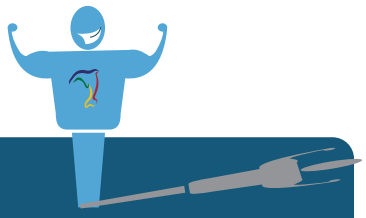
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

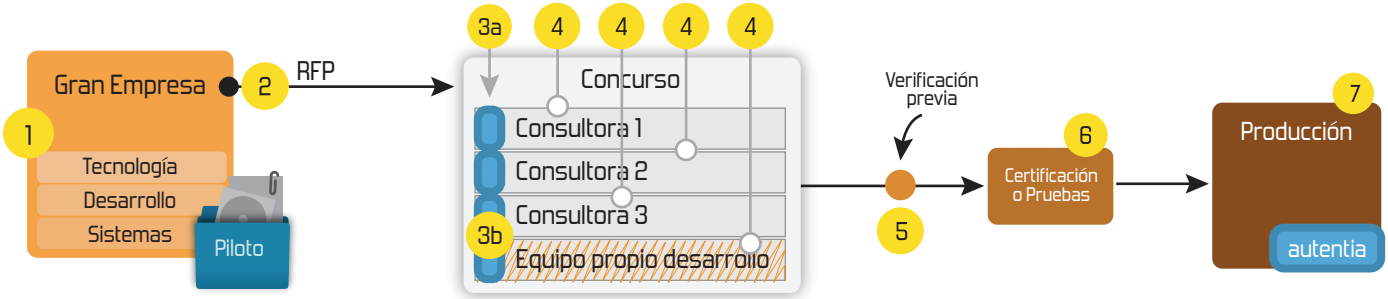
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Alberto Carrasco Montenegro

Puedes encontrarme en Autentia

XMLEncryption en Java

En Autentia estamos siempre trabajando con nuevas tecnología y hoy, os vamos a mostrar una introducción al estándar de seguridad XML Encryption a través del lenguaje Java

1. ¿Qué es *XMLEncryption*?

XML Encryption es un lenguaje cuya función principal es asegurar la confidencialidad de partes de documentos *xml*, a través de la encriptación parcial del documento transportado, si bien puede encriptarse también el documento en su totalidad. *XML Encryption* se puede aplicar a cualquier recurso Web, incluyendo contenido que no es *xml*.

La especificación del cifrado de W3C's XML se ocupa de la seguridad de los datos usando técnicas de cifrado. Según lo definido por la especificación, con el cifrado de *xml*, las etiquetas de *xml* contienen los datos cifrados.

A modo de ejemplo, se supone el siguiente documento *xml* que contiene información acerca del pago realizado por un cliente con su tarjeta de crédito:

```
<?xml version='1.0'?>
  <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <CreditCard Limit='5,000' Currency='USD'>
          <Number>4019 2445 0277 5567</Number>
          <Issuer>Example Bank</Issuer>
          <Expiration>04/02</Expiration>
      </CreditCard>
  </PaymentInfo>
```

Este documento contiene información confidencial del cliente que interesa mantener en secreto. Una versión de este documento cifrando el elemento con la información acerca de la tarjeta de crédito podría ser la siguiente:

```
<?xml version='1.0'?>
  <PaymentInfo xmlns='http://example.org/paymentv2'>
    <Name>John Smith</Name>
    <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
          <CipherData>
            <CipherValue>A23B45C56</CipherValue>
          </CipherData>
    </EncryptedData>
  </PaymentInfo>
```

Cifrando el elemento entero *CreditCard* desde su etiqueta inicial hasta su etiqueta final, la identidad del elemento queda oculta. El elemento de *CipherData* contiene la serialización cifrada del elemento de *CreditCard*.

Este es un ejemplo muy simple que se muestra a modo ilustrativo. En realidad, el estándar *XML Encryption* emplea diversos algoritmos de cifrado, tanto simétricos como asimétricos, de forma que los documentos *xml* cifrados suelen contener más información, como por ejemplo:

- Información contenida en el documento inicial, cifrada.
- Algoritmo empleado en el cifrado de dicha información.
- Clave cifrada.
- Algoritmo empleado en el cifrado de la clave.

La principal ventaja que proporciona este estándar reside en que proporciona seguridad sobre la información en un trayecto punto a punto donde cada receptor debe ser capaz de acceder solo a cierta parte de la información. Del mismo modo la información permanece oculta ante posibles escuchas del canal, ya que va cifrada.

No se entrará en grandes detalles en el estudio de las especificaciones del W3C para implementar mecanismos que utilicen el estándar *XML Encryption*, pero pueden consultarse todas ellas en la web de la W3C *http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/*

2. El *framework* de Apache para Java

Existen diversos *frameworks* que contienen una implementación del estándar *XML Encryption* para poder emplear su tecnología de seguridad en aplicaciones convencionales.

Uno de ellos es el *framework* para Java de Apache y puede obtenerse de forma gratuita desde su web. Este es el que se utilizará para implementar el ejemplo que se estudiará en este documento.

2.1. Instalación y requisitos

El ejemplo estudiado en este documento se desarrollo y ejecutó en un entorno *Windows XP* que disponía de la distribución Java *j2sdk-1.4.2*, que puede obtenerse gratuitamente desde la web de *SUN* *http://java.sun.com/products/archive/j2se/1.4.2/index.html*

Con la versión de Java utilizada es necesario utilizar una distribución estable de *Xalan*, necesaria para el proceso de documentos *xml*. La utilizada en este ejemplo fue la 2.7 (es necesario una 2.2 o superior) que se puede obtener en la dirección *http://apache.gva.es/xml/xalan-j/* . El archivo utilizado fue el *xalan-j_2_7_0-bin.zip*.

De la misma forma, y destinado igualmente al proceso de documentos *xml*, es necesario utilizar la distribución de Apache *Commons*. La versión utilizada fue la 1.0.4 y puede obtenerse en la dirección *http://jakarta.apache.org/site/downloads/downloads_commons-logging.cgi* . El archivo utilizado fue el *commons-logging-1.0.4.zip* .

El *framework* para Java de Apache utilizado es el que lleva la distribución *xmlsec-1.2.97.jar* que puede descargarse desde la web de Apache en la dirección *http://xml.apache.org/security/dist/java-library/*

Una vez descomprimidos los dos ficheros *.zip* y junto con el archivo *.jar* correspondiente al *framework*, el conjunto de paquetes necesarios para poder utilizar dicho *framework* es el siguiente:

- xalan-j_2_7_0**serializer.jar**
- xalan-j_2_7_0**xalan.jar**
- xalan-j_2_7_0**xercesImpl.jar**
- xalan-j_2_7_0**xml-apis.jar**
- commons-logging-1.0.4**commons-logging.jar**
- **xmlsec-1.2.97.jar**

Estos paquetes deben ser visibles a las clases que utilicen el *framework*, de forma que deberán ser incluidos en el *classpath* a la hora de compilar y ejecutar.

Para poder utilizar el *framework* es fundamental disponer de un proveedor de JCE (*Java Criptography Extensión*) para poder utilizar toda la variedad de algoritmos que ofrece la criptografía moderna y valerse de su potencia y fiabilidad. El proveedor de *SUN* que viene por defecto con la versión de Java especificada es muy básico e insuficiente para nuestros propósitos.

En este caso se ha utilizado el proveedor gratuito proporcionado por el equipo de *Bouncy Castle* (*http://www.bouncycastle.org/*) y se procederá a explicar su instalación:

- Descargar el paquete del proveedor de JCE. En este caso se utilizó el paquete *bcprov-jdk14-130.jar* que puede descargarse desde *http://www.bouncycastle.org/latest_releases.html*
- Copiar dicho paquete a los directorios <JAVA_HOME>/jre/lib/ext/ y *C:\Archivos de programa\Java\j2re1.4.2\lib\ext* (incluir en ambos).
- Añadir el proveedor de JCE descargado a la lista de los ya existentes, añadiendo la siguiente línea al fichero *java.security* situado en <JAVA_HOME>/jre/lib/security/java.security y en C:\Archivos de programa\Java\j2re1.4.2\lib\security (incluir en ambos):

  security.provider.<n>=org.bouncycastle.jce.provider.BouncyCastleProvider

  donde **<n>** corresponde al orden de preferencia que empleará Java cuando necesite un proveedor de seguridad. En este caso se añadió en sexto lugar, a continuación de los que vienen por defecto de SUN, de forma que se añadió la siguiente línea en los ficheros indicados:

  security.provider.6=org.bouncycastle.jce.provider.BouncyCastleProvider

  Es importante que los primeros proveedores indicados con dicho orden de preferencia sean los que vienen por defecto de SUN, de forma que debe situarse en último lugar, como hemos indicado, el que se desea incluir para nuestros propósitos.

Una vez cumplidos estos requisitos, se puede utilizar el framework de Java especificado para utilizar el estándar XML Encryption. A continuación se explicará un pequeño ejemplo realizado con dicho framework.

3. Un ejemplo sencillo

En este apartado se procederá a explicar un ejemplo sencillo utilizando el *framework* de Apache que proporciona soporte para el estándar *XML Encryption*.

3.1. Idea

La aplicación contiene tres programas principales (Cliente, Tienda y Banco) que interaccionarán entre ellos con las siguientes premisas:

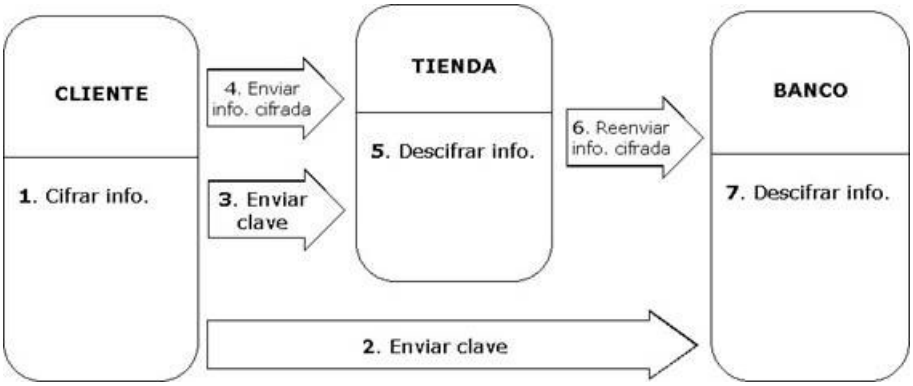
- Cliente: Debe enviar información al servidor del banco a través del servidor de la Tienda. Cada uno de ellos debe poder acceder sólo a cierta información y no así al resto de ella. Para ello cifra la información inicial para que solo sea accesible en parte para la Tienda y en parte para el Banco.
- Tienda: Recibe la información cifrada del Cliente y debe reenviarla al servidor del Banco del Cliente. Debe poder acceder a cierta información, tal cual se acaba de explicar, y reenviar la información recibida desde el Cliente al servidor del Banco.
- Banco: Recibe la información del servidor de la Tienda. Realiza el acceso sólo a cierta información cifrada del Cliente, recibida a través del servidor de la Tienda.

¿Cómo se consigue que el mismo documento sea accesible sólo a ciertas partes por cada uno de los receptores? La información que se desea hacer visible para cada receptor se cifra con una clave que se le proporcionará para que sólo él pueda acceder.

En este caso, la información enviada por el Cliente será un documento *xml* que contiene los datos relativos a una compra online, cuyo pago la Tienda realiza a través de un servidor que comunica con el banco del Cliente. Se cifrarán los elementos *Name* y *Amount* para que solo pueda accederlos el servidor de la Tienda, el elemento *CreditCardNumber* para que sólo pueda accederlo el servidor del Banco, y el elemento *Date* se dejará sin cifrar. El fichero *xml* con la información de la compra del Cliente será de este estilo:

```
<?xml version='1.0'?>
   <PaymentInfo xmlns='http://trincadordepasta.com/payments'>
      <Name>John Doe</Name>
      <Amount>125.00</Amount>
      <CreditCardNumber>1234-5678-8765-4321</CreditCardNumber>
      <Date>July 6, 2005</Date>
   </PaymentInfo>
```

El siguiente diagrama refleja el desarrollo del escenario indicado:



Las claves enviadas previamente (2, 3 en el gráfico superior) a la Tienda y al Banco para que puedan acceder a la información para la que tengan permiso, son las claves para el cifrado de las claves empleadas en el cifrado de la información de los elementos *xml*. Cabe recordar que el documento *xml* cifrado contendrá, entre otras cosas, información sobre la clave de cifrado de la clave empleada en el cifrado de la información *xml*, como puede ser su valor cifrado, el algoritmo empleado para su cifrado, etc. Recordar igualmente que la información enviada en los pasos 4 y 6 es la misma, esto es, el documento *xml* del Cliente cifrado en parte para que acceda solo la Tienda y en parte para que acceda solo el Banco.

Como se explicó anteriormente, la información cifrada no sólo permanece oculta ante usuarios no autorizados en el trayecto punto a punto (Cliente-Tienda-Banco), sino también frente a usuarios externos que pudieran escuchar el canal y no dispongan de ninguna clave.

3.2. Requisitos para implementación

Como se dijo anteriormente, el ejemplo se desarrollará utilizando el lenguaje Java. Necesitará de las siguientes tecnologías software implementadas para el mismo:

- *Framework* para implementar el estándar *XML Encryption*. Se utilizará el de Apache, cuya instalación se explico anteriormente.
- *Sockets* para implementar los mecanismos de transporte de información basados en *TCP/IP*. Se utilizarán los que ya vienen con la propia distribución de Java.

La información que utiliza el flujo de la aplicación procederá de ficheros en formato *xml*, cuyo procesamiento para cifrar y descifrar se realizará a través de los mecanismos que proporciona el *framework* de Apache que se utilizará.

3.3. Implementación de la seguridad

3.3.1 Cifrado

Para realizar la función de cifrado del documento *xml* es necesario el código que se expone a continuación. La idea es que cifre la información de un elemento del documento *xml,* de forma que el documento cifrado contenga la información cifrada y la información de la clave cifrada. Para poder descifrar la clave se almacenará su clave de descifrado en un archivo de forma que pueda recuperarse en un futuro la clave original, y con ella la información del elemento cifrado original.

En primer lugar será necesario importar las siguientes librerías para poder utilizar funciones de procesamiento de documentos *xml,* algoritmos de cifrado y funcionas básicas de entrada/salida:

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.Key;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
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.w3c.dom.Document;
import org.w3c.dom.Element;
```

En la clase en que se implemente el método de cifrado debe inicializarse en antes de nada el *framework* de Apache para poder utilizarlo. El código necesario es el siguiente:

```
static
        {
            org.apache.xml.security.Init.init();
        }
```

Antes de pensar en cifrar el documento *xml,* debe realizarse la implementación necesaria para parsearlo y poder utilizar las librerías de procesamiento *xml* de Java. Se empleo la siguiente función, que recibe como argumento el nombre del fichero a parsear y devuelve un objeto *Document*:

```
private static Document parseFile(String fileName)
        throws Exception
    {
        javax.xml.parsers.DocumentBuilderFactory dbf =
javax.xml.parsers.DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        javax.xml.parsers.DocumentBuilder db = dbf.newDocumentBuilder();
        Document document = db.parse(fileName);

        return document;
    }
```

Como se dijo antes, la clave que se utilizará para cifrar el elemento *xml* irá cifrada en el documento *xml* resultante. Se necesita generar una clave para cifrar la clave que se empleará en el cifrado del elemento *xml*. Se utilizó el siguiente código:

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

Para el cifrado de la información se utilizará un algoritmo simétrico (en concreto AES) de forma que será necesaria una función para generar la clave simétrica que se utilizará tanto para el cifrado como para el descifrado. El código encargado de esto es el siguiente:

```
private static SecretKey GenerateSymmetricKey()
        throws Exception
    {
        String jceAlgorithmName = "AES";
        KeyGenerator keyGenerator = KeyGenerator.getInstance(jceAlgorithmName);
        keyGenerator.init(128);
        return keyGenerator.generateKey();
    }
```

Finalmente, se necesitará almacenar en un fichero el resultado de la encriptación del documento *xml* a través del procesamiento de objetos *Document*. Esta tarea la realizamos con el siguiente código:

```
private static void writeEncryptedDocToFile(Document doc, String fileName) throws Exception
    {
        File encryptionFile = new File(fileName);
        FileOutputStream outStream = new FileOutputStream(encryptionFile);

        TransformerFactory factory = TransformerFactory.newInstance();
        Transformer transformer = factory.newTransformer();
        transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(outStream);
        transformer.transform(source, result);

        outStream.close();
    }
```

Con la implementación de estos métodos ya se dispone de casi todo el material auxiliar para realizar la encriptación del documento *xml*. El método principal para encriptar recibe como argumentos el fichero *xml* a encriptar, el fichero donde se guardara el resultado, el elemento a cifrar y la clave de cifrado de la clave que se usará para el cifrado del elemento *xml* . Su estructura sería la siguiente (en azul se marca el código nuevo):

```
public static void Encrypt(String source, String target, String
element, Key keyEncryptKey)
    throws Exception
  {

    // Parsea fichero xml en elmento tipo Document
    Document document = parseFile(source);

    // Genera clave simetrica
    Key symmetricKey = GenerateSymmetricKey();

    // Inicializa cifrador para cifrar la clave de cifrado de //
la informacion del documento xml
    XMLCipher keyCipher = XMLCipher.getInstance(XMLCipher.TRIPLEDES_KeyWrap);
    keyCipher.init(XMLCipher.WRAP_MODE, keyEncryptKey);

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

    // Especifica elemento del fichero xml a cifrar
    Element rootElement = document.getDocumentElement();
    Element elementToEncrypt = rootElement;
    if (element!=null)
    {
        elementToEncrypt = (Element)rootElement.getElementsByTagName
(element).item(0);
        if (elementToEncrypt == null)
        {
            System.err.println("Unable to find element: " + element);
            System.exit(1);
        }
    }

    // Inicializa cifrador para cifrar el elemento xml
    XMLCipher xmlCipher = XMLCipher.getInstance(XMLCipher.AES_128);
    xmlCipher.init(XMLCipher.ENCRYPT_MODE, symmetricKey);

    // Anade informacion de la clave de cifrado
    EncryptedData encryptedDataElement = xmlCipher.getEncryptedData();
```

```
        KeyInfo keyInfo = new KeyInfo(document);
        keyInfo.add(encryptedKey);
        encryptedDataElement.setKeyInfo(keyInfo);

        // Cifra
        boolean encryptContentsOnly = true;
        xmlCipher.doFinal(document, elementToEncrypt, encryptContentsOnly);

        // Escribe el resultado en el fichero destino
        writeEncryptedDocToFile(document, target);
    }
```

Este código será el que empleará el cliente para cifrar la información antes de enviársela al servidor la tienda, a fin de que, al igual que el servidor del banco, sólo pueda acceder a un aparte concreta de la información.

3.3.2 Descifrado

Para descifrar un documento *xml* se necesita en primer lugar, al igual que para cifrar, inicializar el *framework* de Apache tal como se hizo en el apartado anterior:

```
static
        {
            org.apache.xml.security.Init.init();
        }
```

Es necesario importar las mismas librerías que en el apartado anterior.

De igual forma que para el cifrado, se necesita una función que cargue el documento *xml* en un objeto *Document* para que Java pueda procesarlo. El código es similar al de la función *parseFile* del apartado anterior y no se expone.

Se necesitará también una función para cargar desde un archivo la clave necesaria para descifrar la clave con la que se cifro el elemento *xml*. El código empleado para esta tarea es el siguiente:

```
private static Document loadEncryptedFile(String fileName)
        throws Exception
    {
        File encryptedFile = new File(fileName);
        javax.xml.parsers.DocumentBuilderFactory dbf =
javax.xml.parsers.DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(true);
        javax.xml.parsers.DocumentBuilder builder = dbf.newDocumentBuilder();
        Document document = builder.parse(encryptedFile);

        return document;
    }
```

Al igual que en el apartado de cifrado, se usará la función que volcará la información del documento *xml* descifrado en un fichero de texto, a partir de un objeto *Document*. El código es similar al de la función *writeEncryptedDocToFile* del apartado anterior y no se expone.

Con estas funcionalidades implementadas ya hay mucho trabajo hecho para descifrar un documento *xml*. El código resultante del método que descifra es el siguiente (en azul se marca el código nuevo):

```
public static void Decrypt(String source, String target, String
keyFile, int elemCif)
        throws Exception
    {
        // Carga el fichero xml origen en elemento tipo Document
        Document document = loadEncryptedFile(source);

        // Obtiene elemento con datos cifrados
        String namespaceURI = EncryptionConstants.EncryptionSpecNS;

        String localName = EncryptionConstants._TAG_ENCRYPTEDDATA;

        Element encryptedDataElement = (Element)document.getElementsByTagNameNS
(namespaceURI, localName).item(elemCif);

       //Carga la clave de cifrado de la clave que cifro
la //informacion del documento xml
        Key keyEncryptKey = loadKeyEncryptionKey(keyFile);


        // Inicializa cifrador
```

```
    XMLCipher xmlCipher = XMLCipher.getInstance();
    xmlCipher.init(XMLCipher.DECRYPT_MODE, null);

    xmlCipher.setKEK(keyEncryptKey);

    // Descifra
    xmlCipher.doFinal(document, encryptedDataElement);

    // Escribe el resultado en un fichero
    writeDecryptedDocToFile(document, target);
}
```

3.4. Ejecución

No se entrará en detalles sobre la implementación de los programas principales Cliente, Tienda y Banco, pero si se mostrarán a continuación las trazas resultantes de los mismos con resultados satisfactorios.

En primer lugar deben iniciarse los servidores de la Tienda y el Banco (no importa el orden), que quedarán a la espera de que el cliente inicie el proceso enviando la información al servidor de la Tienda. El servidor del Banco no necesita argumentos, si bien el servidor de la Tienda precisa de la dirección *IP* del servidor del Banco. El servidor de la Tienda quedará a la escucha por el puerto 6001 mientras que el del banco quedará a la escucha por el 6000.

A continuación debe iniciarse el cliente con los argumentos que necesite (fichero *xml* con la información que el cliente desea enviar, así como las direcciones *IP* de los servidores de la Tienda y el Banco respectivamente).

Una vez puesto en funcionamiento el proceso, cada elemento del sistema realiza sus tareas y las trazas que pueden observarse por la salida estándar son las siguientes. Las trazas corresponden a una ejecución donde los tres elementos del sistema (Cliente, Tienda y Banco) se encontraban en la misma máquina.

Para el Cliente:

```
-------------------------
CLIENT
-------------------------

[@] Information encrypted for Shop's server
[@] Information encrypted for Bank's server
[@] Connected to Bank's server localhost
[@] Key sended to Bank's server localhost
[@] Connected to Shop's server localhost
[@] Key sended to Shop's server localhost
[@] Information sended to Shop's server localhost
```

Para el servidor de la Tienda:

```
-------------------------
    SHOP'S SERVER
-------------------------
-> Port: 6001
-------------------------

> Key received and stored
> Information from client received
> Information from client decrypted in decryptedInfoN.xml
> Connected to Bank's server localhost
> Information sended to Bank's server localhost
```

Para el servidor del Banco:

```
-------------------------
      BANK'S SERVER
-------------------------
-> Port: 6000
-------------------------

>> Key received and stored
>> Information from Shop's server received
>> Information from Shop's server decrypted in decryptedInfoS.xml
```

Como puede observarse, al finalizar el proceso el servidor de la Tienda habrá almacenado el fichero

decryptedInfoN.xml y el Banco el fichero *decryptedInfoS.xml*. Estos ficheros contendrán el fichero *xml* cifrado enviado inicialmente por el cliente con los elementos para los que cada uno de ellos tiene acceso, descifrados. Recordemos que el servidor de la Tienda tendría acceso a los elementos *Name* y *Amount* y el del Banco al elemento *CreditCardNumber*. El elemento *Date* no se cifró por parte del Cliente y es visible a cualquiera de los dos.

A continuación se mostrará el contenido de los ficheros *decryptedInfoN.xml* y *decryptedInfoS.xml*. Como podrá observarse, cada uno de ellos contendrá cifrados los elementos para los que no tiene acceso mediante una clave de descifrado. Dichos elementos contendrá diversos campos de información, como son el algoritmo de cifrado de la información, la información cifrada, la clave cifrada, algoritmo con el que se cifró la clave, etc.

decryptedInfoN.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<PaymentInfo xmlns="http://trincadordepasta.com/payments">
<Name>John Doe</Name>
<Amount>125.00</Amount>
<CreditCardNumber>
<xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
<xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">IqsIrGZI7Up3E1gLNhSp8J0OyStfDs+3WvMVShFeSZs=
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedKey>
</ds:KeyInfo>
<xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">iCH/ayrr5EFP/4PLec0WiFEAQJ39qJjXhfKQhSTNQYkc30r4rSawiBfnDu4CVCrO
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</CreditCardNumber>
<Date>July 6, 2005</Date>
</PaymentInfo>
```

decryptedInfoS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<PaymentInfo xmlns="http://trincadordepasta.com/payments">
<Name>
<xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
<xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">w8ff5iUEu306HNJljCgdB58GfaUWVle2y6g23dQwKw4=</xenc:CipherValue>
</xenc:CipherData></xenc:EncryptedKey>
</ds:KeyInfo>
<xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">3wJ8V/ywma3mKafSUA+yrR3wsVCmNu1pNx/MOfQooMU=
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</Name>
<Amount>
<xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
<xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">ASwzSlC0cnuQxOrHzQ4XWENXxCqBrtA2TgHNMFdpk9E=
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedKey>
</ds:KeyInfo>
<xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">A33RsELEmkEn/tcrCNn58cvv0Cud3gxxiyy9wXNFnms=
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</Amount>
<CreditCardNumber>1234-5678-8765-4321</CreditCardNumber>
<Date>July 6, 2005</Date>
</PaymentInfo>
```

4. Fuentes utilizadas

- Página con las especificaciones de la W3C sobre *XML Encryption*: *http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/*
- Página del proyecto Apache sobre *XML Encryption*: *http://xml.apache.org/security/*
- Introducción a seguridad, *XML Encryption* con Java, ejemplos: *http://www.devx.com/xml/Article/28701/0/page/1*
- Proveedor de JCE gratuito, Bouncy Castle:

http://www.bouncycastle.org/

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Pruebas Web con JWebUnit	Os mostramos como automatizar las pruebas de caja negra (desde el punto de vista de usuario final) de vuestro Web con el Framework gratuito JWebUnit. Esta técnica es perfecta para crear test de regresión de aplicaciones Web complejas.
Soporte XML en Eclipse con X-MEN	Alejandro Perez nos enseña como potenciar el entorno eclipse para facilitarnos el trabajo con ficheros xml, gracias al pluggin X-MEN
Edición de ficheros RSS con RssEditor	Os mostramos como crear y administrar los ficheros de canales RSS a través de la extensión gratuita de FireFox RssEditor
JDO con OJB	Os mostramos como configurar el entorno OJB de apache para construir la primera aplicación JDO
Transformación de XML y XSL en JSPs	Os mostramos como poder utilizar XML y XSL en JSPS, combinado con el Patrón MVC
Desarrollo Struts con XDoclet	Alejandro Perez nos enseña como simplificar el desarrollo de aplicaciones J2EE basadas en Struts, automatizando la generación de código con XDoclet
AdRotator en Página ASP .Net	Ismael Caballero nos cuenta como utilizar el control AdRotator en una página ASP .Net
Aplicación de Patrones de Diseño en Java	En este tutorial os mostramos como las técnicas avanzadas de diseño (como patrones de diseño) contribuyen a la contrucción de aplicaciones profesionales en Java.
Lectores de RSS	Os mostramos como instalar y utilizar un lector de RSS (sindicación simple) gratuito, llamado FeedReader, que nos permiten agregar titulares y noticias de nuestros Webs favoritos
Patrones de diseño J2EE	Os mostramos una interpretación particular de los patrones de diseño J2EE