Manual del integrador del MiniApplet v1.6 del Cliente @firma y la compatibilidad de sus despliegues con AutoFirma 1.6.5 y el Cliente @firma móvil

NOTA: Todas las referencias realizadas a **miniapplet.js** deberán entenderse realizadas a **autoscript.js**

Índice de contenidos

1 Introducción 8
1.1 Productos de terceros incluidos con el MiniApplet @firma 9
1.2 Adecuación al Esquema Nacional de Seguridad 9
2 Requisitos mínimos 10
2.1 Entorno Cliente 10
2.1.1 MiniApplet Cliente @firma 10
2.1.2 AutoFirma.... 14
2.2 Dispositivos móviles 14
2.3 Entorno Servidor 14
3 Funcionalidad proporcionada por el MiniApplet @firma 15
3.1 Formatos de firma soportados por el MiniApplet @firma 15
3.1.1 CAdES (CMS Advanced Electronic Signature).... 15
3.1.2 XAdES (XML Advanced Electronic Signature).... 15
3.1.3 FacturaE (Factura electrónica) 16
3.1.4 PAdES (PDF Advanced Electronic Signature).... 16
3.1.5 ODF (Open Document Format – Firmas de documentos LibreOffice / OpenOffice.org).... 17
3.1.6 OOXML (Office Open XML – Firmas de documentos Microsoft Office).... 17
3.2 Selección automática del formato de firma.... 18
3.3 Formatos de firma no recomendados.... 18
3.4 Uso de certificados y claves privadas por parte del MiniApplet @firma.... 19
3.4.1 Establecimiento manual del almacén de certificados y claves 20
3.4.2 Uso de tarjetas inteligentes 22
3.4.3 Uso del DNIe.... 23
3.4.4 Información específica al uso de claves con NSS (Firefox en Windows o cualquier navegador en Linux).... 24
3.5 Envío anónimo de estadísticas por parte del MiniApplet y AutoFirma 30
4 Despliegue del MiniApplet @firma.... 31
4.1 Importación de las bibliotecas JavaScript 31
4.1.1 Importación en páginas Web generadas dinámicamente 32

4.2 Carga del MiniApplet 32
4.2.1 Entornos no compatibles con Applets y problemas durante la carga 33
4.2.2 Carga directa de AutoFirma y los clientes móviles 34
4.3 Configuración del idioma 35
4.4 Restricción según desfase horario con el servidor 36
4.5 Firma del JAR del MiniApplet 37
4.6 Despliegue en entornos de Web dinámica y en servidores no estáticos 37
5 El proceso de firma electrónica 39
5.1 Selección del contenido a firmar 39
5.1.1 La conversión de datos a Base64 39
5.1.2 Selección de contenido desde ficheros en disco 40
5.1.3 Selección de datos remotos 41
5.1.4 Selección de objetivo de las contrafirmas 41
5.2 Selección del certificado y clave privada de firma 41
5.2.1 Nota técnica: La cadena de confianza de un certificado 42
5.3 Firma 43
5.4 Recogida de resultados 43
6 Funciones del MiniApplet @firma 45
6.1 Uso del API desde JavaScript 45
6.2 Obtención de resultados 45
6.2.1 Obtención directa de resultados 46
6.2.2 Obtención de resultados mediante *Callbacks* 47
6.3 Firma electrónica 48
6.4 Firmas electrónicas múltiples 50
6.4.1 Cofirmas 51
6.4.2 Contrafirmas 53
6.5 Firmas/Multifirmas trifásicas 58
6.5.1 Realizar firma trifásicas con el MiniApplet Cliente @firma 59
6.5.2 Servicio de firma trifásica 60
6.6 Firmas/Multifirmas masivas 64

6.7 Firma por lotes predefinidos 65
6.7.1 Configuración del servicio 66
6.7.2 Creación de los lotes 67
6.7.3 Respuesta a una ejecución de un lote 74
6.7.4 Descripción del modo transaccional de ejecución de los lotes 76
6.8 Firma/multifirma y guardado del resultado 77
6.9 Gestión de ficheros 80
6.9.1 Guardado de datos en disco 80
6.9.2 Selección y recuperación de un fichero por parte del usuario 81
6.9.3 Selección y recuperación de múltiples ficheros por parte del usuario 83
6.10 Utilidad 85
6.10.1 Eco 85
6.10.2 Obtención de los mensajes de error 86
6.10.3 Conversión de una cadena Base64 a texto 86
6.10.4 Conversión de un texto a cadena Base64 87
6.10.5 Descarga de datos remotos 88
6.10.6 Selección de certificado 89
6.10.7 Obtención del log de la aplicación 90
7 Configuración de las operaciones 91
7.1 Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma 91
7.1.1 Parámetros adicionales no soportados 92
7.2 Selección automática de certificados 92
7.3 Configuración del filtro de certificados 93
7.4 Configuración de la política de firma 99
7.4.1 Configuración manual 99
7.4.2 Política de firma de la AGE v1.9 100
7.4.3 Política de firma de la AGE v1.8 101
7.4.4 Política de firma de Factura electrónica (Facturae) 102
7.5 Configuración de parámetros de la JVM 102
7.5.1 Configuración de la Máquina Virtual de Java para tratamiento de ficheros grandes 102

7.6 Configuración a través de propiedades del sistema 103
8 Gestión de errores 104
9 Compatibilidad con dispositivos móviles y AutoFirma 106
9.1 Arquitectura del sistema 107
9.1.1 Entornos de escritorio (Windows, Linux y Mac OS X) con nuevos navegadores 108
9.1.2 Dispositivos móviles, Internet Explorer 10 e inferiores, Microsoft Edge y Safari 10 108
9.2 Despliegues compatibles 110
9.3 Configuración de la invocación a AutoFirma 112
9.4 Servicios auxiliares de AutoFirma y el Cliente @firma móvil 113
9.4.1 Notas sobre los servicios de almacenaje y recuperación 114
9.5 Notificaciones al usuario 114
9.6 Limitaciones 115
9.6.1 Limitaciones de formato 115
9.6.2 Limitaciones funcionales 116
9.7 Compatibilidad de versiones 118
10 AutoFirma WebStart 118
10.1 Requisitos 119
10.2 Despliegue de AutoFirma WebStart 120
10.3 Advertencias al usuario 120
11 Información específica para firmas CAdES 121
11.1 Algoritmos de firma 121
11.2 Firmas CAdES implícitas o explícitas 121
11.3 Parámetros adicionales 121
11.3.1 Firma y cofirma 121
11.3.2 Contrafirma 124
12 Información específica para firmas XAdES 126
12.1 Tratamiento de las hojas de estilo XSL de los XML a firmar 126
12.2 Algoritmos de firma 127
12.3 Algoritmos de huella digital para las referencias 128
12.4 Situación del nodo de firma en XAdES Enveloped 128

12.5 Uso de estructuras *Manifest* en firmas XAdES 130
12.5.1 Formatos de XAdES compatibles con estructuras *Manifest* 131
12.5.2 Cofirmas y contrafirmas *Manifest* 131
12.5.3 Firma *manifest* a partir de la huella digital de los datos 132
12.6 Parámetros adicionales 133
12.6.1 Firma y cofirma 133
12.6.2 Contrafirma 142
12.7 Firmas XAdES "explícitas" 144
13 Información específica para firma de facturas electrónicas 145
13.1 Operaciones no soportadas y notas de interés 145
13.2 Algoritmos de firma 146
13.3 Parámetros adicionales 146
14 Información específica para firma de documentos en formato ODF 147
14.1 Algoritmos de firma 147
15 Información específica para firma de documentos en formato OOXML 147
16 Información específica para firmas PAdES 148
16.1 Creación de una firma visible 148
16.2 Inserción de una imagen en un documento PDF antes de ser firmado 153
16.3 Operaciones no soportadas y notas de interés 154
16.4 Firma de documentos PDF cifrados o protegidos con contraseña 155
16.5 Algoritmos de firma 155
16.6 Parámetros adicionales 155
17 Problemas conocidos 165
17.1 El MiniApplet @firma no funciona adecuadamente con las versiones de Firefox de la 42.0 a la 47.0 165
17.2 No se puede acceder al almacén de claves de Firefox 49.0 y superiores 165
17.3 No se puede acceder al almacén de claves de Firefox 58 165
17.4 Uso de ciertos algoritmos con Windows XP 166
17.5 Diálogos de advertencia al usuario en Java 7u55 y posteriores (incluyendo Java 8) 166
17.6 Error "El conjunto de claves no existe" al firmar con el almacén de claves de Windows 168
17.7 No se detecta la inserción/extracción del DNIe en el lector (u otra tarjeta inteligente) 168

17.8 El MiniApplet no detecta ningún certificado bajo Mozilla / Firefox en Linux 168

17.9 El MiniApplet no permite la firma de PDF con ciertos certificados 169

17.10 El servicio de firma trifásica genera un error al realizar firmas XAdES en servidores JBoss 169

17.11 Las firmas con DNIe requieren que se introduzca el PIN del DNIe por cada operación de firma 170

17.12 No aparecen los certificados de las tarjetas de la FNMT o empleado público al ejecutar el Cliente @firma desde Firefox..... 170

17.13 Error en la firma con tarjetas de la FNMT y empleado público desde Firefox 170

17.14 Se solicita confirmar la operación de firma cuando se seleccionan algunos certificados de una tarjeta de la FNMT 171

17.15 El Cliente @firma no muestra en OS X el título de los diálogos de cargar y guardado de ficheros 171

17.16 Error al cargar el listado de certificados después del cambio en caliente del almacén por defecto 171

17.17 AutoFirma no puede comunicarse con el navegador en macOS 171

17.18 Sólo se realiza la firma del primer documento de una serie cuando se realizan las firmas desde Google Chrome..... 172

17.19 No se abre la aplicación de firma al realizar la firma desde Google Chrome 173

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 puede integrarse en trámites web por medio del JavaScript de despliegue que lo acompaña. De esta forma, permite que se ejecuten operaciones de firma en estos trámites usando los certificados del ciudadano. En caso de no poder ejecutarse el MiniApplet por problemas de compatibilidad de entorno, razones confianza o permisos restringidos, el JavaScript de despliegue derivará las tareas de firma en una aplicación de nativa instalada en el sistema (AutoFirma y Clientes @firma móvil). Estas aplicaciones deberán estar instaladas previamente en el sistema. Las instrucciones y explicaciones detalladas en el presente documento se refieren por norma al MiniApplet, pero afectan igualmente a AutoFirma y los clientes @firma móviles salvo en los aspectos detallados en el apartado Compatibilidad con dispositivos móviles y AutoFirma.

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

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

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

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

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

1.1 Productos de terceros incluidos con el MiniApplet @firma

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

- JXAdES (https://github.com/universitatjaumei/jxades)
- SpongyCastle (https://rtyley.github.io/spongycastle/)
- 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/)

1.2 Adecuación al Esquema Nacional de Seguridad

Los productos de la Suite de @firma pueden contener entre los algoritmos disponibles, algunos no recomendados por la Guía 807 del Esquema Nacional de Seguridad (ENS; editada por el Centro Criptológico Nacional, CCN) vigente en el momento de publicación de este documento. Por lo que queda bajo la responsabilidad de las aplicaciones que hacen uso de estos productos el configurar adecuadamente las llamadas a los mismos para generar el resultado esperado, válido y adecuado para ese momento y el nivel de seguridad deseado, utilizando para ello algoritmos de la familia SHA-2 tal y como especifica dicha norma para la generación de firmas electrónicas.

Puede consultar la norma vigente desde el siguiente enlace:

https://www.ccn-cert.cni.es/series-ccn-stic/800-guia-esquema-nacional-de-seguridad/513-ccn-stic-807-criptologia-de-empleo-en-el-ens/file.html

2 Requisitos mínimos

2.1 Entorno Cliente

2.1.1 MiniApplet Cliente @firma

Entorno de ejecución de Java:

- 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 y 10, 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 Yosemite (10.10.5 o superior) o El Capitán (10.11.1).

Navegador Web:

- Mozilla Firefox 4.0 o superior
  - Mozilla Firefox, entre sus versiones 40 y 47, sufre un problema que dificulta el uso de applets. Se recomienda el uso de la versión 49 o superior.
  - En caso de querer utilizar el almacén de claves de Firefox independientemente del navegador que utilice el usuario, se recomienda utilizar un Firefox de 32 bits en caso de que el sistema operativo objetivo sea Wndows y un Firefox de 64 bits en caso de OS X. De esta forma, la arquitectura del navegador por defecto del sistema y la de Firefox será la misma y la JVM cargada por el Java Plugin podrá acceder al almacén del sistema y al de Firefox.
- 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).

  - Las versiones 8 y 9 de 64bits de Internet Explorer requieren que se tenga una versión de Java de 64 bits. Sin embargo, Internet Explorer 10 y superiores, independientemente de la arquitectura del navegador, siempre utilizan Java 32 bits.
  - Internet Explorer puede funcionar en modo de compatibilidad con versiones anteriores. El Cliente @firma no es compatible con los modos de compatibilidad con Internet Explorer 7 y anteriores, independientemente de la versión real del navegador utilizado.
- Microsoft Edge 20 o superior. Es necesario tener previamente instalado el programa AutoFirma en el sistema del usuario.

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.

Nota para usuarios de Firefox 49 o superior y Windows: La carga del almacén de claves y certificados de Firefox 49 o superior por parte del MiniApplet @firma necesita que el sistema tenga instalado los entornos de ejecución redistribuibles de Microsoft Visual C++ 2015. 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 2015 (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=53840

El proceso de instalación puede requerir permisos de administrador.

Nota para usuarios de Microsoft Edge: Microsoft Edge no soporta la ejecución de Applets de Java, por lo que los usuarios necesitarán tener preinstalado AutoFirma para ejecutar las operaciones de firma desde este navegador. Adicionalmente, no se soporta la comunicación a

través de sockets entre AutoFirma y Microsoft Edge, por lo que los despliegues del MiniApplet deberán configurar un servidor intermedio para dar soporte a este navegador.

2.1.1.1 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

2.1.1.2 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"/>
```

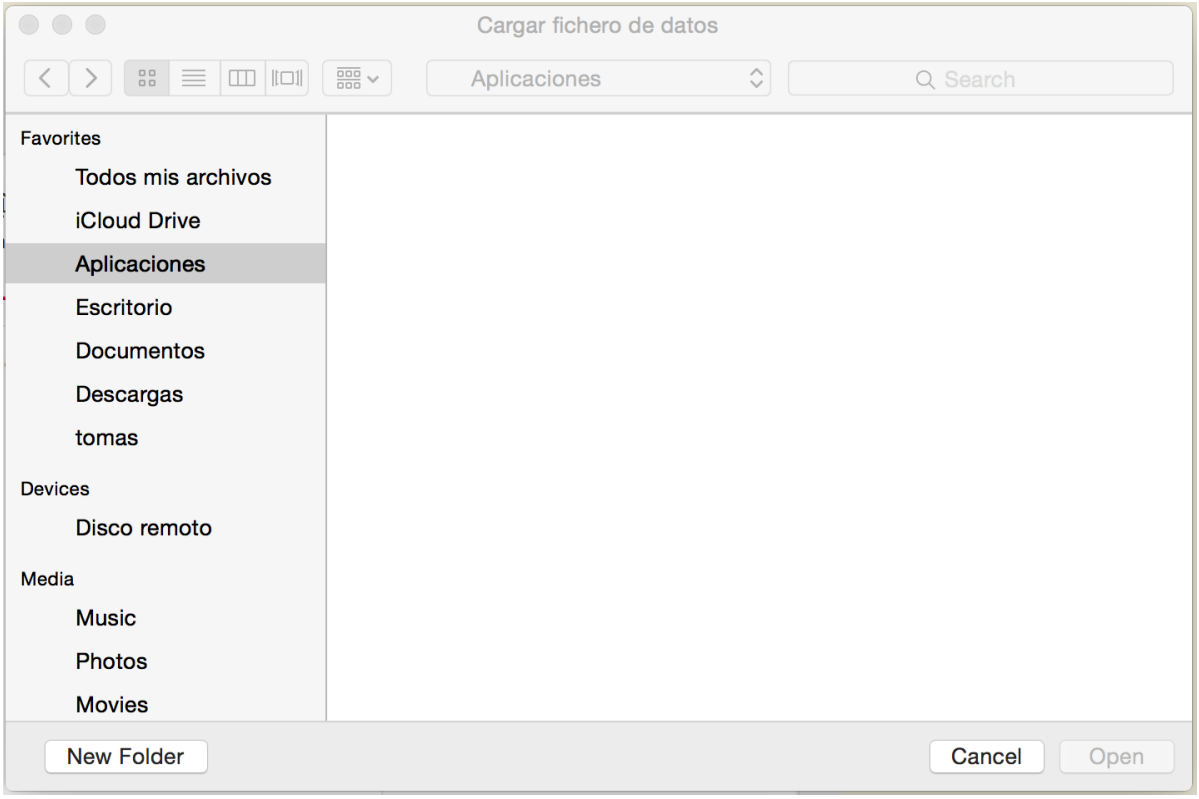
Puede encontrar más información sobre complementos de navegador (*plugins*) en Internet Explorer sobre Metro en Windows 8 en:

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

2.1.1.3 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

2.1.2 AutoFirma

Para conocer los modos de instalación de AutoFirma y los requisitos del sistema, consulte el documento "*AF_manual_instalacion_y_gestion_ES*".

2.2 Dispositivos móviles

Los *Applets* Java, como el MiniApplet @firma, no puede ejecutarse desde dispositivos móviles como 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 y AutoFirma 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 1.7 o superior con JEE 6 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 de formatos de firma, pero no otras funcionalidades como sobres digitales o cifrados simétricos, como hacía el Applet Cliente @firma. Adicionalmente, se proporciona un conjunto de métodos de utilidad y opciones de operación.

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

AutoFirma y los clientes móviles de @firma no soportan todas las operaciones del MiniApplet @firma. Para consultar las funcionalidades disponibles en cada una de estas operaciones, consulte el apartado Compatibilidad con dispositivos móviles y AutoFirma.

3.1 Formatos de firma soportados por el MiniApplet @firma

El Cliente @firma soporta los siguientes formatos de firma, entre los que se incluyen los formatos baseline en su nivel "B-Level". Las firmas en estos formatos, pueden ser mejoradas a posteriori con otros productos para incluir la información longeva acorde al resto de formatos baseline.

3.1.1 CAdES (CMS Advanced Electronic Signature)

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

- CAdES B-Level / CAdES-BES
- CAdES-EPES

Las firmas CAdES que el Cliente @firma genera por defecto no incluyen ni los datos firmados (firma explícita) ni política de firma y concuerdan simultáneamente con los formatos CAdES B-Level y CAdES-BES.

Hay que tener en cuenta que el perfil CAdES-EPES también se considera de tipo B-Level, así que, cuando declaremos una política en nuestra firma, un validador de firmas podría identificarla tanto como EPES como B-Level.

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 B-Level / XAdES-BES
- XAdES-EPES

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

- *Enveloping* (Por defecto)

- *Enveloped*
- *Internally Detached*

Hay que tener en cuenta que el perfil XAdES-EPES también se considera de tipo B-Level, así que, cuando declaremos una política en nuestra firma, un validador nos podría indicar que nuestra firma es EPES o B-Level según su configuración.

Hay que tener en cuenta que el perfil XAdES-EPES también se considera de tipo B-Level, así que, cuando declaremos una política en nuestra firma, un validador de firmas podría identificarla tanto como EPES como B-Level.

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

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
- PAdES B-Level / PAdES-BES
- PAdES-EPES

Por defecto, el Cliente @firma genera firmas PAdES-Básico a menos que se indique una política de firma AGE 1.9 o superior (en cuyo caso, por adecuación, se pasa a generar PAdES-EPES) o se indique explícitamente mediante un parámetro adicional que se desea usar PAdES B-Level / PAdES-BES.

Hay que tener en cuenta que el perfil PAdES-EPES también se considera de tipo B-Level, así que, cuando declaremos una política en nuestra firma, un validador de firmas podría identificarla tanto como EPES como B-Level.

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.

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).

3.1.5.1 Algoritmo de firma para documentos ODF

El formato ODF únicamente admite el algoritmo de firma *SHA1withRSA*, por lo que siempre se usará ese algoritmo con independencia del valor que se indique como algoritmo en la invocación (aunque se indique un valor distinto a *SHA1withRSA*, se usará siempre *SHA1withRSA*).

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 2013	Office 2016
Word (docx)	NO	N/A	SÍ	N/A	SÍ	SÍ
Excel (xlsx)	NO	N/A	SÍ	N/A	SÍ	SÍ
PowerPoint (pptx)	NO	N/A	SÍ	N/A	SÍ	SÍ
Project	NO	NO	NO	NO	NO	NO
OneNote	NO	NO	NO	NO	NO	NO
Visio	NO	NA	NO	NO	NO	NO
XPS	NO	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 OOXML 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 8u51 como mínimo.

3.2 Selección automática del formato de firma

El MiniApplet @firma permite que se indique la palabra clave "AUTO" como formato de firma. En este caso, el comportamiento variará según la operación de firma que se intente realizar:

- En una operación de firma simple, se realizará la firma usando como formato aquel predefinido para el tipo de dato de entrada:
    - Si los datos son un documento PDF, se utilizará el formato de firma PAdES.
    - Si los datos son una factura electrónica, se utilizará el formato de firma FacturaE.
    - Si los datos son XML, se utilizará el formato de firma XAdES.
    - Si los datos son un documento ODF, se utilizará el formato de firma ODF.
    - Si los datos son un documento OOXML, se utilizará el formato de firma OOXML.
    - En cualquier otro caso, se utilizará el formato de firma CAdES.

  ADVERTENCIA: El uso del valor "AUTO" como formato en las operaciones de firma se encuentra disponible a partir de la versión 1.4 del MiniApplet. No se recomienda el uso de este valor cuando el objeto de la firma sea el registro de documentación del usuario por parte de la entidad o motivo similar que permita que implique el registro de las firmas del usuario por parte de la entidad y sea susceptible de que se firmen documentos en diversos formatos, ya que se dificultaría la gestión de las firmas.

- En una operación de cofirma o contrafirma, se realizará esta cofirma o contrafirma en el formato que estuviese la firma electrónica de entrada.

3.3 Formatos de firma no recomendados

El MiniApplet @firma soporta, además de los mencionados, la realización de firmas en los formatos CMS/PKCS#7 y XMLDSig (XML Digital Signature). Sin embargo, se recomienda que no se haga uso de estos formatos por estar obsoletos.

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.

Los formatos CMS y XMLDSig se incluyen en el MiniApplet @firma con el único objetivo de proporcionar compatibilidad con aplicaciones que hiciesen uso de estos formatos. Su uso está totalmente desaconsejado y no se proporciona soporte sobre estos formatos.

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

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

La norma general sigue este simple algoritmo:

1. Si el navegador Web es Mozilla Firefox, con independencia del sistema operativo, se usa el almacén propio de Mozilla Firefox (NSS, *Netscape Security Services*) más los módulos PKCS#11 (*Public Key Cryptography Specification number 11*) que Firefox tuviese configurados, como tarjetas inteligentes, DNIe (Documento Nacional de Identidad electrónico), HSM (*Hardware Security Module*), etc.
2. Si el navegador es cualquier otro (Internet Explorer, Opera, Chrome o Safari), se usa el almacén predeterminado del sistema operativo:
    a. Windows: CAPI (*Cryptography Application Programming Interface*)
    b. Apple macOS: Llavero de macOS
    c. Linux: Almacén compartido 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).
- Almacén NSS de Mozilla (requiere que esté instalado Mozilla Firefox).
- PKCS11 (*Public Key Cryptography Specification number 11*).

En sistemas Windows, puede darse el caso de que el usuario utilice un perfil temporal, con lo que el usuario no contará con certificados ni tarjetas instaladas en el almacén de Windows. Cuando el Cliente @firma detecte este caso, hará uso del driver Java de DNIe para acceder al DNIe en caso de estár insertado en un lector del equipo. Además, buscará en el sistema una serie predeterminada de bibliotecas PKCS#11 y tratará de utilizar las tarjetas inteligentes insertadas y asociadas a estas bibliotecas.

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.

3.4.1 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 puede realizar durante la carga del MiniApplet/AutoFirma o en tiempo de ejecución.

- Para configurar el uso de un almacén durante el proceso de carga pasaremos el tipo de almacén como segundo parámetro del método utilizado: `cargarMiniApplet(String, String)` o `cargarAppAfirma(String, String)`. Consulte la sección de despliegue del MiniApplet para mayor información sobre estos métodos.
- Para configurar el uso de un almacén durante la ejecución del Applet o AutoFirma se utilizará el método `setKeyStore(String)`. En caso de ser AutoFirma el cliente en ejecución y estar usándose la comunicación por sockets, se reiniciará la comunicación para asegurar que las siguientes operaciones usan el almacén proporcionado.

El tipo de almacén 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_SHARED_NSS`
  - Almacén NSS del sistema. Compatible únicamente con sistemas Linux.
- `KEYSTORE_MOZILLA`
  - Almacén NSS de Mozilla (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 (tarjetas 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 un 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.

ADVERTENCIA: Los almacenes que hacen uso de un fichero o biblioteca requieren una contraseña de acceso. Esta contraseña se preguntará directamente al usuario cuando se requiera el acceso al almacén.

ADVERTENCIA: El acceso al almacén de Mozilla desde Windows, cuando no se realiza desde el propio navegador, puede requerir la instalación de algún entorno de ejecución de Visual C++. Si se quiere trabajar con este caso de uso, consulte los requisitos de Mozilla Firefox en el apartado "2.1.1 MiniApplet Cliente @firma".

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_MOZILLA);`
- `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.

3.4.2 Uso de tarjetas inteligentes

El MiniApplet tiene acceso a las claves de las tarjetas inteligentes a partir de sus controladores cuando están instalados en el sistema. Para utilizar los certificados en tarjeta en las operaciones de firma, se puede acceder a ellos desde un almacén PKCS#11 configurado o desde cualquiera de los almacenes de sistema disponibles:

- `KEYSTORE_WINDOWS`: El almacén de Windows carga automáticamente todas las tarjetas insertadas para las que se haya instalado su controlador CSP o el MiniDriver correspondiente de Windows Update. En caso de detectarse un DNIe insertado o una tarjeta CERES, se hará uso de los mismos a partir de un controlador interno de la aplicación para corregir problemas detectados con Java en los controladores oficiales.

- `KEYSTORE_MOZILLA/ KEYSTORE_SHARED_NSS`: Los almacenes de Mozilla se componen de un almacén interno y el conjunto de controladores PKCS#11 de las tarjetas instaladas en el sistema. El MiniApplet cargará automáticamente el almacén interno y todos los dispositivos detectados. Debido a problemas detectados con Java y los controladores oficiales de DNIe y tarjetas CERES, en caso de detectarse insertada cualquiera de estas tarjetas se utilizará un controlador interno de la aplicación, ignorándose los PKCS#11 configurados en el almacén de Mozilla. En caso de detectar alguna de estas tarjetas también se ignorarán el resto de dispositivos insertados para evitar problemas entre los distintos controladores.

- `KEYSTORE_APPLE`: El llavero de OS X se compone de un almacén internos y el conjunto de controladores de las tarjetas insertadas. Debido a problemas detectados con Java y los controladores oficiales de DNIe y tarjetas CERES, en caso de detectarse insertada cualquiera de estas tarjetas se utilizará un controlador interno de la aplicación.

Por regla general, se considera que sólo debería haber una tarjeta inteligente insertada en el momento de firmar. En caso de encontrarse varias, se dará prioridad al DNIe y las tarjetas CERES. En dichos casos, es posible que los certificados del resto de tarjetas no aparezcan disponibles para firmar o den error durante la firma.

El uso de las tarjetas CERES de la FNMT y del DNIe se realiza a través de las bibliotecas de JMulticard, por lo que no es necesario tener instalados sus controladores en el equipo para poder firmar con ellas.

3.4.3 Uso del DNIe

El Cliente @firma utiliza la biblioteca JMulticard para permitir firmar con DNIe 2.0 y 3.0 sin necesidad de que los usuarios tengan instalados los controladores de la tarjeta. Esta biblioteca se utilizará siempre que se encuentre un DNIe insertado en un lector del equipo y se inserte su PIN en el diálogo de JMulticard.

AutoFirma solicita el PIN del DNIe antes de listar los certificados del almacén y de que el usuario indique qué certificado desea utilizarla para firmar. Este comportamiento emula el de los controladores PKCS#11 de las tarjetas en donde el PIN es necesario para listar los certificados contenidos por la tarjeta y sigue la lógica de que si un usuario ha insertado el DNIe en el lector es porque lo desea utilizar. Cuando el usuario inserta el PIN, se listan sus certificados y se abre el canal seguro con la tarjeta y, en el momento de firmar, se utiliza este canal seguro para realizar la operación de firma. A continuación, se cierra el canal seguro.

Las operaciones de firma realizadas posteriormente desde la misma instancia del MiniApplet/AutoFirma, solicitarán el PIN de la tarjeta sólo en el momento de realizar la firma, momento en el cual se volverá a abrir el canal seguro con la tarjeta.

Si se recargase el almacén por medio de la opción correspondiente del diálogo de selección de certificados, el controlador se reiniciaría y volvería a pedir el PIN de la tarjeta para listar los certificados.

El diálogo de solicitud de PIN de JMulticard mostrará una casilla para indicar que se desea recordar la contraseña de la tarjeta durante el resto de la sesión. Esta casilla funciona cuando se utiliza simultáneamente con la función "`setStickySignatory()`" y permite que, mientras el certificado de la tarjeta esté preseleccionado no se pida el PIN para realizar las subsiguientes operaciones. En el momento en el que se cierra la instancia del MiniApplet (cambio de página) o de AutoFirma (se utiliza comunicación por servidor intermedio, se cumple el tiempo máximo de inactividad en la comunicación por sockets, etc.), dejará de surtir efecto la configuración de "`setStickySignatory()`" y se volverá a pedir el PIN de la tarjeta.

En el caso de ejecutar el MiniApplet/Autofirma desde Internet Explorer o Chrome en Windows y cancelar el diálogo de PIN del DNIe de JMulticard, se cargará el almacén del sistema normalmente. Si se tiene instalado el controlador oficial del DNIe en el equipo esto puede implicar que los certificados del DNIe se listen también en el diálogo de selección de certificados ya que será el controlador oficial el que los cargue. En estos casos, también se usará el controlador oficial para realizar la firma.

3.4.4 Información específica al uso de claves con NSS (Firefox en Windows o cualquier navegador en Linux)

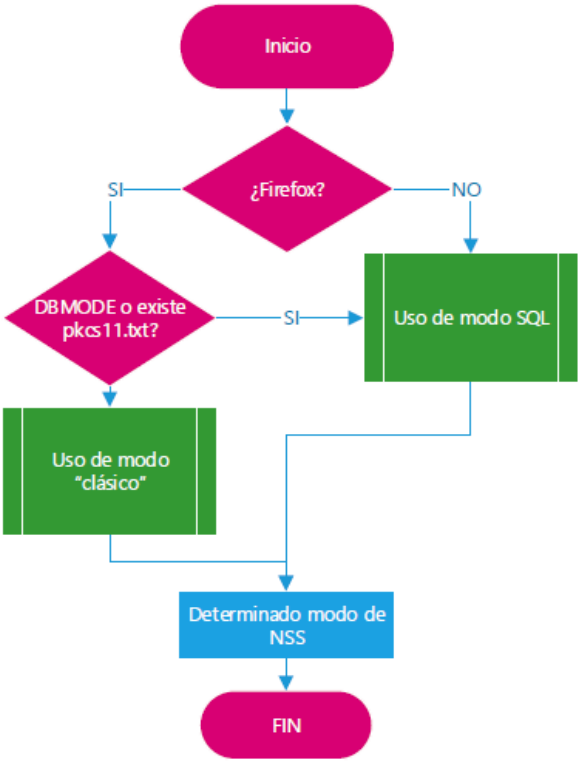
El uso de almacenes de claves de tipo NSS, usado por Mozilla Firefox en cualquier sistema operativo (Windows, macOS y Linux) y por todos los navegadores (Firefox, Chrome, etc.) en sistemas operativos Linux, presenta ciertas particularidades que conviene se tengan presentes.

3.4.4.1 Modo de funcionamiento de NSS (solo aplica a Linux)

En Linux, NSS puede almacenar internamente las claves en modo "clásico" o en bases de datos SQL (SQLite), según lo haya configurado el usuario o el administrador del sistema.

¿Cómo determina el MiniApplet o AutoFirma en qué modo debe usar estos almacenes de claves? Si el navegador NO es Firefox (Chrome, etc.), se usará siempre el modo SQL, y cuando el navegador es Firefox se comprobará si la variable de entorno de sistema operativo `NSS_DEFAULT_DB_TYPE` está establecida al valor "`sql`" o si en el perfil de usuario de NSS existe el fichero "`pkcs11.txt`", y si se cumple al menos una de estas dos condiciones, se usará el modo SQL.

Así, en sistemas operativos Linux se seguiría el siguiente flujo de trabajo:



Es importante tener esto en cuentra si se realizan cambios en la configuración de Firefox en Linux, ya que, si se pasa, por ejemplo, de modo SQL a modo "clásico" y no se elimina el fichero "`pkcs11.txt`" que existía mientras Firefox estuvo funcionando en modo SQL (no se borra automáticamente al cambiar el modo de Firefox), tanto el MiniApplet como AutoFirma intentarán usar el modo SQL por encontrar este archivo exsitente en el directorio de perfil, con independencia de si la variable de entorno `NSS_DEFAULT_DB_TYPE` está establecida o no o su valor.

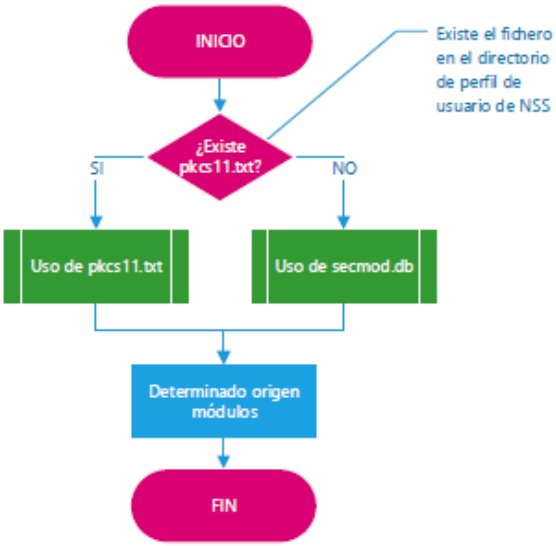
3.4.4.2 Origen de la lista de módulos PKCS#11 de NSS (solo aplica a Linux)

La forma de indicar en Firefox y NSS la lista de módulos PKCS#11 a usar (tarjetas inteligentes, etc.) varía según el modo de uso de NSS, teniendo:

- Modo "clásico".
  - Único posible en Firefox en Microsoft Windows o Apple macOS.
  - En Linux, solo Firefox puede usarlo (no otros navegadores).
  - La lista de módulos PKCS#11 a usar se guarda en el fichero "`secmod.db`".
- Modo SQL
  - Solo puede usarse en Linux (nunca en Windows o Apple macOS).
  - Único posible en los navegadores distintos a Firefox en Linux (Chrome, etc.).
  - La lista de módulos PKCS#11 a usar se guarda en el fichero "`pkcs11.txt`".

De forma coherente con lo expuesto en el punto "Modo de funcionamiento de NSS (solo aplica a Linux)", la norma es: Siempre que exista el fichero "`pkcs11.txt`" este debe ser usado como fuente de la lista de módulos PKCS#11, usándose "`secmod.db`" únicamente en otro caso (cuando no exista el fichero "`pkcs11.txt`").

Así, en sistemas operativos Linux se seguiría el siguiente flujo de trabajo:



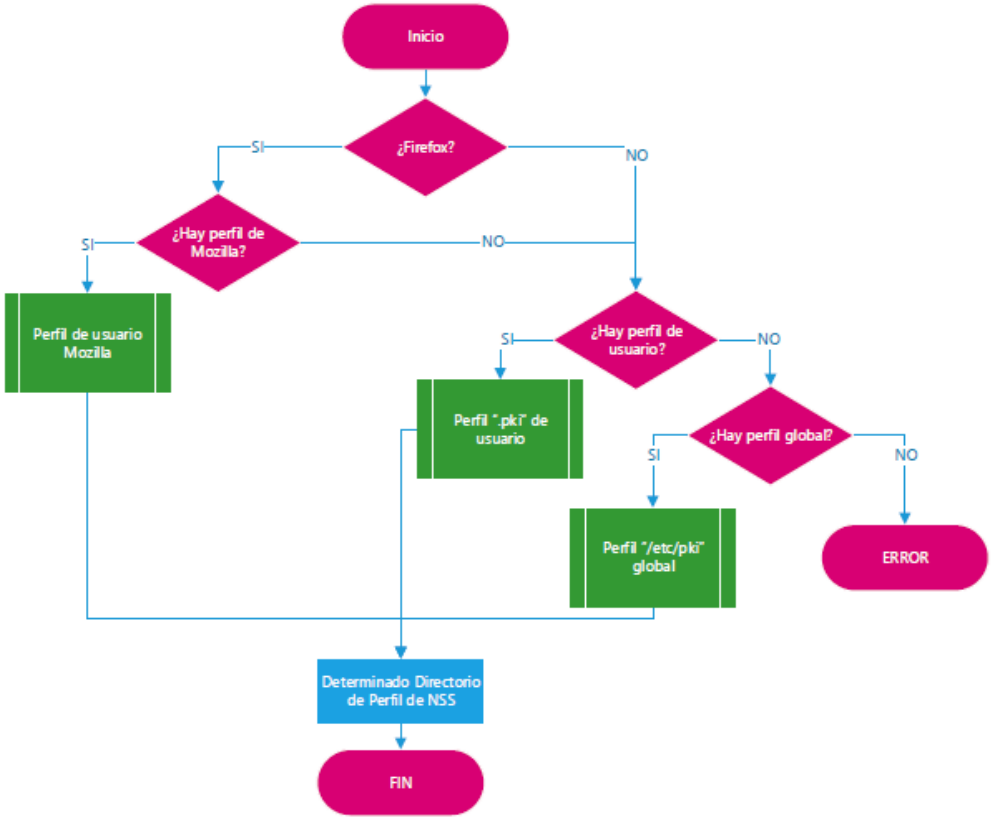
3.4.4.3 Directorio del perfil de usuario de NSS o Firefox

NSS (tanto con Firefox como con otros navegadores compatibles, como Chrome en Linux) necesita conocer el directorio donde se almacena la configuración del usuario.

¿Cómo determina el MiniApplet o AutoFirma ese directorio? En sistemas Windows o Apple macOS siempre se utiliza el directorio de perfil de usuario de Mozilla Firefox, ya que ese el es único navegador que usa NSS, pero en sistema operativo Linux, al usarse NSS por más navegadores (Firefox, Chrome, etc.), hay tres posibles alternativas:

1. Perfil de usuario de Firefox
    a. Solo lo usa el navegador Firefox, nunca otros, aunque sean compatibles con NSS.
2. Directorio de usuario de NSS ("`~/.pki`").
    a. Es el por defecto de los navegadores distintos a Firefox en Linux (Chrome, etc.), pero en configuraciones excepcionales también puede ser usado por Firefox.
    b. No se usa en sistemas operativos distintos a Linux (como Windows o macOS).
3. Directorio de sistema de NSS ("`/etc/pki`", compartido por todos los usuarios).
    a. Se usa únicamente cuando no existe ningún otro directorio de perfil de NSS.
    b. No se usa en sistemas operativos distintos a Linux (como Windows o macOS).

Así, en sistemas operativos Linux se seguiría el siguiente flujo de trabajo:



3.4.4.4 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.

3.4.4.5 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 (en base a su directorio de perfil de usuario de NSS / Firefox), 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 desease 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.

3.4.4.5.1 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/ior5ad1g.default-1435315801996
Default=1

[Profile1]
Name=default-1427127404614
IsRelative=1
Path=Profiles/ior5ad1g.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 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.

3.5 Envío anónimo de estadísticas por parte del MiniApplet y AutoFirma

Para poder mejorar la calidad del servicio, tanto el MiniApplet como AutoFirma envían información anónima de uso al servicio Google Analytics. Esta información contiene la dirección IP de la máquina que ejecuta la aplicación de firma.

Para evitar este envío de información, el integrador debe configurar como propiedad de sistema la variable `es.gob.afirma.doNotSendAnalytics` con el valor `true`.

Puede consultar como usar propiedades de sistema en el MiniApplet en el apartado Configuración a través de propiedades del sistema.

Adicionalmente, si el usuario configura una variable de entorno con mismo nombre y valor en su sistema operativo, tampoco se enviarán estas estadísticas de uso.

En el caso de AutoFirma, también puede desactivar el envío de estadísticas desde el menú de preferencias de la aplicación.

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 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 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. ElMiniApplet 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>
```

4.2.1 Entornos no compatibles con Applets y problemas durante la carga

Tanto si se solicita la carga del MiniApplet desde un entorno no compatible (entornos móviles, Google Chrome versión 42 o superior, Microsoft Edge, etc.), como cuando no es posible cargar el

Applet (falta de permisos, Java no instalado,…), el JavaScript de despliegue trata de delegar las operaciones que debía ejecutar el MiniApplet en una aplicación nativa de firma.

En el caso de entornos de escritorio, si el usuario tiene instalado AutoFirma, esta aplicación será lanzada para ejecutar las operaciones de firma que correspondan.

En el caso de un dispositivo Android o iOS, se utilizará la aplicación cliente @firma móvil correspondiente a ese sistema.

Puede encontrar el listado de funcionalidades del MiniApplet compatibles con AutoFirma y los clientes móviles en el apartado Compatibilidad con dispositivos móviles y AutoFirma.

Tenga en cuenta también que los clientes móviles y determinados entornos que ejecuten AutoFirma necesitan de servicios externos que posibiliten la comunicación entre la página Web y el cliente de firma. Puede encontrar información detallada sobre este punto en el apartado Servicios auxiliares de AutoFirma y el Cliente @firma móvil.

4.2.2 Carga directa de AutoFirma y los clientes móviles

Es posible que un integrador decida desechar directamente el MiniApplet @firma, de tal forma que se deriven todas las operaciones de firma en AutoFirma y los clientes @firma móviles. Para cargar directamente estas aplicaciones, utilice en lugar del método de carga descrito del MiniApplet el siguiente método de carga:

```
cargarAppAfirma(codebase, keystore)
```

Los parámetros `codebase` y `keystore` son exactamente iguales a los del método de carga del MiniApplet. Consulte los ejemplos anteriores para conocer su utilidad y modo de uso.

Un ejemplo del uso de este método es:

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

El parámetro `keystore` tendrá efecto cuando se cargue AutoFirma como herramienta para la realización de la firma, no cuando se utilice el cliente @firma móvil. Para conocer más detalles de la funcionalidad y uso de AutoFirma y los clientes @firma móviles, consulte el apartado Compatibilidad con dispositivos móviles y AutoFirma.

La comunicación entre la aplicación de firma y la página del trámite web se realizará cuando sea posible a través de un socket (entornos con AutoFirma y navegadores web que lo soporten) y si no a través de servicios externos. Es posible, sin embargo, que un integrador deseche el uso de la comunicación vía socket en pro de la comunicación mediante servidor intermedio, para así mantener la compatibilidad de versiones de AutoFirma que no la soportasen, por ejemplo. En ese

caso, puede configurar el modo de comunicación mediante el servicio intermedio llamando al método `setForceWSMode(boolean)` antes de invocar al método de carga.

Por ejemplo:

```
<script type="text/javascript">
      MiniApplet.setForceWSMode(true);
      MiniApplet.cargarMiniApplet("http://www.miweb.com/afirma");
</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 del integrador establecer estos textos en el idioma que corresponda en el momento de invocar a estos métodos:

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

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, checkURL)
```

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.
- `checkURL`
  - URL contra la que se realizara la petición para obtener la hora del servidor. Si no se indica este parámetro, se usará una página web aleatoria dentro del propio dominio.

Según la configuración de su servidor web, es posible que no se pueda determinar la hora a través de las peticiones realizadas. De ser así, no se mostrará nada al usuario para no perjudicar la realización del trámite. También es posible que su servidor sólo transmita la hora en la primera carga de la página y no cuando esta se refresca. Si ese fuese el caso, sólo se advertiría de la diferencia horaria en la primera carga de la página y no sí el usuario le da a refrescar la página desde su navegador.

Téngase en cuenta también que el desfase horario se calcula en el momento de invocar al método `checkTime`. Así pues, si el usuario modificase la hora de su sistema después de comprobación de invocar a este método, podría realizar operaciones de firma sin que se le mostrasen advertencias.

En caso de que desee bloquear de forma completa la firma de datos con una hora de sistema incorrecta, asegúrese de que el servidor de la página web de comprobación de hora siempre envía la hora en las respuestas de sus peticiones y llame al método `checkTime` antes de cada operación de firma.

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 y en servidores no estáticos

El MiniApplet 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.
- Servidores de Portlets.
- Etc.

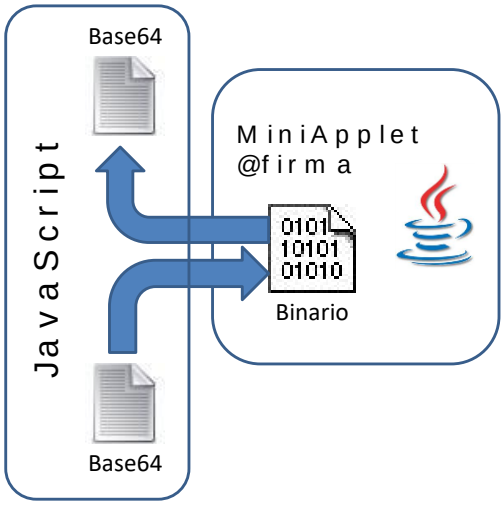
5 El proceso de firma electrónica

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 descodificación binaria de estos datos en Base64.



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

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

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

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

5.1.2 Selección de contenido desde ficheros en disco

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

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

Si se desea restringir el tipo de fichero a firmar, es posible indicar a los distintos métodos de firma una serie de parámetros adicionales entre los que encontramos:

i. Listado de extensiones de fichero.
   a. En caso de hacerlo, al usuario sólo le aparecerán por defecto en el diálogo de selección los ficheros cuya extensión esté entre las indicadas. Para configurar este comportamiento se utilizará el parámetro extra "`filenameExts`" y se le pasará como parámetro el listado de extensiones separadas por el carácter coma (','). Por ejemplo:
      - `filenameExts=pdf`
        - Sólo mostrará por defecto los ficheros con extensión ".pdf".
      - `filenameExts=pdf,swf`
        - Sólo mostrará por defecto los ficheros que tengan la extensión ".pdf" o ".swf"

- Incluso si se ha indicado este parámetro, el usuario podrá volver a ver todos los archivos seleccionar según si el diálogo de carga de ficheros su sistema operativo dispone de esta capacidad.

ii. Descripción textual del tipo de fichero a abrir.
   a. Mediante el parámetro "`filenameDescription`".

iii. Directorio por defecto donde buscar el fichero a abrir.
   a. Mediante el parámetro "`filenameCurrentDir`".

iv. Nombre propuesto para el fichero a abrir.
   a. Mediante el parámetro "`filenameActualName`".

Para saber más acerca de la configuración de parámetros adicionales de configuración, consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma.

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

El MiniApplet dispone de un método JavaScript para la descarga de datos remotos. Cuando los datos que se desean procesar se encuentran en una URL (HTTP o HTTPS), es posible hacer uso de este método de descarga para obtener esos datos codificados en Base64. De esta forma, pueden indicarse normalmente los datos descargados como entrada del método de firma del MiniApplet o cualquier otro dato que se reciba en Base64.

Este método tendrá las mismas restricciones de acceso que el JavaScript del navegador, 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).

El método en cuestión es `downloadRemoteData`. Consulte el apartado Descarga de datos remotos para más detalles.

5.1.4 Selección de objetivo de las contrafirmas

El caso de las contrafirmas es especial, ya 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.

5.2.1 Nota técnica: La cadena de confianza de un certificado

Los certificados forman normalmente parte de una cadena de confianza, en el que el certificado cuyo titular es usuario final ha sido a su vez firmado por otro certificado, el de una Autoridad de Certificación intermedia (CA), y este a su vez por una Autoridad de Certificación raíz (la autoridad intermedia puede no existir y que el certificado del usuario haya sido firmado directamente por la CA raíz o que haya varias autoridades intermedias encadenadas, ese aspecto depende de la arquitectura seleccionada por el emisor de los certificados).

Por ejemplo, un certificado de DNIe, usualmente tiene esta cadena:

CA Raíz DNIe.

    CA Intermedia DNIe.

        Certificado DNIe de ciudadano.

Tanto el MiniApplet como la aplicación AutoFirma incluyen en la firma, siempre que sea posible y no se indique lo contrario, la cadena completa de confianza del certificado usado para firmar, siguiendo de esta manera las recomendaciones en este sentido.

5.2.1.1 Algoritmos de huella digital para la firma de certificados y su relación con la cadena de confianza

Como se ha comentado anteriormente, los certificados van firmados por su antecesor en la cadena de confianza (excepto el certificado raíz, que es auto-firmado), y estas firmas, como cualquier otra firma, tienen sus algoritmos de huella digital.

Muchos certificados de CA Raíz o CA intermedias que originalmente fueron firmados mediante el algoritmo de huella digital SHA-1, han debido migrarse a SHA-2 (usualmente a SHA-256) debido a la obsolescencia e inseguridad de SHA-1, y en la mayoría de los casos se ha optado por refirmar exactamente el mismo certificado (con las mismas claves, pública y privada) solo que con distinto algoritmo de huella, en vez de sustituirlo por un certificado nuevo (con claves nuevas).

En estos casos, al tener un certificado raíz (o raíz intermedia) SHA-2 las mismas claves que su homólogo SHA-1, ambos forman una cadena de certificación correcta, pero solo uno de ellos puede aparecer en la cadena.

El Cliente @firma usa la cadena de certificados que devuelve el subsistema de gestión de claves del equipo del firmante (NSS en Firefox, MS-CAPI en Windows, etc.), por lo que no es posible forzar el uso de certificados SHA-2 en la cadena ni determinar cuál de los dos se usará en el caso de que ambos estén importados en este subsistema. Consulte con su fabricante (Mozilla en el Caso de Firefox, Microsoft en el caso de otro navegador en Windows, Apple en el caso de otro navegador en OS X, etc.) para obtener más información sobre las reglas de conformación de cadenas de confianza cuando hay disponibles varios certificados equivalentes pero con distinta algoritmia de huella digital.

Conviene resaltar un caso especial, que es el uso de tarjetas inteligentes en Mozilla Firefox (en cualquier sistema operativo). En este caso, la conformación de la cadena de confianza se delega en el controlador PKCS#11 (habitualmente suministrado por el fabricante de la tarjeta, pero puede ser de un fabricante independiente), y su corrección y contenido depende tanto de las funcionalidades de este como de si la propia tarjeta contiene o no la cadena completa de certificados para cada una de sus entradas. Consulte en este caso con el fabricante de su módulo PKCS#11 (FNMT-RCM para tarjetas CERES, Cuerpo Nacional de Policía en el caso de DNIe, etc.) para obtener más información.

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 firma, puede indicarse el formato “AUTO” para indicar que se realice la firma con un formato acorde al tipo de datos proporcionados. Consulte el apartado Selección automática del formato de firma para más información.

En el caso de la cofirma y contrafirma, puede utilizarse el formato “AUTO” para indicar que debe realizarse una firma con el mismo formato que la firma original.

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

5.4 Recogida de resultados

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

En el primer caso el MiniApplet @firma proporciona un método de utilidad (`saveDataToFile()`) para guardar resultados en disco. En este caso es siempre el usuario el que elige nombre del fichero

y directorio de destino por motivos de seguridad, y debemos acordarnos que aunque le proporcionemos los datos en Base64, lo que se almacenará en el fichero es la descodificación binaria de estos.

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.

Una alternativa a invocar la función de firma y posteriormente a la de guardado, es llamar al método `signAndSaveToFile()`, con el que se ejecutará la operación deseada y posteriormente se dará la opción al usuario de guardar el fichero. El resultado de la operación es el mismo que el que devolvería la llamada a la operación de firma correspondiente, incluso si el usuario decide no guardar los datos a disco.

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

6 Funciones 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
- Firmas/Multifirmas masivas
- Firma de lotes
- Firma/multifirma y guardado
- 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 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.

ADVERTENCIA: Su despliegue sólo será compatible con AutoFirma y los clientes móviles si utiliza el modo asíncrono para la obtención de resultados. Para conocer más detalles acerca de la compatibilidad de los despliegues del MiniApplet con las aplicaciones nativas, consulte el apartado Compatibilidad con dispositivos móviles y AutoFirma.

Los métodos de operación del MiniApplet 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 a ejecución de las operaciones de una forma secuencial, que viene a ser el modo común de uso, y facilita la migración desde el *Applet* Cliente @firma al MiniApplet.

Sin embargo, este modo no disfruta de las ventajas que se consiguen mediante el modo de operación asíncrono que se obtiene mediante el uso de *callbacks* (véase el apartado "6.2.2 Obtención de resultados mediante *Callbacks*"), que es además el que permite que nuestros desarrollos sea compatibles con AutoFirma y 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 AutoFirma y 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 qué 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 de firma, cofirma o contrafirma.
- Gestionar el error devuelto por una operación de 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 o por el usuario.

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

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

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

En esta función:

- `dataB64`: Datos en forma de cadena en Base64 que deseamos firmar. También puede no indicar la firma (usar `null`) para que se muestre al usuario un diálogo de carga de fichero. Si los datos que necesita firmar son un texto, cárguelo y conviértalo a base 64 tal como se indica en el apartado Conversión de un texto a cadena Base64.
- `algorithm`: Algoritmo de firma. Consulte el apartado Algoritmos de firma para conocer los algoritmos disponibles.

- `format`: Formato de firma. Consulte el apartado Formatos de firma soportados por el MiniApplet @firma para consultar aquellos disponibles.
- `params`: Parámetros adicionales para la configuración de la firma. Consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma para saber cómo realizar el paso de parámetros y el apartado de información específica del formato de firma que desee realizar para saber los parámetros soportados por el formato en cuestión.
- `successCallback`: Función JavaScript que se ejecutará una vez se obtenga el resultado de la operación de firma. Esta función recibirá como parámetros la firma resultante y el certificado utilizado en la operación. 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 para obtener más información.

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

Los clientes móviles no permiten identificar qué certificado se utilizó en la operación de firma, por lo que el parámetro del certificado no tomará ningún valor en la función *callback* cuando se utilice esta aplicación de firma.

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", "CAdES", null, successFunction,
        errorFuntion);
…
```

6.3.1.1.3 Firma electrónica de un texto introducido por el usuario:

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

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

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

6.3.1.1.4 Firma electrónica permitiendo al usuario seleccionar parámetros de firma:

```
…
  var params = "expPolicy=FirmaAGE";
  var modes = document.getElementsByName("rbMode");

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

  MiniApplet.sign(
       dataB64, "SHA1withRSA", "CAdES", params, successFunction, errorFunction
  );
…

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

6.4 Firmas electrónicas múltiples

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

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

6.4.1 Cofirmas

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

El resultado de esta operación se debe gestionar asíncronamente mediante funciones c*allback*. 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. Este valor sólo reproduce el formato, no las propiedades de las firmas originales. Por ejemplo, si cofirmásemos una firma XAdES-EPES, indicando "`AUTO`" como valor, agregaríamos a esta una cofirma XAdES B-Level / 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 parámetros la cofirma resultante y el certificado utilizado en la operación. 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 para más información.

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

Los clientes móviles no permiten identificar qué certificado se utilizó en la operación de firma, por lo que el parámetro del certificado no tomará ningún valor en la función *callback* cuando se utilice esta aplicación de firma.

6.4.1.1 Ejemplos:

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

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

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

6.4.1.1.2 Cofirma electrónica del resultado de una firma:

```
…
  // Funcion que realiza la cofirma a partir de los datos de firma
  function cosignFunction (signatureB64) {
     MiniApplet.coSign(
         signatureB64, dataB64, "SHA1withRSA", "XAdES", null, saveDataFunction,
         showError
     );
  }
  …
  // Función que almacena los datos generados por la cofirma en el campo
  // "resultId" de un formulario y lo envia
  function saveDataFunction (cosignB64, certificateB64) {
       document.getElementById("resultId").value = cosignB64;
       document.getElementById("firmante").value = certificateB64;
```

```
        document.getElementById("formulario").submmit();
    }
    …
    // Función para firmar datos. Si termina correctamente la operación de firma se
    // llama a la función "cosignFunction" con el resultado de la operación y, si
    // ésta 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 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. 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.

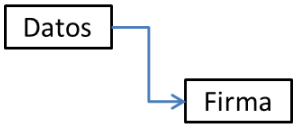
- *`params`*: Parámetros adicionales para la configuración de la contrafirma. Si se introduce un nulo, se usará la configuración por defecto para el formato de firma establecido. Consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma para saber cómo realizar 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 parámetros la contrafirma resultante y el certificado utilizado en la operación. 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 para más información.

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

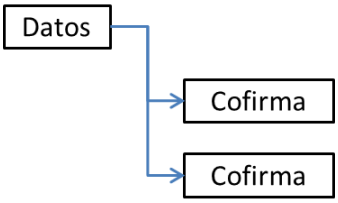
Los clientes móviles no permiten identificar qué certificado se utilizó en la operación de firma, por lo que el parámetro del certificado no tomará ningún valor en la función *callback* cuando se utilice esta aplicación de firma.

6.4.2.1 Creación del árbol de firmas

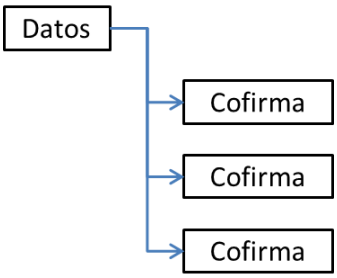
La operación de contrafirma se realiza sobre otra firma. Esta puede ser una firma simple, una cofirma u otra contrafirma. Estas operaciones de firma, cofirma y contrafirma van agregando firmas a un documento y, ya que las contrafirmas se realizan sobre firmas previas, se forma lo que se ha dado en llamar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



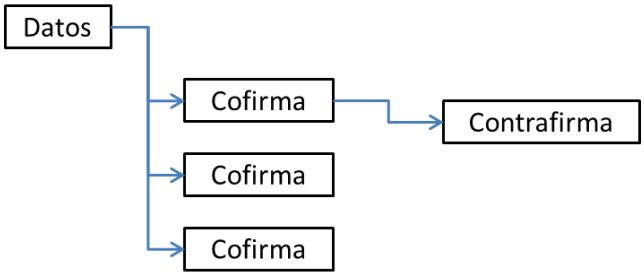
Si realizásemos una cofirma sobre el resultado de la operación anterior obtendríamos lo siguiente:



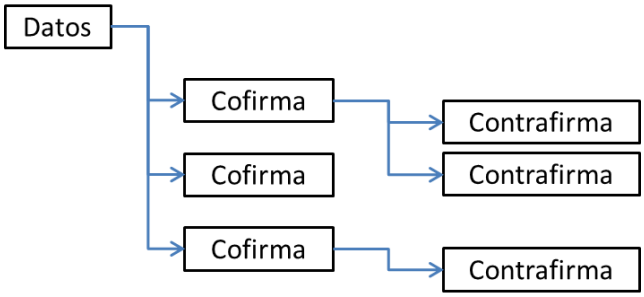
Tenga en cuenta que una firma y una cofirma están al mismo nivel y son equivalentes. No tiene importancia cual fue la primera en realizarse (firma) y cual la siguiente o siguientes porque todas son cofirmas. Las cofirmas son firmas sobre los datos originales, por lo tanto todas dependen de estos. Así,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

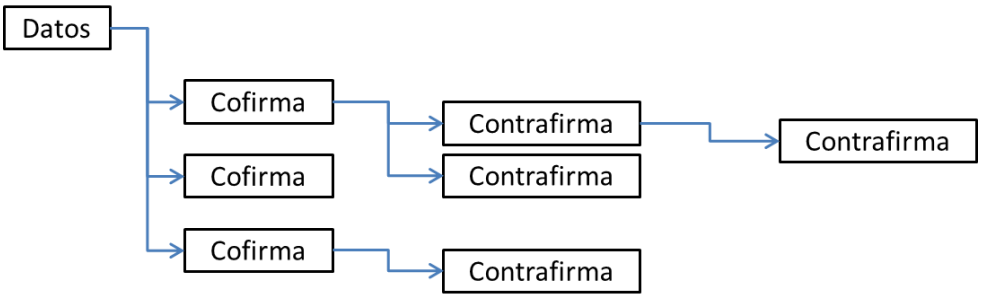


Una persona que contrafirme, puede estar interesada en contrafirmar más de una firma simultáneamente. Este sería el caso de un notario que valida con la suya las firmas de las dos partes de un contrato. Esto se representa con una firma sobre cada una de las firmas que se contrafirmen, que no tienen por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



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

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



6.4.2.2 Selección de los nodos que se desean contrafirmar

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

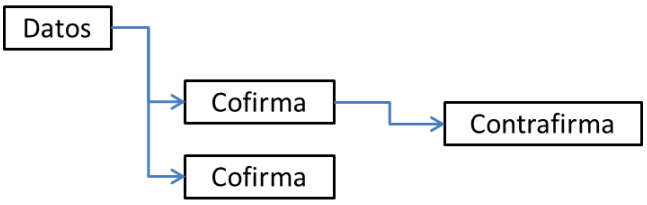
- **Nodos hoja del árbol** (`LEAFS`): Se firmarán sólo las firmas del árbol de los que no depende ningún otro nodo.
- **Todo el árbol de firma** (`TREE`): Se firman todos los nodos de firma del árbol.

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

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

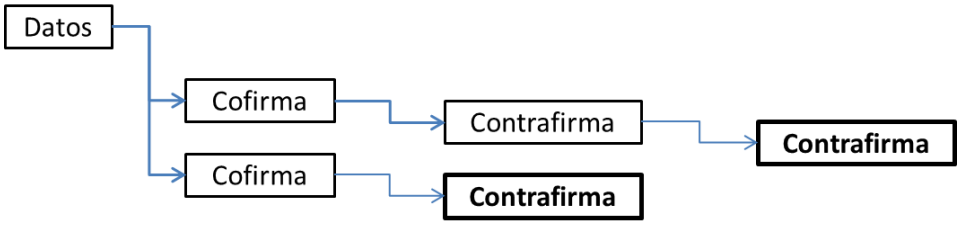
- `leafs`: Contrafirma todas las firmas que sean nodos hoja del árbol. Este es el valor por defecto.
- `tree`: Contrafirma todas las firmas del árbol.

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

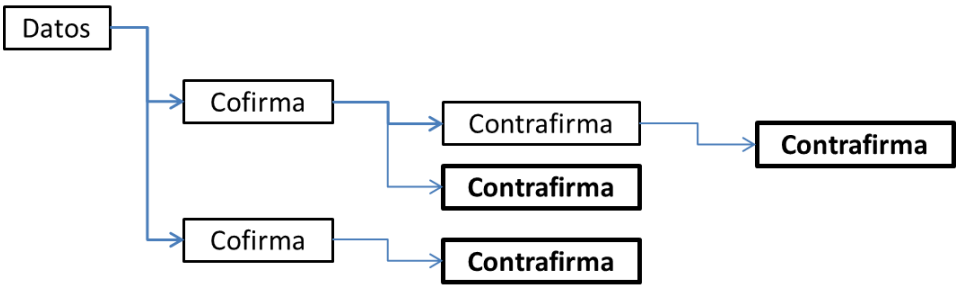


Cada una de las configuraciones dará el siguiente resultado:

- `target=leafs`



- `target=tree`



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

6.4.2.3 Ejemplos:

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

6.4.2.3.1 Contrafirma electrónica de una firma seleccionada por el usuario:

```
var filenameDataB64;
try {
    MiniApplet.getFileNameContentBase64(
```

```
            "Fichero de firma", "xsig", "Firma XAdES", null, counterSignCallback,
             errorCallback
        );
    } catch (e) {
        return;
    }

…
    // Advertencia: Esta operación no funcionara en AutoFirma en Chrome con
    // comunicación por servidor intermedio sin que que el usuario intervenga
    // entre ambas operaciones
    function counterSignCallback (filename, signatureB64) {
        MiniApplet.counterSign(
                signatureB64, "SHA1withRSA", "XAdES", null, successCallback,
                errorCallback
        );
    }
…
```

6.4.2.3.2 Contrafirma electrónica del resultado de una firma:

```
…
    // Advertencia: Esta operación no funcionara en AutoFirma en Chrome con
    // comunicación por servidor intermedio sin que que el usuario intervenga
    // 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", "CAdES", "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 sea menor.
  - Requiere el despliegue de servicios en el servidor.

Como nota importante, debe recalcarse 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`, `FacturaEtri`, `PAdEStri` y `XAdEStri`, respectivamente, como identificador del formato.

- **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",
       successCalback, errorCallback
  );
…
```

6.5.1.1.2 Cofirma trifásica CAdES:

```
…
  MiniApplet.coSign(
       signB64, dataB64, "SHA512withRSA", "CAdEStri",
       "serverUrl=http://miweb.com/TriPhaseSignerServer/SignatureService",
       successCalback, 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, FacturaE y PAdES (firmas y multifirmas) y admite las mismas opciones de configuración que las firmas monofásicas de estos formatos.

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.

Advertencia: Algunos servidores de aplicaciones y entornos 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.

6.5.2.1 Configuración del servicio de firma trifásica

Tal y como se distribuye, el servicio de firma trifásica no necesita configuración para permitir generar firmas trifásicas de la forma normal del MiniApplet @firma y AutoFirma. 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 un fichero "`tpsconfig.properties`". Este fichero puede situarse en un directorio del servidor y trasladarse la ruta al servicio por medio de la variable de entorno "`clienteafirma.config.path`". Si se proporciona esta variable al arrancar el servidor de aplicaciones y se le configura la ruta de un directorio, se buscará el fichero en el directorio. Si no, lo buscará en el classpath del despliegue.

Desde este fichero es posible configurar las siguientes propiedades:

- *Access-Control-Allow-Origin*
  - Permite establecer el origen permitido de las peticiones. El servicio de firma trifásica agregará el valor de esta propiedad en las respuestas del servicio.
  - Si se establece como valor un asterisco ('`*`'), se indica que se pueden realizar peticiones desde cualquier dominio.
  - Valor por defecto: `*`
- *alternative.xmldsig*
  - Permite habilitar el modo de compatibilidad con bibliotecas de firma XML que puedan encontrarse en el *classpath* del servidor de aplicaciones. Este tipo de bibliotecas pueden interferir con las que incluye el propio Oracle Java e impedir realizar firmas XAdES. Ejemplos de bibliotecas que provocan estos errores son XERCES/XALAN.
  - Al indicar el valor `true`, se habilitará el modo de compatibilidad con estas bibliotecas, lo que obligará al servicio a buscar diversos paquetes de clases para encontrar un modo de completar las firmas.
  - No se garantiza la compatibilidad con todas las versiones de estas bibliotecas.
  - Valor por defecto: `false`
- *document.manager*
  - Clase que se encargará de gestionar los documentos que se deben firmar y las firmas generadas.
  - El *Document Manager* por defecto imita un proceso de firma monofásica.
  - Consulte el apartado 6.5.2.2 Configuración del Document Manager del servicio para más detalles.

6.5.2.2 Configuración del Document Manager del servicio

Integrados en el servicio trifásico se distribuyen, a modo de ejemplo, dos *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.).

El *Document Manager* "`es.gob.afirma.triphase.server.SelfishDocumentManager`", es necesario que esté disponible en el sistema para las funcionalidades de firma móvil en los casos donde no existe una funcionalidad monofásica pero esta se requiere por compatibilidad, como por ejemplo:

- Cliente @firma Android:
    - XAdES.
- Cliente @firma iOS:
    - XAdES, CAdES y PAdES.

En estos casos las aplicaciones móviles seleccionan automáticamente el modo trifásico y "`es.gob.afirma.triphase.server.SelfishDocumentManager`" como *Document Manager*, no siendo necesaria ninguna acción ni configuración por parte del integrador.

Así, si se dispone de este *Document Manager* en servidor, sería posible que un mismo sitio Web configurado para firmas monofásicas con el MiniApplet Cliente @firma o AutoFirma, funcionase sin problemas con dispositivos móviles.

En cualquier otro caso, el uso de las firmas trifásicas emulando firmas monofásicas está desaconsejado, ya que se hace un mal uso de los recursos de red.

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.

6.5.2.2.1 Notas específicas para configuración del ejemplo "FileSystemDocumentManager"

6.5.2.2.1.1 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:

`documento.pdf`	se indicaría como	`ZG9jdW1lbnRvLnBkZg==`
`firma.xsig`	se indicaría como	`ZmlybWEueHNpZw==`

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/password`"), 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" (":").

6.5.2.2.1.2 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 ruta 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 y es compatible con el MiniApplet y AutoFirma cuando se ejecuta mediante comunicación por sockets. No lo es con AutoFirma con comunicación por servidor intermedio ni con los clientes móviles.

Al usar esta función en un proceso de firma con tarjetas inteligentes, es posible que se pida el PIN de la tarjeta por cada operación de firma, según sea el comportamiento definido por el controlador de la propia tarjeta. En caso de usarse una tarjeta CERES o el DNIe, el controlador será JMulticard y este mostrará en el diálogo de selección de PIN una casilla que podrá seleccionarse para que la aplicación recuerde el PIN y no lo vuelva a solicitar mientras la opción configurada por "`setStickySignatory()`" esté configurada y no se reinicie la aplicación.

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 Firma por lotes predefinidos

El MiniApplet y AutoFirma incorporan una funcionalidad de firma de lotes de datos. Gracias a esta característica es posible definir externamente un lote de firma mediante un fichero XML. Las operaciones de firma definidas en el lote se ejecutarán de forma desatendida por el cliente una vez se seleccione el certificado de firma, con la concurrencia de los módulos servidores, y devolviéndose el resultado global de la ejecución del lote.

La función de firma de lotes es:

```
function signBatch (batchB64, batchPreSignerUrl, batchPostSignerUrl, params,
                    successCallback, errorCallback);
```

En esta función:

- *`batch64`*: Es el fichero de lote codificado en base 64. Para saber cómo construir el fichero de lote consulte el apartado Creación de los lotes.
- *`batchPreSignerUrl`*: URL del servicio de prefirma de lotes. Este servicio se despliega junto con el servicio de firma trifásica.
- *`batchPostSignerUrl`*: URL del servicio de prefirma de lotes. Este servicio se despliega junto con el servicio de firma trifásica.
- *`params`*: Parámetros para la configuración global de la operación. Permite definir los filtros para determinar qué certificados de firma pueden utilizarse. 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 específico Configuración del filtro de certificados para conocer los parámetros que se pueden establecer.
- *`successCallback`*: Función JavaScript que se debe ejecutar en caso de que la firma del lote finalice correctamente. Esta función recibirá como parámetro el XML resultado, en base64, con los resultados parciales de la operación.
- *`errorCallback`*: Función JavaScript que se debe ejecutar en caso de que la firma del lote finalice con errores. Esta función recibirá dos parámetros: el tipo de error producido y un texto descriptivo del error.

Para hacer uso de esta funcionalidad será necesario desplegar el servicio de firma trifásica. Consulte el apartado Servicio de firma trifásica para más información.

6.7.1 Configuración del servicio

Para la firma por lotes, además de la configuración del fichero del servicio de firma trifásica, requiere la configuración de un fichero adicional en servidor llamado "`signbatch.properties`". Este fichero es obligatorio y debe estar situado en la raíz del CLASSPATH. En caso de no encontrarlo, el sistema fallará en fase de despliegue (de la aplicación WAR JEE).

Este fichero se lee una única vez durante el despliegue de la aplicación, por lo que, en caso de cambiar los valores, debe redesplegarse todo el servicio.

El fichero debe contar con las siguientes propiedades de configuración:

- `allowedsources`
  - Debe contener, separadas por punto y coma ("`;`"), los orígenes de datos permitidos. Puede usarse un asterisco ("`*`") como comodín, pero únicamente al final de cada fuente de datos indicada.
  - El nombre especial de origen "`base64`" indica que se aceptarán directamente los mismos datos a firmar como Base64, explicitados como el nombre de su fuente de datos.
  - Este parámetro es obligatorio, si no se indica no se aceptará ninguna fuente de datos, y por lo tanto no se podrá realizar ninguna firma.
  - Se admiten las siguientes fuentes de datos:
    - Base64:
      - Indicada como "`base64`".
    - HTTP (como el acceso lo inicia el servidor, debe tenerse cuidado de no otorgar permisos a dominios o datos internos no deseados):
      - Por ejemplo: "`http://*`", "`http://atos.net/*`", "`http://atos.net/doc.pdf`", etc.
    - HTTPS (como el acceso lo inicia el servidor, debe tenerse cuidado de no otorgar permisos a dominios o datos internos no deseados):
      - Por ejemplo: "`https://*`", "`https://google.com/*`", "`https://127.1.2.44/doc.pdf`", etc.
    - FTP (como el acceso lo inicia el servidor, debe tenerse cuidado de no otorgar permisos a dominios o datos internos no deseados):
      - Por ejemplo: "`ftp://*`", "`ftp://ftp.atos.net/*`", "`ftp://ftp.atos.net/doc.pdf`", etc.
    - Fichero en disco (debe tenerse mucho cuidado de no otorgar permisos sobre ficheros de sistema o confidenciales):
      - Por ejemplo: "`file://C:\files\*`", "`file://c:\files\doc.pdf`", etc.
- `concurrentmode`
  - Es un valor opcional, puede tener dos valores:

        - `true`
            - Indica que se permite el proceso en paralelo de las entradas del lote.
        - `false`
            - El lote se procesará secuencialmente, solo un proceso simultáneo.
- `maxcurrentsigns`
    - Debe contener, en el caso de que se haya indicado el valor "`true`" para el parámetro "`concurrentmode`", el número máximo de firmas a procesar concurrentemente.
- `tmpdir`
    - Directorio a usar para los ficheros temporales. Si o se indica se usará el del sistema.

Un ejemplo de fichero de configuración podría ser:

```
maxcurrentsigns=10

tmpdir=C:/salida/temp

# Operacion concurrente (true) o en serie (false)
concurrentmode=false

# Fuentes de datos permitidas, separadas por
allowedsources=base64;file://*;http://*;https://*;ftp://*
```

6.7.2 Creación de los lotes

Los integradores que deseen usar esta funcionalidad deben crear (externamente, el MiniApplet no tiene ninguna función expresa para esto) el XML de definición del lote de firma, que debe seguir este esquema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="signbatch">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="singlesign" maxOccurs="unbounded" minOccurs="1">
           <xs:complexType>
             <xs:sequence>
               <xs:element type="xs:string" name="datasource"/>
               <xs:element name="format">
                 <xs:simpleType>
                   <xs:restriction base="xs:string">
                     <xs:enumeration value="XAdES"/>
                     <xs:enumeration value="CAdES"/>
                     <xs:enumeration value="PAdES"/>
                     <xs:enumeration value="FacturaE"/>
                   </xs:restriction>
                 </xs:simpleType>
               </xs:element>
               <xs:element name="suboperation">
                 <xs:simpleType>
                   <xs:restriction base="xs:string">
                     <xs:enumeration value="sign"/>
```

```
                    <xs:enumeration value="cosign"/>
                    <xs:enumeration value="countersign"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="extraparams">
                <xs:simpleType>
                 <xs:restriction  base="xs:base64Binary" />
                </xs:simpleType>
              </xs:element>
              <xs:element name="signsaver">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="class"/>
                    <xs:element name="config">
                      <xs:simpleType>
                        <xs:restriction  base="xs:base64Binary" />
                      </xs:simpleType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute type="xs:string" name="Id" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute type="xs:integer" name="concurrenttimeout" use="optional"/>
      <xs:attribute name="stoponerror" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="true"/>
            <xs:enumeration value="false"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="algorithm" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="SHA1withRSA"/>
            <xs:enumeration value="SHA256withRSA"/>
            <xs:enumeration value="SHA384withRSA"/>
            <xs:enumeration value="SHA512withRSA"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Un posible ejemplo de XML creado siguiendo este esquema podría ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<signbatch stoponerror="false" algorithm="SHA256withRSA" concurrenttimeout="30">
        <singlesign Id="7725374e-728d-4a33-9db9-3a4efea4cead">
                <datasource>http://google.com</datasource>
                <format>XAdES</format>
                <suboperation>sign</suboperation>
                <extraparams>
                        Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpTaWduYXR1cmVJZD03NzI1Mzc0ZS03M
jhkLTRhMzMtOWRiOS0zYTRlZmVhNGNlYWQNCg==
                </extraparams>
                <signsaver>
                        <class>es.gob.afirma.signers.batch.SignSaverFile</class>
                        <config>
```

```
                                Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpGaWxlTmFtZT1DXDpcXFVzZXJ
zXFx0b21hc1xcQXBwRGF0YVxcTG9jYWxcXFRlbXBcXEZJUk1BMS54bWwNCg==
                        </config>
                </signsaver>
        </singlesign>
        <singlesign Id="93d1531c-cd32-4c8e-8cc8-1f1cfe66f64a">
                <datasource>SG9sYSBNdW5kbw==</datasource>
                <format>CAdES</format>
                <suboperation>sign</suboperation>
                <extraparams>
                        Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpTaWduYXR1cmVJZD05M2QxNTMxYy1jZ
DMyLTRjOGUtOGNjOC0xZjFjZmU2NmY2NGENCg==
                </extraparams>
                <signsaver>
                        <class>es.gob.afirma.signers.batch.SignSaverFile</class>
                        <config>
                                Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpGaWxlTmFtZT1DXDpcXFVzZXJ
zXFx0b21hc1xcQXBwRGF0YVxcTG9jYWxcXFRlbXBcXEZJUk1BMi54bWwNCg==
                        </config>
                </signsaver>
        </singlesign>
</signbatch>
```


6.7.2.1 Componentes de un lote

6.7.2.1.1 Cabecera de definición de lote

En el ejemplo, es la línea "`<signbatch stoponerror="false" algorithm="SHA256withRSA" concurrenttimeout="30">`". Contiene los siguientes atributos configurables por el integrador:

1. "`stoponerror`"

   Cuando se establece a "`false`" se indica que el proceso debe continuar incluso si alguna de las firmas del lote no puede completarse, y cuando se establece a "`true`" el proceso se para en el momento en el que se produce el primer error.

   Por defecto, se considera que el valor es "`true`", se parará la ejecución tras el primer error.

2. "`algorithm`"

   Indica el algoritmo de firma a usar para todo el lote, y puede tener los siguientes valores:

   - SHA1withRSA (**No se recomienda su uso por obsoleto**)
   - SHA256withRSA
   - SHA384withRSA
   - SHA512withRSA

3. "`concurrenttimeout`"

   Número de segundos que deberá durar como máximo cada fase de una una operación de firma. Si se excediese este tiempo, se detendría y se consideraría que se produjo un

error. Este valor sólo aplica cuando la firma se realiza de forma concurrente. Por defecto, se esperarán 30 segundos.

6.7.2.1.2 Definición de cada firma dentro del lote

Dentro del elemento de definición de lote debemos incluir uno o varios elementos de tipo "`singlesign`" (por cada firma que queramos incluir en el lote debemos incluir un elemento de este tipo), que debe incluir en su origen un identificador único. Podemos observar que en el ejemplo tenemos dos cabeceras de definición de firma, ya que el lote contiene dos firmas. Observemos la cabecera de la primera firma:

"`<singlesign id="7725374e-728d-4a33-9db9-3a4efea4cead">`", que indica que es una firma dentro del lote identificada por la cadena "`7725374e-728d-4a33-9db9-3a4efea4cead`".

Cada una de las firmas dentro del lote puede ser configurada individualmente con los siguientes parámetros:

6.7.2.1.2.1 Origen de los datos a firmar

El origen de los datos debe indicarse dentro del elemento "`datasource`" del XML, por ejemplo: "`<datasource>http://google.com</datasource>`"

El origen de los datos a firmar puede indicarse (siempre que cumpla los permisos indicados en el fichero de configuración descrito en la sección "Configuración del servicio"):

- Con una URL. En este caso el servidor (nunca el cliente) descargará directamente los datos a firmar.
  - Se admite HTTP, HTTPS y FTP.
    - Si se usa SSL con certificado cliente es necesario que el certificado cliente esté instalado en el almacén de certificados personales del Entorno de Ejecución de Java (JRE).
  - En este caso, la URL debe devolver los datos finales a firmar, sin ninguna codificación intermedia.
    - Así, por ejemplo, para firmar un PDF que esté disponible en una URL debemos indicar directamente la URL que apunte a ese PDF, como: `https://forja-ctt.administracionelectronica.gob.es/webdav/users/soporte_afirma/public/@FirmaV5p0_ANEXO_PSC.pdf`
  - Si la URL apunta a una huella digital (para los modos de firma que lo soporten), esta debe estar igualmente en binario, como: `https://miweb.com/firmas/hash001.bin`
  - En el ejemplo, puede observarse que en la primera firma se indica que el origen de los datos es la URL "*`http://google.com`*".
- Indicando directamente los datos a firmar codificados en Base64. En este caso el servidor descodificará el Base64 para obtener los datos a firmar.

- Puede usarse también para indicar directamente una huella (en los modos de firma que lo soporten), teniéndose en este caso que indicar en Base64.
- En el ejemplo, puede observarse que en la primera firma se indica que el origen de los datos es el Base64 "*SG9sYSBNdW5kbw==*", que descodificado equivale a la cadena de texto "*Hola Mundo*".

6.7.2.1.2.2 Formato de firma

El formato de firma a utilizar debe indicarse dentro del elemento "`format`" del XML, por ejemplo "`<format>XAdES</format>`".

Se admiten los siguientes formatos:

- *XAdES : "XAdES"*
- *CAdES: "CAdES"*
- *PAdES: "PAdES"*
  - Con este formato solo se pueden firmar y cofirmar documentos PDF. No se permiten contrafirmas.
- *FacturaE: "FacturaE"*
  - Con este formato solo se permite la firma de facturas electrónicas. Ni cofirmas, ni contrafirmas.

6.7.2.1.3 Operación de firma

La operación concreta de firma a realizar debe indicarse dentro del elemento "`suboperation`" del XML, por ejemplo "`<suboperation>sign</suboperation>`".

Se admiten las siguientes operaciones:

- Firma: "`sign`"
- Cofirma: "`cosign`"
- Contrafirma: "`countersign`"

6.7.2.1.4 Parámetros adicionales para la firma

Los parámetros adicionales para el formato y la operación concreta de firma deben indicarse dentro del elemento "`extraparams`" del XML, por ejemplo "`<extraparams>bW9kZT1pbXBsaWNpdA0Kc2lnbmF0dXJlUHJvZHVjdGlvbkNpdHk9TWFkcmlk</extraparams>`".

Estos parámetros adicionales deben indicarse codificando su representación textual como Base64. Así, las siguientes propiedades (indicando cada parámetro en una línea de texto con el formato *nombre_parámetro=valor*):

```
mode=implicit

signatureProductionCity=Madrid
```

Para una representación visible del carácter “intro” se utilizará la partícula ‘`\n`’. Así, los parámetros anteriores quedaría como:

```
mode=implicit\nsignatureProductionCity=Madrid
```

Al codificar cadena a Base64, obtendríamos:

bW9kZT1pbXBsaWNpdA0Kc2lnbmF0dXJlUHJvZHVjdGlvbkNpdHk9TWFkcmlk

Si no se desea establecer parámetros adicionales debe dejarse el nodo vacío.

6.7.2.2 Configuración del guardado de la firma

El guardado de la firma una vez esta se completa es una tarea que realiza igualmente el servidor, utilizando para ello clases especiales de guardado que el integrador debe codificar según sus necesidades.

Las clases deben situarse en el CLASSPATH de la aplicación Web en el lado servidor (normalmente dentro de un fichero JAR). Consulte la documentación de su servidor de aplicaciones para realizar esta tarea.

Una forma muy cómoda de hacerlo es editando el WAR (es un fichero ZIP con la extensión cambiada) antes de desplegarlo, y añadiendo el JAR con las nuevas clases en el directorio `WEB-INF/lib`. Hay que tener cuidado de no alterar el resto de contenidos ni la estructura de carpetas antes de volver a comprimir el WAR (como ZIP).

Estas clases deben implementar el interfaz *SignSaver*, que tiene la siguiente forma:

```
package es.gob.afirma.signers.batch;

import java.io.IOException;
import java.util.Properties;

/** Interfaz para el guardado, almacenaje o envío de firmas una vez realizadas.
 * @author Tomás García-Merás. */
public interface SignSaver {

	/** Guarda una firma electrónica.
	 * @param sign Definición de la firma que se hizo.
	 * @param dataToSave Datos a guardar, resultado de la firma electrónica.
	 * @throws IOException Si hay problemas durante el proceso. */
	void saveSign(final SingleSign sign, final byte[] dataToSave) throws IOException;

	/** Deshace un guardado previo (para los modos transaccionales).
	 * @param sign Identificador de la firma a deshacer. */
	void rollback(final SingleSign sign);

	/** Configura cómo ha de guardarse la firma electrónica.
	 * cada implementación requerirá unas propiedades distintas dentro del
	 * objeto de propiedades.
	 * @param config Propiedades de configuración. */
	void init(final Properties config);

	/** Obtiene las propiedades de configuración.
	 * @return Propiedades de configuración. */
	Properties getConfig();
```

```
}
```

Como se puede observar, el objeto de guardado recibe sus parámetros de funcionamiento y configuración en un fichero de propiedades de Java en el método "`init`", que es invocado siempre inmediatamente después de ser instanciado.

La forma de indicar qué clase de guardado a usar y con qué configuración es mediante el nodo "`signsaver`", que contiene a su vez dos nodos:

- "`class`", con el nombre cualificado de la clase a usar.
- "`config`", con las propiedades de configuración codificadas en Base64.

Así, en el ejemplo tenemos el siguiente nodo:

```
<signsaver>
        <class>es.gob.afirma.signers.batch.SignSaverFile</class>
        <config>
        Iw0KI1RodSBBdWcgMjAgMTI6MTM6NDEgQ0VTVCAyMDE1DQpGaWxlTmFtZT1DXDpcXFVzZXJzXFx0b21hc1xcQXBwRGF0Y
VxcTG9jYWxcXFRlbXBcXEZJUk1BMi54bWwNCg==
        </config>
</signsaver>
```

Este indica que debe usarse la clase de guardado "*`es.gob.afirma.signers.batch.SignSaverFile`*" con la configuración "`Iw0KI1RodSBBdWcgMjAgMTI6MTM6NDEgQ0VTVCAyMDE1DQpGaWxlTmFtZT1DXDpcXFVzZXJzXFx0b21hc1xcQXBwRGF0YVxcTG9jYWxcXFRlbXBcXEZJUk1BMi54bWwNCg==`", que si la descodificamos vemos que contiene:

```
FileName=C:\\Users\\tomas\\AppData\\Local\\Temp\\FIRMA2.xml
```

Que es la configuración que necesita para guardar la firma (esta clase simplemente guarda la firma en el fichero que se le indique en la configuración.

Se incluyen con la implementación dos ejemplos de clases de guardado:

- Guardado a fichero:
  - Clase: `es.gob.afirma.signers.batch.SignSaverFile`
  - Parámetros de configuración:
    - `FileName:` Indica el fichero donde debe guardarse la firma (no crea directorios, estos deben existir y deben tenerse los permisos adecuados).
  - **Esta clase es un simple ejemplo, y no debe usarse directamente en producción, ya que presenta problemas de seguridad (podrían indicarse que se sobrescriba un fichero de sistema).**
    - Ver "Notas sobre el ejemplo SignSaverFile".
- Envío a una dirección HTTP POST:
  - Clase: `es.gob.afirma.signers.batch.SignSaverHttpPost`
  - Parámetros de configuración:
    - `PostUrl:`

- URL a la que enviar los datos vía HTTP POST. Puede ser tanto HTTP como HTTPS
  - `PostParamName`
    - Indica el parámetro de la petición POST donde deben adjuntarse los datos.
      - Estos se adjuntarán siempre en Base64 con alfabeto URL SAFE.
- Esta clase realiza una llamada HTTP POST a la dirección indicada, enviando como datos del POST un parámetro con el nombre indicado cuyo valor será la codificación Base64 del resultado de la firma. Es necesario entonces que el integrador provea este servicio HTTP POST para recibir y tratar la firma enviada (habitualmente un servicio Web).
  - La URL indicada debe ser accesible desde el servidor (que es quien realiza el envío).

6.7.2.2.1 Notas sobre el ejemplo SignSaverFile

La clase `es.gob.afirma.signers.batch.SignSaverFile` es, como se ha comentado anteriormente, únicamente un ejemplo de implementación el del interfaz `es.gob.afirma.signers.batch.SignSaver` que puede resultar de utilidad para depuración, pero que no debe ser utilizada nunca en entornos reales.

Para evitar su exposición accidental, se distribuye con la escritura a disco deshabilitada mediante una variable final:

```
/** El guardado real está deshabilitado por defecto, habilitar para usar esta clase
 * para depuración. No debe usarse para entornos reales, ya que no hay comprobaciones de
 * qué ficheros pueden sobrescribirse. */
private static final boolean DISABLED = true;
```

Si desea usarla para pruebas con escrituras reales en disco, debe modificar los fuentes de la clase cambiando el valor de dicha variable, volverla a compilar y sustituir en el proyecto la clase existente por la modificada:

```
private static final boolean DISABLED = false;
```

Se encuentre o no habilitada esta variable, el resultado de la operación siempre será "DONE_AND_SAVED" cuando la firma se genere correctamente, independientemente de que nunca se guarde en disco.

6.7.3 Respuesta a una ejecución de un lote

Cuando se termina de procesar un lote de firma, el cliente recibe como respuesta un XML codificado en Bas64 que describe como ha resultado el proceso.

Este XML es acorde al siguiente esquema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="signs">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="signresult" maxOccurs="unbounded" minOccurs="0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute type="xs:string" name="id" use="required"/>
              <xs:attribute type="xs:string" name="result" use="required"/>
              <xs:attribute type="xs:string" name="description" use="optional"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Un ejemplo de XML devuelto podría ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<signs>
        <signresult id="001-XAdES" result="DONE_AND_SAVED" description=""/>
        <signresult id="002-CAdES" result="DONE_AND_SAVED" description=""/>
        <signresult id="003-CAdES" result="DONE_AND_SAVED" description=""/>
        <signresult id="004-CAdES" result="DONE_AND_SAVED" description=""/>
</signs>
```

En él distinguimos un nodo "`signresult`" por cada una de las firmas del lote, con su correspondiente identificador.

Este puede tener tres atributos:

- "`id`": Identificador de la firma.
- "`result`": Resultado del proceso.
- "`description`": Descripción del resultado del proceso (opcional).

6.7.3.1 Tipos de resultados de la ejecución de un lote

El valor de resultado de una firma individual de un lote, es el estado en el que se quedó esta firma al finalizar el proceso del lote.

En el caso de que no se indique que el proceso se interrumpa cuando se detecte un error, todas las firmas aparecerán con el estado de éxito como resultado o con alguno de los resultados de error.

En el caso de que sí se indique que el proceso se interrumpa cuando se detecte un error y este se produzca, las firmas pueden quedar en alguno de los estados intermedios.

Los distintos estados por los que puede pasar una firma de un lote son:

- Estado de éxito

- DONE_AND_SAVED
  - La firma se generó y guardó correctamente.

- Estados de error
  - DONE_BUT_ERROR_SAVING
    - Error al guardar la firma.
  - ERROR_PRE
    - Error en la primera fase del proceso de firma trifásica.
  - ERROR_POST
    - Error en la tercera fase del proceso de firma trifásica.

- Estados intermedios
  - NOT_STARTED
    - La firma no se ha iniciado.
  - DONE_BUT_NOT_SAVED_YET
    - La firma se ha generado pero aún no se ha guardado.
  - DONE_BUT_SAVED_SKIPPED
    - La firma se generó correctamente pero no se guardará.
  - SKIPPED
    - No se realizará la firma.
  - SAVE_ROLLBACKED
    - La firma se guardó, pero se volvió a eliminar.

6.7.4 Descripción del modo transaccional de ejecución de los lotes

Cuando se indica que un lote debe pararse en caso de error (con el atributo "`stoponerror="true"`" en la cabecera del XML de definición), se activa un modo transaccional, que sigue el siguiente proceso:

- Las firmas son generadas íntegramente en servidor, pero no se guardan hasta que no se realizan todas las del lote. Si una generación fallase se interrumpe todo el proceso y se da por perdido.
- Una vez están generadas, se comienza el proceso de guardado de firmas (el orden del lote no es relevante, el programa puede guardarlas en un orden distinto). Si un guardado falla, se deshacen los guardados que sí se hubiesen completado adecuadamente, llamando para ello al método "`rollback(final SingleSign sign)`" de la clase de guardado (*SignSaver*) definida en el lote para cada firma.
  - Es responsabilidad del integrador implementar adecuadamente este método "`rollback(final SingleSign sign)`" en su clase de guardado.
    - En los ejemplos provistos de implementaciones de *SignSaver*, si se usa salvado en un archivo del sistema de ficheros, el "`rollback`" simplemente borra el fichero creado, pero si se usa el ejemplo de HTTP POST, el

"`rollback`" no deshace la operación (realmente no hace nada, ya que no hay forma de deshacer una llamada HTTP).

6.7.4.1 Tamaño máximo del XML de definición de lote

El XML de definición de lote no puede exceder en tamaño de 1.024KB. Si necesitase componer trabajos mayores, estos deberán fraccionarse en varios lotes. Recuerde que puede establecer como datos a firmar una URL, accesible desde el servidor de firma, desde la que obtener los datos.

6.8 Firma/multifirma y guardado del resultado

El integrador del sistema puede ordenar al MiniApplet y AutoFirma que ejecute una operación de firma, cofirma o contrafirma y a continuación ofrezca al usuario guardar la firma recién generada.

Para llevar a cabo esta operación el integrador deberá proporcionar los mismos parámetros que en la operación de firma, además del tipo de operación (firma, cofirma o contrafirma) y una sugerencia para el nombre del fichero que se guardará.

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

```
function signAndSaveToFile (operation, dataB64, algorithm, format, params, filename, successCallback, errorCallback);
```

En esta función:

- *`operation`*: Operación criptográfica que se solicita realizar. Esta puede ser '`sign`', '`cosign`' o '`countersign`' (firma, cofirma o contrafirma, respectivamente).
- *`dataB64`*: Datos o firma electrónica en forma de cadena en Base64 sobre la que deseamos operar. También se puede no indicar este parámetro (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 se desea cofirmar o contrafirmas una firma y no conoce el formato de firma utilizado para la firma original, indique el valor "`AUTO`" para especificar que se utilice el mismo formato. Por ejemplo, si contrafirmásemos una firma CAdES-EPES, indicando "`AUTO`" como valor, agregaríamos a esta una contrafirma CAdES-BES salvo que indicásemos a través del parámetro *`params`* la política de firma que queremos utilizar.
- *`params`*: Parámetros adicionales para la configuración de la contrafirma. Si se introduce un nulo, se usará la configuración por defecto para el formato de firma establecido. Consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma para saber cómo realizar el paso de parámetros y el apartado de información específica del formato de firma que desee realizar para saber los parámetros soportados por el formato en cuestión.

- `filename`: Nombre propuesto para el fichero generado. El usuario podrá cambiar este nombre, ya que tan sólo aparecerá como nombre por defecto en el diálogo de guardado de datos. Puede no indicarse un valor (usar `null`), en cuyo caso se usará el nombre "Firma" seguido de la extensión que corresponda al formato en cuestión.
- `successCallback`: Función JavaScript que se ejecutará una vez se obtenga el resultado de la operación. Esta función recibirá como parámetros la firma, cofirma o contrafirma resultante y el certificado utilizado en la operación. 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. 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 para más información.

El resultado de esta operación puede obtenerse directamente o gestionarme mediante las funciones *callback*. Este resultado se obtiene codificado en base64. El resultado que se obtendrá será el de la operación de firma ignorándose en cualquier caso el resultado del guardado de los datos.

Si se desea restringir el tipo de fichero a guardar, es posible indicar una serie de parámetros adicionales entre los que encontramos:

v. Listado de extensiones de fichero.
   a. En caso de hacerlo, al usuario sólo le aparecerán por defecto en el diálogo de guardado los ficheros cuya extensión esté entre las indicadas. Para configurar este comportamiento se utilizará el parámetro extra "*`filenameSaveExts`*" y se le pasará como parámetro el listado de extensiones separadas por el carácter coma (','). Por ejemplo:
      - *`filenameSaveExts=pdf`*
        - Sólo mostrará por defecto los ficheros con extensión ".pdf".
      - *`filenameSaveExts=pdf,swf`*
        - Sólo mostrará por defecto los ficheros que tengan la extensión ".pdf" o ".swf"

- Incluso si se ha indicado este parámetro, el usuario podrá volver a ver todos los archivos seleccionar según si el diálogo de carga de ficheros su sistema operativo dispone de esta capacidad.

vi. Descripción textual del tipo de fichero a guardar.
   a. Mediante el parámetro "*`filenameSaveDescription`*".

vii. Directorio por defecto donde guardar el fichero.

a. Mediante el parámetro "*filenameSaveCurrentDir*".

viii. Nombre propuesto para el fichero a guardar.

a. Mediante el parámetro "*filenameSaveActualName*".

Para saber más acerca de la configuración de parámetros adicionales de configuración, consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma.

6.8.1.1 Ejemplos:

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

6.8.1.1.1 Cofirma de una firma, guardado del resultado en disco y anexión del mismo a un formulario:

```
…
  function addToFormCallback (signB64, certB64) {
       document.getElementById('signature').value = signB64;
  }
…
  MiniApplet.signAndSaveToFile ('cosign', signB64, "SHA1withRSA", "CAdES", null,
       "MiCofirma.csig", addToFormCallback, errorCallback);
…
…

<!-- Fragmento HTML con un campo de texto en donde el usuario puede insertar el
texto que desea guardar -->
…
<form>
  <input name="sign" id="signature" type="hidden" />
…
</form>
…
```

6.8.1.1.2 Firma y guardado de un documento:

```
…

  MiniApplet.signAndSaveToFile('sign', null, "SHA1withRSA", "XAdES", "format=XAdES
       Enveloping", null, addToFormCallback, errorCallback);
…

<!-- 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: Al no indicarse datos de entrada se permitirá cargar un fichero de disco y cuando aparezca el diálogo de guardado, al no haberse establecido un nombre de fichero por defecto, se propondrá el mismo nombre del fichero cargado al que se le agrega la extensión del formato de firma.

6.9 Gestión de ficheros

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

6.9.1 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,
successCallback, errorCallback);
```

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 fichero que se propone para el guardado. 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.
- *sucessCallback*: Nombre de la función *callback* javascript que se ejecutará en caso de que el guardado de datos finalice correctamente. Esta función no recibe parámetros. Si se omite este parámetro, o se establece a `null`, no se ejecutara ninguna función al terminar la operación.
- *errorCallback*: Nombre de la función *callback* javascript que se ejecutará en caso de que el guardado de datos finalice con errores. Esta función recibe 2 parámetros: el tipo de error y el mensaje de error. Si se omite este parámetro, o se establece a `null`, no se ejecutara ninguna función al terminar la operación.

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.

6.9.1.1 Ejemplos:

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

6.9.1.1.1 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);
…
```

6.9.1.1.2 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.

6.9.2 Selección y recuperación de un fichero por parte del usuario

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 la carga de datos en disco es:

```
      function getFileNameContentBase64(title, extensions, description, filePath,
successCallback, errorCallback);
```

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.
- *successCallback*: Opcional. Función *callback* de éxito que se invocará cuando se seleccione un fichero.
- *errorCallback*: Opcional. Función *callback* de error que se invocará cuando no se seleccione un fichero o se produzca un error durante su carga.

El uso de los parámetros *callback* es necesario para la compatibilidad con AutoFirma mediante la comunicación con sockets.

Cuando se han indicado el parámetro con la función *callback* de éxito el resultado de la carga se recibe en esta mediante dos parámetros. El primero de ellos es el nombre del fichero y el segundo su contenido en base 64. En caso de haberse indicado el parámetro *callback* de error y producirse uno, se recibirá en un primer parámetro el tipo de error producido y en un segundo el mensaje de error asociado.

Cuando no se indican parámetros *callback*, este método devuelve el nombre del fichero seleccionado seguido del contenido del mismo en base 64, separados por el carácter “|”. Si el usuario cancelase el diálogo de selección de fichero devolvería *null* y, en caso de producirse un error durante la operación de carga o conversión a base 64, se lanzaría una excepción.

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

Advertencia: El uso de este método, aún con las funciones de callback, hace que el despliegue del MiniApplet no sea compatible con el Cliente Móvil ni AutoFirma en la comunicación por servidor intermedio. Como alternativa, 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.

6.9.2.1 Ejemplos:

6.9.2.1.1 Carga de un fichero y recogida de su nombre

```
…
  var fileName;
  var fileContentB64;
  MiniApplet.getFileNameContentBase64(
       “Seleccionar fichero”, “jpg,gif,png”, “Imagen”, null,
```

```
        loadDataSuccessCallback, loadDataErrorCallback
        );
…
  function loadDataSuccessCallback(name, contentB64) {
        filename = name;
        fileContentB64 = contentB64;
  }
  function loadDataErrorCallback(errorType, errorMsg) {
        alert("Error: " + errorMsg);
  }
…
```

6.9.2.1.2 Carga de un fichero y recogida de su nombre sin funciones callback (Sólo válido para el MiniApplet)

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

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

6.9.2.1.3 Carga de un fichero con ruta preseleccionada

```
…
  var fileName;
  var fileContentB64;
  MiniApplet.getFileNameContentBase64(
        "Seleccionar fichero", "jpg,gif,png", "Imagen", "c:/pruebas/Entrada.png",
        loadDataSuccessCallback, loadDataErrorCallback
        );
…
  function loadDataSuccessCallback(name, contentB64) {
        filename = name;
        fileContentB64 = contentB64;
  }
  function loadDataErrorCallback(errorType, errorMsg) {
        alert("Error: " + errorMsg);
  }
…
```

6.9.3 Selección y recuperación de múltiples ficheros por parte del usuario

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,
        successCallback, errorCallback);
```

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.
- *successCallback*: Opcional. Función *callback* de éxito que se invocará cuando se seleccione un fichero.
- *errorCallback*: Opcional. Función *callback* de error que se invocará cuando no se seleccione un fichero o se produzca un error durante su carga.

El uso de los parámetros *callback* es necesario para la compatibilidad con AutoFirma mediante la comunicación con sockets.

Cuando se han indicado el parámetro con la función *callback* de éxito el resultado de la carga se recibe en esta mediante dos parámetros. El primero de ellos es un *array* con los nombres de los ficheros y el segundo un *array* con sus contenidos correspondientes en base 64. Se reciben de tal forma que el nombre del primer elemento del array de nombres se corresponderá con el contenido del primer elemento del segundo parámetro y así sucesivamente. En caso de haberse indicado el parámetro *callback* de error y producirse uno, se recibirá en un primer parámetro el tipo de error producido y en un segundo el mensaje de error asociado.

Cuando no se indican parámetros *callback*, 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, aún con las funciones de callback, hace que el despliegue del MiniApplet no sea compatible con el Cliente Móvil ni AutoFirma en la comunicación por servidor

intermedio. Como alternativa, para la firma de multiples ficheros simultáneamente prepare su aplicación para integrar un proceso de firma de lote.

6.9.3.1 Ejemplos:

6.9.3.1.1 Carga de ficheros y recogida de sus nombres

```
…
  var fileNames;
  var fileContentB64s;
  MiniApplet. getMultiFileNameContentBase64(
       "Seleccionar fichero", "jpg,gif,png", "Imagen", null,
       loadDataSuccessCallback, loadDataErrorCallback
       );
…
  function loadDataSuccessCallback(namesArray, contentB64sArray) {
       fileNames = namesArray;
       fileContentB64s = contentB64sArray;
  }
  function loadDataErrorCallback(errorType, errorMsg) {
       /* Si el error no se debe a la cancelación por el usuario, lo mostramos. */
       if (!"es.gob.afirma.core.AOCancelledOperationException".equals(errorType)) {
              /* Método del integrador para mostrar logs */
              showLog(errorMsg);
       }
  }
…
```

6.9.3.1.2 Carga de ficheros y recogida de sus nombres sin funciones callback

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

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

6.10 Utilidad

6.10.1 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
```

6.10.2 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 Gestión de errores apartado para más información.

6.10.3 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.

6.10.3.1 Ejemplos

6.10.3.1.1 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);
…
```

6.10.4 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.

6.10.4.1 Ejemplos

6.10.4.1.1 Firma de un texto insertado por el usuario

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

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

6.10.5 Descarga de datos remotos

El MiniApplet @firma proporciona un método para la descarga de datos remotos. Este método accede a una URL establecida por el integrador, descarga el contenido que en ella se encuentra e invoca a un método *callback* al que le proporciona los datos descargados en Base64.

La descarga de los datos se realiza desde JavaScript y sufre las limitaciones que aplican los navegadores a este tipo de descargar. Por norma, sólo podrán descargarse datos accesibles desde una URL situada en el mismo dominio y mismo esquema (HTTP o HTTPS) que la web que integra el JavaScript de despliegue.

La descarga de los datos se realiza por medio del método GET de HTTP y se realiza de forma asíncrona. Es decir, se descargan los datos en hilo de ejecución distinto al proceso principal. Una vez termina el proceso de descarga, se invoca a un método *callback* establecido por el integrador, de tal forma que así recuperar el control de la operación.

La función JavaScript para descarga de datos remotos es:

```
function downloadRemoteData(url, successCallback, errorCallback);
```

En esta función:

- *url*: URL de acceso a los datos a descargar.
- *successCallback*: Nombre de la función que se invocará en caso de finalizar correctamente la descarga de los datos. Esta función recibirá como único parámetro los datos descargados en Base64.

- *errorCallback*: Nombre de la función que se invocará en caso de finalizar con errores la descarga de los datos. Esta función recibirá como único parámetro el error que originó el problema.

Este método devuelve no devuelve nada y su ejecución es asíncrona.

6.10.5.1 Ejemplos

6.10.5.1.1 Descarga de datos y posterior firma

…

```
var mostrarError = function(e) {
     alert("Error en la descarga de los datos: " + e);
}

var iniciarFirma = function(datosB64) {
     MiniApplet.sign(
            datosB64,
            algorithm,
            format,
            null,
            successCallback,
            errorCallback);
}


var url = "http://midominio.com/informe.pdf";

MiniApplet.downloadRemoteData (url, iniciarFirma, mostrarError);
```

…

6.10.6 Selección de certificado

El MiniApplet puede permitir al usuario seleccionar un certificado sin necesidad de realizar una operación de firma. Para esto, se utiliza el mismo diálogo de selección que en las operaciones de firma tradicionales.

La utilidad de este método es conocer de antemano el certificado que el usuario quiere seleccionar para la firma, de tal forma que puede realizar validaciones previas sobre el certificado y/o mostrar información de este en su aplicación web.

La función JavaScript para permitir seleccionar un certificado de usuario es:

```
function selectCertificate(params, successCallback, errorCallback);
```

En esta función:

- *params*: Parámetros de configuración de la operación que afecten a la selección de certificados. Más concretamente, se pueden configurar filtros de certificados, como se indica en el apartado Configuración del filtro de certificados, y la selección automática de certificado, como se describe en el apartado Selección automática de certificados.

- *successCallback*: Opcional. Nombre de la función que se invocará en caso de finalizar correctamente la selección de certificado. Esta función recibirá como único parámetro el certificado seleccionado en Base64.
- *errorCallback*: Opcional. Nombre de la función que se invocará en caso de finalizar con errores la descarga de los datos. Esta función recibirá como parámetros, el tipo de error que se produjo y el mensaje de error asociado.

Este método permite recuperar directamente el certificado seleccionado como resultado de la función o recuperarlo a través de las funciones *callback*, si se establecen. Para mantener la compatibilidad con las aplicaciones nativas será necesario hacerlo siempre a través de las funciones *callback*.

El método de selección de certificado solicitará la contraseña de los almacenes utilizados si es necesario para poder listar los certificados que contienen, pero nunca utilizará las claves privadas de los certificados, así que en ningún caso pedirá las contraseñas para ello.

Este método también es compatible con la función `setStickySignatory(boolean)` (véase el apartado Firmas/Multifirmas masivas) con la cual es posible fijar el certificado seleccionado por el usuario. De esta forma se utilizará el mismo certificado durante las siguientes operaciones de firma o selección.

6.10.6.1 Ejemplos

6.10.6.1.1 Selección de certificado y envío a servidor para su análisis

…

```
var mostrarError = function(errorType, errorMessage) {
     alert("Error en la descarga de los datos: " + errorMessage);
}

var enviarCertificado = function(certB64) {
     document.getElementById("cert").value = certB64;
     document.getElementById("formulario").submit();
}

var extraParams = "filters.0=issuer.contains:FNMT\n" +
                  "filters.1=issuer.contains:Policia";

MiniApplet.selectCertificate (extraParams, enviarCertificado, mostrarError);
```

…

6.10.7 Obtención del log de la aplicación

El MiniApplet y AutoFirma (cuando se comunica mediante sockets con el navegador) permiten obtener las trazas de su propia ejecución para después operar con el con ellas como se desee.

La utilidad de este método es, como mecánica general ante los errores o como medio particular para la resolución de incidencias, poder recuperar la traza de ejecución del Cliente @firma para poder rastrear los motivos que llevaron a un error en la aplicación.

La función JavaScript para permitir seleccionar un certificado de usuario es:

```
function getCurrentLog(successCallback, errorCallback);
```

En esta función:

- *successCallback*: Opcional. Nombre de la función que se invocará en caso de recuperar correctamente la traza de ejecución. Esta función recibirá como único parámetro un texto plano con información del entorno de ejecución y la traza.
- *errorCallback*: Opcional. Nombre de la función que se invocará en caso de finalizar con errores la carga de la traza de ejecución. Esta función recibirá como parámetros, el tipo de error que se produjo y el mensaje de error asociado.

Este método permite recuperar directamente la traza de ejecución o recuperarla a través de las funciones *callback*, si se establecen. Para mantener la compatibilidad con las aplicaciones nativas será necesario hacerlo siempre a través de las funciones *callback*.

6.10.7.1.1 Obtención del log de la aplicación y envío a servidor para su análisis

```
…

  var mostrarError = function(errorType, errorMessage) {
       alert("Error al recuperar las trazas: " + errorMessage);
  }

  var enviarTrazas = function(trace) {
       document.getElementById("trz").value = MiniApplet.getBase64FromText(trace);
       document.getElementById("formulario").submit();
  }
…
  MiniApplet.getCurrentLog(enviarTrazas, mostrarError);
…
```

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, contrafirma y selección de certificados. 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 filtros 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 DNIe
  - `filters=issuer.rfc2254: (O=DIRECCION GENERAL DE LA POLICIA);keyusage.nonrepudiation:true;nonexpired:`
    - Certificados de firma del DNIe 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 una 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 sub-filtro 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 DNIe.

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 DNIe: Filtra los certificados del almacén para que sólo se muestren los certificados de firma de los DNIe disponibles desde ese almacén.
    - Para establecer este filtro de certificados se indicará la propiedad *filter* con el valor *dnie:* en el parámetro de configuración de la operación de firma, cofirma o contrafirma.
    - Ejemplo:
        - `filters=dnie:`
- 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 DNIe.
    - 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:
        - `filters=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 DNIe.
    - 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:
        - `filters=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 DNIe. 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:
  - `filters=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 DNIe, se mostrará en su lugar el certificado de firma de ese mismo DNIe.

  - **Advertencia**: Se aconseja el uso del filtro de certificados cualificados ("qualified:"), que actúa de igual manera, pero distingue entre los certificados de autenticación y firma de un mayor número de certificados.
  - 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:
    - `filters=ssl:45553a61`
    - `filters=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:
    - `filters=qualified:45553a61`
    - `filters=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:
    - `filters=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:
  - `filters=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.

- <u>Filtro RFC2254 en base al *Subject* del certificado:</u> 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:
    - `filters=subject.rfc2254:(CN=*12345678z*)`
      - Este filtro mostrará sólo aquellos certificados en los que aparezca la cadena "12345678z" en el *CommonName* de su *Subject*.
- <u>Filtro RFC2254 en base al *Issuer* del certificado:</u> 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:
    - `filters=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 DNIe y los de CERES.
  - Este filtro puede aplicarse de forma recursiva, de tal forma que permitirá el uso del certificado si cualquier de los certificados de la cadena de certificación por encima de él mismo cumple con la expresión indicada. Para utilizar recursivamente este filtro se usará el valor *issuer.rfc2254.recurse:* seguido de la expresión RFC2254.
  - Ejemplo:
    - `filters=issuer.rfc2254.recurse:(CN=*FNMT*)`
      - Este filtro mostrará sólo aquellos certificados en los que alguno de los certificados de su cadena de certificación tenga la partícula "FNMT" en el nombre común

- 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.
    - Ejemplos:
        - `filters=subject.contains:JUAN ESPAÑOL ESPAÑOL`
            - Este filtro mostrará sólo aquellos certificados en los que aparezca la cadena "JUAN ESPAÑOL ESPAÑOL" en el *Subject*.
        - `filters=subject.contains:11111111H`
            - Este filtro mostrará sólo aquellos certificados en los que aparezca la cadena "11111111H" en algún punto del *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:
        - `filters=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:
        - `filterss=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 DNIe.

- <u>Filtro por identificador de directiva:</u> Filtra los certificados por aquellos que poseen un identificador de directiva concreto. Esto es útil para mostrar sólo determinado tipo de certificados de una autoridad de certificación.
  - Para establecer este filtro se usará el valor *policyid:* seguido por listado de OIDs, separados por comas (‘,’), por los que se quieran filtrar.
  - Ejemplo:
    - `filters=policyid:1.3.6.1.4.1.18332.3.4.1.2.11`
      - Este filtro mostrará sólo aquellos certificados con el identificador de directiva “`1.3.6.1.4.1.18332.3.4.1.2.11`”.
- <u>Filtro de seudónimo:</u> Filtra los certificados, listando únicamente los certificados de pseudónimo y aquellos que no tienen un certificado de seudónimo asociado. Así, quedan ocultos los certificados que tienen un certificado equivalente de seudónimo, lo que evita que aparezca su nombre real en los datos de firma.
  - Para establecer este filtro se usará la palabra clave *pseudonym:*
  - Ejemplo:
    - `filters=pseudonym:`
- <u>Filtro de almacenes externos:</u> Permite deshabilitar el botón de carga de almacenes PKCS#12 en el diálogo de selección de certificados. De esta forma sólo podrán usarse los certificados del almacén seleccionado por el integrador o los por defecto del navegador en caso de que el integrador ni especificase ningún almacén.
  - Para establecer este filtro se usará la palabra clave *disableopeningexternalstores*
  - Ejemplo:
    - `filters=disableopeningexternalstores`
- <u>Filtro en base a certificado codificado:</u> Filtra los certificados para seleccionar uno concreto proporcionado a través del filtro. Esto es de utilidad cuando, después de una operación realizada con un certificado, se quiere restringir futuras operaciones para que se realicen con el mismo certificado.
  - Para establecer este filtro se usará el valor *encodedcert:* seguido del certificado codificado en base 64. Esto es, tal como se devuelve a trasvés del callback en los métodos de firma y selección de certificado.
  - Ejemplo:
    - `filters=encodedcert:MIIIcjCCBlqgAwIB……radvEjJ=`

- Este filtro mostrará sólo aquel certificado que hemos proporcionado en el filtro, en caso de que exista en el almacén.

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 (CAdES, PAdES y XAdES) tienen una variante EPES (*Explicit Poliyc-based Electronic Signature*) que declaran los atributos correspondientes a la política de firma.

El MiniApplet @firma permite la generación de firmas EPES (CAdES-EPES, PAdES-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 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 acordes 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 acordes 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 (CAdES, 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.

Los parámetros de esta política son los siguientes:

- CAdES
  - `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 CAdES y los datos ocupen menos de 1 MB. Los datos no se incluirán en la firma cuando su tamaño sea mayor, siguiendo, de forma general, la especificación establecida por la política de firma de la AGE:

> *En el caso de que, debido al tamaño de los datos a firmar, no resulte técnicamente posible o aconsejable realizar las firmas con el formato anteriormente descrito* (que la firma contenga los datos firmados)*, se generará la estructura de firma detached, que incluye el hash del documento original en la firma.*

Debido a la generalidad de la especificación, se permite al integrador definir qué tamaños de datos pueden incluirse dentro de la firma y cuáles no para su aplicación, por lo que, en caso de haberse indicado expresamente un modo de firma (parámetro `mode`), se utilizará el valor establecido por el integrador.

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.

Los parámetros de esta política son los siguientes:

- CAdES
  - `policyIdentifier=2.16.724.1.3.1.1.2.1.8`
  - `policyIdentifierHash=7SxX3erFuH31TvAw9LZ70N7p1vA=`
  - `policyIdentifierHashAlgorithm=1.3.14.3.2.26`
  - `policyQualifier=http://administracionelectronica.gob.es/es/ctt/politicafirma/politica_firma_AGE_v1_8.pdf`
  - `mode=` ** Ver nota **

Nota: Respecto al modo en CAdES (implícito o explícito, dado que la política indica que la firma debe ser implícita siempre que el tamaño del documento sea razonable. Como no se especifica exactamente qué tamaño es razonable, si el integrador especifica directamente un modo mediante el parámetro, este se respeta, mientras que si no indica ningún modo, se generarán firma implícitas para tamaños inferiores a 1MB, y explícitas tamaños mayores del contenido a firmar.

- 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.

7.5.1 Configuración de la Máquina Virtual de Java para tratamiento de ficheros grandes

Hay ocasiones en las que el MiniApplet no puede tratar ficheros de gran tamaño por escasez de memoria disponible para la máquina virtual.

Para paliar esta situación podemos indicar a la JVM que disponga de una mayor cantidad máxima de memoria usando la opción "-Xmx" de Java. Por ejemplo, para que Java disponga de 2GB de memoria debemos indicar:

```
MiniApplet.JAVA_ARGUMENTS += "-Xms512M -Xmx2048M ";
```

No obstante, debemos tener mucha precaución al usar estas opciones, ya que si el equipo no dispone de la memoria que solicitamos como máximo, el MiniApplet no funcionará en absoluto, por lo que debemos usarlo únicamente en entornos donde estemos seguros de que los equipos dispondrán de una de una determinada cantidad de memoria libre.

Puede encontrar más información sobre las opciones de configuración de memoria de la JVM en:

- https://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html

Tenga en cuenta que esta configuración no se traslada a AutoFirma en caso de que sean estos los que realicen la firma. AutoFirma tiene prefijada la reserva máxima de 1GB de memoria en el caso de su versión de 32bits y 8Gb de memoria máximos en el caso de la versión de 64bits (en este caso, no se produce error si el equipo cliente no cuenta con tanta memoria). En pruebas realizadas, esta configuración permite realizar firmas de hasta 130 MB en el caso de la versión de 32bits y hasta 400MB (sin llegar al límite máximo de memoria reservada) en el caso de AutoFirma de 64bits.

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, sucessCallback,
           showErrorCallback);
  } catch(e) {
      showErrorCallback(MiniApplet.getErrorType(), MiniApplet.getErrorMessage());
  }
```

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

9 Compatibilidad con dispositivos móviles y AutoFirma

Cuando el entorno del ciudadano firmante no puede realizar la carga del MiniApplet, el JavaScript de despliegue deriva las tareas de firma a aplicaciones nativas capaces de realizarlas sin necesidad de que el integrador deba gestionar por su parte esta delegación de tareas. De esta forma pueden completarse trámites electrónicos aún en entornos en los que no es posible ejecutar Applets.

Entornos en los que no es posible ejecutar Applets son los navegadores Web de los dispositivos móviles, que no admiten complementos capaces de realizar firmas electrónicas (Applets de Java, ActiveX, etc.). También ocurre con algunos navegadores de escritorio, como Google Chrome, Mozilla Firefox o Microsoft Edge o bajo otras situaciones en las que no es posible ejecutar el MiniApplet:

- Java no está instalado en el sistema.
- El usuario no concedió permisos para la ejecución del *Applet*.
- La configuración de seguridad del equipo no permitió la ejecución del *Applet*.
- Etc.

Las aplicaciones que soportan la ejecución de estas tareas de firma son: AutoFirma, para entornos Windows, Linux y Mac OS X; el Cliente @firma Android, para entornos Android; y el Cliente @firma iOS, para entornos iOS. Nos referiremos a estos dos últimos de forma general como Clientes @firma móviles.

Aunque estas aplicaciones pueden ejecutar las operaciones de firma en lugar del MiniApplet, existen limitaciones en cuanto al resto de operaciones, así como una serie de criterios y buenas prácticas que deberá seguir el integrador para garantizar la compatibilidad con ambas herramientas.

Para que la versión nativa de AutoFirma y los Clientes @firma móviles puedan responder a las peticiones de firma, es necesario que estén instaladas en el equipo local antes de iniciar el trámite de firma. Es responsabilidad del integrador, alertar de este hecho cuando sea susceptible que los usuarios no tengan instalada la aplicación, ya que no es posible para el navegador determinar si el usuario la tiene instalada o no.

Consulte el manual "*AF_manual_instalacion_y_gestion_ES*" para conocer más detalles acerca de la instalación y configuración de AutoFirma y las aplicaciones móviles.

De AutoFirma existe, además de la versión nativa, una versión WebStart de la aplicación. La funcionalidad de ambas versiones es la misma pero, mientras que la versión nativa debe estar instalada en el sistema antes de iniciar la operación de firma, la versión WebStart se instala automáticamente en el momento de llamar al método de carga del MiniApplet. Consulte el apartado AutoFirma WebStart para saber más acerca de esta versión de AutoFirma.

Siempre que se utilice una aplicación nativa o AutoFirma WebStart como alternativa al uso de applets, ha de tenerse en cuenta que estas aplicaciones deben poder confiar en los certificos SSL utilizados por cualquier servicio externo al que se deda conectar. En caso contrario, fallaría la conexión. Esto es especialmente crítico cuando se hace uso de la comunicación por servidor intermedio o se utiliza la operación de firma trifásica. Para saber más sobre este tema consulte el apartado "Comunicación con servicios externos" del manual "*AF_manual_instalacion_y_gestion_ES*".

Si se desease delegar en AutoFirma y los clientes @firma móviles las tareas de firma en todos los casos, se podría hacer tal como se describe en el apartado Carga directa de AutoFirma y los clientes móviles.

9.1 Arquitectura del sistema

A pesar de que es el JavaScript de despliegue el que gestiona la delegación de tareas en la aplicación nativa correspondiente a cada entorno, deben cumplirse una serie de requisitos para que esto sea posible.

AutoFirma es una herramienta nativa que puede instalarse en Windows, Linux y Mac OS X como herramienta independiente de firma de datos locales. Sin embargo, al instalarse en el sistema también registran el protocolo "`afirma`", mediante lo cual atienden a las llamadas realizadas a este protocolo. La llamada por protocolo es un mecanismo de invocación de aplicaciones (si acaso la propia aplicación que realiza la llamada no puede atender a este protocolo) mediante el que puede pasarse una cantidad limitada de información a modo de parámetros. Sin embargo, este protocolo sólo permite la comunicación en un sentido ya que la aplicación invocada no puede hacer referencia a la instancia de aquella que la invocó.

Es por esto que, para permitir la comunicación bidireccional, Autofirma establece 2 mecanismos distintos:

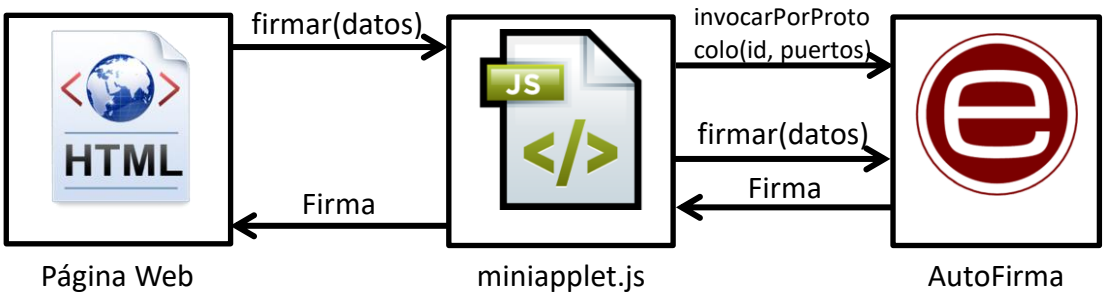
- La comunicación mediante *socket*. Este sistema se utiliza en Internet Explorer 11, Chrome y Firefox, y se explica en el apartado Entornos de escritorio (Windows, Linux y Mac OS X) con nuevos navegadores.
- La comunicación mediante un servidor intermedio. Este sistema puede utilizarse potencialmente en cualquier navegador (y el Cliente puede configurarse para que así sea) aunque, dado que ofrece un peor rendimiento, sólo se utiliza por defecto en los entornos que no soportan la comunicación por sockets (dispositivos móviles, Internet Explorer 10, Edge, Safari 10…). Este modo se explica en el apartado Dispositivos móviles, Internet Explorer 10 e inferiores.

IMPORTANTE: Para que nuestro despliegue funcione en todos los entornos posibles, es necesario que se desplieguen los servicios del servidor intermedio

9.1.1 Entornos de escritorio (Windows, Linux y Mac OS X) con nuevos navegadores

En la comunicación por *socket*, AutoFirma abre un *socket* de comunicación cifrado a través del cual poder establecer una canal de comunicación bidireccional.

La arquitectura de comunicación seguida en este proceso se describe a continuación:



Primeramente, el trámite ordena la operación de firma por medio del JavaScript de despliegue del MiniApplet. Al detectar este que no es posible ejecutar el MiniApplet, lanzará AutoFirma mediante una invocación por protocolo en el que se le proporcionarán una serie de puertos aleatorios (dentro del rango determinado para el uso de aplicaciones de terceros) a través de los cuales se deberá intentar la apertura del *socket* y un identificador de sesión aleatorio que deberán presentar todas las peticiones realizadas a través del mismo.

La aplicación AutoFirma abrirá este *socket* cifrando el canal mediante un certificado SSL generado en el momento de la instalación (consulte el documento "*AF_manual_instalacion_y_gestion_ES*" para más detalles) y el JavaScript enviará la orden de firma a través del mismo. El *socket* se abrirá en uno de los puertos enviados en la invocación, así que el JavaScript de despliegue deberá previamente comprobar cuál de ellos es el que se ha utilizado finalmente antes de realizar la petición. AutoFirma devolverá el resultado de la operación a través del propio *socket*.

Una vez abierta la aplicación, cualquier petición posterior se realizará a través del mismo socket, sin necesidad de relanzarla. Pasado un tiempo determinado después de la última operación realizada, AutoFirma se cerrará, de tal forma que la siguiente operación sí deberá relanzarla, proporcionando nuevos puertos y un nuevo identificador de sesión. El tiempo de espera por defecto que tardará la aplicación en cerrarse después de la última operación realizada es de 1 minuto.

La comunicación a través del socket se cifra mediante el protocolo TLS v1.1/v1.2. Es necesario tener habilitados estos protocolos en el navegador para poder establecer la comunicación con AutoFirma.

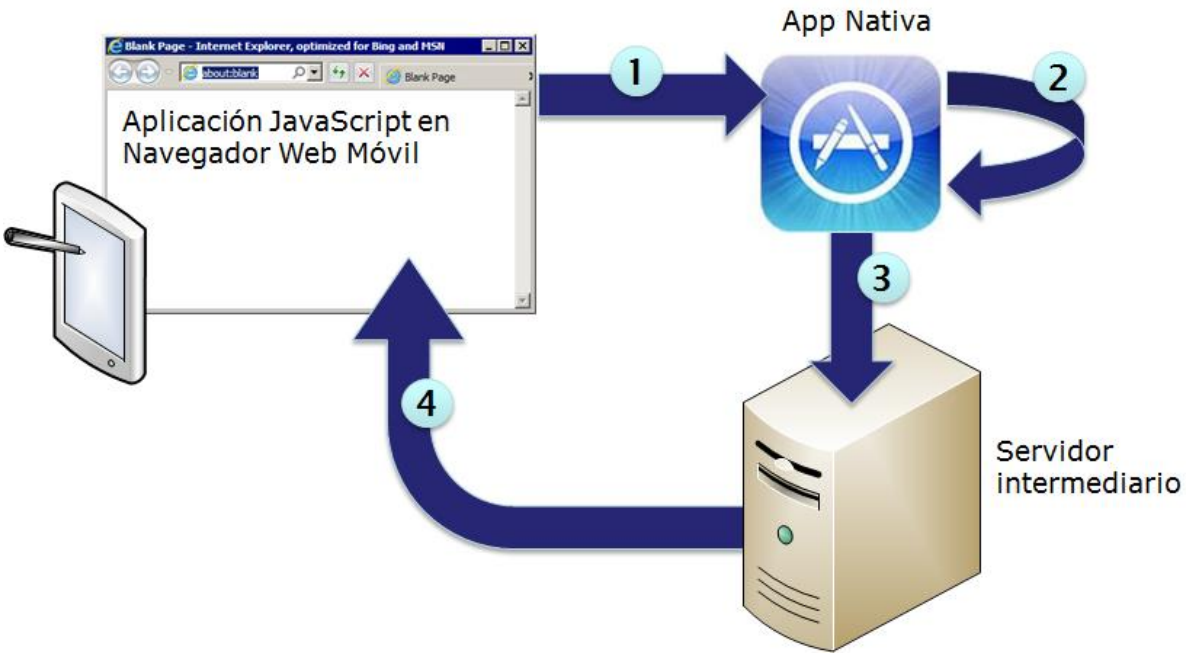
Puede consultar la dirección de descarga de AutoFirma y su proceso de instalación en el documento "*AF_manual_instalacion_y_gestion_ES*".

9.1.2 Dispositivos móviles, Internet Explorer 10 e inferiores, Microsoft Edge y Safari 10

Dado que los sistemas móviles imponen limitaciones en cuanto a la creación de servicios que puedan ejecutarse en paralelo con otras aplicaciones ha sido necesario idear otro tipo de canal de

comunicación bidireccional entre la página web que integra el trámite y la aplicación de firma. También ocurre que algunos navegadores no soportan o restringen la comunicación a través de un socket local, por lo que no es posible utilizar la solución anteriormente descrita para estos casos.

La arquitectura de comunicación con la que funcionan los clientes @firma móviles y AutoFirma cuando no es posible la comunicación por sockets es la siguiente:



1. Cuando detecta que está en un dispositivo móvil o no se puede cargar el MiniApplet @firma, 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, etc.), 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.

Existe otro caso de uso de este medio de comunicación, en el cual los datos de entrada para la aplicación móvil no se pueden transmitir mediante la llamada por protocolo debido a su tamaño. En ese caso, la página web primeramente los sube al servidor intermedio y a continuación realiza la invocación por protocolo a la aplicación para que esta los descargue y continúe normalmente con el proceso.

Nota: Debe tenerse en cuenta que para el correcto envío de datos desde AutoFirma al servidor intermedio, cuando la comunicación se realiza sobre SSL, requiere que la JVM utilizada confíe en el certificado SSL utilizado. En caso de no ser un certificado SSL reconocido por defecto por Java, el usuario (o un administrador del sistema) deberá introducir los certificados emisores en el almacén de confianza de la JVM utilizada (la incorporada con AutoFirma en Windows y macOS, o la del sistema en Linux).

La instalación de los Clientes @firma móviles se realiza desde la tienda de aplicaciones de Android e iOS. Puede consultar los enlaces de descarga en el documento "*AF_manual_instalacion_y_gestion_ES*".

9.2 Despliegues compatibles

Para que nuestros despliegues del MiniApplet sean compatibles con todos 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, 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 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.

- Por motivos seguridad, este método no admite direcciones URL locales (127.0.0.1 / *localhost*). Si se desea hacer pruebas en local, establezca la dirección IP configurada en el equipo.
- 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>
```

- Obtención de resultados mediante *callbacks*
  - El MiniApplet permite obtener los resultados de las operaciones de forma inmediata o mediante la definición de funciones *callback*. Sin embargo, para que nuestro despliegue sea compatible con los clientes nativos se debe usar siempre el modo de obtención de resultados mediante *callbacks*.
  - Consulte el apartado Obtención de resultados mediante *Callbacks* para más detalle de cómo usar este modo.
- Evitar el uso iterativo de las llamadas a los métodos del MiniApplet (por ejemplo, múltiples llamadas en bucle al método de firma)
  - Las llamadas a las aplicaciones nativas se realizan de forma asíncrona e implican que una instancia de la aplicación las atienda. Sin embargo, sólo se admite que haya una instancia de la aplicación levantada, así que sólo se atenderá una de las peticiones lanzadas.
  - Para el permitir la compatibilidad con las aplicaciones móviles no debemos solicitar que la aplicación realice una operación hasta que no haya finalizado la operación anterior. Esto lo haremos invocando a la siguiente operación desde los métodos *successCallback* y/o *errorCallback* definidos para la operación anterior. Por ejemplo:
  - Un ejemplo de uso de este comportamiento es:

```
var indice;
var datos = new Array();
datos[0] = "…";
datos[1] = "…";
…
function procesarFirma(signatureB64, certB64) {
   enviar(signatureB64);
   MiniApplet.sign( datos[++indice], algoritmo, formato, params,
                    procesarFirma, errorFirma);
}
…
function firmar() {
   indice = 0;
   MiniApplet.sign( datos[indice], algoritmo, formato, params,
```

```
                    procesarFirma, errorFirma);
}
```

- **ADVERTENCIA:** Google Chrome no permite más de una llamada externa al navegador (como las llamadas a la aplicación nativa) por cada interacción del usuario. En el caso de la comunicación por sockets de AutoFirma esto no supone un problema pero, en el caso de la comunicación a través de un servidor intermedio de AutoFirma y los clientes móviles, esto supone que sólo se atenderá la primera operación solicitada.

- 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 implementan los clientes nativos de @firma.

9.3 Configuración de la invocación a AutoFirma

Tras la invocación por protocolo a AutoFirma, el JavaScript de despliegue realiza varios intentos de conexión a través de los distintos puertos generados. Si no consigue conectar con la aplicación en ese tiempo, lanza un error indicando que no se ha podido conectar con la aplicación. El tiempo de espera total viene determinado aproximadamente por el número de intentos de conexión que realiza y el tiempo de espera que se produce entre cada uno de ellos. Este tiempo ronda por defecto los 25 segundos.

Dado que la invocación por protocolo a AutoFirma puede no tener una respuesta inmediata, debido a que implica levantar una máquina virtual Java, levantar la aplicación y abrir el puerto, más que puede ser necesario que el usuario dé su consentimiento para la ejecución de la aplicación, puede darse el caso de ser corto en la primera ejecución. Es por eso que el JavaScript de despliegue permite la configuración de este valor mediante un par de variables. Estas son:

- `AUTOFIRMA_LAUNCHING_TIME`: Tiempo de espera en milisegundos entre intentos de conexión con AutoFirma. Por defecto, 2000 milisegundos.
- `AUTOFIRMA_CONNECTION_RETRIES`: Número de reintentos de conexión que se intentarán antes de determinar que no es posible conectar con la AutoFirma. Por defecto, 10 reintentos.

La configuración de estas variables se realizará por medio del objeto `MiniApplet`. Por ejemplo:

```
<script type="text/javascript">
     // Se realizarán 15 reintentos de conexión con AutoFirma en caso
     // de ser necesarios
     MiniApplet.AUTOFIRMA_CONNECTION_RETRIES = 15;
     MiniApplet.cargarMiniApplet("http://miweb.com/afirma");
</script>
```

9.4 Servicios auxiliares de AutoFirma y el 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.

Estos servicios se distribuyen en forma de archivos WAR que deberán ser desplegados en un servidor de aplicaciones Java. Los archivos en los que se distribuyen son:

- afirma-signature-retriever.war
- afirma-signature-storage.war

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.

9.4.1 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.

9.4.1.1 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.

9.5 Notificaciones al usuario

Es **obligatorio que el usuario tenga instalado AutoFirma o el Cliente @firma móvil**, según corresponda, 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.

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";
}
```

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

Puede consultar los enlaces de descarga de AutoFirma y los clientes @firma móvil en el documento "*AF_manual_instalacion_y_gestion_ES*".

9.6 Limitaciones

Debe tenerse en cuenta que las versiones actuales de los distintos clientes móviles no implementan toda la funcionalidad disponible en el MiniApplet, y que los clientes móviles de Android e iOS cuentan con distinta funcionalidad entre sí. 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 soporta todos los formatos). 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

9.6.2 Limitaciones funcionales

Una limitación común a las aplicaciones móviles es que 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. AutoFirma sí utiliza el almacén del navegador o el configurado por el integrador en caso de haberlo hecho.

9.6.2.1 Sistemas Android

- `function getFileNameContentBase64(title, extension, description, filePath, successCallback, errorCallback)`
  - 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, filePath, successCallback, errorCallback)`
  - 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.
- `function signBatch(batchB64, batchPreSignerUrl, batchPostSignerUrl, params, successCallback, errorCallback)`
  - Los clientes móviles no permiten realizar operaciones de firma por lotes.

9.6.2.2 Sistemas iOS

- `function saveDataToFile(dataB64, title, fileName, extension, description, successCallback, errorCallback)`
  - 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, filePath, successCallback, errorCallback)`

- 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, filePath, successCallback, errorCallback)`
  - 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.
- `function signBatch(batchB64, batchPreSignerUrl, batchPostSignerUrl, params, successCallback, errorCallback)`
  - Los clientes móviles no permiten realizar operaciones de firma por lotes.

9.6.2.3 AutoFirma

La última versión nativa de AutoFirma y AutoFirma WebStart son compatibles con todas las funciones del MiniApplet cuando se utiliza la comunicación por sockets. Sin embargo, no era así en versiones anteriores de la aplicación nativa. Si los usuarios de su aplicación son susceptibles de tener instalada una versión anterior de AutoFirma, tenga en cuenta las siguientes consideraciones:

- AutoFirma 1.6 en modo de comunicación por sockets admite todas las operaciones del MiniApplet. Sin embargo, en la comunicación a través de servidor intermedio y se mantienen las limitaciones de AutoFirma 1.5.
- AutoFirma 1.6 en módo de comunicación por servidor intermedio (al igual que AutoFirma 1.5 y anteriores en todos los casos) no es compatible con las funciones:
  - `function getFileNameContentBase64(title, extension, description, filePath, successCallback, errorCallback)`
    - 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, filePath, successCallback, errorCallback)`

        - 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.
        - Se puede emular el funcionamiento del método setStickySignatory mediante el uso de filtros de certificados y el parámetro extra *headless*. Para ello, al terminar la primera operación de firma (o si se utiliza el método `selectCertificate`) se puede tomar el certificado de firma y (mediante JavaScript o un proceso en servidor) se definirá un filtro de certificados que lo seleccione sólo a él. Al establecer este filtro y el parámetro `headless=true` en las siguientes firmas, se seleccionará automáticamente ese certificado.
    - `function getCurrentLog()`
        - AutoFirma no permite obtener las trazas generadas durante su ejecución.
- AutoFirma 1.4.2 y 1.4.3, adicionalmente, no era compatibles con las funciones:
    - `function signAndSaveToFile(operationId, dataB64, algorithm, format, params, outputFileName, successCallback, errorCallback)`
    - `function setKeyStore(ksType)`
    - `function selectCertificate(params, successCallback, errorCallback)`

9.7 Compatibilidad de versiones

Los despliegues del MiniApplet @firma 1.6, obviando las limitaciones funcionales, son compatibles con:

- AutoFirma v1.4.2 y superiores
    - Compatible con sistemas Windows 7 y superiores; sistemas Linux; MacOS 8 y superiores.
    - http://firmaelectronica.gob.es/Home/Descargas.html
- Cliente @firma móvil Android v1.4 y superiores
    - Puede ver los requisitos de compatibilidad en la Play Store.
    - https://play.google.com/store/apps/details?id=es.gob.afirma&hl=es
- Cliente @firma móvil iOS v1.4 y superiores
    - Puede ver los requisitos de compatibilidad en la Apple Store.
    - https://itunes.apple.com/us/app/cliente-firma-movil/id627410001?mt=8

10 AutoFirma WebStart

AutoFirma WebStart es una versión de AutoFirma preparada para su despliegue JNLP. Este modo de despliegue permite que una aplicación se descargue e instale la primera vez que se cargue. En

cargas subsiguientes en el mismo dominio la aplicación se cargará desde la caché de Java en lugar de volverse a descargar.

AutoFirma WebStart se cargará en al menos dos ocasiones como parte de una operación. La primera será en el momento de llamar al método de carga de la aplicación. Esta primera llamada tiene como único fin obligar a que se realice la descarga de la aplicación si no lo estaba ya. La segunda llamada se realizará en el momento de la firma. Al llegar a esta segunda llamada ya la aplicación estará descargada y, por tanto, no será necesario volver a hacerlo evitando que la operación de firma dure demasiado tiempo y pueda interpretarse como un error en la operación.

Al contrario que la versión nativa de AutoFirma, sí es necesario tener instalado Java en el sistema para poder ejecutar AutoFirma WebStart (Java 8 o superior) y, al igual que con el MiniApplet @firma, la aplicación debe estar firmada con un certificado de firma emitido por una autoridad de confianza. Adicionalmente, es recomendable registrar en el Manifest de la aplicación el dominio desde el que se le va a invocar para evitar mostrar diálogos de advertencia adicionales al usuario. Sin embargo, esto no es obligatorio e implicaría volver a firmar la aplicación.

AutoFirma WebStart tiene exactamente la misma funcionalidad que la versión nativa de AutoFirma y soporta tanto la comunicación por sockets como la comunicación a través de servidor intermedio. Cuando se despliegue dentro de una aplicación web con SSL cliente, sin embargo, es recomendable que la aplicación sea accesible desde una URL sin SSL cliente para evitar problemas de comunicación por sockets.

AutoFirma WebStart está sujeto a los mismos requisitos que AutoFirma en cuanto a la confianza en los certificados SSL utilizados en los servicios a los que deba acceder. Consulte el apartado “Servicios auxiliares de AutoFirma y el Cliente @firma móvil” del manual de AutoFirma para obtener más información al respecto.

10.1 Requisitos

Para que pueda ejecutarse AutoFirma WebStart en el entorno del usuario debe cumplir los siguientes requisitos:

- Java 8 o superior.
- Sistema operativo Windows, Linux o macOS.
- Navegador web Intenet Explorer, Edge, Chrome o Safari.

AutoFirma WebStart no es compatible con Mozilla Firefox, debido a que no es posible instalar en su almacén de confianza, desde la aplicación WebStart, los certificados necesarios para que se comuniquen la aplicación y el navegador.

En caso de que el usuario utilizase Mozilla Firefox o que su entorno no cumpliese cualquiera de los requisitos anteriores, no se invocará a AutoFirma WebStart. En su lugar se invocará al cliente nativo correspondiente al sistema (AutoFirma nativo o cliente móvil).

10.2 Despliegue de AutoFirma WebStart

Para permitir el uso de AutoFirma WebStart como herramienta alternativa de firma dentro de un flujo de firma web, será necesario cumplir los siguientes requisitos:

- Desplegar el archivo "`afirma-server-simpleafirma-webstart.war`" en un servidor de aplicaciones Java para dar acceso al servicio "`jnlp`" que proporciona.
  - Este fichero WAR contiene internamente un fichero de propiedades "`autofirma-jnlp.properties`". Este fichero contiene una única propiedad "`codebase`", en el que se deberá configurar la URL base en la que se publica el fichero JAR con la aplicación Java WebStart. A modo de aclaración, si se publican en el mismo directorio los ficheros JAR del MiniApplet y AutoFirma WebStart, la URL prorporcionada al método de carga del MiniApplet/AutoFirma y la URL configurada en la propiedad "`codebase`" del fichero "`autofirma-jnlp.properties`" serían la misma.
- Invocar al método `setJnlpService(String)` del JavaScript de despliegue antes del método de carga del cliente. Este método establece la URL del servicio anteriormente desplegado. Este servicio generará el fichero JNLP que se utilizará para cargar AutoFirma WebStart.

Un ejemplo de uso sería:

```
MiniApplet.setJnlpService(
      Constants.URL_BASE_SERVICES +
      "/afirma-server-simpleafirma-webstart/jnlp");
MiniApplet.cargarMiniApplet(Constants.URL_BASE_APPLET);
…
```

10.3 Advertencias al usuario

La carga de la aplicación WebStart mediante una llamada por protocolo, como se hace con AutoFirma WebStart, implica que el navegador web notifique al usuario que se va a ejecutar la máquina virtual de Java. Esto se materializa en un diálogo del navegador, distinto para cada uno de los navegadores soportados, en donde se pide permiso para ejecutar la aplicación. En caso de aceptar, se cargará la máquina virtual de Java y se cargará la aplicación, ya sea desde la caché de Java o descargándola de internet si no se encontró allí. A continuación, se comprobará la firma de la aplicación y se le preguntrará al usuario si desea ejecutar la aplicación de AutoFirma, mostrando la información del firmante.

Estos diálogos se mostrarán cada vez que se realice la carga de la aplicación y se ejecute una operación, salvo que el usuario seleccione expresamente que no se vuelvan a mostrar. El aspecto de los diálogos y el nivel de advertencia de los mensajes variará según el navegador web, la confianza que se tenga en el certificado de firma de la aplicación y si se registró expresamente dentro de su fichero manifest el dominio desde el que se carga.

11 Información específica para firmas CAdES

11.1 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 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. Igualmente, si los certificados del usuario se encuentran en tarjeta inteligente, asegúrese de disponer de la última versión de su controlador para garantizar la compatibilidad con estos algoritmos de firma. En caso de que, aún así, no pueda utilizar este algoritmo con su tarjeta inteligente, consulte la información de compatibilidad de su tarjeta y/o pruebe con otro algoritmo, como `SHA256withRSA`.

11.2 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=implicit
```

Consulte la sección "Parámetros adicionales" para mayor información.

11.3 Parámetros adicionales

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

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

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

		CAdES-EPES.
`includeOnlySignningCertificate`	`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.
`signatureProductionCity`	[Texto]	Agrega a la firma un campo con la ciudad en la que se realiza la firma. La codificación debe ser UTF-8.
`signatureProductionPostalCode`	[Texto]	Agrega a la firma un campo con el código postal en donde se realiza la firma. La codificación debe ser UTF-8.
`signatureProductionCountry`	[Texto]	Agrega a la firma un campo con el país en la que se realiza la firma. La codificación debe ser UTF-8.
`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 indicado en esta propiedad menos 1.
`commitmentTypeIndication`***n***`Identifier`	`1`	Establece que el CommitmentTypeIndications número ***n*** (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***`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).
`signingCertificateV2`	[Booleano]	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.

11.3.2 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 `policyIdentifier`, al igual que es obligatorio también dar valor al parámetro `policyIdentifierHashAlgorithm`.
`policyIdentifierHashAlgorithm`	`SHA1`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA1.

	SHA-256	Indica que la huella digital indicada en el parámetro policyIdentifierHash se calculó mediante el algoritmo SHA-256.
	SHA-384	Indica que la huella digital indicada en el parámetro policyIdentifierHash se calculó mediante el algoritmo SHA-284.
	SHA-512	Indica que la huella digital indicada en el parámetro policyIdentifierHash se calculó mediante el algoritmo SHA-512.
policyQualifier	[URL hacia documento]	URL (universalmente accesible) hacia el documento (normalmente PDF) que contiene una descripción textual de la política de firma. Este parámetro es opcional incluso si se desea generar firmas CAdES-EPES.
includeOnlySignningCertificate	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.
signatureProductionCity	[Texto]	Agrega a la firma un campo con la ciudad en la que se realiza la firma. La codificación debe ser UTF-8.
signatureProductionPostalCode	[Texto]	Agrega a la firma un campo con el código postal en donde se realiza la firma. La codificación debe ser UTF-8.
signatureProductionCountry	[Texto]	Agrega a la firma un campo con el país en la que se realiza la firma. La codificación debe ser UTF-8.
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.
commitmentTypeIndication***n***Identifier	1	Establece que el CommitmentTypeIndication

		número ***n*** es "**Prueba de origen**".
	2	Establece que el CommitmentTypeIndication número ***n*** es "**Prueba de recepción**".
	3	Establece que el CommitmentTypeIndication número ***n*** es "**Prueba de entrega**".
	4	Establece que el CommitmentTypeIndication número ***n*** es "**Prueba de envío**".
	5	Establece que el CommitmentTypeIndication número ***n*** es "**Prueba de aprobación**".
	6	Establece que el CommitmentTypeIndication número ***n*** es "**Prueba de creación**".
`commitmentTypeIndication`***n***`CommitmentTypeQualifiers`	[OID]	Lista de OID separados por el caracter '|' que se aportan como calificadores adicionales del CommitmentTypeIndication número n (atributo opcional).

12 Información específica para firmas XAdES

12.1 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

12.2 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.

12.3 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 "12.6" para más información.

12.4 Situación del nodo de firma en XAdES Enveloped

Por defecto, el MiniApplet 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 insertara 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']
```

12.5 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-xmldsig-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/xmldsig#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/xmlenc#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.

12.5.1 Formatos de XAdES compatibles con estructuras *Manifest*

Las firmas XAdES con *Manifest* son compatibles con el formato *Enveloping* y cualquier modo del formato *Detached*, **pero no con el formato *Enveloped***. Dado que el sentido del uso del *Manifest* es desligar la referencia de la firma con la referencia a los datos firmados, no tiene sentido esta separación cuando la firma comparte estructura con los datos firmados, como ocurre en el formato XAdES Enveloped.

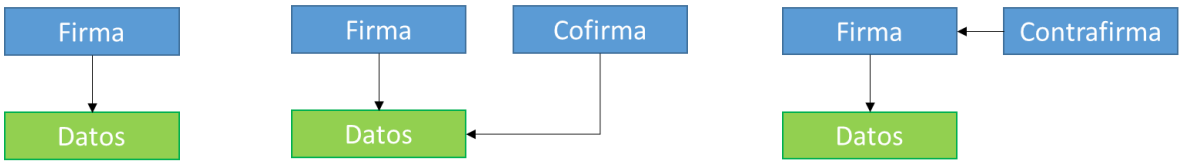
12.5.2 Cofirmas y contrafirmas *Manifest*

Las firmas XAdES funcionan en base a referencias. Una firma XAdES tradicional, hace referencia a los datos firmados, que pueden ser internos o externos; una cofirma hace referencia a los mismos datos que la primera firma; y la contrafirma hace referencia a una firma.

Cuando se realiza una firma manifest, lo que se hace es insertar en medio un nuevo conjunto de referencias a los datos, el manifest. El uso de este objeto en las firmas es claramente correcto, pero su uso en las operaciones de cofirma y contrafirma pueden llevar a efectos indeseados, ya que se introduce un nuevo manifest entre la cofirma/contrafirma y la firma original.

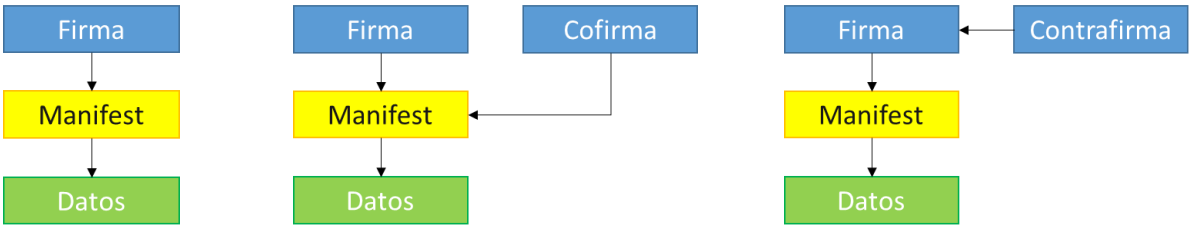
A continuación, se muestran varios gráficos ilustrando las posibilidades:

- Operaciones sin *manifest:*



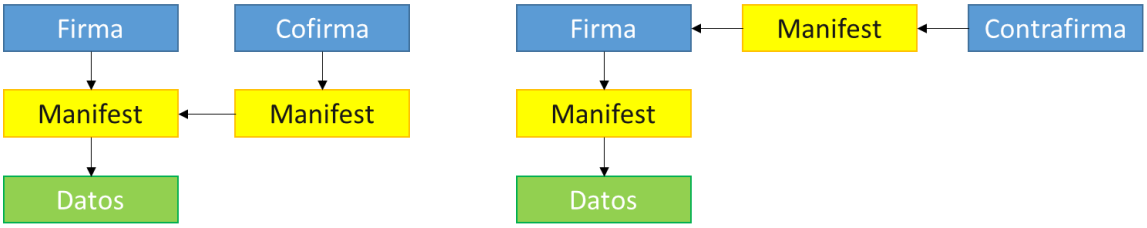
Firma tradicional | Cofirma sobre firma tradicional | Contrafirma sobre firma tracional

- Operaciones con *manifest* en la firma original:



Firma *manifest* Cofirma sobre firma *manifest* Contrafirma sobre firma *manifest*

- Multifirmas *manifest* sobre firmas *manifest*:



Cofirma *manifest* sobre firma *manifest* Contrafirma *manifest* sobre firma *manifest*

Por regla general y salvo que se desee expresamente crear este efecto, **para la realización de cofirmas y contrafirmas XAdES nunca deberían definir el atributo** `useManifest=true`.

12.5.3 Firma *manifest* a partir de la huella digital de los datos

Es común querer hacer uso de las firmas de *manifest* como medio de no importar los datos dentro de la firma electrónica (XAdES Externally Detached) para reducir su tamaño, el coste computacional (importante con la firma trifásica) y el tiempo necesario para generarla. Sin embargo, aunque no se necesiten los datos para validar la firma, sí son necesarios para generarla, ya que deben introducirse sus referencias en el manifest y calcular la huella digital (hash) que resulta de derreferenciar estos datos. Esto hace que el Cliente @firma (o el servidor trifásico) deban descargarse los datos para generar la firma, lo que penaliza todo el proceso.

Alternativamente, si ya disponemos de la huella digital de los datos, es posible generar firmas *manifest* sin necesidad de descargarlos. Para hacer esto, sólo necesitaremos:

1. Huella digital de los datos.
2. Algoritmo de huella digital utilizado para generar la huella. Por ejemplo, “SHA-512”.
3. Referencia a los datos. Esta referencia debe ser una URI a partir de la cual nuestro sistema sea capaz de derreferenciar los datos para su comprobación. No es necesario que el Cliente @firma sea capaz de entender esa referencia, ya que no recuperará los datos en ningún momento. Por ejemplo, si el número de asiento del documento en nuestro repositorio fuese el 3086, podríamos usar “urn:id:3086”.

Para generar la firma *manifest* con esta configuración de ejemplo y sin pasarle los datos, realizaríamos la firma con los siguientes *extraParams*.

```
format=XAdES Externally Detached
useManifest=true
precalculatedHashAlgorithm=SHA-512
uri=urn:id:3086
```

En la llamada al método de firma, además de estos *extraParams*, indicaríamos la huella del documento como el parámetro de datos a firmar:

```
MiniApplet.sign(huellaB64, "SHA512withRSA", "XAdES", extraParams, successCallback, errorCallback);
```

Como resultado, obtendremos una firma XAdES Detached con un *manifest*, en el que aparecerá la referencia indicada a los datos y su huella digital.

12.6 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.

12.6.1 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 *Enveloped*. 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 *Enveloped*.
`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. **ADVERTENCIA:** La cofirma de un documento en el que al menos una de las firmas no incluye la transformación XPATH, dará lugar a un documento de firma que potencialmente será validado incorrectamente por los validadores de firma. Por este motivo, sólo se permite el uso de este parámetro en la operación de firma (no en la de cofirma).
	`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 *Enveloping*. Este es el formato que se utiliza por defecto cuando no se indica ninguno.
	`XAdES Enveloped`	Genera firmas en formato *Enveloped*.
	`XAdES Detached`	Genera firmas en formato *Internally Detached*.
`includeOnlySignningCertificate`	`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 `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 firmado.
`referencesDigestMethod`	`http://www.w3.org/2000/09/xmldsig#sha1`	Usa el algoritmo SHA1 para el cálculo de las huellas digitales de las referencias XML firmadas.
	`http://www.w3.org/2001/04/xmlenc#sha256`	Usa el algoritmo SHA-256 para el cálculo de las huellas digitales de las referencias XML firmadas. Este es el valor recomendado.
	`http://www.w3.org/2001/04/xmlenc#sha512`	Usa el algoritmo SHA-512 para el cálculo de las huellas digitales de las referencias XML firmadas.
`mimeType`	[Texto en formato MIME-Type]	MIME-Type de los datos a firmar. Si no se indica este parámetro el sistema intenta auto-detectar el tipo, estableciendo el más aproximado (que puede no ser el estrictamente correcto).
`encoding`	[URI]	Codificación de los datos a firmar (ver la documentación del elemento Object de XMLDSig para más información). Un uso incorrecto de este parámetro puede provocar la generación de

		una firma inválida. Si se proporcionan datos a firmar previamente codificados en Base64 pero se desea sean considerados como su forma descodificada, debe establecerse este valor a `http://www.w3.org/2000/09/xmldsig#base64` y especificarse el tipo real en el parámetro `mimeType`. Por ejemplo, para firmar una imagen PNG haciendo que la firma se refiera a su forma binaria directa, puede proporcionarse la imagen directamente codificada en Base64 indicando el `encoding` como `http://www.w3.org/2000/09/xmldsig#base64` y el `mimeType` como `image/png`. El valor debe ser siempre una URI.
`outputXmlEncoding`	[Texto]	Codificación del XML de salida. Si no se indica este valor se intenta auto-detectar a partir del XML de entrada (si los datos a firmar son un XML).
`contentTypeOid`	[OID o URN de tipo OID]	Identificador del tipo de dato firmado. Este parámetro es complementario (que no excluyente) al parámetro `mimeType`.
`canonicalizationAlgorithm`	`http://www.w3.org/TR/2001/REC-xml-c14n-20010315`	Se firma el XML con canonizado XML 1.0 inclusivo (valor por defecto).
	`http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments`	Se firma el XML con canonizado XML 1.0 inclusivo con comentarios.
	`http://www.w3.org/2001/10/xml-exc-c14n#`	Se firma el XML con canonizado XML 1.0 exclusivo.
	`http://www.w3.org/2001/10/xml-exc-c14n#WithComments`	Se firma el XML con canonizado XML 1.0 exclusivo con comentarios.
`xadesNamespace`	[URL]	URL de definición del espacio de nombres de XAdES (el uso de este parámetro puede

		condicionar la declaración de versión de XAdES). Si se establece este parámetro es posible que se necesite establecer también el parámetro `signedPropertiesTypeUrl` para evitar incoherencias en la versión de XAdES.
`signedPropertiesTypeUrl`	[URL]	URL de definición del tipo de las propiedades firmadas (*Signed Properties*) de XAdES. Si se establece este parámetro es posible que se necesite establecer también el parámetro `xadesNamespace` 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 éstas dejándolas sin firmar.
	`false`	Si se firma un XML con hojas de estilo, firma también las hojas de estilo (valor por defecto, consultar notas adicionales sobre firma de hojas de estilo).
`avoidBase64Transforms`	`true`	No declara transformaciones Base64 incluso si son necesarias.
	`false`	Declara las transformaciones Base64 cuando se han codificado internamente los datos a firmar en Base64 (valor por defecto).
`headless`	`true`	Evita que se muestren diálogos gráficos adicionales al usuario (como por ejemplo, para la *dereferenciación* de hojas de estilo enlazadas con rutas relativas).
	`false`	Permite que se muestren diálogos gráficos adicionales al usuario.
`xmlTransforms`	[Número]	Número de transformaciones a aplicar al contenido firmado. Debe indicarse posteriormente igual número de parámetros `xmlTransformnType`, sustituyendo *n* por un ordinal consecutivo, comenzando en 0 (ver notas adicionales sobre indicación de transformaciones adicionales).
`xmlTransform`***n***`Type`	`http://www.w3.org/2000/09/xmld`	Indica que los datos que se proporcionan para firmar ya están codificados en base64 y se debe

	`sig#base64`	declarar esta transformación adicional para que se decodifiquen antes de firmarlos. Esta transformación base64 es adicional a la transformación necesaria para pasar los datos a través de los métodos de firma del cliente.
	`http://www.w3.org/TR/1999/REC-xpath-19991116`	El contenido se debe procesar mediante esta transformación XPATH antes de ser firmado. Únicamente es aplicable cuando se firma contenido XML.
	`http://www.w3.org/2002/06/xmldsig-filter2`	El contenido se debe procesar mediante esta transformación 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 `xmlTransformnType`.
`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.
`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.
`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).

12.6.1.1 Indicación de transformaciones adicionales al contenido a firmar

Respecto al uso de los parámetros `xmlTransform`***n***`Type`, `xmlTransform`***n***`Subtype` y `xmlTransformnBody`, sus valores van ligados, aceptándose las siguientes combinaciones:

- Transformación XPATH
  - Tipo: `http://www.w3.org/TR/1999/REC-xpath-19991116`
  - Subtipos: No tiene subtipos.
  - Cuerpo: Especificado mediante sentencias de tipo XPATH.
- Transformación XPATH2
  - Tipo: `http://www.w3.org/2002/06/xmldsig-filter2`
  - Subtipos:
    - `subtract`: Resta.
    - `intersect`: Intersección

        - `union`: Unión
    - Cuerpo: Especificado mediante sentencias de tipo XPATH2.
- Transformación 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 "`encodign=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=…
```

12.6.2 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.
`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.
`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).

12.7 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.

Las firmas XAdES explícitas están deprecadas en el MiniApplet y AutoFirma, y serán eliminadas en próximas versiones de estas herramientas.

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 utiliza el parámetro de configuración "`mode`" con el valor "`explicit`":

```
mode=explicit
```

Este parámetros solo tienen efecto actualmente en AutoFirma y en el MiniApplet Cliente @firma.

No se soportan firmas explícitas XAdES en los nuevos modos de operación:

- Firmas por lotes predefinidos.
- Firmas desde los clientes móviles (Android e iOS).

Dado que el modo explícito se soporta con la única finalidad de dar soporte a instalaciones obsoletas que lo estuviesen usando (con Applet o MiniApplet Cliente @firma), ningún programa o funcionalidad añadida recientemente soportará este modo, recomendándose el Uso de estructuras *Manifest* en firmas XAdES como alternativa.

El uso de esta configuración no soportada en las nuevas características del cliente puede dar lugar a resultados inesperados y la firma puede ser generada de forma explícita o no, sin necesidad de que siga siendo así en el futuro.

Si algún organismo no pudiese migrar al Uso de estructuras *Manifest* en firmas XAdES, y necesitase alguno de los nuevos modos de operación marcados como no compatibles con firmas XAdES explícitas, deberá implementar por sí mismo y de forma externa al Cliente @firma estos mecanismos. Esto implicaría:

- Utilizar en la método de la operación a realizar (firma, cofirma o firma por lote), la huella digital SHA-1 de los datos a firmar en lugar de los propios datos.
- Configurar como parámetro adicional de la operación la propiedad "mimeType", estableciendo como valor "hash/sha1".

Estas operaciones pueden realizarse desde JavaScript justo antes de la invocación de la operación de firma. Una vez hecho esto, no se debe configurar el parámetro adicional "mode".

Para conocer cómo establecer parámetros adicionales en sus operaciones de firma, consulte el apartado Paso de parámetros adicionales a los métodos de firma, cofirma y contrafirma.

13 Información específica para firma de facturas electrónicas

13.1 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.

13.2 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.

13.3 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
`signatureProductionCity`	[Texto]	Agrega a la firma un campo con la ciudad en la que se realiza la firma.
`signatureProductionProvince`	[Texto]	Agrega a la firma un campo con la provincia en la que se realiza la firma.
`signatureProductionPostalCode`	[Texto]	Agrega a la firma un campo con el código postal en donde se realiza la firma.
`signatureProductionCountry`	[Texto]	Agrega a la firma un campo con el país en el que se realiza la firma.

14 Información específica para firma de documentos en formato ODF

La operación de contrafirma no está soportada por el estándar ODF ni por el Cliente @firma.

14.1 Algoritmos de firma

El formato de firma ODF, relativo a documentos LibreOffice y OpenOffice, solo soporta el algoritmo `SHA1withRSA`.

El programa ignorará cualquier valor indicado como algoritmo de firma, utilizándose siempre `SHA1withRSA`.

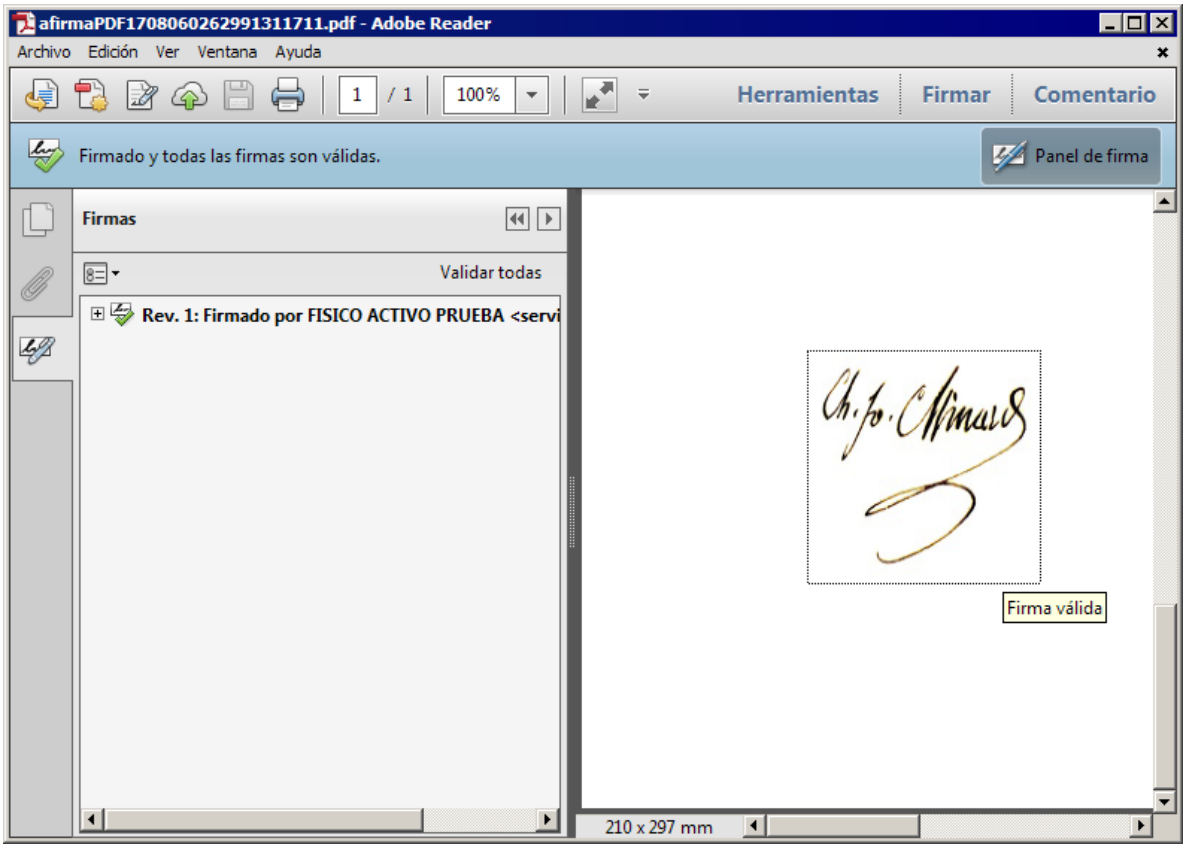
15 Información específica para firma de documentos en formato OOXML

La operación de contrafirma no está soportada por el estándar OOXML ni por el Cliente @firma.

16 Información específica para firmas PAdES

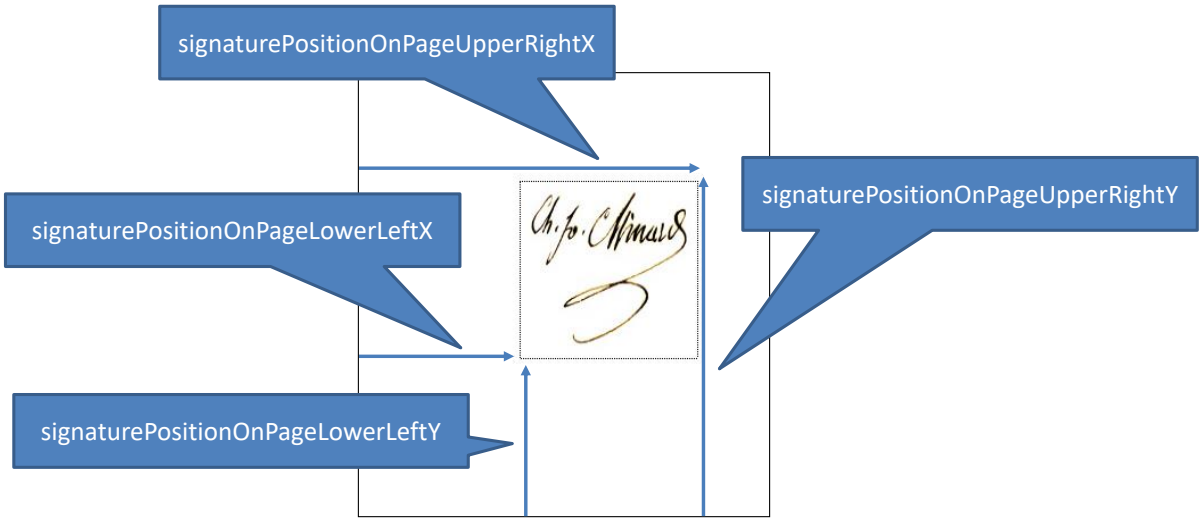
16.1 Creación de una firma visible

El MiniApplet Cliente @firma permite la creación de firmas visibles dentro de un documento PDF, que 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:

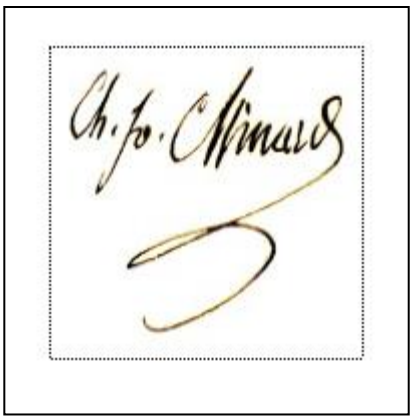


Estas coordenadas, así como la página de inserción se establecen usando los parámetros adicionales, por ejemplo:

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

Los documentos PDF comienzan su numeración de páginas desde uno (1). Si se indica -1 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 se 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.
El uso de este texto 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.

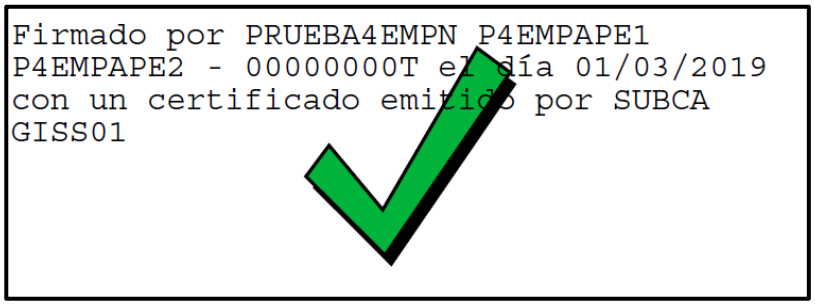
layer2FontColor

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

includeQuestionMark

Indica si debe permitirse al lector de PDF mostrar junto a la firma una marca que índique el resultado obtenido al validarla. La apariencia de esta marca depende completamente del lector de PDF utilizado y es este el que decide si se muestra o en que casos lo hace. Por ejemplo, la marca podría no mostrarse cuando se definiese una imagen de fondo en la firma.


Firmado por PRUEBA4EMPN P4EMPAPE1
P4EMPAPE2 - 00000000T el día 01/03/2019
con un certificado emitido por SUBCA
GISS01

16.2 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 "16.1", 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`.

16.3 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.
- Versiones antiguas de Adobe Acrobat/Reader no soportan múltiples firmas cuando hay firmas PAdES-BES.
- El formato PAdES sólo puede utilizarse sobre documentos PDF.
- No se firman los posibles adjuntos o empotrados que pudiese contener el documento PDF.

16.4 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/

16.5 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.

16.6 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.

includeOnlySignningCertificate (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 o `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 QPDF.
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

Configura si debe realizarse una firma de aprobación (por defecto) o certificada.

Una firma de aprobación o de formulario se realiza sobre un campo de firma de formulario del documento (preexistente o creado automáticamente en el momento de la firma). Un documento puede contener tantas firmas de aprobación como necesite. Esta es la opción común de firma.

Una firma certificada o de documento se aplica sobre un campo de firma identificado como de documento (preexistente o creado automáticamente en el momento de la firma). Un documento puede contener un único campo de este tipo y por tanto una única firma certificada. En caso de agregarse una firma certificada al documento, esta debe ser la primera que se agregue. Si hubiese alguna firma previa el resultado no sería válido.

Independientemente de sus nombres, ambos tipos de firma aplican a todo el documento (lo firman por completo), sólo cambia la designación del campo en el que se almacenan.

Una firma certificada restringe modificaciones posteriores sobre el documento. Las modificaciones permitidas vendrán determinadas por el nivel de certificación aplicado a la firma. El Cliente @firma permite configurar el nivel de certificación de una firma por medio del parámetro `certificationLevel` y un valor numérico:

- *0* = Firma sin certificar. Esta sería una firma de aprobación. Es el valor por defecto.
- *1* = Firma certificada de autor. Tras este tipo de firma certificada, no se permite ningún cambio posterior en el documento (no se pueden agregar firmas, ni rellenar formularios).
- *2* = Firma certificada de autor para formularios. Tras este tipo de firma certificada, sólo se permite el relleno de los campos de formulario (no se pueden agregar firmas).
- *3* = Firma certificada común. Tras este tipo de firma certificada, sólo se permite el relleno de los campos de formulario y la creación de firmas de aprobación.

compressPdf

Si se establece a `true` o no se indica, se comprime el PDF resultante (firmado) para que ocupe menos tamaño. Si se establece a `false`, el PDF no se comprime. Esta propiedad sólo se aplica si se trata de un PDF v4 o superior.
Igualmente, esta propiedad no tiene efecto en los PDF-A1, este tipo de PDF nunca se comprime al guardarse.

pdfVersion

Versión de PDF que se declarará en el documento de salida. Los valores que pueden declararse y la versión de PDF asociada a cada uno de ellos son los siguientes:

- 2: PDF 1.2
- 3: PDF 1.3
- 4: PDF 1.4

- 5: PDF 1.5
- 6: PDF 1.6
- 7: PDF 1.7

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 B-Level / 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`.

Este texto puede incluir una serie de palabras clave que serán sustituidas por los textos apropiados del titular o emisor del certificado de firma:

- **$$SUBJECTCN$$** Nombre común (CN, Common Name) dentro del X.500 Principal del titular del certificado de firma.
- **$$ISSUERCN$$** Nombre común (CN, Common Name) dentro del X.500 Principal del emisor del certificado de firma.
- **$$CERTSERIAL$$** Número de serie del certificado de firma.
- **$$SIGNDATE=*PATRÓN*$$** Fecha de la firma, donde *PATRÓN* debe indicar el formato en el que debe mostrarse la fecha, siguiendo el esquema definido por Oracle para la clase `SimpleDateFormat`. Así, por ejemplo, el texto "*Firmado por $$SUBJECTCN$$ el día $$SIGNDATE=dd/MM/yyyy$$.*" resultará finalmente en el PDF como "*Firmado por Tomás García-Merás el día 04/01/2016.*" suponiendo que el CN del titular del certificado de firma es *Tomás García-Merás* y que la firma se realiza el *04/01/2016*.
- **$$GIVENNAME$$** Nombre declarado del titular del certificado. Este valor podría no aparecer en el certificado, en cuyo caso, el patrón se sustituirá por cadena vacía.
- **$$SURNAME$$** Apellidos declarados del titular del certificado. Este valor podría no aparecer en el certificado, en cuyo caso, el patrón se sustituirá por cadena vacía.
- **$$ORGANIZATION$$** Organización declarada del titular en el certificado. Este valor podría no aparecer en el certificado, en cuyo caso, el patrón se sustituirá por cadena vacía.
- **$$REASON$$** Razón por la que se firma el PDF. Este valor podría no aparecer en el certificado, en cuyo caso, el patrón se sustituirá por cadena vacía.
- **$$LOCATION$$** Ciudad en la que se firma el PDF. Este valor podría no aparecer en el certificado, en cuyo caso, el patrón se sustituirá por cadena vacía.
- **$$CONTACT$$** Información de contacto del firmante del PDF. Este valor podría no aparecer en el certificado, en cuyo caso, el patrón se sustituirá por cadena vacía.

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 descargara 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

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.

17 Problemas conocidos

17.1 El MiniApplet @firma no funciona adecuadamente con las versiones de Firefox de la 42.0 a la 47.0

Firefox 42.0 y superiores sufren de un problema de compatibilidad con el entorno de ejecución de Applets de Java (Java Plugin). Este error es conocido y se encuentra documentado por Mozilla:

https://bugzilla.mozilla.org/show_bug.cgi?id=1222036

Para solventar este error, actualice a la última versión de Mozilla Firefox. Tenga en cuenta que a partir de Mozilla Firefox 49.0 es posible que se tenga que instalar los entornos de ejecución redistribuibles de Microsoft Visual C++ 2015.

Si no le es posible instalar una versión actualizada de Mozilla Firefox, deshabilite por completo el soporte de Applets de Java en Firefox (consulte la documentación de Mozilla para ello) e instale AutoFirma para realizar las firmas prescindiendo del MiniApplet Cliente @firma.

17.2 No se puede acceder al almacén de claves de Firefox 49.0 y superiores

La carga del almacén de claves y certificados de Firefox 49 y superiores por parte del MiniApplet @firma necesita que el sistema tenga instalado los entornos de ejecución redistribuibles de Microsoft Visual C++ 2015. Si no consigue acceder a sus certificados y claves privadas desde el Cliente @firma, necesitará descargar este software e instalarlo manualmente.

El entorno de ejecución redistribuible de Microsoft Visual C++ 2015 puede descargarse desde:

- https://www.microsoft.com/en-us/download/details.aspx?id=53840

Una vez en el enlace, seleccione el idioma y la arquitectura adecuada para su sistema operativo.

El proceso de instalación puede requerir permisos de administrador.

17.3 No se puede acceder al almacén de claves de Firefox 58

El navegador web Firefox basa su almacén de claves y certificados en NSS y, concretamente, en Firefox 58 se hace uso de NSS 3.34. Se han encontrado problemas de compatibilidad entre esta versión de NSS y Java que impiden que AutoFirma pueda acceder al almacén de certificados internos del navegador. Este problema está relacionado con la forma en la que Firefox 58 genera los almacenes de claves.

Un usuario que desee utilizar Firefox 58 puede, como solución parcial, instalarse una versión previa de Firefox, por ejemplo, Firefox 56, instalar sus certificados en el almacén y, a continuación, actualizar el navegador. Por defecto, el propio navegador se actualizará al reiniciarlo, así que los certificados deben instalarse en el momento de abrirlo la primera vez.

Pueden descargarse versiones anteriores de Firefox desde:

https://ftp.mozilla.org/pub/firefox/releases/

17.4 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.

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

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

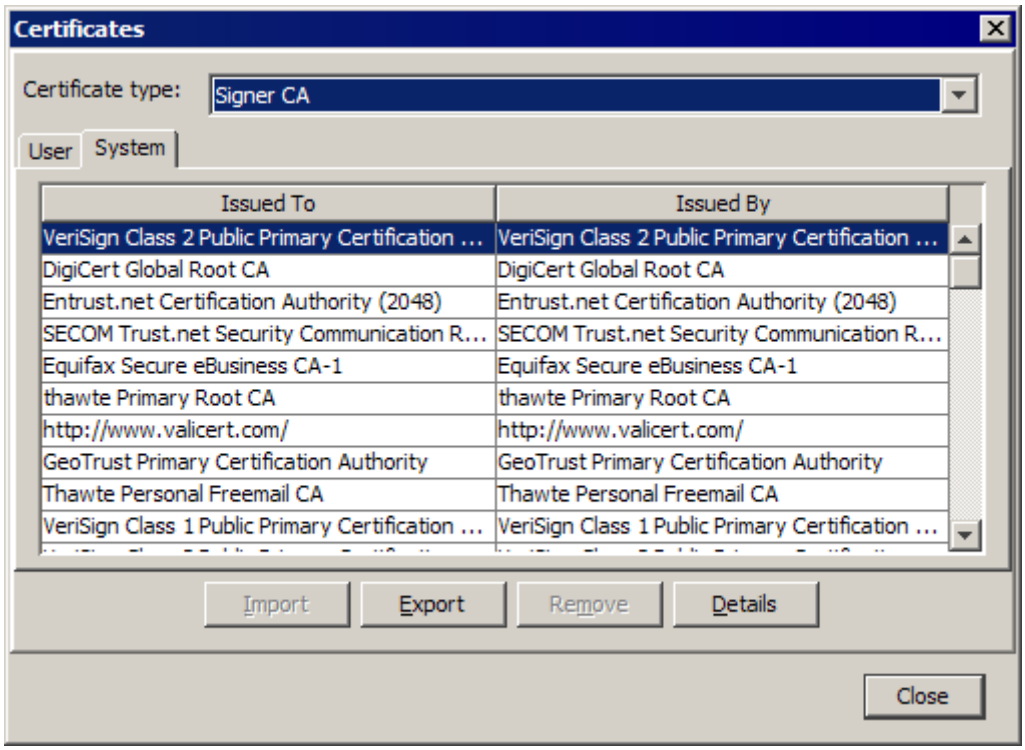
Java, a partir de la versión 7u55 obliga a los Applets a indicar el sitio Web (o al menos su dominio) en el que se van a publicar, mostrando un diálogo de advertencia a los usuarios finales si no se hace así.

Para evitar estos diálogos y agregar una medida de seguridad adicional, 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/deploy/manifest.html#CIHCFDCD
    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.

17.6 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 inteligentes, 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 (CSP, MiniDriver, etc.) de Windows instalados en el sistema del usuario.

Pruebe a actualizar tanto el 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.

17.7 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.

Puede forzar a la recarga del almacén mediante el botón de actualizar del diálogo de selección de certificados ( ). Si el cliente sigue sin detectar la tarjeta, pruebe a insertar la tarjeta antes de iniciar la operación de firma.

17.8 El MiniApplet no detecta ningún certificado bajo Mozilla / Firefox en Linux

El MiniApplet Cliente @firma, cuando se ejecuta en Linux, 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 encuentran en algún otro directorio, es posible hacérselo saber al Applet mediante el sistema indicado en el apartado Forzar ruta del almacén de Mozilla Firefox.

17.9 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.

17.10 El servicio de firma trifásica genera un error al realizar firmas XAdES en servidores JBoss

A partir de determinada versión del servidor de aplicaciones JBoss (7 / EAP 6), 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:** 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 este JAR, los por defecto de Java, en lugar de a las bibliotecas del servidor de aplicaciones.

- **Solución 3:** Si todo lo anterior fracasase, pero supiésemos que ninguna otra aplicación hace uso de estas bibliotecas del servidor de aplicaciones, podríamos sustituirlas por la versión 1.4.6 de Xerces y sus dependencias. De esta forma, se podría ejecutar la operación de firma, aunque varias funcionalidades de JBoss relacionadas con los despliegues seguros conforme a la arquitectura definida por RedHat podrían verse afectados.

17.11 Las firmas con DNIe requieren que se introduzca el PIN del DNIe por cada operación de firma

El DNIe y los controladores que le dan soporte están desarrollados conforme a diversas normativas de seguridad, entre ellas, la norma europea EN14890. Esta norma define la necesidad de que el PIN del DNIe se presente ante cada una de las operaciones de firma.

Cualquier aplicación que no solicite el PIN de su DNIe por cada operación de firma viola esta norma de seguridad y pone en peligro sus datos ya que esa aplicación estaría almacenando, al menos temporalmente, el PIN de su DNIe.

Por seguridad, el MiniApplet y AutoFirma solicitarán el PIN de su DNIe por cada operación de firma, incluso en aquellos casos en los que se ejecute un proceso de firma masiva.

Tenga en cuenta que la contrafirma de un documento con múltiples firmas puede implicar firmar varias veces, aunque sólo se genere una única firma electrónica. Así pues, este tipo de firmas pueden requerir que el usuario inserte varias veces el PIN de su DNIe.

Pueden consultar más información acerca del DNIe en el siguiente enlace:

http://www.dnielectronico.es/PortalDNIe/

17.12 No aparecen los certificados de las tarjetas de la FNMT o empleado público al ejecutar el Cliente @firma desde Firefox

El Cliente @firma permite el uso de las claves y certificados ubicados en tarjetas de la FNMT o tarjetas de empleado público. Sin embargo, debido a un problema de compatibilidad con el controlador PKCS#11 de la tarjeta, no es posible acceder a sus claves después de recargar el almacén de claves o extraer y volver a insertar la tarjeta.

Si no le aparecen sus claves de la tarjeta de FNMT al ejecutar el Cliente @firma desde Firefox, asegúrese de que la tarjeta no se ha extraído o se ha forzado a la recarga del almacén mediante el botón actualizar del diálogo de selección de certificados ( ). Inserte su tarjeta antes de iniciar el MiniApplet o AutoFirma y compruebe que el propio navegador es capaz de acceder a los certificados de su tarjeta.

17.13 Error en la firma con tarjetas de la FNMT y empleado público desde Firefox

Debido a un problema de compatibilidad con el controlador PKCS#11 de las tarjetas de la FNMT, se sabe que en determinados casos la firma con los certificados de tarjetas produce un error. Este caso es común en las firmas posteriores a la primera con AutoFirma (comunicación con sockets). También ocurre que, si se realizan firmas consecutivas de forma inmediata, todas suelen terminar correctamente. En cambio, si existe al menos una pequeña espera entre ellas, el controlador no es capaz de acceder a las claves y falla la operación de firma.

Así mismo, se han encontrado diferencias entre el comportamiento del controlador PKCS#11 y el controlador CSP para el almacén a la tarjeta a través del almacén de Windows, e incluso entre las implementaciones 32 y 64 bits de los propios controladores. Según el controlador utilizado puede ocurrir que se pida el PIN de la tarjeta por cada operación de firma o que sólo se pida una única vez para todas las operaciones.

Este problema carece de solución actualmente. Se recomienda al usuario que al hacer uso de las tarjetas de la FNMT utilice los navegadores Chrome, Internet Explorer o Edge. En caso de necesitar o preferir el uso de Firefox, se recomienda instalar la versión de AutoFirma de 32 bits para minimizar el el problema.

17.14 Se solicita confirmar la operación de firma cuando se seleccionan algunos certificados de una tarjeta de la FNMT

El controlador de las tarjetas de la FNMT determina que, al firmar con algunos certificados, el usuario debe dar su consentimiento expreso, para lo cual le muestra un mensaje solicitando su autorización. Este comportamiento puede resultar confuso por ocurrir sólo con algunos certificados de la tarjeta, pero no es posible modificarlo desde el Cliente @firma.

17.15 El Cliente @firma no muestra en OS X el título de los diálogos de cargar y guardado de ficheros

Las nuevas versiones de OS X omiten el título de los diálogos de carga y guardado de ficheros. En caso en que el integrador delegue en el Cliente @firma la selección y el guardado de las firmas generadas, debería tener la precaución de informar al usuario de esto para que en todo momento sepa qué operación está realizando (carga de un fichero de datos para firma, carga de un fichero de firma para cofirma/contrafirma, guardado de una firma generada…).

17.16 Error al cargar el listado de certificados después del cambio en caliente del almacén por defecto

Se ha detectado que después de haber cargado el almacén del sistema en Windows (realizando una operación de firma, por ejemplo) puede producirse un error al cargar el almacén de Mozilla Firefox después de usar el método `setKeyStore` para cambiar entre ellos. Este error se debe a que al realizar el cambio en caliente no se han cargado correctamente las dependencias necesarias del almacén de Mozilla.

Este problema no tiene solución actualmente.

17.17 AutoFirma no puede comunicarse con el navegador en macOS

En algunos casos la instalación de AutoFirma en macOS finaliza sin errores pero no se instala el perfil de seguridad que permiten que AutoFirma se comunique de forma segura con el navegador web. En estos casos, al realizar una operación de firma se arrancará correctamente AutoFirma pero

este no será capaz de transmitir el resultado de la firma al navegador web. Esto puede generar un error del navegador con el texto “No se ha podido conectar con AutoFirma.”.

Para solventar este problema será necesario configurar manualmente la confianza en los certificados de AutoFirma.

Para ello:

1. Acceda a la aplicación “Acceso a llavero”.
2. Acceda al llavero “SISTEMA” y a la opción “Certificados”.
3. En el listado de certificados mostrados deben aparecer los certificados “127.0.0.1” y “AutoFirma ROOT”. Si el icono que aparece junto a estos muestra el signo ‘+’, se confía en los certificados y la comunicación con AutoFirma debería funcionar correctamente. Si no, continue con el proceso.
4. Haga clic sobre el certificado “127.0.0.1” y pulse en la opción “Confiar”.
5. En el diálogo que debe haber aparecido, despliegue el listado “Al utilizar este certificado” y seleccione la opción “Confiar siempre”.
6. Repita los pasos 4 y 5 para el certificado “AutoFirma ROOT”.
7. Compruebe que en ambos certificados aparece ahora el símbolo ‘+’ junto a su icono.
8. Cierre la ventana de los llaveros.
9. Introduzca la contraseña de su usuario en el diálogo para confirmar el cambio en la configuración de seguridad.

17.18 Sólo se realiza la firma del primer documento de una serie cuando se realizan las firmas desde Google Chrome

El navegador Google Chrome dispone de un mecanismo de seguridad en base al cual no permite realizar más de una llamada a una URL externa ante una única interacción del usuario, como la pulsación de un botón, por ejemplo. Esta limitación aplica a la llamada al Cliente @firma, tanto a AutoFirma en entornos de escritorio como a Cliente de firma móvil en entornos Android, por lo que sólo se podrá abrir la aplicación una sóla vez ante una única operación del usuario.

La realización de firmas en serie con el cliente se debe realizar siempre invocando una operación de firma una vez ha terminado la anterior (comúnmente desde las funciones callback que notifican el final de una firma). En el caso por defecto de AutoFirma esta limitación no afecta al usuario, ya que sólo se invoca la aplicación la primera vez y se le solicitan firmar los distintos documentos a través de un socket.

Sin embargo, la limitación impuesta por Google Chrome sí afecta a AutoFirma cuando se fuerza que a comunicación entre AutoFirma y el navegador se realice mediante servidor intermedio (mediante el uso de la sentencia “`MiniApplet.setForceWSMode(true)`”) y afectará siempre al uso del Cliente de Firma Android. En estos casos, la aplicación se abrirá para procesar la primera firma de la serie, pero se ignorarán las llamadas para procesar los siguientes documentos.

Cuando se deba realizar la firma de múltiples documentos simultáneamente, se recomienda aplicar las siguientes directrices:

- Utilizar el modelo de firma de lotes de documentos.
- Nunca forzar la comunicación por servidor intermedio.
- Recomendar a los usuarios de Android el uso de un navegador distinto a Google Chrome.

Esta limitación está relacionada con la descrita en el apartado "17.19 No se abre la aplicación de firma al realizar la firma desde Google Chrome".

17.19 No se abre la aplicación de firma al realizar la firma desde Google Chrome

El navegador Google Chrome dispone de un mecanismo de seguridad en base al cual no permite realizar una llamada a una URL externa pasados unos pocos segundos entre la interacción del usuario con la página, como la pulsación de un botón, y la propia llamada. Esta limitación aplica a la llamada al Cliente @firma, tanto a AutoFirma en entornos de escritorio como a Cliente de firma móvil en entornos Android, por lo que la aplicación no se abrirá si transcurre demasiado tiempo entre la solicitud de firma del usuario y la llamada a la aplicación.

Esto no debe afectar al uso de AutoFirma desde Chrome ya que, por defecto, la llamada a la aplicación se realiza de forma inmediata y los datos se transmiten a través de un socket.

Sin embargo, la limitación impuesta por Google Chrome sí puede afectar AutoFirma cuando se fuerza que a comunicación entre AutoFirma y navegador se realice mediante servidor intermedio (mediante el uso de la sentencia "`MiniApplet.setForceWSMode(true)`") y en cualquier caso al uso del Cliente de Firma Android. En estos casos, si los datos no pueden enviarse a través de la URL de invocación a la aplicación, deberán subirse primeramente al servidor intermedio para que después la aplicación de firma los descague. Cuando esta subida de los datos dure más de unos pocos segundos, la aplicación de firma no se abrirá.

Por regla general, para evitar los problemas derivados de esta restricción de navegador Chrome, se deberían seguir las siguientes sugerencias:

- Utilizar el modelo de firma trifásica con un DocumentManager a medida (no con el por defecto), para que los datos a transmitir entre el navegador y la aplicación sean siempre pequeños.
- Nunca forzar la comunicación por servidor intermedio.
- Recomendar a los usuarios de Android el uso de un navegador distinto a Google Chrome.

Esta limitación está relacionada con la descrita en el apartado "17.18 Sólo se realiza la firma del primer documento de una serie cuando se realizan las firmas desde Google Chrome".

Esta obra está bajo una licencia Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 Unported.