Guía Implementación de referencia
Firma de Transacciones
Receta Digital Nacional

Salud.uy

Versión 1.0 / abril 2021

Equipo Receta Digital Nacional

Ministerio de Salud Pública
Ministerio de Economía y Finanzas
agesic
Uruguay Presidencia

Control de Cambios

Fecha	Versión	Responsables	Cambios
08/04/2021	1.0	Equipo Receta Digital Nacional	• Versión inicial del documento

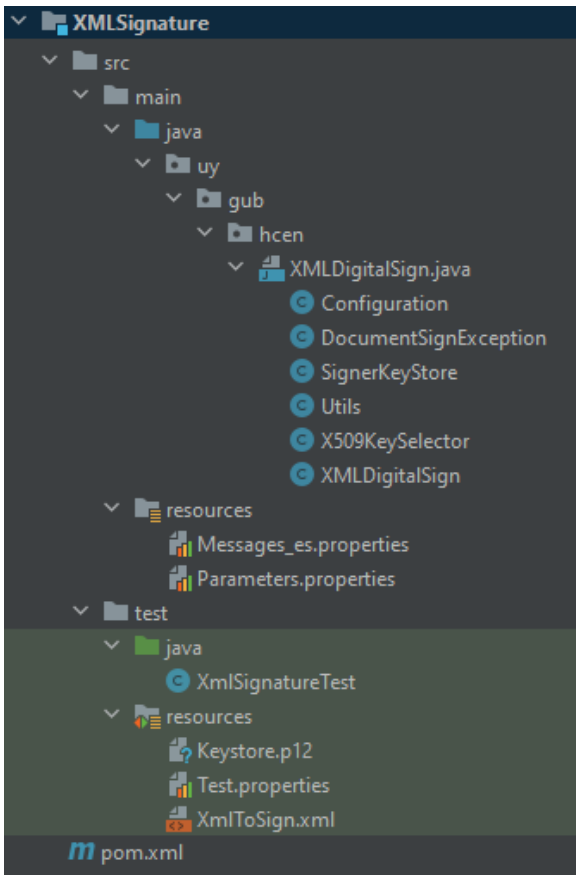
ÍNDICE

1. Introducción

El presente documento contiene una implementación de referencia de firma de XML para recetas digitales a abril de 2021. Para el desarrollo de este documento se ha tenido en cuenta la especificación del documento "Especificación de firma para receta digital".

La implementación aquí detallada está escrita en Java 8 y fue probada con el SDK 1.8.0. Se comentarán las partes relevantes para mostrar a grandes rasgos la solución y su cumplimiento con la especificación.

Las explicaciones de código fuente referencian al proyecto "XMLSignature" que sigue la estructura de la siguiente imagen:



Se aclara que esta implementación es simplemente una referencia. Cada firmante puede implementar su propia lógica de firma, siempre siguiendo la especificación.

2. Implementación

2.1. Clase Principal

El archivo XMLDigitalSign.java es autocontenido. En él están todas las clases necesarias para soportar los dos métodos principales:

- XMLDigitalSign.signDocument: Firma un XML con los parámetros provistos.
- XMLDigitalSign.verifyDigitalSignature: Verifica si una firma es válida, obteniéndola del documento firmado provisto. Este método se incluye en la implementación únicamente para poder verificar que la firma se realizó correctamente, a modo de prueba para el desarrollador.

2.1.1. signDocument

A continuación, se muestra la firma del método y se describen sus parámetros:

```
public static byte[] signDocument(byte[] document, String keystorePath,
    String keystorePassword, String keystoreAlias) throws DocumentSignException
```

- document: Arreglo de bytes con el documento XML a firmar.
- keystorePath: Ruta en el sistema de archivos de la keystore que contiene la clave pública y privada para realizar la firma.
- keystorePassword: Contraseña de la keystore.
- keystoreAlias: Alias de la keystore.

El método puede retornar una excepción de tipo DocumentSignException y el retorno es un arreglo de bytes con el documento XML firmado.

En primer lugar, se inicializa internamente la keystore con la información provista de ruta y contraseña:

```
SignerKeyStore signerKeyStore = new SignerKeyStore(keystorePath, keystorePassword);
```

La clase SignerKeyStore contiene la lógica de lectura del keystore para leer la clave privada y el certificado almacenado. Soporta los formatos PKCS12 y JKS.

Luego se obtiene el nodo raíz del XML a firmar y se crea una XMLSignatureFactory:

```
Document nonSignedDocument = Utils.byteArrayToDocument(document);
Node sigParent = nonSignedDocument.getDocumentElement();
XMLSignatureFactory sigFactory = XMLSignatureFactory.getInstance();
```

Sobre esta XMLSignatureFactory se definen los parámetros de firma (algoritmos y disposición

"enveloped") y se crea con ellos un objeto SignedInfo:

```
String referenceURI = ""; // Empty string means referencing whole document
String digestMethod = DigestMethod.SHA256; // This is http://www.w3.org/2001/04/xmlenc#sha256
String signatureAlgorithm = "http://www.w3.org/2001/04/xmldsig-more#rsa-sha256";
String canonicalizationMethod = CanonicalizationMethod.INCLUSIVE; // This is http://www.w3.org/TR/2001/REC-xml-c14n-20010315

List<Transform> transforms = new ArrayList<>();
transforms.add(sigFactory.newTransform(Transform.ENVELOPED, (TransformParameterSpec)null));

Reference ref = sigFactory.newReference(referenceURI, sigFactory.newDigestMethod(digestMethod, null), transforms, null, null);

SignedInfo signedInfo = sigFactory.newSignedInfo(
    sigFactory.newCanonicalizationMethod(canonicalizationMethod, (C14NMethodParameterSpec)null),
    sigFactory.newSignatureMethod(signatureAlgorithm, null),
    Collections.singletonList(ref));
```

Con la clase auxiliar SignerKeyStore se obtiene la clave privada de la keystore:

```
PrivateKey privateKey = signerKeyStore.getSignerPrivateKey(keystoreAlias);
```

Se procede a crear el elemento KeyInfo que en el XML resultante tendrá la firma del documento junto a la información del firmante (certificado utilizado), donde nuevamente con la ayuda de la clase auxiliar SignerKeyStore se obtiene esta vez el certificado:

```
DOMSignContext dsc = new DOMSignContext(privateKey, sigParent);

// Create the KeyInfo containing the X509Data.
KeyInfoFactory kif = sigFactory.getKeyInfoFactory();
List x509Content = new ArrayList();
x509Content.add(signerKeyStore.getSignerCertificate(keystoreAlias));
X509Data xd = kif.newX509Data(x509Content);
KeyInfo ki = kif.newKeyInfo(Collections.singletonList(xd));
```

Finalmente, se firma el documento con los objetos SignedInfo y KeyInfo, retornando un XML que se transforma a arreglo de bytes:

```
// Create the XMLSignature
XMLSignature signature = sigFactory.newXMLSignature(signedInfo, ki);

// Generate and sign the ENVELOPED SIGNATURE
```

```
signature.sign(dsc);

return Utils.documentToByteArray(nonSignedDocument);
```

2.1.2. verifyDigitalSignature

A continuación, se muestra la firma del método y se describen sus parámetros:

```
public static void verifyDigitalSignature(byte[] document) throws DocumentSignException
```

- document: Arreglo de bytes con el documento XML firmado.

El método no tiene retorno (void). Si no arroja una excepción, la firma se considera válida. En caso contrario, la excepción será de tipo DocumentSignException.

El primer paso de la lógica es crear una XMLSignatureFactory y obtener el nodo Signature del XML firmado:

```
XMLSignatureFactory factory = XMLSignatureFactory.getInstance();

Document signedDocument = Utils.byteArrayToDocument(document);
NodeList signatureNodes = signedDocument.getElementsByTagNameNS(XMLSignature.XMLNS, "Signat
ure");
if (signatureNodes.getLength() == 0) {
    throw new DocumentSignException(Configuration.getInstance().getMessage("NO_SIGNATURE_EL
EMENT_ERROR"));
}
Node signatureNode = signatureNodes.item(signatureNodes.getLength() - 1);
```

Finalmente se crea un contexto de validación con el nodo obtenido, se extrae la firma y se valida:

```
DOMValidateContext valContext = new DOMValidateContext(new X509KeySelector(), signatureNode
);
XMLSignature extractedSignature = factory.unmarshalXMLSignature(valContext);

boolean validationResult = extractedSignature.validate(valContext);
if (!validationResult) {
    throw new DocumentSignException(Configuration.getInstance().getMessage("CORRUPT_DOCUMEN
T"));
}
```

3. Clase de prueba

En la clase XmlSignatureTest se encuentra un método "main" con la ejecución de una firma y su validación. En primer lugar, se cargan propiedades por configuración, un keystore de prueba y el XML a firmar:

```
Properties testProperties = new Properties();
testProperties.load(XmlSignatureTest.class.getResourceAsStream("/Test.properties"));

String keystorePath
= new File(XmlSignatureTest.class.getResource("/Keystore.p12").toURI()).getAbsolutePath();
String keystorePassword = testProperties.getProperty("KEYSTORE_PASSWORD");
String keystoreAlias = testProperties.getProperty("KEYSTORE_ALIAS");
byte[] document = readXmlDocument(XmlSignatureTest.class.getResourceAsStream("/XmlToSign.xml"));
```

Luego se procede a firmar e imprimir el documento firmado:

```
byte[] signedDocument = XMLDigitalSign.signDocument(document, keystorePath,
keystorePassword, keystoreAlias);
System.out.println("Document was signed.");
System.out.println(new String(signedDocument, StandardCharsets.UTF_8));
```

Finalmente, el documento firmado es verificado:

```
XMLDigitalSign.verifyDigitalSignature(signedDocument);
System.out.println("Signature is valid, no errors were thrown.");
```