# SEGURIDAD 3º GETT – Práctica B – Curso 2018/2019.

Francisco J. Fernández Masaguer

10/9/2018

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 criptografica 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 (*keytool o KeyExplorer*).
- Aprender a manejar certificados usando una clase Java.
- 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 en el sistema tendrá su propio *keystore* y *truststore*. El *keystore* guarda las credenciales que cada usuario (clientes o servidor) necesita para poder autenticarse correctamente, mientras que *truststore* (almacen de confianza) guarda los certificados necesarios para poder verificar la autenticacion de los demas usuarios. Por ejemplo, cuando el servidor se autentica ante el cliente, se toman las credenciales del servidor 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 deberá poseer dos certificados, una para autenticación y otro distinto para firma, que se denotarán por *CertAuth* y *CertFirma*. El certificado de autenticacion sera el usado por la capa

[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*.

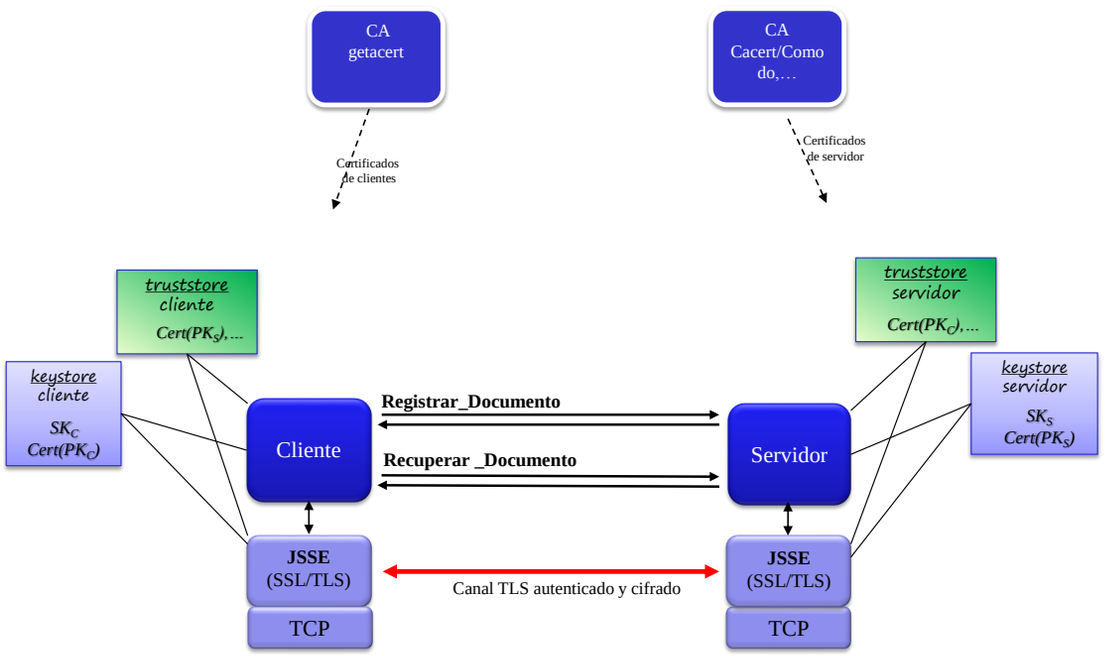


Figure 1: Componentes de la aplicacion

TLS/SSL, mientras que el certificado de firma sera el usado para la validacion, a nivel de aplicación, de las firmas que realice el usuario.

De forma general, la aplicacion constará (basicamente) de dos servicios (El canal SSL usado para ambos servicio 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. *Recuperar_Documento.*Mediante ese segundo servicio, el propietaro podrá, posteriormente, recuperar el documento firmado y almacenado por el registrador.

El detalle concreto de cada servicio es el siguiente:

1. **REGISTRAR_DOCUMENTO.**

   **PETICION.**

   REGISTRAR_DOCUMENTO.REQUEST *(nombreDoc, tipoConfidencialidad,* $E_{PK_S}(documento)$*, firmaDoc,* $CertFirma_C$*).*

   En este servicio:

   - *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.

- $E_{PK_S}(documento)$, es el cifrado de clave pública del documento realizado con la clave publica del servidor (cifrado de capa de aplicación). Esta clave publica se obtendra del Certificado de clave publica del servidor almacenado en el trustore del cliente. Este cifrado podrá realizarse con cualquiera de los algoritmos de cifrado de clave pública soportados por el JCA (RSA, ...). Si *tipoConfidencialidad*=PRIVADO sí se realizara este cifrado, pero si *tipoConfidencialidad*=PUBLICO no se realizará este cifrado, enviándose el documento sin cifrar.
- *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 la clave publica de firma del propietario. El certificado debera incluir la identidad del propietario (*idPropietario*), que se obtendrá a partir del certificado mediante el metodo *X509Certificate.getIssuerDN()*.

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 seguro SSL.

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".
- Verificará la firma del documento. Si es incorrecta devolvera el mensaje de error "FIRMA INCORRECTA".
- Si *tipoConfidencialidad* es PRIVADO, descifrará el documento usando la clave privada correspondiente.
- 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 el documento (sin cifrar) junto con el numero de registro y el sello temporal. Mas exactamente, usando la clave privada de firma del servidor, computará la firma:
  - $SigRD = Sig_R(idRegistro, selloTemporal, documento, firmaDoc)$.
- Si el documento viene con caracter confidencial (*tipoConfidencialidad*=PRIVADO), cifrara el documento. Este cifrado se realizará usando un algoritmo de cifrado simetrico en bloque (elegido, por el grupo, entre los que incorpora el Provider SunJCE) e introducido a la aplicación al arranque de ésta y que soporte claves de 128 bits minimo. La longitud de la clave será de 128 bits. El modo de operacion a usar en el algoritmo de cifrado en bloque será el CBC. La clave simétrica de cifrado se almacenara el el *keyStore* del servidor.
- 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 $(0, idRegistro, selloTemporal, SigRD, CertFirma_S)$.

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-512, 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 los 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 su identidad).

   A la recepcion de la petición, el servidor:

- Comprobara si existe el documento.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,* $tipoConfidencialidad, idRegistro, selloTemporal, E_{PK_C}(documento), SigRD, CertFirma_S$ *).*

donde:

- $E_{PK_C}(documento)$, es el cifrado de clave pública del documento realizado con la clave pública del cliente (cifrado de capa de aplicación). Esta clave publica se obtendra del Certificado de clave publica del cliente correspondiente al certificado de autenticación y que se deberá almacenar en el trustore del servidor. Este cifrado podrá realizarse con cualquiera de los algoritmos de cifrado de clave pública soportados por el JCA (RSA, ...).

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 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".

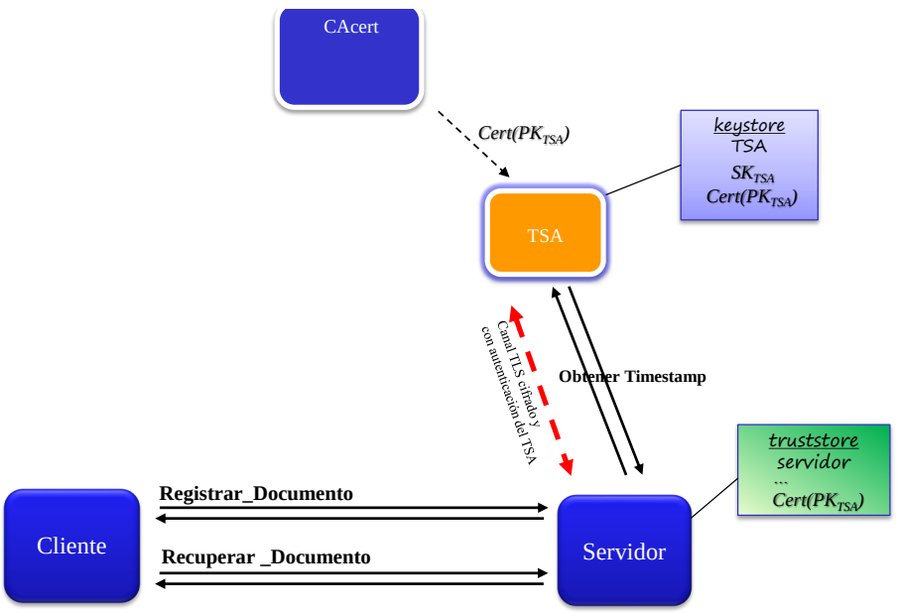


Figure 2: Incorporacion de una autoridad confiable de tiempos (TSA)

## 1.2 Incorporación de una autoridad de tiempos certificados.

En esta situacion, cuando el servidor reciba una peticion de registro de documento en lugar de generar el selloTemporal de forma local, lo obtendrá solicitandoselo a una autoridad de tiempos. La figura debajo ilustra los componentes y flujo que participan en el protocolo para la obtencion del tiempo certificado.

Las primitivas de peticion y respuesta del protocolo de obtencion del timestamp seran:

- OBTENER_TIMESTAMP.REQUEST *(h(documento))*.

  donde $h$ es la funcion hash *SHA-512*.

- OBTENER_TIMESTAMP.RESPONSE
  $selloTemporalTSA = selloTemporal, Sig_{TSA}(h(documento), selloTemporal)$.

  donde *selloTemporal* es la fecha y hora obtenida por el TSA en el momento de la petición. Las firmas por parte del TSA deberan realizarse con el algoritmo *SHA1withDSA* y claves DSA de 1024 bits.

  Al recibir la respuesta, el servidor deberá validará la firma del TSA. Si es incorrecta debera devolver al propietario un error de "Fallo de Firma de TimeStamp". Si es correcta el mensaje SigRD quedará como sigue:

  - $SigRD = Sig_R(idRegistro, selloTemporalTSA, documento, firmaDoc)$.

Si hubiese algun problema o error en la obtencion del tiempo certificado, se devolverá al cliente el error "ERROR DE OBTENCION DE TIEMPO CERTIFICADO" en la primitiva REGISTRAR_DOCUMENTO.RESPONSE (*nError*). Si no hay ningun error, guardará el tiempo certificado en la base de datos y enviará al cliente la respuesta:

- Registrar_documento.Response $(0, idRegistro, selloTemporalTSA, SigRD, CertFirma_S)$.

En cuanto al servicio de recuperar documento, la primitiva de respuesta quedará como sigue:

- Recuperar_documento.Response $(0, idRegistro, selloTemporalTSA, documento, SigRD, CertFirma_S)$.

En las dos primitivas anteriores, el cliente propietario al recibir la respuesta, deberá verificar tambien la firma del sello *selloTemporalTSA*.

*NOTA*. Nótese aquí que en el caso del TSA solo se trabaja con una pareja publico-privada, tanto para autenticacion como para firma. Por lo tanto solo se necesita un certificado, el denominado en la figura $Cert(PK_{TSA})$. El almacenamiento de este certificado en el cliente, puede realizarse bien en el truststore del cliente o bien en un almacen exprofeso añadido aparte, pero no debe usarse el almacen de credenciales.

### 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.

## 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 *keytoolIUI* (disponible para windows, Mac y Linux) *o KeyExplorer* (disponible solo para windows). Para acceder desde el programa en Java a los almacenes se deberá usar el *API Java del KeyStore* (ver documentacion en FAITIC Api KeyStrore)

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

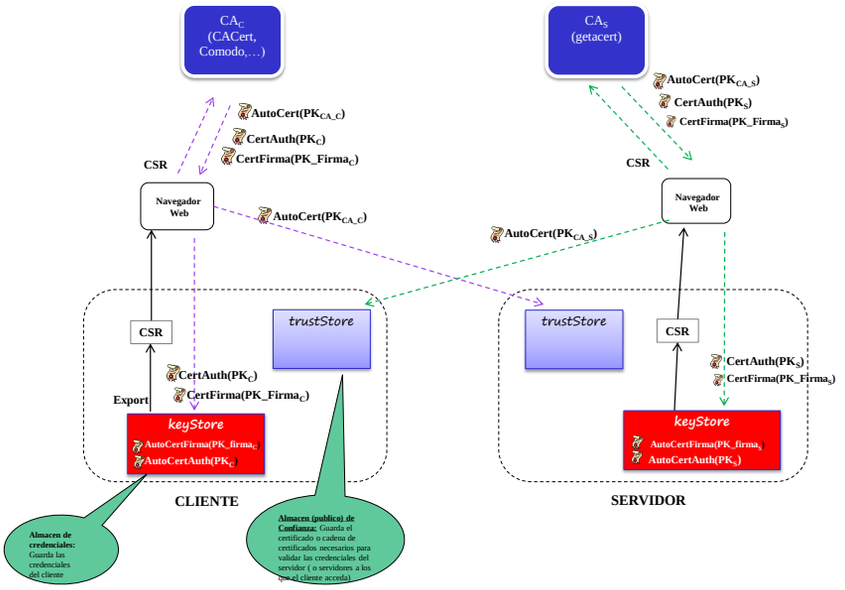


Figure 3: Obtención de certificados de cliente y servidor y clave publica CA root

- Se elige primero una autoridad de certificación en Internet que proporcione certificados gratis ante peticiones de certificados en formato CSR (Certificate Signing Request). Clientes y servidores deben usar autoridades de certificación distintas. Por ejemplo y de acuerdo con la figura 4, los clientes usarán la autoridad *GetAcert* (http://www.getacert.com), mientras que el servidor y el TSA usaran *CACert* (www.cacert.org), *Comodo* u otra cualquiera que el alumno seleccione.
- Se ilustra debajo, por ejemplo, la Web de getacert: http://www.getacert.com/.
- Accediendo al *keystore* (de cliente y de servidor) con el *keytool*, se generan primero las parejas de claves de cliente y servidor. Luego, tambien desde el *keytool, se generan* los CSR para las claves públicas de cliente y servidor.[2]
- Usando getacert, por ejemplo, en su página home se va a la opcion *"Submit certificate signing request(CSR) "*, para enviar al certificador la peticion CSR de certificado. Para incorporar el

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


Figure 4: Pagina web de getacert (www.getacert.com)

CSR generado por el *keytool* a la pagina web de *getacert*, simplemente se abre con un editor de texto el CSR generado y mediante una operacion de copy-paste, se copia el contenido completo del fichero al recuadro de la pagina Web correspondiente. Luego se envia la peticion y el servidor de *getacert* devuelve el certificado de la clave solicitada junto con la clave publica de *getacert* en forma *autocertificada*.

- Desde el *keytool*, se importan los certificados al *keystore* y *truststore* correspondiente (el certificado de cliente al *truststore* del servidor, y el del servidor al almacen de confianza del cliente).
- Se importa el autocertificado con la clave pública de la CA al almacen de confianza (truststore) *cacerts* tanto de cliente como del servidor. [3]

## 2.3 Suite SSL a usar.

La suite SSL a usar ha de ser configurable. Al comenzar la ejecucion de la aplicacion del lado del cliente, se presentaran en pantalla las suites 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.[4]

## 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.

[3] *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*.

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

## 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.
- 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.

# 3 Grupos y fecha de evaluación.

La práctica se puede realizar en grupos de hasta 4 alumnos.

La fecha tope para la entrega de la práctica será el dia 7 de Enero de 2018 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 8 de Enero, la evaluación se realizara entre los dias 9 y 15 de Enero a horas concertadas via Faitic.

En la evaluación de la práctica deberán estar presentes todos los integrantes del grupo.

# 4 Mejoras puntuables.

Requeriran al menos la realización de la funcionalidad básica para poder valorarse. Aparte de aquellas que pueda aportar el grupo, se proponen las dos siguientes:

## 4.1 Incorporar una jerarquía de certificación en el cliente de 2 niveles. (0.1 puntos).

## 4.2 Implementar una oficina de registro independiente del cliente y del servidor. (0.25 puntos).

Descrita abreviadamente, constaría de 3 agentes:

1. Cliente que se registra. El proceso seria similar al descrito en el punto 1.1 para el servicio de Registro solo que ahora el cliente lo hace ante una oficina de registro
2. Oficina (u oficinas) de registro. Tiene dos propositos:
   (a) Atender el registro del cliente.
   (b) Comunicar o enviar al servidor de almacenamiento el documento registrado por el cliente.
3. Servidor de almacenamiento. Almacena los documentos de los clientes. Los clientes listan y recuperan sus documentos accediendo a este servidor (en vez de al servidor de la oficina de registro) mediante dos servicios similares a los descritos en el punto 1.2 y 1.3.

# 5 Valoración.

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

| Funcionalidad | Valoración |
|---|---|
| Establecimiento del handshake TLS | 0.65 |
| Funcionalidad básica (punto 1.1) | 1.25 |
| Autoridad de tiempos certificados (punto 1.2) | 0.3 |
| Nuevo TrustManager de cliente (punto 1.3) | 0.3 |
| TOTAL | 2.5 |
| Mejoras propuestas por el grupo | hasta 0.25 |

Table 1: Valoración de la práctica

# 6 Bibliografía.

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