

Entrega 3 - Aplicaciones Web Seguras

Esta entrega se divide en tres partes diferenciadas pero relacionadas con la seguridad, que se detallan a continuación.

En la primera parte esta entrega, HTTPS, se transmitirá la información entre servidor y clientes a través de un canal seguro mediante el protocolo HTTPS. En esta parte es necesario utilizar el servidor Tomcat ya instalado en la máquina virtual.

En la segunda parte de esta entrega, Protección de contenidos, se protegerán los contenidos utilizando distintos métodos de encriptación. Por un lado, utilizaremos las librerías Java de encriptación de ficheros, que nos permiten proteger cualquier tipo de contenido digital (vídeos, documentos, etc.). Por otro, utilizaremos XML Encryption para encriptar los ficheros XML que podéis encontrar en el Racó. XML Encryption permite cifrar ficheros XML manteniendo el formato XML. Revisad los apuntes de teoría para conocer su funcionamiento.

En la tercera parte de esta entrega, Uso de JSON Web Token, se debe añadir una nueva operación en el servicio web REST de la Entrega 2 que devuelva un JSON Web Token (JWT). De nuevo, revisad los apuntes de teoría para saber cómo funciona JWT.

Primera parte – HTTPS

Se tiene que crear una aplicación web segura con J2EE que utilice el servidor Tomcat. No es necesario que sea muy compleja, con la funcionalidad de registro de usuarios propuesta en la Entrega 1 es suficiente.

Ejercicio 1. Configurar HTTPS en el servidor Tomcat

En [1] podéis encontrar el tutorial detallado de instalación de certificados para un servidor Tomcat. Como en esta entrega vamos a utilizar el **servidor Tomcat integrado en Netbeans 17**, deberemos realizar algunos pasos específicos que se detallan a continuación. Utilizaremos certificados autofirmados, así que el navegador nos mostrará una excepción de seguridad. En [2] podéis encontrar algunas indicaciones para evitar este problema, aunque no es necesario que lo hagáis en la práctica. En un caso real, dispondríais de un certificado correcto emitido por una Autoridad de Certificación reconocida y este problema no se produciría.

Nota: En la máquina virtual hay un problema con la configuración de dependencias de los proyectos que utilizan Tomcat y no compilan. En el fichero pom.xml de la aplicación web hay que cambiar las versiones de las siguientes dependencias: maven-compiler-plugin tiene que ser la versión 3.10.1 y maven-war-plugin tiene que ser la versión 3.3.2. Después hay que ejecutar el comando Clean and build, que descarga las dependencias y compila el proyecto. Ahora debería compilarse sin problema.

Certificados autofirmados del servidor

Para generar certificados autofirmados del servidor se usará la herramienta **keytool** distribuida con el kit de desarrollo de Java (Java SDK). El certificado generado se almacena en un repositorio de claves (keystore). Los certificados se almacenan en los repositorios asociándoles un alias.

En la versión 17 del JDK el funcionamiento de **keytool** ha cambiado respecto a versiones anteriores y es necesario crear las claves privadas, después una solicitud de firma de certificado (csr) y finalmente se crea el certificado. Una vez hecho esto, se puede guardar el certificado en el keystore para que el servidor lo pueda utilizar para establecer conexiones seguras con los clientes (normalmente navegadores). A continuación, se detallan los comandos para realizar estas operaciones desde un terminal de la máquina virtual de la asignatura.

Generar las claves pública y privada que se asociarán al certificado y el certificado. Este comando os pedirá un password para el keystore, que seguiremos utilizando después:

```
Prompt>/usr/lib/jvm/java-17-oracle/bin/keytool -genkeypair -keysize
2048 -keyalg RSA -alias isdcm -dname "CN=<Tu_nombre>, OU=FIB, O=UPC,
L=Barcelona, S=Barcelona, C=ES"

Enter keystore password:

Generating 2.048 bit RSA key pair and self-signed certificate
(SHA256withRSA) with a validity of 90 days

for: CN=<Tu_nombre>, OU=FIB, O=UPC, L=Barcelona, ST=Barcelona, C=ES
```

Después es necesario crear la csr y guardarla en un fichero. De nuevo, os pide el password (tiene que ser el mismo que hayáis puesto en el comando anterior).

```
Prompt>/usr/lib/jvm/java-17-oracle/bin/keytool -certreq -alias isdcm >
certreq-isdcm.csr

Enter keystore password:
```

Después, es necesario crear el certificado a partir de la csr. También hay que guardarlo en un fichero, tal y como se indica en las opciones del comando. De nuevo, os pide el password (tiene que ser el mismo que hayáis puesto en el comando anterior).

```
Prompt>/usr/lib/jvm/java-17-oracle/bin/keytool -gencert -alias isdcm -
infile certreq-isdcm.csr -outfile cert-isdcm.pem -rfc

Enter keystore password:
```

Si queréis consultar el certificado que se ha creado, podéis utilizar el siguiente comando, que os debería dar un resultado similar al que aparece aquí.

```
Prompt>/usr/lib/jvm/java-17-oracle/bin/keytool -printcert -alias isdcm
-file cert-isdcm.pem

Owner: CN=<Tu_nombre>, OU=FIB, O=UPC, L=Barcelona, ST=Barcelona, C=ES
Issuer: CN=<Tu_nombre>, OU=FIB, O=UPC, L=Barcelona, ST=Barcelona, C=ES
Serial number: 4c5a6285725134c7
Valid from: Thu Feb 22 12:33:05 CET 2024 until: Wed May 22 13:33:05
CEST 2024
Certificate fingerprints:
    SHA1:
2C:D2:23:08:7F:B6:13:29:D0:9A:D4:89:0C:AD:86:B0:BF:98:E7:7B
    SHA256:
BA:A2:7A:0C:A2:3A:77:63:58:09:95:57:81:A2:BB:90:C0:A8:54:4C:49:9C:C0:F
0:81:70:FB:45:7E:95:A1:06
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: 2B 56 7E B8 35 4E EA 91      2E 26 61 6E C4 91 DD DE
+V..5N...&an....
0010: DC 0A 6D 9E                      ..m.
]
]
```

Por último, es necesario añadir el certificado al keystore para poder utilizarlo desde Tomcat.

```
Prompt>/usr/lib/jvm/java-17-oracle/bin/keytool -importcert -file cert-
isdcm.pem -alias isdcm

Enter keystore password:

Certificate reply was installed in keystore
```

En ningún comando hemos indicado dónde queremos que nos guarde el almacén de claves (opción `-keystore path_al_fichero`), por lo que el fichero se crea en el directorio `/home/alumne` y se llama `.keystore`. Podéis copiarlo o crearlo directamente en otra ubicación.

Podéis comprobar que el fichero se ha creado bien con el comando que lista los contenidos del keystore:

```
Prompt>/usr/lib/jvm/java-17-oracle/bin/keytool -v -list -keystore
/home/alumne/.keystore
```

```
Enter keystore password:

Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: isdcm
Creation date: 22 feb 2024
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=<Tu_nombre>, OU=FIB, O=UPC, L=Barcelona, ST=Barcelona, C=ES
Issuer: CN=<Tu_nombre>, OU=FIB, O=UPC, L=Barcelona, ST=Barcelona, C=ES
Serial number: 4c5a6285725134c7
Valid from: Thu Feb 22 12:33:05 CET 2024 until: Wed May 22 13:33:05
CEST 2024
Certificate fingerprints:
    SHA1:
2C:D2:23:08:7F:B6:13:29:D0:9A:D4:89:0C:AD:86:B0:BF:98:E7:7B
    SHA256:
BA:A2:7A:0C:A2:3A:77:63:58:09:95:57:81:A2:BB:90:C0:A8:54:4C:49:9C:C0:F
0:81:70:FB:45:7E:95:A1:06
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: 2B 56 7E B8 35 4E EA 91      2E 26 61 6E C4 91 DD DE
+V..5N...&an....
0010: DC 0A 6D 9E                      ..m.
]
]
```

Nota importante: En la máquina virtual hay otra herramienta **keytool** en `/usr/bin/keytool`. Si no ponéis el path completo, ésta es la herramienta que se llama por defecto y da errores de formato de keystore a la hora de poner en marcha el servidor Tomcat.

El siguiente paso es editar el fichero `server.xml` dentro del Tomcat instalado en Netbeans. Hay que ir a la pestaña Services, seleccionar Servers y darle al botón derecho del ratón sobre Apache Tomcat para que salga el menú, tal y como se ve en la Figura 1. Seleccionad la opción `Edit server.xml`.

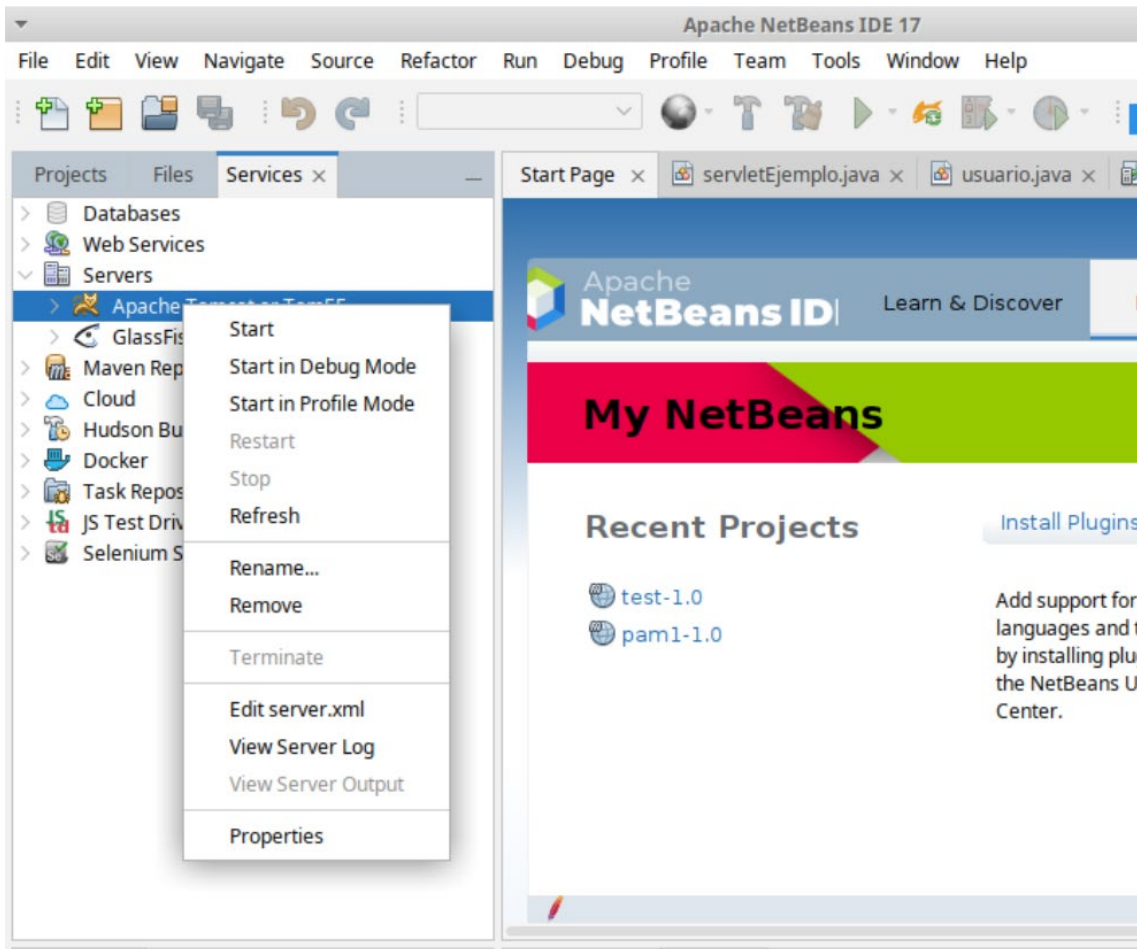


Figura 1. Opción Edit server.xml

Una vez abierto, debéis añadir el conector seguro (puerto 8443) dentro del fichero server.xml. Para hacerlo hay que copiar el siguiente conector en el fichero (el password es el de vuestro keystore). Podéis ver un ejemplo dentro del server.xml en la Figura 2:

```
<!-- Define an SSL Coyote HTTP/1.1 Connector on port 8443 -->
<Connector
  protocol="org.apache.coyote.http11.Http11NioProtocol"
  port="8443"
  maxThreads="150"
  SSLEnabled="true">
  <SSLHostConfig>
    <Certificate
      certificateKeystoreFile="${user.home}/.keystore"
      certificateKeystorePassword="<Tu_password>"
      type="RSA"
    />
  </SSLHostConfig>
</Connector>
```

Después, ya podéis poner en marcha Tomcat y comprobar que la conexión segura se ha activado. Esto se puede ver en la Figura 3. En la Figura 4 hemos ido a la página web (Con “Configuración avanzada” en el navegador), que se muestra correctamente, aunque el navegador nos indica que hay un error de certificado.

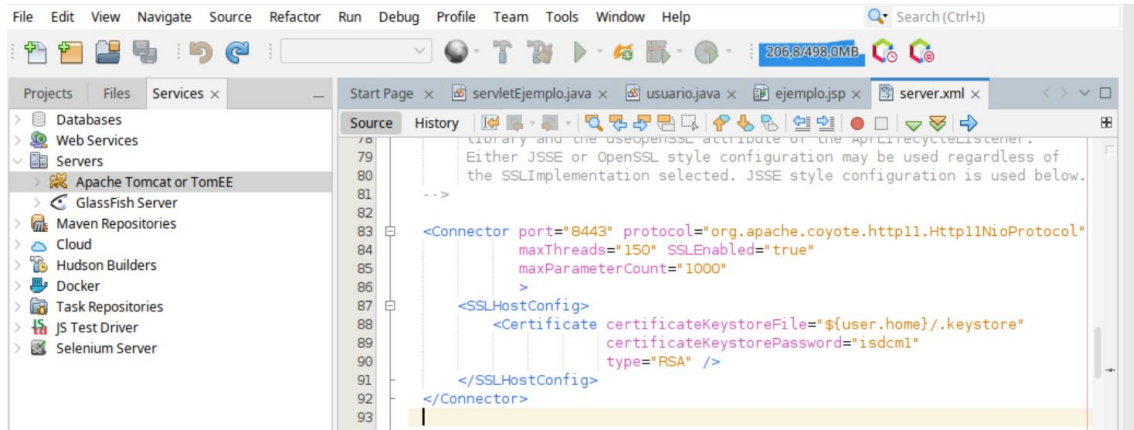


Figura 2. Modificar Connector 8443

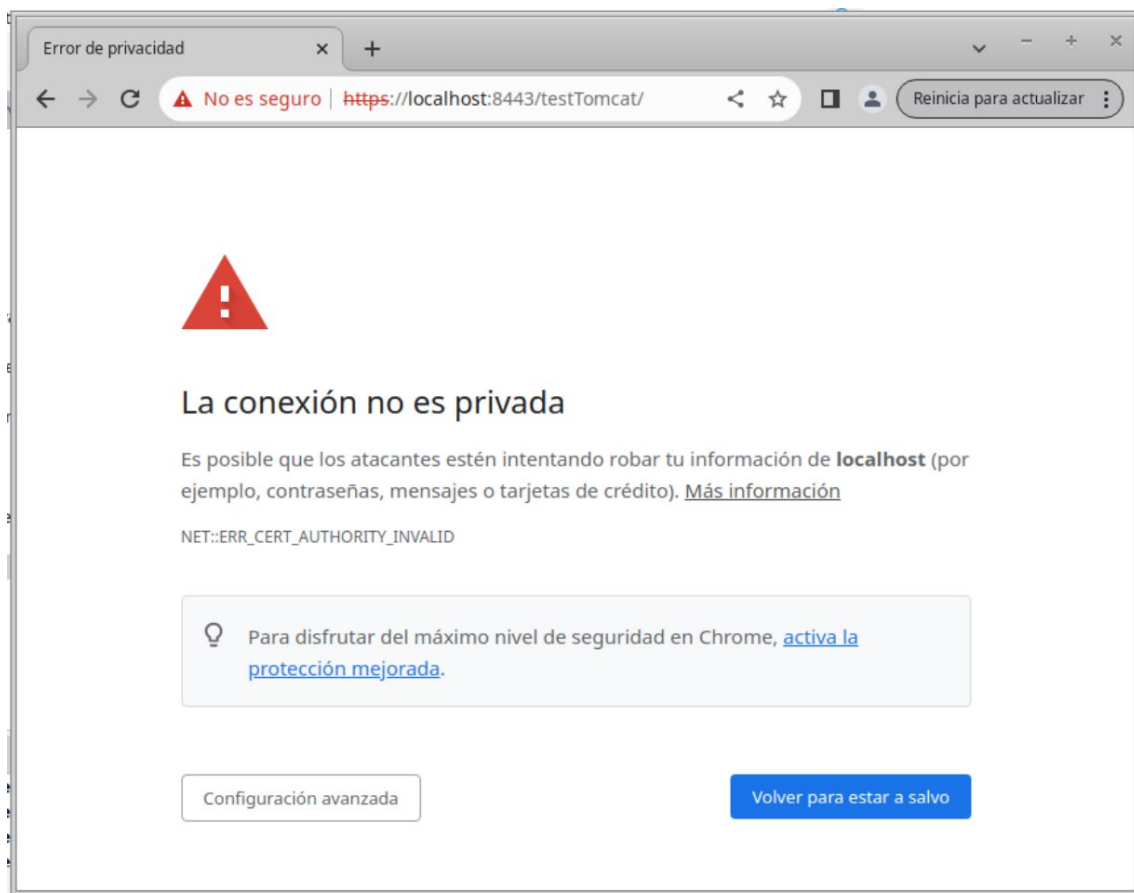


Figura 3. Puerto 8443 activado, excepción de seguridad

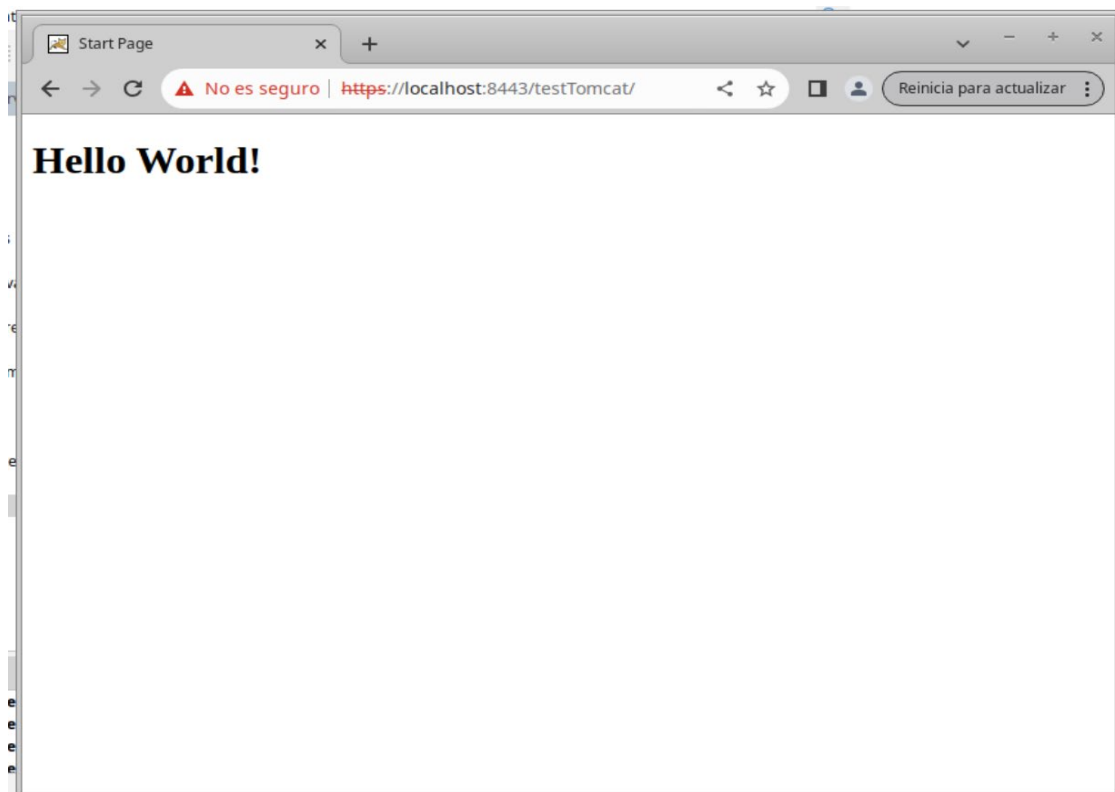


Figura 4. Página web segura, aunque con error de certificado

Segunda parte – Protección de contenidos

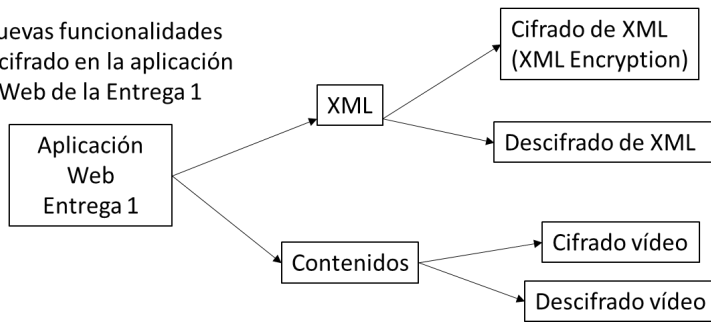
En esta segunda parte vamos a trabajar la protección de contenidos digitales. Tenéis que implementar dos tipos de protección, cifrado / descifrado de ficheros multimedia y cifrado / descifrado de ficheros de metadatos definidos en formato XML.

Para proteger los ficheros multimedia y ficheros XML, se pide añadir unas nuevas funcionalidades en la aplicación de la Entrega 1, tal y como se muestra en la Figura 5. En este caso, trabajaremos sobre la versión original sobre el servidor Glassfish.

En el caso de los ficheros multimedia, tendréis que cifrar los contenidos digitales (vídeos). En caso de haber utilizado vídeos on-line en las entregas anteriores, podéis cifrar cualquier tipo de fichero (imágenes, audio, etc.).

Para proteger los ficheros XML, se pide encriptar y desencriptar un fichero XML, tal y como se muestra también en la Figura 5. En este caso, debéis proteger los objetos digitales (Digital Items), que son ficheros en formato XML que contienen tanto el contenido (embebido o referenciado), como los metadatos (título, autor, fecha de creación, etc.). Podéis encontrar un fichero XML de ejemplo en el Racó (did1Film1.xml).

Nuevas funcionalidades de cifrado en la aplicación Web de la Entrega 1



Nota: XML encryption puede aplicar a todo el documento, un elemento o el contenido de un elemento (texto)

Nota: En caso de no tener ficheros de vídeo en el disco local, se puede cifrar cualquier otro tipo de contenido digital (imagen, documentos Word, pdf) ya que la manera de cifrar es la misma.

Figura 5. Integración aspectos de seguridad en la Entrega 1

Ejercicio 2.1 Cifrado de contenidos

En esta tarea tendréis que implementar un método que permita encriptar los contenidos digitales que gestiona vuestra aplicación web, es decir, cualquiera de los vídeos que tenéis en el servidor. En caso de haber utilizado vídeos on-line, podéis cifrar cualquier tipo de contenido digital almacenado en el disco duro de vuestro ordenador.

En lenguaje Java tenéis a vuestra disposición el paquete `javax.crypto` que contiene clases e interfaces para operaciones criptográficas, como por ejemplo la clase `Cipher`.

Ejercicio 2.2 Protección de los objetos digitales

Java proporciona diferentes librerías que permiten encriptar documentos XML. La más utilizada es la Apache XML Security for Java library [3], que soporta las recomendaciones del W3C XML-Signature Syntax and Processing [4] y XML Encryption Syntax and Processing [5].

La librería proporciona ejemplos para encriptar/desencriptar **documentos XML** en `\samples\org\apache\xml\security\samples\encryption`. En la Figura 6 tenemos un ejemplo de código para encriptar XML y en la Figura 7 tenemos un ejemplo de código para desencriptar.

```

XMLCipher keyCipher = XMLCipher.getInstance(XMLCipher.AES_128);

SecretKey symmetricKey = new SecretKeySpec(key.getBytes(), "AES");
xmlCipher.init(XMLCipher.ENCRYPT_MODE, symmetricKey);
boolean encryptContentsOnly = false;

xmlCipher.doFinal(node.getOwnerDocument(), (Element)node, encryptContentsOnly);
  
```

Figura 6. Código para encriptar XML

```

XMLCipher keyCipher = XMLCipher.getInstance(XMLCipher.AES_128);

SecretKey symmetricKey = new SecretKeySpec(key.getBytes(), "AES");
xmlCipher.init(XMLCipher.DECRYPT_MODE, symmetricKey);

Document document = xmlCipher.doFinal(node.getOwnerDocument(), (Element)node);
  
```

Figura 7. Código para desencriptar XML

A continuación, tenéis un objeto digital (Figura 8) que contiene tanto los metadatos de una película, como una referencia a la película (film1mp4v.mp4):

```
<DIDL xmlns="urn:mpeg:mpeg21:2002:02-DIDL-NS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg21:2002:02-DIDL-NS didl.xsd">
  <Item>
    <Component>
      <Descriptor>
        <Statement mimeType="text/xml">
          <metadata>
            <id>1</id>
            <titulo>No time to die</titulo>
            <autor>Cary Joji Fukunaga</autor>
            <anyo>2021</anyo>
            <duracion>163</duracion>
            <reproducciones>1150</reproducciones>
          </metadata>
        </Statement>
      </Descriptor>
      <Resource ref="film1mp4v.mp4" mimeType="video/mp4"/>
    </Component>
  </Item>
</DIDL>
```

Figura 8. Objeto digital - XML

En el Racó podéis encontrar el fichero XML del objeto a encriptar (diFilm1.xml), y los ficheros XSD que utiliza (didl.xsd, didlmodel.xsd).

En esta tarea tenéis que implementar dos métodos: uno para encriptar el fichero XML o cualquiera de sus elementos o contenido de un elemento y otro para desencriptar un fichero XML previamente encriptado.

Para que os funcione correctamente, tenéis que añadir la dependencia de la Figura 9 al fichero pom.xml de vuestro proyecto Netbeans:

```
<dependency>
  <groupId>org.apache.santuario</groupId>
  <artifactId>xmlsec</artifactId>
  <version>1.5.8</version>
  <type>jar</type>
</dependency>
```

Figura 9. Dependencia librería xmlsec versión 1.5.8

Tercera parte – Uso de JSON Web Token

En esta tercera parte vamos a implementar la creación de un JSON Web Token [6], que se devolverá como respuesta a una nueva operación de login que debéis implementar en el servicio web REST de la Entrega 2.

La operación debe tener la cabecera tal y como se muestra en la Figura 10:

```
/**
 * POST method to login in the application
 * @param username
 * @param password
 * @return
 */
@Path("login")
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Response Login(@FormParam("username") String username,
                      @FormParam("password") String password)
```

Figura 10. Cabecera operación login en el servicio web REST

Es necesario utilizar las librerías Java que se muestra en la Figura 11:

```
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import java.util.Date;
import javax.json.Json;
import javax.json.JsonObject;
```

Figura 11. Librerías necesarias para generar JWT's en Java

Que se corresponden con la dependencia que se muestra en la Figura 12 que se debe añadir al fichero pom.xml:

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.7.0</version>
</dependency>
```

Figura 12. Dependencia para la librería jjwt versión 0.7.0

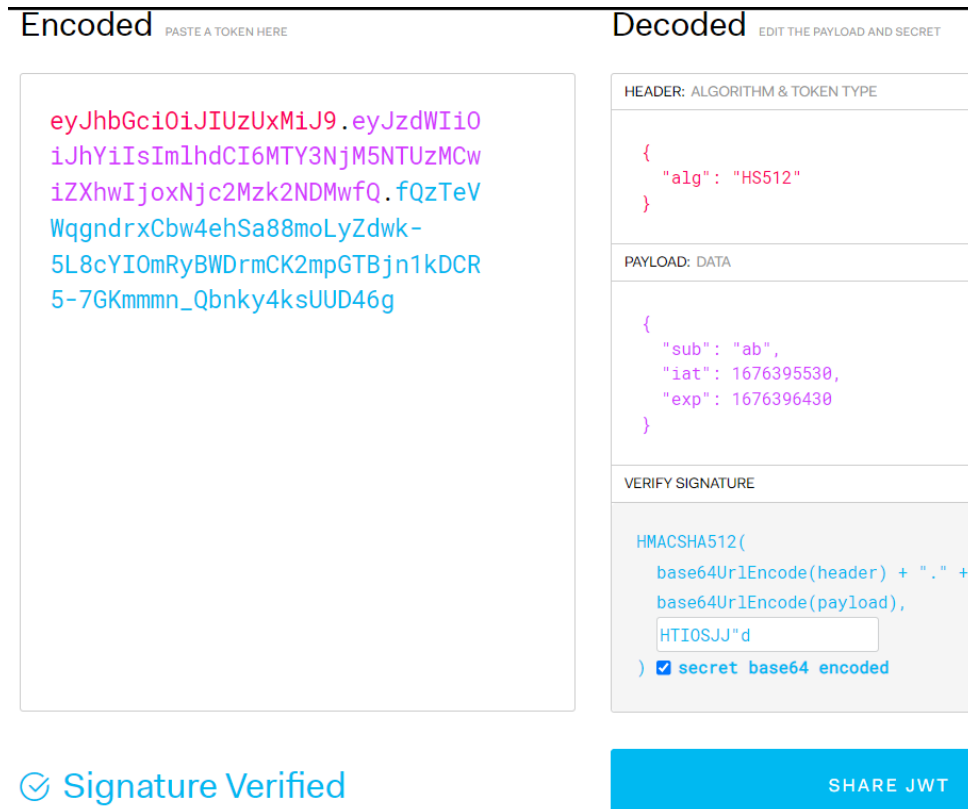
Podéis llamar a esta operación desde un formulario html, una página jsp o un servlet. En cualquier caso, lo que se persigue es obtener el JWT generado por el servidor y comprobar que es correcto en la web <https://jwt.io/>. En esta web se puede verificar la firma de un JWT codificado en base 64 y ver su contenido.

Además, se puede proteger contenido expresado en formato JSON, como, por ejemplo, un JWT, utilizando la librería Java jose4j [7], que es una librería con licencia Apache 2 que implementa JWS, JWE, JWA y JWK del IETF JOSE Working Group [8]. Podéis aplicar cualquier operación al JWT que se ha generado anteriormente para comprobar su funcionamiento. Se debe explicar en detalle en el informe el uso que hagáis de esta librería.

Como ejemplo, el JWT (es el valor que empieza por ey y acaba en Zg) de la Figura 13 se puede ver en la web <https://jwt.io/> y muestra los valores que se pueden ver en la Figura 14:

```
{"JWT": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhYiIsImIhdCI6MTY3Nm5NTUzMwZlZlXhwIjoxNjc2Mzk2NDMwZlZlV1VHY_boOWxvCNsy5Ik-TkMLYWgFIQ1FtDRX1WsBUuksa_qaWvhlz05qD85yVElBXdr0gH8qCJdyxiCFkXEDZg"}
```

Figura 13. Ejemplo de JWT generado



The image shows the jwt.io interface with a JWT token pasted into the 'Encoded' field. The token is decoded, showing the following structure:

- Encoded:** eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhYiIsImIhdCI6MTY3Nm5NTUzMwZlZlXhwIjoxNjc2Mzk2NDMwZlZlV1VHY_boOWxvCNsy5Ik-TkMLYWgFIQ1FtDRX1WsBUuksa_qaWvhlz05qD85yVElBXdr0gH8qCJdyxiCFkXEDZg
- Decoded:**
 - HEADER: ALGORITHM & TOKEN TYPE:** { "alg": "HS512" }
 - PAYLOAD: DATA:** { "sub": "ab", "iat": 1676395530, "exp": 1676396430 }
 - VERIFY SIGNATURE:** HMACSHA512(base64UrlEncode(header) + "." + base64UrlEncode(payload), HTIOSJJ"d) ☒ secret base64 encoded

At the bottom, it shows a green checkmark and the text "Signature Verified" and a blue button labeled "SHARE JWT".

Figura 14. Visualización del JWT en jwt.io

Forma de entrega

Debéis entregar un informe que muestre que habéis configurado correctamente Tomcat con HTTPS. Además, debéis entregar el código de las distintas funcionalidades de cifrado. En el informe, indicad si habéis realizado alguna modificación al esquema de aplicaciones propuesto en la Figura 5. Utilizad la plantilla de informe publicada en el Racó.

Referencias

- [1] Tutorial tomcat y SSL, <https://tomcat.apache.org/tomcat-9.0-doc/ssl-howto.html>
- [2] Cómo generar certificados en localhost que no dé advertencias de seguridad en el navegador, <https://www.jasoft.org/Blog/post/como-generar-certificados-https-para-desarrollo-local-que-no-produzcan-errores>
- [3] Apache XML Security for Java library, <http://santuario.apache.org/javaindex.html>
- [4] W3C XML Signature Syntax and Processing, <http://www.w3.org/TR/xmlsig-core/>
- [5] W3C XML Encryption Syntax and Processing, <http://www.w3.org/TR/xmlenc-core/>
- [6] IETF RFC 7519 JSON Web Token, <https://www.rfc-editor.org/rfc/rfc7519>
- [7] Jose4J library, https://bitbucket.org/b_c/jose4j/src/master/
- [8] IETF JOSE Working Group, <https://wiki.ietf.org/group/jose>