Manual del integrador del MiniApplet del Cliente @firma

Índice de contenidos

1 Introducción

El MiniApplet @firma es una herramienta de firma electrónica que funciona en forma de *applet* de Java integrado en una página Web mediante JavaScript.

El Cliente hace uso de los certificados digitales X.509v3 y de las claves privadas asociadas a estos que estén instalados en el repositorio o almacén de claves y certificados (*KeyStore*) del sistema operativo o del navegador Web (Internet Explorer, Mozilla Firefox, etc.), así como de los que estén en dispositivos (tarjetas inteligentes, dispositivos USB) configurados en el mismo (como por ejemplo, el DNI Electrónico o DNIe).

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

El MiniApplet @firma es Software Libre publicado que se puede usar, a su elección, bajo licencia GNU General Public License versión 2 (GPLv2) o superior o bajo licencia European Software License 1.1 (EUPL 1.1) o superior.

Puede consultar la información relativa al proyecto Cliente @firma, dentro del cual se encuentra el MiniApplet @firma y descargar los fuentes de la aplicación en la siguiente dirección Web:

http://forja-ctt.administracionelectronica.gob.es/web/inicio

Tanto los binarios como los ficheros fuente empaquetados pueden descargarse desde:

http://administracionelectronica.gob.es/es/ctt/clienteafirma

1.1 Productos de terceros incluidos con el MiniApplet @firma

El MiniApplet @firma incluye, entre otros, los siguientes productos de terceros:

- JXAdES (https://forja.uji.es/)
- BouncyCastle (http://www.bouncycastle.org/)
- Código derivado de iText v2.1.7 (http://itextpdf.com/)
- Código derivado de Apache ORO (http://jakarta.apache.org/oro/)
- Código derivado de Java Mime Magic Library (http://jmimemagic.sourceforge.net/)
- Código derivado de .beID (http://eid.belgium.be/)
- Código derivado de OpenJDK (http://openjdk.java.net/)

2 Requisitos mínimos

2.1 Entorno Cliente

Entorno de ejecución de Java:

- Java SE 6 Update 38 (1.6.0_38) o superior, en 32 (x86)
- Java SE 7 Update 10 (1.7.0_10) o superior en 32 (x86) o 64 (x64/AMD64) bits

Sistema operativo:

- Windows XP SP3, Vista SP2, 7 SP1, 8 o superior, en 32 (x86) o 64 (x64) bits
- Windows Server 2003 R2 SP2 o superior, en 32 (x86) o 64 (x64) bits
- Linux 2.6 o superior (soporte prestado para Ubuntu y Guadalinex) , en 32 (x86) o 64 (x64/AMD64) bits
- Mac OS X Snow Leopard (10.6.8 o superior), Lion (10.7.2 o superior) o Mountain Lion (10.8.1 o superior)

Navegador Web:

- Mozilla Firefox 4.0 o superior
  - En Windows únicamente se soporta Firefox en 32 bits.
- Google Chrome 15 o superior
- Apple Safari 5 o superior
- Microsoft Internet Explorer 7 o superior, en 32 o 64 bits

Nota para usuarios de Firefox 9 o superior y Windows XP o Windows Server 2003: La carga del almacén de claves y certificados de Firefox 9 o superior por parte del MiniApplet @firma necesita que el sistema tenga instalado el entorno de ejecución redistribuible de Microsoft Visual C++ 2005. Si no consigue acceder a sus certificados y claves privadas desde el MiniApplet @firma, necesitará descargarlo e instalarlo manualmente. Este puede descargarse de:

- Windows XP y Windows Server 2003 en arquitectura x86:
  - http://www.microsoft.com/download/en/details.aspx?id=3387
- Windows XP y Windows Server 2003 en arquitectura x64:
  - http://www.microsoft.com/download/en/details.aspx?id=21254

El proceso de instalación puede requerir permisos de administrador.

2.1.1 Compatibilidad con Windows 8

El MiniApplet @firma no es compatible con Internet Explorer 10 en su versión Metro, y debe ser ejecutado con la versión de escritorio de Internet Explorer 10.

Para automatizar en cierta manera el cambio de Internet Explorer desde Metro hasta el escritorio clásico de Windows 8 se debe incluir la siguiente meta-información en la cabecera de la página HTML:

```
<meta http-equiv="X-UA-Compatible" content="requiresActiveX=true"/>
```

Puede encontrar más información sobre complementos de navegador (*plugins*) en Internet Explorer 10 sobre Metro en Windows 8 en:

- http://msdn.microsoft.com/en-us/library/ie/hh968248%28v=vs.85%29.aspx

2.1.2 Compatibilidad con Mac OS X

2.1.2.1 Java en versiones posteriores a la actualización de Apple 2012-006

La actualización 2012-006 de Apple Java para OS X deshabilita por completo la ejecución de *applets* de Java y aplicaciones Java WebStart en navegadores Web, introduciendo una incompatibilidad total con el Cliente @firma.

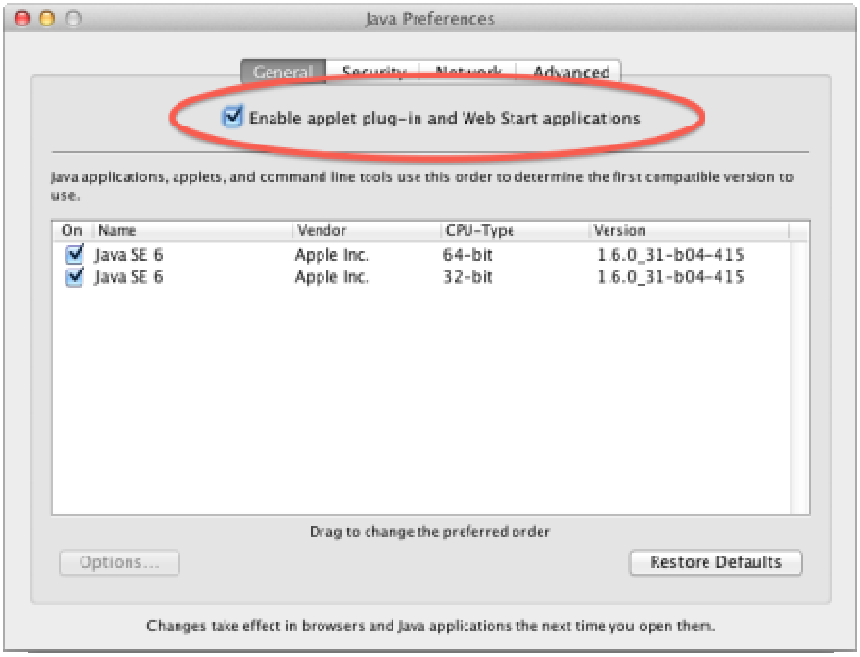
Puede solventar este inconveniente de dos formas alternativas:

1. Volviendo a habilitar manualmente el soporte de *applets* de Java y aplicaciones Java WebStart siguiendo las instrucciones descritas en la siguiente página Web: http://support.apple.com/kb/HT5559
2. Instalando Oracle JRE 7 para Mac OS X
    a. Es importante tener en cuenta que Oracle JRE 7 es incompatible con las versiones de 32 bits del navegador Web Google Chrome (las únicas actualmente disponibles).

2.1.2.2 Java en versiones posteriores a la actualización de Apple 2012-003

Por defecto, tras instalar la actualización de Java 2012-003 de Apple, Mac OS X no permite la ejecución de *applets* o aplicaciones Java Web Start, lo cual provoca que el MiniApplet @firma no funcione.

Para habilitar los *applets* de Java y las aplicaciones Web Start en Mac OS X es necesario indicarlo desde el panel de “Preferencias de Java” dentro de las Preferencias generales de Mac OS X y marcar la casilla “Activar módulo de Applet y aplicaciones Web Start”.

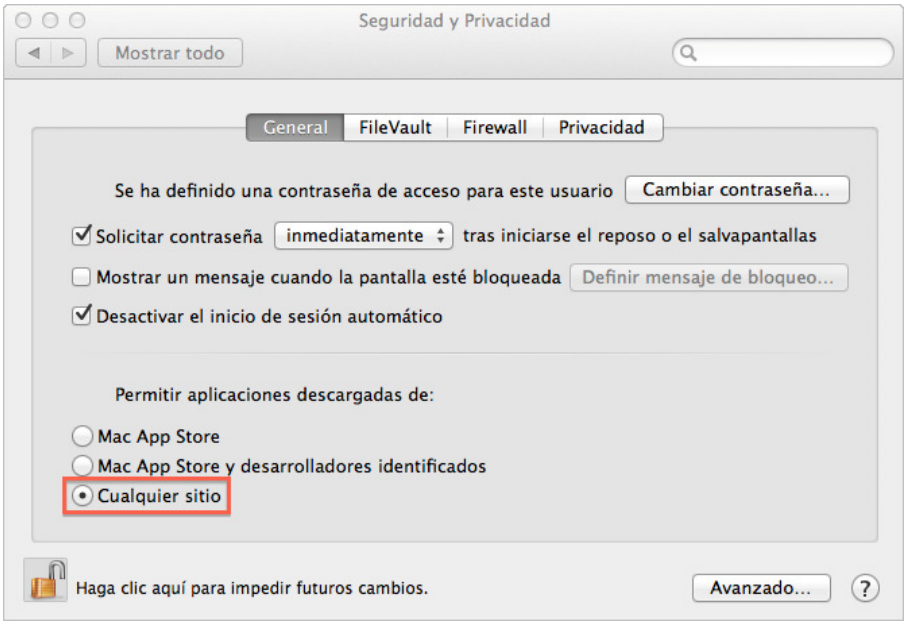


Como medida de seguridad, si el usuario no ejecuta *applets* de Java por un periodo de tiempo prolongado, Mac OS X deshabilita automáticamente la ejecución de *applets* y aplicaciones Java Web Start, por lo que será necesario comprobar que esta ejecución está permitida antes de iniciar el MiniApplet Cliente @firma, independientemente de si esta ejecución ya fue habilitada anteriormente.

2.1.2.3 Mountain Lion (10.8.1 y superiores)

Mac OS X Mountain Lion introduce, como medida de seguridad una restricción a la ejecución de aplicaciones descargadas a través de Internet, como son los *applets* de Java.

Por defecto, Mac OS X no permite esta ejecución a menos las aplicaciones se hayan descargado a través de la Apple Mac App Store (o eventualmente que el desarrollador que firma la aplicación esté autorizado por la propia Apple). Para permitir la ejecución del *applet* @firma descargado desde una página Web normal, es necesario indicarlo mediante la opción de Seguridad y Privacidad (dentro de Preferencias) de Mac OS X marcando la opción “Permitir aplicaciones descargadas de: Cualquier sitio”.



2.2 Entorno Servidor

- Servidor de aplicaciones JEE compatible con Servlets de Java.
  - Apache Tomcat, Oracle GlassFish, RedHat JBoss, IBM WebSphere, Oracle Application Server, etc.
- JRE 1.6 o superior (recomendado JRE 1.6.0_30 o JRE 7.0.1_01).

Es posible realizar un despliegue del MiniApplet @firma en un sistema servidor que cuente únicamente con un servidor Web (como Apache Web Server) o con un servidor de aplicaciones no compatible con JEE (como Microsoft Internet Information Server, IIS).

3 Funcionalidad proporcionada por el MiniApplet @firma

El MiniApplet @firma proporciona únicamente funcionalidades de firma electrónica (incluyendo multifirmas) sobre un conjunto selecto de formatos de firma, pero no otras funcionalidades como sobres digitales o cifrados simétricos. Adicionalmente, se proporciona un conjunto reducido de métodos de utilidad y opciones de operación.

Si necesita un formato de firma no soportado por el MiniApplet o una funcionalidad no soportada por este, consulte el catálogo de aplicaciones @firma para determinar cuál es la más apropiada para sus necesidades.

3.1 Formatos de firma soportados por el MiniApplet @firma

3.1.1 CAdES (CMS Advanced Electronic Signature)

Formato avanzado de firma binaria. Se soportan las siguientes variantes de CAdES:

- CAdES-BES
- CAdES-EPES

En cualquier caso, es posible siempre incluir el contenido firmado en la propia firma (firma implícita) o excluirlo (firma explícita). Por defecto, no se incluyen en la firma ni los datos firmados, ni política de firma alguna. Es decir, cuando no se introduce ninguna configuración, se generan firmas CAdES-BES explícitas.

Consulte el apartado Información específica para firmas CAdES para más información sobre la configuración de las firmas CAdES.

3.1.2 XAdES (XML Advanced Electronic Signature)

Se soportan las siguientes variantes de XAdES:

- XAdES-BES
- XAdES-EPES

Con independencia de la variante, es posible realizar las firmas XAdES en tres diferentes modos:

- Enveloping
- Enveloped
- Internally Detached

Consulte el apartado Información específica para firmas XAdES para más información sobre los modos de firma XAdES y otros condicionantes de uso.

Por defecto, cuando no se indican parámetros de configuración adicionales, el MiniApplet genera firmas XAdES-BES enveloping.

3.1.3 FacturaE (Factura electrónica)

Formato para la firma de facturas electrónicas conforme a la especificación 3.1 del estándar.

Al configurar este formato se establecen todas sus propiedades obligatorias, como la política de firma, por lo que no deberán establecerse manualmente.

El formato de factura electrónica no permite cofirma ni contrafirmar las facturas.

Consulte el apartado Información específica para firma de facturas electrónicas para más información sobre las opciones de configuración de las facturas electrónicas.

3.1.4 PAdES (PDF Advanced Electronic Signature)

Formato de firma avanzada de documentos PDF. Se soportan las siguientes variantes de PAdES:

- PAdES-BES

Una salvedad en la realización de firmas PAdES es que no se soporta la firma de ficheros adjuntos o empotrados en los documentos PDF.

Adicionalmente, el MiniApplet @firma impone una limitación en el tamaño de las firmas generadas, por lo que el uso combinado de cadenas de certificados muy largas y sellos de tiempo puede exceder este tamaño límite y provocar un fallo en el proceso de firma.

Consulte el apartado Información específica para firmas PAdES para más información sobre los modos de firma PAdES y otros condicionantes de uso.

Por defecto, el MiniApplet genera firmas PAdES-BES.

3.2 Formatos de firma no soportados por el MiniApplet @firma

El MiniApplet @firma no soporta la realización de firmas en formato CMS/PKCS#7, el formato XMLDSig (XML Digital Signature), ni los formatos de ODF y OOXML para la firma de documentos ofimáticos.

Se recomienda sustituir las firmas CMS por firmas CAdES, ya que este último proporciona compatibilidad hacia atrás con CMS.

Se recomienda sustituir las firmas XMLDSig por firmas XAdES, ya que este último proporciona compatibilidad hacia atrás con XMLDSig.

Si necesita utilizar firmas CMS o XMLDSig de un modo en el que el uso de sus homólogos AdES (Advanced Digital Signature) no es posible, puede utilizar el *applet* completo de @firma. Más información en http://administracionelectronica.gob.es/es/ctt/clienteafirma

3.3 Uso de certificados y claves privadas por parte del MiniApplet @firma

El MiniApplet @firma utiliza un mecanismo automático para determinar cuál será el almacén de certificados y claves que debe usar en cada caso.

La norma general sigue este simple algoritmo:

1. Si el navegador Web es Mozilla Firefox, con independencia del sistema operativo, se usa el almacén propio de Mozilla Firefox (NSS, *Netscape Security Services*) más los módulos PKCS#11 (*Public Key Cryptography Specification number 11*) que Firefox tuviese configurados, como tarjetas inteligentes, DNIe (Documento Nacional de Identidad electrónico), HSM (*Hardware Security Module*), etc.
2. Si el navegador es cualquier otro (Internet Explorer, Opera, Chrome o Safari), se usa el almacén predeterminado del sistema operativo:
    a. Windows: CAPI (*Cryptography Application Programming Interface*)
    b. Mac OS X: Llavero de Mac OS X
    c. Linux: NSS

Adicionalmente, es posible forzar al MiniApplet para que ignore estas reglas y utilizar un almacén fijo. En este caso, los almacenes soportados son:

- PKCS#12 (*Public Key Cryptography Specification number 12*) / PFX (*Personal File Exchange*).
- CAPI (únicamente en sistemas Windows).
- Llavero de Mac OS X, tanto el común del sistema como un llavero independiente en fichero (únicamente en sistemas Mac OS X).
- NSS (requiere que esté instalado Mozilla Firefox).
- PKCS11 (*Public Key Cryptography Specification number 11*).

3.3.1 Establecimiento manual del almacén de certificados y claves

Es posible para el integrador seleccionar manualmente el almacén de certificados de usuario que se debe utilizar, independientemente del sistema operativo o el navegador que utilice el usuario. Sin embargo, hay que tener en cuenta que si se selecciona un almacén no disponible en el entorno del usuario, el MiniApplet dará error al intentar recuperar los certificados del almacén.

La selección manual del almacén de claves se realiza durante la carga del MiniApplet y se mantiene durante toda su ejecución. Una vez cargado, no es posible cambiar de almacén de certificados.

Para forzar el uso de un almacén de certificados concreto pasaremos como segundo parámetro del método de carga (`cargarMiniApplet(codebase, keystore)`, consulte la sección de despliegue del MiniApplet para mayor información sobre este método) el tipo de almacén que se desea utilizar. El tipo se indicará mediante las variables JavaScript dedicadas a tal fin (declaradas en la biblioteca "`miniapplet.js`", que debe estar importada en las páginas Web en cualquier caso). Estas variables son:

- `KEYSTORE_WINDOWS`
  - Almacén de certificados CAPI. Compatible únicamente con sistemas Microsoft Windows.
- `KEYSTORE_APPLE`
  - Llavero de Mac OS X. Compatible únicamente con sistemas Apple Mac OS X.
- `KEYSTORE_FIREFOX`
  - Almacén NSS (Mozilla Firefox, Mozilla Thunderbird, etc.).
- `KEYSTORE_PKCS12`
  - Almacén en fichero PKCS#12 / PFX (*Personal File Exchange*).
- `KEYSTORE_PKCS11`
  - Almacén de claves compatible PKCS#11 (tarjeta inteligentes, aceleradora criptográfica…).

Determinados tipos de almacén permiten indicar el fichero o biblioteca en disco asociado al almacén. Este fichero o biblioteca debe indicarse mediante su ruta absoluta en el sistema del usuario, como parte del mismo parámetro, a continuación del tipo de almacén y separados por signo dos puntos (':'), siguiendo el patrón:

```
TIPO_ALMACEN:RUTA_ALMACEN
```

Los almacenes que permiten indicar el fichero o biblioteca que se debe utilizar son:

- `KEYSTORE_APPLE`
  - Permite indicar el llavero en fichero de tipo llavero en el que se encuentran los certificados de firma.
  - Si no se indica ningún fichero se usa el llavero general del sistema.
- `KEYSTORE_PKCS12`
  - Permite indicar el almacén en fichero de tipo PKCS#12/PFX (normalmente con extensiones .p12 o .pfx) en el que se encuentran los certificados de firma.
  - Si no se indica ningún fichero el MiniApplet solicitará al usuario que seleccione uno mediante un diálogo gráfico.

- `KEYSTORE_PKCS11`
  - Permite indicar la biblioteca que se debe utilizar para acceder al dispositivo que almacena los certificados de firma.
  - Si no se indica ningún fichero el MiniApplet solicitará al usuario que seleccione uno mediante un diálogo gráfico.
  - Es importante reseñar que la biblioteca PKCS#11 es dependiente del sistema operativo y de su arquitectura, por lo que si se indica, por ejemplo, una biblioteca PKCS#11 como una DLL (*Dynamic Link* Library) de 32 bits, no funcionará ni en Linux ni en Mac OS X, pero tampoco en Windows 64 bits si se usa el MiniApplet desde un navegador de 64 bits.

A continuación se listan algunos ejemplos de parámetro que se pueden pasar al método:

- `KEYSTORE_WINDOWS`
  - Configura CAPI (almacén general de claves y certificados de Windows)
- `KEYSTORE_APPLE`
  - Configura el llavero de Mac OS X del sistema
- `KEYSTORE_APPLE + ":/Users/usuario/Library/Keychains/login.keychain"`
  - Configura el llavero `/Users/usuario/Library/Keychains/login.keychain`
- `KEYSTORE_PKCS12`
  - Configura un almacén PKCS#12 que se solicitará al usuario mediante un diálogo gráfico de selección.
- `KEYSTORE_PKCS12 + ":C:\\prueba\\almacen.p12"`
  - Configura el almacén PKCS#12 `C:\\prueba\\almacen.p12`

Así, para cargar el aplicativo MiniApplet indicando manualmente el almacén de certificados que se desea utilizar, utilizaríamos sentencias JavaScript del tipo:

```
MiniApplet.cargarMiniApplet(codeBase, KEYSTORE_WINDOWS);
MiniApplet.cargarMiniApplet(codeBase, KEYSTORE_FIREFOX);
MiniApplet.cargarMiniApplet(codeBase, KEYSTORE_PKCS12 +
":C:\\prueba\\almacen.p12");
MiniApplet.cargarMiniApplet(codeBase, KEYSTORE_PKCS11 +
":C:\\Windows\\System32\\PkcsV2GK.dll");
```

Tenga en cuenta que no todos los almacenes de certificados están disponibles en todos los sistemas. Así pues:

- CAPI sólo está disponible en sistemas Microsoft Windows.
- El llavero de Mac OS X sólo está disponible en sistemas Apple Mac OS X.
- NSS sólo está disponible cuando un producto Mozilla compatible, como Mozilla Firefox o Mozilla ThunderBird, está instalado en el sistema y es de la misma arquitectura (x86, x64, IA64, etc.) que el navegador que utilice el usuario para acceder al MiniApplet.
- Si en un almacén se indica un fichero o biblioteca asociado, sólo estará disponible si este fichero o biblioteca puede encontrarse en la ruta indicada y el usuario dispone de permisos de lectura y ejecución sobre él.
- Un almacén PKCS#11 concreto sólo estará disponible si el fichero o biblioteca puede encontrarse en la ruta indicada, el usuario dispone de permisos de lectura y ejecución sobre él y además es para el mismo sistema operativo y arquitectura que la del navegador que utilice el usuario para acceder al MiniApplet.

4 Despliegue del MiniApplet @firma

Para el uso del MiniApplet son necesarios 3 ficheros principalmente:

- `miniapplet-full.jar`
  - Es el archivo Java en el que se encuentra el *applet* y toda la funcionalidad de la aplicación.
- `miniapplet.js`
  - Es la biblioteca JavaScript que permite la integración y uso del MiniApplet @firma en una página Web.
- `deployJava.js`
  - Es el *Oracle Java Deployment Toolkit*. Una biblioteca JavaScript que proporciona funcionalidades de carga y despliegue de *applets* de Java.
  - Por defecto, esta biblioteca se encuentra en la dirección http://www.java.com/js/deployJava.js, aunque puede descargarla y publicar una copia local si desea usar el MiniApplet @firma en una Intranet o WAN sin acceso a Internet.

Para el despliegue del Cliente debe publicar estos ficheros en su servidor Web (en el caso de `deployJava.js` puede utilizar la copia remota). Una vez desplegados bastará con importar las bibliotecas JavaScript en su página web y cargar el *applet*.

4.1 Importación de las bibliotecas JavaScript

Para poder integrar el MiniApplet en su página Web debe importar en ella las bibliotecas JavaScript de despliegue. Cada una de estas bibliotecas puede estar situada en una dirección distinta, por lo que es importante tener cuidado indicando las URL:

- `miniapplet.js`
  - Su situación depende de la dirección de publicación de su servidor. Puede hacer referencia a ella mediante una URL absoluta o mediante una URL relativa a partir de la dirección de publicación de su página Web.
- `deployJava.js`
  - Oracle pone a disposición de todos los integradores de *applets* una copia optimizada en tamaño de esta biblioteca en la siguiente URL:
    - http://www.java.com/js/deployJava.js
  - Si por cualquier motivo desea hacer un uso del MiniApplet @firma desde su Intranet o WAN y no desea que las páginas accedan directamente a Internet, puede descargar la biblioteca JavaScript y publicarla localmente en su servidor Web o gestor de contenidos.

Una vez determinada la URL de cada una de las bibliotecas, las páginas Web que hagan uso del MiniApplet @firma deben importarlas de forma global a toda la página, por ejemplo, incluyendo las

sentencias JavaScript de importación de bibliotecas en la sección `head` del HTML, tal y como se muestra en el siguiente ejemplo:

```
…
<head>
  <script type="text/javascript" src="http://www.java.com/js/deployJava.js">
  </script>
  <script type="text/javascript" src="http://miweb.com/afirma/miniapplet.js">
  </script>
…
```

En este ejemplo se han usado las siguientes direcciones para cada una de las bibliotecas JavaScript:

- `deployJava.js:` http://www.java.com/js/deployJava.js
- `miniapplet.js:` http://miweb.com/afirma/miniapplet.js

Si la página Web en la que deseamos cargar el Cliente estuviese también en la ruta "http://miweb.com/afirma" se podría hacer referencia a la biblioteca "miniapplet.js" de forma relativa indicando:

```
…
<script type="text/javascript" src="miniapplet.js"></script>
…
```

Cualquier página Web con estas dos bibliotecas JavaScript importadas está lista para cargar el MiniApplet @firma e incorporar la lógica de negocio (JavaScript) asociada a su uso.

Las operaciones que se desean realizar con el MiniApplet se ejecutarán a partir del objeto "`MiniApplet`" definido en `miniapplet.js`. Por ejemplo:

```
MiniApplet.sign(…);
```

4.1.1 Importación en páginas Web generadas dinámicamente

En un sistema Web actual, lo habitual es que las páginas Web no residan pre-construidas en directorios Web, sino que estas se generen dinámicamente ("al vuelo") mediante alguna de las muchas tecnologías disponibles de aplicaciones Web (JSP, ASP, PHP, etc.).

En estos casos es necesario tener en cuenta que debe indicarse la localización de las bibliotecas JavaScript de despliegue mediante una URL absoluta. Este aspecto es igualmente válido para los casos en los que, aun usando páginas Web estáticas, no todos los ficheros de despliegue residen en el mismo directorio.

4.2 Carga del MiniApplet

Una vez tenemos el MiniApplet desplegado y las bibliotecas JavaScript importadas en la página Web, la carga del MiniApplet se puede realizar mediante una simple sentencia JavaScript:

```
cargarMiniApplet(codebase, keystore)
```

Este método recibe dos parámetros:

- `codebase`
  - Debe indicarse la URL de despliegue de la aplicación JEE de @firma
- `keystore` (opcional)
  - Establece el almacén de claves y certificados alternativo indicando su tipo y su fichero asociado si lo hubiese.
  - Consulte con la sección Establecimiento manual del almacén de certificados y claves para más información sobre su formato y uso.

La invocación JavaScript a este método debe realizarse a través del objeto `MiniApplet` y atendiendo a las siguientes observaciones:

- Debe realizarse siempre fuera de una etiqueta HTML (nunca entre dos marcas "<" y ">", por ejemplo, en eventos tipo `onClick` u `onLoad` de JavaScript), y dentro de la sección (que no etiqueta) `body` de HTML.
- Ciertos diálogos gráficos (selección de ficheros, mensajes, etc.) se mostrarán en pantalla centrados respecto a la situación en pantalla de la propia sentencia JavaScript de carga, por lo que puede ser interesante el uso de técnicas HTML y JavaScript para asegurar que la sentencia de carga queda centrada respecto a la parte visible en el navegador de la página HTML, y así garantizar una mejor experiencia de usuario.
- Desde el momento de la invocación hasta la completa carga del MiniApplet pueden pasar unos segundos, y dependiendo del equipo del usuario y su conexión de red, hasta más de un minuto. Intente que la llamada a la sentencia de carga se realice de forma temprana respecto a la carga del resto de componentes de la página para evitar problemas de invocaciones e innecesarias esperas para los usuarios.

A continuación se muestran diferentes ejemplos de carga del MiniApplet @firma:

- Carga el MiniApplet desplegado en la dirección http://www.miweb.com/afirma.

```
<script type="text/javascript">
      MiniApplet.cargarMiniApplet("http://www.miweb.com/afirma");
</script>
```

- Carga el MiniApplet desplegado en la dirección http://www.miweb.com/afirma, pero usando siempre el llavero de Mac OS X como almacén de claves y certificados.

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

```
        MiniApplet.cargarMiniApplet("http://www.miweb.com/afirma",
             KEYSTORE_APPLE);
</script>
```

- Carga el MiniApplet desplegado en la dirección http://www.miweb.com/afirma, pero usando siempre el fichero `c:\store.pfx` (tipo PKCS#12 / PFX) como almacén de claves y certificados. El MiniApplet solicitará posteriormente la contraseña de este almacén al usuario mediante un diálogo gráfico.

```
<script type="text/javascript">
       MiniApplet.cargarMiniApplet(
          "http://www.miweb.com/afirma",
          KEYSTORE_PKCS12 + ":C:\\store.pfx"
       );
</script>
```

4.3 Firma del JAR del MiniApplet

Para la correcta ejecución de la aplicación MiniApplet @firma es necesario que el JAR del *applet* publicado esté correctamente firmado.

El Ministerio de Hacienda y Administraciones Públicas (MINHAP) distribuye bajo acuerdo versiones firmadas (por el propio MINHAP) de cada uno de estos ficheros, pero en caso de no disponer de estas versiones firmadas o si ha necesitado realizar alguna modificación en el MiniApplet y empaquetar su propia versión, deberá firmar usted mismo el archivo JAR del MiniApplet.

Para la firma del fichero JAR se recomienda el uso de la herramienta "Oracle JAR Signing and Verification Tool" (jarsigner) según las instrucciones descritas en la siguiente página Web:

- http://docs.oracle.com/javase/tutorial/deployment/jar/signing.html

Puede obtener esta herramienta de forma gratuita junto al Oracle Java Development Kit (JDK):

- http://www.oracle.com/technetwork/java/javase/

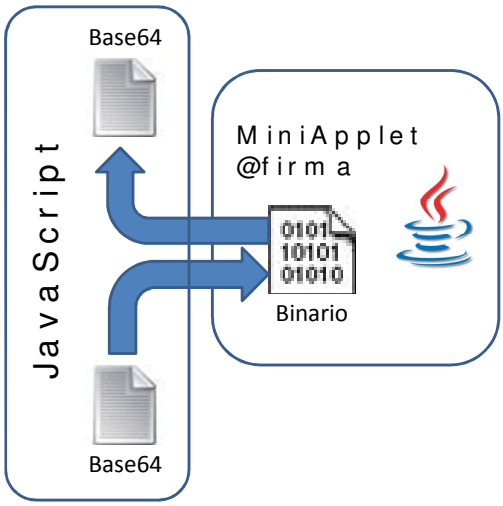
5 El proceso de firma electrónica

Cualquier firma electrónica realizada mediante el MiniApplet sigue un proceso simple de 4 pasos.

5.1 Selección del contenido a firmar

5.1.1 La conversión de datos a Base64

Lo primero que debemos saber antes de trabajar con datos en el MiniApplet @firma es que estos siempre se transfieren en formato Base64, tanto desde el *applet* al JavaScript de la página Web como en sentido inverso, pero el MiniApplet internamente trabaja con la descodificación binaria de estos datos en Base64.



Esto significa que los datos firmados realmente no serán los que nosotros proporcionemos en Base64, sino su descodificación binaria. Así, por ejemplo, si la cadena de texto `SOY UN TEXTO A FIRMAR` tiene como representación en base64 la cadena `U09ZIFVOIFRFWFRPIEEgRklSTUFS`, debemos establecer como datos a firmar siempre `U09ZIFVOIFRFWFRPIEEgRklSTUFS`, pero lo que se firmará será `SOY UN TEXTO A FIRMAR`.

Esta forma de operar permite trabajar sin ambigüedades tanto con textos en diferentes codificaciones (`ASCII`, `UTF-8`, `ISO-8859-1`, etc.) como con datos binarios de cualquier tipo.

El MiniApplet cuenta con dos métodos de utilidad para codificar y descodificar datos en Base64, `getTextFromBase64()` y `getBase64FromText()`. En ambos casos es necesario indicar el juego de caracteres (codificación) del texto para evitar problemas en las conversiones. Si desconoce o

prefiere que se detecte automáticamente la codificación, utilice alguno de los parámetros especiales que se definen en los apartados Conversión de una cadena Base64 a texto y Conversión de un texto a cadena Base64.

5.1.2 Selección de contenido desde ficheros en disco

En muchas ocasiones, el contenido a firmar no es directamente accesible desde JavaScript para ser firmado, y es necesario acceder al disco del usuario para recuperarlo desde un fichero.

Para estas ocasiones, el MiniApplet @firma permite que en las operaciones de firma, cofirma y contrafirma no se especifiquen los datos que se quieren firmar o las firmas que se quieren multifirmar. Cuando no se indican, el *applet* permite al usuario cargar un fichero en disco desde el que cargar estos datos.

Alternativamente, aunque no se recomendable ya que puede producir problemas al cargar ficheros grandes, el MiniApplet cuenta con ciertos métodos de utilidad que permiten la lectura de uno o varios ficheros (`getFileNameContentBase64()` y `getMultiFileNameContentBase64()`). Estos métodos muestran al usuario un diálogo de selección con el que seleccionar los ficheros y devuelven siempre el nombre y contenido de los ficheros en Base64, con independencia del contenido estos. Consulte el apartado de Firmas/Multifirmas masivas

El MiniApplet @firma puede utilizarse para la realización de múltiples firmas de tal forma que un usuario lo perciba como una única operación. Para ello basta que el integrador utilice los métodos de firma, cofirma y contrafirma sobre todos los datos que corresponda.

Para posibilitar que el usuario sólo deba seleccionar el certificado de firma en una ocasión y no para operación individual, se deberá dejar prefijado este certificado mediante el método:

```
function setStickySignatory (sticky);
```

En esta función:

- *`sticky`*: Es un booleano. Si se indica el valor `true`, el próximo certificado que seleccione el usuario (durante la próxima operación de firma/multifirma) quedará fijado y se utilizará para todas las operaciones posteriores. Si se indica el valor `false`, se libera el certificado y se volverá a solicitar al usuario en cada una de las siguientes operaciones.

Este método no devuelve nada.

Adicionalmente, para la generación de multifirmas dentro de un procedimiento masivo es interesante indicar el valor "`AUTO`" como formato de firma. Al hacerlo, las cofirmas y contrafirmas se realizarán en el mismo formato que la firma sobre la que se opera.

Gestión de ficheros para más detalles.

El uso de los métodos `getFileNameContentBase64()` y `getMultiFileNameContentBase64()` rompe la compatibilidad del despliegue con el Cliente @firma móvil.

5.1.3 Selección de objetivo de las contrafirmas

El caso de las contrafirmas es especial en el sentido que lo datos proporcionados no son realmente lo que queremos firmar, sino que son una firma electrónica la cual queremos contrafirmar.

En este caso, la firma ha de proporcionarse de igual forma que si se tratase de los datos reales a firmar (con las mismas consideraciones respecto a la codificación Base64).

Para determinar si debe contrafirmar todo el árbol de firmas o solo los nodos "hoja" deben realizarse indicaciones mediante parámetros adicionales. Consulte el apartado Selección de los nodos que se desean contrafirmar para aprender a configurar los nodos objetivo de la contrafirma.

5.2 Selección del certificado y clave privada de firma

Una vez tenemos establecido en JavaScript el contenido que queremos firmar, el siguiente paso es la elección de la clave privada y su certificado asociado con los que queremos realizar la firma electrónica.

Por motivos de seguridad, el integrador no puede obtener la lista de certificados y claves instaladas en el equipo del usuario, y es siempre responsabilidad es este último su selección, pero lo que si puede el integrador es restringir los posibles certificados a usar en base a una serie de criterios predefinidos, en lo que el MiniApplet concreta como Filtros de certificados.

Una vez se ha aplicado el filtro de certificados, en caso de haberlo establecido, se mostrará al usuario el diálogo de selección con el listado filtrado de certificados. También es posible realizar una selección automática de certificado cuando sólo haya uno seleccionable. Consulte el apartado Selección automática de certificados para saber el modo de configurar esta opción.

En el caso de que ningún certificado cumpla con los criterios de filtrado se producirá una situación de excepción.

5.3 Firma

El MiniApplet cuenta con tres métodos independientes para cada uno de las operaciones soportadas (firma, cofirma y contrafirma). A estos métodos, además de los datos a firmar y filtros de certificados comentados anteriormente es necesario indicar el formato (PAdES, CAdES, XAdES u FacturaE), el algoritmo de firma y otros parámetros adicionales que pudiesen ser necesarios. En el caso de la cofirma y contrafirma, puede utilizarse el formato "AUTO" para indicar que debe realizarse una firma con el mismo formato que la firma original.

Los métodos de firma, cofirma y contrafirma devuelven siempre las firmas codificadas en Base64.

5.4 Recogida de resultados

Cuando desde JavaScript se recibe el resultado de la firma en Base64 lo más común es optar por una entre dos opciones: Guardar los resultados en un archivo en el almacenamiento local del usuario o enviarlos a un servidor.

En el primer caso el MiniApplet @firma proporciona un método de utilidad (`saveDataToFile()`) para guardar resultados en disco. En este caso es siempre el usuario el que elige nombre del fichero y directorio de destino por motivos de seguridad, y debemos acordarnos que aunque le proporcionemos los datos en Base64, lo que se almacenará en el fichero es la descodificación binaria de estos.

Si optamos por enviarlos a un servidor, lo más usual es hacerlo directamente en Base64 y, posteriormente, mediante una aplicación en servidor independiente del MiniApplet, realizar las transformaciones y descodificaciones pertinentes.

6 Funciones disponibles del MiniApplet @firma

El MiniApplet @firma expone una serie de funcionalidades a través de JavaScript que permiten realizar la mayoría de las acciones relativas a las firmas electrónicas en entornos corporativos o de administración electrónica. Estas funcionalidades están divididas en distintos apartados según su tipología:

- Firma electrónica
- Firmas electrónicas múltiples
  - Cofirma
  - Contrafirma
- Configuración
  - Bloqueo del certificado de firma (para procesos masivos).
- Gestión de ficheros
  - Guardado de datos en disco
  - Selección de un fichero por parte del usuario y recuperación de su nombre y contenido.
  - Selección de múltiples ficheros por parte del usuario y recuperación de sus nombres y contenidos.
- Utilidad
  - Obtención de los mensajes de error
  - Conversión de una cadena Base64 a texto.
  - Conversión de un texto a una cadena Base64.

6.1 Uso del API desde JavaScript

El API del MiniApplet @firma se expone automáticamente al entorno JavaScript al importar la biblioteca `miniapplet.js` y está operativo pasados unos segundos desde la invocación al método de carga. Debido a este ligero retraso desde la invocación al método de carga y la finalización de la propia carga (y por lo tanto de la completa operatividad del API), es recomendable que la llamada al método de carga se realice automáticamente (como se ha indicado en el apartado Carga del MiniApplet), pero que el inicio de la lógica de firma dependa de una interacción del usuario con la página Web (como pulsar un botón), así el propio tiempo de reacción del usuario ante el interfaz gráfico permite cargar por completo el MiniApplet.

6.2 Obtención de resultados

Los métodos de operación criptográficos del Cliente obtienen como resultado la firma/multifirma generada. Esta firma puede obtenerse de dos formas: directamente como valor de retorno de las funciones o mediante funciones *callback* que establecen el comportamiento definido para procesar ese resultado. Los métodos del operación del Cliente son únicos, luego se usa un mecanismo u otro según si se han establecido los métodos de *callback* (modo asíncrono) o no (modo síncrono). Para utilizar el modo síncrono (devolución directa del resultado), basta con establecer los parámetros correspondientes de cada función a `null` o, directamente, omitirlos en la llamada.

6.2.1 Obtención directa de resultados

La obtención de forma asíncrona de los resultados es la que se ha utilizado hasta ahora y se hereda del *applet* Cliente @firma. Este modo simplifica a los programadores organizar a ejecución de las operaciones de una forma secuencial, que viene a ser el modo común de uso, y facilita la migración desde el *applet* Cliente @firma al MiniApplet.

Sin embargo, este modo no disfruta de las ventajas que se consiguen mediante el modo de operación asíncrono que se obtiene mediante el uso de *callbacks* (véase el apartado "6.2.2 Obtención de resultados mediante *Callbacks*"), que es además el que permite que nuestros desarrollos sea compatibles con plataformas móviles.

Mientras que este modo de uso hace que el resultado de la operación se obtenga como valor devuelto por la función, los errores pueden detectarse mediante la captura de excepciones. La identificación de estos errores se realizará mediante los métodos `getErrorType()` y `getErrorMessage()`.

Un ejemplo de operación en el que se obtiene de forma síncrona el resultado de las operaciones de firma es:

```
…
  // Llamamos a la operación de firma
  var signature;
  try {
       signature = MiniApplet.sign(null, "SHA512withRSA", "PAdES", null);
  }
  catch (e) {
       // Mostramos un mensaje con el error obtenido
       document.getElementById("resultMessage").innerHTML = "Error" + message;
  }

  // Guardamos la firma en un campo de un formulario
  document.getElementById("result").value = signature;
  // Mostramos un mensaje con el resultado de la operación
  document.getElementById("resultMessage").innerHTML = "La firma finalizó
correctamente";
…
```

6.2.2 Obtención de resultados mediante *Callbacks*

Los métodos de firma, cofirma y contrafirma del MiniApplet se ejecutan de forma asíncrona cuando se establece al menos una de las funciones *callback* para el tratamiento de los resultados. Al hacerlo de esta manera, que es la recomendada, se evita que el script quede bloqueado durante su ejecución y el navegador lo detecte como un funcionamiento anómalo e intente bloquearlo. Así mismo, el uso de este mecanismo permite que nuestro despliegue sea compatible con el Cliente de firma móvil.

Para poder operar con este funcionamiento asíncrono se ha dispuesto el sistema de *callbacks*. Estas *callbacks* son también funciones que definen que se debe hacer con el resultado de la operación. Por ejemplo:

- Podemos definir que muestre en un campo de un formulario un mensaje indicando que la operación ha terminado correctamente.
- Podemos guardar la firma en disco mediante el método proporcionado por el propio MiniApplet.
- Podemos adjuntarla a un campo oculto de un formulario y enviarlo.
- Podemos hacer que se cofirme el resultado de una firma.
- Etc.

Estas *callbacks* se definen como funciones normales JavaScript y pueden servir para 2 propósitos:

- Gestionar el resultado de una operación firma, cofirma o contrafirma.
- Gestionar el error devuelto por una operación firma, cofirma o contrafirma.

La función que gestiona el resultado correcto de las operaciones criptográficas recibe siempre un único parámetro que será el resultado devuelto por la operación (la firma, cofirma o contrafirma generada en base64). Esta puede tener la forma:

```
function successCallback(resultBase64) {
    …
}
```

La función que gestiona los casos de error obtendrá siempre 2 parámetros que definen el tipo de error producido (la clase de excepción cualificada que produjo el error) y el mensaje de error asociado. Esta función puede ser de la forma:

```
function errorCallback(type, message) {
    …
}
```

Para tratar el resultado de las firmas, cofirmas y contrafirmas mediante estas funciones se les pasará el nombre de las funciones de gestión del resultado y los errores como penúltimo y último parámetro, respectivamente.

Por ejemplo:

```
…
// Función que se ejecutará si la firma termina correctamente
function saveSignatureFunction (signature) {
    // Guardamos la firma en un campo de un formulario
    document.getElementById("result").value = signature;
    // Mostramos un mensaje con el resultado de la operación
    document.getElementById("resultMessage").innerHTML = "La firma finalizó
        correctamente";
}
// Función que se ejecutará si el proceso de firma falla
function showErrorFunction (type, message) {
    // Mostramos un mensaje con el resultado de la operación
    document.getElementById("resultMessage").innerHTML = "Error" + message;
}
```

…

```
  // Llamamos a la operación de firma
  MiniApplet.sign(null, "SHA512withRSA", "PAdES", null, saveSignatureFuntion,
       showErrorFuntion);
…
```

6.3 Firma electrónica

Mediante la operación de firma electrónica podemos realizar la firma de los datos seleccionados por el integrador y por el usuario.

El resultado de esta operación se debe gestionar asíncronamente mediante funciones *callback*. Esto garantiza que el JavaScript de nuestra página no se queda bloqueado durante la operación, evitando molestos mensajes por parte del navegador Web. Este uso de la función de firma también garantiza la compatibilidad del despliegue con el Cliente Móvil.

Para ejecutar la operación de firma se utiliza la función JavaScript:

```
function sign(dataB64, algorithm, format, params, successCallback,
  errorCallback);
```

En esta función:

- `dataB64`: Datos en forma de cadena en Base64 que deseamos firmar. También puede no indicar la firma (usar `null`) para que se muestre al usuario un diálogo de carga de fichero. Si los datos que necesita firmar son un texto, cárguelo y conviértalo a base 64 tal como se indica en el apartado Conversión de un texto a cadena Base64.
- `algorithm`: Algoritmo de firma. Consulte el apartado "Algoritmos de firma" para conocer los algoritmos disponibles.
- `format`: Formato de firma. Consulte el apartado Formatos de firma soportados por el MiniApplet @firma para consultar aquellos disponibles.
- `params`: Parámetros adicionales para la configuración de la firma. Consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma para saber cómo realizar el paso de parámetros y el apartado de información específica del formato de firma que desee realizar para saber los parámetros soportados por el formato en cuestión.
- `successCallback`: Función JavaScript que se ejecutará una vez se obtenga el resultado de la operación de firma. Esta función recibirá como único parámetro la firma resultado. Si se omite este parámetro, o se establece a `null`, la firma resultado se obtendrá como valor de retorno de la función.
- `errorCallback`: Función JavaScript que se ejecutará cuando ocurra un error durante la operación de firma. Esta función recibirá dos parámetros que son el tipo y el mensaje de error. Si se omite este parámetro, o se establece a `null`, el error se obtendrá en forma de excepción. Consulte el apartado "7.5 Gestión de errores".

El resultado de esta operación puede obtenerse directamente o gestionarme mediante las funciones *callback*. Este resultado se obtiene codificado en base64.

6.3.1.1 Ejemplos:

A continuación se muestran distintos ejemplos relacionados con la función de firma electrónica:

6.3.1.1.1 Firma electrónica cargando datos desde un fichero:

```
…
  // Funcion que se ejecutara cuando la firma termine correctamente
  function saveSignatureFunction (signature) {
       MiniApplet.saveDataToFile(signature, “Guardar firma”, “firma.pdf”, “pdf”,
              “Adobe PDF”);
  }

  // Funcion que se ejecutara cuando el proceso de firma falle
  function showErrorFunction (type, message) {
       showError(message); // Funcion creada por el integrador para mostrar errores
  }
…

  // Llamamos a la operacion de firma
  MiniApplet.sign(null, “SHA512withRSA”, “PAdES”, null, saveSignatureFuntion,
       showErrorFuntion);
…
```

6.3.1.1.2 Firma electrónica de un texto:

```
…
  var text = “Hola Mundo!!”;
  var dataB64 = MiniApplet.getBase64FromText(text, “default”);

  MiniApplet.sign(dataB64, “SHA1withRSA”, “CAdES”, null, successFunction,
        errorFuntion);
…
```

6.3.1.1.3 Firma electrónica de un texto introducido por el usuario:

```
…
  var text = document.getElementById(“userText”).value;
  var dataB64 = MiniApplet.getBase64FromText(text, “auto”);

  MiniApplet.sign(dataB64, “SHA1withRSA”, “CAdES”, null, successFunction,
        errorFuntion);
…

<!-- Fragmento HTML con un campo de texto en donde el usuario puede insertar el
texto que desea firmar -->
…
<form>
  <textarea name="userText" cols="50" rows="5">Aquí el usuario puede insertar el
texto que quiera</textarea>
</form>
…
```

6.3.1.1.4 Firma electrónica permitiendo al usuario seleccionar parámetros de firma:

```
…
```

```
  var params = “expPolicy=FirmaAGE”;
  var modes = document.getElementsByName(“rbMode”);

  for (i = 0; i <modes.length; i++) {
    if (modes[i].checked) {
      params = params + “\nmode=” + modes[i].value;
      break;
    }
  }

  MiniApplet.sign(
       dataB64, “SHA1withRSA”, “CAdES”, params, successFunction, errorFunction
  );
…
```

```
<!-- Fragmento HTML con botones de radio para la selección del modo de firma -->
…
<form>
  <input type="radio" name="rbMode" value="explicit" checked="checked" />Explícita
<br/>
  <input type="radio" name="rbMode" value="implicit" />Implícita
</form>
…
```

6.4 Firmas electrónicas múltiples

En este apartado se engloban las operaciones que permiten agregar más de una firma electrónica a un documento. Existen dos tipos generales de firmas múltiples:

- Cofirmas. Permiten que varios individuos firmen el mismo documento.
- Contrafirmas: Permite que un firmante refrende una firma electrónica.

6.4.1 Cofirmas

Operación mediante la cual dos o más firmantes muestran su acuerdo con un documento o datos concretos. La cofirma consiste en agregar la información de firma de un firmante a una firma ya existente. Así, será necesario que una persona firme el documento generando así la información de firma y, posteriormente, el resto de los firmantes cofirmen la firma generada. Si la firma generada contenía los datos firmados no serán necesarios nuevamente los datos para la generación de las cofirmas. Un ejemplo de uso de este tipo de firmas es cuando varios individuos firman el mismo contrato como partes contratante y contratista, compuestas, tal vez, por varios individuos cada una teniendo que firmar todos ellos.

El resultado de esta operación se debe gestionar asíncronamente mediante funciones *callback*. Esto garantiza que el JavaScript de nuestra página no se queda bloqueado durante la operación, evitando molestos mensajes por parte del navegador Web. Este uso de la función de firma también garantiza la compatibilidad del despliegue con el Cliente Móvil.

La función JavaScript mediante la cual se realizan las cofirmas es:

```
function coSign(signB64, dataB64, algorithm, format, params,
     successCallback, errorCallback);
```

En esta función:

- *signB64*: Firma electrónica en forma de cadena en Base64 que deseamos cofirmar. Una firma en Base64 es el resultado obtenido por las operaciones de firma, cofirma y contrafirma. También puede no indicar la firma (usar `null`) para que se muestre al usuario un diálogo de carga de fichero.
- *dataB64*: Datos en forma de cadena en Base64 que firmamos originalmente. Puede ser nulo si la firma seleccionada contiene los datos firmados originalmente. Si los datos utilizados originalmente para la firma son un texto, cárguelo y conviértalo a base 64 tal como se indica en el apartado Conversión de un texto a cadena Base64.
- *algorithm*: Algoritmo de firma. Consulte el apartado "Algoritmos de firma" para conocer los algoritmos disponibles.
- `format`: Formato de firma. Consulte el apartado Formatos de firma soportados por el MiniApplet @firma para consultar aquellos disponibles. Si no conoce el formato de firma utilizado para la firma original, indique el valor "`AUTO`" para especificar que se utilice el mismo formato. Este valor sólo reproduce el formato, no las propiedades de la firma original. Por ejemplo, si cofirmásemos una firma XAdES-EPES, indicando "`AUTO`" como valor, agregaríamos a esta una cofirma XAdES-BES salvo que indicásemos a través del parámetro *params* la política de firma que queremos utilizar.
- *params*: Parámetros adicionales para la configuración de la cofirma. Si se introduce un nulo, se usará la configuración por defecto para el formato de firma establecido. Consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma para saber cómo realizar el paso de parámetros y el apartado de información específica del formato de firma que desee realizar para saber los parámetros soportados por el formato en cuestión.
- `successCallback`: Función JavaScript que se ejecutará una vez se obtenga el resultado de la operación de cofirma. Esta función recibirá como único parámetro la cofirma resultado. Esta función recibirá como único parámetro la firma resultado. Si se omite este parámetro, o se establece a `null`, la firma resultado se obtendrá como valor de retorno de la función.
- `errorCallback`: Función JavaScript que se ejecutará cuando ocurra un error durante la operación de cofirma. Esta función recibirá dos parámetros que son el tipo y el mensaje de error. Si se omite este parámetro, o se establece a `null`, el error se obtendrá en forma de excepción. Consulte el apartado "7.5 Gestión de errores".

El resultado de esta operación puede obtenerse directamente o gestionarme mediante las funciones *callback*. Este resultado se obtiene codificado en base64.

6.4.1.1 Ejemplos:

A continuación se muestran distintos ejemplos relacionados con la función de cofirma electrónica:

6.4.1.1.1 Cofirma electrónica cargando una firma desde un fichero sabiendo que esta contiene los datos firmados originalmente:

```
…
  MiniApplet.coSign(null, null, “SHA1withRSA”, “CAdES”, null, successCallback,
        errorCallback);
…
```

6.4.1.1.2 Cofirma electrónica del resultado de una firma:

```
…
  // Funcion que realiza la cofirma a partir de los datos de firma
  function cosignFunction (signatureB64) {
     MiniApplet.coSign(
         signatureB64, dataB64, “SHA1withRSA”, “XAdES”, null, saveDataFunction,
         showError
     );
  }
  …
  // Función que almacena los datos generados por la cofirma en el campo
  // “resultId” de un formulario y lo envia
  function saveDataFunction (cosignB64) {
       document.getElementById(“resultId”).value = cosignB64;
       document.getElementById(“formulario”).submmit();
  }
  …
  // Función para firmar datos. Si termina correctamente la operación de firma se
  // llama a la función “cosignFunction” con el resultado de la operación y, si
  // esta también termina correctamente, se llama a la función “saveDataFunction”
  // con el resultado de la cofirma. Si falla alguna de estas funciones se llama
  // al método “showError”
  function firmar(dataB64) {
       MiniApplet.sign(dataB64, “SHA1withRSA”, “XAdES”, “format=XAdES Detached”,
             cosignFunction, showError);
  }
…
```

6.4.1.1.3 Cofirma electrónica de otra operación de multifirma:

```
…
  function cosignAction (signatureB64) {
       MiniApplet.coSign(
             multiSignatureB64, dataB64, “SHA1withRSA”, “XAdES”, null,
             successCallback, errorCallback
       );
  };
…

  MiniApplet.cosign(signB64, “SHA1withRSA”, “XAdES”, null, cosignAction,
       errorCallback);

…
```

6.4.2 Contrafirmas

Operación mediante la cual una entidad refrenda la firma de otra. La contrafirma consiste en agregar una firma electrónica sobre otra para revalidarla. Ejemplos de uso de este tipo de firmas son cuando un notario confirma que las firmas de uno o más individuos son correctas y pertenecen realmente a estos, o cuando, debido a la caducidad de los certificados de firma, un usuario desea revalidar una firma realizada con un certificado antiguo o con un algoritmo de firma actualmente

inseguro. Para realizar una contrafirma no es necesario disponer del documento que se firmó originalmente.

No todos los formatos de firma permiten la contrafirma de ficheros. Consulte el manual del formato de firma de su interés para conocer si este soporta esta operación.

La función JavaScript mediante la cual se realizan las contrafirmas es:

```
function counterSign(signB64, algorithm, format, params, successCallback,
errorCallback);
```

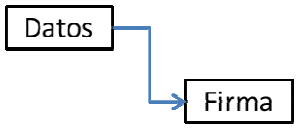
En esta función:

- *signB64*: Firma electrónica en forma de cadena en Base64 que deseamos contrafirmar. Una firma en Base64 es el resultado obtenido por las operaciones de firma, cofirma y contrafirma. También puede no indicar la firma (usar `null`) para que se muestre al usuario un diálogo de carga de fichero.
- *algorithm*: Algoritmo de firma. Consulte el apartado "Algoritmos de firma" para conocer los algoritmos disponibles.
- `format`: Formato de firma. Consulte el apartado Formatos de firma soportados por el MiniApplet @firma para consultar aquellos disponibles. Si no conoce el formato de firma utilizado para la firma original, indique el valor "`AUTO`" para especificar que se utilice el mismo formato. Por ejemplo, si contrafirmásemos una firma CAdES-EPES, indicando "`AUTO`" como valor, agregaríamos a esta una contrafirma CAdES-BES salvo que indicásemos a través del parámetro *params* la política de firma que queremos utilizar.
- *params*: Parámetros adicionales para la configuración de la contrafirma. Si se introduce un nulo, se usará la configuración por defecto para el formato de firma establecido. Consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma para saber cómo realizar el paso de parámetros y el apartado de información específica del formato de firma que desee realizar para saber los parámetros soportados por el formato en cuestión.
- `successCallback`: Función JavaScript que se ejecutará una vez se obtenga el resultado de la operación de contrafirma. Esta función recibirá como único parámetro la contrafirma resultado. Esta función recibirá como único parámetro la firma resultado. Si se omite este parámetro, o se establece a `null`, la firma resultado se obtendrá como valor de retorno de la función.
- `errorCallback`: Función JavaScript que se ejecutará cuando ocurra un error durante la operación de contrafirma. Esta función recibirá dos parámetros que son el tipo y el mensaje de error. Si se omite este parámetro, o se establece a `null`, el error se obtendrá en forma de excepción. Consulte el apartado "7.5 Gestión de errores".

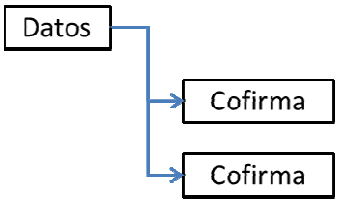
El resultado de esta operación puede obtenerse directamente o gestionarme mediante las funciones *callback*. Este resultado se obtiene codificado en base64.

6.4.2.1 Creación del árbol de firmas

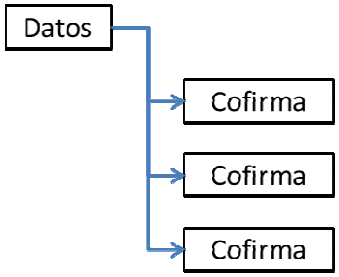
La operación de contrafirma se realiza sobre otra firma. Esta puede ser una firma simple, una cofirma u otra contrafirma. Estas operaciones de firma, cofirma y contrafirma van agregando firmas a un documento y, ya que las contrafirmas se realizan sobre firmas previas, se forma lo que se ha dado en llamar un árbol de firmas. Por ejemplo si realizamos una firma sobre unos datos obtendríamos la siguiente estructura:



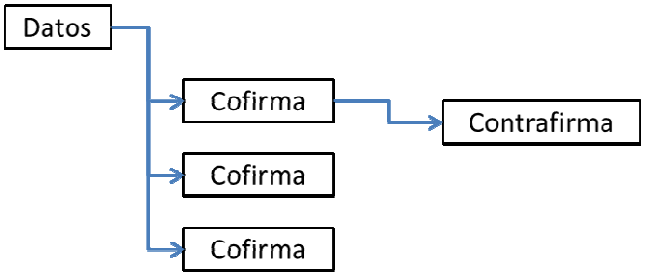
Si realizásemos una cofirma sobre el resultado de la operación anterior obtendríamos lo siguiente:



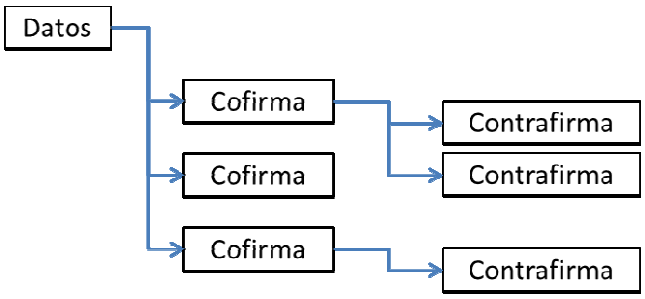
Tenga en cuenta que una firma y una cofirma están al mismo nivel y son equivalentes. No tiene importancia cual fue la primera en realizarse (firma) y cual la siguiente o siguientes porque todas son cofirmas. Las cofirmas son firmas sobre los datos originales, por lo tanto todas dependen de estos. Así, si agregamos una nueva cofirma quedaría de la siguiente forma:



Una contrafirma, en cambio se realiza sobre una firma previa, nunca sobre los datos. Por ejemplo:

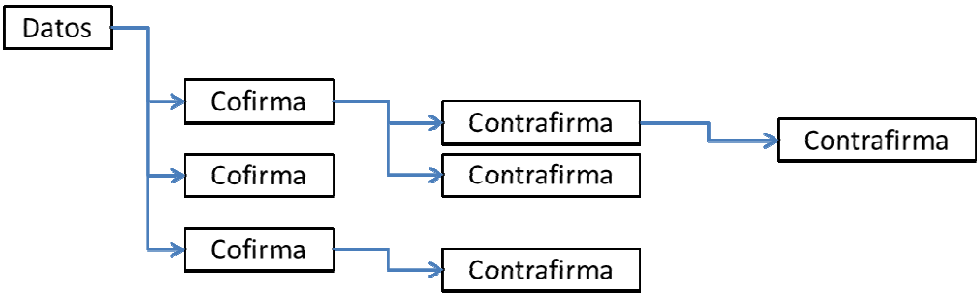


Una persona que contrafirme, puede estar interesada en contrafirmar más de una firma simultáneamente. Este sería el caso de un notario que valida con la suya las firmas de un las dos partes de un contrato. Esto se representa con una firma sobre cada una de las firmas que se contrafirmen, que no tienen por qué ser todas las del árbol de firmas. Por ejemplo, podemos contrafirmar la cofirma anteriormente contrafirmada y otra adicional, quedando así:



Dos contrafirmas situadas a un mismo nivel del árbol no son cofirmas, ni siquiera cuando dependen del mismo nodo. Simplemente, son contrafirmas del mismo nodo.

Siempre es posible seguir creando cofirmas y contrafirmas al árbol las cofirmas siempre dependerán de los datos y las contrafirmas de otro nodo de firma. Este nodo puede ser así una contrafirma, generando nuevos niveles en el árbol:



6.4.2.2 Selección de los nodos que se desean contrafirmar

En función de lo explicado en el apartado anterior, el MiniApplet @firma permite que las contrafirmas se realizan sobre los siguientes objetivos:

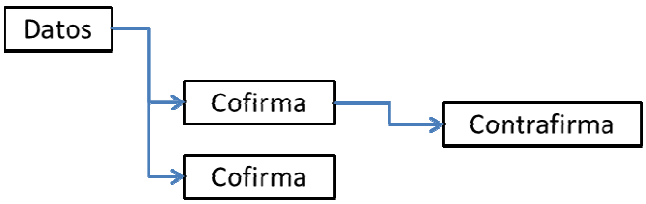
- **Nodos hoja del árbol** (`LEAFS`): Se firmarán sólo las firmas del árbol de los que no depende ningún otro nodo.
- **Todo el árbol de firma** (`TREE`): Se firman todos los nodos de firma del árbol.

La configuración de qué nodos se desean firmar se realiza a través del parámetro `params` de la función de contrafirma, al que, además de toda la configuración específica del formato de firma, el parámetro adicional `target` que indica los nodos a contrafirmar. Si desea conocer cómo utilizar el parámetro `params` para establecer una configuración, consulte el aparatado "Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma".

La clave `target` del `params` puede adoptar uno de los siguientes valores:

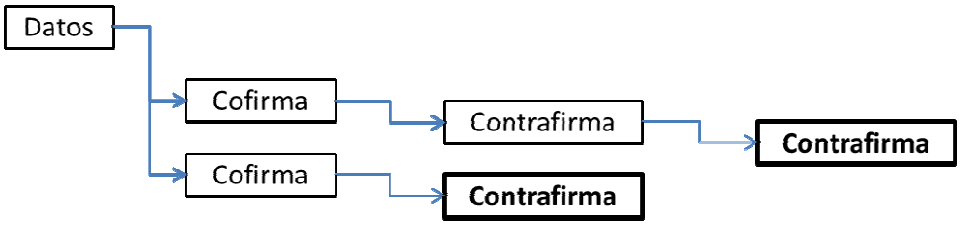
- `leafs`: Contrafirma todas las firmas que sean nodos hoja del árbol. Este es el valor por defecto.
- `tree`: Contrafirma todas las firmas del árbol.

Si, por ejemplo, disponemos del siguiente árbol de firmas:

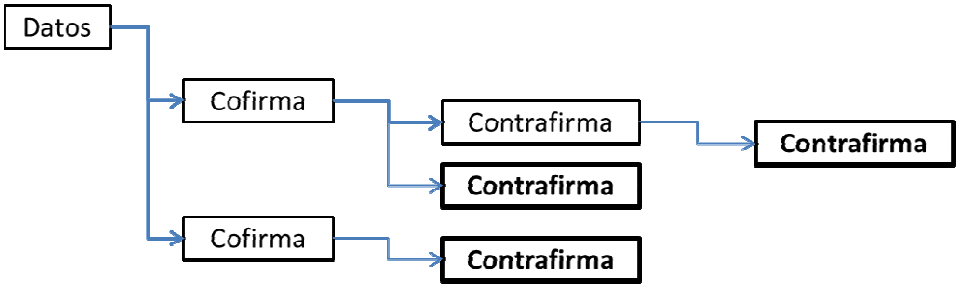


Cada una de las configuraciones dará el siguiente resultado:

- `target=leafs`



- `target=tree`



Si desea realizar contrafirmas más concretas que permitan seleccionar nodos o firmantes concretos del árbol de firmas, consulte el catálogo de aplicaciones @firma para determinar cuál es la más apropiada para sus necesidades.

6.4.2.3 Ejemplos:

A continuación se muestran distintos ejemplos relacionados con la función de contrafirma electrónica:

6.4.2.3.1 Contrafirma electrónica de una firma seleccionada por el usuario:

```
var filenameDataB64;
try {
     filenameDataB64 = MiniApplet.getFileNameContentBase64(
         “Fichero de firma”, “xsig”, “Firma XAdES”
     );
} catch (e) {
     return;
}

var signatureB64;
var separatorIdx = filenameDataB64.indexOf("|");
if ((separatorIdx + 1) < filenameDataB64.length) {
  signatureB64 = filenameDataB64.substring(separatorIdx + 1);
} else {
  /* El fichero no contenía datos */
  return;
}

MiniApplet.counterSign(
     signatureB64, “SHA1withRSA”, “XAdES”, null, successCalback, errorCallback
);
…
```

6.4.2.3.2 Contrafirma electrónica del resultado de una firma:

```
…
function counterSignCallback (signatureB64) {
     MiniApplet.counterSign(
           signatureB64, “SHA1withRSA”, “XAdES”, null, successCallback,
           errorCallback
     );
}
…
```

```
  MiniApplet.sign(dataB64, "SHA1withRSA", "XAdES", null, counterSignCallback,
       errorCallback);
…
```

6.4.2.3.3 Contrafirma electrónica de todo el árbol de firmas:

```
…
  MiniApplet.counterSign(
       signatureB64, "SHA1withRSA", "CAdES", "target=tree", successCallback,
       errorCallback
  );
…
```

6.5 Firmas/Multifirmas masivas

El MiniApplet @firma puede utilizarse para la realización de múltiples firmas de tal forma que un usuario lo perciba como una única operación. Para ello basta que el integrador utilice los métodos de firma, cofirma y contrafirma sobre todos los datos que corresponda.

Para posibilitar que el usuario sólo deba seleccionar el certificado de firma en una ocasión y no para operación individual, se deberá dejar prefijado este certificado mediante el método:

```
function setStickySignatory (sticky);
```

En esta función:

- *`sticky`*: Es un booleano. Si se indica el valor `true`, el próximo certificado que seleccione el usuario (durante la próxima operación de firma/multifirma) quedará fijado y se utilizará para todas las operaciones posteriores. Si se indica el valor `false`, se libera el certificado y se volverá a solicitar al usuario en cada una de las siguientes operaciones.

Este método no devuelve nada.

Adicionalmente, para la generación de multifirmas dentro de un procedimiento masivo es interesante indicar el valor "`AUTO`" como formato de firma. Al hacerlo, las cofirmas y contrafirmas se realizarán en el mismo formato que la firma sobre la que se opera.

6.6 Gestión de ficheros

Estos son método orientados al guardado o carga de ficheros en disco.

6.6.1 Guardado de datos en disco

El MiniApplet @firma permite almacenar datos en el equipo del usuario. Este método es útil para almacenar datos generados como parte de la operación del sistema o las propias firmas generadas por el MiniApplet.

El integrador puede seleccionar los datos que desea almacenar, la propuesta de nombre para el fichero y otros parámetros para el diálogo de guardado. Sin embargo, será el usuario el único que podrá decidir donde desea almacenar los datos y qué nombre tendrá el fichero.

Los datos guardados son los datos indicados en base64 ya descodificados. Es decir, si deseamos almacenar el texto "SOY UN TEXTO A FIRMAR", convertiremos este texto a Base 64 con lo que obtendríamos la cadena "U09ZIFVOIFRFWFRPIEEgRklSTUFS" y se la pasaríamos al método de guardado. Si abrimos el fichero resultante encontraremos que este contiene la cadena "SOY UN TEXTO A FIRMAR".

La función JavaScript para el guardado de datos en disco es:

```
function saveDataToFile(dataB64, title, fileName, extension, description);
```

En esta función:

- *dataB64*: Datos en forma de cadena en Base64 que deseamos almacenar. Comúnmente, esto será el resultado de una operación de firma o unos datos que se habrán procesado previamente para codificarlos a este formato.
- *title*: Título del diálogo de guardado.
- *fileName*: Propuesta de nombre de fichero.
- *extension*: Extensión de guardado propuesta. Los ficheros visibles del diálogo se filtrarán para sólo visualizar los que tienen esta extensión mientras esté seleccionada en el diálogo. Un ejemplo de extrensión es: *pdf*
- *description*: Descripción del tipo de fichero que se va a almacenar. Esta descripción aparecerá asociada a la extensión indicada.

Esta función devolverá *true* cuando los datos queden guardados correctamente. Si el usuario canceló la operación de guardado devolverá *false* y si se produjo algún error durante la operación de guardado se lanzará una excepción.

6.6.1.1 Ejemplos:

A continuación se muestran distintos ejemplos relacionados con el guardado de datos en disco:

6.6.1.1.1 Guardado de una firma electrónica generada por el MiniApplet:

```
…
  function saveDataCallback (dataB64) {
       MiniApplet.saveDataToFile(dataB64, "Guardar firma electrónica",
            "firma.csig", "csig", "Firma binaria");
  }
…
  MiniApplet.coSign(dataB64, "SHA1withRSA", "CAdES", null, saveDataCallback,
       errorCallback);
…
```

6.6.1.1.2 Guardado de datos insertados por el usuario:

```
…
  var text = document.getElementById("userText").value;
  var dataB64 = MiniApplet.getBase64FromText(text, "auto");
```

```
  MiniApplet.saveDataToFile(dataB64, "Guardar", "fichero.txt", "txt", "Texto
      plano");
…

<!-- Fragmento HTML con un campo de texto en donde el usuario puede insertar el
texto que desea guardar -->
…
<form>
  <textarea name="userText" cols="50" rows="5">Aquí el usuario puede insertar el
texto que desee guardar</textarea>
</form>
…
```

NOTA: Este método guarda los datos descodificados, no en base 64, por lo que en este ejemplo se guardaría un fichero de texto con el texto en claro insertado por el usuario.

6.6.2 Selección de un fichero por parte del usuario y recuperación de su nombre y contenido

El MiniApplet @firma permite que el usuario seleccione un fichero de su sistema local y recuperar del mismo su nombre y contenido. Este método nos permite cargar ficheros para firmarlos, multifirmarlos u operar con ellos de cualquier otro modo.

La función JavaScript para el guardado de datos en disco es:

```
function getFileNameContentBase64(title, extensions, description);
```

En esta función:

- *title*: Título del diálogo de selección.
- *extensions*: Listado de extensiones de fichero permitidas. Estas aparecerán separadas por una coma (‘,’) y sin espacios entre ellas. Por ejemplo: `pdf,jpg,txt`. El diálogo sólo mostrará los ficheros con estas extensiones, salvo que el usuario establezca lo contrario en el diálogo.
- *description*: Descripción del tipo de fichero que se espera cargar. Esta descripción aparecerá asociada a las extensiones indicadas.

Este método devuelve el nombre del fichero seleccionado seguido del contenido del mismo en base 64, separados por el carácter “|”. Si el usuario cancelase el diálogo de selección de fichero devolvería *null* y, en caso de producirse un error durante la operación de carga o conversión a base 64, se lanzaría una excepción.

Por ejemplo, si se cargase el fichero `entrada.txt` que contiene el texto “`SOY UN TEXTO A FIRMAR`”, (“`U09ZIFVOIFRFWFRPIEEgRklSTUFS`” si lo codificamos en base 64) el método devolvería el texto “`entrada.txt|U09ZIFVOIFRFWFRPIEEgRklSTUFS`”.

Advertencia: El uso de este método hace que el despliegue del MiniApplet no sea compatible con el Cliente Móvil. En su lugar, no indique los datos que desea firma, cofirma o contrafirmar y el MiniApplet permitirá al usuario cargar un fichero de datos/firma sobre el que operar.

6.6.2.1 Ejemplos:

6.6.2.1.1 Carga de un fichero y recogida de su nombre

```
…
  var fileNameContentB64;
  try {
       fileNameContentB64 = MiniApplet.getFileNameContentBase64(
           “Seleccionar fichero”, “jpg,gif,png”, “Imagen”
       );
  } catch (e) {
       return;
  }

  var filename = fileNameContentB64.substring(0, fileNameContentB64.indexOf(“|”));
…
```

6.6.2.1.2 Carga y firma de un fichero

```
…

  var fileNameContentB64;
  try {
       fileNameContentB64 = MiniApplet.getFileNameContentBase64(
           “Seleccionar fichero”, “pdf”, “Adobe PDF”
       );
  } catch (e) {
       return;
  }

  var separatorIdx = fileNameContentB64.indexOf(“|”);
  var filename = fileNameContentB64.substring(0, separatorIdx);
  var dataB64;
  if ((separatorIdx + 1) < fileNameContentB64.length) {
    dataB64 = fileNameContentB64.substring(separatorIdx + 1);
  } else {
    /* El fichero no contenía datos */
    return;
  }

  MiniApplet.sign(dataB64, “SHA1withRSA”, “CAdES”, null, successCallback,
       errorCallback);
…
```

6.6.3 Selección de múltiples ficheros por parte del usuario y recuperación de sus nombres y contenidos

El MiniApplet @firma permite que el usuario seleccione una serie de ficheros de su sistema local y recuperar el nombre y contenido de los mismos. Este método nos permite cargar ficheros para firmarlos, multifirmarlos u operar con ellos de cualquier otro modo.

La función JavaScript para el guardado de datos en disco es:

```
function getMultiFileNameContentBase64(title, extensions, description);
```

En esta función:

- *title*: Título del diálogo de selección.
- *extensions*: Listado de extensiones de fichero permitidas. Estas aparecerán separadas por una coma (‘,’) y sin espacios entre ellas. Por ejemplo: pdf, jpg, txt. El diálogo sólo mostrará los ficheros con estas extensiones, salvo que el usuario establezca lo contrario en el diálogo.
- *description*: Descripción del tipo de fichero que se espera cargar. Esta descripción aparecerá asociada a las extensiones indicadas.

Este método devuelve una cadena con el patrón:

Nombre1|Contenido1|Nombre2|Contenido2|…NombreX|ContenidoX

Esto es el listado de nombres y contenidos de fichero concatenados todos con el carácter ‘|’. Un Nombre X se refiere al nombre de uno de los ficheros seleccionados mientras que el Contenido X se refiere al contenido de ese mismo fichero.

Por ejemplo, si se seleccionasen los ficheros `entrada.txt` e `imagen.jpg`, se obtendría la cadena de texto
“`entrada.txt|U09ZIFVOIFRFWFRPIEEgRklSTUFS|imagen.jpg|A1NSHA1NQW212ASYAN45YMD2MWQEI JHAUHS7SASSNH7DAUDASDA9844`”, en donde seguido del texto `entrada.txt` y limitado por los caracteres ‘|’ aparece el contenido de ese fichero convertido a base 64 y seguido de l texto `imagen.jpg` y separado por ‘|’ aparece el contenido de ese otro fichero.

Advertencia: El uso de este método hace que el despliegue del MiniApplet no sea compatible con el Cliente Móvil. En su lugar, no indique los datos que desea firma, cofirma o contrafirmar y el MiniApplet permitirá al usuario cargar un fichero de datos/firma sobre el que operar. No es posible la selección múltiple de ficheros mediante este mecanismo.

6.6.3.1 Ejemplos:

6.6.3.1.1 Carga de ficheros y recogida de sus nombres

```
…
  var fileNameContentB64;
  try {
       fileNameContentB64 = MiniApplet.getMultiFileNameContentBase64(
           “Seleccionar múltiples fichero”, “jpg,gif,png”, “Imagen”
       );
  } catch (e) {
       /* Si el error no se debe a la cancelación por el usuario, lo mostramos. */
       if (!“es.gob.afirma.core.AOCancelledOperationException”.equals(
             MiniApplet.getErrorType())) {
```

```
                    /* Método del integrador para mostrar logs */
                    showLog(MiniApplet.getErrorMessage());
        }
        return;
    }

    var i = 0;
    var separatorIdx1 = 0;
    var separatorIdx2 = -1;

    var filenames = new Array();
    do {
      separatorIdx2 = fileNameContentB64.indexOf("|", separatorIdx1);
      filenames[i] = fileNameContentB64.substring(separatorIdx1, separatorIdx2);
      separatorIdx1 = fileNameContentB64.indexOf("|", separatorIdx2 + 1) + 1;
      i++;
    } while (separatorIdx1 > 0);
…
```

6.6.3.1.2 Carga y firma de ficheros

```
…
    var fileNameContentB64;
    try {
        fileNameContentB64 = MiniApplet.getMultiFileNameContentBase64(
            "Seleccionar ficheros", "pdf", "Adobe PDF"
        );
    } catch (e) {
        /* Si el error no se debe a la cancelación por el usuario, lo mostramos. */
        if (!"es.gob.afirma.core.AOCancelledOperationException".equals(
                MiniApplet.getErrorType())) {
                    /* Método del integrador para mostrar logs */
                    showLog(MiniApplet.getErrorMessage());
        }
        return;
    }

    var separatorIdx1 = fileNameContentB64.indexOf("|") + 1;
    var separatorIdx2 = 0;
    var dataB64;
    while (separatorIdx2 > -1 && separatorIdx1 > 0 && separatorIdx1 <
          fileNameContentB64.length) {

      separatorIdx2 = fileNameContentB64.indexOf("|", separatorIdx1);
      if (separatorIdx2 == -1) {
        dataB64 = fileNameContentB64.substring(separatorIdx1);
      } else {
        dataB64 = fileNameContentB64.substring(separatorIdx1, separatorIdx2);
      }

      MiniApplet.sign(dataB64, "SHA1withRSA", "CAdES", null, successCallback,
            errorCallback);

      if (separatorIdx2 > -1) {
        separatorIdx1 = fileNameContentB64.indexOf("|", separatorIdx2 + 1) + 1;
      }
    }
…
```

6.7 Utilidad

6.7.1 Eco

El MiniApplet dispone de un método que únicamente muestra por consola el mensaje:

```
MiniApplet cargado y en ejecuci\u00F3n
```

El único propósito de este método es que los integradores puedan llamarlo para comprobar si el applet está correctamente cargado. En caso contrario, fallará su ejecución.

6.7.2 Obtención de los mensajes de error

Cuando un método del MiniApplet falla en su ejecución lanza una excepción y almacena el mensaje que describe el error. El integrador puede acceder a este mensaje e identificar el tipo de error o mostrárselo al usuario.

La función JavaScript para recuperar el mensaje de error producido por la aplicación es:

```
function getErrorMessage();
```

Este método devuelve el mensaje de error antecedido por el nombre cualificado de la excepción que lo produjo.

Para ver ejemplos del uso de este método y la gestión de errores del MiniApplet, consulte el apartado Gestión de errores.

6.7.3 Conversión de una cadena Base64 a texto

El MiniApplet @firma proporciona un método para la conversión de una cadena Base64 a un texto plano con una codificación concreta. Este método permite mostrar al usuario en texto plano información que se posea Base64. Casos en los que puede ser necesario esto son cuando se carga el contenido de un fichero de texto plano desde disco por medio de alguno de los métodos de carga o cuando los datos son el resultado de una firma XML, por ejemplo.

La función JavaScript para recuperar el texto plano correspondiente a la cadena en Base64 es:

```
function getTextFromBase64(dataB64, charset);
```

En esta función:

- *dataB64*: Son los datos Base64 que se quieren mostrar como texto plano.
- *charset*: Es el juego de caracteres que se debe utilizar para la conversión. Los más comunes son `utf-8`, `iso-8859-1` e `iso-8859-15`.
    - Si se desea utilizar el juego de caracteres por defecto, indique "`default`" o pase un valor nulo.
    - Si desea que se intente detectar el juego de caracteres automáticamente, indique "`auto`".

Este método devuelve una cadena con el texto plano correspondiente.

6.7.3.1 Ejemplos

6.7.3.1.1 Conversión de una firma XML generada previamente

```
…
  function showCallback (signatureB64) {
       var text = MiniApplet.getTextFromBase64(xmlSignB64, “utf-8”);
       alert(text);
  }
…
  MiniApplet.sign(dataB64, “SHA1withRSA”, “XAdES”, “format=XAdES Enveloped”,
        showTextCallback, errorCallback);
…
```

6.7.4 Conversión de un texto a cadena Base64

El MiniApplet @firma proporciona un método para la conversión de un texto plano con una codificación concreta a una cadena Base64. Este método permite pasar al método de firma un texto insertado por el usuario, por ejemplo.

La función JavaScript para convertir de texto plano a Base64 es:

```
function getBase64FromText(plaintText, charset);
```

En esta función:

- *plainText*: Es el texto plano que se desea convertir a Base64.
- *charset*: Es el juego de caracteres del texto plano. Los más comunes son `utf-8`, `iso-8859-1` e `iso-8859-15`.
    - Si se desea utilizar el juego de caracteres por defecto, indique “`default`” o pase un valor nulo.
    - Si desea que se intente detectar el juego de caracteres automáticamente, indique “`auto`”.

Este método devuelve una cadena Base64 que es la codificación del texto plano indicado.

6.7.4.1 Ejemplos

6.7.4.1.1 Firma de un texto insertado por el usuario

```
…
  var text = “Hola Mundo!!”;
  var dataB64 = MiniApplet.getBase64FromText(text, “utf-8”);

  MiniApplet.sign(dataB64, “SHA1withRSA”, “CAdES”, null, successCallback,
        errorCallback);
…
```

7 Configuración de las operaciones

7.1 Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma

Una peculiaridad del uso del API del MiniApplet desde JavaScript es que, para mejorar la fiabilidad, todos los pasos de valores de parámetros se realizan mediante cadenas de texto (los datos binarios se codifican en Base64 para poder transmitirlos como textos), pero en los métodos de firma, cofirma y contrafirma se acepta un tipo de parámetro (usualmente denominada `params`) que es una representación textual de una colección de propiedades Java (`Properties`).

Las propiedades establecidas mediante `params` tienen distintas utilidades según el formato de firma utilizado (indicar una variante del formato, especificar una política de firma, etc.), pero siempre siguen el mismo formato:

```
nombreParam1=valorParam1

nombreParam2=valorParam2

…
```

Y desde JavaScript deben concatenarse cada una de las líneas usando el carácter especial de nueva línea (`\n`) como separador:

```
var params='nombreParam1=valorParam1\nnombreParam2= valorParam2';
```

Es importante respetar el nombre original de las propiedades ya que puede existir diferenciación entre mayúsculas y minúsculas.

Consulte la documentación JavaDoc de cada formato de firma para más información sobre los parámetros aceptados por cada uno de ellos.

7.2 Selección automática de certificados

Para aquellos casos en los que sólo exista un certificado en el almacén de certificados al que se acceda o cuando se descarten certificados y sólo haya uno que es posible seleccionar, es posible indicar al MiniApplet que lo seleccione automáticamente en lugar de mostrar al usuario el diálogo de selección con este único certificado. Esto podemos configurarlo mediante la propiedad *headless*.

Si agregamos a la lista de propiedades que configuran las operaciones de firma, cofirma y contrafirma el parámetro *headless* con el valor *true*, el diálogo de selección no se mostrará al usuario cuando sólo haya un certificado disponible. En su lugar, se seleccionará automáticamente este certificado y se continuará con la operación. La línea que debería agregarse a la configuración es, por tanto:

headless=true

Por defecto, si no se establece la propiedad *headless* o se indica un valor distinto de true, se mostrará el diálogo de selección de certificados aun cuando sólo haya un certificado para seleccionar.

Para saber cómo establecer la propiedad *headless* en las operaciones de firma consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma.

7.3 Configuración del filtro de certificados

El MiniApplet @firma dispone de dos filtros de certificados que se pueden aplicar para la selección automática de un certificado de firma concreto. Los filtros en cuestión son:

- Filtro DNIe: Filtra los certificados del almacén para que sólo se muestren los certificados de firma de los DNIe disponibles desde ese almacén.
    - Para establecer este filtro de certificados se indicará la propiedad *filter* con el valor *dnie:* en el parámetro de configuración la operación de firma, cofirma o contrafirma. Esto es: *filter=dnie:*
- Filtro SSL: Filtra los certificados del almacén para que sólo se muestre aquellos con un número de serie concreto (comúnmente sólo será uno). Existe un caso especial. Si el número de serie resulta ser de un certificado de autenticación de un DNIe, se mostrará en su lugar el certificado de firma de ese mismo DNIe.
    - Para establecer este filtro de certificados se indicará la propiedad *filter* con el valor *ssl:*, seguido por el número de serie del certificado, en el parámetro de configuración la operación de firma, cofirma o contrafirma. Esto es: *filter=ssl:Nº_serie*. El número de serie se debe indicar en hexadecimal:
    - Ejemplos:
        - filter=ssl:45553a61
        - filter=ssl:03ea
- Filtro de certificados cualificados de firma: Filtra los certificados del almacén para que sólo se muestre aquellos con un número de serie concreto (comúnmente sólo será uno). En el caso de que este certificado no esté cualificado para firma, se buscará un certificado parejo que sí lo esté en el almacén. Si se encontrase se seleccionaría este nuevo certificado y, si no, se seleccionará el certificado al que corresponde el número de serie.
    - Para establecer este filtro de certificados se indicará la propiedad *filter* con el valor *qualified:*, seguido por el número de serie del certificado, en el parámetro de configuración la operación de firma, cofirma o contrafirma. Esto es: *filter=qualified:Nº_serie*. El número de serie se debe indicar en hexadecimal:
    - Ejemplos:
        - filter=*qualified*:45553a61
        - filter=*qualified*:03ea

Si se estable un filtro de certificados y ninguno de los certificados disponibles cumple con los criterios de filtrado, se lanzará una excepción indicando que no se ha encontrado ningún certificado que cumpla con los criterios indicados y se cancelará la operación.

Si más de un certificado cumpliesen los criterios de filtrado, se mostrarán estos en el diálogo de selección de certificados.

Si tan sólo un certificado cumple con las condiciones del filtro establecido y, al igual que ocurriría si sólo existiese un certificado en el almacén utilizado, se ha configurado la opción *headless* en las propiedades adicionales de la operación, se seleccionará automáticamente ese certificado sin mostrar el diálogo de selección al usuario. Consulte el apartado Selección automática de certificados para conocer cómo configurar la propiedad *headless*.

Para saber cómo establecer la configuración de los filtros de certificados en las operaciones de firma consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma.

7.4 Configuración de la política de firma

7.4.1 Configuración manual

La política de firma de una firma electrónica identifica diversos criterios que se han cumplido durante la construcción de esta firma o requisitos que cumple la propia firma. Esta política de una firma electrónica se identifica mediante varios atributos declarados en la firma. Todos los formatos avanzados de firma (CAdES y XAdES) tienen una variante EPES (*Explicit Poliyc-based Electronic Signature*) que declaran los atributos correspondientes a la política de firma.

El MiniApplet @firma permite la generación de firmas EPES (CAdES-EPES y XAdES-EPES) para lo cual es necesario indicar las propiedades de la política en el método de firma que se vaya a utilizar.

Consulte el apartado específico de configuración del formato de firma que desee utilizar para conocer las propiedades disponibles para la configuración de la política de firma.

Para saber cómo establecer estas las propiedades de firma, consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma.

Tenga en cuenta que el que una firma incluya los atributos correspondientes a una política de firma concreta no significa que cumpla los criterios de la política. Si desea que sus firmas se ajusten a una política de firma lea las restricciones impuestas por esa política y genere firmas acorde a ella antes de configurarla. De esta forma, podrá asegurarse de que sus firmas son compatibles con otros sistemas y entornos en los que se utilicen firmas acorde a la política en cuestión.

7.4.2 Política de firma de la AGE v1.8

En el MiniApplet @firma se ha incluido un mecanismo para la configuración rápida y sencilla de la política de firma de la Administración General del Estado (AGE) v1.8. Para configurar esta política concreta basta con indicar la siguiente propiedad en la operación de firma, cofirma o contrafirma, cuando se utilice un formato avanzado de firma (CAdES y XAdES).

- `expPolicy=FirmaAGE`

Esta propiedad se expandirá a las necesarias para el cumplimiento de la política de firma de la AGE, lo que equivale a introducir las propiedades:

- CAdES
  - `policyIdentifier=2.16.724.1.3.1.1.2.1.8`
  - `policyIdentifierHash=7SxX3erFuH31TvAw9LZ70N7p1vA=`
  - `policyIdentifierHashAlgorithm=1.3.14.3.2.26`
  - `policyQualifier=http://administracionelectronica.gob.es/es/ctt/politicafirma/politica_firma_AGE_v1_8.pdf`
  - `mode=implicit`
- XAdES
  - `policyIdentifier=urn:oid:2.16.724.1.3.1.1.2.1.8`
  - `policyIdentifierHash=V8lVVNGDCPen6VELRD1Ja8HARFk=`
  - `policyIdentifierHashAlgorithm=http://www.w3.org/2000/09/xmldsig#sha1`
  - `policyQualifier=http://administracionelectronica.gob.es/es/ctt/politicafirma/politica_firma_AGE_v1_8.pdf`
  - `format=XAdES Detached`
  - `mode=implicit`

La propiedad `format`, sólo se aplicará cuando el formato de firma sea XAdES.

La propiedad `mode`, sólo se aplicará cuando el formato de firma sea CAdES y los datos ocupen menos de 1 MegaByte. Con tamaños mayores de datos no se incluirán estos en las firmas.

Si se configura para la operación alguna propiedad individual que coincida con alguna de las propiedades a las que se expande la política, se ignorará esa propiedad individual y prevalecerá el valor impuesto por la política. Por ejemplo, si se configurasen las propiedades `expPolicy=FirmaAGE` y `format=XAdES Enveloping`, para una operación de firma con formato XAdES, se generaría una firma XAdES Detached con la política de firma de la AGE establecida. Es decir, se ignoraría que se estableció la propiedad `format=XAdES Enveloping`.

Para saber cómo configurar propiedades en las operaciones de firma, consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma.

Para más información sobre la política de firma de la AGE puede consultar la guía de implementación de la política que está en el área de descargas de la iniciativa Política de firma de la AGE en http://administracionelectronica.gob.es/es/ctt/politicafirma.

7.4.3 Política de firma de Factura electrónica (Facturae)

Para la firma de facturas electrónicas se deberá utilizar siempre el formato `FacturaE`. Este formato configura automáticamente las propiedades necesarias para la firma de facturas electrónicas, incluida la política de firma.

Las firmas generadas siempre son según la especificación 3.1 de factura electrónica.

Para saber cómo configurar propiedades en las operaciones de firma, consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma.

7.5 Gestión de errores

La gestión de errores del MiniApplet se basa en el sistema de captura de excepciones desde JavaScript, propagadas desde Java. Los métodos del *applet* lanzan excepciones cuando se produce algún error durante su ejecución, y es posible capturar estas excepciones desde JavaScript y ver su tipo y descripción gracias a los métodos del MiniApplet.

Estos métodos son:

- `getErrorType()`: Devuelve el nombre cualificado de la clase de excepción. Algunos ejemplos son:
  - `java.io.FileNotFoundException`: Lanzada cuando no se encuentra el fichero seleccionado por el usuario.
  - `es.gob.afirma.keystores.common.AOCertificatesNotFoundException`: Lanzada cuando no hay certificados en el almacén o ninguno pasa el filtro establecido.
  - `es.gob.afirma.core.AOCancelledOperationException`: Cuando el usuario ha cancelado alguno de los diálogos que se le mostró durante la operación (selección de fichero, selección de certificado, inserción de contraseña…).
- getErrorMessage(): Devuelve el mensaje asociado al error que se ha producido. Algunos ejemplos son:
  - `El sistema no puede encontrar el archivo especificado`
  - `El almacen no contenia entradas validas`
  - `Operación cancelada por el usuario`

Estos mensajes no están internacionalizados (siempre se muestran en castellano) y no contienen caracteres especiales (como las tildes españolas). Es recomendable, que el integrador identifique el tipo de errores que puede producir su despliegue y, cuando los detecte con getErrorType() actúe como corresponda mostrando un mensaje personalizado si fuese necesario.

A continuación se muestra un ejemplo simple de gestión de errores en el MiniApplet mediante JavaScript:

```
var DEBUG = true;
function firmar() {

  var signature;
  try {
    sign("Hola", "SHA1withRSA", "CAdES", null, saveDataCallback, errorCallback);
  } catch(e) {
    alert("Se produjo un error al ejecutar la operación de firma ");
    return;
  }
}

…

function saveSignatureCallback (signature)
  try {
    MiniApplet.saveDataToFile(signature, "Guardar firma", "firma.csig", null,
         null);
  } catch(e) {
    alert("Se produjo un error al ejecutar la operación de firma");
    return;
  }
}
```

Una forma de completar el ejemplo anterior sería mostrar al usuario más información acerca de la naturaleza del error, de forma que pueda intentar subsanarlo:

```
function showErrorCallback (errorType, errorMessage) {
  if (errorType().equals("java.io.FileNotFoundException")) {
    alert("Error: No se ha seleccionado un fichero de datos válido");
  }
  else if (errorType().equals(
      "es.gob.afirma.keystores.common.AOCertificatesNotFoundException") {
    alert("Error: No se ha encontrado ningún certificado de firma válido");
  }
  else {
    alert("Error: Se produjo un error durante la operación de firma");
  }
}
…
  var signature;
  try {
      MiniApplet.sign("Hola", "SHA1withRSA", "CAdES", null, sucessCallback,
           showErrorCallback);
  } catch(e) {
      showErrorCallback(MiniApplet.getErrorType(), MiniApplet.getErrorMessage());
  }
```

Revise la documentación JavaDoc de cada método del MiniApplet para comprobar que excepciones puede lanzar y el motivo de las mismas. Cualquier otra excepción no contemplada en la documentación JavaDoc se considera un error propio de la aplicación. Cuando el comportamiento de la herramienta difiera según el tipo de error, gestione siempre mediante la cláusula `else` los errores no documentados.

8 Información específica para firmas CAdES

8.1 Algoritmos de firma

Las firmas CAdES aceptan los siguientes algoritmos de firma (deben escribirse exactamente como aquí se muestran):

- `SHA1withRSA`
- `MD5withRSA`
- `MD2withRSA`
- `SHA256withRSA`
- `SHA384withRSA`
- `SHA512withRSA`

No es recomendable usar los algoritmos `MD5withRSA` y `MD2withRSA` por estar obsoletos y ser vulnerables. Por la misma razón, es igualmente conveniente evitar el algoritmo `SHA1withRSA`.

El algoritmo más seguro, y por lo tanto el recomendado para su uso es `SHA512withRSA`. Tenga en cuenta que si el usuario utiliza una versión de Java anterior a la recomendada en el apartado de requisitos mínimos del Entorno Cliente, es posible que no pueda generar firmas electrónicas con este algoritmo desde algunos almacenes de certificados.

8.2 Parámetros adicionales

A continuación se detallan los parámetros adicionales que aceptan cada una de las formas de generación de firmas.

Es posible que el uso de parámetros no contemplados en las siguientes tablas provoque otros cambios de funcionamiento. No obstante **no se dará soporte** al aplicativo si se usan parámetros no documentados, asumiendo el integrador todo el riesgo y responsabilidad derivados del uso de parámetros o valores distintos de los aquí descritos.

8.2.1 Firma y cofirma

Nombre del parámetro	Valores posibles	Descripción
`mode`	`explicit`	La firma resultante no incluirá los datos firmados. Si no se indica el parámetro `mode` se configura automáticamente este comportamiento.
	`implicit`	La firma resultante incluirá internamente una copia de los datos firmados. El uso de este valor podría generar firmas de gran tamaño.
`contentTypeOid`	OID	Identificador del tipo de dato firmado.
`contentDescription`	[Texto]	Descripción textual del tipo de datos firmado.
`policyIdentifier`	[OID o URN	Identificador de la política de firma,

	de tipo OID]	necesario para generar firmas CAdES-EPES.
`policyIdentifierHash`	[Valor en Base64]	Huella digital de la política de firma. Es obligatorio indicar este parámetro si de indicó también `policyIdentifier`, al igual que es obligatorio también dar valor al parámetro `policyIdentifierHashAlgorithm`.
`policyIdentifierHashAlgorithm`	`SHA1`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA1.
	`SHA-256`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-256.
	`SHA-384`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-284.
	`SHA-512`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-512.
`policyQualifier`	[URL hacia documento]	URL (universalmente accesible) hacia el documento (normalmente PDF) que contiene una descripción textual de la política de firma. Este parámetro es opcional incluso si se desea generar firmas CAdES-EPES.

8.2.2 Contrafirma

Nombre del parámetro	Valores posibles	Descripción
`policyIdentifier`	[OID o URN de tipo OID]	Identificador de la política de firma, necesario para generar firmas CAdES-EPES.
`policyIdentifierHash`	[Valor en Base64]	Huella digital de la política de firma. Es obligatorio indicar este parámetro si de indicó también `policyIdentifier`, al igual que es obligatorio también dar valor al parámetro `policyIdentifierHashAlgorithm`.
`policyIdentifierHashAlgorithm`	`SHA1`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA1.
	`SHA-256`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-256.
	`SHA-384`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-284.

	`SHA-512`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-512.
`policyQualifier`	[URL hacia documento]	URL (universalmente accesible) hacia el documento (normalmente PDF) que contiene una descripción textual de la política de firma. Este parámetro es opcional incluso si se desea generar firmas CAdES-EPES.

9 Información específica para firmas XAdES

9.1 Tratamiento de las hojas de estilo XSL de los XML a firmar

Cuando se firma o cofirma (no aplica a la contrafirma) un XML que contiene hojas de estilo, estas se firman igualmente (a menos que se indique lo contrario con el parámetro `ignoreStyleSheets`, descrito más adelante), cumpliendo las siguientes reglas:

- Firmas XML *Enveloped*
    - Hoja de estilo con ruta relativa
        - No se firma.
    - Hoja de estilo remota con ruta absoluta
        - Se restaura la declaración de hoja de estilo tal y como estaba en el XML original
        - Se firma una referencia (*canonizada*) a esta hoja remota
    - Hoja de estilo empotrada
        - Se restaura la declaración de hoja de estilo tal y como estaba en el XML original
- Firmas XML *Externally Detached*
    - Hoja de estilo con ruta relativa
        - No se firma.
    - Hoja de estilo remota con ruta absoluta
        - Se firma una referencia (*canonizada*) a esta hoja remota
    - Hoja de estilo empotrada
        - No es necesaria ninguna acción
- Firmas XML *Enveloping*
    - Hoja de estilo con ruta relativa
        - No se firma.
    - Hoja de estilo remota con ruta absoluta
        - Se firma una referencia (*canonizada*) a esta hoja remota
    - Hoja de estilo empotrada
        - No es necesaria ninguna acción

- Firmas XML *Internally Detached*
    - Hoja de estilo con ruta relativa
        - No se firma.
    - Hoja de estilo remota con ruta absoluta
        - Se firma una referencia (*canonizada*) a esta hoja remota
    - Hoja de estilo empotrada
        - No es necesaria ninguna acción

9.2 Algoritmos de firma

Las firmas XAdES aceptan los siguientes algoritmos de firma (deben escribirse exactamente como aquí se muestran):

- `SHA1withRSA`
- `SHA256withRSA`
- `SHA384withRSA`
- `SHA512withRSA`

No es recomendable usar el algoritmo `SHA1withRSA` por estar obsoleto y ser vulnerables.

El algoritmo más seguro, y por lo tanto el recomendado para su uso es `SHA512withRSA`. Tenga en cuenta que si el usuario utiliza una versión de Java anterior a la recomendada en el apartado de requisitos mínimos del Entorno Cliente, es posible que no pueda generar firmas electrónicas con este algoritmo desde algunos almacenes de certificados.

9.3 Parámetros adicionales

A continuación se detallan los parámetros adicionales que aceptan cada una de las formas de generación de firmas.

Es posible que el uso de parámetros no contemplados en las siguientes tablas provoque otros cambios de funcionamiento. No obstante **no se dará soporte** al aplicativo si se usan parámetros no documentados, asumiendo el integrador todo el riesgo y responsabilidad derivados del uso de parámetros o valores distintos de los aquí descritos.

9.3.1 Firma y cofirma

Nombre del parámetro	Valores posibles	Descripción
`format`	`XAdES Enveloping`	Genera firmas en formato *Enveloping*. Este es el formato que se utiliza por defecto cuando no se indica ninguno.
	`XAdES Enveloped`	Genera firmas en formato *Enveloped*.
	`XAdES Detached`	Genera firmas en formato *Internally Detached*.
`policyIdentifier`	[URL]	Identificador de la política de firma (normalmente una URL hacia la política en formato XML procesable), necesario para generar firmas XAdES-EPES.
`policyIdentifierHash`	[Valor en Base64]	Huella digital de la política de firma. Es obligatorio indicar este parámetro si el valor indicado en `policyIdentifier` no es universalmente accesible. Si se da valor a este parámetro es obligatorio también dar valor al parámetro `policyIdentifierHashAlgorithm`.
`policyIdentifierHashAlgorithm`	`SHA1`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA1.
	`SHA-256`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-256.
	`SHA-384`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-384.
	`SHA-512`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-512.
`policyQualifier`	[URL hacia documento]	URL (universalmente accesible) hacia el documento (normalmente PDF) que contiene una descripción textual de la política de firma. Este parámetro es opcional incluso si se desea generar firmas XAdES-EPES.
`policyDescription`	[Texto]	Descripción textual de la política de firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado. Este parámetro es opcional incluso si se desea generar firmas XAdES-EPES.

signerClaimedRole	[Texto]	Agrega a la firma un campo con el cargo atribuido al firmante. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
signerCertifiedRole	[Texto]	Agrega a la firma un campo con el cargo confirmado del firmante. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
signatureProductionCity	[Texto]	Agrega a la firma un campo con la ciudad en la que se realiza la firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
signatureProductionProvince	[Texto]	Agrega a la firma un campo con la provincia en la que se realiza la firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
signatureProductionPostalCode	[Texto]	Agrega a la firma un campo con el código postal en donde se realiza la firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
signatureProductionCountry	[Texto]	Agrega a la firma un campo con el país en el que se realiza la firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
referencesDigestMethod	http://www.w3.org/2000/09/xmldsig#sha1	Usa el algoritmo SHA1 para el cálculo de las huellas digitales de las referencias XML firmadas.
	http://www.w3.org/2001/04/xmlenc#sha256	Usa el algoritmo SHA-256 para el cálculo de las huellas digitales de las referencias XML firmadas. Este es el valor recomendado.
	http://www.w3.org/2001/04/xmlenc#sha512	Usa el algoritmo SHA-512 para el cálculo de las huellas digitales de las referencias XML firmadas.
	http://www.w3.org/2001/04/xmlenc#ripemd160	Usa el algoritmo RIPEMD160 para el cálculo de las huellas digitales de las referencias XML firmadas.
mimeType	[Texto en formato MIME-Type]	MIME-Type de los datos a firmar. Si no se indica este parámetro el sistema

		intenta auto-detectar el tipo, estableciendo el más aproximado (que puede no ser el estrictamente correcto).
encoding	[Texto]	Codificación de los datos a firmar. Un uso incorrecto de este parámetro puede provocar la generación de una firma inválida.
contentTypeOid	[OID o URN de tipo OID]	Identificador del tipo de dato firmado. Este parámetro es complementario (que no excluyente) al parámetro mimeType.
canonicalizationAlgorithm	http://www.w3.org/TR/2001/REC-xml-c14n-20010315	Se firma el XML con canonizado XML 1.0 inclusivo (valor por defecto).
	http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments	Se firma el XML con canonizado XML 1.0 inclusivo con comentarios.
	http://www.w3.org/2001/10/xml-exc-c14n#	Se firma el XML con canonizado XML 1.0 exclusivo.
	http://www.w3.org/2001/10/xml-exc-c14n#WithComments	Se firma el XML con canonizado XML 1.0 exclusivo con comentarios.
xadesNamespace	[URL]	URL de definición del espacio de nombres de XAdES (el uso de este parámetro puede condicionar la declaración de versión de XAdES).
ignoreStyleSheets	true	Si se firma un XML con hojas de estilo, ignora estas dejándolas sin firmar.
	false	Si se firma un XML con hojas de estilo, firma también las hojas de estilo (valor por defecto, consultar notas adicionales sobre firma de hojas de estilo).
avoidBase64Transforms	true	No declara transformaciones Base64 incluso si son necesarias.
	false	Declara las transformaciones Base64 cuando se han codificado internamente los datos a firmar en Base64 (valor por defecto).
headLess	true	Evita que se muestren diálogos gráficos adicionales al usuario (como por ejemplo, para la *dereferenciación* de hojas de estilo enlazadas con rutas relativas).

	`false`	Permite que se muestren diálogos gráficos adicionales al usuario.
`applySystemDate`	`true`	La firma incluirá un atributo con la fecha y hora de la firma, obtenidas del reloj del ordenador del usuario (valor por defecto).
	`false`	Se omite la inclusión de la fecha y hora de la firma como atributo de esta última.
`xmlTransforms`	[Número]	Número de transformaciones a aplicar al contenido firmado. Debe indicarse posteriormente igual número de parámetros `xmlTransformnType`, sustituyendo *n* por un ordinal consecutivo, comenzando en 1 (ver notas adicionales sobre indicación de transformaciones adicionales).
`xmlTransformnType`	`http://www.w3.org/2000/09/xmldsig#base64`	El contenido se transforma en Base64 antes de ser firmado.
	`http://www.w3.org/TR/1999/REC-xpath-19991116`	El contenido se transforma mediante XPATH antes de ser firmado. Únicamente es aplicable cuando se firma contenido XML.
	`http://www.w3.org/TR/1999/REC-xslt-19991116`	El contenido se transforma mediante XSLT antes de ser firmado. Únicamente es aplicable cuando se firma contenido XML.
	`http://www.w3.org/2002/06/xmldsig-filter2`	El contenido se transforma mediante XPATH2 antes de ser firmado. Únicamente es aplicable cuando se firma contenido XML.
`xmlTransformnSubtype`	[Texto]	Subtipo de la transformación *n*. Los valores aceptados y sus funcionalidades dependen del valor indicado en `xmlTransformnType`.
`xmlTransformnBody`	[Texto]	Cuerpo de la transformación *n*. Los valores aceptados y sus funcionalidades dependen del valores indicados en `xmlTransformnType` y en `xmlTransformnSubtype`.

9.3.1.1 Indicación de transformaciones adicionales al contenido a firmar

Respecto al uso de los parámetros `xmlTransformnType`, `xmlTransformnSubtype` y `xmlTransformnBody`, sus valores van ligados, aceptándose las siguientes combinaciones:

- Transformación XPATH
  - Tipo: `http://www.w3.org/TR/1999/REC-xpath-19991116`
  - Subtipos: No tiene subtipos.
  - Cuerpo: Especificado mediante sentencias de tipo XPATH.
- Transformación XPATH2
  - Tipo: `http://www.w3.org/2002/06/xmldsig-filter2`
  - Subtipos:
    - `subtract`: Resta.
    - `intersect`: Intersección
    - `union`: Unión
  - Cuerpo: Especificado mediante sentencias de tipo XPATH2.
- Transformación XSLT
  - Tipo: `http://www.w3.org/TR/1999/REC-xslt-19991116`
  - Subtipos: No tiene subtipos.
  - Cuerpo: Especificado mediante sentencias de tipo XSLT.
- Transformación BASE64
  - Tipo: `http://www.w3.org/2000/09/xmldsig#base64`
  - Subtipos: No tiene subtipos.
  - Cuerpo: No tiene cuerpo.

No es posible especificar transformaciones complejas que incluyan varias sentencias. En su lugar, puede declararse una sucesión de transformaciones simples que produzcan el mismo resultado. Cada una de las transformaciones se aplicará de forma ordenada sobre el resultado de la anterior.

9.3.2 Contrafirma

Nombre del parámetro	Valores posibles	Descripción
`policyIdentifier`	[URL]	Identificador de la política de firma (normalmente una URL hacia la política en formato XML procesable), necesario para generar firmas XAdES-EPES.
`policyIdentifierHash`	[Valor en Base64]	Huella digital de la política de firma. Es obligatorio indicar este parámetro si el valor indicado en `policyIdentifier` no es universalmente accesible. Si se da valor a este parámetro es obligatorio también dar valor al parámetro `policyIdentifierHashAlgorithm`.
`policyIdentifierHashAlgorithm`	`SHA1`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA1.
	`SHA-256`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-256.
	`SHA-384`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-384.
	`SHA-512`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA-512.
`policyQualifier`	[URL hacia documento]	URL (universalmente accesible) hacia el documento (normalmente PDF) que contiene una descripción textual de la política de firma. Este parámetro es opcional incluso si se desea generar firmas XAdES-EPES.
`policyDescription`	[Texto]	Descripción textual de la política de firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado. Este parámetro es opcional incluso si se desea generar firmas XAdES-EPES.
`signerClaimedRole`	[Texto]	Agrega a la firma un campo con el cargo atribuido al firmante. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
`signerCertifiedRole`	[Texto]	Agrega a la firma un campo con el cargo

		confirmado del firmante. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
`signatureProductionCity`	[Texto]	Agrega a la firma un campo con la ciudad en la que se realiza la firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
`signatureProductionProvince`	[Texto]	Agrega a la firma un campo con la provincia en la que se realiza la firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
`signatureProductionPostalCode`	[Texto]	Agrega a la firma un campo con el código postal en donde se realiza la firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado.
`signatureProductionCountry`	[Texto]	Agrega a la firma un campo con el país en el que se realiza la firma. En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML contrafirmado.
`encoding`	[Texto]	Fuerza una codificación para la firma resultante. Un uso incorrecto de este parámetro puede provocar la generación de una firma inválida.
`applySystemDate`	`true`	La firma incluirá un atributo con la fecha y hora de la firma, obtenidas del reloj del ordenador del usuario (valor por defecto).
	`false`	Se omite la inclusión de la fecha y hora de la firma como atributo de esta última.

10 Información específica para firma de facturas electrónicas

10.1 Operaciones no soportadas y notas de interés

- Las facturas electrónicas se firman con el formato XAdES Enveloped pero con unas particularidades concretas que no es posible replicar mediante el formato XAdES del Cliente @firma.
    - Es necesario utilizar el formato FacturaE para la firma de facturas.
- El formato FacturaE sólo puede utilizarse sobre facturas electrónicas acordes al estándar.
- Las facturas electrónicas no soportan las operaciones de cofirma ni contrafirma.
    - Si se intenta hacer una operación de cofirma o contrafirma sobre una factura electrónicas se notificará que no es posible porque ésta ya cuenta con una firma.

10.2 Algoritmos de firma

Las firmas de FacturaE aceptan los siguientes algoritmos de firma (deben escribirse exactamente como aquí se muestran):

- `SHA1withRSA`
- `SHA256withRSA`
- `SHA384withRSA`
- `SHA512withRSA`

El algoritmo más seguro, y por lo tanto el recomendado para su uso es `SHA512withRSA`. Sin embargo, tenga en cuenta que si el usuario utiliza una versión de Java anterior a la recomendada en el apartado de requisitos mínimos del Entorno Cliente, es posible que no pueda generar firmas electrónicas con los algoritmos SHA-2 (SHA256, SHA384 y SHA512) desde algunos almacenes de certificados.

10.3 Parámetros adicionales

A continuación se detallan los parámetros adicionales que acepta el formato FacturaE para la configuración de la operación y la firma electrónica generada.

Es posible que el uso de parámetros no contemplados en las siguientes tablas provoque otros cambios de funcionamiento. No obstante **no se dará soporte** al aplicativo si se usan parámetros no documentados, asumiendo el integrador todo el riesgo y responsabilidad derivados del uso de parámetros o valores distintos de los aquí descritos.

Nombre del parámetro	Valores posibles	Descripción
`signerCertifiedRole`	[Texto]	Agrega a la firma un campo con el cargo confirmado del firmante.
`signatureProductionCity`	[Texto]	Agrega a la firma un campo con la ciudad en la que se realiza la firma.
`signatureProductionProvince`	[Texto]	Agrega a la firma un campo con la provincia en la que se realiza la firma.

`signatureProductionPostalCode`	[Texto]	Agrega a la firma un campo con el código postal en donde se realiza la firma.
`signatureProductionCountry`	[Texto]	Agrega a la firma un campo con el país en el que se realiza la firma.

11 Información específica para firmas PAdES

11.1 Operaciones no soportadas y notas de interés

- Las firmas PAdES no admiten contrafirmas.
- Una cofirma PAdES consiste en la adición de una firma adicional al documento PDF, sin que se establezca ninguna relación de interdependencia con las firmas existentes.
    - Cofirmar un documento PDF es completamente equivalente a firmar un documento PDF ya firmado.
- Únicamente es posible usar PAdES para documentos PDF.
- No se firman los posibles adjuntos o empotrados que pudiese contener el documento PDF.
- No se soporta la firma de PDF cifrados con certificados o con algoritmo AES256.

11.2 Algoritmos de firma

Las firmas PAdES aceptan los siguientes algoritmos de firma (deben escribirse exactamente como aquí se muestran):

- `SHA1withRSA`
- `SHA256withRSA`
- `SHA384withRSA`
- `SHA512withRSA`

El estándar PAdES recomienda no usar el algoritmo `SHA1withRSA` por no ser el más seguro.

El algoritmo más seguro, y por lo tanto el recomendado para su uso es `SHA512withRSA`. Sin embargo, tenga en cuenta que si el usuario utiliza una versión de Java anterior a la recomendada en el apartado de requisitos mínimos del Entorno Cliente, es posible que no pueda generar firmas electrónicas con los algoritmos SHA-2 (SHA256, SHA384 y SHA512) desde algunos almacenes de certificados.

11.3 Parámetros adicionales

A continuación se detallan los parámetros adicionales que aceptan cada una de las formas de generación de firmas.

Es posible que el uso de parámetros no contemplados en las siguientes tablas provoque otros cambios de funcionamiento. No obstante **no se dará soporte** al aplicativo si se usan parámetros no documentados, asumiendo el integrador todo el riesgo y responsabilidad derivados del uso de parámetros o valores distintos de los aquí descritos.

Nombre del parámetro	Valores posibles	Descripción
`signReason`		Incluye un campo en el diccionario PDF con el motivo por el que se ha realizado la firma.
`signField`		Nombre del campo del PDF donde debe insertarse la firma generada. Si el PDF contiene campos de firma con nombre, pre-creados y vacíos, puede usarse este parámetro para indicarlo.
`signatureProductionCity`		Incluye un campo en el diccionario PDF con la ciudad donde se ha realizado la firma.
`signerContact`		Incluye un campo en el diccionario PDF con información de contacto del firmante (usualmente una dirección de correo electrónico).
`ownerPassword`		Contraseña del PDF a firmar en el caso de que este estuviera cifrado.
`headLess`	`true`	Evita mostrar diálogos gráficos adicionales al usuario (como por ejemplo, la solicitud de contraseñas de los PDF).
	`false`	Permite mostrar diálogos gráficos adicionales al usuario (valor por defecto).
`avoidEncryptingSignedPdfs`	`true`	No cifra el PDF firmado aunque el PDF original si lo estuviese.
	`false`	Cifra el PDF firmado cuando PDF original lo estuviese (valor por defecto).
`allowSigningCertifiedPdfs`	`true`	Permite la firma de PDF certificados (la firma resultante puede ser inválida)
	`false`	No permite la firma de PDF certificados (valor por defecto).

12 Problemas conocidos

12.1 Error al firmar ficheros mayores de 4 megabytes

El uso de versiones antiguas de Java y otros factores, como la limitación de memoria del sistema del usuario, pueden provocar que no sea posible cargar y firmar ficheros mayores de 4 Mb.

Para sortear este problema es necesario que el usuario se actualice a la última versión disponible de Java.

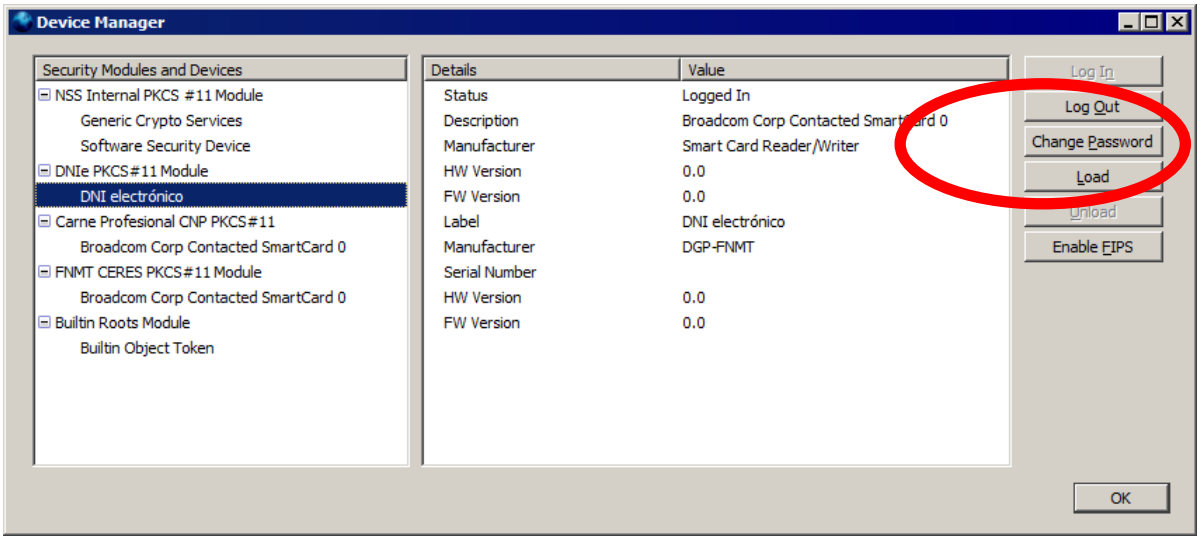
Por parte del integrador, es posible evitar parcialmente este problema abandonando, siempre que sea posible, el uso de los métodos de carga de fichero `getFileNameContentBase64` y `getMultiFileNameContentBase64`, en favor de la carga de datos realizada al llamar a las operaciones criptográficas sin indicar los datos a firmar/multifirma. Por ejemplo, en lugar de llamar al método `getFileNameContentBase64` para cargar un fichero y luego pasarle los datos obtenidos al método `sign`, llamaremos al método `sign` sin datos para que sea el mismo el que permita cargar el fichero de datos.

12.2 Con el navegador Mozilla Firefox y DNIe (DNI Electrónico) el *applet* se queda bloqueado y no muestra el diálogo de selección de certificados, desbloqueándose si retiro el DNIe del lector

El controlador PKCS#11 oficial del DNIe no admite que se establezcan varias sesiones de forma simultánea, y si por cualquier razón (sesión SSL, etc.) el propio navegador Web Mozilla / Firefox tiene ya establecida una comunicación con el DNIe en el momento en el que el Cliente @firma también lo necesita, este último se queda bloqueado esperando a que en navegador Mozilla / Firefox cierre su sesión. El cierre de la sesión contra el DNIe por parte de Mozilla / Firefox puede tardar varios minutos si el usuario no interviene, por lo que conviene forzar manualmente este cierre:

- Extraer el DNIe del lector y volverlo a insertar justo en el momento en el que se solicita la contraseña del Repositorio Central de certificados de Mozilla Firefox (antes de introducirla). Es posible que Mozilla / Firefox reabra la sesión en la reinserción (adelantándose al Cliente @firma), por lo que quizás necesite repetir la operación.

- Podemos indicar a Mozilla / Firefox que cierre la sesión pulsando el botón “Log out” teniendo el dispositivo “DNIe PKCS#11 Module” seleccionado en la ventana “Dispositivos de Seguridad” del menú Opciones de Mozilla Firefox. Al igual que en el método anterior, a veces es necesario repetir la operación varias veces, ya que Mozilla / Firefox reabre automáticamente la comunicación con el DNIe sin dar tiempo al Cliente @firma a utilizarlo. En otras ocasiones, el botón aparece deshabilitado aunque Mozilla / Firefox tenga una sesión abierta contra el dispositivo, con lo que no es posible aplicar este método.



Este problema surge principalmente en sistemas Linux/Solaris. Para estos sistemas se recomienda el uso del controlador OpenDNIe para DNI electrónico. Puede encontrar este controlador en:

https://forja.cenatic.es/projects/opendnie/

12.3 No se detecta la inserción/extracción del DNIe en el lector (u otra tarjeta inteligente)

A veces puede ocurrir que el navegador no detecta la extracción o introducción del DNIe (u otra tarjeta inteligente) en el lector, por lo que si no hemos introducido la tarjeta previamente a que se arranque el cliente de firma, no se encontrará el certificado. Otro posible caso es que una vez cargado el cliente, se extraiga la tarjeta y, al realizar una operación de firma, el navegador muestre los certificados de la tarjeta (aunque ya no esté presente) fallando al intentar utilizarlo.

Este es un problema del navegador en la gestión de los dispositivos criptográficos (PKCS#11 para Mozilla y CSP para Internet Explorer), que no informa a la sesión abierta en el almacén de certificados de los cambios que se producen en el mismo.

La solución más rápida al problema es el insertar la tarjeta antes de que se produzca la carga del cliente de firma.

12.4 El *applet* no detecta ningún certificado bajo Mozilla / Firefox

El Cliente @firma, cuando se ejecuta en Linux o Sun Solaris necesita que las bibliotecas NSS estén situadas en "/usr/lib", "/lib" o al menos dentro de uno de los directorios incluidos en la variable de entorno LD_LIBRARY_PATH.

12.5 El MiniApplet no permite la firma de PDF con ciertos certificados

Las firmas de documentos PDF realizadas externamente (que es el método utilizado por el Cliente y el MiniApplet @firma) tienen un tamaño máximo de octetos que pueden ocupar dentro del PDF.

Como la firma incluye la cadena de certificación completa, si esta es muy extensa puede llegar a agotarse este espacio y resultar en una firma inválida o corrupta.