Manual del Integrador del

Cliente de Firma de @Firma 5

(Versión 2.3.5)

Documento nº:	TI-20-1178-@Firma-Global-MICF-008
Revisión:	008
Fecha:	10-12-2007
Período de retención:	Permanente durante su período de vigencia + 3 años después de su anulación

CONTROL DE COMPROBACIÓN Y APROBACIÓN

Documento nº: TI-20-1178-@Firma-Global-MICF-005

Revisión: 008

Fecha: 10/12/2007

REALIZADO

23/06/2006

Julio

Pedreira

Paz

Analista Programador

10/12/2007

Manuel

Reyes

Cayetano

Analista Programador

COMPROBADO

10-12-2007

Rafael Carlos

Vázquez

Carmona

Director @firma

APROBADO

10-12-2007

Rafael Carlos

Vázquez

Carmona

Director @firma

CONTROL DE MODIFICACIONES

Documento nº: TI-20-1178-@Firma-Global-MICF-008

Revisión: 008

Fecha: 10-12-2007

Rev. 001

Fecha 14/06/2006

Autor/es JPP

Descripción Documentación inicial

Rev. 002

Fecha 23/06/2006

Autor/es JPP

Descripción Actualización de la documentación para ajustarla a la 2ª entrega del Cliente (v 1.1)

- Documentación del nuevo operador de selección de certificados #NOT_MATCHES#
- Cambios en la forma de integración de la firma web (se firma a un fichero, para permitir adjuntos mayores)

Añadida documentación sobre el nuevo *plug-in XAdES* (v 1.0) para el Cliente

Actualizada la documentación para incluir los nuevos ficheros de ejemplo

Rev. 003

Fecha 24/11/2006

Autor/es MRC

Descripción Actualización de la documentación.

Añadido módulo de cifrado y sobre digital.

Actualización de los ejemplos.

Rev.	004
Fecha	05/01/2007
Autor/es	MRC
Descripción	Modificación del instalador y sistema de plugins

Rev.	005
Fecha	18/05/2007
Autor/es	MRC
Descripción	Actualización del documento.

Rev.	006
Fecha	20/06/2007
Autor/es	MAV
Descripción	Añadida documentación sobre las opciones de visualización.

Rev.	6.0
Fecha	23/08/2007
Autor/es	MAV
Descripción	Liberación de versión 2.3.1 para corregir un bug relativo al filtrado de los certificados.

Rev.	6.1
Fecha	10/07/2007
Autor/es	RCVC
Descripción	Liberación de versión 2.3.3 con adaptación de la DLL y el cliente a formatos propietarios de tarjetas Siemens

Rev.	007
Fecha	25/06/2007

Autor/es	MRC
Descripción	Incorporación del formato de firma CADES-BES

Rev.	008
Fecha	10/12/2007
Autor/es	MRC
Descripción	Ampliado plugin XAdES: - Incorporación de formatos de firma enveloping y enveloped. - Incorporados modos de firma implícita y explícita. - Mejorado de procesado de DOM. Incorporación de descifrado de sobres digitales en Internet Explorer y Firefox. Mejorada la accesibilidad a librerías dinámicas. Creación de punto de instalación único (directorio personal). Funcionalidad de distribuciones Añadidos anexos con matriz de compatibilidad del cliente de firma y errores conocidos

CONTROL DE DISTRIBUCIÓN

Documento nº: TI-20-1178-@Firma-Global-MICF-008

Revisión: 008

Fecha: 10-12-2007

Copias Electrónicas:

La distribución de este documento ha sido controlada a través del sistema de información.

Copias en Papel:

La vigencia de las copias impresas en papel está condicionada a la coincidencia de su estado de revisión con el que aparece en el sistema electrónico de distribución de documentos.

El control de distribución de copias en papel para su uso en proyectos u otras aplicaciones es responsabilidad de los usuarios del sistema electrónico de información.

Fecha de impresión 26/05/2008 1:44

Distribución en Papel:

Nombre o Cargo y (Organización)	Nº de Ejemplares	Referencia de la carta de transmisión y fecha

Índice

1 Objeto

El objeto de este documento es servir de guía al integrador del Cliente de Firma de la plataforma de validación y Firma electrónica, @Firma 5.

2 Alcance

El presente documento detalla el Cliente de Firma de Arroba Firma 5 y abarca los siguientes aspectos:

- Requisitos del Cliente
    - Navegadores soportados
    - Sistemas operativos soportados
    - Otros requistos
- Componentes del Cliente
    - Applet instalador
    - Applet de firma
    - *Plugin XAdES*
- Funcionalidad del Instalador
    - Comprobación si el Cliente y/o el *plugin* están instalados
    - Instalar el Cliente y/o el *plugin*
    - Desinstalar
    - Ruta de instalación
    - Comprobar si el Cliente y/o el *plugin* están actualizados
    - Actualizar el Cliente/*plugin*
- Funcionalidad del Cliente:
    - Firma Web
    - Firma
    - Firma Masiva
    - Co-firma
    - Multi-Firma

  - Cifrado
  - Descifrado
  - CMS encriptado
  - CMS sobre digital
  - CMS 'firmar y envolver'
- Configuración del cliente:
  - algoritmos y formatos
  - selección de certificados
- Otras funcionalidades
- Ejemplos que abarquen los aspectos anteriormente descritos.

3 Siglas

DNI-e	DNI electrónico
DOM	*Document Object Model* (Modelo de objetos en documentos)
JAR	*Java Archive*
JRE	*Java Runtime Environment*
MAP	Ministerio de Administraciones Públicas
PKI	*Public Key Infrastructure*
Plugin	(o *AddOn*) Módulo que se puede enchufar en una aplicación para ampliar su funcionalidad
TI	Telvent Interactiva
URI	*Uniform Resource Identifier* (Identificador Uniforme de Recursos). También se usa URL
URL	*Uniform Resource Locator* (Localizador Uniforme de Recursos). También se usa URI.
WYSIWYS	*What You See Is What You Sign* (lo que ves es lo que firmas)
XAdES	*XML Advanced Electronic Signature* (firma electrónica avanzada XML)

4 Documentos de Referencia

[JAVADOC] Documentación de los métodos públicos de los applets Instalador y de Firma en la carpeta *javadoc*.

5 Introducción

El Cliente de Firma es una herramienta de Firma Electrónica basada en Applets Java integrados en páginas web mediante JavaScript, que permiten una fácil implantación en aplicaciones web ya existentes o que se vayan a desarrollar.

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 (*keystore*) del navegador web (*Internet Explorer, Mozilla, Firefox*) así como de los que estén en dispositivos (*SmartCards, USBKey*) 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, usbtoken o navegador) ubicado en su PC. De hecho, nunca llega a salir del navegador, sino que 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 Formularios Web (con y sin adjuntos)
- Firma Ficheros binarios
- Firma Masiva ficheros binarios
- Co-firma (CoSign) → Multifirma al mismo nivel.
- Contra-Firma (CounterSign) → Multifirma en cascada.

Como complemento al cliente de firma e incluido en el mismo applet, 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 CMS.

El Cliente se distribuye con un instalador. El instalador permite comprobar si el Cliente o el *plugin* XAdES ya han sido instalados y actualizados. En caso de no haber sido instalados/actualizados permite instalar/actualizar el Cliente/*plugin* en local. En caso de que ya hayan sido instalados permite invocarlos sin descargarlos de nuevo del servidor. Éste mecanismo hace que sólo sea necesario descargar el Cliente/*plugin* una vez. Las demás veces se ejecutará desde la propia máquina del usuario.

6 Requisitos mínimos

- Sistema Operativo
    - Windows 2000 / XP
    - Linux (Guadalinex, Ubuntu)

- Navegador web:
    - Firefox 1.5 o superior
    - Mozilla 1.7 o superior
    - Internet Explorer 5.5 o superior

- JRE 1.4.2 o superior (instalada en el navegador)

- Certificado digital de usuario instalado en el navegador o disponible a través de un módulo PKCS#11 instalado en el navegador (caso del DNI-e)

7 Componentes del Cliente

7.1 Instalador

El instalador se compone de los siguientes ficheros:

- **instaladorClienteFirmaAFirma5.jar**: Contiene los ficheros binarios para la ejecución del instalador (las clases o ficheros *.class*)

- **clienteFirmaAFirma.jar**: Correspondiente a los binarios que componen el cliente de firma.

- **clienteFirmaAFirma5.zip**: Contiene los ficheros binarios y librerías necesarios para la ejecución del Cliente de Firma en sí (ficheros *.jar*). El instalador los descomprime en una carpeta llamada .clienteFirmaArrobaFirma5 en el directorio de usuario para poder ejecutarlos en local.

- **linuxLibraries.zip** y **win32libraries.zip:** Contienen librerías del sistema necesarias para la ejecución del Cliente de Firma en sí. Son dependientes del sistema operativo. El instalador detecta el sistema operativo del usuario y descomprime las librerías del fichero adecuado en un directorio dependiente del sistema operativo (en la carpeta de *Windows* o en un directorio del *LD_LIBRARY_PATH* en *Linux*).

- **xades-plugin.zip:** Contiene los ficheros binarios necesarios para permitir al cliente firmar en formato *XAdES*. El instalador lo descomprime en la carpeta del resto del Cliente, para permitir su ejecución en local.

- **instalador.js** (y otros ficheros JavaScript): Contienen funciones JavaScript para automatizar los procesos del instalador. Actúan como librería y aunque su uso es recomendado para el integrador, no es estrictamente necesario.

7.2 Cliente

El cliente se compone de:

- **Clases** de la aplicación, agrupadas en ficheros *.jar* almacenados (por el Instalador) en un directorio de usuario.

- **Librerías** de sistema (*.dll* en *Windows*, *.so* y *.bin* en *Linux*) almacenadas (por el Instalador) en un directorio de usuario.

- **Ficheros JavaScript** (firma.js, firmaWeb.js y otros): Contienen funciones para la automatización de los procesos de firma. Almacenadas en el servidor web y modificables por los integradores. Son opcionales y se puede operar sin ellas, pero facilitan los procesos más comunes.

- **Fichero de configuración** (version.properties): Contiene información sobre la versión y los *plugins* instalados.

7.2.1 Notas sobre la versión

Debido a las modificaciones realizadas en ésta versión se han de tener en cuenta los siguientes puntos ante posibles problemas en la instanciación del cliente.

1. Deberá desinstalarse completamente cualquier cliente previo para que el funcionamiento sea el correcto. Básicamente el problema reside en que la DLL correspondiente a nuestra librería ha sido modificada para incorporarle la funcionalidad de descifrado de sobre digital en IE, pero también se ha modificado el lugar donde se instala para facilitar la instalación en entornos con privilegios limitados. Es decir, el nuevo cliente sólo instala archivos en el directorio especificado (*%user_home%/.clienteFirmaArrobaFirma5* por defecto). No obstante, la máquina virtual sigue buscando antes la librería en windows que en nuestro directorio, por lo que si no se borra las DLLs de Windows dará un error UnsatisfiedLinkError al descifrar un sobre digital. **Habrá por lo tanto que borrar aunque sea manualmente la librería Afirma5Library.dll de Windows**. Si se registra algún error en Internet Explorer se recomienda comprobar si la librería cargada es la correcta y se está cargando del directorio de usuario.

2. Por otro lado, y para facilitar la factorización de librerías en firefox (aislar casos de multinstanciación) se ha modificado la clase CryptoManager e incluido en la distribución del cliente. Está configurado para que el classpath cargue antes la clase modificada que la original, sin embargo podemos encontrar problemas si se ha instalado en la máquina el cliente de @firma4. Éste requería que en el directorio *%java_home%/lib/ext* se copiara la librería jss33.jar, la cual se cargará antes que nuestro applet por lo que no serviría la modificación efectuada y lanzará una excepción de NoSuchMethodException. **Esta librería deberá ser borrada**. Como norma general se recomienda no copiar librerías ajenas a la JRE a los directorios de ésta, ya que puede dar problemas frente a cambios de versión. En este caso particular, si se intenta cargar el cliente frente a una jss33 distinta, se producirá una excepción de seguridad.

7.3 El *plugin* XAdES

El *plugin* XAdES v2.3 se compone de diversas librerías Java (.jar) que serán almacenados por el instalador en un directorio de usuario. Para la instalación de este plugin, el fichero *version.properties* del servidor deberá estar convenientemente configurado mediante las siguientes líneas:

```
plugins.names=XAdES
plugins.XAdES.version.mayor=2
plugins.XAdES.version.minor=4
plugins.XAdES.version.build=0
plugins.XAdES.jars=xom.jar, XMLSIGtools.jar, xades-plugin.jar
plugins.XAdES.install.jars=xades-plugin.zip
plugins.XAdES.class=com.telventi.afirma.cliente.signatureformat.XAdESSignatureFormat
```

El plugin versión 2.3 añade ciertos formatos al cliente de firma, por lo que requiere que éste tenga como mínimo la versión 2.3.5. Los formatos que incorpora son los siguientes:

1. XADES_DETACHED y XMLDSIG_DETACHED. Equivalente al comportamiento de la versión anterior del plugin, por lo tanto se establecen como formatos de firma XML por defecto, es decir, es equivalente a indicar al cliente que el formato va a ser XADES o XMLDSIG. Acepta los modos explícito e implícito.

2. XADES_ENVELOPING y XMLDSIG_DETACHED. Genera una firma enveloping sobre los datos indicados. Acepta los modos explícito e implícito.

3. XADES_ENVELOPED y XMLDSIG_ENVELOPED. Genera una firma enveloped del formato indicado. Por requisitos de éste tipo de firma electrónica, la entrada deberá ser XML y no aceptará modo de firma explícito.

Posteriormente se tratarán los distintos formatos de firma en cuanto a su estructura y composición.

El comportamiento de los modos implícito y explícito es una aproximación al comportamiento de éstos en firmas con cuerpo ASN1 (CMS, CAdES, etc…). Debido a que los datos en una firma XML deben estar referenciados mediante una URI, si se indica que los datos no se incluyan en la firma (modo explícito) lo que incluirá la firma electrónica es el hash de los datos firmados y una referencia a éste, mientras que si se utiliza el modo implícito se incorporará los datos completos. El modo explícito sólo está disponible para los formatos detached y enveloping por restricciones del formato enveloped.

El plugin intentará identificar el *mimetype* de los datos firmados, no obstante si esto no fuese posible le asignará un tipo por defecto (*application/octet-string*).

8 El Instalador

Para poder ejecutar el Instalador desde una página web, hay que cargar el Applet Instalador. En HTML esto se hace mediante las etiquetas *APPLET* (obsoleta) u *OBJECT*. Mediante una de dichas etiquetas se ha de cargar la clase *com.telventi.afirma.cliente.instalador.InstaladorCliente.class* del fichero *instaladorClienteFirmaAFirma5.jar* pasándole como parámetros *userAgent* y *appName* los valores de *window.navigator.userAgent* y *window.navigator.userAgent* respectivamente.

En el fichero ***instalador.js*** (que depende de ***time.js*** y de ***appletHelper.js***) hay una función llamada ***cargarAppletInstalador*** que realiza todas estas operaciones de forma automática. Puede recibir como parámetro la ruta al fichero *instaladorClienteFirmaAFirma5.jar* si no está en la misma que la página web que lo invoca. Existe un fichero llamado ***constantes.js*** en el cual se definen algunos parámetros, los cuales se usarán en los distintos ficheros JavaScript, como el citado para la ruta base del applet.

Esta función carga el Instalador de forma invisible en una variable JavaScript llamada **instalador**. El contenido de dicha variable es válido cuando el valor de la variable ***instaladorCargado*** es verdadero (*true*). Esto es así porque la carga de los Applet se realiza de forma asíncrona, y hasta que está cargado sus métodos no son accesibles. Por esta razón, es aconsejable cargarlo en el método ***onload*** del ***body***. No obstante, y como veremos posteriormente, existe una función llamada ***cargarAppletFirma*** que si detecta que el instalador no está cargado lo carga automáticamente, por lo que no sería necesario usar las dos, sino únicamente ésta última.

Usando los mencionados ficheros JavaScript, una página web que quisiera hacer uso del instalador podría ser:

```
<html>
  <head>
    <script type="text/javascript" language="javascript" src="constantes.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/time.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/appletHelper.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/instalador.js"></script>

    […]

  </head>

  <body onload="cargarAppletInstalador()">

  […]

  </body>
</html>
```

8.1 Comprobar si el Cliente/*plugin* está instalado

Para comprobar si el Cliente/*plugin* ya está instalado, el instalador tiene un método llamado ***isInstalado***que devuelve verdadero o falso (*true* o *false*) en función de si el Cliente ya está instalado en local. Si no se le pasa parámetro, se refiere al cliente. Si como parámetro se le pasa la cadena xades, se referirá al *plugin XAdES*.

Cómo ya se ha dicho, este método no puede invocarse antes de que el Instalador termine de cargarse. En el fichero JavaScript ***instalador.js*** se incluye una función llamada ***isInstalado*** que espera a que esto suceda antes de invocarlo.

Por ejemplo:

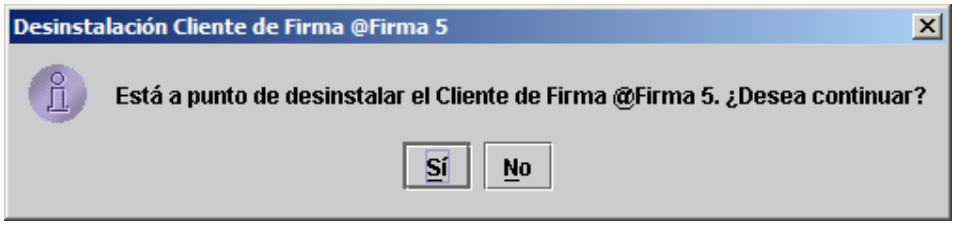
```
<html>
  <head>
    <script type="text/javascript" language="javascript" src="constantes.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/time.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/appletHelper.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/instalador.js"></script>

    […]

  </head>

  <body onload="cargarAppletInstalador()">

  […]
    <a href="#" onclick="alert('Cliente instalado:' + isInstalado()); return false;">
    Comprobar instalación cliente (true=correcta, false=no instalado)
    </a>
    <a href="#" onclick="alert('Plugin XAdES instalado:' + isInstalado('xades')); return false;">
    Comprobar instalación plugin XAdES (true=correcta, false=no instalado)
    </a>
    <br>
  […]

  </body>
</html>
```

8.2 Instalar el Cliente/*plugin*

Para instalar el Cliente o el *plugin XAdES*, el instalador tiene un método llamado ***instalar***. Si lo que se quiere instalar es el Cliente, no se le pasa parámetro. Si se quiere instalar el *plugin,* hay que pasarle la cadena 'xades' como parámetro

Este método realiza las siguientes operaciones:

1. Solicita permiso al usuario para proceder a la instalación del Cliente / *plugin*



2. Comprueba si el Cliente/*plugin* ya está instalado, en cuyo caso advierte al usuario y le permite sobrescribir o cancelar la instalación

3. Instala las clases y las librerías del Cliente/*plugin* en la máquina del usuario. Al finalizar la instalación puede ser necesario reiniciar el navegador.

Cómo ya se ha dicho, este método no puede invocarse antes de que el Instalador termine de cargarse. En el fichero JavaScript ***instalador.js*** se incluye una función llamada ***instalar*** que espera a que esto suceda antes de invocarlo.

Por ejemplo:

```
<html>
  <head>
    <script type="text/javascript" language="javascript" src="constantes.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/time.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/appletHelper.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/instalador.js"></script>
    […]

  </head>

  <body onload="cargarAppletInstalador()">

  […]
    <a href="#" onclick="instalar(); return false;">
    Instalar el cliente
    </a>
    <a href="#" onclick="instalar('xades'); return false;">
    Instalar el plugin XAdES
    </a>
    <br>
  […]
  </body>

</html>
```

8.3 Actualizar el Cliente

Para actualizar el Cliente o el *plugin XAdES*, el instalador tiene un método llamado ***actualizar***. Si lo que se quiere actualizar es el Cliente, no se le pasa parámetro. Si se quiere actualizar el *plugin,* hay que pasarle la cadena 'xades' como parámetro

Este método realiza las siguientes operaciones:

1. Solicita permiso al usuario para la actualización:

2. Sobrescribe los ficheros en el directorio del Cliente con los ficheros actualizado

Como en casos anteriores, existe una función *JavaScript* en el fichero **instalador.js** que se asegura de que el instalador esté cargado. Esta función se llamama **actualizar**.

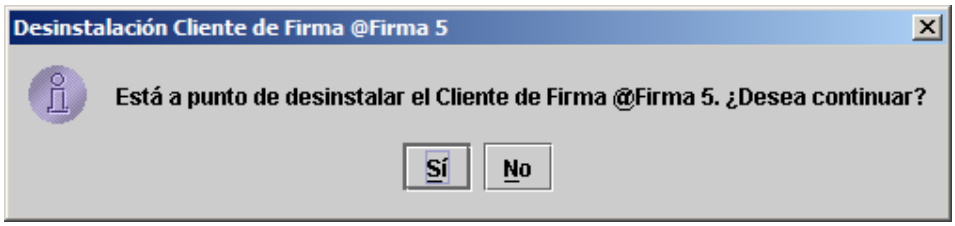
Por ejemplo:

```
<html>
  <head>
    <script type="text/javascript" language="javascript" src="constantes.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/time.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/appletHelper.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/instalador.js"></script>

    […]

  </head>

  <body onload="cargarAppletInstalador()">

  […]
    <a href="#" onclick="actualizar(); return false;">
    Actualizar el cliente
    </a>
    <br>
  […]

  </body>
</html>
```

8.4 Desinstalar el Cliente

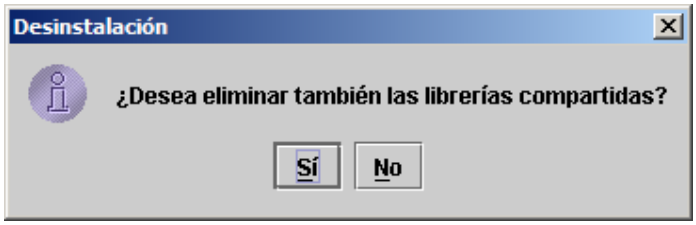
Para desinstalar el Cliente, el instalador tiene un método llamado ***desinstalar***. Este método realiza las siguientes operaciones:

1. Solicita permiso al usuario para proceder a la des-instalación del Cliente



2. Elimina las clases del Cliente

3. Solicita permiso al usuario para eliminar las librerías compartidas



Cómo ya se ha dicho, este método no puede invocarse antes de que el Instalador termine de cargarse. En el fichero JavaScript ***instalador.js*** se incluye una función llamada ***desinstalar*** que espera a que esto suceda antes de invocarlo.

Por ejemplo:

```
<html>
  <head>
    <script type="text/javascript" language="javascript" src="constantes.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/time.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/appletHelper.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/instalador.js"></script>

    […]

  </head>

  <body onload="cargarAppletInstalador()">

  […]
    <a href="#" onclick="desinstalar(); return false;">
    Desinstalar el cliente
    </a>
    <br>
  […]

  </body>

</html>
```

8.5 Obtener el directorio de instalación del Cliente

Para obtener el directorio de instalación del Cliente, el instalador tiene un método llamado ***getInstallationDirectory*** que devuelve una cadena con la ruta al directorio de instalación. Debido a mejoras introducidas en la versión 2.3.5 del cliente, la instalación se realizará sólo en un directorio de usuario, liberando las restricciones frente a cuentas de usuario con permisos restringidos sobre directorios de sistema.

Es posible configurar el directorio de instalación modificando la variable *installDirectory* en el fichero ***constantes.js***. En este fichero además se incluye una variable (*distinctDistroDir*) para la configuración e instalación de distintas distribuciones del cliente. Estas distribuciones se crearán como directorios diferenciados dentro del directorio de instalación por lo que permite tener varias versiones del cliente coexistiendo dentro de la misma máquina.

En el fichero JavaScript ***instalador.js*** se incluye una función llamada ***getDirectorioInstalacion*** que espera a que esto suceda antes de invocarlo.

```
<html>
  <head>
    <script type="text/javascript" language="javascript" src="common-js/time.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/appletHelper.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/instalador.js"></script>

    […]

  </head>

  <body onload="cargarAppletInstalador()">

  […]
    <a href="#" onclick="alert('Directorio de instalación: '+getDirectorioInstalacion(); return
false;">
    Directorio de instalación del cliente
    </a>
    <br>
  […]

  </body>
</html>
```

9 Cliente de Firma

9.1 Cargar el Cliente

Para cargar el Cliente de Firma en una página WEB debemos invocar la función JavaScript *cargarAppletFirma* incluida en el fichero *instalador.js*. Esta función:

1. Carga el applet Instalador
2. Espera a que esté cargado
3. Comprueba si el Cliente está instalado en local
    a. Si no lo está, lo instala según se ha descrito
4. Carga el applet de firma (Cliente)

Cómo ya se ha comentado, la carga de los applets es asíncrona y hay que esperar a que finalice. En este caso será cuando la variable *clienteFirmaCargado* contenga verdadero (*true*).

Si se quiere que se cargue en segundo plano mientras el usuario lee y rellena la página, se puede invocar en el método *onload* de *body*. Por ejemplo:

```
<html>
  <head>
    <script type="text/javascript" language="javascript" src="constantes.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/time.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/appletHelper.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/instalador.js"></script>

    […]

  </head>

  <body onload="cargarAppletFirma()">

  […]

  </body>
</html>
```

9.2 Inicio del proceso de firma

Antes de iniciar un proceso de firma se debe invocar el método *initialize* del Cliente, que borra las entradas y salidas de anteriores procesos de firma.

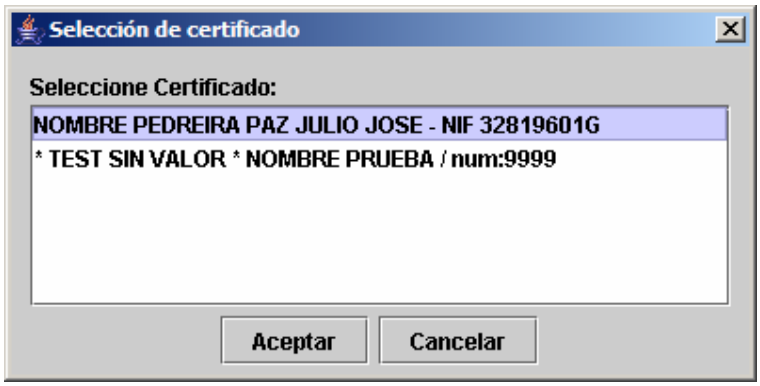
En el fichero JavaScript *firma.js* se incluye un método que lo invoca después de cargar y esperar a que esté cargado el Cliente, si hace falta.

9.3 Firma WEB

9.3.1 Qué es la firma web

En el proceso de firma web una parte de una página web (como un formulario o la página entera) puede ser firmada digitalmente. Para ello,

1. Se compone un HTML por medio de JavaScript
2. Se muestran al usuario los datos a firmar
3. Se le solicita permiso para firmarlo
4. Se le pide al usuario que seleccione un certificado con el que firmar



5. Se solicita (si es necesario) la contraseña para acceder a la clave privada del certificado

6. Se firma el HTML generado

9.3.2 Qué puede firmar el componente firma web

Se puede firmar digitalmente cualquier elemento de un documento HTML (o el documento mismo).

En los campos modificables por el usuario, se firman los valores seleccionados por el mismo. Esto incluye también adjuntos, que son firmados por el Cliente. A la hora de firmarse, se muestra al usuario la página web resultante que le permite verificar lo que realmente va a firmar.

La firma web del cliente sigue la política WYSIWYS (What You See Is What You Sign), es decir, lo que el usuario ve es lo que firma.

9.3.3 Qué no firma el Cliente en la firma web

El cliente no firma las imágenes. Esto hay que tenerlo en cuenta a la hora de diseñar la parte del documento que se ha de firmar, pues es la que se mostrará al usuario (sin imágenes).

El cliente recoge todos los estilos *CSS* del documento que se definan mediante *LINK* o *STYLE*. Sin embargo, aquellas hojas de estilo que no se enlacen directamente (sino que se importen mediante la directiva *@import*) no se incluirán en el HTML si el usuario utiliza Firefox. Por lo tanto, los estilos necesarios para mostrar correctamente la parte a firmar deben ir incluidos mediante *STYLE* o referenciados directamente mediante *LINK*, nunca mediante *import*.

9.3.4 Cómo hacer una firma web

Para hacer una firma web, se puede pasar un HTML al Cliente para que muestre al usuario y éste decida si firmarlo mediante el método ***webSign*** del Cliente. Este método recibe una cadena HTML como parámetro.

Este método:

1. Muestra el documento al usuario. Se visualizará la página html indicada tal y como se ha especificado. Esto quiere decir que en el visualizador los campos aparecerán habilitados si es así como estaban definidos en el documento original. Así pues, cualquier modificación de los datos contenidos en esta ventana se verá reflejada en la posterior firma.

2. Solicita permiso para firmar el documento mostrado.

3. Muestra los certificados disponibles para firmarlo al usuario, y le solicita que elija uno (a no ser que se hayan restringido los certificados válidos mediante el método ***setCertFilter*** o se haya especificado uno concreto con el método ***setMandatoryCertificateCondition***; ambos métodos se verán en el apartado de configuración del cliente)

4. En caso de que esté protegido por contraseña se la solicita al usuario

5. Firma el documento

Una vez invocado:

1. Si el método ***isError*** del cliente devuelve ***false***

    a. El valor devuelto (por la función JavaScript **firmaWeb** o el método del Cliente **webSign**) es la ruta de un fichero (local) que contiene la firma del HTML [en el formato por defecto: CMS (implícito, es decir, que no contiene los datos firmados) y con los algoritmos por defecto: RSA y SHA1 codificada en base 64. En el apartado relativo a la configuración del Cliente se verá cómo cambiar estos parámetros. El contenido del fichero puede ser leído como texto (p. ej. firma XAdES) con el método *getTextFileContent* del cliente o, si es binario (p. ej. firma XAdES), codificado en base 64 con el método *getFileBase64Encoded* del cliente. Ambas funciones están descritas en el apartado Otras funcionalidades de

este documento. La comunicación con el servidor de firma queda relegada a la aplicación donde se integra el cliente, así pues, el encargado de crear un método para enviar los ficheros devueltos por el cliente de firma a el servidor es el propio integrador, ya que el cliente de firma en ningún momento se conecta con el servidor de firma, es un proceso independiente.

2. Si el método ***isError*** devuelve ***true***

   a. El método ***getErrorMessage*** devuelve una cadena con el mensaje de error

   b. En este caso el error se habrá mostrado ya al usuario, a no ser que se haya invocado el método ***setShowErrors*** con el parámetro ***false*** que hace que el Cliente no informe por su cuenta de los errores al usuario

Por ejemplo:

```
clienteFirma.setShowErrors(false);
clienteFirma.initialize();
var rutaFicheroFirma= clienteFirma.webSign(html);
if(!clienteFirma.isError())
{
	document.body.formulario.inputFicheroFirma.value= rutaFicheroFirma;
}
else
{
	alert("Se ha producido un error: "+clienteFirma.getErrorMessage());
}
```

Se provee una función JavaScript llamada ***firmaWeb*** en el fichero ***firmaWeb.js*** que recibe como parámetros un elemento HTML y un documento HTML y compone un HTML y lo firma como se ha descrito (y devuelve la ruta al fichero que contiene la firma). En resumen, este método genera un HTML a partir de un elemento HTML con los valores actuales de los campos, incluyendo el contenido de los ficheros, y lo firma. Esto ahorra al integrador generar un HTML *ad-hoc* con los datos introducidos por el usuario para su firma web. El fichero ***firmaWeb.js*** depende otros, como vemos a continuación en un ejemplo práctico:

```
<script type="text/javascript" language="javascript" src="constantes.js"></script>
<script type="text/javascript" language="javascript" src="../common-js/time.js"></script>
<script type="text/javascript" language="javascript" src="../common-js/appletHelper.js"></script>
<script type="text/javascript" language="javascript" src="../common-js/instalador.js"></script>
<script type="text/javascript" language="javascript" src="../common-js/firma.js"></script>
<script type="text/javascript" language="javascript" src="../common-js/htmlEscape.js"></script>
<script type="text/javascript" language="javascript" src="../common-js/utils.js"></script>
<script type="text/javascript" language="javascript" src="../common-js/styles.js"></script>
<script type="text/javascript" language="javascript" src="../common-js/firmaWeb.js"></script>

[…]

<script type="text/javascript" language="javascript">
	function enviar()
	{
		clienteFirma.initialize();
		clienteFirma.setShowErrors(false);

		var formulario = document.getElementById("formulario");
		var ruta = webSign(formulario, document);
		if(!clienteFirma.isError())
		{
			var fichero = document.getElementById("fichero");
```

```
                        fichero.value = ruta;
                        return true; // Enviar
                }
                else
                {
                        alert("No se ha podido firmar: "+clienteFirma.getErrorMessage());
                        return false;
                }
        }
</script>

[…]

<form id="formulario" action="/enviarFirma">
        <input type="text" id="fichero" style="visibility: hidden; display: none;" value=>
        DNI: <input type="text"><br>
        <input type="submit" onclick="return enviar();">
</form>
```

9.4 Firma electrónica

El proceso de firma electrónica es análogo al de firma web, con la diferencia que los datos no tienen por qué ser HTML. El proceso varía pues, fundamentalmente, en la entrada de datos.

Se permiten diferentes tipos de datos a firmar (solo se puede firmar un tipo cada vez):

- **un fichero**: se establece qué fichero firmar mediante el método ***setFileuri***, que recibe como parámetro de entrada una cadena con la ruta al fichero a firmar

- **datos codificados en base 64**: se establecen mediante el método ***setData***, que recibe una cadena con los datos codificados en base 64.

- **Un *hash* codificado en base 64:** se establece mediante el método ***setHash***, que recibe un *hash* codificado en base 64 como parámetro

- Si no se invoca ninguno de estos métodos, el Cliente solicitará al usuario un fichero para firmar

El método que se ha de invocar para firmar es ***sign***, aunque también se puede llamar a la función JavaScript ***firmar*** (en ***firma.js***) que, como en los casos anteriores, espera si es necesario a que el cliente esté cargado y lo instala si es necesario.

Por ejemplo:

```
<script type="text/javascript" language="javascript">

function enviar()    {

                    var fichero= document.getElementById("fichero");
```

```
                clienteFirma.initialize();
                clienteFirma.setFileuri(fichero.value);
                clienteFirma.setShowErrors(false);
                firmar();

                if(!clienteFirma.isError())
                {
                        var firmaB64 = document.getElementById("firmaB64");
                        firmaB64.value = clienteFirma.getSignatureBase64Encoded();
                        return true; // Enviar
                }
                else
                {
                        alert("No se ha podido firmar: "+clienteFirma.getErrorMessage());
                        return false;
                }
        }
</script>

[…]

<form id="formulario" action="/enviarFirma">
        <input type="hidden" id="firmaB64"><br>
        Fichero a firmar: <input type="file" id="fichero">
        <input type="submit" onclick="return enviar();">
</form>
```

9.5 Firma masiva

El caso de la firma masiva parecido al anterior. La firma masiva sólo permite firmar a partir de *hashes* codificados en base 64. Para ello se invoca las veces que sea necesario el método ***addMassiveHash*** que recibe una cadena con un *hash* codificado en base 64 como parámetro.

El caso de error (***clienteFirma.isError()==true***), es como los anteriores. En caso de que no haya error, las firmas se recogen mediante el método ***getSignaturesBase64Encoded***, que devuelve las firmas de los *hashes* codificadas en base 64, en el mismo orden que los *hashes* correspondientes y separadas entre sí por el signo de admiración (***!***).

Por ejemplo:

```
clienteFirma.initialize();

clienteFirma.addMassiveHash(hash1);
clienteFirma.addMassiveHash(hash2);
clienteFirma.sign();
if(!clienteFirma.isError())
{
        var signaturesArray= clienteFirma.getSignaturesBase64Encoded().split("!");
        for(i=0; i<signaturesArray.length; i++)
        {
                var signature= signaturesArray[i];

                [...]

        }
}
```

9.6 Co-firma (*co-sign*)

La *co-firma* permite a varios usuarios firmar un mismo documento.

El caso de la *co-firma* es igual al de la firma normal, pero hay que pasar además al Cliente la firma electrónica (el CMS) de los demás firmantes. Esto se puede hacer de diferentes maneras:

- mediante un fichero que contenga el CMS codificado en base 64, con el método *setElectronicSignatureFile*, que recibe como parámetro una cadena con la ruta al fichero.
- especificándolo directamente en base 64, con el método *setElectronicSignature* que recibe como parámetro una cadena con la firma en base 64.
- Si no se especifica, se pedirá al usuario que elija un fichero con la firma en base 64.

Una vez especificados los parámetros necesarios, se invoca el método *coSign*. La salida es análoga a la de la firma digital.

9.7 Contra-firma (*counter-sign*)

La contra-firma permite a un usuario firmar las firmas de otros usuarios.

El caso de la contra-firma es igual al de la co-firma, pero hay que especificar además qué firmas contra-firmar. Para ello, puede ser necesario conocer la estructura de firmantes del documento.

Para conocer la estructura de firmantes el Cliente tiene el método *getSignersStructure.* Este método devuelve una cadena que contiene los nombres de los firmantes separados por un retorno de carro (*\n* en JavaScript). Al comienzo del nombre hay tantos tabulados (*\t*) como nivel ocupe el firmante en el documento. Por ejemplo, si A y B co-firman un documento y C contra-firma la firma de A, entonces la cadena devuelta sería *A\n\tC\nB*. En el fichero *demoMultifirma.html* se puede ver un ejemplo de cómo tratar esta cadena.

Se puede especificar qué firmantes firmar de diferentes maneras:

- firmar todas las hojas (firmas no contra-firmadas): invocando el método *counterSignLeafs*
- firmar todos los nodos: (todas las firmas) invocando el método *counterSignTree*
- firmar todos los nodos (firmas) de un firmante: con el método *setSignersToCounterSign* que recibe como parámetro una cadena de nombres de firmantes separados por *\n* e invocando el método *counterSignSigners*
- firmar nodos (firmas) determinados: con el método *setSignersToCounterSign* que recibe como parámetro una cadena de índices (partiendo de 0) de nodos (según los devuelve *getSignersStructure*) separados por *\n* e invocando el método *counterSignSigners*

La salida es análoga al caso anterior.

9.8 Inicio del proceso de cifrado

Conviene inicializar el cliente de cifrado antes de invocar las funciones que lo componen, debido a que comparte recursos con los procesos de firma y podría haber incompatibilidad en la entrada de datos. Existe una función JavaScript que lo reinicia automáticamente a sus valores por defectos y su llamada es análoga a la del proceso de firma descrita en el apartado 9.1.

Existe una librería JavaScript llamada ***cripto.js*** que contiene gran parte de las funciones de cifrado aquí descritas, para facilitar la integración de este componente.

9.9 Cifrado de datos

Para iniciar el proceso de cifrado habrá que introducir previamente los datos a cifrar. Es posible especificar los datos a cifrar de diferentes formas:

- **datos o texto en plano**: se especifica cual es la cadena a cifrar mediante el método ***setPlainData***. Internamente se realizará las codificaciones correspondientes para garantizar la fiabilidad del cifrado y posterior descifrado.

- **fichero**: es posible especificar que los datos a cifrar provienen de un archivo indicándole la ruta a la llamada del cifrador (***cipherFile***), o bien, usando la función ***setFileuri*** para especificar dicho archivo. En ambos casos habrá que usar la ruta absoluta del fichero.

Por defecto el cliente de cifrado define como algoritmo de cifrado AES y generación automática de clave, aunque posteriormente veremos las posibilidades de configuración de estos parámetros. Tras indicar la configuración del cifrador y los datos a cifrar podemos realizar la llamada al método ***cipherData*** o ***cipherFile*** del applet, o la correspondiente función JavaScript ***cifrar*** (en ***cripto.js***). El comportamiento de la llamada es análogo al resto de llamadas al applet, indicando si la ejecución se ha llevado a cabo de forma correcta o los errores en caso negativo.

Los datos cifrados se podrán obtener una vez haya finalizado mediante la llamada al método ***getCipherData***, devolviendo este una cadena codificada en formato Base 64. Es posible almacenar los datos cifrados en un archivo mediante la función ***saveCipherDataToFile***, a la cual le pasaremos la ruta absoluta del archivo destino (atención, el archivo destino será sobrescrito para evitar problemas a la hora de descifrar). El contenido del archivo destino no se codifica, por lo que no se recomienda su edición, ya que podría alterar gravemente el contenido plano del mensaje cifrado o incluso destruirlo.

Un ejemplo de aplicación de lo anterior para un proceso completo de cifrado sería el siguiente:

```
<html>
  <head>
    <script type="text/javascript" language="javascript" src="constantes.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/time.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/appletHelper.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/instalador.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/cripto.js"></script>
    <script type="text/javascript" language="javascript">
       function cifrar()
```

```
        {
                var texto= document.getElementById("campo1").value;
                clienteFirma.initialize();
                clienteFirma.setKeyMode(GENERATEKEY);
                clienteFirma.setCipherAlgorithm(AES);
                clienteFirma.setPlainData(texto);
                clienteFirma.setShowErrors(false);
                cifrarDatos();
                if(!clienteFirma.isError()){
                        var datosCifrados = clienteFirma.getCipherData();
                        var campoCifrado =document.getElementById(campo2);
                        campoCifrado.value = datosCifrados;
                        return true;
                }else{
                        alert("No se ha podido cifrar los datos: "+clienteFirma.getErrorMessage());
                        return false;
                }
        }
</script>


    […]

  </head>

  <body onload="cargarAppletFirma();">

  […]
    <label>Datos planos</label><br/>
    <textarea id=campo1 cols=20 rows=5 nowrap>Introduzca texto plano aquí</textarea>
    <br/><br/><input type=button value=Cifrar onClick=cifrar();/><br/><br/>
    <label>Datos cifrados</label><br/>
    <textarea id=campo2 cols=20 rows=5 nowrap readonly></textarea>
  […]

  </body>
</html>
```

Este ejemplo básico captura el texto introducido en un área de texto, la cifra con generación automática de clave y el algoritmo AES y la muestra en un segundo área de texto tras pulsar un botón. Para más información, consultar el ejemplo incluido y la documentación adicional.

9.10 Descifrado de datos

De manera similar al cifrado, deberemos especificar cuales son los datos a descifrar, y al igual que antes podremos especificar los datos cifrados mediante dos métodos distintos:

- **datos o texto cifrado**: se especifica cual es la cadena a descifrar mediante el método *setCipherData*. Los datos de entrada estarán en base 64 (igual que la salida del algoritmo de cifrado) para evitar la aparición de caracteres extraños o no imprimibles. Internamente estos datos se decodificaran a la base apropiada y se descifrarán.

- **fichero**: también es posible especificar que los datos a cifrar provienen de un archivo indicándole la ruta (*decipherFile*), o usando la función *setFileuri* para especificarla. También aquí se deberá especificar la ruta absoluta del fichero.

Evidentemente para descifrar datos no podremos auto generar una clave, sino que tendremos que especificarle una siempre. En caso que se intente iniciar el método de descifrado sin especificar la clave supondrá un fallo automático. Los datos descifrados se pueden recuperar mediante la llamada a la función *getPlainData*. También tenemos un método para escribir estos

datos recuperados a un archivo mediante la llamada a ***savePlainDataToFile*** y pasándole la ruta absoluta del archivo destino.

Un ejemplo básico para descifrar sería el siguiente:

```
<html>
  <head>
    <script type="text/javascript" language="javascript" src="constantes.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/time.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/appletHelper.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/instalador.js"></script>
    <script type="text/javascript" language="javascript" src="common-js/cripto.js"></script>
    <script type="text/javascript" language="javascript">
        function descifrar()
        {
                var textoCifrado= document.getElementById("campo1").value;
                var clave=document.getElementById(clave).value;
                var archivoOrigen=document.getElementById(origen).value;
                clienteFirma.initialize();
                clienteFirma.setKey(clave);
                clienteFirma.setKeyMode(USERINPUT);
                clienteFirma.setCipherAlgorithm(AES);
                clienteFirma.setShowErrors(false);
                descifrarArchivo(archivoOrigen);
                if(!clienteFirma.isError()){
                        var datosPlanos = clienteFirma.getPlainData();
                        var campoPlano=document.getElementById(campo2);
                        campoPlano.value = datosCifrados;
                        var archivoDestino=document.getElementById(destino).value;
                        clienteFirma.savePlainDataToFile(archivoDestino);
                        return true;
                }else{
                        alert("No se ha podido descifrar los datos: "+clienteFirma.getErrorMessage());
                        return false;
                }
        }
</script>


    […]

  </head>

  <body onload="cargarAppletFirma();">

  […]
    <label>Fichero cifrado:</label>
    <input type=file id=origen/>
    <label>Fichero plano (introduzca URI):</label>
    <input type=text id=destino value=/>
    <br/><br/><input type=button value=Descifrar onClick=descifrar();/><br/><br/>
    <label>Datos descifrados</label><br/>
    <textarea id=campo2 cols=20 rows=5 nowrap readonly></textarea>
  […]

  </body>

</html>
```

9.11 Creación de estructuras CMS cifradas

El cliente permite la posibilidad de incluir datos cifrados en un mensaje CMS. Se contemplan tres estructuras distintas que se describirán más detenidamente posteriormente. Estas son:

- CMS encriptado (*EncryptedCMS*).
- CMS envuelto (sobre digital, *EnvelopedCMS*).

- CMS firmado y envuelto (*Sign&Envelop*).

Las tres opciones están dentro de RFC 3852: *Cryptographic Message Syntax* correspondiente a las versión 3 de CMS, pero debido a sus características, tanto CMS envuelto como 'firmado y envuelto' no son compatibles con PKCS#7.

9.11.1 CMS encriptado

Esta estructura está basada en un mensaje criptográfico que sólo contiene el texto cifrado y opcionalmente el algoritmo utilizado para el cifrado. No contiene ninguna información sobre la clave, emisor o receptor. La metodología de creación es:

1. Se establece los datos a incluir en el mensaje mediante una llamada a ***setData***, pasándole la cadena a incluir como datos, o ***setFileuri***, para incluir un fichero. Opcionalmente se definen el algoritmo de cifrado, la clave y el modo de clave.

2. Se realiza una llamada a ***buildCMSEncrypted***.

3. El CMS generado se puede recuperar como un *String* codificado en base 64 mediante el método ***getCMSData*** o guardarla en un archivo con la operación ***saveDataToFile***.

9.11.2 CMS envuelto

Mediante la creación de un CMS envuelto obtenemos un sobre digital en el cual podremos incluir contenido cifrado sólo visible por los receptores que le indiquemos. Posteriormente veremos la estructura generada y comentaremos algunos detalles sobre ella.

El procedimiento de creación es el siguiente:

1. Definimos los datos a incluir en el sobre digital de igual manera que en el apartado anterior, indicando los datos mediante ***setData*** o un fichero mediante ***setFileuri***. Definimos también el resto de parámetros opcionales.

2. Establecemos los receptores válidos para el mensaje mediante una llamada a la función ***setRecipientsToCMS*** especificándole como parámetros una cadena con los diferentes archivos con la clave pública de los diferentes sujetos separados por retornos de carro. Estos ficheros deberán indicar su ruta completa y pueden ser formato CER o DER. Como mínimo habrá que especificar un receptor válido.

3. Hacemos la llamada al método ***buildCMSEnveloped***. Tras la llamada nos solicitará que indiquemos el emisor del mensaje mediante la selección de nuestro certificado digital, aunque es opcional indicar el emisor es recomendable.

4. Una vez concluida, podremos obtener el resultado mediante la operación ***getCMSData*** o guardarla en un archivo con la operación ***saveDataToFile***.

9.11.3 CMS firmado y envuelto

Similar al CMS envuelto, pero los datos además de cifrarse son firmados por el emisor. El procedimiento es el siguiente:

1. Definimos los datos a incluir en el sobre digital de igual manera que en el apartado anterior, indicando los datos mediante *setData*. Para incluir un fichero le indicaremos la dirección absoluta del fichero en la llamada.

2. Establecemos los receptores válidos para el mensaje mediante una llamada a la función *setRecipientsToCMS* especificándole como parámetros una cadena con los diferentes archivos con la clave pública de los diferentes sujetos separados por retornos de carro. Estos ficheros deberán indicar su ruta completa y pueden ser formato CER o DER. Como mínimo habrá que especificar un receptor válido.

3. Hacemos la llamada al método *signAndPack* si hemos especificado datos o al método *signAndPackFile* utilizando la ruta del fichero. Tras la llamada nos solicitará que indiquemos el emisor del mensaje mediante la selección de nuestro certificado digital, en esta ocasión es obligatoria para firmar los datos.

4. Una vez concluida, podremos obtener el resultado mediante la operación *getCMSData* o guardarla en un archivo con la operación *saveDataToFile*.

9.12 Recuperación de mensajes CMS cifrados

Se puede recuperar los mensajes cifrados mediante la invocación del método de cliente *recoverCMS*, la cual devolverá en función de la estructura del CMS de entrada los datos cifrados en su interior. Como se ha visto anteriormente, ciertas estructuras CMS deben ser descifradas con un certificado específico, por lo que se requerirá que alguno de los certificados autorizados en la estructura esté presente en el explorador en el cual se realiza la acción.

10 Configuración del Cliente

10.1 Algoritmos y formatos

10.1.1 Algoritmos de firma digital

El cliente permite usar dos algoritmos diferentes de firma digital: *sha1WithRsaEncryption* (SHA1 y RSA) y *md5WithRsaEncryption* (MD5 con RSA). El primero de ellos es más seguro (es el por defecto) pero se incluye el segundo por compatibilidad y flexibilidad. Se pueden cambiar con el método *setSignatureAlgorithm,* que recibe como parámetro una de las dos cadenas citadas. Se recomienda no utilizar MD5 ya que su algoritmo está comprometido.

10.1.2 Algoritmos de cifrado

Los algoritmos de cifrado permitidos son los siguientes: *AES* (por defecto), *CAST5*, *IDEA*, *SERPENT*, *TDES*, *TWOFISH* y *RC5*. Para establecer el algoritmo deberemos invocar la función *setCipherAlgorithm* y podemos recuperar el algoritmo actual con el método *getCipherAlgorithm*.

10.1.3 Modo de clave

Definen de qué manera se trata la clave de cifrado. Existen dos posibilidades *GENERATEKEY* y *USERINPUT*.

10.1.4 Formato de firma electrónica

El cliente permite crear estructuras *CMS (*por defecto) y *CADES-BES*. También permite crear firmas digitales (formato *NONE,* solo vale para firma sencilla, no para multifirmas).

Si se instala el *plugin XAdES* también estará disponible los formatos **XADES** y **XMLDSig**, con las modificaciones frente a los formatos específicos indicados anteriormente (apartado 7.3).

El formato se puede cambiar con el método *setSignatureFormat* (que recibe como parámetro una de las cadenas: CMS, CADES, XADES, XMLDSIG, NONE).

10.1.5 Modo de firma electrónica

Permite definir cómo se trata los datos firmados dentro de la firma electrónica. Pueden tratarse de forma **explícita** (no incluye los datos) o **implícita** (encapsula los datos dentro de la firma electrónica). Este parámetro sólo es útil actualmente para los formatos CMS y CADES y para los formatos XMLDSig y XAdES especificados.

Se debe tener precaución a la hora de utilizar este parámetro, ya que los parámetros del applet permiten especificar por ejemplo un modo implícito para firmar un hash y obviamente esto no

proporciona una firma electrónica útil debido a que no disponemos de los datos originales para insertarlos en el cuerpo de la firma, sino solamente el hash asociado a estos.

10.2 Filtrado y selección de certificados

El Cliente permite filtrar los certificados que el usuario puede escoger para firmar así cómo obligarle a firmar con uno determinado.

En ambos casos se hace mediante expresiones regulares java y un mini-lenguaje de expresiones que permite filtrar por número de serie del certificado del sujeto, por DN del sujeto o del emisor o por hash del certificado del sujeto calculado con MD5 o SHA1.

Las condiciones pueden ser:

- simples: {campo#operador{valor}}, siendo
    - campo:
        - SUBJECT.SN: Número de serie del certificado del sujeto
        - SUBJECT.DN: *Domain Name* del certificado del sujeto
        - SUBJECT.FP(MD5) / SUBJECT.FP(SHA1): Hash del certificado del sujeto en MD5 o SHA1
        - ISSUER.DN: *Domain Name* del certificado del emisor. Sirve para filtrar certificados por su entidad emisora (FNMT, DGP, …).
    - Operador
        - =: Igual
        - #MATCHES#: Que cumple una expresión regular Java, p. ej, para seleccionar el certificado de firma del DNI-e:
            - `{ISSUER.DN#MATCHES#{"CN=AC DNIE 00(1|2|3),OU=DNIE,O=DIRECCION GENERAL DE LA POLICIA,C=ES"}&&{SUBJECT.DN#MATCHES#{".*(FIRMA).*"}}}`
        - #NOT_MATCHES#: Que NO cumple una expresión regular Java, p. ej, para descartar los certificados de autenticación:
            - `{SUBJECT.DN#NOT_MATCHES#{".*(AUTENTICACIÓN).*"}}`
    - La cadena contra la que se opera
    - Por ejemplo, para indicar un *serial number* (número de serie) determinado de un certificado:
        - `{SUBJECT.SN={"1014673794"}}`
- Compuestas: {condicion_simple_o_compuesta nexo condicion_simple_o_compuesta}
    - Nexo:

- &&: Y lógico
- ||: O lógico

- Por ejemplo, para seleccionar el certificado con *serial number* `1014673794` emitido por la FNMT:

```
{SUBJECT.SN={"1014673794"}&&ISSUER.DN={"OU = FNMT Clase 2 CA,O= FNMT,C = ES}}
```

Mediante el método ***setMandatoryCertificateFilter*** que recibe una condición, se especifica un solo certificado con el que se puede firmar. Si no hay ningún certificado que cumpla la condición o hay más de uno, se producirá un error.

Mediante el método ***setCertificateFilter***, que recibe una condición, se especifica un filtro que han de cumplir aquellos certificados que se le vayan a dar a elegir a usuario para firmar.

10.3 Opciones de visualización

En el cliente existen varios métodos que permiten configurar ciertas opciones de visualización de mensajes.

Estos métodos son los siguientes:

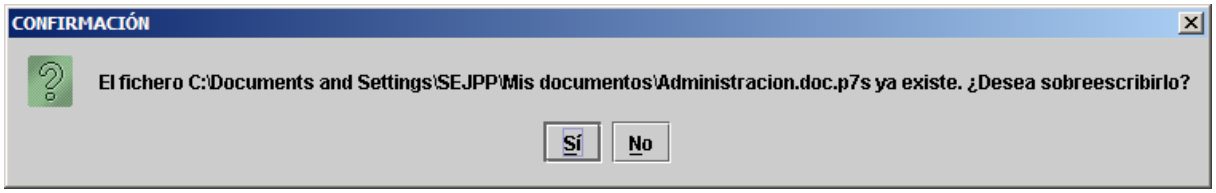
- El método ***setShowHashMessage,*** que recibe un booleano, permite especificar si se desea mostrar o no el mensaje informativo acerca del hash de los datos a firmar.
- El método ***setShowErrors*** con el parámetro ***false***, permite especificar que el Cliente no informe por su cuenta de los errores al usuario.

11 Otras funcionalidades

11.1 Guardar la firma en un fichero

El método *saveSignToFile* permite guardar la última firma generada en un fichero. Se puede especificar la ruta al fichero con *setFileuri*, que recibe una cadena con la ruta al fichero de salida. Si no se especifica, se permitirá elegir al usuario.

Si el fichero ya existe, se pide confirmación:



11.2 Obtener el certificado usado para firmar

El método *getSignCertificateBase64Encoded* devuelve una cadena con el certificado asociado a la firma, codificado en base 64.

11.3 Leer el contenido de un fichero de texto

El método *getTextFileContent* que recibe como parámetro una URI a un fichero devuelve el contenido del mismo como una cadena. Si el fichero está almacenado en local, la URI comenzará por file:///.

11.4 Leer el contenido de un fichero en base 64

El método *getFileBase64Encoded* que recibe dos parámetros (ruta al fichero y un booleano que indica si mostrar o no gráficamente al usuario el progreso en la lectura del fichero)

11.5 Obtener el hash de un fichero

El método *getFileHashBase64Encoded* devuelve una cadena con el hash de un fichero codificado en base 64.

11.6 Obtener la estructura de un CMS encriptado o envuelto

Como método de diagnóstico podemos obtener la estructura de un CMS encriptado o envuelto mediante la llamada a los métodos *formatEncryptedCMS* y *formatEnvelopedCMS* respectivamente, pasándole como parámetros la cadena en base 64 correspondiente al CMS del que queramos obtener su estructura.

11.7 Obtener y fijar la clave de cifrado

Para obtener la clave que se ha utilizado para cifrar ciertos datos deberemos ejecutar el método *getKey*, el cual nos devolverá la clave codificada en base 64. Para fijar una clave usaremos *setKey*, adjuntando como parámetro la clave deseada en base 64.

12 Ejemplos

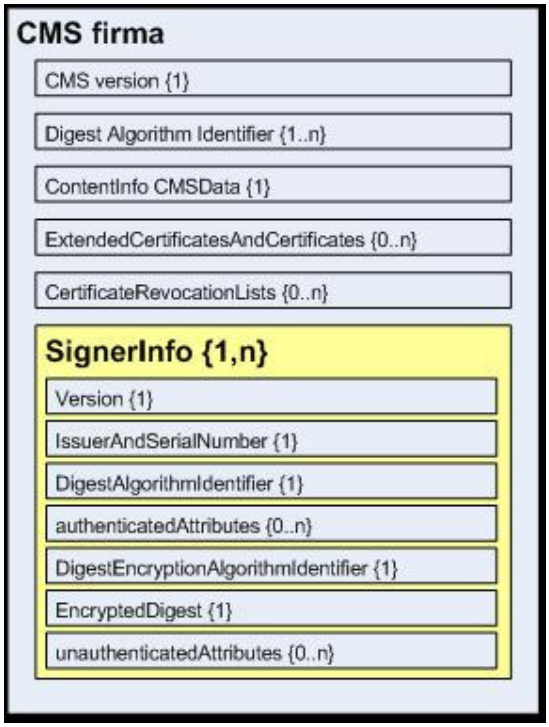
En la carpeta web-instalador se adjuntan varios ejemplos más completos que los aquí citados:

- *demoInstalador.html*: Ejemplo de la funcionalidad del instalador
- *demoFirmaMasiva.html*: Ejemplo de firmas masivas
- *demoMultifirma.html*: Ejemplos de firma, co-firma y contra-firma
- *demoVisorNodosMultifirma.html*: Ejemplo de tratamiento de multifirmas y su estructura de nodos.
- *demoFirmaWeb/demoFirmaWeb*-01.*html*: Ejemplo de firma web
- *demoCifrado.html*: Ejemplo de cifrado.
- *demoSobreDigital*: Ejemplo de CMS encriptado, CMS envuelto y CMS firmado y envuelto.

13 Estructura de formatos

13.1 Firma CMS

Al igual que otros elementos CMS que describiremos posteriormente, la estructura de una firma CMS viene definida en la RFC 3852, aunque en este caso en especial, y para mantener la compatibilidad con PKCS#7, el elemento *signedData* especificado en la RFC indicada se encuentra limitado. La estructura empleada por el cliente de firma es la siguiente:

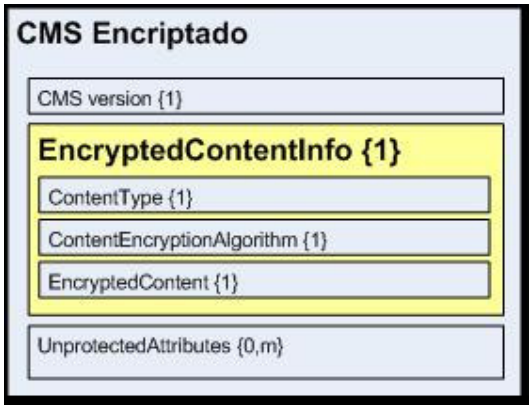


- CMS Version. Indica las diferentes versiones del mensaje. Para que la compatibilidad con PKCS#7 se mantenga debe ser 0.
- Digest Algorithm Identifier. Identifica el algoritmo utilizado.
- ContentInfo. Secuencia de parámetros que identifican el contenido del mensaje. Comprende el tipo de contenido (*contentType*) que en nuestro caso será *Data* y *content* que se refiere a la secuencia de bytes correspondiente a los datos mismos.
- Extended Certificates And Certificates. Opcional. Permite especificar una cadena de certificados para la validación de los distintos certificados firmantes.
- Certificate revocation Lists. Opcional. Permite especificar las CRL para los certificados utilizados.

- Signer Info. Estructura que especifica la información de los diferentes firmantes del contenido del mensaje. Se subdividen en los siguientes campos:
    - Versión. Especifica la versión de esta estructura y será siempre 1 para PKCS#7.
    - Issuer And Serial Number. Especifica el certificado usado mediante el emisor y número de serie de éste.
    - Digest Algorithm Identifier. Identifica el algoritmo utilizado.
    - Authenticated Attributes. Opcional. Secuencia de atributos firmados que especifican ciertos parámetros importantes para la interpretación del contenido. Si el tipo del contenido fuese distinto de *Data*, sería obligatorio incorporar como atributos el tipo empleado y el hash del contenido, pero en nuestro caso esto no es posible ya que siempre tendremos el *ContentType Data*.

- **Digest Encryption Algorithm**. Describe que algoritmo se ha usado en la encriptación de la firma y resumen del documento.
- **Encrypted Digest**. Hash del mensaje encriptado empleando la clave privada del certificado y el algoritmo especificado antes
- **Unauthenticated Attributes**. Opcional. Atributos no firmados definidos en PKCS#9, como por ejemplo las contrafirmas.

13.2 CMS encriptado



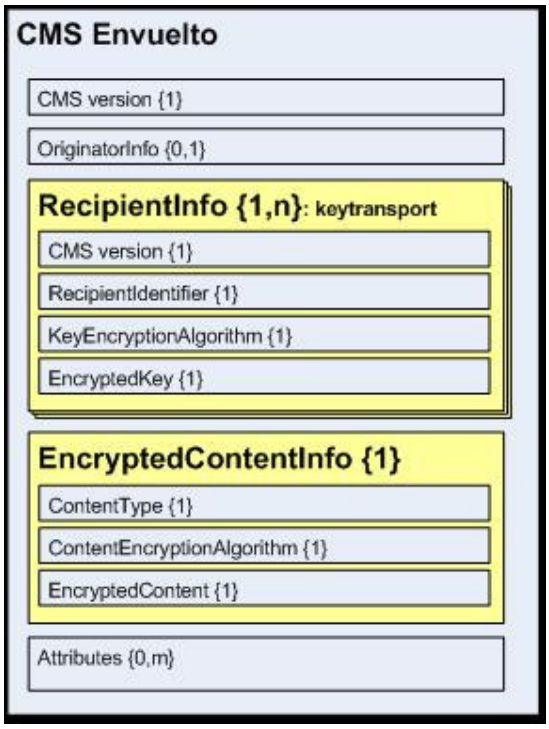
La estructura vendría definida como sigue:

- **CMS Version**. Será 0 si no existen *UnprotectedAtributes* o 2 en caso contrario.
- **Encrypted Content Info**. Subestructura que se define mediante los siguientes campos:
  - **Content Type**. Define el tipo de contenido.
  - **Content Encryption Algorithm**. Define el algoritmo utilizado para encriptar el contenido.
  - **Encrypted Content**. Contenido encriptado usando el algoritmo especificado anteriormente.
- **Unprotected Attributes**. Opcional. Secuencia de parámetros auxiliares definidos por otros estándares.

Como se puede apreciar, esta estructura no contiene la clave de cifrado ni ningún método de transmisión de esta, por lo que si se usa como mensaje se debe buscar un método para compartir una clave privada.

13.3 CMS Envuelto

Esta estructura se identifica con el sobre digital identificado en la RFC 3852 como **Enveloped CMS** y sigue la siguiente estructura:



- **CMS Version**. Viene determinada en función de los parámetros presentes en la estructura generada. Para que fuese compatible con la estructura especificada en PKCS#7 debería ser 0, pero quitaría mucha de las opciones más importantes que incorpora esta solución.
- **Originator Info**. Define el emisor del mensaje. Aunque es opcional, su presencia viene determinada por los algoritmos utilizados internamente.
- **Recipient Info**. Define los receptores válidos para el mensaje actual. Por requerimientos de la tecnología utilizada serán de tipo **keytransport**, ya que necesitamos incorporar la clave simétrica utilizada en el cifrado. Se distinguen los siguientes campos:
  - **Version**. Puede ser 0 o 2 en función de los datos incluidos en **Recipient Identifier**. En nuestro caso será 2.
  - **Key Encryption Algorithm**. Define el algoritmo por el cual se ha encriptado la clave simétrica adjunta. Se utilizan algoritmos asimétricos y en nuestro caso RSA.
  - **Encrypted Key**. Clave utilizada para encriptar el contenido del mensaje cifrada utilizando el algoritmo anteriormente definido.
- **Encrypted Content Info**. Estructura igual que la contenida en CMS encriptado:
  - **Content Type**. Define el tipo de contenido.
  - **Content Encryption Algorithm**. Define el algoritmo utilizado para encriptar el contenido.
  - **Encrypted Content**. Contenido encriptado usando el algoritmo especificado especificado.
- **Unprotected Attributes**. Conjunto de atributos no cifrados definidos o necesarios por otros estándares.

13.4 CMS Firmado y envuelto

Esta estructura es un atajo para crear un CMS envuelto en cuyo interior se encuentra un mensaje firmado. Esto significa que la única diferencia en cuanto a la estructura es que el *conten type* de la subestructura *Encrypted Content Info* sería un *Signed Data* como el definido en el CMS Firmado.

La diferencia fundamental es que los parámetros a especificar no son tan libres, ya que por ejemplo es obligatorio especificar el emisor, ya que tenemos que firmar el mensaje con su certificado.

Este es un ejemplo de cómo se pueden anidar estructuras CMS. Por ejemplo, podríamos insertar un CMS envuelto en un CMS firmado (obviando la utilidad que pudiese tener o no) simplemente generando el CMS envuelto y especificando el resultado de la salida como datos de entrada para la creación del CMS firmado, y así sucesivamente.

13.5 Firma CADES

El nuevo formato de firma electrónica CAdES se considera la evolución de CMS (RFC 3852) y viene definido en el documento ETSI TS 101 733.

En CAdES se define un nuevo concepto de firma electrónica conocido como *políticas de firma*, en las cuales se definen metodologías para aportar consistencia a la hora de la validación de firmas electrónicas. El tratamiento de éstas políticas viene definido en el documento ETSI TS-101-734.

CAdES permite distintos formatos de firma al igual que su homologo para XMLDSIG (XAdES). Entre ellos podemos distinguir CAdES-T, CAdES-C, CAdES-T, CAdES-X y CAdES-A, todos ellos pudiendo generarse sobre CAdES-EPES o CAdES-BES, según se basen en Políticas de Firma o no, respectivamente.

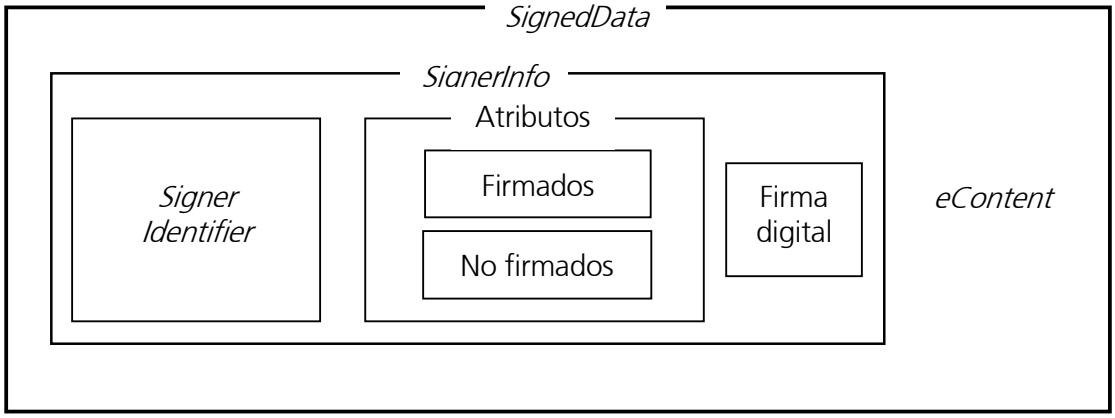
La gran (y única, respecto a lo que envoltorios de Firmas Electrónicas se refiere) diferencia entre CMS y CAdES radica en el atributo firmado *SigningCertificate* presente en éste último, lo que confiere mayor seguridad a la firma electrónica al evitar ataques por suplantación de certificado.

13.6 Estructura de CAdES

13.6.1 Estructura de CAdES-BES

A continuación se definen que elementos se encuentran de manera obligatoria u opcional dentro de una firma en formato CAdES-BES. La estructura general comprende las siguientes partes:

- Los datos de usuario firmados definidos en CMS (RFC 3852).
- Colección de atributos firmados obligatorios definidos tanto en CMS como en ESS (RFC 2634).
- Atributos firmados obligatorios definidos en CAdES (ETSI 101 703).
- La firma digital correspondiente tanto a los datos como a los atributos firmados definidos.
- Opcionalmente puede llevar tanto otros atributos firmados como atributos sin firmar (CAdES, CMS y ESS).



Al igual que en CMS, los elementos obligatorios son:

Content Type	RFC 3852
Message Digest	RFC 3852
ESS Signing Certificate v2	RFC 2634

Los atributos opcionales son:

Signing time	RFC 3852	Indica la fecha y hora a la que el firmante ha firmado los datos.
Content hints	RFC 2634	Informa acerca de las capas internas de la firma electrónica cuando se encuentran varios *eContents* anidados.
Content reference	RFC 2634	Enlaza un *SignedData* con otro.
Content identifier	RFC 2634	Proporciona un identificador que permite referenciar el presente *SignedData* desde otro.
Commitment type indication	ETSI 101 733	Indica explícitamente al verificador de la firma el cometido que tiene ésta. Está asociado a la política de firma definida.
Signer location	ETSI 101 733	Asocia un lugar geográfico al firmado de los datos tal como se recomienda en ***ITU-T***

		Recommendation F.1
Signer attributes	ETSI 101 733	Establece otros atributos del firmante, tales como *Role*.
Content timestamp	ETSI 101 733	*TimeStampToken* de los datos en el momento previo al firmado.
Counter Signature	RFC 3852	Contrafirma de los datos tal como se especificaba en CMS (RFC 3852). Este atributo no es firmado.

Debemos de tener en cuenta que no todos los elementos descritos en la tabla anterior deberán estar presentes ya que en el cliente de firmado sólo contemplamos la generación de CAdES-BES, por lo que la mayoría de estos atributos son prescindibles.

14 Formato de firma XAdES y XMLDSignature

Los formatos de firma XAdES (ETSI TS 101 903) y XMLDSignature (RFC 3275) proporcionadas mediante plugin facilitan la generación de éstos formatos de firmas basados en una estructura XML. La estructura general de una firma XML es:

```
<Signature ID?>
     <SignedInfo>
        <CanonicalizationMethod/>
        <SignatureMethod/>
        (<Reference URI?>
         (<Transforms>)?
         <DigestMethod>
         <DigestValue>
        </Reference>)+
     </SignedInfo>
     <SignatureValue>
     (<KeyInfo>)?
     (<Object ID?>)*
     (<QualyfingProperties> --sólo XAdES)
</Signature>
```

Además, una firma XAdES incorpora una serie de atributos firmados y no firmados que aportan consistencia a la firma electrónica. La existencia de estos atributos facilita la incorporación de otros formatos de firma electrónica (XAdES-T, XAdES-C, etc.) que por sus características y restricciones no es posible generarlas con el cliente de firmado de @Firma. La estructura de atributos XAdES es la siguiente:

```
<QualyfingProperties>
     <SignedProperties>
            (<SigningTime>)?
            (<SigningCertificate>)?
            (<SignerRole>)?
            ...
     </SignedProperties>
     <UnsignedProperties>
            (<CounterSignature>)?
            (<CompleteCertificateRefs>)?
            (<SignatureTimeStamp>)?
```

```
            ...
        </UnsignedProperties>
    </QualyfingProperties >
```

Dentro de XAdES encontramos varias versiones cuyas diferencias respecto al cliente resumimos a continuación:

- XAdES v1.1.1. Generada por la versión 1.2 del cliente. Versión inicial de XAdES.
- XAdES v1.2.2. No generada por el cliente.
- XAdES v1.3.2. Generada por el cliente a partir de la versión 2.1.

Entre estas versiones existen ciertas diferencias en cuanto a formato, espacio de nombres, distribución de elementos, etc. Por lo que recomendamos consultar la documentación del estándar para más información.

Desde el punto de vista de las versiones de XAdES las principales diferencias entre versiones son las siguientes:

- XAdES v1.1.1.
  1. Versión inicial de XAdES.
- XAdES v1.2.2.
  1. Los elementos <xades:SigningTime> y <xades:SigningCertificate> pasan a ser opcionales; por el contrario, en caso de no existir <xades:SigningCertificate> se exige que la información de clave pública contenida en <ds:KeyInfo> haya sido protegida por la firma digital (evita ataques de suplantación de certificado). El cliente al no generar esta versión no se ve afectado.
  2. Se modifica el namespaces de los elementos <xades:DigestMethod> y <xades:DigestValue>, pasando a denominarse <ds:DigestMethod> y <ds:DigestValue>, respectivamente (donde ds es el namespace de XMLDsig).
  3. Cambio de la estructura del elemento <xades:SignatureTimeStamp>.
  4. El cliente no se ve afectado al no generar esta versión.
- XAdES v1.3.2.
  1. Nuevo cambio de la estructura del elemento <xades:SignatureTimeStamp>.
  2. El cliente no genera el formato XAdES-T, por lo que nunca incorporará el sellado de tiempo.

Desde el punto de vista de estructura distinguimos tres variantes en firmas XML:

1. Firma XML detached. Son aquellas en las que la relación entre el objeto firmado y la firma electrónica no está identificada por relaciones de parentesco, es decir, son independientes. Podemos verlo en el siguiente ejemplo:

```
<Signature Id="id1">
  <SignedInfo>
      <CanonicalizationMethod/>
      <SignatureMethod/>
      <Reference URI="???">
          <DigestMethod/>
          <DigestValue/>
      </Reference>
      …
  </SignedInfo>
  <SignatureValue Id="id2"/>
  <KeyInfo >…</KeyInfo>
</Signature>
```

   La URI indicada en el objeto *Reference* resaltado no tiene porqué estar referenciada en el documento XML asociado a la firma, así pues se podría utilizar URIs externas o referencias a elementos dentro de un elemento XML padre.

2. Firma XML enveloping. Son aquellas en las que la relación entre el objeto firmado y la firma electrónica se identifica por un parentesco hijo-padre. El objeto firmado se introduce en un elemento *Object* perteneciente a la propia firma electrónica.

```
<Signature Id="id1">
  <SignedInfo>
      <CanonicalizationMethod/>
      <SignatureMethod/>
      <Reference URI="#objetoFirmado">
          <Transforms/>
          <DigestMethod/>
          <DigestValue/>
      </Reference>
      …
  </SignedInfo>
  <SignatureValue Id="id2"/>
  <KeyInfo >…</KeyInfo>
  <Object Id="objetoFirmado">
      …
  </Object>
</Signature>
```

Este formato de firma electrónica requiere que el elemento referenciado sea accesible directamente.

3. **Firma XML enveloped**. Son aquellas en las que la relación entre el objeto firmado y la firma electrónica está identificada por una relación de padre-hijo, es decir, el objeto firmado contiene la firma electrónica. Es evidente que al tratarse de una firma XML los únicos datos que pueden firmarse siguiendo este formato son aquellos que tengan una estructura XML.

```
<ObjetoFirmado atributos>
  <Nodo1 Id="nodo_1">
      <Subnodo1_1/>
      <Subnodo1_2 value="x">
  </Nodo1>
  <Nodo2 Id="nodo_2">
      <Subnodo2_1 Id="texto">
          Esto es una prueba
      </Subnodo2_1>
  </Nodo2>
  <Signature Id="id1">
      <SignedInfo>
          <CanonicalizationMethod/>
          <SignatureMethod/>
          <Reference URI="">
              <Transforms>
                  <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
              </Transforms>
              <DigestMethod/>
              <DigestValue/>
          </Reference>
          …
      </SignedInfo>
      <SignatureValue Id="id2"/>
      <KeyInfo >…</KeyInfo>
  </Signature>
</ObjetoFirmado>
```

Como se puede observar la URI de la referencia resaltada aparece vacía. Además, esta referencia siempre deberá contener la transformación ***enveloped-signature***, la cual indicará que el elemento al que hace referencia el elemento *Reference* actual es el padre del elemento *Signature*.

Teniendo lo anterior en cuenta se ha realizado una ampliación en los formatos admitidos en el plugin XAdES. La firma generada por los clientes anteriores a 2.1 se puede esquematizar de la siguiente manera:

```
<Signature Id="S0">
  <SignedInfo>
  <CanonicalizationMethod/>
      <SignatureMethod/>
      <Reference URI="#D0">
          <DigestMethod/>
          <DigestValue/>
      </Reference>
      <Reference URI="#S0-SignedProperties">
          <DigestMethod/>
          <DigestValue/>
      </Reference>
  </SignedInfo>
  <SignatureValue Id="S0-SIG"/>
  <KeyInfo>
      <KeyValue>
          <RSAKeyValue>
              <Modulus/>
              <Exponent/>
          </RSAKeyValue>
      </KeyValue>
      <X509Data>
          <X509Certificate>/>
      </X509Data>
  </KeyInfo>
  <Object>
      <QualifyingProperties Target="#S0">
          <SignedProperties Id="S0-SignedProperties">
              <SignedSignatureProperties>
                  <SigningTime/>
                      <SigningCertificate>
                          <Cert>
                              <CertDigest>
                                   <DigestMethod/>
                                   <DigestValue/>
                              </CertDigest>
                              <IssuerSerial>
                                   <X509IssuerName/>
                                   <X509SerialNumber/>
                              </IssuerSerial>
                          </Cert>
                      </SigningCertificate>
                </SignedSignatureProperties>
              <SignedDataObjectProperties/>
          </SignedProperties>
      </QualifyingProperties>
    </Object>
    <Object ID/> /*Contiene los datos firmados*/
</Signature>
```

Lo que se identifica como una firma *enveloping*. Sin embargo la firma generada por el cliente 2.3 genera una estructura semejante a la generada por la plataforma (formato *detached*), y que se corresponde con el siguiente esquema:

```
<AFIRMA>
  <CONTENT ID/> /*Contiene los datos firmados*/
    <Signature>
      <SignedInfo>
              <CanonicalizationMethod/>
          <SignatureMethod/>
          <Reference URI="#D0">
              <DigestMethod/>
              <DigestValue/>
          </Reference>
          <Reference URI="#S0-SignedProperties">
              <DigestMethod/>
              <DigestValue/>
          </Reference>
      </SignedInfo>
      <SignatureValue Id="S0-SIG"/>
      <KeyInfo>
          <KeyValue>
              <RSAKeyValue>
                  <Modulus/>
                  <Exponent/>
              </RSAKeyValue>
          </KeyValue>
          <X509Data>
              <X509Certificate>/>
          </X509Data>
      </KeyInfo>
      <Object>
          <QualifyingProperties Target="#S0">
              <SignedProperties Id="S0-SignedProperties">
                  <SignedSignatureProperties>
                      <SigningTime/>
                      <SigningCertificate>
                          <Cert>
                              <CertDigest>
                                  <DigestMethod/>
                                  <DigestValue/>
                              </CertDigest>
                              <IssuerSerial>
                                  <X509IssuerName/>
                                  <X509SerialNumber/>
                              </IssuerSerial>
                          </Cert>
                      </SigningCertificate>
                      <SignaturePolicyIdentifier>
                          <SignaturePolicyImplied/>
```

```
                    </SignaturePolicyIdentifier>
                </SignedSignatureProperties>
                <SignedDataObjectProperties/>
            </SignedProperties>
        </QualifyingProperties>
        </Object>
    </Signature>
</AFIRMA>
```

Como se puede apreciar, se ha envuelto el elemento *<Signature>* e incorporado los datos firmados en Base64 dentro del elemento *<CONTENT>*, no necesitando ya el *<Object>* usado anteriormente.

Actualmente se ha adquirido un compromiso intermedio teniendo como objetivo ampliar la capacidad de firma electrónica en formato XML. Se admite por lo tanto la firma *enveloping* similar a los clientes antiguos, *detached* similar a la originada por la plataforma y siguiendo la estructura anterior y se ha añadido *enveloped* para completar el abanico de posibilidades.

15 Sistema de plugins

El sistema puede incorporar plugins si fuese necesario. Estos estarán contenidos en un archivo ZIP y para que el instalador sea capaz de reconocerlos habrá que realizar ciertas modificaciones sobre el archivo de configuración del servidor *version.properties*. Entre estas modificaciones habrá que indicar la localización del archivo ZIP, el nombre del plugin, la clase que implementa el interfaz de formato, los jars y librerías necesarias y la versión del plugin. Esto sólo deberá realizarse si se quiere instalar un plugin sobre un servidor ya desplegado previamente y donde no existan previamente los parámetros a continuación.

Para especificar los diferentes plugins que el instalador debe instalar se añadirá la siguiente línea:

plugins.names = PLUGIN_1, PLUGIN_2,…, PLUGIN_N

Y para cada uno de los plugins indicados se deberán añadir las siguientes líneas:

plugins.PLUGIN_N.version.mayor = <versión_mayor>

plugins.PLUGIN_N.version.minor = <versión_menor>

plugins.PLUGIN_N.version.build = <num_build>

plugins.PLUGIN_N.jars* = <cadena de jars>

plugins.PLUGIN_N.libs = <cadena de librerías>

plugins.PLUGIN_N.version.bDate = <fecha de creación>

plugins.PLUGIN_N.install.jars* = <fichero ZIP que contiene los jar>

plugins.PLUGIN_N.install.libs.windows = <fichero ZIP de las librerías para windows>

plugins.PLUGIN_N.install.libs.linux = <fichero ZIP de las librerías para linux>

plugins.PLUGIN_N.class* = <nombre de clase principal>

Los parámetros marcados con asterisco (*) son obligatorios. Si alguno de los parámetros no obligatorios no hiciese falta, no se tiene por qué añadir a la configuración.

Anexo A. Matriz de compatibilidad

A.1 Windows XP

Windows XP JRE 1.4.2		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	OK	OK	OK	OK
	Plugin XML/XAdES	OK	OK	OK	OK
	Cifrado	OK	OK	OK	OK
	Sobre digital	OK	OK	OK	OK
	Instalación y funcionamiento	OK	OK	OK	OK

Windows XP JRE 1.5		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	OK	OK	OK	OK
	Plugin XML/XAdES	OK	OK	OK	OK
	Cifrado	OK	OK	OK	OK
	Sobre digital	OK	OK	OK	OK
	Instalación y funcionamiento	Problemas con la gestión del foco (ver aptdo B.1.1)		OK	OK

Windows XP JRE 1.6		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	OK	OK	OK	OK
	Plugin XML/XAdES	OK	OK	OK	OK

Windows XP JRE 1.6		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
	Cifrado	OK	OK	OK	OK
	Sobre digital	OK	OK	OK	OK
	Instalación y funcionamiento	Problemas con la gestión del foco (ver aptdo B.1.1)		Problemas con la gestión del foco. (ver aptdo B.1.1)	

A.2 Windows 2000

Windows 2000 JRE 1.4.2		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	OK	N/A	OK	OK
	Plugin XML/XAdES	OK	N/A	OK	OK
	Cifrado	OK	N/A	OK	OK
	Sobre digital	Versión de Crypto.dll demasiado baja para descifrado. (ver aptdo B.2.1) Cifrado OK	N/A	OK	OK
	Instalación y funcionamiento	OK	N/A	OK	OK

Windows 2000 JRE 1.5		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	OK	N/A	OK	OK
	Plugin XML/XAdES	OK	N/A	OK	OK
	Cifrado	OK	N/A	OK	OK
	Sobre digital	Versión de Crypto.dll demasiado baja para descifrado. (ver aptdo B.2.1)	N/A	OK	OK

Windows 2000 JRE 1.5		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
		Cifrado OK			
	Instalación y funcionamiento	Problemas con la gestión del foco (a partir de 1.5.0_06)	N/A	OK	OK

Windows 2000 JRE 1.6		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	OK	N/A	OK	OK
	Plugin XML/XAdES	OK	N/A	OK	OK
	Cifrado	OK	N/A	OK	OK
	Sobre digital	Versión de Crypto.dll demasiado baja para descifrado. (ver aptdo B.2.1) Cifrado OK	N/A	OK	OK
	Instalación y funcionamiento	Problemas con la gestión del foco (ver aptdo B.1.1)	N/A	Problemas con la gestión del foco (ver aptdo B.1.1)	

A.3 Windows Vista

Windows Vista JRE 1.4.2		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	N/A	OK	OK	OK
	Plugin XML/XAdES	N/A	OK	OK	OK
	Cifrado	N/A	OK	OK	OK
	Sobre digital	N/A	CNG no soporta descifrado. (ver aptdo B.2.2) Cifrado OK	OK	OK

Windows Vista JRE 1.4.2		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
	Instalación y funcionamiento	N/A	OK	OK	OK

Windows Vista JRE 1.5		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	N/A	OK	OK	OK
	Plugin XML/XAdES	N/A	OK	OK	OK
	Cifrado	N/A	OK	OK	OK
	Sobre digital	N/A	CNG no soporta descifrado. (ver aptdo B.2.2) Cifrado OK	OK	OK
	Instalación y funcionamiento	N/A	Persisten algunos problemas. Mejorado.	Problemas con la gestión del foco (ver aptdo B.1.1)	

Windows Vista JRE 1.6		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	N/A	OK	OK	OK
	Plugin XML/XAdES	N/A	OK	OK	OK
	Cifrado	N/A	OK	OK	OK
	Sobre digital	N/A	CNG no soporta descifrado. (ver aptdo B.2.2) Cifrado OK	OK	OK
	Instalación y funcionamiento	N/A	Persisten algunos problemas. Mejorado.	Problemas con la gestión del foco (ver aptdo B.1.1)	

A.4 Distribuciones de Linux

Las distribuciones de Linux sobre las que se ha testeado específicamente el funcionamiento del cliente de firma son las siguientes:

- Guadalinex v3 / Ubuntu v5.10
- Red Hat v4.0

Esto no es óbice para que no funcionen sobre otras distribuciones de Linux (o versiones de alguna concreta) que no se hayan reflejado en este apartado explícitamente.

Linux JRE 1.4.2		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	N/A	N/A	OK	OK
	Plugin XML/XAdES	N/A	N/A	OK	OK
	Cifrado	N/A	N/A	OK	OK
	Sobre digital	N/A	N/A	OK	OK
	Instalación y funcionamiento	N/A	N/A	OK	OK

Linux JRE 1.5		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	N/A	N/A	OK	OK
	Plugin XML/XAdES	N/A	N/A	OK	OK
	Cifrado	N/A	N/A	OK	OK
	Sobre digital	N/A	N/A	OK	OK
	Instalación y funcionamiento	N/A	N/A	OK Nota: No se han producido errores de foco.	

Linux JRE 1.6		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0
Funcionalidades	Firmado electrónico	N/A	N/A	OK	OK
	Plugin XML/XAdES	N/A	N/A	OK	OK
	Cifrado	N/A	N/A	OK	OK
	Sobre digital	N/A	N/A	OK	OK
	Instalación y funcionamiento	N/A	N/A	OK Nota: No se han producido errores de foco.	

A.4.1 Guadalinex 4.x

La distribución Guadalinex 4.x viene con ciertas librerías preinstaladas que resultan incompatibles para la correcta ejecución del Cliente de Firma 2.3.5. Debido a esta especial configuración de librerías, realizamos un estudio separado de esta.

Guadalinex 4 JRE 1.4.2		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0 (preinstalado)
Funcionalidades	Firmado electrónico	N/A	N/A	OK	OK
	Plugin XAdES	N/A	N/A	OK	OK
	Cifrado	N/A	N/A	OK	OK
	Sobre digital	N/A	N/A	OK	OK
	Instalación y funcionamiento	N/A	N/A	OK	OK

Guadalinex 4 JRE 1.5 (preinstalada)		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0 (preinstalado)
Funcionalidades	Firmado electrónico	N/A	N/A	BUG	BUG
	Plugin XAdES	N/A	N/A	BUG	BUG
	Cifrado	N/A	N/A	OK	OK
	Sobre digital	N/A	N/A	BUG	BUG
	Instalación y funcionamiento	N/A	N/A	OK No se han producido errores de foco.	

Guadalinex 4 JRE 1.6		Navegadores			
		IE6	IE7	Firefox 1.5	Firefox 2.0 (preinstalado)
Funcionalidades	Firmado electrónico	N/A	N/A	OK	OK
	Plugin XAdES	N/A	N/A	OK	OK
	Cifrado	N/A	N/A	OK	OK
	Sobre digital	N/A	N/A	OK	OK
	Instalación y funcionamiento	N/A	N/A	OK No se han producido errores de foco.	

Anexo B. Problemas frecuentes y Errores Conocidos

B.1 Problemas comunes

Durante la experiencia de uso del cliente de firma, se han encontrado una serie de problemas comunes que suelen producirse con su uso. Estos problemas son completamente ajenos al cliente de @firma:

B.1.1 Problema: Pérdida de foco en ventanas

- **Descripción**: en ocasiones, las ventanas del cliente pierden el foco, haciendo imposible la interacción del usuario.

- **Causa**: se trata de un bug reconocido por SUN a partir de la JRE 1.5.0_06 que bloquea ciertas ventanas Java en IE y Mozilla, perdiendo el foco y haciendo imposible la interacción con el usuario.

- **Solución**: hasta la versión de JRE 1.7, no se espera que se solvente dicho problema. Una posible solución es cambiar el foco a otras ventanas, o minimizar/maximizar el navegador, para intentar que recupere el foco, aunque no siempre resulta efectivo, por lo que se deberá reiniciar el navegador y reintentar la operación.

B.1.2 Problema: No se detecta la inserción/extracción del DNIe en el lector

- **Descripción:** el navegador no detecta la extracción o introducción del DNIe en el lector, por lo que si no hemos introducido la tarjeta previamente a que se instancie el CrytoManager del navegador (provocado por el arranque del cliente de firma), no se encontrará el certificado. Otro posible caso es que una vez cargado el cliente, se extraiga la tarjeta. Al realizar una operación de firma, el navegador mostrará el certificado del DNIe de la tarjeta (aunque ya no esté presente) fallando al intentar utilizarlo.

- **Causa:** se trata de un problema del navegador en la gestión de los dispositivos criptográficos (PKCS11 para Mozilla y CSP para IE), que no permite que el cliente de firma realice una petición de recarga de dichos dispositivos.

- **Solución:** Insertar la tarjeta previamente a la carga del cliente de firma.

B.1.3 Problema: No se detecta el certificado del DNIe tras una autenticación infructuosa

- **Descripción**: el navegador no detecta el certificado del DNIe si se introduce mal el PIN la primera vez, incluso aunque posteriormente el usuario sí lo introduzca correctamente.

- **Causa**: se trata de un problema del CSP (Cryptographic Service Provider) del DNI electrónico.

- **Solución**: volver a introducir la tarjeta en el lector y proceder a autenticarse de nuevo.

B.1.4 Problema: Mensajes de error por incompatibilidades entre librerías

- **Descripción**: los applets que componen el Cliente de Firma 2.3.5 muestran mensajes informando sobre incompatibilidades entre librerías. Estos mensajes pueden mostrarse durante la instalación y durante la ejecución del Cliente, pudiendo ser diferente el comportamiento en función del explorador y/o el sistema operativo.

- **Causa**: pueden surgir incompatibilidades debidas a la existencia de librerías previas a la instalación del Cliente 2.3.5, ya estén instaladas por clientes antiguos o por software de terceros o el sistema operativo. El applet Instalador ha sido configurado para que analice la presencia de librerías incompatibles preinstaladas. En caso de que se encuentren librerías no aptas para la correcta ejecución del Cliente de Firma 2.3.5, se mostrará un mensaje advirtiendo de la situación al usuario y facilitando una posible solución. Estos mensajes también se mostrarán si se llega a ejecutar el Cliente sobre librerías incompatibles.

- **Solución**: Los mensajes contienen información para el usuario final de cómo solucionar los problemas.
  El procedimiento estándar de actuación es copiar la librería conflictiva desde el directorio de instalación del cliente (por defecto, **%USER_HOME%/.clienteFirmaArrobaFirma5**) al directorio donde se encuentra la librería conflictiva (suele ser el directorio de extensiones de la JRE, **%JAVA_HOME%/lib/ext**), sobrescribiendo si fuese necesario.
  En caso de que se encuentre un problema relativo a la copia de librerías nativas dependientes, normalmente causado por problemas de permisos sobre ciertos directorios, se deberán copiar las librerías nativas ubicadas en el directorio de instalación del cliente a un directorio dentro del path del sistema, o bien incorporar el directorio de instalación al path.

B.2 Errores Conocidos

Durante la realización de las pruebas, se han encontrado algunos problemas que se detallan a continuación

B.2.1 Problema: Descifrado de sobre digital en Windows 2000

- **Descripción**: se ha detectado un fallo al descifrar un sobre digital provocado por la Crypto API (versión 5.131.2195.6661) de Windows 2000 . El error indica que no se han encontrado certificados autorizados. En Firefox 2.0 funciona correctamente.

- **Causa**: problema de la versión de la CryptoAPI de Windows 2000.

- **Solución**: utilizar Mozilla Firefox v2.0

B.2.2 Problema: Descifrado de sobre digital en Windows Vista

- **Descripción**: se ha detectado un fallo en la funcionalidad de descifrado de sobre digital a través de CNG (*Cryptoapi Next Generation*) de Windows Vista. El error indica que no se han encontrado certificados válidos.

- **Causa**: Microsoft ha adoptado como nueva política de seguridad no admitir ciertos procedimientos relacionados con las claves privadas que consideran no seguras.

- **Solución**: utilizar Mozilla Firefox v2.0.

B.2.3 Problema: No se recuperan certificados en Guadalinex 4

- **Descripción**: El cliente de firma informa de que no existen certificados en Mozilla Firefox para Guadalinex 4.x. (Firefox 2.0 y JRE 1.5)

- **Causa**: Existe un problema de compatibilidad entre las librerías de Firefox (JSS y NSS) y la JRE 1.5. El problema parece estar relacionado con la versión de Ubuntu de la que parte Guadalinex 4.x.

- **Solución**: Utilizar una JRE distinta a 1.5 para la ejecución del applet soluciona este problema.

Anexo C. Evolución del cliente

C.1 Evolución del cliente de firma

A continuación se detalla grosso modo las funcionalidades más importantes incorporadas en cada versión del cliente de firma:

Versión 1.1

- Incorporación de funciones de cifrado.
- Incorporación de modos de firma explícito e implícito en formato CMS.
- Añadida compatibilidad entre CMS y PKCS#7.
- Añadidos demostraciones de uso.

Versión 1.2

- Incorporación de los formatos de firma XMLDSignature y XAdES v1.1.1 como plugin.
- Mejoras en el módulo de cifrado. Incorporación de claves alfanuméricas no seguras.
- Corrección de bug de pérdida de foco en diálogos Java.
- Añadida la capacidad de introducir atributos firmados y no firmados.
- Añadidos ejemplos.
- Mejorada la instalación e integración de plugins.

Versión 2.1

- Adaptación del formato de firma XADES a estándar 1.3.2.
- Nueva distribución de elementos de firma XML.
- Corrección de bugs relativos a representación gráfica en Linux.
- Corrección de bugs de generación de firmas XML.
- Mejora del sistema de firmado en bloque.

Versión 2.3

- Corrección de bug de interpretación del árbol de firmas CMS.
- Corrección de contrafirma y cofirma CMS.
- Añadida capacidad de acceso a ficheros remotos.

Versión 2.3.1

- Corrección de bug relativo al filtrado de los certificados.

Versión 2.3.2

- Posibilidad de parametrización de la aparición o no de la ventana que informa al usuario acerca del hash que va a proceder a firmar.

- Aparición del disclaimer del instalador en negrita.

Versión 2.3.3

- Modificación de la DLL y del cliente para adaptación a formato propietario de tarjetas SIEMENS, que no utilizan como alias de certificados cadenas con codificación UTF-8

Versión 2.3.5

- Compatibilidad con Windows Vista e Internet Explorer 7.

- Testeada compatibilidad con Red Hat v4

- Incorporación del formato de firma CADES-BES.

- Adición de funcionalidad para el formato de firma XAdES (detached, enveloped y enveloping, incorporación de mime types)

- Descifrado de sobres digitales.

- Incorporación del sistema de distribuciones, mediante el cual pueden coexistir diferentes versiones del cliente instaladas en la misma máquina del usuario.

- Se robustece el sistema de instalación y los mensajes de error al usuario.

- Se solventa el problema de Mozilla Firefox cuando el cliente está publicado en entorno seguro HTTPS.

- Se ha modificado el mecanismo de carga de DLLs dinámicas, que corrige el problema de múltiples instancias del cliente en el mismo navegador.