



DIRECCIÓN DE TECNOLOGÍAS DE LA INFORMACIÓN Y
LAS COMUNICACIONES

MiniApplet @firma

Manual del integrador del MiniApplet v1.3 del Cliente @firma

Índice de contenidos

| | | |
|-------|--|----|
| 1 | Introducción | 7 |
| 1.1 | Productos de terceros incluidos con el MiniApplet @firma | 7 |
| 2 | Requisitos mínimos | 9 |
| 2.1 | Entorno Cliente | 9 |
| 2.1.1 | Habilitación de Applets de Java en navegador Web (cualquier entorno operativo) | 10 |
| 2.1.2 | Compatibilidad con Windows 8 y superiores..... | 10 |
| 2.1.3 | Compatibilidad con Apple OS X..... | 11 |
| 2.2 | Dispositivos móviles | 12 |
| 2.3 | Entorno Servidor | 12 |
| 3 | Funcionalidad proporcionada por el MiniApplet @firma | 13 |
| 3.1 | Formatos de firma soportados por el MiniApplet @firma | 13 |
| 3.1.1 | CAdES (CMS Advanced Electronic Signature)..... | 13 |
| 3.1.2 | XAdES (XML Advanced Electronic Signature)..... | 13 |
| 3.1.3 | FacturaE (Factura electrónica) | 14 |
| 3.1.4 | PAdES (PDF Advanced Electronic Signature)..... | 14 |
| 3.1.5 | ODF (Open Document Format – Firmas de documentos LibreOffice / OpenOffice.org).... | 14 |
| 3.1.6 | OOXML (Office Open XML – Firmas de documentos Microsoft Office)..... | 14 |
| 3.2 | Formatos de firma no soportados por el MiniApplet @firma | 15 |
| 3.3 | Uso de certificados y claves privadas por parte del MiniApplet @firma..... | 15 |
| 3.3.1 | Establecimiento manual del almacén de certificados y claves | 16 |
| 3.3.2 | Uso exclusivo de certificados accesibles mediante PKCS#11 en Mozilla Firefox..... | 19 |
| 3.3.3 | Forzar ruta del almacén de Mozilla Firefox..... | 19 |
| 4 | Despliegue del MiniApplet @firma | 21 |
| 4.1 | Importación de las bibliotecas JavaScript | 21 |
| 4.1.1 | Importación en páginas Web generadas dinámicamente | 22 |
| 4.2 | Carga del MiniApplet..... | 23 |
| 4.3 | Configuración del idioma | 24 |
| 4.4 | Restricción según desfase horario con el servidor..... | 25 |
| 4.5 | Firma del JAR del MiniApplet | 26 |

| | | |
|-------|---|----|
| 4.6 | Despliegue en entornos de Web dinámica, servidores de aplicaciones y en general servidores no estáticos | 27 |
| 5 | El proceso de firma electrónica..... | 28 |
| 5.1 | Selección del contenido a firmar..... | 28 |
| 5.1.1 | La conversión de datos a Base64 | 28 |
| 5.1.2 | Selección de contenido desde ficheros en disco..... | 29 |
| 5.1.3 | Selección de objetivo de las contrafirmas..... | 29 |
| 5.2 | Selección del certificado y clave privada de firma | 29 |
| 5.3 | Firma | 30 |
| 5.4 | Recogida de resultados | 30 |
| 6 | Funciones disponibles del MiniApplet @firma | 31 |
| 6.1 | Uso del API desde JavaScript..... | 31 |
| 6.2 | Obtención de resultados..... | 31 |
| 6.2.1 | Obtención directa de resultados..... | 32 |
| 6.2.2 | Obtención de resultados mediante <i>Callbacks</i> | 32 |
| 6.3 | Firma electrónica..... | 34 |
| 6.4 | Firmas electrónicas múltiples..... | 37 |
| 6.4.1 | Cofirmas | 37 |
| 6.4.2 | Contrafirmas..... | 39 |
| 6.5 | Firmas/Multifirmas trifásicas | 44 |
| 6.5.1 | Realizar firma trifásicas con el MiniApplet Cliente @firma | 45 |
| 6.5.2 | Servicio de firma trifásica..... | 46 |
| 6.6 | Firmas/Multifirmas masivas..... | 49 |
| 6.7 | Gestión de ficheros | 49 |
| 6.7.1 | Guardado de datos en disco | 50 |
| 6.7.2 | Selección de un fichero por parte del usuario y recuperación de su nombre y contenido | 51 |
| 6.7.3 | Selección de múltiples ficheros por parte del usuario y recuperación de sus nombres y contenidos..... | 53 |
| 6.8 | Utilidad | 55 |
| 6.8.1 | Eco | 55 |
| 6.8.2 | Obtención de los mensajes de error | 56 |

| | | |
|-------|--|----|
| 6.8.3 | Conversión de una cadena Base64 a texto | 56 |
| 6.8.4 | Conversión de un texto a cadena Base64 | 57 |
| 7 | Configuración de las operaciones | 58 |
| 7.1 | Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma | 58 |
| 7.1.1 | Parámetros adicionales no soportados..... | 58 |
| 7.2 | Selección automática de certificados..... | 58 |
| 7.3 | Configuración del filtro de certificados..... | 59 |
| 7.4 | Configuración de la política de firma | 64 |
| 7.4.1 | Configuración manual | 64 |
| 7.4.2 | Política de firma de la AGE v1.9 | 64 |
| 7.4.3 | Política de firma de la AGE v1.8 | 66 |
| 7.4.4 | Política de firma de Factura electrónica (Facturae) | 66 |
| 7.5 | Configuración de parámetros de la JVM..... | 67 |
| 7.6 | Configuración a través de propiedades del sistema | 67 |
| 8 | Gestión de errores..... | 67 |
| 9 | Compatibilidad con dispositivos móviles y AutoFirma | 70 |
| 9.1 | Arquitectura del sistema | 70 |
| 9.2 | Despliegues compatibles..... | 71 |
| 9.3 | Servicios auxiliares del Cliente @firma móvil | 72 |
| 9.3.1 | Notas sobre los servicios de almacenaje y recuperación..... | 73 |
| 9.4 | Notificaciones al usuario | 74 |
| 9.5 | Enlaces de descarga | 76 |
| 9.5.1 | Enlaces Android..... | 76 |
| 9.5.2 | Enlaces iOS | 77 |
| 9.5.3 | Enlaces AutoFirma..... | 77 |
| 9.6 | Limitaciones | 78 |
| 9.6.1 | Limitaciones de formato | 78 |
| 9.6.2 | Limitaciones funcionales | 78 |
| 9.7 | Compatibilidad del MiniApplet @firma v1.3 con aplicaciones móviles y AutoFirma | 80 |
| 10 | Información específica para firmas CAdES..... | 81 |

| | | |
|--------|--|-----|
| 10.1 | Algoritmos de firma..... | 81 |
| 10.2 | Firmas CAdES implícitas o explícitas | 81 |
| 10.3 | Parámetros adicionales..... | 81 |
| 10.3.1 | Firma y cofirma | 81 |
| 10.3.2 | Contrafirma | 83 |
| 11 | Información específica para firmas XAdES..... | 86 |
| 11.1 | Tratamiento de las hojas de estilo XSL de los XML a firmar | 86 |
| 11.2 | Algoritmos de firma..... | 87 |
| 11.3 | Algoritmos de huella digital para las referencias..... | 87 |
| 11.4 | Situación del nodo de firma en XAdES Enveloped | 87 |
| 11.5 | Uso de estructuras <i>Manifest</i> en firmas XAdES..... | 89 |
| 11.6 | Parámetros adicionales..... | 90 |
| 11.6.1 | Firma y cofirma | 90 |
| 11.6.2 | Contrafirma | 96 |
| 11.7 | Firmas XAdES “explícitas” | 97 |
| 12 | Información específica para firma de facturas electrónicas | 99 |
| 12.1 | Operaciones no soportadas y notas de interés..... | 99 |
| 12.2 | Algoritmos de firma..... | 99 |
| 12.3 | Parámetros adicionales..... | 99 |
| 13 | Información específica para firmas PAdES..... | 101 |
| 13.1 | Creación de una firma visible..... | 101 |
| 13.2 | Insertión de una imagen en un documento PDF antes de ser firmado..... | 105 |
| 13.3 | Operaciones no soportadas y notas de interés..... | 106 |
| 13.4 | Firma de documentos PDF cifrados o protegidos con contraseña | 106 |
| 13.5 | Algoritmos de firma..... | 107 |
| 13.6 | Parámetros adicionales..... | 107 |
| 14 | Problemas conocidos | 114 |
| 14.1 | Uso de ciertos algoritmos con Windows XP..... | 114 |
| 14.2 | Google Chrome no ejecuta ni el MiniApplet Cliente @firma ni ningún otro Applet de Java a pesar de tener correctamente instalado el JRE | 114 |
| 14.2.1 | Microsoft Windows..... | 114 |

| | | |
|--------|--|-----|
| 14.2.2 | Apple OS X y Linux..... | 115 |
| 14.3 | Diálogos de advertencia al usuario en Java 7u55 y posteriores (incluyendo Java 8)..... | 115 |
| 14.4 | Error “el conjunto de claves no existe” al firmar con el almacén de claves de Windows | 116 |
| 14.5 | Con el navegador Mozilla Firefox y DNle (DNI Electrónico) el <i>Applet</i> se queda bloqueado y no muestra el diálogo de selección de certificados, desbloqueándose si retiro el DNle del lector | 117 |
| 14.6 | No se detecta la inserción/extracción del DNle en el lector (u otra tarjeta inteligente)..... | 118 |
| 14.7 | El <i>Applet</i> no detecta ningún certificado bajo Mozilla / Firefox..... | 118 |
| 14.8 | El MiniApplet no permite la firma de PDF con ciertos certificados | 119 |
| 14.9 | El servicio de firma trifásica genera un error al realizar firmas XAdES en servidores JBoss u otros | 119 |
| 14.9.1 | Despliegue del servicio de firma trifásica sobre JBoss..... | 120 |

1 Introducción

El MiniApplet Cliente @firma (también referido en este manual como “MiniApplet” o simplemente “Cliente”) es una herramienta de firma electrónica que funciona en forma de *Applet* de Java integrado en una página Web mediante JavaScript.

El MiniApplet Cliente @firma 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 DNle).

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 MiniApplet Cliente @firma le envía los datos a firmar y éste los devuelve firmados.

El MiniApplet Cliente @firma no almacena ningún tipo de información personal del usuario, ni hace uso de cookies ni ningún otro mecanismo para la gestión de datos de sesión. El MiniApplet sí almacena el log de su última ejecución a efectos de ofrecer soporte al usuario si se encontrase algún error durante la ejecución del *Applet*. Sólo se almacena el log de la última ejecución del MiniApplet, este no contiene ningún tipo de información personal y el integrador no tiene acceso al mismo. Es el usuario quien debe remitirlo.

El MiniApplet Cliente @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 (EURL 1.1) o superior.

Puede consultar la información relativa al proyecto Cliente @firma, dentro del cual se encuentra el MiniApplet Cliente @firma y descargar el código fuente 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/clientefirma>

1.1 Productos de terceros incluidos con el MiniApplet @firma

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

- JXAdES (<https://github.com/universitatjaumei/jxades>)
- 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 JUniversalCharDet (<https://code.google.com/p/juniversalchardet/>)
- 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 bits (x86).
 - Se recomienda adoptar Java 8 o si no fuese posible usar al menos Java 6u45 o 7u76.
 - En Apple OS X no se soporta (por obsolescencia) la versión 6 del JRE de Apple, siendo necesario usar las versiones 7 u 8 del JRE de Oracle.
- Java SE 7 Update 10 (1.7.0_10) o superior.
 - Se recomienda adoptar Java 8 o si no fuese posible usar al menos Java 7u76.
 - En 32 (x86) o 64 (x64/AMD64) bits según la arquitectura del navegador Web.
 - En Internet Explorer se recomienda siempre usar versiones de 32 bits.
- Java SE 8
 - Se recomienda usar al menos la versión 8u51.
 - En 32 (x86) o 64 (x64/AMD64) bits según la arquitectura del navegador Web.
 - En Internet Explorer se recomienda siempre usar versiones de 32 bits.

Sistema operativo:

- Windows XP SP3, Vista SP2, 7 SP1, 8 o superior, en 32 (x86) o 64 (x64) bits.
 - Se recomienda abandonar Windows XP en favor de Windows 7 o superior.
- 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.
 - Se recomienda al menos un Linux basado en la versión 3 o superior del núcleo (Linux Kernel).
- Apple OS X Mavericks (10.9.5 o superior) o Yosemite (10.10).

Navegador Web:

- Mozilla Firefox 4.0 o superior
 - En Windows únicamente se soporta Firefox en 32 bits.
- Google Chrome versiones de la 15 a la 41. A partir de, Chrome 41, es necesario tener previamente instalado el programa AutoFirma en el sistema del usuario.
- Apple Safari 6.2 o superior (soporte prestado únicamente para la versión OS X).
- Microsoft Internet Explorer 8 o superior (se recomienda usar siempre versiones de 32 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 los entornos de ejecución redistribuibles de Microsoft Visual C++

2005 y 2013. Si no consigue acceder a sus certificados y claves privadas desde el MiniApplet @firma, necesitará descargarlos e instalarlos manualmente. Estos pueden descargarse de:

- Visual Studio 2013 (una vez en el enlace, seleccione el idioma y la arquitectura adecuada para su Sistema operativo).
 - <https://www.microsoft.com/en-us/download/details.aspx?id=40784>
- Visual Studio 2005
 - 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 Habilitación de Applets de Java en navegador Web (cualquier entorno operativo)

En cualquier entorno operativo actual, y debido a los riesgos de seguridad que ello implica, es necesario habilitar los Applets de Java en los navegadores Web.

Para ello, es necesario seguir un proceso que es diferente según el navegador Web, y que puede variar según el sistema operativo.

Consulte la documentación del navegador Web para obtener instrucciones sobre cómo habilitar los Applets de Java, y siga atentamente las indicaciones de Oracle para este proceso, que puede encontrar en:

http://java.com/en/download/help/enable_browser.xml

2.1.2 Compatibilidad con Windows 8 y superiores

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

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"/>
```

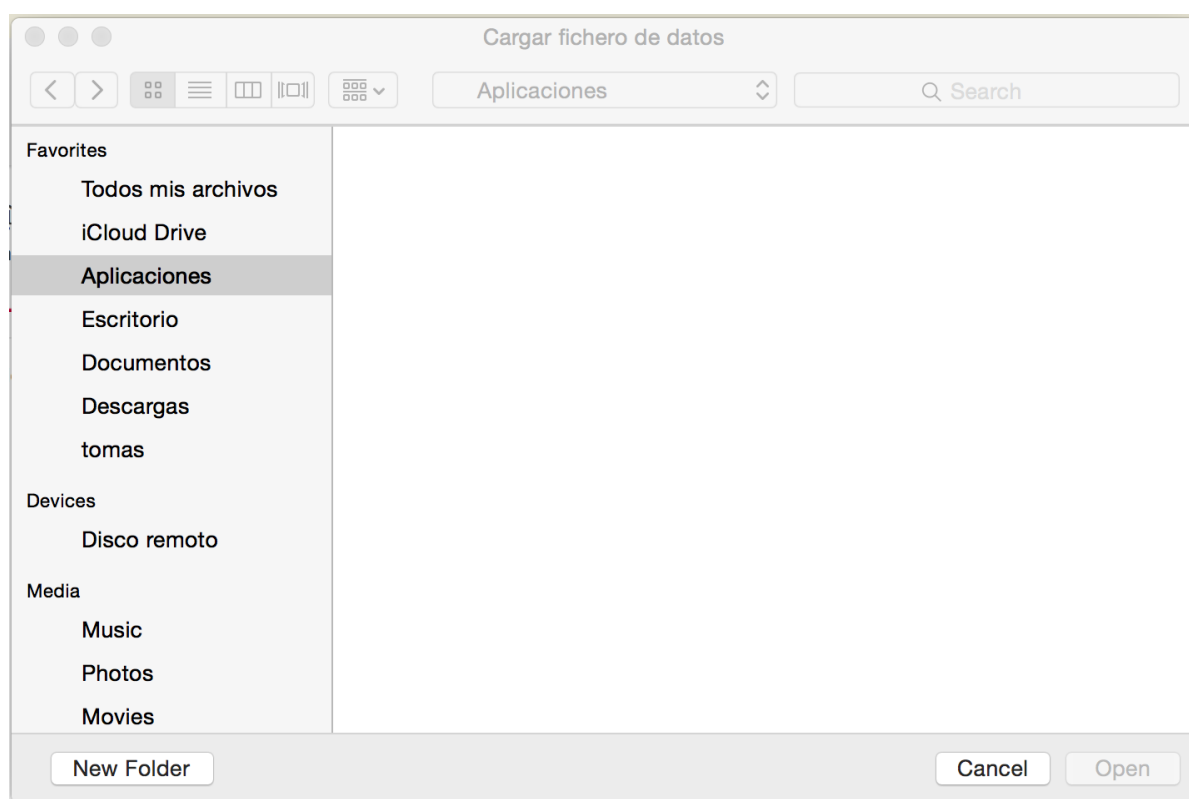
Puede encontrar más información sobre complementos de navegador (*plugins*) en Internet Explorer sobre Metro en Windows 8 en:

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

2.1.3 Compatibilidad con Apple OS X

Los Applets de Java en OS X con Safari tienen restringido por defecto el acceso al sistema de ficheros, lo cual puede causar cierta confusión con el MiniApplet Cliente @firma en OS X, porque da la impresión de funcionar apropiadamente, pero cuando se solicita abrir o guardar un fichero, no es posible.

En particular, lo que el usuario aprecia al intentar abrir un fichero es que el diálogo de selección se abre, pero no muestra ningún archivo:



Para conseguir que el MiniApplet tenga acceso no restringido al sistema de ficheros del usuario es necesario que este configure Safari para habilitar el “Modo Inseguro” para el sitio Web en concreto que publique el MiniApplet.

Para ello, siga las indicaciones de Apple, que puede encontrar aquí:

<http://support.apple.com/kb/HT5954>

2.2 Dispositivos móviles

Los *Applets* Java, como el MiniApplet @firma, no puede ejecutarse desde dispositivos Google Android o Apple iOS; sin embargo, el correcto despliegue del MiniApplet hará que sus aplicaciones puedan ejecutar las operaciones de firma utilizando las aplicaciones nativas del proyecto Cliente @firma para estos sistemas en lugar del MiniApplet.

Consulte el apartado [Compatibilidad con dispositivos móviles](#) para saber más.

2.3 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 (recomendada última versión disponible de Java 7) con JEE 5 como mínimo.

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 firmas múltiples) 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 solo puede utilizarse sobre facturas electrónicas y sobre ellas sólo es posible realizar la operación de firma. No permite cofirmarlas ni contrafirmarlas.

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-Básico (por defecto)
- PAdES-BES

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

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-Básico a menos que se indique una política de firma AGE 1.9 o superior (en cuyo caso, por adecuación, se para a generar PAdES-BES) o se indique explícitamente mediante un parámetro adicional que se desea usar PAdES-BES.

3.1.5 ODF (Open Document Format – Firmas de documentos LibreOffice / OpenOffice.org)

El MiniApplet Cliente @firma es capaz de realizar firmas ODF, de forma acorde a la siguiente tabla de compatibilidad:

| OpenOffice.org 3.2 / 3.3 | | |
|-----------------------------|----------------------------|-----------------|
| Impress (Presentaciones) | Calc (Hojas de Cálculo) | Writer (Textos) |
| | | |

Es importante reseñar que las firmas ODF no son acordes al Esquema Nacional de Interoperabilidad (ENI).

3.1.6 OOXML (Office Open XML – Firmas de documentos Microsoft Office)

Es posible firmar documentos OOXML en formatos reconocidos por Microsoft Office. El soporte de las firmas realizadas con el MiniApplet Cliente @firma en cuanto a versiones y programas de Microsoft Office es la siguiente:

| | Office 2007 | Office 2008 | Office 2010 | Office 2011 | Office 2012 |
|-------------------|-------------|-------------|-------------|-------------|-------------|
| Word (docx) | NO | N/A | SÍ | N/A | SÍ |
| Excel (xlsx) | NO | N/A | SÍ | N/A | SÍ |
| PowerPoint (pptx) | NO | N/A | SÍ | N/A | SÍ |
| Project | NO | NO | NO | NO | NO |
| OneNote | NO | NO | NO | NO | NO |
| Visio | NO | NA | NO | NO | NO |
| XPS | NO | NO | NO | NO | NO |

SÍ = Compatible, NO = No compatible, N/A = No aplica (estas versiones de Office para OS X no permiten mostrar o comprobar las firmas OOXML).

Es importante reseñar que las firmas ODF no son acordes al Esquema Nacional de Interoperabilidad (ENI).

3.1.6.1 Versiones de Java compatibles con las firmas OOXML

Por errores conocidos de la máquina virtual de Java en su versión 6, únicamente es posible realizar este tipo de firmas con Java 7 y superiores, recomendándose el uso de Java 8u45 como mínimo.

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 ningún otro formato no nombrado explícitamente en este manual.

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/clientefirma>

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, DNle (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*).

Con independencia del almacén que use el MiniApplet Cliente @firma, el navegador puede combinar el uso de unos u otros almacenes para tareas relacionadas con las conexiones SSL, aspecto que no influye en estas reglas de selección. Así, por ejemplo Mozilla Firefox hará uso del almacén NSS para las conexiones de Red, pero utilizará el de Java a la hora de cargar Applets desde sitios Web con SSL.

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", dentro del objeto `MiniApplet`, 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_JAVA
 - Almacén en fichero JKS (*Java KeyStore*).
- KEYSTORE_JCEKS
 - Almacén en fichero JCEKS (*Java Cryptography Extension KeyStore*).
- KEYSTORE_JAVACE
 - Almacén en fichero de tipo CaseExactJKS (*Case Exact Java KeyStore*).
- 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:

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

ADVERTENCIA: La aplicación AutoFirma ignora la configuración del almacén de certificados configurado y el navegador web utilizado. Siempre utiliza el almacén de certificados del sistema. Para saber más de AutoFirma y su despliegue, consulte el apartado “9 Compatibilidad con dispositivos móviles y AutoFirma”.

3.3.2 Uso exclusivo de certificados accesibles mediante PKCS#11 en Mozilla Firefox

Estableciendo a `true` la propiedad del sistema

`es.gob.afirma.keystores.mozilla.LoadSscdOnly` (debe hacerse antes de la carga del MiniApplet) se activa un modo especial de funcionamiento que provoca que se ignoren los certificados y claves del almacén central de certificados de Mozilla Firefox (NSS) y se usen exclusivamente los accesibles mediante módulos de seguridad PKCS#11 adicionales.

Esta funcionalidad puede resultar útil para forzar el uso de tarjetas inteligentes o dispositivos de almacén USB y descartar los certificados alojados en el almacén software del navegador.

Consulte el apartado Configuración a través de propiedades del sistema para saber cómo establecer propiedades de sistema.

3.3.3 Forzar ruta del almacén de Mozilla Firefox

El MiniApplet detecta de forma automática dónde está instalado el navegador Mozilla Firefox y, de esta forma, cómo acceder al su almacén de certificados y cuál es el almacén correspondiente al usuario actual, si hubiese varios usuarios definidos dentro de la misma cuenta del sistema operativo. Si no se encontrase ningún activo el perfil de ningún usuario de entre los definidos, se cargaría el perfil por defecto.

Existen ocasiones muy concretas en las que no es posible detectar dónde está instalado el navegador como, por ejemplo, si se utiliza un navegador Firefox modificado (como una edición Portable) o si se intenta acceder al almacén de una cuenta concreta de Firefox sin usar el propio navegador. En estos casos, el integrador podrá usar las siguientes propiedades del sistema para localizar los recursos necesarios para indicar al MiniApplet donde encontrarlo:

- `es.gob.afirma.keystores.mozilla.UseEnvironmentVariables`
 - Establezca esta variable a `“true”` para indicar que deben leerse alguna de las dos variables que se describen a continuación.
- `AFIRMA_NSS_HOME`
 - Directorio con las bibliotecas NSS compatibles con la versión de a la que pertenezca el almacén al que deseamos acceder.
- `AFIRMA_PROFILES_INI`
 - Ruta completa (incluyendo el nombre de archivo) hacia el fichero `profiles.ini` que contiene la información de perfiles de usuario de Firefox.
 - Por ejemplo:
 - `AFIRMA_PROFILES_INI=c:\Tomas\profiles.ini`

- Si el fichero referenciado en esta variable de entorno no existe o no se tienen permisos de lectura sobre él, se ignora, pasándose entonces a usar el fichero de perfiles por defecto de Firefox.

Si un integrador deseara desde un sistema integrado acceder a un almacén de una cuenta concreta de Firefox, deberá configurar estas variables de entorno.

Consulte el apartado Configuración a través de propiedades del sistema para saber cómo establecer propiedades de sistema.

3.3.3.1 Aclaraciones sobre la configuración de perfiles de Mozilla Firefox

La funcionalidad descrita anteriormente debe aplicarse únicamente cuando se está seguro de la localización de los directorios de Mozilla Firefox y esta no es la normal para ellos, lo cual se suele dar únicamente en despliegues internos muy controlados de versiones alteradas de Mozilla Firefox, como las versiones que se denominan “Portables”.

El fichero de configuración `profiles.ini` de Firefox contiene información sobre los perfiles de usuario instalados en un entorno de ejecución concreto de Firefox. Un ejemplo de este fichero podría ser el siguiente:

```
[General]

StartWithLastProfile=1


[Profile0]

Name=default-1415619763084

IsRelative=1

Path=Profiles/ior5adlg.default-1435315801996

Default=1


[Profile1]

Name=default-1427127404614

IsRelative=1

Path=Profiles/ior5adlg.default-1435315801996


[Profile2]

Name=default-1435315801996

IsRelative=1
```

Path=Profiles/ior5ad1g.default-1435315801996

En este fichero se puede observar que hay configurados tres perfiles de usuario, siendo el perfil por defecto el primero (el número cero).

No obstante, el que esté declarado como por defecto únicamente quiere decir que este será el usado en Firefox si el usuario no indica lo contrario. Dado que el MiniApplet Cliente @firma se inicia estando ya un perfil activo (puesto que se inicia desde un Firefox cuando ya hay un perfil cargado), es el estado de activo o no activo el factor que se usa para seleccionar el perfil a usar.

Un perfil se considera activo cuando su directorio (que en este ejemplo, y para el primer perfil sería Profiles/ior5ad1g.default-1435315801996) contiene un fichero llamado `parent.lock` o `lock`.

Adicionalmente, es importante recalcar que en una exportación o copia de perfiles (tanto si se hace de forma manual como si se usan las herramientas para tal fin de Firefox) pueden quedarse varios perfiles como activos simultáneamente (varios perfiles contendrán un fichero llamado `parent.lock` o `lock`, aunque solo debería contenerlo uno entre todos los perfiles).

En estos casos, el MiniApplet Cliente @firma seleccionará el primer perfil activo que encuentre (siguiendo el orden establecido en `profiles.ini`), y si se desea alterar este comportamiento será necesario subsanar el problema eliminando todos los ficheros `parent.lock` o `lock` de todos los directorios de perfil estando Firefox detenido.

4 Despliegue del MiniApplet @firma

Para el uso del MiniApplet son necesarios 2 ficheros principalmente:

- `miniapplet-full_X.jar`
 - Es el archivo Java en el que se encuentra el *applet* y toda la funcionalidad de la aplicación.
 - 'X' es el número de versión del *applet*.
- `miniapplet.js`
 - Es la biblioteca JavaScript que permite la integración y uso del MiniApplet @firma en una página Web.

Para el despliegue del MiniApplet Cliente @firma debe publicar estos ficheros en su servidor Web. 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 la biblioteca JavaScript de despliegue:

- `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.

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://miweb.com/afirma/miniapplet.js">

  </script>

...
```

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

- `miniapplet.js`: <http://miweb.com/afirma/miniapplet.js>

Si la página Web en la que deseamos cargar el MiniApplet Cliente @firma 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 esta biblioteca JavaScript importada 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 la biblioteca JavaScript de despliegue mediante una URL absoluta.

...

```
<script type="text/javascript" src="http://miweb.com/afirma/miniapplet.js">  
</script>
```

...

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:

- 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 desde una función JavaScript desplegado en la dirección <http://www.miweb.com/afirma>.

```
function cargar() {  
    MiniApplet.cargarMiniApplet("http://www.miweb.com/afirma");  
}
```

- 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 Configuración del idioma

El MiniApplet detecta el idioma del sistema del usuario, de tal forma que si se dispone de los textos traducidos a ese idioma, los mostrará traducidos a los usuarios en los distintos diálogos gráficos. Sin embargo, es posible forzar que se utilice un idioma concreto de los que dispone.

El idioma se puede configurar mediante el método JavaScript:

```
setLocale(locale)
```

Este método debe invocarse a través del objeto `MiniApplet` y recibe como parámetro:

- `codebase`
 - Código del idioma que se desea establecer según la ISO 639.

Este método deberá invocarse siempre antes del método de carga del MiniApplet y del método `checkTime`, en caso de usarse.

Por ejemplo:

- Configuración del idioma gallego:

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


Si no se dispone de los textos del MiniApplet traducidos al idioma configurado, se usarán los textos del idioma por defecto (Español).

Actualmente, el MiniApplet dispone de los textos traducidos en los siguientes idiomas:

- Español/Castellano (es_ES) (Idioma por defecto)
- Gallego (gl_ES)

Los siguientes métodos JavaScript permiten al integrador establecer explícitamente mensajes y textos que aparecerán al usuario en los distintos diálogos gráficos del applet. Es responsabilidad suya establecer estos textos en el idioma que corresponda en el momento de invocar a estos métodos:

- `saveDataToFile (dataB64, title, fileName, extension, description)`
- `getFileNameContentBase64 (title, extensions, description, filePath)`
- `getMultiFileNameContentBase64 (title, extensions, description, filePath)`

Consulte la documentación de estos métodos para conocer la descripción de cada uno de los parámetros configurables.

4.4 Restricción según desfase horario con el servidor

Al realizar una firma en el equipo del usuario, esta registra el momento de la firma (*signingTime*) usando la hora del propio equipo. En caso de que la hora y/o fecha del equipo se encuentre mal configurada, es posible que una validación posterior de la firma provoque errores, sobre todo si se trabaja también con sellos de tiempo.

El MiniApplet no puede modificar la hora del sistema del usuario, pero si puede advertirle de esta situación para que la corrija, o incluso bloquear la carga.

Para hacer esta comprobación se puede utilizar el método JavaScript:

```
checkTime(checkType, maxMillis)
```

Este método debe invocarse a través del objeto `MiniApplet` y recibe como parámetros:

- `checkType`
 - Tipo de verificación que se desea realizar. Admite los valores:
 - `MiniApplet.CHECKTIME_NO`: No realiza ningún tipo de comprobación.
 - `MiniApplet.CHECKTIME_RECOMMENDED`: Realiza la comprobación horaria y, en caso de encontrar un desfase, pedirá al usuario que lo corrija antes de continuar.
 - `MiniApplet.CHECKTIME_OBLIGATORY`: Realiza la comprobación horaria y, en caso de encontrar un desfase, bloqueará la carga del Applet y pedirá al usuario que lo corrija.
- `maxMillis`

- Milisegundos máximos que se permiten de desfase. Se recomienda que se indique un periodo mínimo de 1 minuto (60.000 milisegundos) para facilitar la corrección de la hora en el equipo del usuario.

4.5 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/>

Es preferible en cualquier caso utilizar certificados aptos para firma de aplicaciones que estén reconocidos por Oracle como de confianza.

4.6 Despliegue en entornos de Web dinámica, servidores de aplicaciones y en general servidores no estáticos

El MiniApplet Cliente @firma necesita que todos sus recursos (JAR de Java, ficheros JavaScript y página HTML de despliegue) se encuentren en el mismo directorio (misma ruta Web). Cuando se despliega este en servidores Web no estáticos (como un servidor de Portlets, un servidor de páginas activas de Microsoft, etc.), es responsabilidad del integrador hacer que los recursos puedan ser referenciados tal y como si estuviesen en un servidor estático o bien modificar tanto HTML como JavaScript para introducir referencias absolutas donde pudiese ser necesario.

Como norma general, no se proporciona soporte técnico para problemas de despliegue en entornos Web con servidores no estáticos, como pueden ser:

- Servidores de aplicaciones usando páginas dinámicas (JSP, JSF, ASP, etc.).
- Mapeos virtuales de los directorios que puedan afectar a los recursos del MiniApplet Cliente @firma.
- Servidores de Portlets.
- Etc.

5 El proceso de firma electrónica

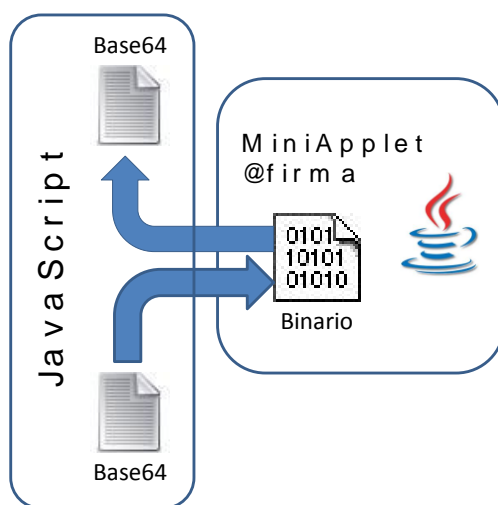


Cualquier firma electrónica realizada mediante el MiniApplet sigue un proceso simple de 4 pasos.

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 decodificación binaria de estos datos en Base64.



Esto significa que los datos firmados realmente no serán los que nosotros proporcionemos en Base64, sino su decodificación binaria. Así, por ejemplo, si la cadena de texto `SOY UN TEXTO A FIRMAR` tiene como representación en base64 la cadena `U09ZIFVOIFRFRWFRPIEEgRklSTUFS`, debemos establecer como datos a firmar siempre `U09ZIFVOIFRFRWFRPIEEgRklSTUFS`, 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 decodificar 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 recomienda 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 [Selección de un fichero por parte del usuario y recuperación de su nombre y contenido](#) 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 los 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 de 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 (pero no en firmas simples), 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 decodificación 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 decodificaciones pertinentes.

En la llamada a `saveDataToFile()` pueden indicarse los textos que debe mostrar el diálogo, por lo que debe tener la precaución, por coherencia, de usar el mismo idioma que ha configurado para el MiniApplet.

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 MiniApplet Cliente @firma 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 de operación del MiniApplet Cliente @firma 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 síncrona 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 la 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 Cliente @firma.

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
    var message = MiniApplet.getErrorMessage();
    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 dos parámetros que serán el resultado devuelto por la operación (la firma, cofirma o contrafirma generada en base64) y el certificado utilizado para firmar (codificado en base64). Esta puede tener la forma:

```
function successCallback(signatureBase64, certificateB64) {  
    ...  
}
```

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 (signatureB64) {  
    // Guardamos la firma en un campo de un formulario  
    document.getElementById("result").value = signatureB64;  
    // 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.
 - Cuando los datos a firmar se encuentran en una URL (HTTP o HTTPS) accesible en el mismo dominio que el Applet, es posible indicar directamente esta URL en vez de los datos a firmar. En este caso, el programa JavaScript realizará automáticamente la descarga de esos datos desde la URL y su conversión a Base64 para su posterior firma.
 - Si se usa el MiniApplet Cliente @firma esta URL tendrá las mismas restricciones de acceso que las por defecto para JavaScript del navegador y el servidor Web, lo cual, en condiciones normales, obliga a que los datos estén en el mismo dominio que la página Web que aloja el MiniApplet, y debe ser accesible por el mismo protocolo (HTTP o HTTPS).

- Si se usa la aplicación AutoFirma, no existirá ninguna restricción de acceso (excepto las definidas por los sistemas cortafuegos o de seguridad y la configuración general del entorno en el que se ejecuta).
- En el caso de ejecutarse AutoFirma en lugar del MiniApplet @firma (véase [Compatibilidad con dispositivos móviles y AutoFirma](#)), la URL proporcionada en lugar de los datos se le pasará a AutoFirma para que sea la propia aplicación la que descargue los datos.
- **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 [Gestión de errores](#).

El resultado de esta operación puede obtenerse directamente o gestionarse 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.
// Advertencia: Esta operación no se puede ejecutar en AutoFirma desde Chrome
// porque no soporta 2 llamadas al applet sin interacción con el usuario.
function saveSignatureFunction (signatureB64, certificateB64) {
    MiniApplet.saveDataToFile(signatureB64, "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", "CAAdES", 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", "CAAdES", 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", "CAAdES", 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 en cofirmas y contrafirmas (el parámetro no es válido en firmas simples). Este valor sólo reproduce el formato, no las propiedades de la firmas originales. 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 [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 envía
function saveDataFunction (cosignB64, certificateB64) {
    document.getElementById("resultId").value = cosignB64;
    document.getElementById("firmante").value = certificateB64;
    document.getElementById("formulario").submit();
}
...
// 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);
```

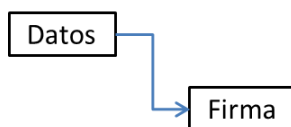
En esta función:

- *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 en cofirmas y contrafirmas (el parámetro no es válido en firmas simples). 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 [Gestión de errores](#).

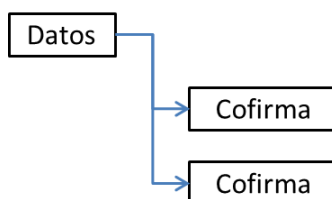
El resultado de esta operación puede obtenerse directamente o gestionarme mediante las funciones *callback*. Este resultado se obtiene codificado en base64.

6.4.2.1 Creación del árbol de firmas

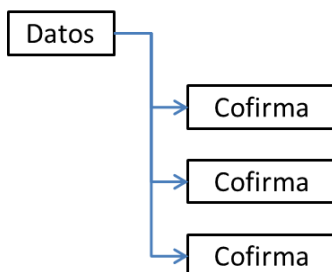
La operación de contrafirma se realiza sobre otra firma. Esta puede ser una firma simple, una cofirma u otra contrafirma. Estas operaciones de firma, cofirma y contrafirma van agregando firmas a un documento y, ya que las contrafirmas se realizan sobre firmas previas, se forma lo que se ha dado en llamar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



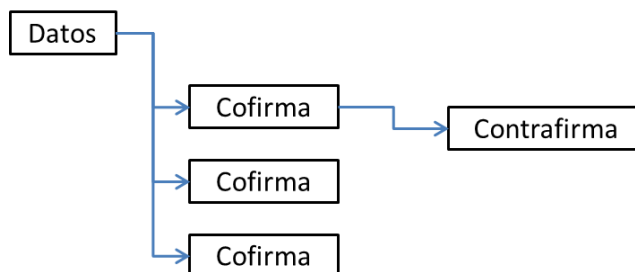
Si realizásemos una cofirma sobre el resultado de la operación anterior obtendríamos lo siguiente:



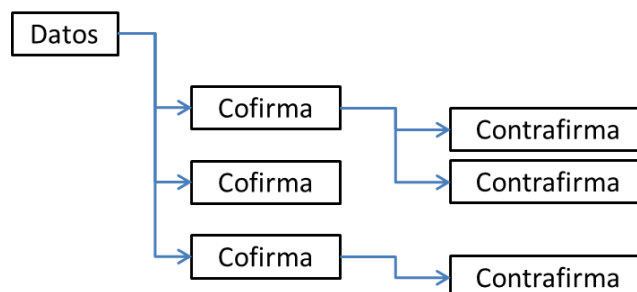
Tenga en cuenta que una firma y una cofirma están al mismo nivel y son equivalentes. No tiene importancia cual fue la primera en realizarse (firma) y cual la siguiente o siguientes porque todas son cofirmas. Las cofirmas son firmas sobre los datos originales, por lo tanto todas dependen de estos. Así,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

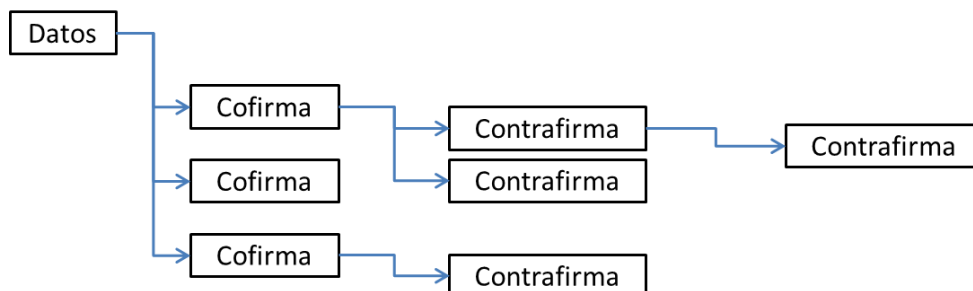


Una persona que contrafirme, puede estar interesada en contrafirmar más de una firma simultáneamente. Este sería el caso de un notario que valida con la suya las firmas de las dos partes de un contrato. Esto se representa con una firma sobre cada una de las firmas que se contrafirman, que no tienen por 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:

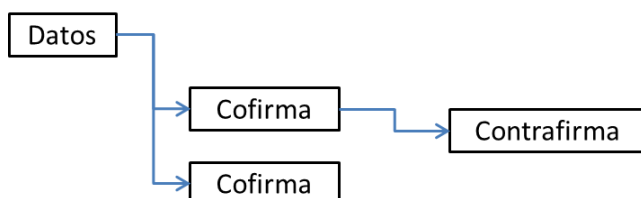
- **Nodos hoja del árbol (LEAFS):** Se firmarán sólo las firmas del árbol de los que no depende ningún otro nodo.
- **Todo el árbol de firma (TREE):** Se firman todos los nodos de firma del árbol.

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 apartado “Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma”.

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

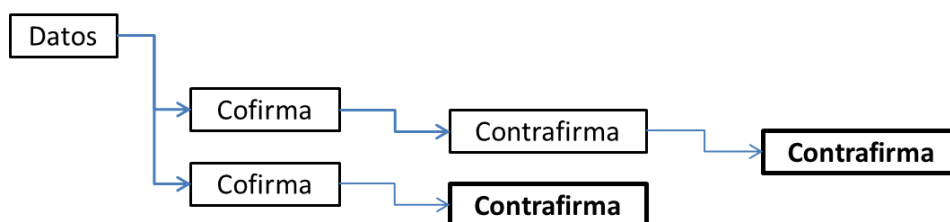
- `leafs`: Contrafirma todas las firmas que sean nodos hoja del árbol. Este es el valor por defecto.
- `tree`: Contrafirma todas las firmas del árbol.

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

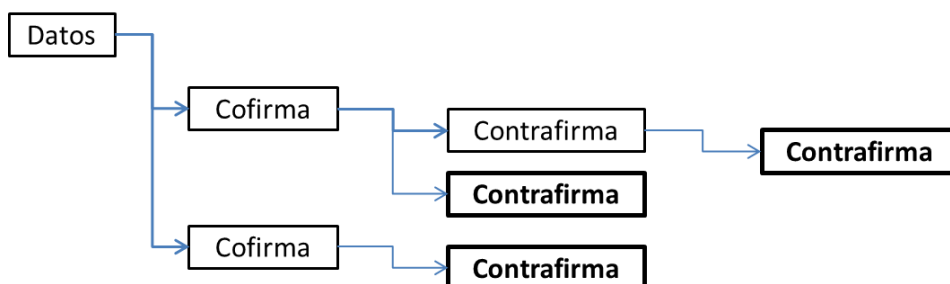


Cada una de las configuraciones dará el siguiente resultado:

- `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, successCallback, errorCallback
);
...

```

6.4.2.3.2 Contrafirma electrónica del resultado de una firma:

```
...
// Advertencia: Esta operación no con AutoFirma ejecutado desde Chrome sin
// que el usuario intervinese entre ambas operaciones
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", "CAAdES", "target=tree", successCallback,
    errorCallback
);
...

```

6.5 Firmas/Multifirmas trifásicas

El MiniApplet @firma, siguiendo la operativa normal, realiza las firmas electrónicas siguiendo los siguientes pasos:

1. Construye la estructura que el usuario debe firmar según el formato de firma y la configuración seleccionada.
2. Realiza la firma digital de esos datos con el certificado del usuario.
3. Finalmente, compone la firma en el formato de firma electrónica configurado.

Sin embargo, el MiniApplet y AutoFirma también permiten que la primera y tercera de las fases mencionadas se realicen de forma externa (en un servidor remoto), mientras que la segunda fase, la firma digital con el certificado del usuario, se realiza internamente. Esta división del proceso de firma en 3 fases es lo que comúnmente se llama **Firma Trifásica**.

Esta operativa resulta de mucho interés en determinados casos:

- El **origen y/o el destino de la información es un servidor**, de tal forma que se pueden pre-procesar los datos en servidor (Fase I) y mandar al usuario sólo la información mínima necesaria, firmarla el usuario en su sistema (Fase II) y, con el resultado, componer la firma en servidor (Fase III) para seguidamente enviarla a donde corresponda.
- Se **necesita firmar documentos muy grandes**. La firma trifásica interesa en este caso porque la mayor carga de proceso recaería sobre el servidor y no sobre el sistema del usuario que presuntamente tendrá muchos menos recursos.

El uso de la firma trifásica en estos casos conlleva una serie de ventajas y desventajas:

- **Ventajas**
 - El documento no viaja a través de la red.
 - Mayor facilidad para desarrollos sobre dispositivos móviles y menos propenso a errores debido a que la parte cliente no se vería expuesta a las muchas variables del entorno que pueden afectar a los distintos formatos de firma (versiones preinstaladas de bibliotecas, cambios en Java,...). Las operaciones más complejas se realizan en servidor, un entorno mucho más controlado.
- **Desventajas:**
 - Implica un mayor número de conexiones de red, aunque el tráfico de red, según el caso, podría ser menor.
 - Requiere el despliegue de servicios en el servidor.

Como nota importante, debe recalcar que el procedimiento de firma trifásico es útil únicamente cuando los ficheros residen en servidor y se implementan las clases adecuadas para su obtención y almacenamiento. Cuando el documento a firmar reside en cliente, no solo se produce un innecesario tráfico de red, sino que se aumenta la posibilidad de fallo y se incrementan las necesidades de memoria del MiniApplet.

6.5.1 Realizar firma trifásicas con el MiniApplet Cliente @firma

Para obligar al MiniApplet @firma o AutoFirma a generar una firma de forma trifásica es necesario realizar las siguientes acciones.

- **Establecer el formato de firma trifásica**
 - Para la firma en los formatos CAdES, PAdES y XAdES se usará CAdEStri, PAdEStri y XAdEStri, respectivamente, como identificador del formato.

- La versión actual no soporta la firma de facturas electrónicas en forma trifásica. El formato “FacturaEtri” no existe. La firma de facturas electrónicas tampoco puede realizarse mediante el formato XAdES, ni de forma monofásica, ni trifásica, independientemente de la política de firma que se utilice.
- **Configurar parámetro con la URL del servidor**
 - En los parámetros extra de la operación se deberá configurar la URL del servicio de firma trifásica. Este parámetro se configurará con la clave `serverUrl`.
 - Para saber más de los parámetros extra de configuración consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma.

6.5.1.1 Ejemplos:

A continuación se muestran algunos ejemplos de operación trifásica:

6.5.1.1.1 Firma trifásica XAdES:

```
...
MiniApplet.sign(
    dataB64, "SHA1withRSA", "XAdESTri", "format=XAdES
    Detached\nserverUrl=http://miweb.com/TriPhaseSignerServer/SignatureService",
    successCallback, errorCallback
);
...
```

6.5.1.1.2 Cofirma trifásica CAdES:

```
...
MiniApplet.coSign(
    signB64, dataB64, "SHA512withRSA", "CAdESTri",
    "serverUrl=http://miweb.com/TriPhaseSignerServer/SignatureService",
    successCallback, errorCallback
);
...
```

6.5.2 Servicio de firma trifásica

Este servicio es el que realiza la primera y tercera fase del proceso de firma trifásica. Junto al MiniApplet @firma se distribuye el archivo WAR “afirma-server-triphase-signer.war” que despliega un servicio web de tipo REST con las funcionalidades de firma trifásica. Este servicio no es dependiente de ningún servidor de aplicaciones concreto. Consulte el manual de su servidor de aplicaciones para saber cómo desplegar este fichero WAR.

El servicio de firma trifásica distribuido junto al MiniApplet soporta los formatos CAdES, XAdES y PAdES (firmas y multifirmas) y admite las mismas opciones de configuración que las firmas monofásicas de estos formatos.

Advertencia: Se han detectado problemas con JBoss 7 debido a que incorpora de serie la versión 1.5.X de la biblioteca XMLSec, que introduce un problema de compatibilidad con las firmas XAdES.

Si está seguro de que ninguna de sus aplicaciones hace uso de esta biblioteca, elimínela para solventar este problema.

Es muy recomendable habilitar el servicio de firma trifásica cuando se realizan despliegues compatibles con el Cliente @firma móvil. Para saber más acerca del Cliente @firma móvil consulte el apartado Compatibilidad con dispositivos móviles.

Algunos servidores de aplicaciones incorporan bibliotecas que se cargan automáticamente en cualquier despliegue realizado, lo que puede conllevar a problemas de compatibilidad con el servicio de firma trifásica, en especial con la firma XAdES. Uno de estos servidores es JBoss en sus versiones 7 y EAP 6. Puede consultar como solventar este problema en el apartado El servicio de firma trifásica genera un error al realizar firmas XAdES en servidores JBoss u otros.

6.5.2.1 Configuración del servicio de firma trifásica

Ta como se distribuye, el servicio de firma trifásica no necesita configuración para permitir generar firmas trifásicas de la forma normal del MiniApplet @firma. Esto es, cuando el integrador pase como parámetro los datos que desea firmar y quiera recibir la firma resultante.

Existe la posibilidad, sin embargo, de configurar el servicio trifásico, mediante el fichero `config.properties` contenido en el WAR, para establecer un *Document Manager* (Administrador de documentos) distinto que recoja los datos de otra fuente y los envíe a otro destino.

Integrados en el servicio trifásico se distribuyen 2 *Document Manager* distintos que el integrador del servicio puede configurar mediante la clave `"document.manager"` del fichero de propiedades:

- `es.gob.afirma.triphase.server.SelfishDocumentManager`: Es el *Document Manager* por defecto y emula el comportamiento de la firma monofásica del cliente. Es decir, recibe los datos a firmar y devuelve la firma.
- `es.gob.afirma.triphase.server.FileSystemDocumentManager`: En este caso se enviaría, en lugar de los datos a firmar, el nombre de un fichero localizado en el servidor para que sea el que se firme. La firma resultante se guardará en otro directorio del servidor y se devolverá al MiniApplet únicamente la cadena "OK" para confirmar que la operación finalizó correctamente.
 - Este *Document Manager* permite que los datos (cuyo origen y destino sea un servidor) no viajen hasta el lado cliente reduciendo considerablemente el tráfico de red y la fiabilidad y velocidad del proceso.
 - Si se configura este *Document Manager* se pueden configurar otras tres propiedades en el fichero de configuración del servicio:
 - `indir`: Directorio del servidor en donde se encuentran los documentos de datos.

- `outdir`: Directorio del servidor en donde se almacenan las firmas generadas.
- `overwrite`: Configura si se deben sobrescribir los ficheros de firma si ya existe una con el mismo nombre (`true`) o no (`false`).

El *Document Manager* “`es.gob.afirma.triphase.server.FileSystemDocumentManager`” se proporciona únicamente a modo de ejemplo de cómo puede implementarse el servicio usando un simple sistema de ficheros, pero no es un desarrollo preparado para llevarse a producción (por motivos de seguridad, escalabilidad, etc.).

Un desarrollador Java podría crear nuevos *Document Manager* a medida e integrarlos en el servicio. Para ello deberá implementar la interfaz `es.gob.afirma.triphase.server.DocumentManager`. Por ejemplo, se podría crear un *Document Manager* que recogiese los datos a firmar de un gestor de contenidos y almacenase la firma resultante en base de datos.

6.5.2.1.1 Notas específicas para configuración del ejemplo “`FileSystemDocumentManager`”

6.5.2.1.1.1 *Parámetros de uso y descripción del funcionamiento*

`FileSystemDocumentManager` es un simple simulador de repositorio de contenidos que funciona sobre un sistema de ficheros, donde el identificador del documento es realmente el nombre del fichero, tanto en la entrada (fichero a firmar) como en la salida (fichero con la firma hecha).

En él, los nombres de ficheros se indican codificados en Base64, para evitar problemas de codificación, teniendo por ejemplo:

| | | |
|----------------------------|-------------------|-----------------------------------|
| <code>documento.pdf</code> | se indicaría como | <code>ZG9jdW11bnRvLnBkZg==</code> |
| <code>firma.xsig</code> | se indicaría como | <code>ZmlybWEueHNpZw==</code> |

El servicio devuelve también, codificado en Base64, el nombre del fichero (sin ruta, suponiendo que esta es siempre `outdir`) en el cual se ha almacenado la firma resultante.

Estos ficheros se leen y escriben siempre tomando como base los directorios indicados (`outdir` e `indir`), pero por tratarse de un ejemplo, no se controla que se usen rutas relativas para acceder a partes privadas del sistema operativo (por ejemplo, indicando “`../../../../../../../../etc/passwd`”), siendo responsabilidad del integrador, implementar las medidas de seguridad apropiadas.

Es importante tener en cuenta que los nombres de fichero utilizados deben cumplir las restricciones del sistema de ficheros donde residan `outdir` e `indir`. Así, por ejemplo, en un sistema de ficheros NTFS no deberíamos nunca indicar un nombre de ficheros que contuviese el carácter “dos puntos” (“:”).

6.5.2.1.1.2 Configuración en alta disponibilidad con varios nodos

Los directorios configurados para el despliegue del servicio trifásico (`outdir` e `indir`) deben ser siempre directorios visibles y compartidos por todas las instancias en ejecución.

Este aspecto es especialmente importante en configuraciones de servidores de aplicaciones en alta disponibilidad, donde puede haber varios nodos que presten el servicio trifásico, cada uno de ellos en un sistema de ficheros diferente, donde sí se especifica una ruta local, puede que esta apunte a un directorio distinto en cada nodo (distinto servidor, disco diferente, otro sistema de ficheros, etc.).

El que todos los nodos accedan al mismo directorio referenciado en la configuración se puede lograr fácilmente usando un almacenamiento compartido entre todos ellos (con el mismo punto de montaje), mediante enlaces simbólicos, etc. Es importante también asegurarse de que todos los nodos tienen los permisos adecuados sobre los directorios configurados.

6.6 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. El valor "AUTO" no es válido para firmas simples.

6.7 Gestión de ficheros

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

6.7.1 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 extensión 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.

6.7.1.1 Ejemplos:

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

6.7.1.1.1 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);  
...
```

6.7.1.1.2 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.

6.7.2 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, filePath);
```

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.
- *filePath*: Opcional. Ruta absoluta del fichero que se debería seleccionar por defecto o sólo el nombre de fichero sugerido.

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”, (“U09ZIFVOIFRFWRPIEEgRklSTUFS” si lo codificamos en base 64) el método devolvería el texto “`entrada.txt|U09ZIFVOIFRFWRPIEEgRklSTUFS`”.

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 firmar, cofirmar o contrafirmar y el MiniApplet permitirá al usuario cargar un fichero de datos/firma sobre el que operar.

6.7.2.1 Ejemplos:

6.7.2.1.1 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("|"));
...
```

6.7.2.1.2 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);
...
```

6.7.2.1.3 Carga de un fichero con ruta preseleccionada

...

```
var fileNameContentB64;  
try {  
    fileNameContentB64 = MiniApplet.getFileNameContentBase64(  
        "Seleccionar fichero", "jpg,gif,png", "Imagen", "C:/pruebas/Entrada.png"  
    );  
} catch (e) {  
    return;  
}  
  
var filename = fileNameContentB64.substring(0, fileNameContentB64.indexOf("|"));  
...
```

6.7.3 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, filePath);
```

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.
- *filePath*: Opcional. Ruta absoluta de uno de los ficheros que se debería seleccionar por defecto o sólo el nombre sugerido de uno de los ficheros.

Este método devuelve un array de elementos en donde cada uno de ellos sigue el patrón:

Nombre|Contenido

Esto es, el nombre del fichero y su contenido en base 64 separados por el carácter '|'.
Por ejemplo, si se seleccionasen los ficheros entrada.txt e imagen.jpg, se obtendría un array con los elementos:

```
entrada.txt|U09ZIFVOIFRFWFRPIEEgRklSTUFS
```

```
imagen.jpg|A1NSHA1NQW212ASYAN45YMD2MWQEI
```

Seguido del texto `entrada.txt` y separado por el carácter '|' aparece el contenido de ese fichero convertido a base 64 y seguido del 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.

6.7.3.1 Ejemplos:

6.7.3.1.1 Carga de ficheros y recogida de sus nombres

```
...
var fileNameContentB64Array;
try {
    fileNameContentB64Array = 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.getErrorMessage())) {
        /* Método del integrador para mostrar logs */
        showLog(MiniApplet.getErrorMessage());
    }
    return;
}

var filenames = new Array();
for (var i = 0; i < fileNameContentB64Array.length; i++) {
    var separatorIdx = fileNameContentB64Array[i].indexOf("|");
    filenames[i] = fileNameContentB64Array[i].substring(0, separatorIdx);
}
...
```

6.7.3.1.2 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.getErrorMessage())) {
        /* 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;
    }
}
...
```

6.8 Utilidad

6.8.1 Eco

El MiniApplet dispone del método `echo()` que devuelve la cadena:

```
Java vendor: JAVA_VENDOR
Java version: JAVA_VERSION
Java architecture: JAVA_ARCHITECTURE
```

Llamaremos a este método de la forma:

```
MiniApplet.echo();
```

La cadena devuelta por el método `echo()` cuando el MiniApplet se encuentra cargado muestra el fabricante de la máquina virtual, la versión de java que se ejecuta y su arquitectura. Esta misma información se imprime en la consola Java al ejecutar el método precedida por el mensaje:

```
MiniApplet cargado y en ejecución
```

El principal propósito de este método es que los integradores puedan llamarlo para comprobar si el Applet está correctamente cargado y comprobar la versión de Java instalada. En caso contrario, fallará su ejecución.

Cuando el método se invoca desde un entorno en el que no se pueden ejecutar applets, como en un dispositivo móvil, o cuando no se hayan concedido los permisos para la ejecución del applet, el texto devuelto por la función será:

```
Cliente JavaScript
```

6.8.2 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.

6.8.3 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.
 - Este valor por defecto es dependiente del entorno operativo (sistema operativo y navegador Web), por lo que no se recomienda su uso.
 - Si desea que se intente detectar el juego de caracteres automáticamente, indique “auto”.

Este método devuelve una cadena con el texto plano correspondiente.

6.8.3.1 Ejemplos

6.8.3.1.1 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);  
...  

```

6.8.4 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.
 - Este valor por defecto es dependiente del entorno operativo (sistema operativo y navegador Web), por lo que no se recomienda su uso.
 - 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.

6.8.4.1 Ejemplos

6.8.4.1.1 Firma de un texto insertado por el usuario

```
...  
var text = "Hola Mundo!!";  
var dataB64 = MiniApplet.getBase64FromText(text, "utf-8");  
  
MiniApplet.sign(dataB64, "SHA1withRSA", "CAAdES", 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.

7.1.1 Parámetros adicionales no soportados

Revise con cuidado los parámetros admitidos en cada uno de los formatos de firma y en cada una de las operaciones. Si se configura un parámetro no soportado simplemente será ignorado sin ningún mensaje de error en registro.

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 filtros de certificados que se pueden aplicar para restringir los certificados que podrá seleccionar el usuario para realizar una operación de firma o multifirma. Los filtros de certificados se pueden establecer como parámetros adicionales en las operaciones de firma, cofirma y contrafirma. Las claves que nos permiten establecer filtros de certificados son:

- *filter*: Esta clave permite establecer uno y sólo uno de los filtros de certificados que se listan más adelante en este apartado. Por ejemplo:

- *filter=nonexpired*:

- Certificados no caducados

- *filters*: Esta clave permite establecer uno o más de los filtros de certificados que se listan más adelante en este apartado. Los certificados deberán cumplir las condiciones establecidas en todos los certificados listados, o de lo contrario no se mostrarán. Los distintos filtros se deben separar mediante el carácter punto y coma (;). Ejemplos:

- *filters=nonexpired*:

- Certificados no caducados

- *filters=issuer.rfc2254: (O=DIRECCION GENERAL DE LA POLICIA);keyusage.nonrepudiation:true*

- Certificados de firma del DNle

- *filters=issuer.rfc2254: (O=DIRECCION GENERAL DE LA POLICIA);keyusage.nonrepudiation:true;nonexpired*:

- Certificados de firma del DNle no caducados.

- *filters.X*: En esta clave 'X' será un entero igual o mayor que 1. El MiniApplet leerá la clave *filters.1*, a continuación *filters.2* y así hasta que no encuentre una de las claves de la secuencia. Al contrario que con la clave *filters*, basta con que el certificado cumpla uno de estos filtros para que se muestre. No es necesario cumplirlos todos. Cada uno de estas

claves puede declarar varios filtros separados por punto y coma (;) de tal forma que sí se deberán cumplir todos ellos para satisfacer ese subfiltro concreto. Ejemplo:

- `filters.1=issuer.rfc2254:(O=DIRECCION GENERAL DE LA POLICIA);keyusage.nonrepudiation:true`
- `filters.2=issuer.rfc2254:(O=FNMT)`

- La conjunción de estas dos claves en una operación de firma hará que sólo se muestren al usuario los certificados CERES y el de firma del DNle.

Estas tres claves de definición de filtros son excluyentes y tienen la prioridad según la que se listan (*filter*, *filters* y *filters.X*). Es decir, si se establece la propiedad *filter*, no se procesarán las propiedades *filters* y *filters.1*, por ejemplo.

Los filtros disponibles en el MiniApplet son:

- **Filtro DNle:** Filtra los certificados del almacén para que sólo se muestren los certificados de firma de los DNle disponibles desde ese almacén.
 - Para establecer este filtro de certificados se indicará la propiedad *filter* con el valor *dnle*: en el parámetro de configuración de la operación de firma, cofirma o contrafirma.
 - Ejemplo:
 - `filter=dnle:`
- **Filtro de certificados de firma:** Filtra los certificados del almacén para que no se muestren los considerados certificados de autenticación. Esta exclusión no se realiza mediante KeyUsage para evitar que queden excluidos certificados mal identificados. Un ejemplo de certificado que no se mostrará en el diálogo es el de autenticación del DNle.
 - Para establecer este filtro de certificados se indicará la propiedad *filter* con el valor *signingCert*: en el parámetro de configuración de la operación de firma, cofirma o contrafirma.
 - Ejemplo:
 - `filter=signingCert:`
- **Filtro de certificados de autenticación:** Filtra los certificados del almacén para que no se muestren los específicos para firma avanzada reconocida. Esta exclusión no se realiza mediante KeyUsage para evitar que queden excluidos certificados mal identificados. Un ejemplo de certificado que no se mostrará en el diálogo es el de firma del DNle.
 - Para establecer este filtro de certificados se indicará la propiedad *filter* con el valor *authCert*: en el parámetro de configuración de la operación de firma, cofirma o contrafirma.
 - Ejemplo:
 - `filter=authCert:`

- Filtro de certificados SSCD: Filtra los certificados del almacén para que se muestren sólo aquellos emitidos por medio de un dispositivo SSCD (dispositivo seguro de creación de firma), como es el caso de los certificados del DNle. Hay que tener en cuenta que el filtrado se realiza a partir de un atributo QCStatement declarado en el propio certificado. Si la autoridad de certificación no incluye este atributo, no será posible realizar la distinción.
 - Para establecer este filtro de certificados se indicará la propiedad *filter* con el valor *sscd*: en el parámetro de configuración de la operación de firma, cofirma o contrafirma.
 - Ejemplo:
 - `filter=sscd:`
- 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 DNle, se mostrará en su lugar el certificado de firma de ese mismo DNle.
 - 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 de 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 de 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`
- Filtro de certificados caducados: Filtra aquellos certificados que se encuentran fuera de su periodo de validez para que sólo se muestren los certificados vigentes, que son los únicos que pueden generar una firma válida.
 - Para establecer este filtro se usará la palabra clave *nonexpired*:
 - Ejemplo:
 - `filter=nonexpired:`

- Filtro por huella digital (Thumbprint): Filtra los certificados de tal forma que sólo se mostrará aquel que tenga la huella digital indicada. Hay que tener en cuenta que esta huella digital no debe calcularse en base a un fichero (por ejemplo, un “.cer”), sino que es la huella digital de la codificación del certificado.
 - Para establecer este filtro se usará la palabra clave *thumbprint:*, seguida del algoritmo de huella digital utilizado y, separado por el carácter dos puntos (‘:’) la huella digital que se busque en hexadecimal.
 - Ejemplo:
 - `filter=thumbprint:SHA1:30 3a bb 15 44 3a fd d7 c5 a2 52 dc a5 54 f4 c5 ee 8a a5 4d`
 - Este filtro sólo mostrará el certificado cuya huella digital en SHA1 sea la indicada.
- Filtro RFC2254 en base al *Subject* del certificado: Filtra los certificados a partir de una expresión regular creada según la RFC2254 que se aplica sobre el *Subject* del certificado.
 - Para establecer este filtro se usará el valor *subject.rfc2254*: seguido de la expresión RFC2254.
 - Puede revisarse la normativa RFC 2254 en <http://www.faqs.org/rfcs/rfc2254.html>
 - Ejemplo:
 - `filter=subject.rfc2254:(CN=*12345678z*)`
 - Este filtro mostrará sólo aquellos certificados en los que aparezca la cadena “12345678z” en el *CommonName* de su *Subject*.
- Filtro RFC2254 en base al *Issuer* del certificado: Filtra los certificados a partir de una expresión regular creada según la RFC2254 que se aplica sobre el *Issuer* del certificado.
 - Para establecer este filtro se usará el valor *issuer.rfc2254*: seguido de la expresión RFC2254.
 - Puede revisarse la normativa RFC 2254 en <http://www.faqs.org/rfcs/rfc2254.html>
 - Ejemplo:
 - `filter=issuer.rfc2254:(|(O=FNMT)(O=DIRECCION GENERAL DE LA POLICIA))`
 - Este filtro mostrará sólo aquellos certificados cuyo *Issuer* tenga establecido como organización “FNMT” o “DIRECCION GENERAL DE LA POLICIA”, es decir, sólo mostrará los certificados del DNle y los de CERES.
- Filtro de texto en base al *Subject* del certificado: Filtra los certificados según si contienen o no una cadena de texto en el *Principal* de su *Subject*.
 - Para establecer este filtro se usará el valor *subject.contains*: seguido de la cadena de texto que debe contener.
 - Ejemplo:
 - `filter=subject.contains:JUAN ESPAÑOL ESPAÑOL`

- Este filtro mostrará sólo aquellos certificados en los que aparezca la cadena “JUAN ESPAÑOL ESPAÑOL” en el *Subject*.
- Filtro de texto en base al *Issuer* del certificado: Filtra los certificados según si contienen o no una cadena de texto en el *Principal* de su *Issuer*.
 - Para establecer este filtro se usará el valor *issuer.contains*: seguido de la cadena de texto que debe contener.
 - Ejemplo:
 - `filter=issuer.contains:O=EMPRESA`
 - Este filtro mostrará sólo aquellos certificados en los que el *Principal* del *Issuer* muestre el texto “O=EMPRESA”.
- Filtros por uso declarado de los certificados (*KeyUsage*): Colección de filtros que permiten filtrar según el uso declarado de los certificados.
 - Para establecer estos filtros usaremos las siguientes claves según los usos que se quieran comprobar. Las claves irán seguidas de los valores “true” o “false”, según se desee que el uso esté habilitado o no lo esté, respectivamente:
 - *keyusage.digitalsignature*:
 - *keyusage.nonrepudiation*:
 - *keyusage.keyencipherment*:
 - *keyusage.dataencipherment*:
 - *keyusage.keyagreement*:
 - *keyusage.keycertsign*:
 - *keyusage.crlsign*:
 - *keyusage.encipheronly*:
 - *keyusage.decipheronly*:
 - Los *KeyUsages* que no se declaren en el filtro no se tendrán en cuenta.
 - Ejemplos:
 - `filters=keyusage.digitalsignature:true;keyusage.keyencipherment:true`
 - Este filtro mostrará sólo aquellos certificados que tengan establecidos a `true` los *KeyUsage* *digitalsignature* (autenticación) y *keyencipherment* (sobres electrónicos), ignorando el valor del resto de *KeyUsages*. Este filtro mostrará, por ejemplo, los certificados de la FNMT.
 - `filters=keyusage.nonrepudiation:true`
 - Este filtro mostrará sólo aquellos certificados que tengan establecidos a `true` el *nonrepudiation* (firma avanzada). Este filtro mostrará, por ejemplo, el certificado de firma del DNle.

Se ignorará cualquier valor establecido como filtro de certificados distinto a los que se han listado.

Si ningún certificado cumple 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 cumple los criterios de filtrado, se mostrarán todos ellos en el diálogo de selección de certificados.

Si tan sólo un certificado cumple con las condiciones de los filtros establecidos y 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 (CAAdES y XAdES) tienen una variante EPES (*Explicit Poliy-c-based Electronic Signature*) que declaran los atributos correspondientes a la política de firma.

El MiniApplet @firma permite la generación de firmas EPES (CAAdES-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 v1.9

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) v1.9. 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 (CAAdES, XAdES o PAdES).

- `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 manualmente.

IMPORTANTE: El parámetro `expPolicy` funciona únicamente en el MiniApplet Cliente @firma, y no en la aplicación AutoFirma, por lo que no se generarán firmas EPES en invocación por protocolo indicando ese parámetro.

Por ello, y hasta que se subsane esta deficiencia, se recomienda configurar manualmente la Política de Firma de la AGE con sus parámetros, que son los siguientes:

- **CAAdES**
 - `policyIdentifier=2.16.724.1.3.1.1.2.1.9`
 - `policyIdentifierHash= G7roucf600+f03r/o0bAOQ6WAs0=`
 - `policyIdentifierHashAlgorithm=1.3.14.3.2.26`
 - `policyQualifier=https://sede.060.gob.es/politica_de_firma_anexo_1.pdf`
 - `mode=implicit`
- **XAdES**
 - `policyIdentifier=urn:oid:2.16.724.1.3.1.1.2.1.9`
 - `policyIdentifierHash=G7roucf600+f03r/o0bAOQ6WAs0=`
 - `policyIdentifierHashAlgorithm=http://www.w3.org/2000/09/xmldsig#sha1`
 - `policyQualifier=https://sede.060.gob.es/politica_de_firma_anexo_1.pdf`
 - `format=XAdES Detached`
- **PAdES**
 - `policyIdentifier= 2.16.724.1.3.1.1.2.1.9`
 - `policyIdentifierHash= G7roucf600+f03r/o0bAOQ6WAs0=`
 - `policyIdentifierHashAlgorithm=http://www.w3.org/2000/09/xmldsig#sha1`
 - `policyQualifier=https://sede.060.gob.es/politica_de_firma_anexo_1.pdf`

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 CAAdES y los datos ocupen menos de 1 MB. Con tamaños mayores de datos no se incluirán estos en las firmas.

Si se configura para la operación alguna propiedad individual que entre en conflicto con la política indicada (por ejemplo, indicando un formato prohibido por esta), 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 la AGE v1.8

En el MiniApplet @firma se ha incluido también el mecanismo para usar la política de firma versión 1.8 de la AGE. Este mecanismo actúa exactamente igual que el de la política 1.9 salvo porque:

- La política de firma 1.8 de la AGE no está preparada para el formato de firma PAdES.
- El parámetro a configurar es:
 - `expPolicy=FirmaAGE18`

Esta propiedad se expandirá a las necesarias para el cumplimiento de la política de firma de la AGE según la política 1.8.

IMPORTANTE: El parámetro `expPolicy` funciona únicamente en el MiniApplet Cliente @firma, y no en la aplicación AutoFirma, por lo que no se generarán firmas EPES en invocación por protocolo indicando ese parámetro.

Por ello, y hasta que se subsane esta deficiencia, se recomienda configurar manualmente la Política de Firma de la AGE con sus parámetros, que son los siguientes:

- CAdES
 - `policyIdentifier=2.16.724.1.3.1.1.2.1.8`
 - `policyIdentifierHash=7SxX3erFuH3lTvAw9LZ70N7p1vA=`
 - `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`

Debe evitarse en la medida de lo posible el uso de versiones de la política de la AGE anteriores a la 1.9.

7.4.4 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 Configuración de parámetros de la JVM

El JavaScript de despliegue permite establecer propiedades que se le proporcionarán directamente a la máquina virtual para su configuración. Esto se hará mediante la propiedad:

`MiniApplet.JAVA_ARGUMENTS`

Actualmente, esta variable está configurada con los valores “-Xms512M -Xmx512M” que gestionan la memoria reservada para la JVM. Estos valores se pueden suprimir estableciendo otros, o anexar nuevos valores:

```
// Se pisan los valores existentes con nuevas propiedades para la JVM
MiniApplet.JAVA_ARGUMENTS = "-XX:ErrorFile=./hs_err_pid%p.log ";
```

...

```
// Se agregan nuevas propiedades para la JVM
MiniApplet.JAVA_ARGUMENTS += "-Xms512M -Xmx1024M ";
```

Para evitar que las variables se concatenen erróneamente, debe dejarse un espacio al final de la cadena.

7.6 Configuración a través de propiedades del sistema

El JavaScript de despliegue del MiniApplet permite establecer una serie de variables que serán usadas como propiedades del sistema en tiempo de ejecución. Estas variables permiten configurar comportamientos específicos del MiniApplet.

Para establecer estas variables, las asignaremos separadas por espacios a la propiedad:

`MiniApplet.SYSTEM_PROPERTIES`

El nombre de las variables deben ir antecedido por “-D” y tener la forma “clave=valor”. Por ejemplo:

```
MiniApplet.SYSTEM_PROPERTIES = "-DAFIRMA_NSS_HOME=C:/Program Files/Mozilla  
Firefox ";
```

Estas variables solo tienen efecto si se establecen antes de la carga del MiniApplet, si son establecidas con posterioridad se ignoran por completo.

8 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:

```
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, successCallback,  
        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.

9 Compatibilidad con dispositivos móviles y AutoFirma

Los navegadores Web de los dispositivos móviles no admiten complementos capaces de realizar firmas electrónicas (Applets de Java, ActiveX, etc.), por lo que es necesario implementar otros mecanismos para suplir esta carencia. En el caso del MiniApplet @firma, se han desarrollado aplicaciones nativas para sistemas móviles que, de forma transparente para el integrador, sustituirán al MiniApplet y realizarán las operaciones del firma.

Actualmente, existen versiones publicadas del Cliente @firma Móvil para sistemas Android 4 e iOS.

De igual manera, Google Chrome ya no soporta Applets de Java por defecto. Para mantener la compatibilidad con este navegador, debe usarse el mismo mecanismo que se usa en las aplicaciones móviles, en donde la aplicación nativa que sustituirá al MiniApplet será AutoFirma.

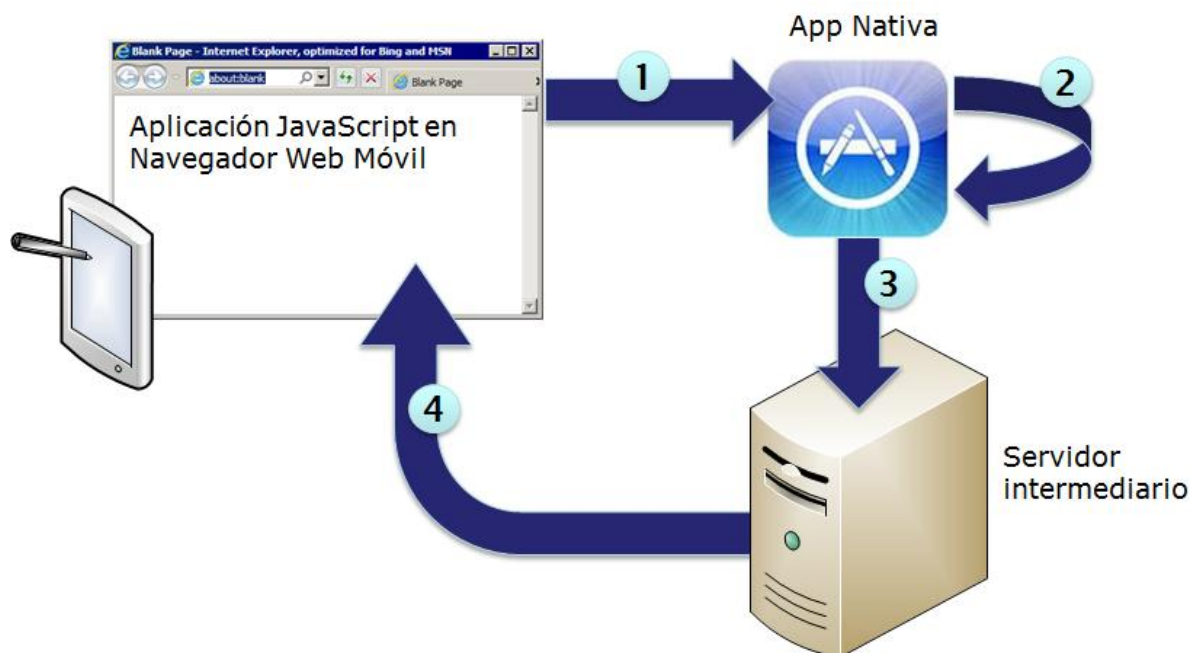
El usuario deberá tener instalador la aplicación móvil del Cliente @firma o AutoFirma, según su entorno, antes de iniciar la operación de firma. Es responsabilidad del integrador informar al usuario de esta necesidad antes de iniciar el proceso de firma (para evitar que inicie el trámite si no se tiene instalado).

No hay ninguna función para que el integrador pueda determinar si el usuario tiene instalada o no la aplicación AutoFirma.

9.1 Arquitectura del sistema

Dado que los sistemas móviles imponen limitaciones en la comunicación que puede establecerse entre el navegador web y una aplicación nativa, ha sido necesario idear una arquitectura de comunicación que permita el traspaso bidireccional de información entre la aplicación nativa y la página web.

La arquitectura es la siguiente:



1. Cuando detecta que está en un dispositivo móvil o no se puede cargar el MiniApplet @firma (navegador no compatible, no se tiene Java instalado, el usuario no dio permisos a la JVM...), en lugar de cargar el Applet, el navegador web realiza una **llamada por protocolo a la aplicación nativa** (previamente instalada) pasándole los datos necesarios para ejecutar la operación designada.
2. **La aplicación nativa realiza la operación** en base a la configuración proporcionada.
3. La aplicación nativa **envía el resultado de la operación a un servidor remoto** del integrador por medio de un servicio publicado en el mismo. Los datos, sean cuales sean (la firma, mensaje de éxito, mensaje de error,...), se envían cifrados.
4. **La página web descarga y borra el resultado** de la operación del servidor por medio de otro servicio web distinto al de guardado. La página descifra el contenido **y continúa con el flujo de operación** determinado por el integrador.

9.2 Despliegues compatibles

Para que nuestros despliegues del MiniApplet sean compatibles con los Cliente @firma nativos, de tal forma que las firmas se realicen mediante estos cuando el entorno de ejecución no permita la ejecución de Applets, deberemos seguir las siguientes pautas:

- Desplegar el MiniApplet mediante la biblioteca JavaScript “miniapplet.js” que se distribuye junto a él.
 - Aunque el MiniApplet puede desplegarse como un Applet normal y corriente, toda la lógica que se encarga de ejecutar el Cliente @firma móvil cuando no es posible ejecutar Applets y procesa las ordenes al Applet se encuentra en esta biblioteca. El cómo se despliega el MiniApplet con esta biblioteca se explica en el apartado [y si](#)

se desea alterar este comportamiento será necesario subsanar el problema eliminando todos los ficheros `parent.lock` o `lock` de todos los directorios de perfil estando Firefox detenido.

- Despliegue del MiniApplet @firma.
- Establecer las rutas de los servicios auxiliares.
 - La falta de un canal de comunicación bidireccional entre el navegador web y el Cliente @firma móvil o AutoFirma, al contrario de lo que ocurre entre el navegador web y un Applet, obliga a construir una arquitectura de comunicación basada en 2 servicios encargados de enviar y recoger mensajes, respectivamente. Para que el navegador y el Cliente @firma móvil hagan uso de este canal se deberán configurar la URL en la que se encuentran estos servicios.
 - La configuración de las URL de los servicios de comunicación se realizará mediante el método:
 - `setServlets(urlServletEnvio, urlServletRecepcion)`
 - La llamada a este método se deberá realizar inmediatamente después de la sentencia de carga del MiniApplet.
 - Esta sentencia es inocua para el MiniApplet. Los servicios sólo se usarán en caso de que se cargue la aplicación nativa.
 - Un ejemplo de uso de esta llamada es:

```
<script type="text/javascript">
    MiniApplet.cargarMiniApplet("http://miweb.com/afirma");
    MiniApplet.setServlets(
        "http://miweb.com/SignatureStorageServer/StorageService",
        "http://miweb.com/SignatureRetrieverServer/RetrieveService");
</script>
```
- Simplificar la operativa.
 - El Cliente @firma Móvil no implementa todavía algunos de los métodos y funcionalidades del MiniApplet. Por lo que conviene no utilizar estos métodos si no es necesario y buscar alternativas compatibles.
 - Consulte el apartado Limitaciones funcionales para conocer los métodos que no implementa el Cliente @firma móvil.

9.3 Servicios auxiliares del Cliente @firma móvil

Junto al MiniApplet se distribuyen 2 Servlets de Java que permiten el guardado y la recogida de datos en servidor para la compatibilidad con el Cliente @firma móvil. Estos Servlet son:

- **SignatureStorageServer:** Este Servlet permite almacenar datos en un directorio del servidor. Los datos se almacenan con un identificador.
- **SignatureRetrieverServer:** Este Servlet permite recuperar datos de un servidor a partir de un identificador. Tras devolver los datos, el servicio borra el fichero temporal en donde se almacenaban. Este servicio nunca devolverá datos que se guardasen hace más de un tiempo máximo configurado, devolviendo error tal como si no hubiese encontrado el fichero de datos. Igualmente, borrará todos aquellos ficheros del directorio temporal que hayan sobrepasado este tiempo máximo desde su creación.

Ambos servicios se pueden configurar mediante el fichero de propiedades `configuration.properties`, contenido por ambos. En este fichero podremos configurar las siguientes variables:

- **tmpDir:** Es el directorio del servidor en donde se almacenarán los datos temporales. Debe contener el mismo valor en los servicios de guardado y recogida de datos. El administrador del sistema puede determinar que sólo estos servicios tengan permiso de lectura y escritura en él. Si no se configura esta propiedad, se usará el directorio temporal del servidor.
- **expTime:** Es el tiempo de caducidad en milisegundos de los ficheros del directorio. Esta propiedad sólo se establece para el servicio de recuperación de datos. Una vez superado ese tiempo desde la creación del fichero, el servicio de recuperación se negará a devolverlo y lo eliminará. El valor por defecto establecido es "60000" (1 minuto)

Estos servicios de guardado y recuperación se distribuyen en forma de WAR y son independientes del software servidor de aplicaciones que se utilice. Consulte la documentación de su software servidor de aplicaciones para saber más acerca de cómo desplegarlos.

El integrador puede sustituir estos Servlets por otros realizados por él mismo siempre y cuando mantenga la interfaz de comunicación. De esta forma, por ejemplo, podría crear Servlets que almacenasen los datos en base de datos o los insertase en un bus temporal de datos.

9.3.1 Notas sobre los servicios de almacenaje y recuperación

La conexión entre los servicios de almacenaje (**SignatureStorageServer**) y los de recuperación (**SignatureRetrieverServer**) se hace mediante sistema de ficheros, compartiendo una simple carpeta temporal, definida en ambos mediante la variable **tmpDir**, variable que debe siempre apuntar a directorios visibles y compartidos por todas las instancias en ejecución.

Este aspecto es especialmente importante en configuraciones de servidores de aplicaciones en alta disponibilidad, donde puede haber varios nodos que presten el servicio trifásico, cada uno de ellos en un sistema de ficheros diferente, donde sí se especifica una ruta local, puede que esta apunte a un directorio distinto en cada nodo (distinto servidor, disco diferente, otro sistema de ficheros, etc.).

El que todos los nodos accedan al mismo directorio referenciado en la configuración se puede lograr fácilmente usando un almacenamiento compartido entre todos ellos (con el mismo punto de montaje), mediante enlaces simbólicos, etc. Es importante también asegurarse de que todos los nodos tienen los permisos adecuados sobre los directorios configurados.

9.3.1.1 Consideraciones de seguridad

Un posible ataque de denegación de servicio sobre este sistema de almacenaje temporal es simplemente hacer muchas peticiones de almacenaje hasta que se alcance la capacidad total del sistema de ficheros.

Los servicios proporcionados no incorporan ninguna medida contra estos ataques, por lo que debe ser el integrador el que las implemente. Algunas de estas medidas podrían ser:

- Establecer cuotas de disco para el directorio configurado en **tmpDir**.
- Detectar (y prevenir) múltiples llamadas al servicio de almacenamiento desde una misma dirección sin estar acompañadas de las respectivas llamadas de recuperación.
- Detectar (y prevenir) múltiples llamadas al servicio de almacenamiento en una frecuencia inusualmente alta.
- Limitar el tamaño de los ficheros que acepta el servicio.
- Etc.

9.4 Notificaciones al usuario

Es **obligatorio que el usuario tenga instalada la aplicación Cliente @firma móvil** compatible con su dispositivo antes de realizar el trámite web para el que se haya desplegado el MiniApplet. Por ello se recomienda a los integradores que, al detectar el tipo de dispositivo del usuario le adviertan y muestren el enlace a la página de descarga de la aplicación, para que la descarguen e instalen antes de continuar.

La biblioteca JavaScript que acompaña al Cliente @firma facilita al usuario la detección de los sistemas operativos móviles compatibles mediante las funciones:

- `function isAndroid()`
 - Detecta si el usuario accede a la página web desde un dispositivo Android.
- `function isIOS()`
 - Detecta si el usuario accede a la página web desde un iPod, iPhone o iPad.
- `function isChrome()`
 - Detecta si el usuario accede a la página web desde un navegador Chrome.

Un ejemplo del uso de estas funciones sería:

```
// Si es un dispositivo Android, mostramos el mensaje de advertencia para Android
if (MiniApplet.isAndroid()) {
    document.getElementById("androidWarning").style.display = "block";
}
```


```
}  
// Si es un dispositivo iOS, mostramos el mensaje de advertencia para iOS  
else if (MiniApplet.isAndroid()) {  
    document.getElementById("iOSWarning").style.display = "block";  
}  
// Si el navegador es Chrome, mostramos el mensaje de advertencia para AutoFirma  
else if (MiniApplet.isChrome ()) {  
    document.getElementById("autoFirmaWarning").style.display = "block";  
}
```


El integrador sería el responsable de preparar esos mensajes de advertencia.


9.5 Enlaces de descarga

El usuario podrá encontrar estas aplicaciones en la tienda de aplicaciones correspondiente a cada sistema. Los enlaces disponibles son:

9.5.1 Enlaces Android

| |
|--|
|  |
| <pre> </pre> |

| |
|---|
|  |
| <pre> </pre> |

| |
|--|
|  |
| <pre> </pre> |

| |
|---|
|  |
| <pre> </pre> |

9.5.2 Enlaces iOS

App Store

```
<a href="https://itunes.apple.com/es/app/cliente-firma-  
movil/id627410001?mt=8&uo=4" target="itunes_store" style="display:inline-  
block;overflow:hidden;background:url(https://linkmaker.itunes.apple.com/ht  
mlResources/assets//images/web/linkmaker/badge_appstore-sm.png) no-  
repeat;width:61px;height:15px;@media only screen{background-  
image:url(https://linkmaker.itunes.apple.com/htmlResources/assets//images/  
web/linkmaker/badge_appstore-sm.svg);}"></a>
```



```
<a href="https://itunes.apple.com/es/app/cliente-firma-  
movil/id627410001?mt=8&uo=4" target="itunes_store" style="display:inline-  
block;overflow:hidden;background:url(https://linkmaker.itunes.apple.com/ht  
mlResources/assets/es_es//images/web/linkmaker/badge_appstore-lrg.png) no-  
repeat;width:135px;height:40px;@media only screen{background-  
image:url(https://linkmaker.itunes.apple.com/htmlResources/assets/es_es//i  
mages/web/linkmaker/badge_appstore-lrg.svg);}"></a>
```

Cliente @firma móvil – Ministerio de Hacienda y Administraciones Públicas

```
<a href="https://itunes.apple.com/es/app/cliente-firma-  
movil/id627410001?mt=8&uo=4" target="itunes_store">Cliente @firma móvil -  
Ministerio de Hacienda y Administraciones Públicas</a>
```

9.5.3 Enlaces AutoFirma

Para instar al usuario que se instale AutoFirma, rediríjalo a la siguiente página web:

<http://firmaelectronica.gob.es/Home/Descargas.html>

9.6 Limitaciones

Debe tenerse en cuenta que las versiones actuales de los distintos cliente móviles no implementan toda la funcionalidad disponible en el MiniApplet, ni es la misma funcionalidad con la que cuenta cada aplicativo móvil. Las limitaciones existentes, ya sea porque aún no se han desarrollado o por la imposibilidad de hacerlo para ese sistema concreto, son las siguientes:

9.6.1 Limitaciones de formato

No todos los formatos de firma del MiniApplet están soportados en su implementación tradicional monofásica en las aplicaciones móviles (AutoFirma sí las soporta). Según el sistema móvil, hay formatos que aún no están implementados. Cuando nuestro despliegue utilice un formato de firma que no esté implementado para el sistema móvil del usuario, se intentará componer la firma de forma trifásica, de tal forma que la construcción del formato de firma se realice en el servidor y sólo la operación de firma digital se realice en el dispositivo del usuario.

Para permitir esta compatibilidad de formatos el integrador debe desplegar el servicio de firma trifásico y configurarlo en la web de despliegue del MiniApplet. Puede consultar cómo realizar estas tareas en el apartado [Firmas/Multifirmas trifásicas](#).

Los formatos monofásicos soportados por cada uno de los sistemas son:

- Android
 - CAdES
 - PAdES
- iOS
 - CAdES
- AutoFirma
 - CAdES
 - XAdES
 - PAdES
 - FacturaE

9.6.2 Limitaciones funcionales

Una limitación común a todas las aplicaciones es que todas utilizan siempre el almacén de certificados del sistema (o el propio de la aplicación en el caso de iOS). Se ignora tanto el almacén configurado en el método de carga del applet como el navegador utilizado para ejecutar las funciones de firma.

9.6.2.1 *Sistemas Android*

- `function getFileNameContentBase64(title, extension, description)`
 - Los clientes nativos no disponen del método de carga de ficheros, ya que esto conllevaría invocar a la aplicación sólo para que nos permitiese seleccionar el fichero de datos.
 - En su lugar, cuando se desee permitir al usuario firmar un fichero localizado en su dispositivo, se deberá ejecutar el método de firma del MiniApplet sin indicar los datos a firmar. De esta forma la aplicación nativa permitirá al usuario seleccionar un fichero y lo firmará directamente.
- `function getMultiFileNameContentBase64(title, extension, description)`
 - Los clientes móviles no disponen de un modo para cargar múltiples ficheros para procesarlos secuencialmente.
- `function setStickySignatory(stick)`
 - Los clientes móviles no permiten fijar un certificado que el usuario seleccionar durante una operación de firma/multifirma de tal forma que este se reutilice en siguientes operaciones.

9.6.2.2 *Sistemas iOS*

- `function saveDataToFile()`
 - El cliente móvil iOS no soporta la operación de guardado en disco, ya que el sistema operativo impide el guardado de datos en un directorio común y accesible para el usuario, por lo que este no podría recuperar la firma guardada.
- `function getFileNameContentBase64(title, extension, description)`
 - Los clientes nativos no disponen del método de carga de ficheros, ya que esto conllevaría invocar a la aplicación sólo para que nos permitiese seleccionar el fichero de datos.
 - En el caso concreto de iOS, la propia aplicación no tiene acceso a ficheros, por lo que sólo se podrán firmar datos proporcionados desde la propia web.
- `function getMultiFileNameContentBase64(title, extension, description)`
 - Los clientes móviles no disponen de un modo para cargar múltiples ficheros para procesarlos secuencialmente.
- `function setStickySignatory(stick)`
 - Los clientes móviles no permiten fijar un certificado que el usuario seleccionar durante una operación de firma / multifirma de tal forma que este se reutilice en siguientes operaciones.

9.6.2.3 AutoFirma

- `function getFileNameContentBase64(title, extension, description)`
 - Los clientes nativos no disponen del método de carga de ficheros, ya que esto conllevaría invocar a la aplicación sólo para que nos permitiese seleccionar el fichero de datos.
 - En su lugar, cuando se desee permitir al usuario firmar un fichero localizado en su dispositivo, se deberá ejecutar el método de firma del MiniApplet sin indicar los datos a firmar. De esta forma la aplicación permitirá al usuario seleccionar un fichero y lo firmará directamente.
- `function getMultiFileNameContentBase64(title, extension, description)`
 - AutoFirma no soporta la carga de múltiples ficheros simultáneamente.
- `function setStickySignatory(stick)`
 - AutoFirma no permite fijar un certificado que el usuario seleccionar durante una operación de firma/multifirma de tal forma que este se reutilice en siguientes operaciones.

9.7 Compatibilidad del MiniApplet @firma v1.3 con aplicaciones móviles y AutoFirma

Las versiones actualmente publicadas de los clientes móviles y AutoFirma en las siguientes tiendas de software son compatibles con la última versión 1.2 del MiniApplet, pero no con esta versión 1.3:

- Google Play (Cliente móvil @firma 1.2)
- Play Store (Cliente @firma móvil 1.1)
- Repositorio Guadalinux (FirmaFácil / AutoFirma 1.3)

Sí es compatible AutoFirma 1.4 y superiores.

Una vez se publique la versión definitiva del MiniApplet @firma v1.3, se publicarán las versiones compatibles en las distintas tiendas y repositorios de software.

10 Información específica para firmas CAdES

10.1 Algoritmos de firma

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

- SHA1withRSA
- 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.

10.2 Firmas CAdES implícitas o explícitas

Las firmas y cofirmas CAdES pueden incluir internamente una copia de los datos firmados (firmas implícitas) o no incluirlos (firmas explícitas o “detached”). El MiniApplet Cliente @firma por defecto genera firmas explícitas, más pequeñas en tamaño.

Si desea generar firmas implícitas, debe indicar el siguiente parámetro adicional:

`mode=explicit`

Consulte la sección “Parámetros adicionales” para mayor información.

10.3 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.

10.3.1 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 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 se 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-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 CAdES-EPES. |
| includeOnlySigningCertificate | true | Indica que debe incluirse en la firma únicamente el certificado del firmante. |
| | false | Indica que debe incluirse en la firma toda la cadena de certificación del certificado firmante. Valor por defecto. |
| commitmentTypeIndications | [Entero] | Indica el número de CommitmentTypeIndications que se van a declarar. Estos son los motivos que se declaran para la firma. Los valores concretos se especifican con commitmentTypeIndication <i>n</i> Identifier y commitmentTypeIndication <i>n</i> Description, donde 'n' va desde 0 hasta el valor indicado en esta propiedad menos 1. |
| commitmentTypeIndication <i>n</i> Identifier | 1 | Establece que el CommitmentTypeIndications número <i>n</i> (contando desde cero) es "Prueba de |

| | | |
|--|---------|---|
| | | origen". |
| | 2 | Establece que el CommitmentTypeIndications número n (contando desde cero) es " Prueba de recepción ". |
| | 3 | Establece que el CommitmentTypeIndications número n (contando desde cero) es " Prueba de entrega ". |
| | 4 | Establece que el CommitmentTypeIndications número n (contando desde cero) es " Prueba de envío ". |
| | 5 | Establece que el CommitmentTypeIndications número n (contando desde cero) es " Prueba de aprobación ". |
| | 6 | Establece que el CommitmentTypeIndications número n (contando desde cero) es " Prueba de creación ". |
| commitmentTypeIndication n Description | [Texto] | Establece la descripción del CommitmentTypeIndications número n . (contando desde cero) Este atributo es opcional. |
| commitmentTypeIndication n DocumentationReferences | [Texto] | Lista de URL separadas por el carácter ' ' que se aportan como referencias documentales del CommitmentTypeIndication número n (atributo opcional). Las URL de la lista no pueden contener el carácter ' ' (ya que este se usa como separador). |
| commitmentTypeIndication n CommitmentTypeQualifiers | [Texto] | Lista de indicadores textuales separados por el carácter ' ' que se aportan como calificadores adicionales del CommitmentTypeIndication número n (atributo opcional). Normalmente son OID. Los elementos de la lista no pueden contener el carácter ' ' (ya que este se usa como separador). |

10.3.2 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 <code>policyIdentifier</code> , al igual que es obligatorio también dar valor al parámetro <code>policyIdentifierHashAlgorithm</code> . |
| <code>policyIdentifierHashAlgorithm</code> | SHA1 | Indica que la huella digital indicada en el parámetro <code>policyIdentifierHash</code> se calculó mediante el algoritmo SHA1. |
| | SHA-256 | Indica que la huella digital indicada en el parámetro <code>policyIdentifierHash</code> se calculó mediante el algoritmo SHA-256. |
| | SHA-384 | Indica que la huella digital indicada en el parámetro <code>policyIdentifierHash</code> se calculó mediante el algoritmo SHA-384. |
| | SHA-512 | Indica que la huella digital indicada en el parámetro <code>policyIdentifierHash</code> se calculó mediante el algoritmo SHA-512. |
| <code>policyQualifier</code> | [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. |
| <code>includeOnlySigningCertificate</code> | true | Indica que debe incluirse en la firma únicamente el certificado del firmante. |
| | false | Indica que debe incluirse en la firma toda la cadena de certificación del certificado firmante. Valor por defecto. |
| <code>commitmentTypeIndications</code> | [Enter o] | Indica el número de <code>CommitmentTypeIndications</code> que se van a declarar. Estos son los motivos que se declaran para la firma. Los valores concretos se especifican con <code>commitmentTypeIndication<i>n</i>Identifier</code> y <code>commitmentTypeIndication<i>n</i>Description</code> , donde ' <i>n</i> ' va desde 0 hasta el valor menos 1 indicado en esta propiedad. |
| <code>commitmentTypeIndication<i>n</i>Identifier</code> | 1 | Establece que el <code>CommitmentTypeIndication</code> número <i>n</i> es "Prueba de origen". |
| | 2 | Establece que el <code>CommitmentTypeIndication</code> número <i>n</i> es "Prueba de recepción". |
| | 3 | Establece que el <code>CommitmentTypeIndication</code> número <i>n</i> es "Prueba de entrega". |
| | 4 | Establece que el <code>CommitmentTypeIndication</code> número <i>n</i> es "Prueba de envío". |
| | 5 | Establece que el <code>CommitmentTypeIndication</code> número <i>n</i> es "Prueba de aprobación". |
| | 6 | Establece que el <code>CommitmentTypeIndication</code> número <i>n</i> es "Prueba de creación". |
| <code>commitmentTypeIndication<i>n</i>CommitmentTypeQualifiers</code> | [OID] | Lista de OID separados por el caracter ' ' que se aportan como calificadores adicionales del <code>CommitmentTypeIndication</code> número <i>n</i> (atributo opcional). |



DIRECCIÓN DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS
COMUNICACIONES

MiniApplet @firma

11 Información específica para firmas XAdES

11.1 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

11.2 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 vulnerable.

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.

11.3 Algoritmos de huella digital para las referencias

XAdES hace cálculos de huella digital para cada una de las referencias firmadas. El algoritmo por defecto para estas huellas es SHA-512. Este puede cambiarse mediante el parámetro adicional `referencesDigestMethod`. Consulte la sección “11.6” para más información.

11.4 Situación del nodo de firma en XAdES Enveloped

Por defecto, el MiniApplet Cliente @firma sitúa la firma electrónica en su nodo “Signature” directamente como hijo de la raíz del XML. No obstante, hay situaciones en las que puede interesar situar este nodo de firma en una situación arbitraria del XML.

Para ello, puede usarse el parámetro adicional “*insertEnvelopedSignatureOnNodeByXPath*”, en el que, mediante una expresión XPath v1, podemos indicar el nodo en el que queremos se inserte la firma (el nodo “Signature” pasará a ser el primer hijo de este). Si la expresión XPath resolviese varios nodos, se usará el primero de ellos.

Por ejemplo, en el siguiente XML:

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

<book category="COOKING">

  <title lang="en">Everyday Italian</title>

  <author>Giada De Laurentiis</author>

  <year>2005</year>

  <price>30.00</price>

</book>

<book category="CHILDREN">

  <title lang="en">Harry Potter</title>

  <author>J K. Rowling</author>

  <year>2005</year>

  <price>29.99</price>

</book>

</bookstore>
```

Si indicamos el parámetro con este valor:

```
insertEnvelopedSignatureOnNodeByXPath = /bookstore/book[1]/title
```

La firma se insertará como nodo hijo del título del primer libro:

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

<book category="COOKING">

  <title lang="en">

    Everyday Italian

    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="S1">

      ...

    </ds:Signature>

  </title>

  <author>Giada De Laurentiis</author>
```



```
<year>2005</year>

<price>30.00</price>

</book>

<book category="CHILDREN">

  <title lang="en">Harry Potter</title>

  <author>J K. Rowling</author>

  <year>2005</year>

  <price>29.99</price>

</book>

</bookstore>
```

Si en la expresión XPath desea referenciar nodos dentro de un espacio de nombres, debe usar funciones XPath como `namespace-uri()` o `local-name()`. Por ejemplo, para seleccionar el primer nodo dentro del espacio de nombres de factura electrónica podríamos usar la expresión:

```
//*[namespace-uri()='http://www.facturae.es/Facturae/2007/v3.0/Facturae']
```

11.5 Uso de estructuras *Manifest* en firmas XAdES

Es posible crear firmas XAdES en las que, siguiendo el punto 2.3 de la especificación XMLDSig (<http://www.w3.org/TR/2000/WD-xmlsig-core-20000510/#sec-o-Manifest>), las referencias XML no se firmen directamente, sino que se firme una estructura de tipo *Manifest* que a su vez contenga las referencias a firmar.

De esta forma, tal y como indica la normativa, la resolución de las referencias incluidas dentro de una estructura *Manifest* es una responsabilidad del validador, y de cara a la propia firma no se resuelven para calcular las huellas digitales (lo que permite hacer firmas XML explícitas). Consulte la especificación XMLDSig para mayor información.

Para crear firmas XAdES con estructuras *Manifest* debe especificarse el siguiente parámetro adicional:

```
useManifest=true
```

Un ejemplo muy simplificado de la estructura de una firma con *Manifest* sería:

```
<ds:Signature Id="Signature-02553">
  <ds:SignedInfo>

    <ds:Reference Id="Reference-894bfd39 "
      Type="http://www.w3.org/2000/09/xmlsig#Manifest" URI="#Manifest-
      36e2de7b">
```

...

```
</ds:Reference>

</ds:SignedInfo>

...

<ds:Object Id="ManifestObject-ffd54e53">

  <ds:Manifest Id="Manifest-36e2de7b">

    <ds:Reference Id="Reference-894bfd39" URI="myscheme://path/file">

      <ds:DigestMethod
        Algorithm="http://www.w3.org/2001/04/xmldsig#sha512"/>

      <ds:DigestValue>...</ds:DigestValue>

    </ds:Reference>

  </ds:Manifest>

</ds:Object>

...

</ds:Signature>
```

En este ejemplo el contenido firmado es “**myscheme://path/file**”, pero al firmar no se ha intentado acceder a ese fichero, y se ha dado por buena la huella digital indicada.

11.6 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.

11.6.1 Firma y cofirma

| Nombre del parámetro | Valores posibles | Descripción |
|---------------------------------------|------------------------------------|--|
| insertEnvelopedSignatureOnNodeByXPath | [Texto (expresión XPath v1)] | Indica, mediante una expresión XPath (v1), el nodo bajo el cual debe insertarse el nodo de firma en el caso de una firma <i>Enveloped</i> . Si la expresión devuelve más de un nodo, se usa solo el primero. Si la expresión no devuelve nodos o está mal construida se lanzará una excepción. |

| | | |
|--------------------------------------|-------------------|--|
| | | Este parámetro solo tiene efecto en firmas <i>Enveloped</i> . |
| useManifest | true | Usa un Manifest de XMLDSig con las referencias de firma en vez de firmar directamente estas referencias. Esto permite que sea opcional la comprobación del destino y huellas digitales de las referencias. |
| | false | Genera las firmas normalmente, sin Manifest (comportamiento por defecto) |
| addKeyInfoKeyValue | true | Incluye el nodo KeyValue dentro de KeyInfo de XAdES (comportamiento por defecto). |
| | false | No incluye el nodo KeyValue dentro de KeyInfo de XAdES. |
| addKeyInfoKeyName | true | Incluye el nodo KeyName dentro de KeyInfo de XAdES. |
| | false | No incluye el nodo KeyName dentro de KeyInfo de XAdES (comportamiento por defecto). |
| avoidXpathExtraTransformsOnEnveloped | true | Evita la inclusión de la transformación XPATH2 que normalmente se añade para posibilitar las cofirmas y que elimina todas las firmas del documento para dejar únicamente el contenido. |
| | false | Incluye la transformación XPATH2 posibilita las cofirmas eliminando todas las firmas del documento para dejar únicamente el contenido (comportamiento por defecto). |
| format | XAdES Enveloping | Genera firmas en formato <i>Enveloping</i> . Este es el formato que se utiliza por defecto cuando no se indica ninguno. |
| | XAdES Enveloped | Genera firmas en formato <i>Enveloped</i> . |
| | XAdES Detached | Genera firmas en formato <i>Internally Detached</i> . |
| includeOnlySigningCertificate | true | Indica que debe incluirse en la firma únicamente el certificado del firmante. |
| | false | Indica que debe incluirse en la firma toda la cadena de certificación del certificado firmante. Valor por defecto. |
| 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 <code>policyIdentifier</code> no es universalmente accesible. Si se da valor a este parámetro es obligatorio también dar valor al parámetro <code>policyIdentifierHashAlgorithm</code> . |
| policyIdentifierHashAlgorithm | SHA1 | Indica que la huella digital indicada en el parámetro <code>policyIdentifierHash</code> se calculó mediante el algoritmo SHA1. |
| | SHA-256 | Indica que la huella digital indicada en el parámetro <code>policyIdentifierHash</code> se calculó mediante el algoritmo SHA-256. |
| | SHA-384 | Indica que la huella digital indicada en el parámetro <code>policyIdentifierHash</code> se calculó mediante el algoritmo SHA-384. |
| | SHA-512 | Indica que la huella digital indicada en el |

| | | |
|--|--|--|
| | | parámetro <code>policyIdentifierHash</code> se calculó mediante el algoritmo SHA-512. |
| <code>policyQualifier</code> | [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. |
| <code>policyDescription</code> | [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. |
| <code>signerClaimedRoles</code> | [Texto] | Agrega a la firma campos con los cargos atribuidos al firmante. Deben separarse los cargos con el carácter “ ” (y este no puede estar en el propio texto de ningún cargo). En el caso de que se firme un XML, la codificación del texto usado debe adecuarse al XML firmado. |
| <code>signatureProductionCity</code> | [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. |
| <code>signatureProductionProvince</code> | [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. |
| <code>signatureProductionPostalCode</code> | [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. |
| <code>signatureProductionCountry</code> | [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. |
| <code>referencesDigestMethod</code> | <code>http://www.w3.org/2000/09/xmldsig#sha1</code> | Usa el algoritmo SHA1 para el cálculo de las huellas digitales de las referencias XML firmadas. |
| | <code>http://www.w3.org/2001/04/xmlenc#sha256</code> | Usa el algoritmo SHA-256 para el cálculo de las huellas digitales de las referencias XML firmadas. Este es el valor recomendado. |
| | <code>http://www.w3.org/2001/04/xmlenc#sha512</code> | Usa el algoritmo SHA-512 para el cálculo de las huellas digitales de las referencias XML firmadas. |
| <code>mimeType</code> | [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). |
| <code>encoding</code> | [Texto] | Codificación de los datos a firmar. Un uso incorrecto de este parámetro puede provocar la generación de una firma inválida. |
| <code>contentTypeOid</code> | [OID o URN de tipo] | Identificador del tipo de dato firmado. Este |

| | | |
|---------------------------|---|--|
| | OID] | parámetro es complementario (que no excluyente) al parámetro <code>mimeType</code> . |
| 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). Si se establece este parámetro es posible que se necesite establecer también el parámetro <code>signedPropertiesTypeUrl</code> para evitar incoherencias en la versión de XAdES. |
| signedPropertiesTypeUrl | [URL] | URL de definición del tipo de las propiedades firmadas (<i>Signed Properties</i>) de XAdES. Si se establece este parámetro es posible que se necesite establecer también el parámetro <code>xadesNamespace</code> para evitar incoherencias en la versión de XAdES. Si no se establece se usa el valor por defecto: http://uri.etsi.org/01903#SignedProperties . |
| 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 <i>dereferenciación</i> de hojas de estilo enlazadas con rutas relativas). |
| | false | Permite que se muestren diálogos gráficos adicionales al usuario. |
| xmlTransforms | [Número] | Número de transformaciones a aplicar al contenido firmado. Debe indicarse posteriormente igual número de parámetros <code>xmlTransformnType</code> , sustituyendo <i>n</i> por un ordinal consecutivo, comenzando en 0 (ver notas |

| | | |
|--|--|--|
| | | adicionales sobre indicación de transformaciones adicionales). |
| xmlTransform n Type | 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/2002/06/xmldsig-filter2 | El contenido se transforma mediante XPATH2 antes de ser firmado. Únicamente es aplicable cuando se firma contenido XML. |
| xmlTransform n Subtype | [Texto] | Subtipo de la transformación n . Los valores aceptados y sus funcionalidades dependen del valor indicado en xmlTransform n Type. |
| xmlTransform n Body | [Texto] | Cuerpo de la transformación n . Los valores aceptados y sus funcionalidades dependen del valores indicados en xmlTransform n Type y en xmlTransform n Subtype. |
| nodeToSign | [Texto] | Identificador del nodo (establecido mediante el atributo "Id") que se desea firmar dentro de un XML. |
| commitmentTypeIndications | [Entero] | Indica el número de CommitmentTypeIndications que se van a declarar. Estos son los motivos que se declaran para la firma. Los valores concretos se especifican con commitmentTypeIndication n Identifier y commitmentTypeIndication n Description, donde ' n ' va desde 0 hasta el valor menos 1 indicado en esta propiedad. IMPORTANTE: Por un error conocido de la versión actual del programa, solo es posible indicar un elemento, por lo que de establecerse un valor a este parámetro debe ser obligatoriamente "1". |
| commitmentTypeIndication n Identifier | 1 | Establece que el CommitmentTypeIndications número n es "Prueba de origen". |
| | 2 | Establece que el CommitmentTypeIndications número n es "Prueba de recepción". |
| | 3 | Establece que el CommitmentTypeIndications número n es "Prueba de entrega". |
| | 4 | Establece que el CommitmentTypeIndications número n es "Prueba de envío". |
| | 5 | Establece que el CommitmentTypeIndications número n es "Prueba de aprobación". |
| | 6 | Establece que el CommitmentTypeIndications número n es "Prueba de creación". |
| commitmentTypeIndication n Description | [Texto] | Establece la descripción del CommitmentTypeIndications número n . Este |

| | | |
|--|--|-----------------------|
| | | atributo es opcional. |
|--|--|-----------------------|

11.6.1.1 Indicación de transformaciones adicionales al contenido a firmar

Respecto al uso de los parámetros `xmlTransform n Type`, `xmlTransform n Subtype` y `xmlTransform n Body`, 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 BASE64. La transformación es inversa, es decir, los datos se descodifican desde Base64 antes de firmarse, por lo que estos deben estar previamente codificados en Base64 e indicarse mediante el parámetro adicional `"encoding=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.

El listado de transformaciones se inicia con aquella declarada con el índice 0. Por ejemplo, si se desean insertar 2 transformaciones adicionales, se deberán establecer los parámetros:

```
xmlTransforms=2
xmlTransform0Type=...
xmlTransform0Subtype=... (Opcional)
xmlTransform0Body=...
xmlTransform1Type=...
xmlTransform1Subtype=... (Opcional)
xmlTransform1Body=...
```

11.6.2 Contrafirma

| Nombre del parámetro | Valores posibles | Descripción |
|-------------------------------|-----------------------|--|
| addKeyInfoKeyValue | true | Incluye el nodo KeyValue dentro de KeyInfo de XAdES (comportamiento por defecto). |
| | false | No incluye el nodo KeyValue dentro de KeyInfo de XAdES. |
| addKeyInfoKeyName | true | Incluye el nodo KeyName dentro de KeyInfo de XAdES. |
| | false | No incluye el nodo KeyName dentro de KeyInfo de XAdES (comportamiento por defecto). |
| 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. |
| signerClaimedRoles | [Texto] | Agrega a la firma campos con los cargos atribuidos al firmante. Deben separarse los cargos con el carácter " " (y este no puede estar en el propio texto de ningún cargo). 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. |
| commitmentTypeIndications | [Entero] | Indica el número de CommitmentTypeIndications que se van a declarar. Estos son los motivos que se declaran para la firma. Los valores concretos se especifican con commitmentTypeIndicationnIdentifier y commitmentTypeIndicationnDescription, donde 'n' va desde 0 hasta el valor menos 1 indicado en esta propiedad. IMPORTANTE: Por un error conocido de la versión actual del programa, solo es posible indicar un elemento, por lo que de establecerse un valor a este parámetro debe ser obligatoriamente "1". |
| commitmentTypeIndicationnIdentifier | 1 | Establece que el CommitmentTypeIndications número n es "Prueba de origen". |
| | 2 | Establece que el CommitmentTypeIndications número n es "Prueba de recepción". |
| | 3 | Establece que el CommitmentTypeIndications número n es "Prueba de entrega". |
| | 4 | Establece que el CommitmentTypeIndications número n es "Prueba de envío". |
| | 5 | Establece que el CommitmentTypeIndications número n es "Prueba de aprobación". |
| | 6 | Establece que el CommitmentTypeIndications número n es "Prueba de creación". |
| commitmentTypeIndicationnDescription | [Texto] | Establece la descripción del CommitmentTypeIndications número n . Este atributo es opcional. |
| mode | implicit / explicit | Permite que se realicen firmas explícitas en lugar de implícitas (por defecto). Consulte el apartado Firmas XAdES "explícitas" . |

11.7 Firmas XAdES "explícitas"

El MiniApplet Cliente @firma considera, por compatibilidad con el Applet @firma, un tipo de firmas XAdES llamadas "XAdES explícitas", en las que no se firman los datos, sino la huella digital de estos.

Este tipo de firmas no son conformes con ningún tipo de normativa ni estándar, y deben ser sustituidas por firmas con estructuras MANIFEST (sección 11.5) ya que en un futuro próximo se extinguirá el uso del tipo "explícito" en las firmas XAdES.

Para configurar el modo explícito con las firmas XAdES se utilizará el parámetro de configuración “mode” con el valor “explicit”:

```
mode=explicit
```

12 Información específica para firma de facturas electrónicas

12.1 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 MiniApplet 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.

12.2 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.

12.3 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 |
|--|------------------|--|
| <code>signatureProductionCity</code> | [Texto] | Agrega a la firma un campo con la ciudad en la que se realiza la firma. |
| <code>signatureProductionProvince</code> | [Texto] | Agrega a la firma un campo con la provincia en la que se realiza la firma. |
| <code>signatureProductionPostalCode</code> | [Texto] | Agrega a la firma un campo con el código postal en donde se realiza la |



DIRECCIÓN DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS
COMUNICACIONES

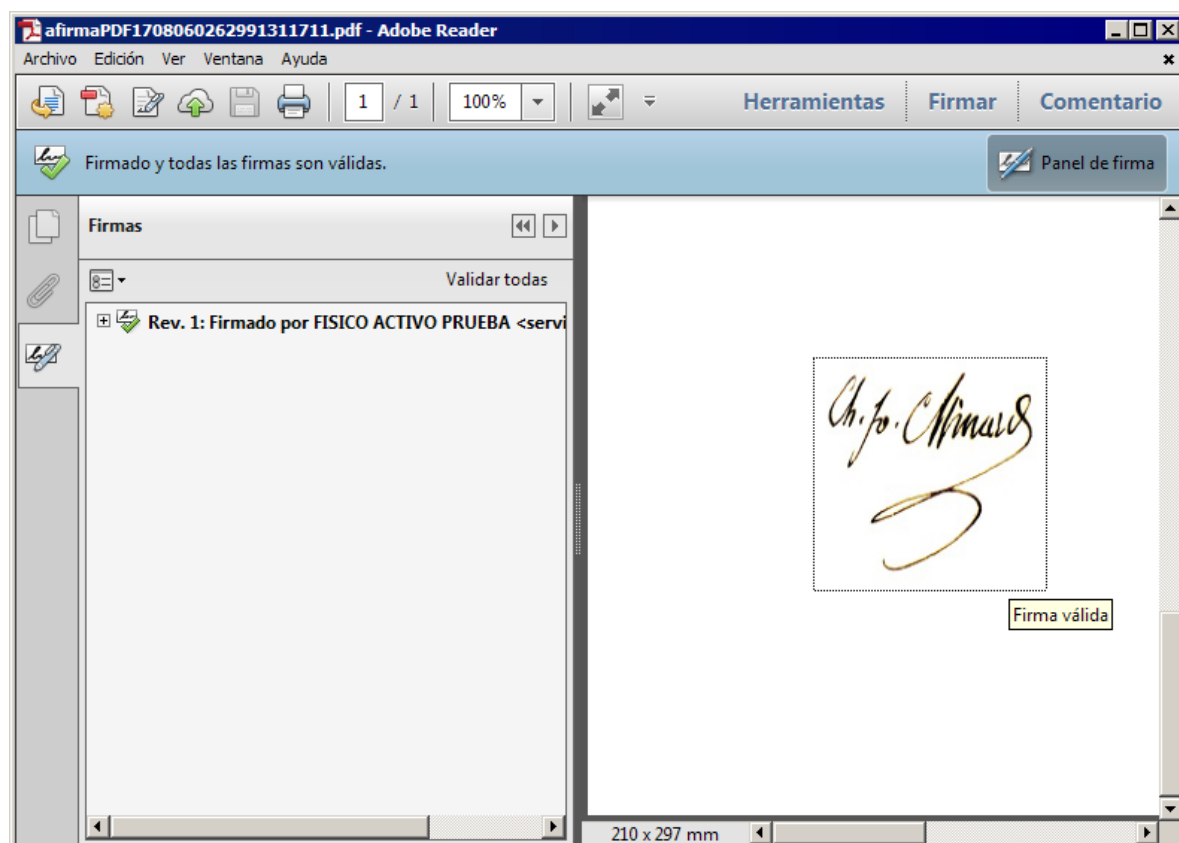
MiniApplet @firma

| | | |
|----------------------------|---------|---|
| | | firma. |
| signatureProductionCountry | [Texto] | Agrega a la firma un campo con el país en el que se realiza la firma. |

13 Información específica para firmas PAdES

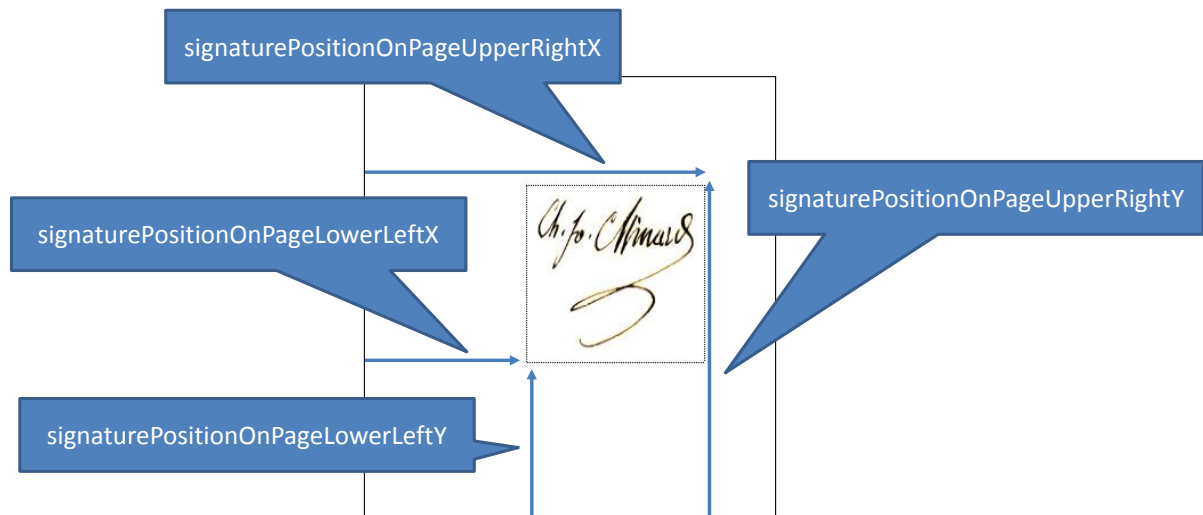
13.1 Creación de una firma visible

El MiniApplet Cliente @firma permite la creación de firmas visibles dentro de un documento PDF, que son lo son tanto en pantalla (por ejemplo, usando Adobe Reader) como en papel una vez impreso el documento.



Para ello debemos indicar, mediante parámetros adicionales, la página en donde situar la visualización de la firma (solo puede haber una, en una única página) y sus coordenadas dentro de esta.

Las coordenadas de la visualización se indican partiendo de la esquina inferior izquierda, según el siguiente diagrama:

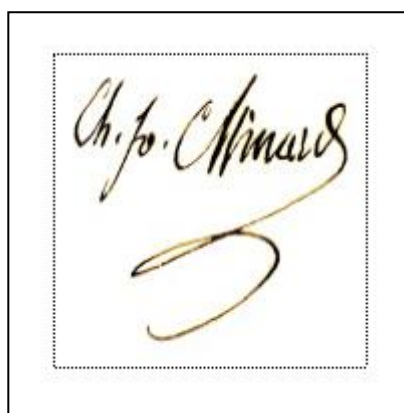


Estas coordenadas, así como la página de inserción se establecen usando los parámetros adicionales, por ejemplo:

```
signaturePositionOnPageLowerLeftX = 100  
signaturePositionOnPageLowerLeftY = 100  
signaturePositionOnPageUpperRightX = 200  
signaturePositionOnPageUpperRightY = 200  
signaturePositionOnPageUpperRightX = 200  
signaturePage = 1
```

Los documentos PDF comienzan su numeración de páginas desde uno (1). Si se indica -666 como página se usa la última página del documento.

Dentro del recuadro marcado por las coordenadas indicadas, es posible mostrar distintos elementos:



- Una imagen:
 - En este caso debe indicarse qué imagen a usar aportando el binario en formato JPEG codificado en Base64.

- `signatureRubricImage = AGFGSFH...`
- La imagen de deforma para adaptarse a las dimensiones del recuadro marcado por las coordenadas, por lo que es importante que ambos tengan la misma relación de aspecto.



- Texto (que puede combinarse con una imagen)
 - Es necesario indicar no solo el texto a sobreimprimir en el cuadro visible, sino también indicaciones sobre su formato (tipo de letra y su tamaño, color, etc.).
 - El texto introduce de forma automática los retornos de carro necesarios para adaptarse al recuadro.
 - El texto aparece siempre sobre la imagen indicada, si se indicó alguna.

Los parámetros para indicar el formato son:

layer2Text

Texto a escribir dentro de la firma visible.

Este texto se escribe únicamente si no se ha especificado una imagen de rúbrica, y necesita que se indique la página y la situación dónde mostrar el recuadro de firma mediante los parámetros `signaturePositionOnPageLowerLeftX`, `signaturePositionOnPageLowerLeftY`, `signaturePositionOnPageUpperRightX`, `signaturePositionOnPageUpperRightY` y `signaturePage`.

layer2FontFamily

Tipo de letra a usar en el texto de la firma visible. Este parámetro requiere que se haya establecido también el parámetro `layer2Text`. Los valores admitidos son:

- 0 = Courier (tipo por defecto)
- 1 = Helvética
- 2 = Times Roman
- 3 = Symbol
- 4 = ZapfDingBats

layer2FontSize

Tamaño de letra a usar en el texto de la firma visible. Este parámetro requiere que se haya establecido también el parámetro `layer2Text`.

Los valores admitidos son numéricos (y el valor por defecto es 12).

layer2FontStyle

Estilo del tipo de letra a usar en el texto de la firma visible. Este parámetro requiere que se haya establecido también el parámetro `layer2Text`.

Los valores admitidos son numéricos, correspondiendo:

- 0 = Normal (estilo por defecto)
- 1 = Negrita
- 2 = Cursiva
- 3 = Negrita y cursiva
- 4 = Subrayado
- 8 = Tachado

Es posible combinar estilos aplicando la operación lógica *o* sobre los valores numéricos a combinar.

layer2FontColor

Color del texto de la firma visible. Este parámetro requiere que se haya establecido también el parámetro `layer2Text`.

Los valores admitidos son textuales (se ignora entre mayúsculas y minúsculas), soportándose:

- *black* = Negro (color por defecto)
- *white* = Blanco
- *gray* = Gris
- *lightGray* = Gris claro
- *darkGray* = Gris oscuro
- *red* = Rojo
- *pink* = Rosa

13.2 Inserción de una imagen en un documento PDF antes de ser firmado

Como ayuda principalmente a la inserción de Códigos Seguros de Validación (CSV), existe la capacidad de insertar una imagen en un documento PDF justo antes de que se produzca la firma.

Para ello, debemos indicar primero una página y una zona dentro de esta para insertar la imagen, usando para ello el mismo sistema de coordenadas descrito en la sección “13.1”, es decir, a partir de la esquina inferior izquierda. La imagen debe proporcionarse en formato JPEG codificado en Base64.

Para indicar la página, podemos usar su número (empezando a contar desde uno como primera página), usar -1 para referirnos a la última página del documento o 0 (cero) para insertar la imagen en todas las páginas.

Es importante recalcar que la imagen se deforma para adaptarse al recuadro marcado por las coordenadas, siendo útil para evitar este efecto que ambos tengan la misma relación de aspecto.

Igualmente, no se proporcionan funcionalidades de rotado, por lo que si se quiere insertar una imagen de lado (por ejemplo, en el margen de la página, esta debe venir rotada en origen.

Los parámetros adicionales a usar para la inserción de imágenes son:

image

Imagen que se desea insertar en el PDF antes de que este sea firmado. La imagen debe proporcionarse en formato JPEG codificado en Base64.

Si el documento ya contiene firmas es posible que se invaliden, por lo que conviene usarlo únicamente en documentos sin firmas previas.

imagePage

Página donde desea insertarse la imagen indicada mediante el parámetro `image`. La numeración de las páginas comienza en uno.

Si se indica -1 como número de página se inserta la imagen en la última página del documento. Si se indica 0 como número de página se inserta la imagen en todas las páginas del documento. Este parámetro es obligatorio, si no se indica una página válida no se insertará la imagen.

imagePositionOnPageLowerLeftX

Coordenada horizontal inferior izquierda de la posición de la imagen (indicada mediante el parámetro `image`) dentro de la página.

Es necesario indicar el resto de coordenadas de la imagen mediante los parámetros `imagePositionOnPageLowerLeftY`, `imagePositionOnPageUpperRightX` e `imagePositionOnPageUpperRightY`.

Es necesario indicar también una página de inserción en el parámetro `imagePage`.

imagePositionOnPageLowerLeftY

Coordenada vertical inferior izquierda de la posición de la imagen (indicada mediante el parámetro `image`) dentro de la página.

Es necesario indicar el resto de coordenadas de la imagen mediante los parámetros `imagePositionOnPageLowerLeftX`, `imagePositionOnPageUpperRightX` e `imagePositionOnPageUpperRightY`.

Es necesario indicar también una página de inserción en el parámetro `imagePage`.

imagePositionOnPageUpperRightX

Coordenada horizontal superior derecha de la posición de la imagen (indicada mediante el parámetro `image`) dentro de la página.

Es necesario indicar el resto de coordenadas de la imagen mediante los parámetros `imagePositionOnPageLowerLeftX`, `imagePositionOnPageLowerLeftY` e `imagePositionOnPageUpperRightY`.

Es necesario indicar también una página de inserción en el parámetro `imagePage`.

imagePositionOnPageUpperRightY

Coordenada vertical superior derecha de la posición de la imagen (indicada mediante el parámetro `image`) dentro de la página.

Es necesario indicar el resto de coordenadas de la imagen mediante los parámetros `imagePositionOnPageLowerLeftX`, `imagePositionOnPageLowerLeftY` e `imagePositionOnPageUpperRightX`.

Es necesario indicar también una página de inserción en el parámetro `imagePage`.

13.3 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.
- Adobe Acrobat/Reader no soporta múltiples firmas cuando hay firmas PAdES-BES.
- Únicamente es posible usar PAdES para documentos PDF.
- No se firman los posibles adjuntos o empotrados que pudiese contener el documento PDF.

13.4 Firma de documentos PDF cifrados o protegidos con contraseña

Si bien es posible firmar documentos PDF cifrados o protegidos con contraseña, deben tenerse en cuenta las siguientes limitaciones:

- No se soporta la firma de PDF cifrados con certificados o con algoritmo AES256.

- Puede que no sea posible, en todos los casos, validar u obtener justificantes de validación de documentos PDF cifrados o protegidos por contraseña usando la plataforma de validación VALIDE del Gobierno de España.
 - <https://valide.redsara.es/valide/>

13.5 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.

13.6 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.

includeOnlySigningCertificate (propiedad compartida con XAdES y CAdES)

Si se establece a `true` se incluye en la firma únicamente el certificado del firmante (y no la cadena de certificación completa). Si no se establece o se establece a `false` se incluirá toda la cadena de certificación.

alwaysCreateRevision

Si se establece a `true` siempre crea una revisión del PDF incluso cuando el documento no contiene ninguna firma previa.

Esto requiere que los documentos de entrada cumplan estrictamente la especificación PDF 1.7 (ISO 32000-1:2008), y puede crear incompatibilidades con documentos PDF acordes a la especificación 1.3 creados con bibliotecas antiguas, como por ejemplo [QPDE](#).

Si se establece a `false`, no crea revisiones en documentos que no contengan firmas previas y sí las crea en documentos que ya contengan alguna firma.

image

Imagen que se desea insertar en el PDF antes de que este sea firmado. La imagen debe proporcionarse en formato JPEG codificado en Base64.

Si el documento ya contiene firmas es posible que se invaliden, por lo que conviene usarlo únicamente en documentos sin firmas previas.

imagePage

Página donde desea insertarse la imagen indicada mediante el parámetro `image`. La numeración de las páginas comienza en uno.

Si se indica -1 como número de página se inserta la imagen en la última página del documento. Si se indica 0 como número de página se inserta la imagen en todas las páginas del documento. Este parámetro es obligatorio, si no se indica una página válida no se insertará la imagen.

imagePositionOnPageLowerLeftX

Coordenada horizontal inferior izquierda de la posición de la imagen (indicada mediante el parámetro `image`) dentro de la página.

Es necesario indicar el resto de coordenadas de la imagen mediante los parámetros `imagePositionOnPageLowerLeftY`, `imagePositionOnPageUpperRightX` e `imagePositionOnPageUpperRightY`.

Es necesario indicar también una página de inserción en el parámetro `imagePage`.

imagePositionOnPageLowerLeftY

Coordenada vertical inferior izquierda de la posición de la imagen (indicada mediante el parámetro `image`) dentro de la página.

Es necesario indicar el resto de coordenadas de la imagen mediante los parámetros `imagePositionOnPageLowerLeftX`, `imagePositionOnPageUpperRightX` e `imagePositionOnPageUpperRightY`.

Es necesario indicar también una página de inserción en el parámetro `imagePage`.

imagePositionOnPageUpperRightX

Coordenada horizontal superior derecha de la posición de la imagen (indicada mediante el parámetro `image`) dentro de la página.

Es necesario indicar el resto de coordenadas de la imagen mediante los parámetros `imagePositionOnPageLowerLeftX`, `imagePositionOnPageLowerLeftY` e

`imagePositionOnPageUpperRightY`.

Es necesario indicar también una página de inserción en el parámetro `imagePage`.

imagePositionOnPageUpperRightY

Coordenada vertical superior derecha de la posición de la imagen (indicada mediante el parámetro `image`) dentro de la página.

Es necesario indicar el resto de coordenadas de la imagen mediante los parámetros

`imagePositionOnPageLowerLeftX`, `imagePositionOnPageLowerLeftY` e

`imagePositionOnPageUpperRightX`.

Es necesario indicar también una página de inserción en el parámetro `imagePage`.

attach

Contenido a añadir como adjunto al PDF, en formato Base64 (el adjunto será el binario decodificado). Este parámetro requiere que se haya establecido también el parámetro `attachFileName`.

attachFileName

Nombre de fichero para adjuntar el contenido binario indicado mediante `attach`. Este parámetro requiere que se haya establecido también el parámetro `attach`.

attachDescription

Descripción del contenido binario indicado mediante `attach`.

certificationLevel

Nivel de certificación de la firma PDF.

Los valores admitidos son numéricos, correspondiendo:

- 0 = Firma ordinaria no certificada (por defecto)
- 1 = Firma de autor. No se permite ningún cambio posterior en el documento
- 2 = Firma de autor certificada para formularios. Se permite únicamente el relleno posterior de los campos del formulario
- 3 = Firma certificada. Se permite únicamente el relleno posterior de los campos del formulario o el añadido de firmas de aprobación

signatureSubFilter

Nombre del sub-filtro en el diccionario PDF para indicar el tipo de la firma. Si no se indica este parámetro por defecto se usa `adbe.pkcs7.detached` (firma PAdES básica). Es posible indicar `ETSI.CAdES.detached` para generar una firma PAdES-BES, si bien el hacerlo puede causar que al añadir firmas adicionales al PDF se invaliden las ya existentes.

signatureField

Nombre del campo en donde insertar la firma. Si el documento PDF tiene ya un campo de firma pre-creado es posible utilizarlo para insertar la firma generada, referenciándolo por su nombre.

Si se indica un nombre de campo de firma que no exista en el documento PDF proporcionado, se generará una excepción.

signaturePage

Página del documento PDF donde insertar la firma. Puede usarse la constante `LAST_PAGE` para referirse a la última página del documento PDF si se desconoce el número total de páginas de este.

Este parámetro se ignora si se ha establecido valor al parámetro `signatureField`, y necesita que se establezcan valores válidos a los parámetros

`signaturePositionOnPageLowerLeftX`, `signaturePositionOnPageLowerLeftY`, `signaturePositionOnPageUpperRightX` y `signaturePositionOnPageUpperRightY`.

signaturePositionOnPageLowerLeftX

Coordenada horizontal inferior izquierda de la posición del recuadro visible de la firma dentro de la página.

Es necesario indicar el resto de coordenadas del recuadro mediante los parámetros

`signaturePositionOnPageLowerLeftY`, `signaturePositionOnPageUpperRightX` y `signaturePositionOnPageUpperRightY`.

Si no se indica una página en el parámetro `signaturePage` la firma se inserta en la última página del documento.

signaturePositionOnPageLowerLeftY

Coordenada vertical inferior izquierda de la posición del recuadro visible de la firma dentro de la página.

Es necesario indicar el resto de coordenadas del recuadro mediante los parámetros

`signaturePositionOnPageLowerLeftX`, `signaturePositionOnPageUpperRightX` y `signaturePositionOnPageUpperRightY`.

Si no se indica una página en el parámetro `signaturePage` la firma se inserta en la última página del documento.

signaturePositionOnPageUpperRightX

Coordenada horizontal superior derecha de la posición del recuadro visible de la firma dentro de la página.

Es necesario indicar el resto de coordenadas del recuadro mediante los parámetros `signaturePositionOnPageLowerLeftX`, `signaturePositionOnPageLowerLeftY` y `signaturePositionOnPageUpperRightY`.

Si no se indica una página en el parámetro `signaturePage` la firma se inserta en la última página del documento.

signaturePositionOnPageUpperRightY

Coordenada vertical superior derecha de la posición del recuadro visible de la firma dentro de la página.

Es necesario indicar el resto de coordenadas del recuadro mediante los parámetros `signaturePositionOnPageLowerLeftX`, `signaturePositionOnPageLowerLeftY` y `signaturePositionOnPageUpperRightX`.

Si no se indica una página en el parámetro `signaturePage` la firma se inserta en la última página del documento.

signatureRubricImage

Imagen JPEG codificada en Base64 de la rúbrica de la firma manuscrita que se desea aparezca como firma visible en el PDF.

layer2Text

Texto a escribir dentro de la "capa 2" de la firma visible.

Este texto se escribe únicamente si no se ha especificado una imagen de rúbrica, y necesita que se indique la página y la situación dónde mostrar el recuadro de firma mediante los parámetros `signaturePositionOnPageLowerLeftX`, `signaturePositionOnPageLowerLeftY`, `signaturePositionOnPageUpperRightX`, `signaturePositionOnPageUpperRightY` y `signaturePage`.

layer2FontFamily

Tipo de letra a usar en el texto de la "capa 2" de la firma visible. Este parámetro requiere que se haya establecido también el parámetro `layer2Text`.

Los valores admitidos son numéricos, correspondiendo:

- 0 = Courier (tipo por defecto)
- 1 = Helvética
- 2 = Times Roman
- 3 = Symbol
- 4 = ZapfDingBats

layer2FontSize

Tamaño de letra a usar en el texto de la "capa 2" de la firma visible. Este parámetro requiere que se haya establecido también el parámetro `layer2Text`. Los valores admitidos son numéricos (y el valor por defecto es 12).

layer2FontStyle

Estilo del tipo de letra a usar en el texto de la "capa 2" de la firma visible. Este parámetro requiere que se haya establecido también el parámetro `layer2Text`. Los valores admitidos son numéricos, correspondiendo:

- 0 = Normal (estilo por defecto)
- 1 = Negrita
- 2 = Cursiva
- 3 = Negrita y cursiva
- 4 = Subrayado
- 8 = Tachado

Es posible combinar estilos aplicando la operación lógica *o* sobre los valores numéricos a combinar.

layer2FontColor

Color del texto de la "capa 2" de la firma visible. Este parámetro requiere que se haya establecido también el parámetro `layer2Text`.

Los valores admitidos son textuales (se ignora entre mayúsculas y minúsculas), soportándose:

- *black* = Negro (color por defecto)
- *white* = Blanco
- *gray* = Gris
- *lightGray* = Gris claro
- *darkGray* = Gris oscuro
- *red* = Rojo
- *pink* = Rosa

signReason

Razón por la que se realiza la firma (este dato se añade al diccionario PDF, y no a la propia firma).

signatureProductionCity (propiedad compartida con XAdES y CAdES)

Ciudad en la que se realiza la firma (este dato se añade al diccionario PDF, y no a la propia firma).

signerContact

Contacto del firmante, usualmente una dirección de correo electrónico (este dato se añade al diccionario PDF, y no a la propia firma).

policyIdentifier (propiedad compartida con XAdES y CAdES)

Identificador de la política de firma. Debe ser un OID (o una URN de tipo OID) que identifique unívocamente la política en formato ASN.1 procesable.

policyIdentifierHash (propiedad compartida con XAdES y CAdES)

Huella digital del documento de política de firma (normalmente del mismo fichero en formato ASN.1 procesable). Si no se indica una huella digital y el parámetro `policyIdentifier` no es una URL accesible universalmente se provocará un error, mientras que si no se indica una huella digital pero el parámetro `policyIdentifier` es una URL accesible universalmente, se descargará el fichero apuntado por la URL para calcular la huella digital *al vuelo*.

policyIdentifierHashAlgorithm (propiedad compartida con XAdES y CAdES)

Algoritmo usado para el cálculo de la huella digital indicada en el parámetro `policyIdentifierHash`. Es obligatorio indicarlo cuando se proporciona una huella digital distinta de 0.

policyQualifier (propiedad compartida con XAdES y CAdES)

URL que apunta al documento descriptivo de la política de firma (normalmente un documento PDF con una descripción textual).

ownerPassword

Contraseña de apertura del PDF (contraseña del propietario) si este estaba cifrado. No se soporta la firma de documentos PDF cifrados con certificados o con algoritmo AES256.

headless

Evita cualquier interacción con el usuario si se establece a `true`, si no se establece o se establece a `false` actúa normalmente (puede mostrar diálogos, por ejemplo, para solicitar las contraseñas de los PDF cifrados). Útil para los procesos desatendidos y por lotes.

allowSigningCertifiedPdfs

Si se establece a `true` permite la firma o cofirma de PDF certificados sin consultarlo al usuario, si se establece a `false` o cualquier otro valor se lanza una excepción en caso de intentar firmar o cofirmar un PDF certificado y si no se establece se mostrará un diálogo al

usuario para que confirme que desea realizar la firma a pesar de que el resultado serán una firma no válida.

Si el parámetro `headless` está establecido a `true`, no podrá mostrar el diálogo de confirmación así que llegados a este punto se lanzará una excepción.

No se soporta el cifrado de documentos PDF con certificados o con algoritmo AES256.

signingCertificateV2 (propiedad compartida con CAdES)

Si se indica a `true` se utilizará *SigningCertificateV2*, si se indica cualquier otra cosa *SigningCertificateV1*. Si no se indica nada, se utilizará V1 para las firmas SHA1 y V2 para el resto.

14 Problemas conocidos

14.1 Uso de ciertos algoritmos con Windows XP

Cuando se realizan firmas sobre Windows XP, en ciertas configuraciones (Service Pack, versión de Java, arquitectura x86 o x64, controladores de tarjeta inteligente), no es posible usar RSA con SHA-2 como algoritmo de firma.

Actualice siempre que sea posible a una versión actual de su sistema operativo, y si no es posible, use SHA-256 como alternativa a SHA-512 o SHA-384 cuando estos dos últimos no funcionen, y SHA-1 cuando no funcione ningún algoritmo de la familia SHA-2 (SHA-224, SHA-256, SHA-384 y SHA-512), si bien en este último caso debe tener siempre en cuenta que SHA-1 es actualmente un algoritmo obsoleto y vulnerable.

14.2 Google Chrome no ejecuta ni el MiniApplet Cliente @firma ni ningún otro Applet de Java a pesar de tener correctamente instalado el JRE

14.2.1 Microsoft Windows

En las versiones de Google Chrome iguales o superiores a la 45 no es posible ejecutar Applets de Java de ningún modo, incluyendo el MiniApplet Cliente @firma.

En las versiones de Google Chrome sobre Microsoft Windows desde la 42 a la 44, la ejecución de Applets de Java está deshabilitada por defecto, por lo que es necesario habilitar manualmente esta característica siguiendo este procedimiento:

1. Escribir la siguiente dirección en la barra de URL del navegador: **chrome://flags**
2. Localizar el apartado: "**Habilitar NPAPI**" y presionar "**Habilitar**".
3. Reiniciar el navegador.

14.2.2 Apple OS X y Linux

En versiones de Google Chrome iguales o superiores a la 42 no es posible ejecutar Applets de Java de ningún modo, incluyendo el MiniApplet Cliente @firma.

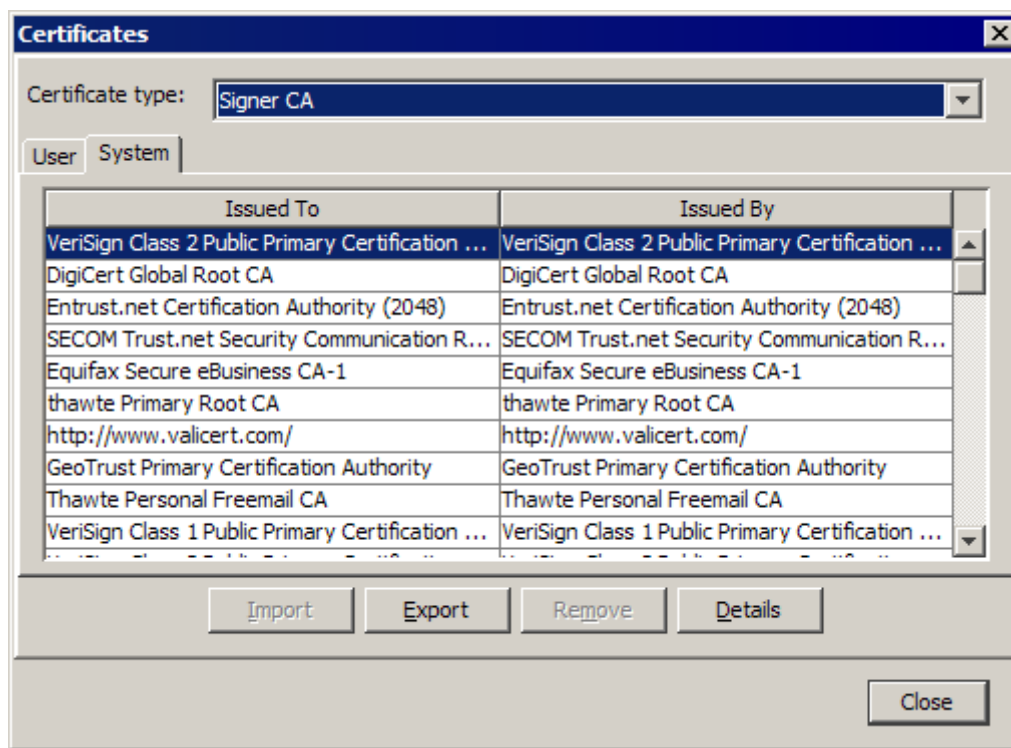
14.3 Diálogos de advertencia al usuario en Java 7u55 y posteriores (incluyendo Java 8)

Dado que el MiniApplet Cliente @firma se distribuye como un Applet de Java genérico susceptible de ser usado en cualquier sitio Web, no incorpora restricciones en este sentido.

Java, a partir de la versión 7u55 obliga a los Applets a indicar el sitio Web (o al menos su dominio) en el que se van a publicar, mostrando un diálogo de advertencia a los usuarios finales si no se hace así.

Para evitar estos diálogos, es necesario un proceso por parte del integrador consistente en los siguientes pasos:

1. Modificación del JAR del MiniApplet para indicar el sitio Web de publicación.
 - a. Abra el JAR como si de un ZIP se tratase (puede cambiar temporalmente la extensión de JAR a ZIP para mayor comodidad).
 - b. Modifique el fichero META-INF/MANIFEST.MF, y dentro de este indique en el atributo Caller-Allowable-Codebase el sitio Web donde va a publicar el MiniApplet, siguiendo las instrucciones publicadas en:
 - http://docs.oracle.com/javase/8/docs/technotes/guides/jweb/security/manifest.html#caller_allowable
 - c. Empaquete de nuevo el JAR, para lo cual puede usar una herramienta normal de compresión ZIP o la propia de empaquetado JAR de Java:
 - <http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jar.html>
2. Refirma del MiniApplet.
 - a. Dado que la modificación del MANIFEST.MF invalidará la firma original del MiniApplet, debe refirmarlo con un certificado propio, para ello puede usar la herramienta JarSigner de Java:
 - <http://docs.oracle.com/javase/tutorial/deployment/jar/signing.html>
 - <http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jarsigner.html>
 - b. Es importante que utilice un certificado de firma de código emitido por una autoridad de certificación reconocida como de confianza por Oracle. Puede encontrar la lista de entidades reconocidas en el “Panel de Control de Java”, dentro del botón “Gestionar Certificados” de la pestaña “Seguridad”, y dentro de la ventana en la sección “Autoridades de Certificación de firmantes” del “Sistema” (esta lista se actualiza con relativa frecuencia, consulte siempre con la versión más reciente de Java):



De forma general, puede encontrar información sobre como modificar el MANIFEST.MF de un programa Java en: <http://docs.oracle.com/javase/tutorial/deployment/jar/modman.html>

Puede encontrar más información sobre este asunto en:

- <http://www.oracle.com/technetwork/java/javase/7u55-relnotes-2177812.html>
- http://bugs.java.com/view_bug.do?bug_id=8033731

En cualquier caso, actualice siempre el entorno de ejecución de Java de los clientes de firma a la última versión disponible.

14.4 Error “el conjunto de claves no existe” al firmar con el almacén de claves de Windows

En ciertas ocasiones, y especialmente cuando se usan tarjetas de FNMT-RCM (CERES, DNle, APE, etc.), al firmar en un entorno operativo Windows, la operación finaliza con error y se muestra en consola el mensaje “El conjunto de claves no existe” (o “Keyset does not exist” si se tiene un Windows en inglés).

Este problema, que si bien puede darse en cualquier versión de Windows es más común en Windows XP, no tiene solución, y se debe a una incompatibilidad de Java con los controladores CAPI de Windows instalados en el sistema del usuario (por ejemplo, los controladores de FNMT-RCM).

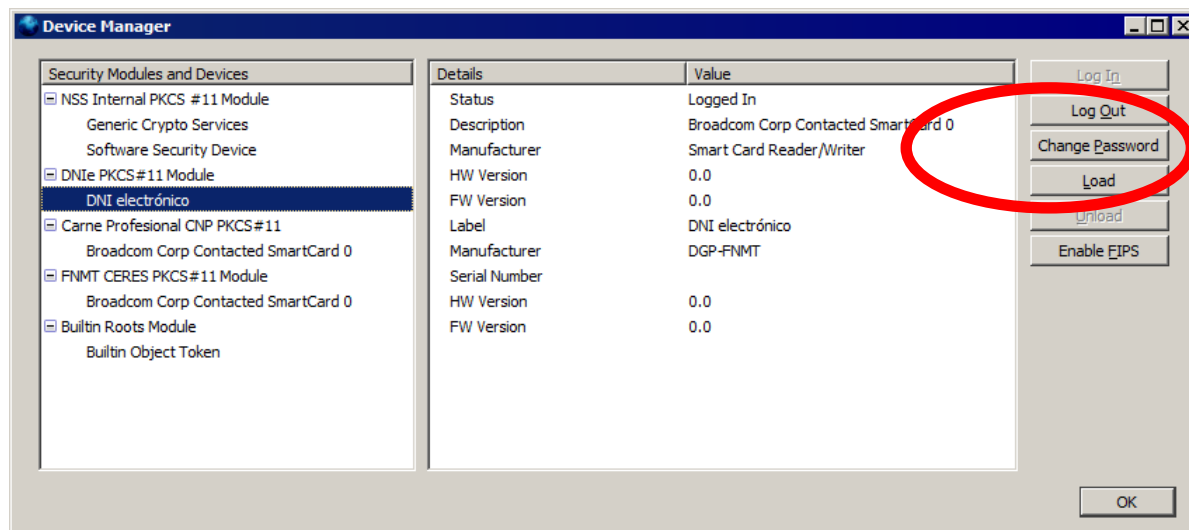
Pruebe a actualizar tanto en entorno de ejecución de Java como los posibles controladores de tarjetas que tenga instalados a la última versión disponible.

Si el problema se da únicamente al intentar firmar con una tarjeta inteligente o un almacén de claves distinto del central de Windows, abra una incidencia contra el proveedor de este hardware/software de almacén de claves.

14.5 Con el navegador Mozilla Firefox y DNle (DNI Electrónico) el Applet se queda bloqueado y no muestra el diálogo de selección de certificados, desbloqueándose si retiro el DNle del lector

El controlador PKCS#11 oficial del DNle 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 DNle en el momento en el que el MiniApplet 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 DNle por parte de Mozilla / Firefox puede tardar varios minutos si el usuario no interviene, por lo que conviene forzar manualmente este cierre:

- Extraer el DNle 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 MiniApplet 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 “DNle 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 DNle sin dar tiempo al MiniApplet 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/>

14.6 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.

14.7 El Applet no detecta ningún certificado bajo Mozilla / Firefox

El MiniApplet 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`.

Si las bibliotecas NSS correspondientes a la versión de Mozilla Firefox instalada se encuentra en algún otro directorio, es posible hacérselo saber al Applet mediante el sistema indicado en el apartado Uso exclusivo de certificados accesibles mediante PKCS#11 en Mozilla Firefox

Estableciendo a `true` la propiedad del sistema

`"es.gob.afirma.keystores.mozilla.LoadSscdOnly"` (debe hacerse antes de la carga del MiniApplet) se activa un modo especial de funcionamiento que provoca que se ignoren los certificados y claves del almacén central de certificados de Mozilla Firefox (NSS) y se usen exclusivamente los accesibles mediante módulos de seguridad PKCS#11 adicionales.

Esta funcionalidad puede resultar útil para forzar el uso de tarjetas inteligentes o dispositivos de almacén USB y descartar los certificados alojados en el almacén software del navegador.

Consulte el apartado Configuración a través de propiedades del sistema para saber cómo establecer propiedades de sistema.

Forzar ruta del almacén de Mozilla Firefox.

14.8 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 MiniApplet Cliente @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.

14.9 El servicio de firma trifásica genera un error al realizar firmas XAdES en servidores JBoss u otros

A partir de determinada versión del servidor de aplicaciones JBoss, este incorpora de serie diversas bibliotecas Java que entran en conflicto con la versión de estas mismas bibliotecas incorporadas en el JRE/JDK de Oracle. A saber, las bibliotecas Xalan y Xerces de Apache. Esto deriva en que durante el proceso de firma se produce un error de *casting* o similar, según sea la operación y versión de JBoss. El error se produce a que la JVM da preferencia a las bibliotecas proporcionadas por el servidor de aplicaciones frente a las suyas propias.

Frente a esto, se propone una sucesión de posibles soluciones de tal forma que si la primera de ellas no es viable se intente la siguiente solución y así sucesivamente:

- **Solución 1:** Revisar la documentación del servidor de aplicaciones en cuestión para comprobar si existe un mecanismo documentado para dar preferencias a las bibliotecas de Java frente a las bibliotecas importadas por el propio servidor de aplicaciones.

- **Solución 2:** Identificar si existe una versión concreta de las bibliotecas afectadas (Xalan y Xerces principalmente) que puedan importarse en el servicio de firma trifásica, que sea compatible con este servicio y no genere conflicto con el servidor de aplicaciones en cuestión. Para incorporar estas bibliotecas, se deberán copiar al directorio interno “WEB-INF/lib” dentro de “afirma-server-triphase-signer.war”. De esta forma, conseguiremos que la JVM cargue estas bibliotecas en lugar de las que proporciona el servidor de aplicaciones.
 - Las bibliotecas de Xalan y Xerces se pueden encontrar en el sitio web de estos proyectos:
 - <https://xml.apache.org/xalan-j/>
 - <http://xerces.apache.org/>
- **Solución 3:** Una opción menos óptima que la anterior, aunque puede que más sencilla, viene a ser identificar el fichero “rt.jar” de la JVM de nuestro servidor e introducirlo en el directorio de bibliotecas del WAR del servicio de firma trifásico (directorio WEB-INF/lib). Al igual que en el caso anterior, así conseguiremos que la JVM dé prioridad a la versión de Xalan/Xerces que incorpora esta JAR, los por defecto de Java, en lugar de a las bibliotecas del servidor de aplicaciones.
- **Solución 4:** Si todo lo anterior fracasase, pero supiésemos que ninguna otra aplicación hace uso de estas bibliotecas del servidor de aplicaciones, podríamos eliminarlas del directorio de JBoss. De esta forma, la única versión que quedaría sería la por defecto de Java y sería esta versión la que se utilizaría. Esta opción es muy radical y requeriría que se probasen las distintas aplicaciones desplegadas para asegurar que no genera errores en ellos.

14.9.1 Despliegue del servicio de firma trifásica sobre JBoss

El caso más común en el que aparece el problema descrito es en JBoss, versiones 7 y EAP 6. Para este caso concreto, se puede configurar el despliegue para indicarle las bibliotecas para las que existen dependencias, de tal forma que no usará sus propias librerías. Para ello, agregaremos al directorio WEB-INF del despliegue un fichero “jboss-deployment-structure.xml” con el siguiente contenido:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.api"/>
      <module name="org.apache.santuario.xmlsec"/>
      <module name="org.apache.xerces" />
      <system export="true">
        <paths>
          <path name="com/sun/org/apache/xerces/internal/dom"/>
        </paths>
      </system export>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```




```
</system>  
</dependencies>  
</deployment>  
</jboss-deployment-structure>
```



Esta obra está bajo una licencia [Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 Unported](https://creativecommons.org/licenses/by-nc-sa/3.0/).