DOCUMENTOS TÉCNICOS
DIVISIÓN INFORMÁTICA
DESARROLLO

Guía XMLEncription Java & .Net

Factura Electrónica

CÓDIGO: - T-5.020.00.001-000010

Índice

1.Objetivo.

El presente documento pretende dar ciertas guías de desarrollo en el uso de XMLEncryption en el contexto de Factura Electrónica tanto para Java como para .Net.

2.Alcance.

La implementación no pretende ser la única ni la mejor solución, sino ser un marco de referencia para un nuevo desarrollo.

Contiene determinadas clases y/o métodos que pueden ser reutilizados, sin embargo muchos detalles que deben considerarse en un sistema de producción han sido obviados (por ejemplo el manejo de errores) para mantener el foco en lograr una encriptación/desencriptación exitosa de datos.

3.Herramientas.

La solución tiene como premisa utilizar las herramientas básicas de cada plataforma.

3.1.Herramientas utilizadas en la implementación Java

- ***J*dk 1.7.051** – Si bien pueden existir librerías que faciliten algunas de las tareas abordadas, la premisa es presentar una solución con las herramientas provistas por el JDK.
- ***Eclipse Juno*** – IDE de desarrollo utilizado para la implementación del ejemplo.

3.2.Herramientas utilizadas en la implementación .Net

- **Visual Studio Express 2013 for Desktop** – IDE de desarrollo que contiene todo lo necesario para la implementación del caso de ejemplo.
- **C#, .net Framework 4.5** - Lenguaje y versión de Framework utilizado en la implementación del ejemplo (ambos disponibles al instalar el Visual Studio Express for Desktop).

3.3.Otras herramientas

- **OpenSSL** – Paquete de herramientas de administración y bibliotecas relacionadas con la criptografía, utilizada para la generación de claves.

4.Pasos previos

Tanto para la implementación Java como .Net, se necesita contar con un keystore de tipo #PKCS12. Este almacén contiene la clave privada y pública protegidas mediante una clave simétrica.

Los almacenes pueden ser administrados con cualquier herramienta como ser keyTool, IBMKeyManager, OpenSSL, etc.

A continuación se proporciona un ejemplo de cómo generar dicho almacén utilizando OpenSSL.

Como primer paso se crea el par clave privada (cakey.pem) y clave pública (cacert.pem) de nuestra CA utilizando el algoritmo RSA (pues DSA solo sirve para firmar). Se elige un largo de bits (2048) acorde a la seguridad que una CA necesita.

```
openssl req -x509 -newkey rsa:4096 -days 3650 -keyout ca\private\cakey.pem -out ca\cacert.pem -config openssl.cfg
```

Se crea para el Cliente el par clave privada (clientkey.pem) y CSR (client.cert.req) que serán enviados a la CA (CSR significa Certificate Signing Request o solicitud para firmado de certificado).

```
openssl req -newkey rsa:1024 -keyout client\private\clientkey.pem -out client\csr\client.cert.req -config openssl.cfg
```

A partir del CSR del Cliente (client.cert.req), se crea un certificado firmado X509 (versión 3) con la clave privada de la CA (clientcert.pem).

```
openssl ca -days 3650 -in client\csr\client.cert.req -out client\signed\clientcert.pem -config openssl.cfg
```

Finalmente se exportan a formato PKCS#12 (client.p12) la clave privada del Cliente (clientkey.pem) y su certificado emitido por la CA (clientcert.pem).

```
openssl pkcs12 -export -out client\client.p12 -inkey client\private\clientkey.pem -in client\signed\clientcert.pem
```

5.Implementación Java

Debido a antiguas regulaciones, el archivo de políticas de jurisdicción por omisión (JCE) incluido en el SDK (jdk1.7.051\jre\lib\security) permite utilizar una criptografía sólida, pero limitada.
Se deben cambiar dichos archivos por sus versiones irrestrictas. Estos nuevos archivos de políticas pueden ser bajados del siguiente link:

http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html

A continuación se incluye el código completo de una clase (XMLEncryptionSample) que expone dos métodos públicos (encrypt, decrypt) utilizando el estándar XMLEncryption (http://www.w3.org/2001/04/xmlenc). Esta clase debe ser adaptada con el manejo de errores que se considere necesario para un ambiente de producción.

Se debe utilizar como algoritmo asimétrico rsa-pkcs1 (http://www.w3.org/2001/04/xmlenc#rsa-1_5) y como algoritmo simétrico: 3DES-CBC (http://www.w3.org/2001/04/xmlenc#tripledes-cbc).

Finalmente como nombre de clave (KeyName) se debe utilizar CERT_DGI_EFACTURA.

```
package uy.gub.dgi.enc;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.security.Key;
import java.security.KeyStore;
```

```
import java.security.PrivateKey;
import java.security.PublicKey;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.apache.xml.security.encryption.EncryptedData;
import org.apache.xml.security.encryption.EncryptedKey;
import org.apache.xml.security.encryption.XMLCipher;
import org.apache.xml.security.keys.KeyInfo;
import org.apache.xml.security.utils.EncryptionConstants;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class XMLEncryptionSample {
	static {
		org.apache.xml.security.Init.init();
	}

	private static Document parseFile(String fileName) throws Exception {
		DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
		dbf.setNamespaceAware(true);
		DocumentBuilder db = dbf.newDocumentBuilder();
		return db.parse(fileName);
	}

	private static SecretKey generateSymmetricKey() throws Exception {
		String jceAlgorithmName = "DESede";
		KeyGenerator keyGenerator = KeyGenerator.getInstance(jceAlgorithmName);
		return keyGenerator.generateKey();
	}

	private static void writeDocToFile(Document doc, String fileName)
			throws Exception {
		FileOutputStream outStream = new FileOutputStream(new File(fileName));
		TransformerFactory factory = TransformerFactory.newInstance();
		Transformer transformer = factory.newTransformer();
		transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
		DOMSource source = new DOMSource(doc);
		transformer.transform(source, new StreamResult(outStream));
		outStream.close();
	}

	public static void encrypt(String source, String target, String ns, String element,
			Key publicKey, String keyName) throws Exception {

		// Lee XML desde archivo
		Document document = parseFile(source);

		// Genera clave simétrica para TripleDes
		Key symmetricKey = generateSymmetricKey();

		// Inicializa cifrador para cifrar la clave simétrica
		XMLCipher keyCipher = XMLCipher.getInstance(XMLCipher.RSA_v1dot5);
		keyCipher.init(XMLCipher.WRAP_MODE, publicKey);

		// Cifra la clave simetrica
		EncryptedKey encryptedKey = keyCipher
				.encryptKey(document, symmetricKey);

		// Especifica el elemento del documento XML a cifrar
		Element rootElement = document.getDocumentElement();
		Element elementToEncrypt = rootElement;
		if (element != null) {
			elementToEncrypt = (Element) rootElement.getElementsByTagNameNS(
					ns, element).item(0);
			if (elementToEncrypt == null) {
```

```
                        System.err.println("No se encuentra el elemento: " + element);
                }
        }

        // Inicializa cifrador para cifrar el Elemento XML
        XMLCipher xmlCipher = XMLCipher.getInstance(XMLCipher.TRIPLEDES);
        xmlCipher.init(XMLCipher.ENCRYPT_MODE, symmetricKey);

        // Agrega informacion de la clave de cifrado
        EncryptedData encryptedDataElement = xmlCipher.getEncryptedData();
        KeyInfo innerKeyInfo = new KeyInfo(document);
        innerKeyInfo.addKeyName(keyName);
        encryptedKey.setKeyInfo(innerKeyInfo);
        KeyInfo keyInfo = new KeyInfo(document);
        keyInfo.add(encryptedKey);
        encryptedDataElement.setKeyInfo(keyInfo);

        // Cifra
        xmlCipher.doFinal(document, elementToEncrypt);

        // Escribe el resultado en el archivo destino
        writeDocToFile(document, target);
    }

    public static void decrypt(String source, String target, Key privateKey)
            throws Exception {
        // Lee XML desde archivo
        Document document = parseFile(source);

        // Obtiene el Elemento XML con datos cifrados
        String namespaceURI = EncryptionConstants.EncryptionSpecNS;
        String localName = EncryptionConstants._TAG_ENCRYPTEDDATA;
        Element encryptedDataElement = (Element) document
                .getElementsByTagNameNS(namespaceURI, localName).item(0);

        // Se desencripta la clave simétrica
        XMLCipher xmlCipher = XMLCipher.getInstance();
        xmlCipher.init(XMLCipher.DECRYPT_MODE, null);
        xmlCipher.setKEK(privateKey);

        // Se reemplaza el nodo encriptado con la información desencriptada
        xmlCipher.doFinal(document, encryptedDataElement);

        // Escribe el resultado en el archivo destino
        writeDocToFile(document, target);
    }
}
```

La carga de la clave privada y pública desde un keystore (#PCKS12) de nombre client.p12 cuyo alias es client, puede programarse de la siguiente manera:

```
PrivateKey privateKey = null;
PublicKey publicKey = null;
….
….

KeyStore keystore = KeyStore.getInstance("PKCS12");
String p12Password = "secreto";
keystore.load(new FileInputStream("client.p12"), p12Password.toCharArray());
privateKey = (PrivateKey) keystore.getKey("client", p12Password.toCharArray());
publicKey = keystore.getCertificate("client").getPublicKey();
```

Finalmente, para efectuar una prueba, se pueden utilizar los métodos encrypt/decrypt de la siguiente manera:

```
encrypt("SinEncriptar.xml", "Encriptado.xml", "http://cfe.dgi.gub.uy", "Compl_Fiscal_Data",
publicKey, "CERT_DGI_EFACTURA");

decrypt("Encriptado.xml", "Desencriptado.xml", privateKey);
```

Es importante señalar que la información a enviar a DGI debe ser encriptada con la clave pública de DGI, de esta manera solo DGI podría desencriptarla.

Para todos los ambientes se debe utilizar la clave pública de DGI, asociada al RUT 214844360018.

6.Implementación .Net

El siguiente ejemplo es una aplicación de consola desarrollada utilizando Visual Studio express 2013 for desktop.

En su método Main, carga el archivo xml a cifrar y el certificado desde el archivo del almacén de certificados, suponiendo que ambos se encuentran en el directorio de la aplicación.

La ejecución sigue de la siguiente manera:

- Se selecciona el elemento del documento que se va a cifrar
- Se cifra el elemento y se guarda el xml obtenido
- Se vuelve a cargar el documento cifrado desde el archivo recientemente guardado.
- Se descifra el documento y se guarda con un nuevo nombre. Este documento debería ser igual al documento inicial.

Para poder ejecutar el ejemplo se debe crear una nueva aplicación de consola y agregar una referencia (Add Reference…) y seleccionamos el “assembly” System.Security.

Sustituir el contenido del archivo Program.cs por el siguiente código (recordar que se debe cambiar el nombre del archivo xml original, el del almacén de certificados y la contraseña, por los correspondientes en su ambiente, el xpath del nodo a cifrar podría ser diferente según el documento utilizado).

Se debe utilizar como algoritmo asimétrico rsa-pkcs1 (http://www.w3.org/2001/04/xmlenc#rsa-1_5) y como algoritmo simétrico: 3DES-CBC (http://www.w3.org/2001/04/xmlenc#tripledes-cbc).

Finalmente como nombre de clave (KeyName) se debe utilizar CERT_DGI_EFACTURA.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.Xml;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using System.Security.Cryptography;

namespace DgiEncriptarXml {
    class Program {
        /// <summary>
        /// Lee y escribe todos los archivos desde el directorio de la aplicación.
        /// </summary>
        /// <param name="args"></param>
        static void Main(string[] args) {

            const string NOMBRE_CLAVE = "CERT_DGI_EFACTURA";

            // Se carga el documento a cifrar
            var xmlDoc = CargarXml(@"EnvioCfe net sin encriptar.xml");
            // Se carga el certificado que se va a utilizar.
            // Se utiliza la clave pública para cifrar y la clave privada para descifrar
            var cert = CargarCertificado("client.p12", "secreto");

            // Se selecciona el nodo del documento que se cifrará
            var xmlNM = new XmlNamespaceManager(xmlDoc.NameTable);
            xmlNM.AddNamespace("ns0", "http://cfe.dgi.gub.uy");
            var elm =
xmlDoc.SelectSingleNode("/ns0:EnvioCFE/ns0:CFE/ns0:eFact/ns0:Compl_Fiscal/ns0:Compl_Fiscal_Data",
xmlNM) as XmlElement;
```

```
            //Encriptamos
            EncriptarXml(elm, cert, NOMBRE_CLAVE);

            //Guardo el documento cifrado
            xmlDoc.Save("EnvioCfe_net_encriptado.xml");

            //Cargo nuevamente el documento cifrado
            xmlDoc = CargarXml("EnvioCfe_net_sin_encriptar.xml");

            // Se descifra el documento
            DesencriptarDocumento(xmlDoc, cert, NOMBRE_CLAVE);

            //Guardo el documento descifrado
            xmlDoc.Save("EnvioCfe_net_desencriptado.xml");

        }

        /// <summary>
        /// Carga un certificado desde un almacén tipo #PKCS12
        /// El certificado podría cargarse desde otros lugares,
        /// por ejemplo, el almacén de windows
        /// </summary>
        /// <param name="path">Camino al archivo</param>
        /// <param name="contrasenia">Contraseña del almacén</param>
        /// <returns></returns>
        private static X509Certificate2 CargarCertificado(string path, string contrasenia) {
            return new X509Certificate2(path, contrasenia, X509KeyStorageFlags.Exportable);
        }

        /// <summary>
        /// Carga el documento xml que contiene el nodo a cifrar
        /// </summary>
        /// <param name="path">Camino al archivo</param>
        /// <returns></returns>
        private static XmlDocument CargarXml(string path) {
            var xmldoc = new XmlDocument();
            xmldoc.PreserveWhitespace = false;
            xmldoc.Load(path);
            return xmldoc;
        }


        /// <summary>
        /// Cifra el nodo utilizando la clave pública del certificado suministrado.
        /// </summary>
        /// <param name="nodoParaEncriptar">Elemento para cifrar</param>
        /// <param name="cert">Certificado </param>
        /// <param name="nombreClave">Nombre para la clave</param>
        private static void EncriptarXml(XmlElement nodoParaEncriptar, X509Certificate2 cert, string
nombreClave) {

            // Se crea una nueva clave TripleDES.
            TripleDESCryptoServiceProvider tDESkey = new TripleDESCryptoServiceProvider();

            // Se crea una nueva instancia de EncryptedXml y la uso para encriptar el elemento con la
clave simétrica.
            EncryptedXml eXml = new EncryptedXml();

            byte[] encryptedElement = eXml.EncryptData(nodoParaEncriptar, tDESkey, false);


            // Se construye el objeto EncryptedData y se carga la información de cifrado deseada
            EncryptedData edElement = new EncryptedData();
            edElement.Type = EncryptedXml.XmlEncElementUrl;
            edElement.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncTripleDESUrl);

            var alg = (RSACryptoServiceProvider)cert.PublicKey.Key;
            // Se cifra la clave simétrica.
            EncryptedKey ek = new EncryptedKey();

            byte[] encryptedKey = EncryptedXml.EncryptKey(tDESkey.Key, alg, false);

            ek.CipherData = new CipherData(encryptedKey);
```

```
            ek.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncRSA15Url);

            //Agrega la clave cifrada al objeto EncriptedData
            edElement.KeyInfo.AddClause(new KeyInfoEncryptedKey(ek));


            // Asigna el Elemento KeyInfoName para especificar el nombre de la clave RSA
            KeyInfoName kin = new KeyInfoName();
            kin.Value = nombreClave;

            // Agrega el KeyInfoName al objeto encriptado
            ek.KeyInfo.AddClause(kin);

            edElement.CipherData.CipherValue = encryptedElement;

            EncryptedXml.ReplaceElement(nodoParaEncriptar, edElement, false);

        }

        /// <summary>
        /// Descifra el documento utilizando la clave privada del certificado suministrado
        /// </summary>
        /// <param name="xmlDoc">Documento a descifrar</param>
        /// <param name="cert">Certificado a utilizar</param>
        /// <param name="nombreClave">Nombre de la clave simétrica</param>
        private static void DesencriptarDocumento(XmlDocument xmlDoc, X509Certificate2 cert, string
nombreClave) {

            EncryptedXml exml = new EncryptedXml(xmlDoc);

            // Agrega el diccionario clave-nombre
            // Este método sólo puede descifrar documentos
            // que contengan la clave especificada
            var privateKey =(RSACryptoServiceProvider)cert.PrivateKey;
            exml.AddKeyNameMapping(nombreClave, privateKey);


            // Descifrar el elemento.
                exml.DecryptDocument();
        }
    }
}
```

Es importante señalar que la información a enviar a DGI debe ser encriptada con la clave pública de DGI, de esta manera solo DGI podría desencriptarla.

Para todos los ambientes se debe utilizar la clave pública de DGI, asociada al RUT 214844360018.