Consejería de Hacienda y Administración Pública

Manual de migración del cliente

Sevilla, noviembre de 2010

Hoja de control		
Fecha	**Autor**	**Descripción**
03/11/2010	RPV	Adaptación manual de migración del cliente MPR

1 Introducción

El Cliente de Firma es una herramienta de Firma Electrónica que funciona en forma de Applet de Java integrado en una página Web mediante JavaScript.

El Cliente hace uso de los certificados digitales X.509 y de las claves privadas asociadas a los mismos que estén instalados en el repositorio o almacén de claves y certificados (*keystore*) del navegador web (*Internet Explorer, Mozilla, Firefox*) o el sistema operativo así como de los que estén en dispositivos (tarjetas inteligentes, dispositivos *USB*) configurados en el mismo (el caso de los DNI-e).

El Cliente de Firma, como su nombre indica, es una aplicación que se ejecuta en cliente (en el ordenador del usuario, no en el servidor Web). Esto es así para evitar que la clave privada asociada a un certificado tenga que "salir" del contenedor del usuario (tarjeta, dispositivo USB o navegador) ubicado en su PC. De hecho, nunca llega a salir del navegador, el Cliente le envía los datos a firmar y éste los devuelve firmados.

El Cliente de Firma contiene las interfaces y componentes web necesarios para la realización de los siguientes procesos (además de otros auxiliares como cálculos de hash, lectura de ficheros, etc…):

- Firma de formularios Web.
- Firma de datos y ficheros.
- Multifirma masiva de datos y ficheros.
- Cofirma (CoSignature) → Multifirma al mismo nivel.
- Contrafirma (CounterSignature) → Multifirma en cascada.

Como complemento al cliente de firma, se encuentra un cliente de cifrado que nos permite realizar las funciones de encriptación y desencriptación de datos atendiendo a diferentes algoritmos y configuraciones. Además permite la generación de sobres digitales.

2 Objeto

El presente documento describe el procedimiento de la migración de las aplicaciones Web que integren la versión 2.4 del cliente @firma para integrar la versión 3.1 del mismo.

3 Alcance

Este manual se ha realizado tomando como base el que se ha realizado una integración estándar del cliente @firma v2.4, esto es:

- Se hace uso de las bibliotecas JavaScript que incorpora este cliente.
- Se hace uso de los métodos publicados en el Applet.

Los pasos detallados en este manual sirven para adaptar los despliegues existentes de la versión 2.4 del cliente @firma a la nueva versión 3.1 del cliente. Esto es, haciendo uso únicamente de las funcionalidades compatibles con las que cuenta la versión 2.4 del cliente, siendo necesario dirigirse al "Manual del Integrador" de la versión 3.1 del cliente si se desea utilizar alguna de las nuevas funcionalidades que ésta incorpora.

4 Migración al cliente v3.1

4.1 Necesidad de la adaptación

La redacción de este manual viene motivada principalmente el cambio metodológico y de arquitectura que se ha realizado en el despliegue del cliente. Mientras que en la versión 2.4 del cliente y anteriores se instalaban las dependencias del cliente y este debía cargarse en cada ejecución, a partir de la versión 3.0 además de sus dependencias se realiza la instalación del propio cliente. Esto implica que, en los despliegues de la versión 3.1 del cliente, no sea necesario cargar el núcleo del cliente desde el servidor, sino, tan sólo, un Bootloader (que hace las veces de instalador) mucho más ligero y que se encarga de comprobar la correcta instalación del cliente y localizar en donde se encuentra para realizar su carga desde disco. Estos cambios se han encapsulado en las bibliotecas JavaScript que acompañan al cliente para que los integradores se vean mínimamente afectados.

Adicionalmente, la nueva versión del cliente cuenta con una nueva arquitectura que divide sus funcionalidades en 3 construcciones distintas:

- **Construcción LITE**: Soporta firmas PKCS#1, CMS/PKCS#7 y CADES, e incorpora todas las capacidades actuales del cliente (firmas, cifrados, acceso a repositorios…).
- **Construcción MEDIA**: Soporta firmas XMLdSig, XAdES y ODF, más las funcionalidades de la construcción LITE.
- **Construcción COMPLETA**: Soporta firmas PDF, además de disponer de las funcionalidades de la construcción MEDIA.

4.2 Despliegue del cliente

El nuevo cliente @firma se despliega de forma similar a las versiones anteriores. Para sustituir el cliente versión 2.4 por la versión 3.1 hay que seguir los siguientes pasos:

1. Sustituir las bibliotecas JavaScript de la versión 2.4 del cliente por las de la versión 3.1. Durante este proceso, al sustituir el fichero "*constantes.js*", deberemos asegurarnos de que las constantes del nuevo cliente tienen asignadas el mismo valor que el del cliente 2.4.
2. Sustituir los ficheros instalables de la versión antigua del cliente por los de la versión 3.1.
3. Adaptar, si procede, los HTML que cargan el cliente según se explica en el apartado **¡Error! No se encuentra el origen de la referencia.**.
4. Revisar los cambios de procedimiento indicados en el apartado **¡Error! No se encuentra el origen de la referencia.** para asegurarnos de que no afecta a nuestra aplicación. En caso de afectarnos, proceder tal como se indica.

4.3 Adaptación de los HTML que integran el cliente

4.3.1 Importación de librerías JavaScript

Las librerías JavaScript del nuevo cliente @firma poseen los mismos métodos que los de la versión anterior de tal forma que sólo se han realizado cambios internos y se ha respetado por completo la interfaz con el integrador. Sin embargo, a las ya conocidas librerías se ha agregado una adicional "*deployJava.js*", (desarrollada por Sun Microsystems) que implementa el nuevo sistema de despliegue de *Applets* adoptado por los navegadores más extendidos del mercado y mediante el que se despliega el nuevo cliente. Esta biblioteca deberá cargarse en toda página Web en la que ya se esté cargando "*instalador.js*". Esto implica **agregar la siguiente sentencia en la cabecera de los HTML** en los que se desee importar (indicando correctamente la ruta al fichero "*deployJava.js*" mediante el atributo *src*):

```
<script type="text/javascript" language="javascript"
src="common/deployJava.js"></script>
```

Por otro lado, se ha eliminado la librería JavaScript "*runApplet.js*". Además, es recomendable eliminar las esperas explícitas a la carga del *applet*. Esto es, las sentencias que hacen uso de los métodos de la biblioteca *"time.js"* y la variable *clienteFirmaCargado*. Por ejemplo:

```
whenTry("clienteFirmaCargado == true", "clienteFirma.setCipherAlgorithm('" +
cipherAlgorithm + "')", "No se ha podido iniciar el Applet de firma.");
```

Por regla general, el navegador Web no termina la carga de la página hasta que no se finaliza la carga del Cliente, por lo que no suele ser necesario agregar sentencias de este tipo.

4.3.2 Carga del *Applet* de firma

La nueva arquitectura del cliente elimina el sistema de módulos (*plugins*) con el que se contaba hasta ahora (debido a nuevas restricciones de seguridad de la JRE1.6u17 y superiores) y establece 3 construcciones distintas que incorporan diferentes funcionalidades (cada una incorporando las funcionalidades de las anteriores). Esta nueva arquitectura requiere que cada vez que se cargue el *applet* mediante el método *cargarAppletFirma()* se indique la construcción mínima que exija nuestra aplicación para funcionar correctamente. Esto lo haremos pasándole los parámetros 'LITE', 'MEDIA' o 'COMPLETA' al método según sea la construcción que necesitemos. Si no se indica nada, se interpretará que se desea la construcción por defecto, que será la 'LITE' salvo que se indique lo contrario mediante la variable `defaultBuild` del fichero "*constantes.js*". El integrador deberá consultar el listado de funcionalidades incorporado en cada construcción del usuario para indicar cual debe utilizar.

Para cargar el *applet* de firma exigiendo que se disponga de al menos la construcción MEDIA, por ejemplo, usaríamos la sentencia:

```
<script type="text/javascript">
	cargarAppletFirma('MEDIA');
</script>
```

También podríamos establecer la variable `defaultBuild` del fichero "*constantes.js*" con el valor `MEDIA` y hacer:

```
<script type="text/javascript">
	cargarAppletFirma();
</script>
```

4.3.2.1 Localización de la llamada al método de carga del Applet

En la versión 3 del Cliente @firma se ha cambiado el modo de despliegue de los Applets para seguir las últimas recomendaciones de Sun Microsystems al respecto.

Debido a estos cambios, la llamada al método `cargarAppletFirma()` no puede ser realizada dentro de una etiqueta XML, y debe situarse dentro del cuerpo de una sección delimitada por etiquetas. La implicación práctica más directa de esta restricción es que ahora no es posible realizar la llamada de carga en la propiedad `onLoad()` de la etiqueta HTML `<body>`, siendo la opción recomendada situar esta llamada en cualquier lugar entre las etiquetas `<body>` y `</body>`.

Cualquier otra llamada al Applet (comprobar si está instalado, obtener la versión, etc.) sigue pudiéndose invocar desde `onLoad()` o cualquier otro gestor de eventos interno a una etiqueta HTML.

Esto aplica igualmente a cualquier proceso de carga del cliente desde un disparador de evento: `onClick()`, `onMouseOver()`...

4.3.2.2 Despliegue del cliente mediante JNLP

La versión 3.1 del Cliente @firma trata por defecto de cargase vía JNLP cuando el sistema del usuario reúne las condiciones necesarias (Java 6 update 10 o superior con arquitectura de 32 bits). Este modo de despliegue evita que el usuario tenga que instalarse el Cliente en local y es la propia JRE quien lo cachea para evitar descargas posteriores. Este método, sin embargo, requiere que el integrador proporcione información adicional en el momento de realizar el despliegue. Para ello debe modificar los ficheros:

- LITE_afirma.jnlp
- MEDIA_afirma.jnlp
- COMPLETA_afirma.jnlp

La estructura de estos ficheros es la siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<jnlp spec="1.0+" codebase="RUTA_DIRECTORIO_DESPLIEGUE">
<information>
        <title>Cliente @firma</title>
        <vendor>Junta de Andalucía</vendor>
        <homepage href="http://www.juntadeandalucia.es" />
        <description>Cliente de firma @firma</description>
</information>
<offline-allowed />
<security>
        <all-permissions />
</security>
<resources>
        <property name="jnlp.packEnabled" value="true" />
        <j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se" />
        <jar href="COMPLETA_j6_afirma5_coreV3.jar" main="true" />
</resources>
<applet-desc name="Cliente @firma" main-class="es.gob.afirma.cliente.SignApplet" width="1" height="1">
<update check="background" />
</jnlp>
```

El integrador deberá modificar la cadena "**RUTA_DIRECTORIO_DESPLIEGUE**", de cada uno de estos ficheros, por la ruta absoluta al directorio en donde se encuentren las distintas construcciones del Cliente ("*COMPLETA_j6_afirma5_coreV3.jar*", "*LITE_j6_afirma5_coreV3.jar*", etc). Por ejemplo:

```
...
<jnlp spec="1.0+" codebase="http://www.juntadeandalucia.es/afirma/">
...
```

Si el integrador desea que **el Cliente @firma se instale siempre**, puede hacerlo pasando el valor `true` como segundo parámetro del método `cargarAppletFirma()`. Por ejemplo:

```
<script type="text/javascript">
	cargarAppletFirma('MEDIA', true);
</script>
```

Si se desea seguir especificando la construcción por defecto a través de la variable `defaultBuild` del fichero de configuración "*constantes.js*", se puede realizar de la siguiente forma:

```
<script type="text/javascript">
	cargarAppletFirma(defaultBuild, true);
</script>
```

4.4 Cambios en los procedimientos

4.4.1 Ensobrado de datos

En la versión 2.4 y anteriores del cliente, por un error en la implementación, se establecía el texto plano que se deseaba ensobrar mediante el método setData(String). En la nueva versión del cliente, tal como se indicaba en la especificación del método, el texto que se le debe pasar a este método debe estar codificado en base 64. Podemos realizar el paso intermedio de pasar de texto plano a texto en base 64 mediante el método: getBase64FromText(String).

Así obtenemos que:

- **Versión 2.4**: clienteFirma.setData("texto plano");
- **Versión 3.1**:clienteFirma.setData(clienteFirma.getBase64FromText("texto plano"));

4.4.2 Devolución de nulos

Las anteriores versiones del cliente disponían de métodos que debían devolver una cadena de texto y, en caso de error o no disponer de datos para su devolución, devolvían cadena vacía. Esta práctica, que si bien evitaba comprobar que el valor de retorno fuese nulo, llevaban a no poder distinguir cuando la operación había finalizado correctamente o si se habían devuelto datos significativos. La nueva versión del cliente, devuelve nulo en estos métodos en los que puede malinterpretarse el resultado si se devolviese cadena vacía. El comportamiento explicado se refleja en el JavaDoc de la nueva versión del cliente y los métodos afectados son:

- getCipherData()
- getPlainData()
- getData()
- getBase64Data()
- getPassword()
- getSignatureBase64Encoded()
- getSignatureText()

En caso de que utilizar alguno de estos métodos en nuestra aplicación, deberemos consultar que el resultado no sea nulo antes de utilizar el valor devuelto. Por ejemplo:

```
	var plainText = clienteFirma.getPlainData();
	if(plainText == null) {
	alert("No se ha podido recuperar el texto plano");
} else {
	alert(plainText);
	}
```

4.4.3 Configuración del fichero de entrada

Existen una serie de métodos de operación que especifican por parámetro el fichero que se desea procesar, en lugar de tomar el fichero configurado mediante setFileuri(String) o setFileuriBase64Encoded(String). Adicionalmente, estos métodos modificaban la configuración del Cliente de tal forma que los ficheros especificados quedaban establecidos como ficheros de entrada para el resto de operaciones. Los métodos en cuestión son:

- getFileBase64Encoded(String strUri, boolean showProgress)
- cipherFile(String strUri)
- decipherFile(String strUri)

- signAndPackFile(String uri)

En la nueva versión 3.1 del Cliente @firma, los métodos mencionados no alteran la configuración del fichero de entrada establecido en el Cliente.

Por ejemplo, dado el siguiente código:

```
…
clienteFirma.setFileuri("foo.txt");
clienteFirma.getFileBase64Encoded("bar.txt", false);
clienteFirma.sign();
…
```

El Cliente @firma v3.1 firmaría el fichero "foo.txt", mientras que las versiones anteriores firmarían "bar.txt".

5 Restricciones llevadas a cabo en la versión 3.1 del cliente

5.1 Funciones y métodos en la interfaz Applet del cliente @firma v3 eliminados respecto a versiones anteriores

5.1.1 public void signHTML(java.io.InputStream is)

JavaScript no soporta el tipo de datos Java `InputStream`, por lo que su uso desde el cliente es imposible, y exponer la función puede llevar a equívocos o causar un uso inapropiado.

La firma realizada por este método es una firma simple, con la configuración establecida, sobre los datos extraídos del flujo de entrada. Podemos emular su comportamiento siguiendo los siguientes pasos:

1. Leyendo los datos del flujo de entrada en cuestión desde la aplicación que utiliza el cliente.
2. Convirtiendo los datos leídos a Base64.
3. Estableciéndolos como entrada del cliente con el método `setData(String)`.
4. Ejecutando la operación de firma mediante el método `sign()` del cliente.

5.1.2 public byte[] getSignature()

JavaScript no soporta el tipo de datos de Java `byte[]`, por lo que su uso es imposible, y exponer la función puede llevar a equívocos o causar un uso inapropiado.

Para recuperar la información de firma puede utilizarse:

- getSignatureText() para las firmas XML. Obtiene la cadena de texto que representa el XML de firma. Puede obtenerse el mismo resultado que con el método getSignature() utilizando el método getBytes() sobre su salida.
- getSignatureBase64Encoded() para cualquier tipo de firma. Devuelve la firma en forma de cadena en base 64. Puede obtenerse el mismo resultado que con el método getSignature() decodificando la cadena en base 64 obtenida.

5.2 Algoritmos de cifrado del cliente @firma v3 eliminados respecto a versiones anteriores

Se han eliminado los siguientes algoritmos de cifrado, por considerarse obsoletos o en desuso:

- CAST5
- IDEA
- Twofish
- Serpent

Aun cuando haya algoritmos de cifrado que se mantengan desde la versión anterior del cliente, es posible que haya cambiado su configuración por defecto. Por regla general, siempre debería cifrarse con la misma versión del cliente con la que se cifró.

6 Glosario de términos

Firma electrónica

Es el conjunto de datos, en forma electrónica, anejos a otros datos electrónicos o asociados funcionalmente con ellos, utilizados como medio para identificar formalmente al autor o a los autores del documento que la recoge.

XML Digital Signature (XMLDSig)

Es una recomendación del W3C que define una sintaxis XML para la firma digital

XML Advanced Signature (XAdES)

Es un conjunto de extensiones a las recomendaciones XML-DSig haciéndolas adecuadas para la firma electrónica avanzada.

RSA

Es un sistema criptográfico de clave pública desarrollado en 1977. En la actualidad, RSA es el primer y más utilizado algoritmo de este tipo y es válido tanto para cifrar como para firmar digitalmente.

XML

Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

Office Open XML (OOXML)

Es un formato de archivo abierto y estándar cuyas extensiones más comunes son .docx, .xlsx y .pptx. Se le utiliza para representar y almacenar hojas de cálculo, diagramas, presentaciones y documentos de texto. Un archivo Office Open XML contiene principalmente datos basados en el lenguaje de marcado XML, comprimidos en un contenedor .zip específico.

Open Document Format (ODF)

Es un formato de fichero estándar para el almacenamiento de documentos ofimáticos tales como hojas de cálculo, memorandos, gráficas y presentaciones. Aunque las especificaciones fueron inicialmente elaboradas por Sun, el estándar fue desarrollado por el comité técnico para Open Office XML de la organización OASIS y está basado en un esquema XML inicialmente creado e implementado por la suite ofimática OpenOffice.org (ver OpenOffice.org XML).

ZIP

Es un formato de almacenamiento sin pérdida, muy utilizado para la compresión de datos como imágenes, programas o documentos.

PDF

Es un formato de almacenamiento de documentos, desarrollado por la empresa Adobe Systems. Este formato es de tipo compuesto (imagen vectorial, mapa de bits y texto).

SHA

Es un sistema de funciones hash criptográficas relacionadas de la Agencia de Seguridad Nacional de los Estados Unidos y publicadas por el National Institute of Standards and Technology (NIST). El primer miembro de la familia fue publicado en 1993 es oficialmente llamado SHA. Sin embargo, hoy día, no oficialmente se le llama SHA-0 para evitar confusiones con sus sucesores. Dos años más tarde el primer sucesor de SHA fue publicado con el nombre de SHA-1. Existen cuatro variantes más que se han publicado desde entonces cuyas diferencias se basan en un diseño algo modificado y rangos de salida incrementados: SHA-224, SHA-256, SHA-384, y SHA-512 (llamándose SHA-2 a todos ellos).

PKCS

Se refiere a un grupo de estándares de criptografía de clave pública concebidos y publicados por los laboratorios de RSA en California. A RSA Security se le asignaron los derechos de licenciamiento para la patente de algoritmo de clave asimétrica RSA y adquirió los derechos de licenciamiento para muchas otras patentes de claves.

W3C

Es un consorcio internacional que produce recomendaciones para la World Wide Web. Está dirigida por Tim Berners-Lee, el creador original de URL (Uniform Resource Locator, Localizador Uniforme de Recursos), HTTP (HyperText Transfer Protocol, Protocolo de Transferencia de HiperTexto) y HTML (Lenguaje de Marcado de HiperTexto) que son las principales tecnologías sobre las que se basa la Web.

OpenOffice.org

Es una suite ofimática libre (código abierto y distribución gratuita) que incluye herramientas como procesador de textos, hoja de cálculo, presentaciones, herramientas para el dibujo vectorial y base de datos. Está disponible para varias plataformas, tales como Microsoft Windows, GNU/Linux, BSD, Solaris y Mac OS X. Soporta numerosos formatos de archivo, incluyendo como predeterminado el formato estándar ISO/IEC OpenDocument (ODF), entre otros formatos comunes. A febrero de 2010, OpenOffice soporta más de 110 idiomas.

Base64

Es un sistema de numeración posicional que usa 64 como base. Es la mayor potencia de dos que puede ser representada usando únicamente los caracteres imprimibles de ASCII. Esto ha propiciado su uso para codificación de correos electrónicos, PGP y otras aplicaciones. Todas las variantes famosas que se conocen con el nombre de Base64 usan el rango de caracteres A-Z, a-z y 0-9 en este orden para los primeros 62 dígitos, pero los símbolos escogidos para los últimos dos dígitos varían considerablemente de unas a otras. Otros métodos de codificación como UUEncode y las últimas versiones de binhex usan

un conjunto diferente de 64 caracteres para representar 6 dígitos binarios, pero éstos nunca son llamados Base64.

ASN.1

Es una norma para representar datos independientemente de la máquina que se esté usando y sus formas de representación internas. Es un protocolo de nivel de presentación en el modelo OSI.

Autoridad de Certificación (CA)

Es una entidad de confianza, responsable de emitir y revocar los certificados digitales o certificados, utilizados en la firma electrónica, para lo cual se emplea la criptografía de clave pública. Jurídicamente es un caso particular de Prestador de Servicios de Certificación.

Certificado Digital

Es un documento digital mediante el cual un tercero confiable (una autoridad de certificación) garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública.

Infraestructura de Clave Pública (PKI)

Es una combinación de hardware y software, políticas y procedimientos de seguridad que permiten la ejecución con garantías de operaciones criptográficas como el cifrado, la firma digital o el no repudio de transacciones electrónicas.

7 Información de contacto

Soporte a la Administración Electrónica
Consejería de Hacienda y Administración Pública
Dirección General de Tecnologías para Hacienda y la Administración Electrónica
soporte.admonelectronica@juntadeandalucia.es