor)

2 Consideraciones de diseño.

2.1 Tipo de claves públicas y certificados.

La aplicación deberá poder soportar claves público-privadas para los dos algoritmos RSA y DSA, tanto para el propietario como para el registrador. *Por ejemplo, un propietario puede tener una clave DSA-1024 mientras el registrador usar una clave RSA-2048.*

La clave simétrica de cifrado de documetos usada por el registrador se almacenará en el *keystore* del servidor.

2.2 Gestión de claves y certificados.

Las claves (tanto simétricas como asimétricas) y certificados (tanto de cliente como de servidor) se guardarán en los almacenes denominados *keystores* (almacenes de claves) y *truststores* (almacenes de confianza).

A efectos de facilitar la creación, almacenamiento y uso de los almacenes de claves, se recomienda utilizar una herramienta grafica, como por ejemplo *KeyExplorer* (disponible para windows, Mac y Linux) (ver enlace en Faitic). Para acceder desde el programa en Java, a los almacenes de claves se deberá usar el *API Java del KeyStore* (ver documentacion en FAITIC Api KeyStore)

Para obtener los certificados de clave pública del cliente y servidor se procedera como sigue:

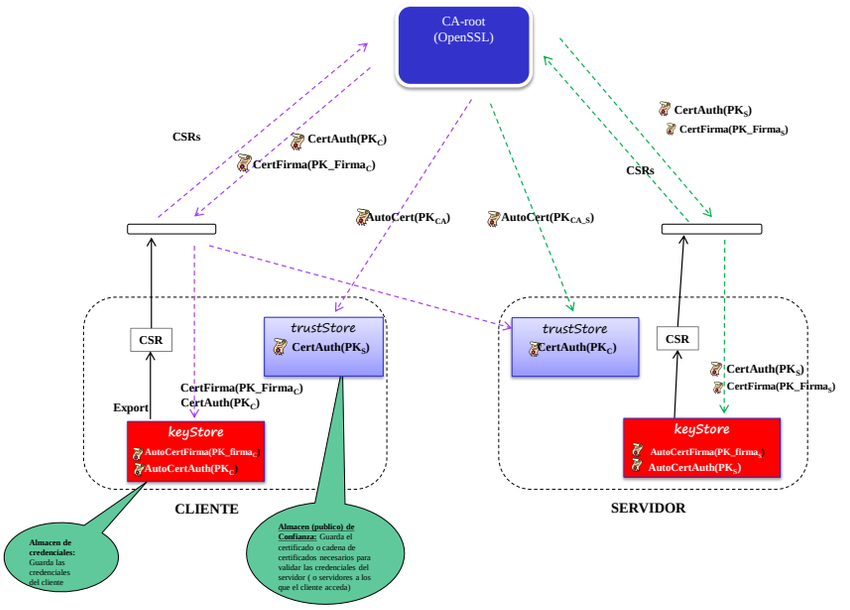

Figure 4: Obtención de certificados de cliente y servidor y clave publica CA root

- Se elige primero una autoridad de certificación que proporcione certificados ante peticiones de certificados en formato CSR (Certificate Signing Request). Clientes y servidores pueden usar la misma autoridad de certificación. se usara para ello la utilidad correspondiente del OpenSSL, documentada en las paginas 40a 47 dela tercera edicion de ***OpenSSL CookBook.***
- Accediendo al *keystore* (de cliente y de servidor) con el *KeyExplorer* (tambien puede realizarse por interfaz de comandos), se generan primero las parejas de claves de cliente y servidor. Luego, tambien desde el *KeyExplorer, se generan* los CSR para las claves públicas de cliente y servidor.[2]. [3]
- A partir de CSRs generados y usando la utilidad de creacion de certificados del OpenSSL (ver manual del ***OpenSSL CookBook.***), crear los certificados correspondientes. Para ello ha de crearse primero una autoridad raiz, **CA-root**, con su certificado correspondiente..
- Importar los certificados creados con el OpenSSL al *keystore* y *truststore* correspondiente (ver figura).
- Se importa el autocertificado con la clave pública de la CA-root al truststore *cacerts* tanto de cliente como del servidor. [4]

2.3 Suite SSL a usar.

La suite SSL a usar ha de ser configurable, *pero debera ser una suite que no incorpore confidencialidad.* Al comenzar la ejecucion de la aplicacion del lado del cliente, se presentaran en pantalla las suites

[2] Nótese que cuando se crea la pareja de claves publica-privada de cliente (o de servidor) desde el *KeyExplorer*, las claves públicas se guardan de forma *autocertificada*.

[3] Alternativamente, este paso tambien puede hacerse directamente con el OpenSSL.

[4] *NOTA IMPORTANTE.* Se recomienda crear un almacen *raiz* de confianza diferente al del sistema (*cacerts*), definido este por medio de la variable (system property) *javax.net.ssl.trustStore, y* cuyo valor por defecto es *cacerts*.

que el cliente tiene disponibles y se permitira al usuario seleccionar una, que es la que luego se usara durante el resto de la comunicacion.[5]

2.4 Formato de mensajes y ficheros.

El formato de envio y recepcion de los mensajes, asi como el formato de almacenamiento de ficheros por el servidor es libre. Es decir, cada grupo puede definirlos como estime mas adecuado.

2.5 Arranque de la aplicación cliente.

Se arrancara la aplicacion de cliente segun la siguiente linea de comandos:

- cliente *keyStoreFile truststoreFile*

En este comando:

- *keystroreFile* es el nombre del fichero cliente usado como almacen de claves y credenciales.
- *truststoreFile* es el nombre del fichero cliente usado como almacen de confianza.

La contraseña de acceso al keystore del cliente se preguntará o proporcionará por pantalla justo inmediatamente despues de introducir por pantalla los datos necesarios para llevar a cabo cualquiera de los servicios 1/2/3 descritos en el punto 1.

La contraseña *del truststore* del cliente no se necesita (salvo para la funcionalidad descrita en el punto 1.3), ya que los accesos al *truststore* son solo en lectura.

2.6 Arranque de la aplicación del servidor.

Se arrancara la aplicacion de cliente segun la siguiente linea de comandos:

- registrador *keyStoreFile contraseñaKeystore truststoreFile algoritmoCifrado*

En este comando:

- *keystroreFile* es el nombre del fichero del servidor usado como almacen de claves y credenciales.
- *truststoreFile* es el nombre del fichero del servidor usado como almacen de confianza.
- *contraseñaKeystore* es la contraseña de acceso al keystore del servidor.
- *algoritmoCifrado.* Cualquiera de los algoritmos de cifrado en bloque que soporte el provider SunJCE y que contemple claves de 128 bits.

2.7 Ficheros de pruebas y tests de evaluación.

Deberá probarse la practica usando ficheros binarios, idealmente imagenes, de más de 1 Mbyte. A la hora de evaluar la practica se someterá esta a los tests siguientes:

- Registrar 3 documentos a nombre de un cliente, desde un cliente en una máquina.
- Simultaneamente, registar 3 documentos de otro cliente, desde otro cliente en otra máquina.

[5] A este respecto, notese que en la negociacion SSL, el servidor, si la implementacion la incorpora, acepta la suite propuesta por el cliente.

- Recuperarlos en el mismo orden en que fueron registrados, verificando que la recuperación es correcta (contenido, ...).
- Verificar que los intentos de recuperación/acceso de un cliente a los documentos privados de otro cliente son denegados por el registrador.
- Comprobar que los documentos recuperados del servidor/registrador no han sido alterados.
- Testear la revocacion/no-revocacion OCSP del certificado del servidor.

3 Grupos y fecha de evaluación.

La práctica se puede realizar en grupos de hasta 3 alumnos.

La fecha tope para la entrega de la práctica será el dia 11 de Enero de 2022 a las 23:59 subida a la plataforma Faitic. Los alumnos que entregen la práctica antes del 15 de Diciembre podrán concertar una cita con el profesor para su evaluación. Para las entregas posteriores al 15 de diciembre y hasta el 11 de Enero (inclusive), la evaluación se realizará los dias siguientes, segun calendario que se publicara en su momento y a horas concertadas con cada grupo.

La evaluación de la práctica se realizará de forma remota y deberán estar presentes todos los integrantes del grupo que deseen ser evaluados.

4 Valoración.

La puntuación de la práctica se distribuirá como indica la tabla siguiente:

FUNCIONALIDAD	VALORACIÓN
Establecimiento del handshake TLS	0.6
Funcionalidad básica (punto 1.1)	1.2
Verificacion de no revocacion (punto 1.2)	0.6
Nuevo TrustManager de cliente (punto 1.3)	0.1
TOTAL	**2.5**

Table 1: Valoración de la práctica

5 Bibliografía.

Consultar documentación en Faitic en el apartado: *Documentos y enlaces/Practicas B.*