Manual del integrador del Cliente @firma v1.7

Índice de contenidos

1 Objeto del documento 6
2 Introducción 7
2.1 Licencia 8
2.1.1 Licencias Java 8
2.2 Recursos 8
2.3 Adecuación al Esquema Nacional de Seguridad 9
3 Requisitos mínimos 10
3.1 Entorno Cliente 10
3.1.1 AutoFirma 10
3.1.2 Cliente móvil Android 11
3.1.3 Cliente móvil iOS 11
3.2 Entorno Servidor 11
4 Operaciones soportadas 12
5 Despliegue del Cliente @firma 13
5.1 Información preliminar 13
5.2 Importación del JavaScript de despliegue 14
5.2.1 Importación en páginas Web generadas dinámicamente 15
5.3 Servicios del Cliente @firma 15
5.3.1 Servicios auxiliares de comunicación 15
5.3.2 Servicios de firma trifásica y firma de lotes 18
5.3.3 Configuración de los servicios 21
5.4 Configuración del *Content Security Policy* 22
6 Uso del Cliente @firma 24
6.1 Carga de la aplicación 25
6.1.1 Configuración de los servicios auxiliares de comunicación 25
6.1.2 Forzado de la comunicación a través de los servicios auxiliares 26
6.1.3 Restricción según desfase horario con el servidor 27
6.1.4 Selección del almacén de claves 29
6.2 Firma electrónica 33

6.2.1 Firma de múltiples documentos (firma masiva) 36
6.2.2 Firma trifásica 38
6.3 Cofirma electrónica 39
6.4 Contrafirma electrónica 41
6.4.1 Selección de nodos 43
6.5 Firma por lotes predefinidos 45
6.5.1 Creación de los lotes 46
6.5.2 Respuesta a una ejecución de un lote 53
6.5.3 Creación de una lógica de guardado personalizada 55
6.6 Selección de certificado 57
6.7 Recuperación de log 58
6.8 Operaciones de gestión de ficheros 59
6.8.1 Guardado de datos en disco 60
6.8.2 Selección y recuperación de un fichero por parte del usuario 62
6.8.3 Selección y recuperación de múltiples ficheros por parte del usuario 64
6.9 Métodos de utilidad JavaScript 66
6.9.1 Conversión de una cadena Base64 a texto 66
6.9.2 Conversión de un texto a cadena Base64 67
6.9.3 Descarga de datos remotos 67
7 Configuración de las operaciones 69
7.1 Paso de parámetros adicionales 69
7.2 Configuración del filtro de certificados 70
7.3 Selección automática de certificados 76
7.4 Configuración de la política de firma 76
7.4.1 Política de firma de la AGE v1.9 77
7.4.2 Política de firma de Factura electrónica (Facturae) 78
7.5 Validación de firmas previas 78
8 Formatos de firma 80
8.1 Configuración de firmas CAdES 81
8.1.1 Algoritmos de firma 81

8.1.2 Firmas CAdES implícitas o explícitas 82
8.1.3 Parámetros adicionales 82
8.2 Configuración de firmas XAdES 86
8.2.1 Algoritmos de firma 87
8.2.2 Algoritmos de huella digital para las referencias 87
8.2.3 Situación del nodo de firma en XAdES Enveloped 88
8.2.4 Transformaciones sobre el contenido a firmar 89
8.2.5 Uso de estructuras *Manifest* en firmas XAdES 90
8.2.6 Tratamiento de las hojas de estilo XSL de los XML a firmar 93
8.2.7 Parámetros adicionales 93
8.3 Configuración de firmas PAdES 104
8.3.1 Algoritmos de firma 105
8.3.2 Operaciones no soportadas y notas de interés 105
8.3.3 Creación de una firma visible 105
8.3.4 Inserción de una imagen en un documento PDF antes de ser firmado 110
8.3.5 Firma de documentos PDF cifrados o protegidos con contraseña 111
8.3.6 Documentos certificados 111
8.3.7 Parámetros adicionales 112
8.4 Configuración de firmas de factura electrónica 117
8.4.1 Operaciones no soportadas y notas de interés 117
8.4.2 Algoritmos de firma 117
8.4.3 Parámetros adicionales 117
9 Compatibilidad con dispositivos móviles y AutoFirma 120
9.1 Requisitos de despliegue 120
9.1.1 Compatibilidad con Google Chrome 121
9.2 Limitaciones 121
9.2.1 Limitaciones de formato 121
9.2.2 Limitaciones funcionales 122
9.3 Notificaciones al usuario 123
10 Problemas conocidos 125

ANEXO I. Comunicación JavaScript de despliegue – Cliente @firma 132

I.1. Comunicación por sockets 132

I.2. Comunicación por servidor intermedio 133

ANEXO II. Firma trifásica 135

II.1. Servicio de firma trifásica 136

II.1.1. Configuración del gestor de documentos del servicio 136

II.1.2. Desarrollo de un gestor de documentos personalizado 138

II.1.3. Configuración de un gestor de documentos personalizado 140

II.2. Realizar firma trifásica con el Cliente @firma 140

II.3. Configuración del gestor de documentos "`FileSystemDocumentManager`" 141

II.3.1. Parámetros de uso y descripción del funcionamiento 141

II.3.2. Configuración en alta disponibilidad con varios nodos 142

1 Objeto del documento

En el presente documento se detalla el proceso de integración y configuración del Cliente @firma para la generación de firmas de usuario en trámites web.

2 Introducción

El Cliente @firma es una herramienta de firma electrónica que permite a sus usuarios generar firmas con sus certificados locales. El Cliente está especialmente orientado a ser integrado dentro de trámites web de tal forma que una aplicación web pueda solicitar al usuario la firma de unos datos a través del Cliente y obtener como respuesta la firma de esos datos.

El Cliente @firma está formado principalmente por dos componentes:

- Una aplicación nativa (AutoFirma, para equipos es de sobremesa; el Cliente móvil Android, para dispositivos Android; o el Cliente móvil iOS, para dispositivos iOS). Esta aplicación debe estar instalada en el dispositivo del usuario antes de iniciar el proceso de firma.

- Un JavaScript de despliegue para la integración del proceso de firma dentro del trámite web.

Además de los componentes mencionados, el uso de determinadas funciones del Cliente @firma o su compatibilidad con determinados entornos puede requerir el despliegue de diversos servicios auxiliares a los que deberá conectar la aplicación cliente.

El Cliente @firma hace uso de los certificados digitales X.509v3 y de las claves privadas asociadas a estos que estén instalados en el repositorio o almacén de claves y certificados (*KeyStore*) del sistema operativo o del navegador Web (Internet Explorer, Mozilla Firefox, etc.) del usuario. También se permite el uso de certificados en dispositivos criptográficos (tarjetas inteligentes, dispositivos USB) instalados en el sistema y con controlador CSP/PKCS#11 compatible (como, por ejemplo, el DNI Electrónico o DNIe) y certificados en almacenes software (PKCS#12/PFX).

El Cliente @firma, como su nombre indica, es una aplicación que se ejecuta en cliente. Es decir, 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 (almacén software o tarjeta) ubicado en su dispositivo.

El Cliente @firma no almacena ningún tipo de información personal del usuario, ni hace uso de cookies ni ningún otro mecanismo para la gestión de datos de sesión. AutoFirma sí almacena el log de su última ejecución a efectos de ofrecer soporte al usuario si se encontrase algún error durante su ejecución.

El Cliente @firma forma parte de la *suite* de productos de @firma, pero no interacciona con ninguno de los servicios del resto de productos. El Cliente @firma sólo generar firmas electrónicas con los certificados de usuario. La validación de estas firmas y su promoción a firmas longevas, por ejemplo, son operaciones que deberán realizarse de forma independiente con otros productos de la *suite* (@firma, VALIDe, Integr@...)

2.1 Licencia

El Cliente @firma es software libre y puede usarse, según se desee, bajo licencia *GNU General Public License* versión 2 (GPLv2) o bajo licencia *European Software License* 1.1 (EUPL 1.1) o superior.

El Cliente @firma incluye, entre otros, los siguientes productos de terceros con licencias compatibles:

- 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 Java Mime Magic Library (http://jmimemagic.sourceforge.net/)
- Apache Santuario (https://santuario.apache.org/)
- Proxy Vole (https://github.com/MarkusBernhardt/proxy-vole)
- Java WebSocket (https://github.com/TooTallNate/Java-WebSocket)

2.1.1 Licencias Java

AutoFirma, en sus versiones para Windows y macOS, incluye una máquina virtual de Java (JVM) para la ejecución de la aplicación.

- Microsoft Windows:
  - OpenJDK JRE 11.0.6 (licencia GPL v2)
- macOS
  - OpenJDK JRE 11.0.6 (licencia GPL v2)

Las licencias de las JVM incluidas en AutoFirma permiten su libre uso por parte de todos los usuarios.

En caso de utilizar el antiguo MiniApplet (disponible hasta el Cliente @firma 1.6.5), la entidad encargada de la implantación de la solución será la responsable de la proporcionar la JVM necesaria a sus empleados y las licencias que esta pueda requerir. En el caso de los ciudadanos o usuarios que no lo utilicen en el desempeño de su actividad laboral, no se requerirá el pago de licencias independientemente de que se utilice OpenJDK u Oracle Java.

Los Clientes móviles para Android e iOS utilizan únicamente el entorno de ejecución proporcionado por el del sistema y no requieren licencia.

2.2 Recursos

Puede consultar la información relativa al proyecto Cliente @firma y descargar el código fuente y los binarios de la aplicación en la siguiente dirección Web:

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

Así mismo, el código del Cliente se encuentra disponible en GitHub y sus distintos módulos se encuentran disponibles en el repositorio central de Maven:

- Fuentes: https://github.com/ctt-gob-es/clienteafirma
- Binarios: https://search.maven.org/search?q=es.gob.afirma

2.3 Adecuación al Esquema Nacional de Seguridad

Los productos de la Suite @firma pueden permitir el uso de algoritmos 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. 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

3 Requisitos mínimos

3.1 Entorno Cliente

Los requisitos mínimos del entorno cliente dependerá de la aplicación nativa utilizada para firmar. Debido a la amplia variedad de navegadores del mercado, sólo se citarán para cada aplicación cliente aquellos navegadores para los que se han realizado pruebas específicas. No excluye esto que pueda funcionar correctamente con otros entornos.

3.1.1 AutoFirma

El uso de AutoFirma como herramienta de firma integrada dentro del proceso de firma de trámites web tiene los siguientes requerimientos en cuanto a entorno operativo:

- Sistema Operativo
  - Microsoft Windows 7 o superior.
    - Soportado directamente en 7, 8, 8.1 y 10.
    - En 32 o 64 bits.
  - Linux.
    - Ubuntu, Fedora, OpenSUSE.
  - Apple OS X Yosemite (10.10) o superior.
    - Soportado directamente en Yosemite, El Capitán, Sierra, High Sierra, Mojave, Catalina y Big Sur.
- Navegadores Web (Cuando es invocada desde una aplicación web)
  - Microsoft Windows
    - Google Chrome 46 o superior.
    - Mozilla Firefox 41.0.1 o superior.
    - Microsoft Internet Explorer 8 o superior.
    - Microsoft Edge Legacy v20 o superior (EdgeHTML).
    - Microsoft Edge (Edge Chromium).
  - Linux
    - Mozilla Firefox 41.0.1 o superior.
  - Apple OS X
    - Apple Safari 9.0 o superior.
    - Google Chrome 46 o superior.
    - Mozilla Firefox 41.0.1 o superior.

En entornos macOS y Windows no es necesario que el usuario tenga instalado un entorno de ejecución de Java, ya que viene incluido en la propia aplicación. En Linux se necesita un entorno de ejecución de Java de Oracle OpenJDK (marcado como dependencia en el instalador integrado de AutoFirma) versión 8 o superior.

Para el acceso a Firefox 49 y superiores en Windows puede ser necesario instalar los entornos de ejecución redistribuibles de Microsoft Visual C++ 2015. Si AutoFirma no puede cargar los certificados

de su almacén de claves, siga las instrucciones descritas en el apartado de errores conocidos No se puede acceder al almacén de claves de Firefox 49.0 y superiores.

3.1.2 Cliente móvil Android

El uso de Cliente móvil Android como herramienta de firma integrada dentro del proceso de firma de trámites web tiene los siguientes requerimientos en cuanto a entorno operativo:

- Sistema Operativo
  - Android 4.3 o superior.
- Navegadores Web (para la invocación por protocolo)
  - Google Chrome.
  - Navegador webkit.

3.1.3 Cliente móvil iOS

El uso de Cliente móvil iOS como herramienta de firma integrada dentro del proceso de firma de trámites web tiene los siguientes requerimientos en cuanto a entorno operativo:

- Sistema Operativo
  - iOS 9 o superior.
- Navegadores Web (para la invocación por protocolo)
  - Apple Safari.

3.2 Entorno Servidor

El Cliente @firma requiere de una serie de servicios auxiliares para el uso de ciertas funcionalidades y la compatibilidad con determinados entornos. Los requisitos de estos servicios son los siguientes:

- Servidor de aplicaciones JEE compatible con Servlets de Java.
  - Apache Tomcat, WildFly, RedHat JBoss, IBM WebSphere, Oracle GlassFish, Oracle Application Server, etc.
- JRE 1.7 o 1.8 (Se recomienda el uso de la última versión disponible de Java 1.8).

Puede saber más acerca de los servicios auxiliares del Cliente @firma, consulte el apartado 5.3 Servicios del Cliente @firma.

4 Operaciones soportadas

El Cliente @firma proporciona funcionalidades de firma electrónica (incluyendo firmas múltiples) con certificados locales, pero no otras operaciones de firma o criptografía como validación de firmas, promoción a formatos longevos, sellado de tiempo, creación de sobres digitales o cifrado. Adicionalmente, el Cliente @firma proporciona un conjunto de métodos de utilidad y opciones de operación.

Las operaciones soportadas por el Cliente @firma son:

- Firma electrónica.
- Firmas electrónicas múltiples (más de un firmante por documento).
    - Cofirma.
    - Contrafirma.
- Firma de lotes de documentos.
- Selección de certificado.
- Funciones de utilidad:
    - Conversión de una cadena Base64 a texto.
    - Conversión de un texto a una cadena Base64.
    - Guardado de datos en disco.
    - Carga de fichero local.
    - Carga de multiples ficheros.

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

5 Despliegue del Cliente @firma

Para integrar el Cliente @firma en su aplicación web se debe publicar junto a la misma el fichero "`autoscript.js`". En este fichero se encuentra el objeto JavaScript que deberemos utilizar para invocar a las distintas operaciones del Cliente y obtener su resultado.

Adicionalmente y según el entorno de ejecución del usuario, es posible que para la comunicación entre el JavaScript de despliegue y el Cliente @firma sea necesario el uso de dos servicios de comunicación. Estos dos servicios, se distribuyen junto al JavaScript de despliegue en forma de archivos WAR ("`afirma-signature-retriever.war`" y "`afirma-signature-storage.war`") y deberían desplegarse en el mismo dominio que la página web desde la que se use el Cliente. También hay funcionalidades del Cliente @firma que requieren el despliegue de un servicio adicional ("`afirma-server-triphase-signer.war`").

Consulte en el apartado 5.3 Servicios del Cliente @firma en qué casos son necesarios estos servicios y así determinar si debe realizar su despliegue.

5.1 Información preliminar

En este apartado se presentan distintos aspectos que el integrador del Cliente @firma debe tener en cuenta antes del despliegue del Cliente @firma:

- Como medida de seguridad, AutoFirma no permite el despliegue en páginas a las que se acceda mediante "`127.0.0.1`" o "`localhost`". Si va a desarrollar en realizar pruebas en su equipo local, deberá tomar alguna de las siguientes alternativas:
    - Acceda a su página y a los servicios del cliente a través de la IP de red que tenga asignada su equipo.
    - Configure en el fichero `hosts` de su equipo un alias para "`127.0.0.1`" y utilícelo como nombre de dominio. La localización del fichero `hosts` según el sistema operativo es:
        - En Windows: `UNIDAD:\\Windows\System32\drivers\etc\`
        - En Linux y macOS: `/etc/hosts`
- La página web, el JavaScript de despliegue y los servicios del Cliente @firma deben ser accesibles desde el mismo dominio. De esta forma se evitarán errores debido a los mecanismos de seguridad del navegador para bloquear ataques de *Cross-Site Scripting* (XSS). Esto es especialmente importante cuando se hace uso de los servicos auxiliares de comunicación (`StorageService` y `RetrieveService`).
    - Despliegue una instancia de los servicios auxiliares de comunicación por cada aplicación en la que integre el cliente o, al menos, por cada dominio.

- Siempre que se utilice el Cliente @firma ha de tenerse en cuenta que la aplicación nativa debe poder confiar en los certificos SSL utilizados por cualquier servicio externo al que se deda conectar. En caso contrario, fallaría la conexión. Esto es crítico cuando se hace uso de la comunicación por servidor intermedio o se utiliza la operación de firma trifásica.
    - En el caso de usar AutoFirma, puede evitar problemas durante el desarrollo y pruebas desactivando la comprobación de los certificados SSL. Puede hacer esto desde el panel de preferencias, en la pestaña "`General`", desactivando la casilla de verificación "`Aceptar sólo conexiones con sitios seguros (Recomendado)`".
    - En el caso de aplicaciones de firma móvil, es imprescindible que se utilicen certificados SSL de confianza, vigentes y expedidos para el dominio en cuestión.
- Codificación UTF-8 de las páginas cuando se proporcionen o recojan textos del cliente.
    - El Cliente @firma interpreta todos los textos, tanto los recibidos como los devueltos en las respuestas, usando el juego de caracteres UTF-8. Para poder transmitirlos y mostrarlos correctamente desde una página web es necesario que esta se encuentre codificada en UTF-8 y lo declare como tal.
    - En caso de no ser posible, se recomienda:
        - Que el Base64 de los textos a proporcionar al Cliente se hayan obtenido desde un entorno en el que se pueda gantizar que originalmente estaban codificados en UTF-8. Por ejemplo, que el texto ya estuviese previamente codificado o que se codifique a través de un servicio.
        - No mostrar directamente al usuario los mensajes devueltos por el propio Cliente.

5.2 Importación del JavaScript de despliegue

Para integrar el Cliente @firma en su página Web debe importar en ella la biblioteca JavaScript "`autoscript.js`". Puede hacer referencia a la misma mediante una URL absoluta o mediante una URL relativa a partir de la dirección de publicación de su página Web.

Por ejemplo, se puede introducir la carga de la biblioteca en la sección `head` del HTML, tal y como se muestra en el siguiente ejemplo:

```
…
<head>
  <script src="https://miweb.com/afirma/js/autoscript.js"></script>
…
```

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

```
…
<head>
  <script src="js/autoscript.js"></script>
…
```

Cualquier página Web con esta biblioteca JavaScript importada está lista para utilizar el Cliente @firma.

5.2.1 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 mediante alguna de las muchas tecnologías disponibles de aplicaciones Web (JSP, ASP, PHP, etc.).

En estos casos es necesario tener en cuenta que la dirección de la biblioteca JavaScript debería establecerse en base a la URL de despliegue de la página si se hace de forma relativa. En caso de duda, utilice la URL absoluta.

5.3 Servicios del Cliente @firma

La compatibilidad del Cliente @firma con determinados entornos y funciones puede requerir el despliegue de una serie servicios auxiliares. Estos servicios se distribuyen en forma de archivos WAR junto al JavaScript de espliegue del Cliente @firma y deberían desplegarse por cada aplicación que desee utilizarlos.

Los archivos WAR en los que se distribuyen estos servicios no requieren el uso de un software servidor de aplicaciones concreto. Consulte el apartado 3.2 Entorno Servidor para saber más de los requisitos de despliegue y la documentación de su servidor de aplicaciones para saber cómo desplegarlos.

En caso de que su aplicación no tenga que ser compatible con los entornos o las funcionalidades listadas a continuación, no será necesario el despliegue de ningún servicio. Esto es, que no tendrá que desplegarlos en los siguientes casos:

- Si no se requiere compatibilidad con Internet Explorer 10 o anterior o con Safari 10.
- Si no se requiere compatibilidad con dispositivos móviles.
- Si no se utilizan operaciones de firma trifásica o firma de lotes.

Tenga en cuenta que los distintos servicios utilizan distintos ficheros de configuración, pero que la lógica para localizar y tratar estos ficheros es la misma para todos ellos. Este aspecto de la configuración se detalla en el apartado 5.3.3 Configuración de los servicios.

5.3.1 Servicios auxiliares de comunicación

Para la comunicación entre el JavaScript de despliegue del Cliente @firma y la propia aplicación de firma se utilizan diversos mecanismos. Uno de estos requiere del uso de dos servicos auxiliares de

comunicación. Estos servicios se distribuyen junto con el JavaScript de despliegue del Cliente @firma y son los siguientes:

- **`StorageService`**
  - Este Servlet permite almacenar datos en un directorio temporal del servidor.
  - Este servicio se despliega por medio del WAR "`afirma-signature-storage.war`".
- **`RetrieveService`**
  - Este Servlet permite recuperar datos de un servidor. Los datos devueltos deben estar almacenados en un directorio temporal predefinido y, tras devolver los datos, el servicio borrará 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.
  - Este servicio se despliega por medio del WAR "`afirma-signature-retriever.war`".

Los servicios de almacenamiento y recuperación en servidor son necesarios en los siguientes casos:

- Cuando el navegador del usuario sea Internet Explorer 10 o anterior (o una versión superior en modo compatibilidad con estas versiones).
- Cuando el usuario utilice un dispositivo móvil y los clientes móviles Android o iOS.
- Cuando la aplicación fuerce intencionadamente la comunicación a través de estos servicios.

En el resto de casos, el JavaScript de despliegue y AutoFirma se comunicarán a través de Sockets. Consulte el apartado ANEXO I Comunicación JavaScript de despliegue – Cliente @firma para saber más sobre los mecanismos de comunicación entre el JavaScript de despliegue y el Cliente @firma.

> **Importante:** Los servicios de almacenamiento y guardado en servidor deben ser accesible desde el mismo dominio en el que se encuentre la página de firma. Si no se hiciese así, el navegador web puede bloquear la conexión con ellos interpretando que se trata de un ataque de *cross-site scripting (XSS)*.

5.3.1.1.1 Configuración de los servicios de almacenaje y recuperación

Los servicios de almacenaje y recuperación sirven para comunicar el JavaScript de despliegue y el Cliente @firma mediante el guardado temporal de los datos en un directorio concreto del servidor.

Este directorio temporal debe ser visible y accesible por todas las instancias en ejecución de los servicios. Este aspecto es especialmente importante en configuraciones de servidores de aplicaciones en alta disponibilidad, donde puede haber varios nodos que presten el servicio, cada uno de ellos en un sistema de ficheros diferente.

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.

Los servicios de almacenaje y recuperación de datos para la comunicación entre el JavaScript de despliegue y el Cliente @firma (`StorageService` y `RetrieveService`, respectivamente), utilizan el mismo fichero de configuración. Este fichero se llama "`intermediate_config.properties`".

Las propiedades disponibles en este fichero de configuración son las siguientes:

- **`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 si estos se desplegasen por separado.
    - A este directorio sólo necesitan acceder los servicios de guardado y recuperación de datos, por lo que el administrador del sistema puede determinar que sólo estos servicios pueden acceder a dicho directorio.
    - En caso de realizarse un despliegue en múltiples nodos, el directorio debería encontrarse en una unidad compartida por todos ellos.
    - 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. Una vez superado ese tiempo desde la creación del fichero, el servicio de recuperación se negará a devolverlo y lo eliminará.
    - Si no se configura esta propiedad, se usará por defecto el valor "`60000`" (1 minuto)
- **`maxFileSize`**: Es el tamaño máximo de fichero permitido expresado en bytes. Su utilidad responde principalmente a motivos de seguridad, para evitar que el directorio del servidor se quede sin espacio si comienzan a subirse datos de gran tamaño.
    - Si no se configura esta propiedad, se usará por defecto el valor "`0`", que indica que no hay límite de tamaño de fichero.
- **`debug`**: Habilita el modo debug cuando se configura el valor "`true`". El modo debug sólo debería habilitarse durante la fase de integración y nunca en entornos productivos. En este modo:
    - Se muestran traza de log adicionales.
    - No se eliminan los ficheros recuperados del servicior intermedio.
    - No se eliminan los ficheros caducados del directorio temporl del servidor intermedio.
    - No se limita el tamaño máximo de los ficheros a guardar.

Un ejemplo de fichero de configuración podría ser:

```
# Directorio para el guardado de los ficheros. Por defecto: Directorio temporal
tmpdir=C:/clienteafirma/temp

# Tiempo de caducidad de los mensajes. Por defecto: 60000 (1 minuto)
expTime=60000

 # Tamano maximo de fichero en bytes. Por defecto: 0 (Sin limite)
maxFileSize=1048576
```

5.3.1.1.1.1 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 más que la limitación del tamaño de fichero, 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.

5.3.2 Servicios de firma trifásica y firma de lotes

Junto al JavaScript de despliegue se distribuye un tercer archivo desplegable ("`afirma-server-triphase-signer.war`") que incorpora los siguientes servicios:

- **`SignatureService`**
  - Servicio para la ejecución de operaciones de firma trifásica.
  - Requerido cuando deseamos utilizar este tipo de operación y cuando se realizan firmas desde las aplicaciones cliente móvil.
- **`BatchPresigner`**
  - Servicio para la carga de documentos y prefirma en los procesos de firma de lotes.
- **`BatchPostsigner`**
  - Servicio para la postfirma en los procesos de firma de lotes y guardado de las firmas.

Así, es necesario desplegar el archivo "`afirma-server-triphase-signer.war`" para realizar operaciones de firma trifásica y firma de lotes de documentos. También es necesario cuando se desea que nuestro despliegue sea compatible con todos los formatos de firma en dispositivos móviles. Puede saber más acerca de las operaciones de firma trifásica y firma de lotes en los apartados 6.2.2 Firma trifásica y 6.5 Firma por lotes predefinidos, respectivamente.

5.3.2.1.1 Configuración del servicio de firma trifásica

El servicio de firma trifásica debe funcionar correctamente con su configuración por defecto. Modifique únicamente sus opciones de configuración para establecer un gestor de documentos propio (*Document Manager*), restringir el origen de las peticiones o falla la generación de firmas XAdES trifásicas. El fichero del que se carga esta configuración del servicio de firma trifásica es "`tps_config.properties`".

Las propiedades que pueden establecerse en este servicio son:

- **`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: `*`
- **`document.manager`**
  - Clase que se encargará de gestionar los documentos que se deben firmar y las firmas generadas.
  - El gestor de documentos (*Document Manager)* por defecto imita un proceso de firma monofásica. Su valor es:
    - `es.gob.afirma.triphase.server.document.SelfishDocumentManager`

El fichero de configuración por defecto será:

```
# Orígenes permitidos
Access-Control-Allow-Origin=*

# Clase DocumentManager
document.manager=es.gob.afirma.triphase.server.document.SelfishDocumentManager
```

Adicionalmente, en este fichero se podrán configurar todas aquellas propiedades que se deseen establecer para el gestor de documentos configurado. Así, por ejemplo, en el gestor de documentos a través de sistema de ficheros ("`FileSystemDocumentManager`") que se proporciona junto al servicio como muestra, se puede configurar un directorio de entrada de los datos y uno de salida para las firmas, así como si se desean sobreescribir los datos en el directorio de salida.

Si se usase este gestor de documentos, por ejemplo, el fichero de configuración podría ser como el siguiente:

```
# Orígenes permitidos
Access-Control-Allow-Origin=*
```

```
# Clase DocumentManager
document.manager=es.gob.afirma.triphase.server.document.FileSystemDocumentManag
er

# Configuracion de la clase FileSystemDocumentManager
indir=/opt/user/clienteafirma/documents
outdir=/opt/user/clienteafirma/signatures
overwrite=false
```

Consulte el apartado II.1.1 Configuración del gestor de documentos del servicio para saber más acerca de los gestores de documentos y sobre el gestor de documentos a través de sistema de ficheros.

5.3.2.1.2 Configuración del servicio de firma de lotes

Para la firma por lotes, además de la configuración del fichero del servicio de firma trifásica, se requiere la configuración de un fichero llamado "`signbatch.properties`".

El fichero debe contar con las siguientes propiedades de configuración:

- **`tmpdir`**
  - Directorio a usar para los ficheros temporales. Si no se indica, se usará el del sistema, aunque es recomendable el uso de un directorio específico para este fin.
- **`concurrentmode`**
  - Es un parámetro opcional que 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. Este es el valor por defecto.
- **`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.
- **`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 (datos proporcionados directamente en Base64)
      - El valor a usar será siempre "`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://misitio.net/*`", "`http://gobierno.es/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.misitio.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.

Un ejemplo de fichero de configuración podría ser:

```
# Directorio temporal
tmpdir=C:/salida/temp

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

# Numero de firmas concurrentes
maxcurrentsigns=10

# Fuentes de datos permitidas, separadas por punto y coma (';')
allowedsources=base64;file://*;http://*;https://*;ftp://*
```

5.3.3 Configuración de los servicios

Independientemente de su función, todos los servicios que acompañan al Cliente @firma siguen la misma lógica para localizar su fichero de configuración y el uso de variables de entorno. En este apartado se explica cómo configurar estos aspectos comunes a todos ellos.

Los servicios del Cliente utilizan uno o varios ficheros de propiedades para su configuración. Estos ficheros tienen nombres prefijados, pero el integrador puede definir su ubicación mediante la variable de entorno "`clienteafirma.config.path`". El valor asignado a esta variable debe ser la ruta del directorio en el que se encontrarán esos ficheros. La variable puede establecerse, por ejemplo, por medio de la variable $JAVA_OPTS al levantar el servidor de aplicaciones. Por ejemplo:

```
JAVA_OPTS="%JAVA_OPTS% -Dclienteafirma.config.path=/opt/usuarios/cliente/conf"
```

También se puede hacer uso de otras variables declaradas por uno mismo o por el propio servidor de aplicaciones. Por ejemplo:

```
JAVA_OPTS="%JAVA_OPTS% -Dclienteafirma.config.path=%CATALINA_HOME%/conf/afirma"
```

En caso de no declararse la variable de entorno, se buscarán los ficheros en el *classpath* de la aplicación, por lo que podría introducirse el fichero de configuración dentro de los WAR de los servicios.

Cada uno de los ficheros de configuración de los servicios del Cliente @firma está compuesto por una serie de propiedades con valores asignados. El valor establecido en esas propiedades será el utilizado por los servicios, pero también es posible heredar parte de la configuración mediante variables de entorno. Para utilizar el valor de una variable de entorno como parte o todo el valor de una de las propiedades de configuración, estableceremos en el fichero el nombre de la variable en cuestión delimitado por las partículas "`${`" y "`}`". Por ejemplo:

Se podría establecer una propiedad en el arranque del servidor de aplicaciones:

```
JAVA_OPTS="%JAVA_OPTS% -Dclienteafirma.config.path=%CATALINA_HOME%/conf/afirma –
DtempDir=%CATALINA_HOME%/temp/afirma"
```

Mientras, en nuestro fichero de configuración podríamos establecer el valor de una propiedad valiéndonos de esa propiedad del sistema que hemos establecido:

```
# Se usará como directorio temporal, el subdirectorio "temp", localizado
# en el directorio configurado por medio de la propiedad de sistema "tempDir".
tmpdir=${tempDir}/temp
```

Los ficheros de configuración se leen una única vez durante la carga de los servicios por lo que, tras realizar un cambio en ellos, será necesario reiniciar el servidor de aplicaciones para que los servicios los apliquen.

5.4 Configuración del *Content Security Policy*

AutoFirma y los clientes móviles atienden las peticiones realizadas desde el JavaScript de despliegue por medio del protocolo "afirma". Si su servidor web utiliza cabeceras CSP (Content Security Policy) para limitar las fuentes de las que cargar los recursos de sus páginas web, es probable que algunos navegadores Web (se ha identificado el caso concreto de Mozilla Firefox) rechacen la llamada a estas URL externas.

En caso de que el servidor no utilice cabeceras CSP, no será necesario hacer nada para el funcionamiento del Cliente @firma. Si, en cambio, sí se utilizan, será necesario agregar a esta cabecera el esquema “afirma://” para garantizar que el navegador permite el acceso a este tipo de URL. Para hacer esto, se agregará la partícula “afirma://*” en la fuente por defecto de la política de seguridad.

Por ejemplo, si la cabecera con la política de seguridad fuese:

```
Content-Security-Policy: default-src 'self' *.site.com; img-src *
```

Se modificaría para convertirlo en:

```
Content-Security-Policy: default-src 'self' *.site.com afirma://*; img-src *
```

6 Uso del Cliente @firma

El API del Cliente @firma se expone automáticamente al entorno JavaScript al importar en la página web la biblioteca "`autoscript.js`". En esta biblioteca está definido el objeto "`AutoScript`" y a partir de él se podrá invocar a todas las operaciones del Cliente. Por ejemplo, para iniciarlizar el Cliente:

```
AutoScript.cargarAppAfirma();
```

O, para firmar:

```
AutoScript.sign(…);
```

La operación de inicialización del cliente no implica una llamada al Cliente @firma, únicamente inicializa el objeto JavaScript, por lo que puede realizarse durante la carga de la página. El resto de llamadas, sin embargo, implican la invocación a una aplicación externa como es el Cliente @firma.

La comunicación con el Cliente @firma se realiza de forma asíncrona. Para poder gestionar esto, todas las funciones que implican llamar al Cliente permite que se les proporcione una función *callback* a través de la cual obtener el resultado.

Para el correcto uso del Cliente @firma deben seguirse siempre las siguientes normas:

- No lanzar operaciones de forma automática. Todas las operaciones deberían desencadenarse a raíz de una acción del usuario. Por ejemplo, no se debe llamar al método de firma durante la carga de la página web. En su lugar, por ejemplo, se debería mostrar un botón "Firmar" y, cuando el usuario pulse dicho botón, llamar a la operación de firma.
- No se deben lanzar simultáneamente varias operaciones del Cliente. Hasta que no se obtenga el resultado de una operación a través de su función *callback*, no se debería llamar a la siguiente. Así, por ejemplo, no se permitiría hacer lo siguiente:

  ```
  AutoScript.sign(…);    // ERROR: La segunda función se llamará antes de
  AutoScript.sign(…);    // terminar la anterior
  ```

  En su lugar, se debería usar:

  ```
  // Funcion callback. Se ejecuta al procesar el resultado de la Firma 1
  function callback(…) {
      AutoScript.sign(…);    // Firma 2. resultado de la Firma 1
  }
  …
  AutoScript.sign(…, callback, …);    // Firma 1
  ```

6.1 Carga de la aplicación

Una vez importado el JavaScript en la página Web, deberemos inicializar el objeto de comunicación con el cliente mediante el método:

```
cargarAppAfirma()
```

Es necesario haber llamado al método de carga del Cliente antes de realizar cualquier otra de las operaciones soportadas por la aplicación.

A continuación, se muestran diferentes ejemplos de carga del Cliente:

- Carga directa desde el código HTML:

```
…
<body>
  <script type="text/javascript">
       AutoScript.cargarAppAfirma();
  </script>
…
```

- Carga desde una función invocada en el momento de firmar:

```
function firmar() {
      AutoScript.cargarAppAfirma();
      AutoScript.sign(
                      dataB64,
                      "SHA512withRSA",
                      "PAdES",
                     null,
                     firmaCorrectaCallback,
                     firmaErrorCallback);
}
```

La función de carga del Cliente sólo se debería invocar una única vez por página web. Así, no se debe utilizar este último ejemplo si se va a llamar varias veces al método de firma.

6.1.1 Configuración de los servicios auxiliares de comunicación

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). Cuando no sea posible la comunicación por socket, el Cliente y la página web se comunicarán a través de los servicios auxiliares de almacenamiento y recuperación que habrá que desplegar junto a la página de firma.

Para saber más sobre los distintos modos de comunicación entre el Javascript de despliegue y el Cliente de firma, consulte el apartado ANEXO I Comunicación JavaScript de despliegue – Cliente @firma.

Por regla general, es buena práctica desplegar los servicios auxiliares de comunicación y configurarlos en nuestra aplicación, ya que posibilitan que el trámite de firma sea compatible con diversos entornos de usuario, como con dispositivos móviles y versiones antiguas de Internet Explorer y Microsoft Edge.

Para la configuración de los servicios de comunicación se usará el método:

```
setServlets (storageServiceUrl, retrieveServiceUrl)
```

Un ejemplo de forzado de la comunicación a través de los servicios auxiliares es:

```
…
<body>
  <script type="text/javascript">
       AutoScript.cargarAppAfirma();
       AutoScript.setServlets(
              "https://gobierno.es/afirma-signature-storage/StorageService",
              "https://gobierno.es/afirma-signature-retriever/RetrieveService");
  </script>
…
```

Recuerde que, para evitar que se produzcan errores de *cross-site scripting (XSS),* los servicios auxiliares de comunicación deben estar disponibles desde el mismo dominio que la página web desde la que se realiza la frma.

Consulte el apartado Servicios auxiliares de comunicación para saber más acerca de los servicios auxiliares de comunicación y los entornos de usuario para los que son necesarios.

6.1.2 Forzado de la comunicación a través de los servicios auxiliares

Una aplicación puede forzar a que siempre se utilice la comunicación a través de los servicios auxiliares, ya sea para mantener un comportamiento homogéneo en todos los entornos, para evitar problemas específicos con algún entorno o para optimizar algún proceso de firma concreto. Esto, sin embargo, debería evitarse en la medida de lo posible.

Se puede forzar la comunicación a través de los servicios auxiliares invocando al método:

```
setForceWSMode(force)
```

Al invocar a este método con el parámetro `true` **antes de invocar al método de carga**, el mecanismo de comunicación quedará prefijado al de servicios auxiliares. Sin embargo, no se recomienda usar este método salvo en casos muy específicos, como cuando se utilizan operaciones de firma trifásica con un gestor de documentos a medida.

Advertencia: Diversas características y medidas de seguridad de los navegadores Firefox, Chrome y Edge afectan al uso de AutoFirma cuando se comunican a través de los servicios auxiliares. Si se forzarse al uso de estos servicios, debe evitarse realizar dos o más llamadas a las funciones que

invoquen al Cliente a raíz de una única interacción del usuario. Por ejemplo, no se podría invocar una llamada a una operación del Cliente desde la función *callback* de una operación anterior.
Google Chrome y Microsoft Edge imponen restricciones adicionales que también afectan a la compatibilidad del despliegue con dispositivos móviles. Para saber más acerca de estas restricciones y la comunicación con servidor intermedio, consulte el apartado 9.1.1 Compatibilidad con Google Chrome.

Un ejemplo de forzado de la comunicación a través de los servicios auxiliares es:

```
…
<body>
  <script type="text/javascript">
       AutoScript.setForceWSMode(true);
       AutoScript.cargarAppAfirma();
  </script>
…
```

6.1.3 Restricción según desfase horario con el servidor

Al realizar una firma en el equipo del usuario, se registra el momento de la firma 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. Por norma general, esto sólo ocurre en casos extremos y no suele ser necesario aplicar la medida que se describe en este apartado.

El cálculo del desfase horario se calcula comparando la hora del equipo del usuario con la información horaria devuelta por un servidor al cargar una página web. El Cliente @firma no puede modificar la hora del sistema, pero en caso de detectar un desfase sí que puede advertir al usuario o incluso bloquear la operación de firma.

Para hacer la comprobación de desfase horario puede utilizar el método JavaScript:

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

Este método debe invocarse antes del método de carga del Cliente y puede recibir como parámetros:

- `checkType`
  - Tipo de verificación que se desea realizar. Admite los valores:
    - `AutoScript.CHECKTIME_NO`: No realiza ningún tipo de comprobación. Este es el valor por defecto.
    - `AutoScript.CHECKTIME_RECOMMENDED`: Realiza la comprobación horaria y, en caso de encontrar un desfase, pedirá al usuario que lo corrija antes de continuar.

  - `AutoScript.CHECKTIME_OBLIGATORY`: Realiza la comprobación horaria y, en caso de encontrar un desfase, pedirá al usuario que lo corrija y evitará que el Cliente se ejecute en cualquier llamada posterior.
- `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.
  - El valor por defecto son 5 minutos (300.000 milisegundos).
- `checkURL`
  - URL contra la que se realizara la petición para obtener la hora del servidor.
  - Por defecto, se usará la URL de la página cargada en el navegador. Si cree que es posible que esto ocasione un mal funcionamiento por parte de su aplicación, indique otra URL dentro de su dominio.

Un ejemplo de uso sería:

```
…
<body>
  <script type="text/javascript">
       AutoScript.checkTime(AutoScript.CHECKTIME_RECOMMENDED, 300000);
       AutoScript.cargarAppAfirma();
  </script>
…
```

Debe notarse que, según la configuración del servidor web, es posible que no se pueda determinar la hora a través de la información que este proporciona. De ser así, no se mostrará ninguna advertencia al usuario para no perjudicar la realización del trámite. También es posible que un 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 si el usuario refrescase 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 la comprobación, podría realizar operaciones de firma sin que se le mostrasen advertencias.

En caso de que desee bloquear de forma completa la firma de datos cuando se detecte una hora incorrecta en el sistema, 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.

6.1.4 Selección del almacén de claves

El Cliente @firma accede al almacén de claves del navegador web o el sistema operativo del usuario para ofrecer sus certificados en las operaciones de firma. Los almacenes utilizados por defecto por cada una de las aplicaciones son:

- **AutoFirma**:
  - Si se utiliza Firefox, se accederá al almacén de claves interno del navegador y se buscarán las tarjetas inteligentes configuradas como dispositivos de seguridad del mismo.
  - En cualquier otro caso, se usará el del sistema operativo.
    - Windows: CAPI (*Cryptography Application Programming Interface*)
    - Apple macOS: Llavero de macOS
    - Linux: Almacén compartido NSS
  - Cuando se detecte un DNIe o una tarjeta CERES conectada al equipo, AutoFirma accederá a ella y dará la posibilidad de firmar con sus certificados, independendientemente de que también se muestren los certificados del navegador o del sistema operativo.
- **Cliente móvil Android**:
  - Utiliza el almacén de claves de Android.
- **Cliente móvil iOS**:
  - Utiliza el almacén de la propia aplicación.

Aunque **se recomienda no seleccionar el uso de un almacén de claves distinto al por defecto**, AutoFirma permite que se configure otro de los almacenes soportados. Los Cliente móviles, en cambio, sólo admiten el almacén por defecto.

Para establecer el almacén de claves al que debe acceder AutoFirma, se deberá utilizar el siguiente método antes de invocar a cualquier operación de firma o selección de certificado:

```
setKeyStore (keystore);
```

En esta función:

- `keystore`
  - Tipo de almacén al que se debe URL de acceso a los datos a descargar.
  - En el objeto AutoScript se han definido las siguientes claves para configurar almacenes de claves:
    - `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…).

Si se selecciona un almacén no disponible en el entorno del usuario, AutoFirma dará error al intentar acceder al almacén de claves. Los clientes móviles únicamente ignorarán esta opción de la configuración.

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 AutoFirma 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 Cliente @firma 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 si se utiliza AutoFirma 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.

A continuación, se muestran ejemplos de selección del almacén de claves:

- Configuración del almacén de Windows:

```
AutoScript.cargarAppAfirma();
AutoScript.setKeyStore(AutoScript.KEYSTORE_WINDOWS);
…
AutoFirma.sign (dataB64, "SHA512withRSA", "CAdES", null, successCallback,
                errorCallback);
```

- Configuración de un almacén PKCS#12 en una ruta conocida:

```
AutoScript.cargarAppAfirma();
AutoScript.setKeyStore(AutoScript.KEYSTORE_PKCS12 +
":/usr/home/usuario/almacen.p12");
…
AutoFirma.sign (dataB64, "SHA512withRSA", "CAdES", null, successCallback,
                errorCallback);
```

- Configuración de un controlador PKCS#11 para el acceso de dispositivo critprográfico:

```
AutoScript.cargarAppAfirma();
AutoScript.setKeyStore(AutoScript.KEYSTORE_PKCS11 +
":C:\\Windows\\System32\\PkcsV2GK.dll");
…
AutoFirma.sign (dataB64, "SHA512withRSA", "CAdES", null, successCallback,
                errorCallback);
```

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

están 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.

6.1.4.1 Uso de tarjetas inteligentes

El Cliente @firma 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. AutoFirma 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.

6.1.4.2 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 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()`", descrita en el apartado 6.2.1 Firma de múltiples documentos (firma masiva), 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 de AutoFirma (se cambia de página, 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 Autofirma usando el almacén de 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.

6.2 Firma electrónica

La operación de firma electrónica nos permite general la firma electrónica de unos datos, que pueden haber sido proporcionados por la aplicación o seleccionados por el usuario.

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

```
sign(dataB64, algorithm, format, params, successCb, errorCb);
```

En esta función:

- `dataB64`
  - Datos que se desean firmar codificados en Base64.
    - Si los datos que necesita firmar son un texto (texto plano, XML, JSON, etc), puede convertirlos a Base64 por medio de las funciones de utilidad proporcionadas en el JavaScript de despliegue, descritas en el apartado 6.9.2 Conversión de un texto a cadena Base64.
  - Si no se proporciona este parámetro (se usa `null`) o si se pasa una cadena vacía, el Cliente @firma mostrará al usuario un diálogo de selección de fichero para que seleccione el documento que desea firmar.
    - Salvo en casos concretos, no se recomienda dejar en manos del usuario la selección del fichero a firmar. Incluso si es necesario hacerlo, se recomienda hacer que el usuario cargue los datos previamente en servidor mediante un componente HTML y seguidamente se realice una operación de firma trifásica.
    - Para la compatibilidad con dispositivos móviles, nunca se deben proporcionar nulos o cadena vacía.
- `algorithm`
  - Algoritmo de firma. Consulte el apartado dedicado a los algorirmos de firma soportados para el formato de firma que desee utilizar.
- `format`
  - Formato de firma. Consulte el apartado 8 Formatos de firma para consultar aquellos disponibles.
- `params`
  - Parámetros adicionales para la configuración de la operación de firma y características particulares del formato de firma seleccionado.
    - Si se introduce un nulo, se usará la configuración por defecto para el formato de firma establecido.
    - Consulte el apartado 7.1 Paso de parámetros adicionales 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.
- `successCb`
  - Función *callback* JavaScript que se ejecutará una vez se obtenga el resultado de la operación de firma. Esta función recibirá hasta tres parámetros:
    1. En el primer parámero se recibe la firma resultante codificada en Base64.
    2. En el segundo parámetro se recibe el certificado usado para firmar codificado en Base64.
    3. En el tercer parámetro, opcionalmente, se recibirá un objeto JSON con el nombre del fichero firmado (en caso de que lo haya seleccionado el usuario).
- `errorCb`
  - Función *callback* JavaScript que se ejecutará cuando ocurra un error durante la operación de firma. Esta función recibirá hasta dos parámetros:
    1. En el primer parámetro se recibe un texto con el tipo del error.

2. En el segundo parámetro se recibe un texto con el mensaje de error.

A continuación, se muestran distintos ejemplos de firma electrónica:

- Firma electrónica de datos:

```
…
  var dataB64 = "SG9sYSBNdW5kbyE="; // Datos a firmar

  AutoScript.sign (dataB64, "SHA512withRSA", "CAdES",
                   "mode=implicit\nexpPolicy=FirmaAGE", successCallback,
                   errorCallback);
…
```

- Firma electrónica de un texto:

```
…
  var dataB64 = AutoScript.getBase64FromText("Hola Mundo!!");

  AutoScript.sign (dataB64, "SHA512withRSA", "XAdES", null, successCallback,
                   errorCallback);
…
```

- Firma electrónica cargando datos desde un fichero:

```
…
  // Función que se ejecutará cuando la firma termine correctamente.
  // Almacenara la firma, el certificado y el nombre del fichero firmado en
  // campos de un formulario y lo enviará a servidor
  function sendSignatureCallback (signatureB64, certificateB64, extraData) {

       // Obtenemos el nombre del fichero cargado para
       var filename = !!extraData ? extraData.filename : null;

       document.getElementByName("signatureField").value = signatureB64;
       document.getElementByName("certificateField").value = certificadteB64;
       if (filename) {
             document.getElementByName("filenameField").value = filename;
       }
       document.getElementByName("formulario").submit();
  }

  // Funcion que se ejecutara cuando el proceso de firma falle
  function showErrorCallback (type, message) {
       showError(message);// Funcion de la aplicación para mostrar errores
  }
…

  // Llamamos a la operacion de firma
  AutoScript.sign (null, "SHA512withRSA", "CAdES", null, sendSignatureCallback,
                   showErrorCallback);
```

```
…
```

6.2.1 Firma de múltiples documentos (firma masiva)

Cuando se deseen firmar multiples datos a través del Cliente @firma debe tenerse en cuenta un aspecto importante: el envío de datos a firmar debe realizarse siempre secuencialmente, nunca en paralelo. Es decir, no enviaremos a firmar un dato hasta no haber recibido el resultado de una operación anterior.

El modo común y recomendado de realizar la firma secuencial de múltiples documentos es solicitar la firma de un documento en la función *callback* en la que se recibe el resultado de la operación anterior.

Un ejemplo de uso es:

```
// Posicion del elemento que se esta procesando
var idx = 0;
// Array con los valores a firmar
var dataArray = [   "UHJpbWVyIGRhdG8gZGUgcHJ1ZWJh",
                    "U2VndW5kbyBkYXRvIGRlIHBydWViYQ==",
                    "VGVyY2VyIGRhdG8gZGUgcHJ1ZWJh",
                    "Q3VhcnRvIGRhdG8gZGUgcHJ1ZWJh"
 ];

// Iniciamos la firma del primer dato a firmar
AutoScript.setStickySignatory(true);
AutoScript.sign(    dataArray[idx],
                    "SHA512withRSA",
                    "CAdES",
                    paramsParam,
                    successCallback,
                    showErrorCallback);
…

// Función callback a ejecutar cuando termine correctamente una firma
function successCallback (signatureB64) {
       // Procesamos la firma
       …

       // Mandamos a firmar el siguiente dato (si quedase alguno)
       ++idx;
       if (idx < dataArray.length) {
              AutoScript.sign(    dataArray[idx],
                                  "SHA512withRSA",
                                  "CAdES",
                                  paramsParam,
                                  successCallback,
                                  showErrorCallback);
```

```
        }
        else {
                // Ya se han generado todas las firmas
                …
        }
}

// Función callback a ejecutar cuando falle la firma
function showErrorCallback (errorType, errorMsg) {
        // Mostramos el error
        alert("Error durante la firma de un documento. Se interrumpirá el
proceso de firma.");
}
```

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:

```
setStickySignatory (sticky);
```

En esta función:

- *sticky*
  - Indica si se debe fijar el siguiente certificado que se utilice.
    - Si se indica el valor "`true`", el próximo certificado que seleccione mediante un filtro o que seleccione directamente el usuario (en una operación de firma o selección de certificado) 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.

Esta función no devuelve nada y sólo es compatible con AutoFirma. En una operación de firma con los clientes móviles, el usuario deberá seleccionar el certificado por cada operación de firma individual.

Al realizar múltiples firmas con tarjetas inteligentes, es posible que se pida el PIN de la tarjeta por cada operación individual, 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 si se ha llamado a la función "`setStickySignatory()`" con el valor "`true`".

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. Puede saber más sobre esta opción en el apartado 8 Formatos de firma.

Otra alternativa para firmar múltiples documentos es la operación de 6.5 Firma por lotes predefinidos.

6.2.2 Firma trifásica

Las firmas realizadas por el Cliente @firma se realizan, por defecto, en el equipo del usuario. Sin embargo, el Cliente también es compatible con las operaciones de firma trifásica, en las que todas las operaciones de tratamiento de los datos y construcción de la estructura de firma se realiza en servidor y sólo la operación de cifrado con la clave privada del certificado del usuario se realiza en el equipo local (la clave del certificado nunca sale del equipo del usuario).

Las operaciones de firma trifásicas ayudan a reducir el envío de información entre el Cliente y el servidor cuando los datos a firmar se encuentran en servidor y cuando la firma generada por el usuario también va a enviarse a servidor. En cambio, este tipo de firmas requieren un mayor procesamiento por parte de dicho servidor, ya que la operación de firma, que por defecto se realiza por entero en el equipo del usuario, en las operaciones de firma trifásica deben realizarse parcialmente en el servidor.

La realización de firmas trifásicas requiere el despliegue del servicio de firma trifásica descrito en el apartado 5.3.2 Servicios de firma trifásica y firma de lotes. Así mismo, en la llamada a la operación de firma que deseemos realizar de forma trifásica deberán establecerse los siguientes datos:

- Se usará el nombre del formato de firma trifásica en lugar del nombre de formato tradicional:
  - Para firmas CAdES: `CAdESTri`
  - Para firmas XAdES: `XAdESTri`
  - Para firmas PAdES: `PAdESTri`
  - Para firmas de factura electrónica: `FacturaETri`
- Se configurará el parámetro adicional "`serverUrl`" con la URL del servicio de firma trifásica.

Un ejemplo de operación de firma trifásica sería:

```
…
// Configuramos una firma CAdES trifásica implicita
var params = "mode=implicit\nserverUrl=https://servidor.com/afirma/afirma-
server-triphase-signer/SignatureService";

AutoScript.sign (dataB64, "SHA512withRSA", "CAdESTri", params,
                 showSignResultCallback, showErrorCallback);
…
```

El proceso trifásico de firma se encuentra disponible para las operaciones de firma, cofirma y contrafirma.

Consulte más información sobre la operativa interna del proceso de firma trifásica en el apartado ANEXO II Firma trifásica.

6.3 Cofirma electrónica

La cofirma es la operación mediante la cual se agrega una nueva firma a una firma previa. La firma generada está al mismo nivel que la firma original. Esta sería la operación mediante la cual dos o más firmantes muestran su acuerdo con un documento.

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. El segundo firmante deberá cofirmar la firma generada por el primer firmante, el tercero cofirmará la firma generada por el segundo y así sucesivamente. Esta operación se puede realizar por entero en el lado cliente o a través del proceso de firma trifásica descrito en el ANEXO II Firma trifásica.

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

```
cosign(signB64, algorithm, format, params, successCb, errorCb);
```

En esta función:

- `signB64`
  - Firma electrónica que se desea cofirmar codificada en Base64.
    - Una firma en Base64 puede ser el resultado obtenido por cualquier operación de firma, cofirma o contrafirma previa.
  - Si no se proporciona este parámetro (se usa `null`) o si se pasa una cadena vacía, el Cliente @firma mostrará al usuario un diálogo de selección de fichero para que seleccione el fichero de firma que desea cofirmar.
    - Salvo en casos concretos, no se recomienda dejar en manos del usuario la selección del fichero de firma. Incluso si es necesario hacerlo, se recomienda hacer que el usuario cargue la firma previamente en servidor mediante un componente HTML y seguidamente se realice una operación de cofirma trifásica.
    - Para la compatibilidad con dispositivos móviles, nunca se deben proporcionar nulos o cadena vacía.
- `algorithm`
  - Algoritmo de firma. Consulte el apartado dedicado a los algorirmos de firma soportados para el formato de firma que desee utilizar.
- `format`
  - Formato de firma. Consulte el apartado 8 Formatos de firma para consultar aquellos disponibles.
    - Si no conoce el formato de firma utilizado en la firma original, indique el valor "`AUTO`" para especificar que se utilice el mismo formato que la firma original.
    - Por norma general, no se puede cofirmar una firma en un formato distinto al que se usó para generar dicha firma.
- `params`

- Parámetros adicionales para la configuración de la operación de cofirma y características particulares del formato de firma seleccionado.
  - Si se introduce un nulo, se usará la configuración por defecto para el formato de firma establecido.
  - Consulte el apartado 7.1 Paso de parámetros adicionales 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.

- `successCb`
  - Función *callback* JavaScript que se ejecutará una vez se obtenga el resultado de la operación de cofirma. Esta función recibirá hasta tres parámetros:
    1. En el primer parámero se recibe la firma resultante codificada en Base64.
    2. En el segundo parámetro se recibe el certificado usado para cofirmar codificado en Base64.
    3. En el tercer parámetro, opcionalmente, se recibirá un objeto JSON con el nombre del fichero de firma cofirmado en caso de que lo haya seleccionado el usuario.
- `errorCb`
  - Función *callback* JavaScript que se ejecutará cuando ocurra un error durante la operación de firma. Esta función recibirá hasta dos parámetros:
    1. En el primer parámetro se recibe un texto con el tipo del error.
    2. En el segundo parámetro se recibe un texto con el mensaje de error.

A continuación, se muestran distintos ejemplos de operaciones de cofirma:

- Cofirma electrónica de una firma ya cargada:

```
…
  var signatureB64 = … // Firma electrónica a cofirmar

  AutoScript.cosign(signatureB64, “SHA512withRSA”, “AUTO”, null,
successCallback, errorCallback);
…
```

- Cofirma electrónica cargando una firma desde un fichero:

```
…
  AutoScript.cosign(null, “SHA512withRSA”, “XAdES”, null, successCallback,
errorCallback);
…
```

- Cofirma electrónica del resultado de una firma:

```
…
```

```
  // Funcion que realiza la cofirma a partir de los datos de firma
  function cosignCallback (signatureB64) {
       AutoScript.cosign(
             signatureB64, "SHA512withRSA", "XAdES", null, sendDataCallback,
showErrorCallback
       );
  }
…
  // Función que almacena los datos generados por la cofirma en el campo
  // "resultId" de un formulario y lo envia
  function sendDataCallback (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) {
       AutoScript.sign(dataB64, "SHA512withRSA", "XAdES", "format=XAdES
Detached", cosignCallback, showErrorCallback);
  }
…
```

6.4 Contrafirma electrónica

La contrafirma es la operación mediante la cual se refrenda una firma electrónica previa. En términos generales, es el firmar una firma electrónica. Este es el tipo de operación que realizaría un organismo, por ejemplo, como prueba de registro de una firma electrónica en su sistema.

La contrafirma no es una operación que soporten todos los formatos de firma. Consulte el apartado dedicado a los algorirmos de firma soportados para el formato de firma que desee utilizar.

La contrafirma se puede realizar por entero en el lado cliente o a través del proceso de firma trifásica descrito en el ANEXO II Firma trifásica.

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

```
countersign(signB64, algorithm, format, params, successCb, errorCb);
```

En esta función:

- `signB64`
  - Firma electrónica que se desea contrafirmar codificada en Base64.

        - Una firma en Base64 puede ser el resultado obtenido por cualquier operación de firma, cofirma o contrafirma previa.
    - Si no se proporciona este parámetro (se usa `null`) o si se pasa una cadena vacía, el Cliente @firma mostrará al usuario un diálogo de selección de fichero para que seleccione el fichero de firma que desea contrafirmar.
        - Salvo en casos concretos, no se recomienda dejar en manos del usuario la selección del fichero de firma. Incluso si es necesario hacerlo, se recomienda hacer que el usuario cargue la firma previamente en servidor mediante un componente HTML y seguidamente se realice una operación de cofirma trifásica.
        - Para la compatibilidad con dispositivos móviles, nunca se deben proporcionar nulos o cadena vacía.
- `algorithm`
    - Algoritmo de firma. Consulte el apartado dedicado a los algorirmos de firma soportados para el formato de firma que desee utilizar.
- `format`
    - Formato de firma. Consulte el apartado 8 Formatos de firma para consultar aquellos disponibles.
        - Si no conoce el formato de firma utilizado en la firma original, indique el valor "`AUTO`" para especificar que se utilice el mismo formato que la firma original.
        - Por norma general, no se puede contrafirmar una firma en un formato distinto al que se usó para generar dicha firma.
- `params`
    - Parámetros adicionales para la configuración de la operación de contrafirma y características particulares del formato de firma seleccionado.
        - Si se introduce un nulo, se usará la configuración por defecto para el formato de firma establecido.
        - La operación de contrafirma admite un parámetro adicional concreto para configurar qué firmas dentro del fichero de firma se quieren contrafirmar. Consulte el apartado 6.4.1 Selección de nodos para más detalle.
        - Consulte el apartado 7.1 Paso de parámetros adicionales 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.
- `successCb`
    - Función *callback* JavaScript que se ejecutará una vez se obtenga el resultado de la operación de cofirma. Esta función recibirá hasta tres parámetros:
        1. En el primer parámero se recibe la firma resultante codificada en Base64.
        2. En el segundo parámetro se recibe el certificado usado para cofirmar codificado en Base64.

3. En el tercer parámetro, opcionalmente, se recibirá un objeto JSON con el nombre del fichero de firma cofirmado en caso de que lo haya seleccionado el usuario.

- `errorCb`
  - Función *callback* JavaScript que se ejecutará cuando ocurra un error durante la operación de firma. Esta función recibirá hasta dos parámetros:
    1. En el primer parámetro se recibe un texto con el tipo del error.
    2. En el segundo parámetro se recibe un texto con el mensaje de error.

A continuación, se muestran distintos ejemplos de contrafirma:

- Contrafirma electrónica de una firma

```
…
  var signatureB64 = … // Firma electrónica a cofirmar

  AutoScript.countersign(signatureB64, “SHA512withRSA”, “AUTO”, “target=true”,
successCallback, errorCallback);
…
```

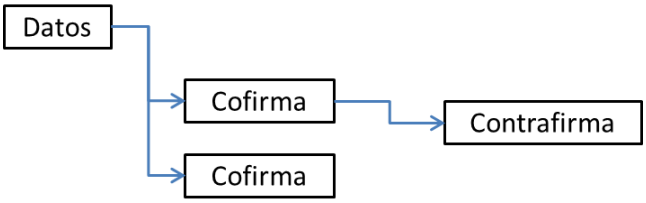
- Contrafirma electrónica del resultado de una firma

```
…
  // Funcion callback que desencadena la operación de contrafirma
  function counterSignCallback (signatureB64) {
       AutoScript.countersign(
                    signatureB64, “SHA512withRSA”, “XAdES”, null,
                    successCallback, errorCallback
       );
  }
…

  // Iniciamos la operacion de firma que después se contrafirmará
  AutoScript.sign(
       dataB64, “SHA512withRSA”, “XAdES”, null, counterSignCallback,
       errorCallback);
…
```

6.4.1 Selección de nodos

La operación de contrafirma se realiza sobre una 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 “árbol de firmas”. Así, por ejemplo, después de realizar varias cofirmas y contrafirmas sobre una firma se podría obtener una estructura como la siguiente:



En esta estructura se refleja que se han firmado los datos dos veces (firma y cofirma) y que una de esas firmas tiene una contrafirma. Debe tenerse en cuenta que las firmas y las cofirmas son equivalentes (ambas firman los datos) y por tanto ambas se pueden considerar cofirmas entre sí.

En relación a esto, el Cliente @firma permite configurar que las contrafirmas se realicen según una de las siguientes políticas de firma:

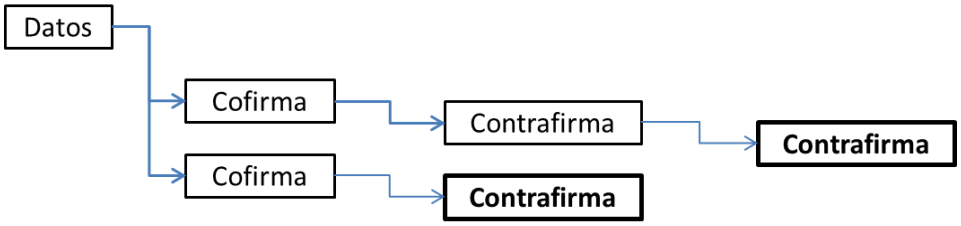
- **Firma de nodos hoja del árbol** (`LEAFS`): Se firmarán sólo las firmas del árbol que no tienen ninguna contrafirma.
- **Firma de todo el árbol de firma** (`TREE`): Se firman todas las firmas del árbol.

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, se le puede agregar la propiedad `target` con la política de selección de nodos. Los valores que admite esta propiedad son:

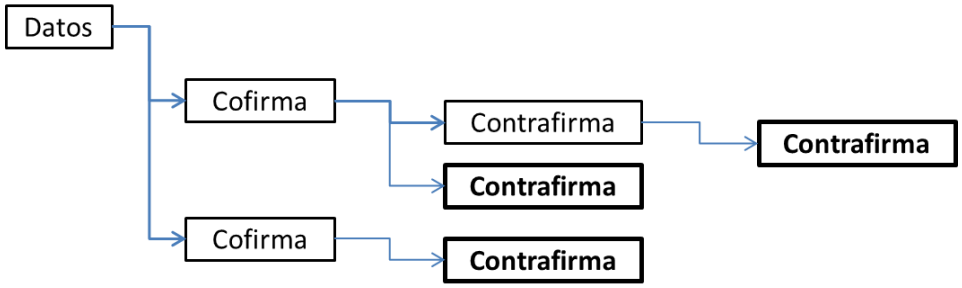
- `leafs`
  - Contrafirma todas las firmas que sean nodos hoja del árbol. Este es el valor por defecto.
- `tree`
  - Contrafirma todas las firmas del árbol.

Por ejemplo, si tomamos como base la estructura de firma anterior, así quedaría al contrafirmarla con cada uno de los parámetros adminitidos:

- `target=leafs`



- `target=tree`



6.5 Firma por lotes predefinidos

AutoFirma incorporan una funcionalidad de firma de lotes de documentos, lo que permite a las aplicaciones enviar a firmar un grupo de datos en una única llamada. Tanto el lote de documentos como la configuración de firma se selecciona mediante un XML que deberá crear la propia aplicación que solicite el lote. La operación de firma de lote permite que el ususario pueda seleccionar el certificado de firma una única vez.

Las firmas de un lote de firma se realizan de forma trifásica, lo cual se traduce en que buena parte del proceso de firma se realiza en servidor y es necesario utilizar los servicios del WAR "afirma-server-triphase-signer.war". Consulte el apartado 5.3.2 Servicios de firma trifásica y firma de lotes para saber más sobre el despliegue y configuración de este archivo.

La aplicación que solicita la firma de un lote de firmas recibe como resultado el listado de resultados de las firmas generadas, pero no las propias firmas ya que estas se componen y procesan en servidor. La aplicación es capaz de configurar qué mecanismo de guardado utilizar, que puede ser uno de los proporcionados de serie por el servicio de firma de lote o uno programado a medida para su sistema.

Advertencia: La operación de firma de lotes actualmente no es compatible con los clientes de firma de Android e iOS.

La función de firma de lotes es:

```
signBatch (batchB64, preSignerUrl, postSignerUrl, params, successCb,
                    errorCb);
```

En esta función:

- `batch64`
  - Es el XML de definición de lote codificado en Base64. Para saber cómo construir el fichero de lote consulte el apartado 6.5.1 Creación de los lotes.
- `preSignerUrl`
  - URL del servicio de prefirma de lotes. Este servicio se despliega junto con el servicio de firma trifásica.
- `postSignerUrl`

- 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 5.3.3 Paso de parámetros adicionales para saber cómo realizar el paso de parámetros y el apartado específico 7.2 Configuración del filtro de certificados para conocer los parámetros que se pueden establecer.
- `successCb`
  - Función callback JavaScript que se debe ejecutar en caso de que la firma del lote finalice correctamente.
  - Esta función se ejecutará incluso si alguna de las firmas del lote no se ha podido generar, en cuyo caso, únicamente se devolverá en el listado de resultados que esa firma produjo un error.
  - Esta función recibirá como parámetros:
    1. Un XML codificado en Base64 con el resultado de cada firma del lote. No se recibe la firma resultante, esta queda almacenada en servidor, sino un indicador de si la operación terminó bien o mal y el motivo.
    2. El certificado utilizado para firmar codificado en Base64.
- `errorCb`
  - Función callback JavaScript que se debe ejecutar en caso de que no se pueda precesar la firma del lote.
  - Esta función recibirá dos parámetros:
    1. El tipo de error producido.
    2. Un texto descriptivo del error.

6.5.1 Creación de los lotes

Las aplicaciones que deseen usar esta funcionalidad deben proporcionar al Cliente un XML con los datos y la configuración del lote de firma. El esquema que debe seguir este XML es:

```
<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>https://miorganismo.gob.es/documentos?id=123</datasource>
		<format>XAdES</format>
		<suboperation>sign</suboperation>
		<extraparams>
			Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpTaWduYXR1cmVJZD03NzI1M
zc0ZS03MjhkLTRhMzMtOWRiOS0zYTRlZmVhNGNlYWQNCg==
		</extraparams>
		<signsaver>
			<class>es.gob.afirma.signers.batch.SignSaverFile</class>
			<config>
	RmlsZU5hbWU9QzpcXFVzZXJzXFxjbGllbnRlXFxEb2N1bWVudHNcXHNpZ25zXFxmaXJtYTEueG1s
			</config>
		</signsaver>
	</singlesign>
	<singlesign Id="93d1531c-cd32-4c8e-8cc8-1f1cfe66f64a">
		<datasource>SG9sYSBNdW5kbw==</datasource>
		<format>CAdES</format>
		<suboperation>sign</suboperation>
		<extraparams>
			Iw0KI1N1biBBdWcgMjMgMTM6NDU6NDAgQ0VTVCAyMDE1DQpTaWduYXR1cmVJZD05M2QxN
TMxYy1jZDMyLTRjOGUtOGNjOC0xZjFjZmU2NmY2NGENCg==
		</extraparams>
		<signsaver>
			<class>es.gob.afirma.signers.batch.SignSaverFile</class>
			<config>
	RmlsZU5hbWU9QzpcXFVzZXJzXFxjbGllbnRlXFxEb2N1bWVudHNcXHNpZ25zXFxmaXJtYTIueG1s
			</config>
		</signsaver>
	</singlesign>
</signbatch>
```

El XML de definición de lote **no puede exceder en tamaño de 1MB**. Si necesitase componer trabajos mayores, estos deberán fraccionarse en varios lotes. Tenga en cuenta que no es necesario que los datos a firmar se incluyan en el propio XML de petición. Puede establecerse como datos a firmar una URL, accesible desde el servidor de firma, desde la que obtener los datos. La configuración de los datos del lote se ve en detalle en los siguientes apartados.

6.5.1.1 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:

- `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.

- `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
- `concurrenttimeout`
    - Número de segundos que deberá durar como máximo cada fase de 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.

6.5.1.2 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:

La cabecera "`<singlesign id="7725374e-728d-4a33-9db9-3a4efea4cead">`", indica que la petición de firma dentro del lote se identifica por la cadena "`7725374e-728d-4a33-9db9-3a4efea4cead`". Cuando obtengamos el resultado de la firma del lote, podremos identificar el resultado de una operación particular mediante este mismo identificador. La aplicación podrá establecer el identificador que desee para cada operación de firma siempre que este no corrompa el XML ni se encuentre repetido dentro del mismo XML.

En los siguientes subapartados, se detallan los aspectos configurables de cada una de las firmas del lote.

6.5.1.2.1 Origen de los datos a firmar

El origen de los datos debe indicarse dentro del elemento "`datasource`" del XML, por ejemplo: "`<datasource>https://miorganismo.gob.es/documentos?id=123</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 5.3.2.1.2 Configuración del servicio de firma de lotes) de las siguientes maneras:

1. 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).
    - La URL debe devolver el binario de los datos a firmar y ser accesible desde el servicio de firma de lotes.
    - Para los formatos y la configuración de firma de firma que lo soporten, la URL puede proporcionar una huella digital (*hash*).
    - En el ejemplo, puede observarse que en la primera firma se indica que el origen de los datos es la URL "`https://miorganismo.gob.es/documentos?id=123`".
2. Indicando directamente los datos a firmar codificados en Base64. En este caso el servidor descodificará el Base64 para obtener los datos a firmar.
    - Para los formatos y la configuración de firma de firma que lo soporten, el Base64 puede ser el de una huella digital (*hash*).
    - En el ejemplo, puede observarse que en la segunda firma se indica que el origen de los datos es el Base64 "*`SG9sYSBNdW5kbw==`*", que descodificado equivale a la cadena de texto "*`Hola Mundo`*".

6.5.1.2.2 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`
- `CAdES`
- `PAdES`
- `FacturaE`

6.5.1.2.3 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:

- `sign`
    - Firmar.
- `cosign`
    - Cofirmar.
- `countersign`
    - Contrafirmar.

6.5.1.2.4 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
```

Según el modo en el que obtenga el Base64, es posible que deba utilizar como representación visible del retorno de carro la partícula '`\n`'. Así, los parámetros anteriores quedarían como:

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

Este es el caso de la función `getBase64FromText()` del JavaScript de despliegue. Si quieramos calcular el Base64 que debemos incluir en el nodo de firma del lote para indicar los datos anteriores, ejecutaríamos:

```
var extraParams = AutoScript.getBase64FromText(
                    "mode=implicit\nsignatureProductionCity=Madrid");
```

La cadena Base64 resultante sería:

```
bW9kZT1pbXBsaWNpdA0Kc2lnbmF0dXJlUHJvZHVjdGlvbkNpdHk9TWFkcmlk
```

Consulte el apartado 6.9.2 Conversión de un texto a cadena Base64 para saber más del método de conversión de texto a Base64.

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

6.5.1.2.5 Configuración del guardado de la firma

Una vez se completan las firmas de un lote, estas se procesan en servidor, utilizando para ello una de las clases de procesamiento de firmas. El servicio de firma de lotes proporciona dos de estas clases, pero, si se desean realizar un proceso distinto, será necesario que un programador agregue la lógica de procesado al servicio.

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`
  - Nombre cualificado de la clase de guardado a usar para el guardado de esa firma.
- `config`

- Propiedades de configuración codificadas en Base64 que se le proporcionarán a la clase de guardado.

Así, en el ejemplo, la primera firma define el siguiente nodo:

```
<signsaver>
	<class>es.gob.afirma.signers.batch.SignSaverFile</class>
	<config>RmlsZU5hbWU9QzpcXFVzZXJzXFxjbGllbnRlXFxEb2N1bWVudHNcXHNpZ25zXFxmaXJtYTEueG1
s</config>
</signsaver>
```

Este indica que debe usarse la clase de guardado "`es.gob.afirma.signers.batch.SignSaverFile`" con la configuración "`RmlsZU5hbWU9QzpcXFVzZXJzXFxjbGllbnRlXFxEb2N1bWVudHNcXHNpZ25zXFxmaXJtYTEueG1s`", que si la descodificamos vemos que contiene:

```
FileName=C:\\Users\\cliente\\Documents\\signs\\firma1.xml
```

Como resultado, tras finalizar el lote de firmas, esta firma particular se guardará en la ruta del servidor definida en la configuración ("`C:\\Users\\cliente\\Documents\\signs\\firma1.xml`").

Se incluyen de serie en el servicio de firma de lotes dos clases de ejemplo de guardado:

- `es.gob.afirma.signers.batch.SignSaverFile`
  - Guardado a fichero.
  - Parámetros de configuración admitidos:
    - `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 no realiza comprobaciones de seguridad básicas.**
    - Ver apartado 6.5.1.2.5.1 *Notas sobre el ejemplo SignSaverFile*.
- `es.gob.afirma.signers.batch.SignSaverHttpPost`
  - Envío a una dirección HTTP POST.
  - Parámetros de configuración:
    - `PostUrl`
      - URL a la que enviar los datos vía HTTP POST. Puede ser tanto HTTP como HTTPS.
    - `PostParamName`
      - Indica el nombre del parámetro de la petición POST donde deben adjuntarse los datos.
  - 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 URL SAFE (los signos '+' sustituidos por '-' y los signos '/' sustituidos por '_') del resultado de la firma. Es necesario entonces que la aplicación interesada en

recibir estos datos habilite un servicio al que se llamará desde el servicio de firma de lotes del Cliente @firma.

Si se desea que las firmas generadas por una aplicación se procesen de forma distinta a la permitida por las clases de guardado incluidas en el servicio, será necesario implementar una clase de guardado propia e integrarla en el servicio de firma. Puede saber más sobre como programar su propia lógica de guardado de firma en el apartado 6.5.3 Creación de una lógica de guardado personalizada.

6.5.1.2.5.1 Notas sobre el ejemplo SignSaverFile

La clase "`es.gob.afirma.signers.batch.SignSaverFile`" es, como se ha comentado anteriormente, únicamente un ejemplo de implementación de la 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:

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

Si desea usarla para pruebas con escrituras reales en disco, debe modificar el fichero fuente 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.

6.5.2 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 Base64 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, cada cual con su correspondiente identificador.

Cada uno de estos nodos tiene los siguientes atributos:

- `id`
  - Identificador de la firma.
- `result`
  - Resultado del proceso.
  - Los valores posible se desciben en el apartado 6.5.2.1 Tipos de resultados de la ejecución de un lote.
- `description`
  - Descripción del resultado del proceso (opcional). Comúnmente utilizado para describir algún error producido.

6.5.2.1 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.

6.5.3 Creación de una lógica de guardado personalizada

Si las clases de guardado de firmas proporcionadas junto al servicio no le proporcionan a una aplicación la funcionalidad necesaria, se pueden programar nuevas clases de guardado de firma.

Las clases con la funcionalidad desarrollada deben situarse en el CLASSPATH de la aplicación Web en el lado servidor, normalmente dentro de uno o más fichero JAR. Consulte la documentación de su servidor de aplicaciones para realizar esta tarea.

Una forma sencilla de agregar las nuevas clases de guardado es editando el WAR del servicio de firma de lotes (es un fichero ZIP con la extensión cambiada) antes de desplegarlo, y añadiendo los archivos 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.

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 la gestión de firmas una vez realizadas. */
public interface SignSaver {

	/** 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(Properties config);

	/** 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(SingleSign sign, byte[] dataToSave) throws IOException;

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

	/** 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.

6.5.3.1 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 de las firmas fallase, se interrumpiría todo el proceso y se marcarían todos como error.
- Una vez están generadas todas las firmas, 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. Un ejemplo real, requeriría que además del servicio de envío de datos tuviese un servicio para el borrado de los mismos.

6.6 Selección de certificado

El Cliente @firma permite a las aplicaciones solicitar un certificado a los usuarios sin necesidad de realizar una operación de firma. Esta operación puede ser útil para construir filtros de certificados para operaciones posteriores, realizar validaciones previas sobre el certificado y/o mostrar información de este en su aplicación web. Sin embargo, **esta operación nunca debe usarse para autenticar a los usuarios**, ya que lo que se obtiene es únicamente la parte pública de los certificados y un mecanismo de autenticación que usase sólo eso podría ser fácilmente sorteado mediante un ataque simple de inyección de código.

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.
  - Se pueden configurar filtros de certificados, como se indica en el apartado 7.2 Configuración del filtro de certificados, y la selección automática de certificado, como se describe en el apartado 7.3 Selección automática de certificados.
- `successCallback`
  - Función *callback* JavaScript que se ejecutará cuando se seleccione un certificado. Esta función recibirá los siguientes parámetros:
    1. El certificado seleccionado codificado en Base64.
- `errorCallback`
  - Función *callback* JavaScript que se ejecutará cuando ocurra un error al seleccionar el certificado. Esta función recibirá hasta dos parámetros:
    1. En el primer parámetro se recibe un texto con el tipo del error.
    2. En el segundo parámetro se recibe un texto con el mensaje de error.

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 6.2.1 Firma de múltiples documentos (firma masiva)) 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.

A continuación, se muestran distintos ejemplos de la operación de selección de certificado:

- 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";

  AutoScript.selectCertificate (extraParams, enviarCertificado, mostrarError);
…
```

6.7 Recuperación de log

AutoFirma permite obtener las trazas de su propia ejecución para después para ayudar a las aplicaciones a identificar los problemas que sufran sus usuarios. Para poder obtener sus trazas de ejecución es necesario que se haya establecido la comunicación entre AutoFirma y el navegador web, por lo que este mecanismo no ayuda a identificar errores en el propio proceso de comunicación.

Este método debería utilizarse sólo de modo excepcional para identificar errores que se den AutoFirma al ejecutarse en el equipo del usuario y que sean susceptibles de ser atendidos por el equipo de soporte de su aplicación o por el propio equipo de soporte de AutoFirma.

Advertencia: La operación recuperación de logs no es compatible con los clientes de firma de Android e iOS, ni con AutoFirma cuando se comunica con el navegador a través de los servicios auxiliares.

La función JavaScript para recuperar las trazas de ejecución es:

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

En esta función:

- `successCallback`
  - Nombre de la función callback que se invocará en caso de recuperar correctamente la traza de ejecución.
  - Esta función recibirá los siguientes parámetros:
    1. Texto plano la traza de ejecución.
- `errorCallback`
  - Nombre de la función callback que se invocará en caso de finalizar con errores la carga de la traza de ejecución.

- Esta función recibirá como parámetros:
  1. El tipo de error que se produjo.
  2. El mensaje de error asociado.

A continuación, se un ejemplo de la operación recuperación de la traza de error:

- Envio de la traza de error tras un error en la firma

```
…
  // Función que se ejecutará cuando el proceso de firma falle
  function signErrorCallback (errorType, errorMessage) {
       // Llamamos a la función de la aplicación para mostrar errores
       showError(errorMessage);
       // Recuperamos y enviamos las trazas si no es un error de cancelación
       if (errorType == "es.gob.afirma.core.AOCancelledOperationException") {
             // No hacemos nada si falla la recuperación de las trazas
             AutoScript.getCurrentLog(enviarTrazasCallback);
       }
  }

  // Función callback que se ejecutara cuando se recuperen las trazas de error
  function enviarTrazasCallback (trace) {
       document.getElementById("trz").value =
                    AutoScript.getBase64FromText(trace);
       document.getElementById("formulario").submit();
  }
…
  // Llamamos a la operación de firma
  AutoScript.sign(null, "SHA512withRSA", "CAdES", null,
                  successCallback, signErrorCallback);
…
```

6.8 Operaciones de gestión de ficheros

Advertencia: Las operaciones de gestión de ficheros no son compatibles con los clientes de firma de Android e iOS. En cada uno de los siguientes subapartados se describen casos de uso alternativos para suplir el uso de los métodos aquí descritos y así mantener la compatibilidad con dispositivos móviles.

AutoFirma permite hacer uso de varias operaciones de carga y guardado de ficheros en disco. Estas operaciones están orientadas a facilitar a las aplicaciones la integración de las operaciones del cliente, sobre todo cuando se desea firmar un fichero que el usuario tenga en su posesión o cuando se desee permitir al usuario, justo después de firmar, guardar copia de la firma que haya generado.

El uso de estas operaciones se puede evitar mediante JavaScript y desarrollos en servidor, lo que por regla general darían lugar a aplicaciones más estables y una mejor experiencia de usuario. También

es probable que su aplicación sea candidata a obtener beneficiosos del uso de la firma trifásica. Consulte el apartado ANEXO II Firma trifásica para obtener más información sobre este tipo de operaciones. En los siguientes apartados se plantean también alternativas para el uso de las distintas operaciones, lo cual también hará la solución compatible con aplicaciones móviles.

6.8.1 Guardado de datos en disco

La función de guardado permite proporcionar a AutoFirma unos datos codificados en Base64 para que este ofrezca al usuario almacenarlo en disco mediante un diálogo de guardado. Un uso común de este método es guardar las firmas generadasalmacene el binario correspondiente en disco.

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 Base64 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`". Si lo que deseamos guardar es una firma o un certificado obtenido en una de las funciones *callback* de las operaciones del Cliente, proporcionaremos al método de carga tal como se recibe en la función.

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

```
saveDataToFile (dataB64, title, fileName, extension, description,
                successCb, errorCb);
```

En esta función:

- `dataB64`
  - Datos que deseamos almacenar codificados como cadena Base64.
  - 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.
  - Algunos sistemas operativos podrían no mostrar el título del diálogo.
- `fileName`
  - Nombre de fichero que aparecerá por defecto en el diálogo de guardado.
- `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.
- `sucessCb`
  - Nombre de la función *callback* JavaScript que se ejecutará en caso de que el guardado de datos finalice correctamente.
  - Si se omite este parámetro, o se establece a `null`, no se ejecutará ninguna función al terminar la operación.
  - Esta función no recibe parámetros.
- `errorCb`
  - Nombre de la función *callback* JavaScript que se ejecutará en caso de que el guardado de datos finalice con errores o cuando el usuario cancele el diálogo de guardado.
  - Si se omite este parámetro, o se establece a `null`, no se ejecutará ninguna función al terminar la operación.
  - Esta función recibe los siguientes parámetros:
    1. En el primer parámetro se recibe un texto con el tipo del error.
    2. En el segundo parámetro se recibe un texto con el mensaje de error.

A continuación, se muestran distintos ejemplos de operaciones de guardado de fichero:

- Guardado de una firma electrónica:

```
…
  function saveDataCb (dataB64, certB64) {
     AutoScript.saveDataToFile (dataB64, “Guardar firma electrónica”,
                                “firma.csig”, “csig”, “Firma binaria”);
  }
…

  AutoScript.cosign (dataB64, “SHA512withRSA”, “CAdES”, null, saveDataCb,
                     errorCb);
…
```

- Guardado de datos insertados por el usuario:

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

  AutoScript.saveDataToFile (dataB64, “Guardar”, “fichero.txt”, “txt”,
                             “Texto plano”);
…

  <!-- Formulario HTML con el texto que desea guardar -->
  <form>
        <textarea name="userText" cols="50" rows="5">Texto</textarea>
```

```
  </form>
…
```

La función de guardado permite únicamente almacenar en disco un dato que ya tenemos cargado en la página web y que, probablemente, ya esté en servidor o se desee enviar al mismo (como en el caso de las firmas electrónicas). Según el caso, se pueden implementar alternativas en la aplicación que eviten el uso de la función de guardado del Cliente @firma. Por ejemplo:

- Si tenemos que deseamos guardar en servidor, en lugar de cargarlos en la página y luego usar esta función, se podría habilitar un servicio para descargarlos directamente desde el navegador.
- Si tenemos los datos cargados en la página, como una firma recién generada, podemos enviarla primeramente al servidor para procesarla y guardarla. A continuación, podríamos informar al usuario del resultado de la operación y ofrecerle el descargar una copia utilizando el mismo método que en el punto anterior.
- Si tenemos los datos en la página página y no deseamos enviarla a servidor, podemos utilizar JavaScript para permitir al usuario descargar estos datos mediante un objeto Blob y un enlace creado al vuelo.

6.8.2 Selección y recuperación de un fichero por parte del usuario

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

La función JavaScript para la carga de ficheros en disco es:

```
getFileNameContentBase64 (title, extensions, description, filePath,
                          successCb, errorCb);
```

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 manualmente en el diálogo.
- `description`
  - Descripción del tipo de fichero que se espera cargar. Esta descripción aparecerá asociada a las extensiones indicadas.
- `filePath`

  - Ruta absoluta del fichero que se debería seleccionar por defecto o sólo el nombre de fichero sugerido.
- `successCb`
  - Función *callback* JavaScript que se invocará cuando se cargue correctamente un fichero.
  - Esta función recibe los siguientes parámetros:
    1. En el primer parámetro se recibe el nombre del fichero seleccionado.
    2. En el segundo parámetro se recibe el contenido del fichero codificado en Base64.
- `errorCb`
  - Función *callback* JavaScript de error que se invocará cuando ocurra un error al cargar un fichero o cuando el usuario cancele el diálogo de selección.
  - Esta función recibe los siguientes parámetros:
    1. En el primer parámetro se recibe un texto con el tipo del error.
    2. En el segundo parámetro se recibe un texto con el mensaje de error.

A continuación, se muestra un ejemplo de carga de fichero:

- Carga de un fichero y recogida de su nombre:

```
…
  var fileName;
  var fileContentB64;
  AutoScript.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);
  }
…
```

La función de carga de fichero permite cargar un fichero del disco del usuario y obtener su contenido y nombre en la página web. Sin embargo, este fichero no debería ser tramitado directamente sin haberse realizado las comprobaciones necesarias para saber que es válido para el objetivo que se desee, salvo que no tengamos ningún tipo de restricción sobre el tipo de fichero. También hay que tener en cuenta que, cuando cargamos datos para firmar, lo común suele ser enviar posteriormente a servidor esos datos junto con la firma generada. Según el caso, el uso del método de carga de ficheros se podría sustituir por alguna de las siguientes lógicas de negocio:

- Si queremos firmar un fichero y no tenemos ningún tipo de restricción sobre el tipo de datos que debe cargarse, podría omitirse la carga independiente de los datos y llamar al método de firma en cuestión sin pasarle datos. En ese caso, será la propia aplicación de firma la que ofrecerá al usuario cargar un fichero para firmarlo.
    - **Advertencia**: Este caso de uso no es compatible con el Cliente iOS.
- Si queremos cargar un fichero para firmarlo, pero antes queremos comprobar que sea un fichero válido, podríamos permitir que el usuario cargue los datos en la aplicación primeramente mediante un componente HTML de carga de fichero. Una vez cargado el documento en servidor podemos realizar las comprobaciones necesarias sobre él y cargar su contenido en el HTML de la página en Base64 para firmarlo. Después sólo deberíamos enviar al servidor la firma generada, ya que los datos ya estarían cargados.
    - Una versión mejorada de esto sería generar la firma de forma trifásica de tal forma que la aplicación no tenga que cargar el contenido del fichero en la página, se pueda firmar el documento mediante un identificador (haciendo uso de un DocumentManager a medida) y la firma se genere directamente en servidor, con lo cual tampoco sería necesario enviarla a posteriori. Consulte el apartado ANEXO II Firma trifásica para saber más sobre este tipo de firmas.

6.8.3 Selección y recuperación de múltiples ficheros por parte del usuario

Además de la operación de carga de un fichero, AutoFirma incorpora una función para la carga de múltiples ficheros que funciona de forma análoga a la anterior.

La función JavaScript para la carga de múltiples ficheros en disco es:

```
getMultiFileNameContentBase64 (title, extensions, description,
                               filePath,  successCb, errorCb);
```

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 manualmente en el diálogo.
- `description`
    - Descripción del tipo de fichero que se espera cargar. Esta descripción aparecerá asociada a las extensiones indicadas.
- `filePath`
    - Ruta absoluta de uno de los ficheros que se debería seleccionar por defecto o sólo el nombre sugerido de uno de los ficheros.

- `successCb`
  - Función *callback* JavaScript de éxito que se invocará cuando se cargue al menos un fichero.
  - Esta función recibe los siguientes parámetros:
    1. En el primer parámetro se recibe un *array* con los nombres de los ficheros seleccionados.
    2. En el segundo parámetro se recibe un *array* con el contenido en Base64 de los ficheros seleccionados.
  - El contenido de los parámetros recibidos será tal que el nombre del primer elemento del *array* de nombres se corresponderá con el contenido del primer elemento del *array* de contenidos, y así para cada uno de los elementos de ambos *arrays*.
- `errorCb`
  - Función *callback* JavaScript de error que se invocará cuando no se seleccionen ficheros o cuando se produzca un error durante su carga.
  - Esta función recibe los siguientes parámetros:
    1. En el primer parámetro se recibe un texto con el tipo del error.
    2. En el segundo parámetro se recibe un texto con el mensaje de error.

A continuación, se muestra un ejemplo de carga de fichero:

- Carga de ficheros y recogida de sus nombres:

```
…
  var fileNames;
  var fileContentB64s;
  AutoScript.getMultiFileNameContentBase64(
                  “Seleccionar fichero”, “jpg,gif,png”, “Imagen”, null,
                  loadDataSuccessCallback, loadDataErrorCallback
);
…
  function loadDataSuccessCallback(namesArray, contentB64sArray) {
       fileNames = namesArray;
       fileContentB64s = contentB64sArray;
  }
  function loadDataErrorCallback(type, msg) {
       /* Mostramos todos los errores menos el de cancelación del diálogo. */
       if (!“es.gob.afirma.core.AOCancelledOperationException”.equals(type)) {
             /* Método del integrador para mostrar logs */
             showLog(msg);
       }
  }
…
```

6.9 Métodos de utilidad JavaScript

Las siguientes secciones muestran diversas funciones de utilidad que incorpora el JavaScript de despliegue de Cliente @firma. Estas funciones no se comunican con la aplicación nativa de firma y pueden ser implementadas por su propia aplicación. Se suministran únicamente para facilitar la integración del Cliente. Entre estas funciones están:

- Conversión de texto a Base64 y viceversa.
- Descarga de datos.

6.9.1 Conversión de una cadena Base64 a texto

El JavaScript de despliegue del Cliente @firma proporciona un método para la conversión de una cadena Base64 a un texto plano. 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. El juego de caracteres que se usará para interpretar el texto siempre será UTF-8.

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

```
getTextFromBase64(dataB64);
```

En esta función:

- `dataB64`
  - Son los datos Base64 que se quieren mostrar como texto plano.

Este método devuelve una cadena con el texto plano correspondiente.

A continuación, se muestra un ejemplo de decodificación de un texto Base64:

- Decodificación de una firma XML generada previamente

```
…
  function showTextCallback (xmlSignB64) {
       var text = AutoScript.getTextFromBase64(xmlSignB64);
       alert(text);
  }
…
  AutoScript.sign(dataB64, “SHA1withRSA”, “XAdES”, “format=XAdES Enveloped”,
showTextCallback, errorCallback);
…
```

6.9.2 Conversión de un texto a cadena Base64

El JavaScript de despliegue del Cliente @firma proporciona un método para la conversión de un texto plano a una cadena Base64. Este método permite pasar al método de firma un texto insertado por el usuario, por ejemplo. Se interpretará que el texto proporcionado es UTF-8.

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

```
getBase64FromText(plaintText);
```

En esta función:

- `plainText`
  - Es el texto plano que se desea convertir a Base64.

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

A continuación, se muestra un ejemplo de decodificación de un texto Base64:

- Firma de un texto insertado por el usuario

```
…
  var text = "Hola Mundo!!";
  var dataB64 = AutoScript.getBase64FromText(text);

  AutoScript.sign (dataB64, "SHA512withRSA", "CAdES", null, successCallback,
                   errorCallback);
…
```

6.9.3 Descarga de datos remotos

El JavaScript de despliegue 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 un 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 se pueda recuperar el resultado de la operación.

La función JavaScript para descarga de datos remotos es:

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

En esta función:

- `url`
  - URL de acceso a los datos a descargar.
- `successCallback`
  - Nombre de la función *callback* que se invocará cuando se terminen de descargar los datos.
  - Esta función recibirá como único parámetro los datos descargados en Base64.
- `errorCallback`
  - Nombre de la función *callback* que se invocará cuando se produzca un error al descargar los datos. Esta función recibirá como parámetros:
    1. En el primer parámetro se recibe un texto con el tipo del error.
    2. En el segundo parámetro se recibe un texto con el mensaje de error.

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

A continuación, se muestra un ejemplo de descarga de datos:

- Descarga de datos y posterior firma

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

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

  var url = "http://midominio.com/informe.pdf";
  AutoScript.downloadRemoteData (url, iniciarFirma, mostrarError);
…
```

7 Configuración de las operaciones

7.1 Paso de parámetros adicionales

Los métodos del API JavaScript para el uso del Cliente @firma trasladan los datos a la aplicación de firma en forma de cadenas de texto. Algunas de estas cadenas son datos binarios codificados en Base64 (como el parámetro de datos a firmar), mientras que otras son textos planos (como los nombres del formato y el algoritmo de firma). En muchas de las operaciones del API existe un parámetro concreto, usualmente denominado `params`, que se utiliza para establecer una serie de propiedades de configuración opcionales. Estos parámetros se expresan en texto plano en forma de tuplas de claves y valores, emulando el formato de los objetos de propiedades de Java (`Properties`).

Los mencionados objetos de propiedades tienen un formato como el que sigue:

```
nombreParam1=valorParam1

nombreParam2=valorParam2

…
```

Para expresar cadenas en este formato desde JavaScript, se concatenarán cada una de las líneas usando el carácter especial de nueva línea (`\n`) como separador:

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

Es importante especificar el nombre de las propiedades exactamente como se indique, ya que puede existir diferenciación entre mayúsculas y minúsculas. Cualquier parámetro no soportado por la operación invocada, simplemente, se ignorará.

Nota: El Cliente @firma interpreta todos los textos, tanto los recibidos como los devueltos en las respuestas, usando el juego de caracteres UTF-8. Para poder transmitirlos y mostrarlos correctamente desde una página web es necesario que esta se encuentre codificada en UTF-8 y lo declare como tal.

En caso de no ser posible, se recomienda:

- Que el Base64 de los textos a proporcionar al Cliente se hayan obtenido desde un entorno en el que se pueda gantizar que originalmente estaban codificados en UTF-8. Por ejemplo, que el texto ya estuviese previamente codificado o que se codifique a través de un servicio.
- No mostrar directamente al usuario los mensajes devueltos por el propio Cliente.

Ejemplo de uso:

```
…
  AutoScript.cargarAppAfirma();

  // Configuramos 3 parámetros adicionales:
```

```
  // - expPolicy: Política de firma
  // - filters: Filtros de certificados vigentes
  // - signerClaimedRoles: Roles del firmante
  var extraParams = "expPolicy=FirmaAGE\nfilters=nonRepudiation:\n" +
                    "signerClaimedRoles=Apoderado";

  // Ejecutamos la operación de firma
  AutoScript.sign (dataB64, "CAdES", "SHA512withRSA", extraParams,
                   successCallback, errorCallback);
…
```

7.2 Configuración del filtro de certificados

El Cliente @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 distintas operaciones de firma y selección de certificados.

Por defecto, AutoFirma no mostrará al usuario los certificados caducados. Sin embargo, si se establecen filtros de certificados, se mostrarán todos aquellos certificados disponibles que cumplan con las condiciones dadas, incluidos los certificados caducados. Para omitir expresamente estos certificados se puede utilizar el filtro "`nonexpired:`", explicado más adelante en este apartado.

Advertencia: El uso de filtros de certificados no esta soportado por los clientes de firma móvil. La configuración de estas propiedades serán ignoradas por las aplicaciones móviles.

Las claves que nos permiten establecer filtros de certificados son:

- *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:false`
        - Motrar todos los certificados, incluso los caducados.
    - `filters=issuer.rfc2254: (O=DIRECCION GENERAL DE LA POLICIA);keyusage.nonrepudiation:true`
        - Mostrar sólo certificados de firma del DNIe.
    - `filters=issuer.rfc2254: (O=DIRECCION GENERAL DE LA POLICIA);keyusage.nonrepudiation:true;nonexpired:`

- Mostrar sólo certificados de firma del DNIe no caducados.

- *filters.X*: En esta clave 'X' será un entero igual o mayor que 1. El Cliente @firma 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. Ejemplos:
  - Mostrar los certificados no caducados del ciudadano con NIF 11111111H:
    - `filters.1=subject.rfc2254:(SERIALNUMBER=*11111111H);nonexpired:`
    - `filters.2=subject.rfc2254:(CN=*11111111H*);nonexpired:`
      - **IMPORTANTE:** Las autoridades certificadoras pueden agregar el DNI del ciudadano en distintos campos del certificado y AutoFirma no dispone de un modo de localizarlo. Sin embargo, la mayor de ellas ya agregan el DNI como parte del RDN "SERIALNUMBER" del subject del certificado. Algunas también lo agregan como parte del CN. Así, este ejemplo, englobaría funcionaria con buena parte de los certificados personales emitidos por autoridades españolas, pero puede que no con todas.
  - Mostrar los certificados CERES y el certificado de firma del DNIe:
    - `filters.1=nonexpired:true;issuer.rfc2254:(O=DIRECCION GENERAL DE LA POLICIA);keyusage.nonrepudiation:true`
    - `filters.2=nonexpired:;issuer.rfc2254:(O=FNMT-RCM)`

Estas claves de definición de filtros son excluyentes y tienen prioridad según el orden en el que se han listado (*filters* y *filters.X*). Es decir, si se establece la propiedad filters, no se procesarán las propiedades del tipo *filters.1*, por ejemplo.

Los filtros disponibles en AutoFirma son:

- Filtro de certificados caducados: Filtra los certificados para que no se muestren aquellos que se encuentren fuera de su periodo de validez, que son los únicos que pueden generar una firma válida. El filtro determina si un certificado está vigente en base a la fecha/hora del equipo del usuario.
  - Para establecer este filtro se usará el valor "*nonexpired:*".
  - Ejemplo:

- `filters=nonexpired:`

Un integrador también puede habilitar o deshabilitar expresamente este filtro siguiéndolo de las palabras "*true*" o "*false*", respectivamente.

Por defecto, si no se indica ningún filtro de certificados, AutoFirma omitirá los certificados que estén fuera de su periodo de validez. Si se desea que se muestren los certificados caducados sin indicar ningún otro filtro, se puede configurar la propiedad de filtrado de la siguiente manera:

- `filters=nonexpired:false`

- Filtro por número de serie de certificados cualificados de firma: Filtra los certificados del almacén para que sólo se muestre aquellos con un número de serie concreto, lo que comúnmente hará que se muestre un único certificado. En el caso de que el certificado identificado por el filtro no sea un certificado cualificado para firma, se buscará en el almacén un certificado parejo que sí lo esté. 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 se usará el valor "*qualified:*" seguido por el número de serie del certificado en hexadecimal.
  - Ejemplos:
    - `filters=qualified:45553a61`
    - `filters=qualified:03ea`
- Filtro en base a certificado codificado: 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 Base64. Esto es, tal como se devuelve a través 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.
- 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 el valor "*thumbprint:*" seguido 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.

- 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 se usará el valor "*dnie:*".
  - Ejemplo:
    - `filters=dnie:`
- Filtro de certificados SSCD: Filtra los certificados del almacén para que se muestren sólo aquellos generados en 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 se usará el valor "*sscd:*".
  - Ejemplo:
    - `filters=sscd:`
- Filtro de certificados de firma: Filtra los certificados del almacén para que se muestren todos los certificados salvo el certificado de autenticación del DNIe. Este filtro se crea para evitar excluir algunos tipos de certificados con los KeyUsage mal declarados.
  - Para establecer este filtro se usará el valor "`signingCert:`".
  - Ejemplo:
    - `filters=signingCert:`

  Para un filtrado más correcto de los certificados de firma, utilice el filtro "*keyusage.nonrepudiation:*".
- Filtro de certificados de autenticación: Filtra los certificados del almacén para que se muestren todos los certificados salvo el certificado de firma del DNIe. Este filtro se crea para evitar excluir algunos tipos de certificados con los KeyUsage mal declarados.
  - Para establecer este filtro se usará el valor "*authCert:*".
  - Ejemplo:
    - `filters=authCert:`

  Para un filtrado más correcto de los certificados de autenticación, utilice el filtro "*keyusage.digitalsignature:*".
- Filtro RFC2254 en base al *Subject* del certificado: Filtra los certificados a partir de una expresión regular creada según la RFC2254 que se aplica sobre el *Subject* del certificado.
  - Para establecer este filtro se usará el valor "*subject.rfc2254:*" seguido de la expresión RFC2254.

- Puede revisarse la normativa RFC 2254 en http://www.faqs.org/rfcs/rfc2254.html
- Ejemplo:
  - `filters=subject.rfc2254:(CN=*12345678z*)`
    - Este filtro mostrará sólo aquellos certificados en los que aparezca la cadena "12345678z" en el *CommonName* de su *Subject*.

- Filtro RFC2254 en base al *Issuer* del certificado: Filtra los certificados a partir de una expresión regular creada según la RFC2254 que se aplica sobre el *Issuer* del certificado.
  - Para establecer este filtro se usará el valor "*issuer.rfc2254:*" seguido de la expresión RFC2254.
  - Puede revisarse la normativa RFC 2254 en http://www.faqs.org/rfcs/rfc2254.html
  - Ejemplo:
    - `filters=issuer.rfc2254:(|(O=FNMT-RCM)(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 la FNMT.
  - 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.
  - Ejemplo:
    - `filters=subject.contains:JUAN ESPAÑOL ESPAÑOL`
      - Este filtro mostrará sólo aquellos certificados en los que aparezca la cadena "JUAN ESPAÑOL ESPAÑOL" en el *Subject*.
- Filtro de texto en base al *Issuer* del certificado: Filtra los certificados según si contienen o no una cadena de texto en el *Principal* de su *Issuer*.
  - Para establecer este filtro se usará el valor "*issuer.contains:*" seguido de la cadena de texto que debe contener.

- Ejemplo:
  - `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:
    - `filters=keyusage.digitalsignature:true;keyusage.keyencipherment:true`
      - Este filtro mostrará sólo aquellos certificados que tengan establecidos a `true` los *KeyUsage* `digitalsignature` (autenticación) y `keyencipherment` (sobres electrónicos), ignorando el valor del resto de *KeyUsages*.
    - `filters=keyusage.nonrepudiation:false`
      - Este filtro mostrará sólo aquellos certificados que no declaren el *KeyUsage* de firma avanzada.

- Filtro por identificador de directiva: 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 el 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`".

- Filtro de seudónimo: 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 el valor "*pseudonym:*".
  - Ejemplo:
    - `filters=pseudonym:`

- Filtro de almacenes externos: 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 el valor "*disableopeningexternalstores*".
  - Ejemplo:
    - `filters=disableopeningexternalstores`

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 ejecutará la operación de error indicando que no se ha encontrado ningún certificado que cumpla con los criterios indicados.

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 7.3 Selección automática de certificados para conocer cómo configurar la propiedad "*headless*".

7.3 Selección automática de certificados

Para aquellos casos en los que sólo exista un certificado en el almacén de certificados o cuando se descarten certificados mediante filtros y sólo haya uno que es posible seleccionar, es posible indicar a AutoFirma 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*.

```
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.

7.4 Configuración de la política de firma

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. Los formatos de firma CAdES, PAdES y XAdES permiten declarar la política de firma que se ha seguido para su generación.

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.

7.4.1 Política de firma de la AGE v1.9

En el Cliente @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 propiedad adicional en la operación de firma deseada.

```
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 para cada uno de los formatos comprendidos en la misma los siguientes:

- CAdES
  - `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.administracion.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.administracion.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.administracion.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 más información sobre la política de firma de la AGE puede consultar la guía de implementación de la política:
https://administracionelectronica.gob.es/ctt/politicafirma#.YBEdyzr0mbg.

Para saber cómo configurar propiedades en las operaciones de firma, consulte el apartado 7.1 Paso de parámetros adicionales.

7.4.2 Política de firma de Factura electrónica (Facturae)

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

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

7.5 Validación de firmas previas

Las operaciones de cofirma y contrafirma se realizan sobre firmas generadas anteriormente. Salvo en casos concretos, el Cliente @Firma permite agregar nuevas firmas a cualquier firma compatible sin restricción. Esto implica que, por ejemplo, se podría contrafirmar una firma invalida por estar tener un certificado caducado.

Es posible que desde su aplicación quiera restringirse, en la medida de lo posible, el que se cofirmen y contrafirmen firmas inválidas, para lo cual puede usarse la opción *checkSignatures*.

```
checkSignatures=true
```

Esta propiedad realiza una verificación criptográfica de la firma y comprueba la caducidad de los certificados utilizados en ella. Esto **no es una validación completa de firma**, ya que para considerar

que la firma es válida deben realizarse otras operaciones no soportadas por el Cliente @firma, como la comprobación del estado de revocación de los certificados o la adhesión de la firma a la política de firma que declare.

Esta propiedad debe usarse para reducir la posibilidad procesar firmas inválidas, pero es responsabilidad de la aplicación saber si son válidas antes de enviarse a procesar o, si es el usuario el que selecciona expresamente estas firmas, el realizar una validación posterior.

Si durante el proceso de cofirma o contrafirma se detecta que la firma de entrada no es válida, el nuevo proceso de firma fallará y se notificará el error a través del método *callback* configurado.

Si se pudiese completar el proceso de validación por algún motivo, el proceso de multifirma continuará normalmente. Este, por ejemplo, sería el caso la contrafirma de una firma CAdES explícita, en la que no se encuentran los datos y, por lo tanto, no se puede verificar la validez del hash firmado.

La verificación de las firmas previas es una operación compatible con los siguientes formatos de firma:

- CAdES
- XAdES
- PAdES

La verficación se realiza tanto en la generación de firmas monofásicas como trifásicas. En el caso de las firmas monofásicas, la validación se realiza al inicio del proceso de firma, mientras que en las firmas trifásicas el proceso se realiza en servidor. Por este motivo, al realizar firmas trifásicas se le pedirá el certificado al usuario incluso si la firma no es válida, ya que esto es algo que no se identificará hasta más adelante en el proceso de firma.

8 Formatos de firma

El Cliente @firma permite la generación de firmas en diversos formatos con diversos perfiles básicos. Las firmas en estos formatos, pueden ser mejoradas a posteriori con otros productos para incluir la información longeva de firma.

Los formatos avanzados soportados son:

- **CAdES**
  - Formato avanzado de firma binaria.
- **XAdES**
  - Formato avanzado de firma XML.
- **PAdES**
  - Formato avanzado de firma de documentos PDF.
- **FacturaE**
  - Formato para la firma de facturas electrónicas. Se trata de una firma XAdES especialmente adaptada para cumplir los requisitos de firma de las facturas electrónicas.

El Cliente @firma soporta por retrocompatibilidad otros formatos de firma, pero ninguno de estos formatos se recoge en la política de firma de la AGE, su uso está desaconsejado y no se proporciona soporte sobre los mismos:

- CMS
  - Formato de firma binaria no avanzado.
  - Se recomienda sustituir por el formato CAdES, con el que es compatible.
- XMLdSig
  - Formato de firma XML no avanzado.
  - Se recomienda sustituir por el formato XAdES, con el que es compatible.
- ODF
  - Formato de firma basado en XMLdSig y utilizado por OpenOffice/LibreOffice.
  - Se recomienda sustituirlo por firmas PAdES sobre documentos PDF para seguir gestionándolas junto al documento firmado o por firmas CAdES o XAdES si su sistema puede gestionar por separado el documento y la firma electronica.
- OOXML
  - Formato de firma basado en XAdES y utilizado por Microsoft Office.
  - Se recomienda sustituirlo por firmas PAdES sobre documentos PDF para seguir gestionándolas junto al documento firmado o por firmas CAdES o XAdES si su sistema puede gestionar por separado el documento y la firma electronica.

En los siguientes apartados, se proporciona información adicional de los principales formatos de firma soportados, junto con el listado de opciones del Cliente @firma para configurarlos.

8.1 Configuración de firmas CAdES

Las firmas CAdES generadas por el Cliente @firma son, por defecto, acordes a todas las siguientes versiones del formato de firma:

- v1.7.3 (ETSI TS 101 733 v1.7.3)
- v1.7.4 (ETSI TS 101 733 v1.7.4)
- v1.8.1 (ETSI TS 101 733 v1.8.1)
- v1.8.3 (ETSI TS 101 733 v1.8.3)
- v2.1.1 (ETSI TS 101 733 v2.1.1)
- v2.2.1 (ETSI TS 101 733 v2.2.1)

Las firmas CAdES no declaran cuál es su versión de formato de firma, por lo que puede considerarse que una misma firma se ajusta a todas las versiones anteriormente listadas.

El Cliente @firma permite generar firmas CAdES acordes a los siguientes perfiles de firma:

- CAdES-BES
  - Todas las firmas CAdES generadas por el Cliente @firma sin política de firma son consideradas CAdES-BES.
- CAdES-EPES
  - Todas las firmas CAdES generadas por el Cliente @firma con política de firma son consideradas CAdES-EPES.

Hay que tener en cuenta que algunas de las firmas CAdES generadas por el Cliente @firma también pueden considerarse de tipo B-Level. Sin embargo, el Cliente @firma no incluye un modo de operación que permita asegurar que las firmas generadas sean acordes a este perfil.

Algunas de las propiedades de configuración listadas en el apartado 8.1.3 Parámetros adicionales pueden afectar a la compatibilidad de las firmas generadas con algunas versiones o perfiles del formato. Consulte el apartado específico de cada propiedad para saber si esta afecta o no a la compatibilidad.

Las firmas CAdES que el Cliente @firma genera por defecto no incluyen ni los datos firmados (firma explícita) ni política de firma y se ajusta al perfil CAdES-BES.

8.1.1 Algoritmos de firma

Las firmas CAdES generadas por el Cliente @firma aceptan los siguientes algoritmos de firma:

- `SHA512withRSA`
- `SHA384withRSA`
- `SHA256withRSA`
- `SHA1withRSA` (No recomendado)

No es recomendable usar el algoritmo `SHA1withRSA` por estar obsoleto y ser vulnerable, mientras que el algoritmo

El algoritmo más seguro y, por lo tanto, el recomendado para su uso es `SHA512withRSA`.

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.

8.1.2 Firmas CAdES implícitas o explícitas

Las firmas CAdES pueden incluir internamente una copia de los datos firmados (firmas implícitas) o no incluirlos (firmas explícitas o "*detached*"). El Cliente @firma por defecto genera firmas explícitas, más pequeñas en tamaño, pero es posible que desee generar firmas implícitas para disponer en un sólo fichero de los datos y la firma electrónica, así como tener toda la información necesaria para la validación completa de la firma. Para generar firmas implícitas, debe indicar el siguiente parámetro adicional:

```
mode=implicit
```

8.1.3 Parámetros adicionales

A continuación, se detallan los parámetros adicionales que aceptan cada una de las operaciones de firma.

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

8.1.3.1 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. En las cofirmas, este parámetro se ignorará si los datos ya estaban contenidos en la firma original o si no se proporcionan los datos.

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.
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).
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`	`true`	Se incluirá el atributo SigningCertificateV2 en la firma.
	`false` (U otro valor)	Se incluirá el atributo SigningCertificate en la firma
	Sin especificar	Se incluirá SigningCertificate si la firma utiliza un algoritmo de firma SHA1 y SigningCertificateV2 para el resto de algoritmos.

8.1.3.2 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.
`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).
`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).

8.2 Configuración de firmas XAdES

Las firmas XAdES generadas por defecto con el Cliente @firma son acordes a la versión 1.3.2 del formato de firma (ETSI TS 101 903 v1.3.2).

Es posible configurar las operaciones de firma para que se generen conforme a la versión 1.4.1 del formato (ETSI TS 101 903 v1.4.1). Para ello, será necesario configurar los parámetros adicionales `xadesNamespace` y `signedPropertiesTypeUrl`.

El Cliente @firma permite generar firmas acordes a los siguientes perfiles:

- XAdES-BES
  - Todas las firmas XAdES generadas por el Cliente @firma sin política de firma son consideradas XAdES-BES.
- XAdES-EPES
  - Todas las firmas XAdES generadas por el Cliente @firma con política de firma son consideradas XAdES-EPES.

Hay que tener en cuenta que algunas de las firmas XAdES generadas por el Cliente @firma también pueden considerarse de tipo B-Level. Sin embargo, El Cliente @firma versión 1.7 no incluye un modo de operación que permita asegurar que las firmas generadas sean acordes a este perfil.

Con independencia del perfil de firma, es posible realizar las firmas XAdES en cuatro modos diferentes:

- *Enveloping* (Por defecto)
- *Enveloped*
- *Internally Detached* (también referida en este documento como *Detached*)
- *Externally Detached*

Algunas de las propiedades de configuración listadas en el apartado 8.2.7 Parámetros adicionales pueden afectar a la compatibilidad de las firmas generadas con algunas versiones del formato o perfiles de firma. Consulte el apartado específico de cada propiedad para saber si esta afecta o no a la compatibilidad.

8.2.1 Algoritmos de firma

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

- `SHA512withRSA`
- `SHA384withRSA`
- `SHA256withRSA`
- `SHA1withRSA` (No recomendado)

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`.

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.

8.2.2 Algoritmos de huella digital para las referencias

XAdES hace cálculos de huella digital (*hash*) 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`.

En el caso de las referencias de las firmas con manifest, el algoritmo de huella se toma del parámetro adicional `precalculatedHashAlgorithm`. En caso de que este parámetro no se indique, se utilizaría el algoritmo del parámetro `referencesDigestMethod` o, de no estar configurado, el algoritmo SHA-512.

Consulte la sección "8.2.7 Parámetros adicionales" para conocer los valores que pueden adoptar estos parámetros.

8.2.3 Situación del nodo de firma en XAdES Enveloped

Cliente @firma sitúa por defecto la firma electrónica en las firmas XAdES Enveloped en un 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 otra posición del XML.

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

Por ejemplo, en el siguiente XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
</bookstore>
```

Si indicamos el parámetro con este valor:

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="COOKING">
  <title lang="en">
    Everyday Italian
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="S1">
      …
    </ds:Signature>
  </title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
```

```
  <year>2005</year>
  <price>29.99</price>
</book>
</bookstore>
```

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

```
//*[namespace-uri()='http://www.facturae.es/Facturae/2007/v3.1/Facturae']
```

8.2.4 Transformaciones sobre el contenido a firmar

Es posible declarar transformaciones adicionales sobre el contenido a firmar. Esto se realizará mediante las siguientes propiedades:

- `xmlTransform`***n***`Type`
  - Tipo de transformación.
- `xmlTransform`***n***`Subtype`
  - Subtipo de transformación.
- `xmlTransformnBody`:
  - Transformación específica.

Las combinaciones de valores que pueden adoptar estas propiedades son:

- 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 decodifican 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=…
```

8.2.5 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* no es responsabilidad del sistema validador, sino de la aplicación que generó la firma o una delegada por la misma. Al unir esto con el uso de referencias a los datos externas a la firma, obtenemos la ventaja de que la firma no contendrá los datos firmados (muy recomendable para trabajar con datos grandes) y que la plataforma validadora no tiene que acceder a los datos para comprobar la validez de la propia firma. Consulte la especificación XMLDSig para mayor información.

De igual manera que el sistema validador no tiene que acceder a los datos referenciados desde el *manifest*, no será responsabilidad del Cliente @firma acceder a estos datos para su análisis y el cálculo de su huella digital. La propia aplicación que ordena la firma será la encargada de proporcionar al Cliente @firma las referencias a los datos y la huella digital de los mismos.

Las referencias a los datos firmados en un Manifest se expresan mediante una URI. Esta URI no tiene necesariamente que ser una URL, así puede estructurarse de la manera necesaria para ayudar a determinar inequívocamente cuales son los datos que se firman. Por ejemplo, podríamos construir una URI en forma de URN, en la que referenciásemos a un recurso concreto dentro de uno de nuestros gestores de contenido, para lo que podríamos crear una URN a medida que tuviese los datos necesarios para identificar el repositorio de datos y el recurso en cuestión:

```
urn:rp:mirp:asset:CDC8D258
```

Un sistema de validación externo no resolverá la URI que referencia a los datos desde un *Manifest*, ya que puede que esta sólo sea accesible y/o comprensible desde el entorno en el que se generó la firma. Por ende, la propia aplicación que firma los datos mediante *Manifest* o una aplicación delegada

por la misma debe poder asegurar que los datos referenciados mediante la URI no cambiaran y ser capaz de comprobarlos calculando su huella digital (*hash*).

Para aprovechar todas las ventajas de las firmas *Manifest*, **todas las firmas *Manifest* generadas por el Cliente @firma serán *externally detached***, independientemente de la configuración establecida.

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 asignado la huella digital proporcionada como la correspondiente a los datos.

8.2.5.1 Generar firma manifest con el Cliente @firma

Para crear firmas XAdES con estructuras *manifest* desde el Cliente @firma debe especificarse el parámetro adicional `useManifest` con el valor "`true`".

Adicionalmente, se deberá indicar la URI y la huella digital (*hash*) de los datos firmados. El Cliente @firma permite firmar más de un dato simultáneamente mediante *manifest* así que será posible indicar más de una URI y huella digital al configurar una firma *manifest*. Como resultado, se obtendrá una única firma que engloba todas las referencias. Para que esta firma tuviese validez completa, todos los datos referenciarlos deberían mantenerse sin cambios a lo largo del tiempo.

Para generar firmas manifest deberán indicarse a través de los parámetros adicionales de la aplicación las propiedades que se listan a continuación por cada referencia a firmar. Para ello, se

sustituirá la 'X' del nombre del parámetro por el número de referencia en cuestión (empezando en 1):

- `uriX`
  - Obligatorio. URI que referencia a los datos.
- `mdX`
  - Obligatorio. Huella digital (hash) en Base64 de los datos. El algoritmo de huella digital se indicará mediante el parámetro "`precalculatedHashAlgorithm`".
- `mimeTypeX`
  - Opcional. MimeType correspondiente a los datos referenciados. Si no se indica, se usará "`application/octet-stream`".
- `contentTypeOidX`
  - Opcional. OID correspondiente al tipo de dato referenciado. Si no se indica, se intentará extrapolar a partir del MimeType. En caso de no conseguirlo, no se usará ninguno.
- `encodingX`
  - Opcional. URI identificadora de la codificación de los datos si estuviesen codificados. Por defecto, no se usa ninguna.
- `precalculatedHashAlgorithm`
  - Algoritmo de huella digital que se ha utilizado para calcular la huella de los datos. Este parámetro aplica a todas las referencias insertadas. Si no se indica, se hereda la configuración del parámetro "`referencesDigestMethod`". Si este otro parámetro tampoco se indicase se interpretará que las huellas digitales se han calculado con el algoritmo SHA-512.

Al realizar firmas *manifest* no será necesario indicar datos en el parámetro de datos del método de firma del Cliente. El propio Cliente detectará que se va a realizar una firma manifest y no solicitará estos datos al usuario.

Por ejemplo, podríamos hacer una firma *manifest* de unos datos usando los siguientes parámetros adicionales (*extraParams*):

```
useManifest=true
uri1=urn:id:3086
md1=4hrn/3Y9c/fn/uyq12w+D9A2aKc=
mimeType1=plain/xml
precalculatedHashAlgorithm=SHA-1
```

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

```
AutoScript.sign (null, "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.

8.2.6 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`. Las reglas que se siguen para procesar las hojas de estilo son las siguientes:

Referencia / Formato	**XAdES *Enveloped***	**XAdES *Enveloping***	**XAdES *Internally Detached***	**XAdES *Externally Detached***
Ruta relativa a la hoja de estilo	No se firma	No se firma	No se firma	No se firma
Ruta absoluta a la hoja de estilo	Se incluye la declaración a la hoja de estilo y se firma una referencia canonizada a la hoja de estilo.	Se firma una referencia canonizada a la hoja de estilo.	Se firma una referencia canonizada a la hoja de estilo.	No se firma
Hoja de estilo empotrada	Se incluye la declaración a la hoja de estilo	No es necesaria ninguna acción	No es necesaria ninguna acción	No se firma

8.2.7 Parámetros adicionales

A continuación, se detallan los parámetros adicionales que aceptan cada una de las operaciones de firma.

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

8.2.7.1 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. Se ignora en la operación de cofirma. 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)
`uri`***n***	[URI]	URI que referencia a los datos que se desean firma dentro de una firma *manifest*. '***n***' indica el número de referencia de entre las que se quieren firmar, empezando en '1'.
`md`***n***	[Texto Base64]	MIME-Type de los datos asociados a la referencia '***n***' en una firma *manifest*. Si no se indica este parámetro, se utilizará el tipo 'application/octet-stream'.
`mimeType`***n***	[Texto en formato MIME-Type]	MIME-Type de los datos asociados a la referencia '***n***' en una firma *manifest*. Si no se indica este parámetro, se utilizará el tipo 'application/octet-stream'.
`contentTypeOid`***n***	[OID o URN de tipo OID]	Identificador del tipo de dato firmado para la referencia número '***n***' en una firma *manifest*. Este parámetro es complementario (que no excluyente) al parámetro `mimeTypeX`.
`encoding`***n***	[URI]	Codificación de los datos asociados a la referencia número '***n***' en una firma manifest. Un uso incorrecto de este parámetro puede provocar la generación de una firma inválida.
`precalculatedHashAlgorithm`	`SHA1`	Indica que las huellas digitales de los datos referenciados en el *manifest* se calcularon mediante el algoritmo SHA1.

	SHA-256	Indica que las huellas digitales de los datos referenciados en el *manifest* se calcularon mediante el algoritmo SHA-256.
	SHA-384	Indica que las huellas digitales de los datos referenciados en el *manifest* se calcularon mediante el algoritmo SHA-384.
	SHA-512	Indica que las huellas digitales de los datos referenciados en el *manifest* se calcularon mediante el algoritmo SHA-512.
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*.
	XAdES Externally Detached	Genera firmas en formato *Externally 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.
	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. Este es el comportamiento por defecto.
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	http://uri.etsi.org/01903/v1.3.2#	URL de definición del espacio de nombres de XAdES correspondiente a la versión 1.3.2 de XAdES. Este es el valor por defecto.

	`http://uri.etsi.org/01903/v1.4.1#`	URL de definición del espacio de nombres de XAdES correspondiente a la versión 1.4.1 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`	`http://uri.etsi.org/01903#SignedProperties`	URL de definición del tipo de las propiedades firmadas (*Signed Properties*) de XAdES. Este es el valor por defecto.
	`http://uri.etsi.org/01903/1.3.2#SignedProperties`	URL de definición del tipo de las propiedades firmadas (*Signed Properties*) de XAdES v1.3.2.
	`http://uri.etsi.org/01903/1.4.1#SignedProperties`	URL de definición del tipo de las propiedades firmadas (*Signed Properties*) de XAdES v1.4.1. Si se establece este parámetro es posible que se necesite establecer también el parámetro `xadesNamespace` para evitar incoherencias en la versión de XAdES.
`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/xmldsig#base64	Indica que los datos que se proporcionan para firmar ya están codificados en Base64 y se debe 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).

8.2.7.2 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	[Texto 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).

8.3 Configuración de firmas PAdES

El Cliente @firma permite generar firmas PAdES acordes a las partes 2 y 3 del estándar ETSI TS 102 778 V1.2.1.

Los perfiles de firma soportados son los descritos en el mencionado estándar:

- PAdES-Básico
  - Las firmas PAdES generadas por defecto son PAdES Básicas.
- PAdES-BES
  - Las firmas PAdES en las que se declara el subfiltro "`ETSI.CAdES.detached`" son PAdES-BES.
- PAdES-EPES
  - Las firmas PAdES en las que se configura política de firma son PAdES-EPES.

Hay que tener en cuenta que algunas de las firmas PAdES generadas por el Cliente @firma también pueden considerarse de tipo B-Level. Sin embargo, no se incluye un modo de operación que permita asegurar que las firmas generadas sean acordes a este perfil.

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.

8.3.1 Algoritmos de firma

Las firmas PAdES aceptan los siguientes algoritmos de firma:

- `SHA512withRSA`
- `SHA384withRSA`
- `SHA256withRSA`
- `SHA1withRSA` (No recomendado)

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

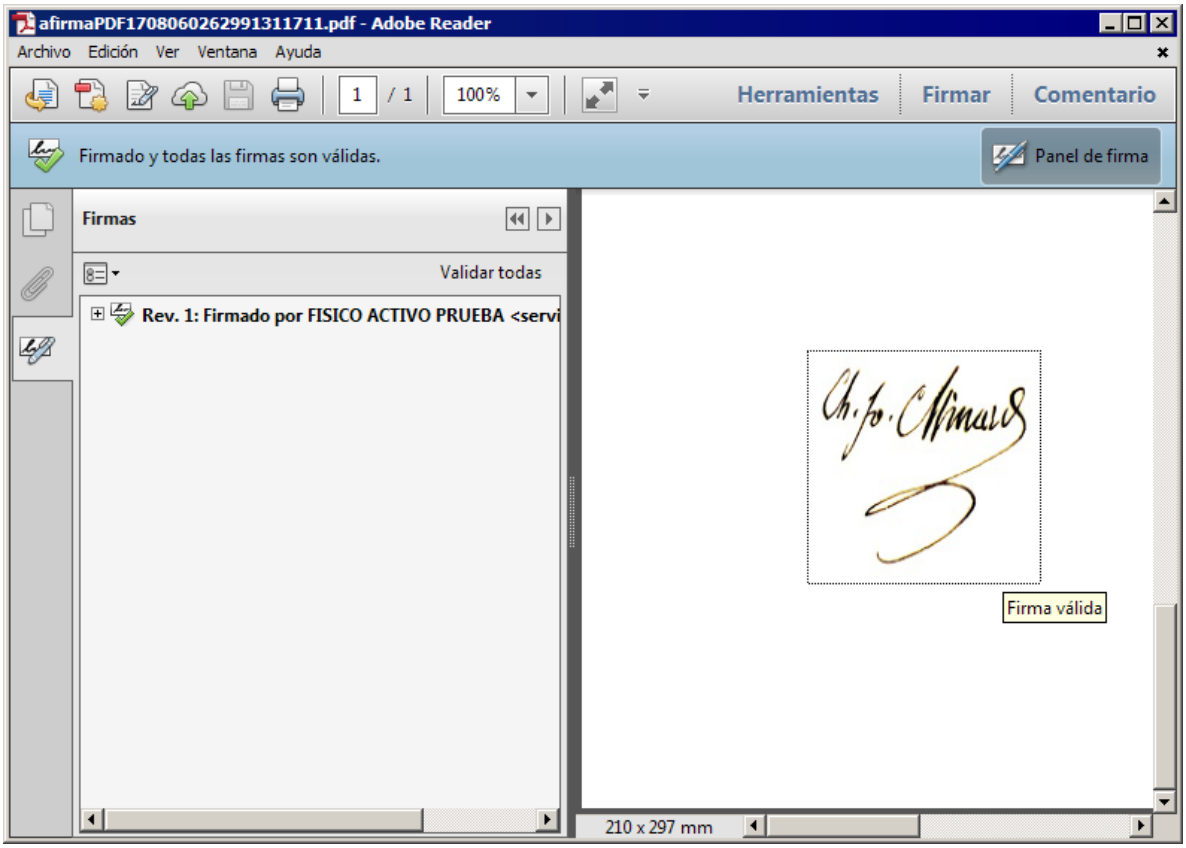
Si los certificados del usuario se encuentran en tarjeta inteligente, asegúrese de disponer de la última versión de su controlador para garantizar la compatibilidad con estos algoritmos de firma. En caso de que, aún así, no pueda utilizar este algoritmo con su tarjeta inteligente, consulte la información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

8.3.2 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.

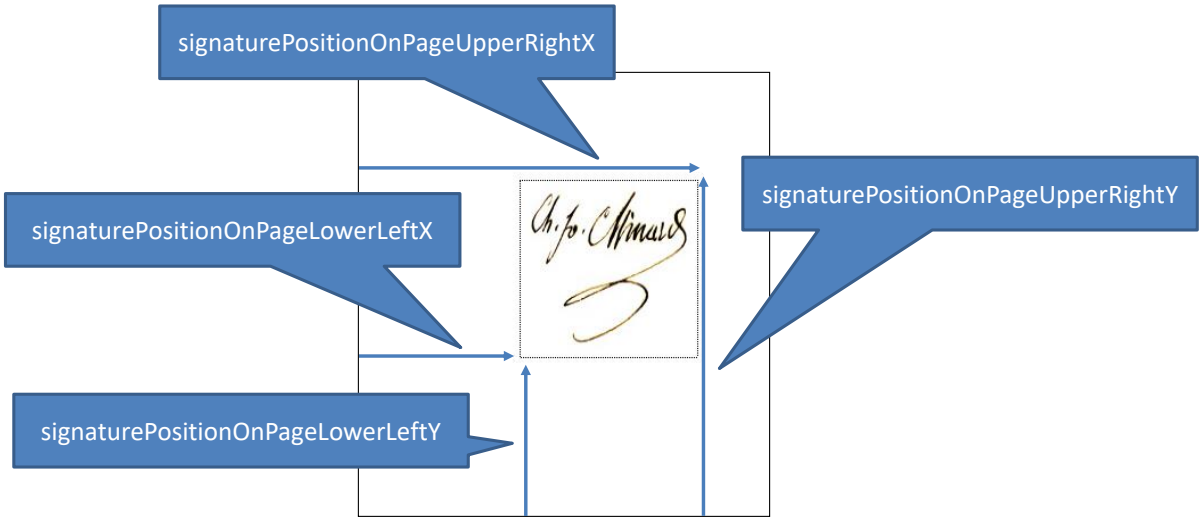
8.3.3 Creación de una firma visible

El 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 las coordenadas dentro de esta. Cada firma del documento, sólo puede tener una representación visible en una de sus páginas.

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

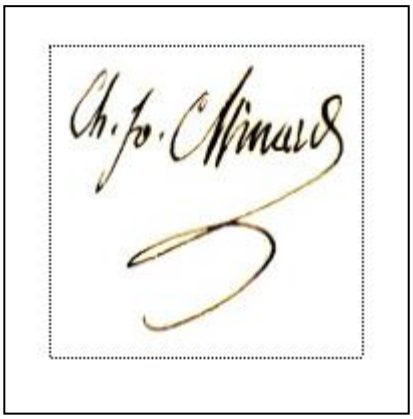


Las coordenadas del campo de firma y la página en la que se desea insertar se establecen usando los parámetros adicionales, por ejemplo:

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

Los documentos PDF comienzan su numeración de páginas desde uno (1). Si se indica un valor de página negativo, se empezará a contar desde la última página del documento hacia atrás. Por ejemplo, si se configura en el número de página el valor -1, la firma se insertará en la última página del documento. Si se configura el valor -2, se insertará en la penúltima página.

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



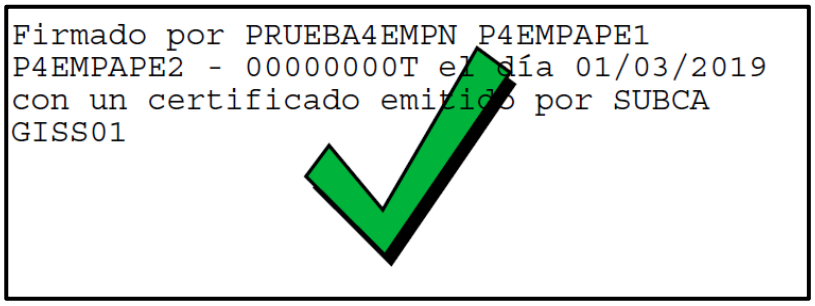
- Una imagen:
  - En este caso debe indicarse qué imagen usar aportando el binario en formato JPEG codificado en Base64.
    - La imagen de firma se configura a través del parámetro adicional `signatureRubricImage`.
  - La imagen se deforma para adaptarse a las dimensiones del recuadro marcado por las 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, 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.
  - El texto puede incluir una serie de patrones que serán sustituidos en el momento de la firma:
    - **$$SUBJECTCN$$** Nombre común (CN, Common Name) dentro del X.500 Principal del titular del certificado de firma. Salvo que se indique lo contrario, los identificadores de usuario encontrados en este parámetro se procesarán para ocultar parte de sus caracteres.
    - **$$ISSUERCN$$** 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.

Las propiedades para configurar la visualización de un texto de la firma se listan a continuación:

- `layer2Text`
  - Texto a escribir dentro de la firma visible.
- `layer2FontFamily`
  - Tipo de letra a usar en el texto de la firma visible.
- `layer2FontSize`
  - Tamaño de letra a usar en el texto de la firma visible.
- `layer2FontStyle`
  - Estilo del tipo de letra a usar en el texto de la firma visible. Cada estilo se identifica mediante un valor numérico y es posible combinar estilos aplicando la operación lógica *o* sobre los valores numéricos de cada uno de ellos.
- `layer2FontColor`
  - Color del texto de la firma visible.
- `obfuscateCertText`
  - Indica si deben ofuscarse o no los identificadores de usuario localizados en el texto visible del PDF.
- `signatureRotation`
  - Número de grados que rotar el texto del campo de firma en sentido horario.
  - En caso de indicar rotación, se ignorará cualquier configuración de imagen de firma.
- `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. Por ejemplo, la marca podría no mostrarse cuando se definiese una imagen de fondo en la firma.



Consulte el apartado 8.3.7 Parámetros adicionales para saber más sobre los valores que se pueden asignar a las propiedades anteriores.

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

El Cliente @firma permite, principalmente como ayuda para la inserción de Códigos Seguros de Verificación (CSV), insertar una imagen en un documento PDF justo antes de firmarlo.

Para agregar una imagen debemos configurar una página y una zona dentro de esta para insertarla, usando para ello el mismo sistema de coordenadas descrito en el apartado 8.3.3 Creación de una firma visible, 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.

Debe tenerse en cuenta que el agregar imágenes al PDF puede invalidar firmas previas que tuviese el documento. Asegúrese de no utilizar esta funcionalidad cuando el documento ya contenga firmas.

Es importante recalcar también 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.
- `imagePage`
  - Página donde desea insertarse la imagen.
- `imagePositionOnPageLowerLeftX`
  - Coordenada horizontal inferior izquierda de la posición de la imagen.
- `imagePositionOnPageLowerLeftY`
  - Coordenada vertical inferior izquierda de la posición de la imagen.

- `imagePositionOnPageUpperRightX`
  - Coordenada horizontal superior derecha de la posición de la imagen.
- `imagePositionOnPageUpperRightY`
  - Coordenada vertical superior derecha de la posición de la imagen.

Consulte el apartado 8.3.7 Parámetros adicionales para obtener más información sobre los posibles valores que pueden adoptar estos parámetros.

8.3.5 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 pueden firmar como parte de un proceso de firma masiva documentos PDF cifrados.
- 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/

8.3.6 Documentos certificados

Las firmas de un PDF pueden ser catalogadas como firmas de aprobación (por defecto) o firmas certificadas.

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 al conjunto de datos de todo el documento, nunca sólo a los datos de un formulario. 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`. Los tipos de firma que puede crear son:

- Firma sin certificar.

  - Esta sería una firma de aprobación y es el valor Es el valor por defecto.
  - Este es el tipo de firma generada por defecto por el Cliente y se configura con el valor: `0`
- 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).
  - Se configura con e valor: `1`
- 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).
  - Se configura con e valor: `2`
- 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.
  - Se configura con e valor: `3`

8.3.7 Parámetros adicionales

A continuación, se listan las propiedades adicionales que pueden configurarse en las firmas en formato PAdES con el Cliente @firma.

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
`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.
`alwaysCreateRevision`	`true`	Siempre creará una revisión al firmar. Requiere que el documento cumpla la especificación PDF 1.7 (ISO 32000-1:2008)
	`false`	No creará revisión en la primera firma y sí en las siguientes.
`image`	[Texto Base64]	Imagen JPEG que insertar en el PDF.
`imagePage`	[Entero positivo]	Insertar imagen en el número de página indicado.
	`0`	Insertar en todas las páginas.
	`-1`	Insertar imagen en la última página.

imagePositionOnPageLowerLeftX	[Entero positivo]	Coordenada horizontal desde la esquina inferior izquierda de la página a la esquina inferior izquierda de la imagen.
imagePositionOnPageLowerLeftY	[Entero positivo]	Coordenada vertical desde la esquina inferior izquierda de la página a la esquina inferior izquierda de la imagen.
imagePositionOnPageUpperRightX	[Entero positivo]	Coordenada horizontal desde la esquina inferior izquierda de la página a la esquina superior derecha de la imagen.
imagePositionOnPageUpperRightY	[Entero positivo]	Coordenada vertical desde la esquina inferior izquierda de la página a la esquina superior derecha de la imagen.
attach	[Texto Base64]	Contenido a añadir como adjunto al PDF. Requiere establecer attachFileName.
attachFileName	[Texto]	Nombre del que asignar al fichero adjunto.
attachDescription	[Texto]	Descripción del documento adjunto.
certificationLevel	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	true	Comprime el PDF firmado para que ocupe menos tamaño. Sólo se aplica si se trata de un PDF v4 o superior. Este es el valor por defecto.
	false	Nunca se comprime el PDF firmado.
pdfVersion	2	Se declara que la versión del PDF de salida es 1.2.
	3	Se declara que la versión del PDF de salida es 1.3.
	4	Se declara que la versión del PDF de salida es 1.4.
	5	Se declara que la versión del PDF de salida es 1.5.
	6	Se declara que la versión del PDF de salida es 1.6.
	7	Se declara que la versión del PDF de salida es 1.7.
signatureSubFilter	[Texto]	Subfiltro declarado. Por defecto se utiliza el de las firmas básicas (adbe.pkcs7.detached). Puede usarse la cadena "ETSI.CAdES.detached" para crear firmas BES.
signatureField	[Texto]	Nombre del campo de firma preexistente en el que insertar la firma.
signaturePage	[Entero positivo]	Se genera una firma visible PDF en el número de página indicado.

	-1	Se genera una firma visible PDF en la última página. Este es el comportamiento por defecto.
	-2	Agrega una nueva página al documento y agrega la firma visible PDF en ella.
signaturePositionOnPageLowerLeftX	[Entero positivo]	Coordenada horizontal desde la esquina inferior izquierda de la página a la esquina inferior izquierda del campo de firma visible.
signaturePositionOnPageLowerLeftY	[Entero positivo]	Coordenada vertical desde la esquina inferior izquierda de la página a la esquina inferior izquierda del campo de firma visible.
signaturePositionOnPageUpperRightX	[Entero positivo]	Coordenada horizontal desde la esquina inferior izquierda de la página a la esquina superior derecha del campo de firma visible.
signaturePositionOnPageUpperRightY	[Entero positivo]	Coordenada vertical desde la esquina inferior izquierda de la página a la esquina superior derecha del campo de firma visible.
signatureRubricImage	[Texto Base64]	Imagen JPEG que mostrar en el campo de firma visible.
layer2Text	[Texto]	Texto que mostrar en el campo de firma visible.
layer2FontFamily	0	El texto de la firma visible se mostrará con fuente Courier. Este es el valor por defecto.
	1	El texto de la firma visible se mostrará con fuente Helvética.
	2	El texto de la firma visible se mostrará con fuente Times Roman.
	3	El texto de la firma visible se mostrará con fuente Symbol.
	4	El texto de la firma visible se mostrará con fuente ZapfDingBats.
layer2FontSize	[Entero positivo]	Tamaño de fuente del texto de la firma visible.
layer2FontStyle	0	Texto de la firma visible sin estilo. Valor por defecto.
	1	Texto de la firma visible en negrita.
	2	Texto de la firma visible en cursiva.
	4	Texto de la firma visible subrayado.
	8	Texto de la firma visible tachado.
layer2FontColor	black	El texto de la firma visible será de color negro. Este es el valor por defecto.
	white	El texto de la firma visible será de color blanco.
	gray	El texto de la firma visible será de color gris.
	lightGray	El texto de la firma visible será de color gris claro.
	darkGray	El texto de la firma visible será de color gris oscuro.

	red	El texto de la firma visible será de color rojo.
	pink	El texto de la firma visible será de color rosa.
obfuscateCertText	true	Se ofuscan los identificadores de usuario extraidos del CN o DN del certificado y mostrados en la firma visible PDF. No se ofuscan los datos de los certificados de seudónimo. Este es el valor por defecto.
	false	No se ofusca la información de los certificados.
obfuscationMask	[Texto]	Criterios de ofuscación de los identificadores de usuario en las firmas visibles PDF. Debe mostrar el siguiente patrón: caracter;longitudDigitos;posiciones;desplazamiento En este patrón: • caracter: Es el carácter que usar para ofuscar caracteres. • longitudDigitos: Número mínimo de dígitos que debe tener una cadena de texto para que se considere que debe ofuscarse. • posiciones: Listado de posiciones que indica qué caracteres deben mostrarse. El listado se expresa con una sucesión de true/false separados por comas (','), en donde true indica que el carácter debe mostrarse y false que no. • desplazamiento: Indica si se admite el desplazamiento de posiciones de la máscara para mostrar todos los caracteres indicados (true) o si esta debe respetarse (false).
signReason	[Texto]	Razón por la que se realiza la firma.
signatureProductionCity	[Texto]	Ciudad en la que se realiza la firma.
signerContact	[Texto]	Información de contacto del firmante.
signerClaimedRoles	[Texto]	Listado de roles declarados por el firmante (separados por "|")
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	[Texto 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 hacia el documento que contiene una descripción textual `de` la política de firma.
`ownerPassword`	[Texto]	Contraseña de apertura del PDF. No se soporta la firma de documentos PDF cifrados con certificados o algoritmo A256.
`headless`	`true`	No interrumpe el proceso de firma solicitando interacción del usuario.
	`false`	Muestra diálogos al usuario si requiere de su autorización o algún dato adicional para firmar. Este es el valor por defecto.
`allowSigningCertifiedPdfs`	`true`	Permite la firma de documentos PDF certificados. El resultado podría invalidar firmas anteriores del PDF.
	`false`	Produce un error al firmar documentos PDF certificados.
	Se omite	En caso de detectar que el documento PDF de entrada está certificado, se aplicará uno de los siguientes comportamientos: • Si la firma es monofásica (formato "PAdES"), se advertirá al usuario de que la firma podría invalidar firmas anteriores y se le permitirá elegir si firmar o cancelar la operación. • Si la firma es trifásica (formato "PAdESTri"), fallará la operación de firma. Tenga en cuenta que, en los clientes móviles, las firmas podrían ser trifásicas independientemente del nombre de formato indicado.
`allowCosigningUnregisteredSignatures`	`true`	Permite firmar documentos PDF con firmas previas no registradas.
	`false`	No permite firmar documentos PDF con firmas previas no registradas.
`signingCertificateV2`	`true`	Se utiliza el atributo `signingCertificateV2` en las firmas.
	`false`	Se utiliza el atributo `signingCertificateV1` en las firmas.
	Se omite	Se utiliza el atributo `signingCertificateV1` en las firmas SHA1withRSA y `signingCertificateV2` en el resto.

8.4 Configuración de firmas de factura electrónica

El formato de factura electrónica configura todas las propiedades imprescindibles para generar una firma válida de factura, como la política de firma o las referencias a los nodos, por lo que no deberán establecerse manualmente.

Este formato sólo puede utilizarse sobre facturas electrónicas y éstas sólo admiten la operación de firma. No permiten cofirmarlas ni contrafirmarlas.

La política de firma utilizada por defecto para firmar es la 3.1, aunque puede configurarse para el uso de la política 3.0 mediante los parámetros adicionales "`policyIdentifier`" y "`policyIdentifierHash`".

8.4.1 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 configurando directamente el formato XAdES en el 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.

8.4.2 Algoritmos de firma

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

- `SHA512withRSA`
- `SHA384withRSA`
- `SHA256withRSA`
- `SHA1withRSA` (No recomendado)

El algoritmo más seguro, y por lo tanto el recomendado para su uso es `SHA512withRSA`.

8.4.3 Parámetros adicionales

Aquí se listan los parámetros adicionales que acepta el formato FacturaE para la configuración de la operación y de 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.
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.
signerClaimedRoles	emisor	Declara que el firmante es el emisor de la factura. Este es el valor por defecto.
	receptor	Declara que el firmante es el receptor de la factura.
	tercero	Declara que el firmante es un tercero con respecto a la factura.
	supplier	Declara que el firmante es el emisor de la factura. Este es el valor por defecto.
	customer	Declara que el firmante es el receptor de la factura.
	third party	Declara que el firmante es un tercero con respecto a la factura.
policyIdentifier	http://www.facturae.es/politica_de_firma_formato_facturae/politica_de_firma_formato_facturae_v3_1.pdf	Identificador de la política de firma 3.1. Este es el valor por defecto.
	http://www.facturae.es/politica de firma formato facturae/politica de firma formato facturae v3_0.pdf	Identificador de la política de firma 3.0.
policyIdentifierHash	Ohixl6upD6av8N7pEvDABhEL6hM=	Huella digital para configurar la política de firma 3.1. Este es el valor por defecto.

	`xmfh8D/Ec/hHeE1IB4zPd61zHIY=`	Huella digital para configurar la política de firma 3.0.
`policyIdentifierHashAlgorithm`	`SHA1`	Indica que la huella digital indicada en el parámetro `policyIdentifierHash` se calculó mediante el algoritmo SHA1.

9 Compatibilidad con dispositivos móviles y AutoFirma

Una aplicación web puede invocar a las operaciones del Cliente @firma por medio de su JavaScript de despliegue. Para que estas operaciones puedan ejecutarse, es necesario que se haya instalado previamente en el equipo la aplicación del Cliente correspondiente a ese entorno. La aplicación de firma puede ser:

- AutoFirma.
  - En equipos de sobremesa (Windows, Linux o macOS).
- Cliente de firma Android.
  - En disposivos Android.
- Cliente de firma iOS.
  - En dispositivos iOS.

No todas estas aplicaciones funcionan de igual modo y las aplicaciones móviles no soportan todas las operaciones del Cliente. Por este motivo, un desarrollador que desee que su despliegue funcione en entornos móviles deberá tener en cuenta una serie de restricciones a la hora de intregar las funciones en la aplicación.

9.1 Requisitos de despliegue

Para que nuestro despliegue del Cliente @firma sea compatible con los clientes móviles, deben desplegarse también los servicios auxiliares de comunicación y de firma trifásica. Para ello se desplegarse los archivos:

- `afirma-signature-storage.war`
- `afirma-signature-retriever.war`
- `afirma-server-triphase-signer.war`

La descripción de estos servicios puede encontrarse en el apartado 5.3 Servicios .

Además de su despliegue y configuración, es necesario indicar al Cliente donde se encuentran los servicios desplegados:

- Para establecer la ubicación de los servicios de comunicación se utilizará el método `setServlets`. La descripción del uso de este método se realiza en el apartado 0 La función de carga del Cliente sólo se debería invocar una única vez por página web. Así, no se debe utilizar este último ejemplo si se va a llamar varias veces al método de firma.
- Configuración de los servicios auxiliares de comunicación.
- Para establecer la ubicación del servicio de firma trifásica se utilizará el parámetro adicional `serverUrl`, descrito en el apartado ANEXO II Firma trifásica.

9.1.1 Compatibilidad con Google Chrome

Google Chrome establece por seguridad varias restricciones en las llamadas a URLs externas. Estas limitaciones afectan al uso del Cliente @firma desde Chrome en los siguientes escenarios:

- Cuando se utiliza el cliente móvil de firma.
- Cuando se fuerza el uso de AutoFirma a través de los servicios auxiliares de comunicación.

Estos escenarios no engloban el comportamiento por defecto de AutoFirma en un entorno de escritorio, ya que en ese caso la comunicación se realiza a través de *sockets*.

Las restricciones que se establecen son las siguientes:

- Sólo se permite una única llamada a una URL externa, como las llamadas a la aplicación nativa, por cada interacción del usuario con la página.
    - Esto impide encadenar múltiples operaciones de firma, invocando cada una de ellas desde la función *callback* ejecutada al recibir el resultado de la operación anterior.
- No se permite la llamada a una URL externa pasados unos segudos desde la interacción del usuario.
    - Esto impide firmar datos locales de gran tamaño, ya que la subida de los datos al servidor a través de los servicios auxiliares podría requerir más tiempo del permitido antes de hacer la llamada a la aplicación.

La adaptación de los procedimientos de un trámite para la compatibilidad con Chrome en los escenarios descritos puede ser completa. Esta adaptación requeriría que:

- Si se van a firmar múltiples datos dentro de una misma página, se deba preparar esta para que el usuario pulse un botón “Firmar” por cada uno de los datos. De esta forma, habría una interacción del usuario por cada uno de los datos a firmar.
- Si se desean firmar datos de gran tamaño, estos datos deberían enviarse previamente al servidor y se debería configurar una operación de firma trifásica con un gestor de documentos (*DocumentManager*) a medida para que los cargue y postprocese las firmas como deba.

9.2 Limitaciones

Debe tenerse en cuenta que las versiones actuales de los distintos clientes móviles no implementan toda la funcionalidad del Cliente @firma, 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:

9.2.1 Limitaciones de formato

Los clientes móviles de Android e iOS son capaces de realizar firmas en todos los formatos avanzados soportados por AutoFirma:

- CAdES.
- XAdES.
- PAdES.
- FacturaE.

Sin embargo, no todos estos formatos están soportados de forma nativa y algunos deben ser construidos a través de una operación trifásica de firma.

Los formatos soportados nativamente por cada aplicación son:

- Android:
  - CAdES
  - PAdES
- iOS
  - CAdES

El resto de formatos se intentará realizar siempre de forma trifásica, incluso si en la llamada de firma no se ha indicado el formato de firma trifásica (XAdEStri, PAdEStri…). Sí será obligatorio establecer el parámetro adicional para indicar la dirección del servicio de firma trifásica (`serverUrl`).

9.2.2 Limitaciones funcionales

Los clientes de firma móvil no permiten la configuración del almacén de claves que deben utilizar ni su comportamiento:

- El cliente de firma Android siempre usará el almacén del sistema o un DNIe 3.0 por NFC (si se encuentra habilitada esta opción en la aplicación).
- El Cliente de firma iOS siempre usará el almacén de la propia aplicación.

Este comportamiento hace que ambas aplicaciones ignoren las llamadas a los siguientes métodos de configuración:

- `setKeyStore(keystore)`
- `setStickySignatory(sticky)`

También se ignorarán los parámetros adicionales para la configuración de filtros de certificados y la selección automática de certificados cuando sólo haya uno en el almacén.

Por otra parte, las aplicaciones móviles no implementan las operaciones auxiliares para la carga de ficheros y la aplicación iOS tampoco soporta la función de guardado:

- `getFileNameContentBase64(title, extension, description, filePath, successCallback, errorCallback)`

- `getMultiFileNameContentBase64(title, extension, description, filePath, successCallback, errorCallback)`

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

En el apartado 6.8 Operaciones de gestión de ficheros, se plantean casos de uso alternativos para evitar el uso de estos métodos.

Las aplicaciones móviles tampoco son compatibles con la opción de llamar al método de firma sin proporcionarle los datos a firmar para que sea la propia aplicación la que se los solicite al usuario. Los datos deben proporcionárselos siempre la aplicación.

Por último, las aplicaciones móviles no son compatibles con el método de firma de lotes ni con el de recuperación del log de la aplicación:

- `signBatch (batchB64, batchPreSignerUrl, batchPostSignerUrl, params, successCallback, errorCallback)`

- `getCurrentLog (successCallback, errorCallback)`

9.3 Notificaciones al usuario

Es obligatorio que el usuario tenga instalado el Cliente de firma Android o iOS, según corresponda, antes de realizar una operación de firma desde su dispositivo móvil. Se recomienda por ello que se advierta al usuario antes de alcanzar la operación de la firma de la necesidad de instalar esta aplicación. El *javascript* de despliegue del Cliente facilita a las aplicaciones la labor de detectar el entorno del usuario mediante las siguientes 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.

Al detectar que el usuario accede a la aplicación desde Android o iOS, la aplicación puede, por ejemplo, mostrarle al usuario el enlace para la instalación de la aplicación desde la tienda de aplicaciones correspondiente

Un ejemplo del uso de estas funciones sería:

```
// Si es Android, mostramos el mensaje de advertencia para Android
if (AutoScript.isAndroid()) {
      document.getElementById("androidWarning").style.display = "block";
}
// Si es iOS, mostramos el mensaje de advertencia para iOS
else if (AutoScript.isIOS()) {
      document.getElementById("iOSWarning").style.display = "block";
```

```
}
```

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

10 Problemas conocidos

Se han detectado una serie de situaciones problemáticas asociadas al uso del Cliente @firma y sus servicios. Un usuario o aplicación puede verse afectado por estas situaciones si obtiene los siguientes errores al utilizar el Cliente @firma:

1. No se puede acceder al almacén de claves de Firefox 49.0 y superiores
2. No se puede acceder al almacén de claves de Firefox 58
3. No se detecta la inserción/extracción del DNIe en el lector (u otra tarjeta inteligente)
4. Falla la operación de firma con DNIe o una tarjeta de la FNMT
5. No se permite la firma de PDF con ciertos certificados
6. El servicio de firma trifásica genera un error al realizar firmas XAdES en servidores JBoss
7. Las firmas con DNIe requieren que se introduzca el PIN del DNIe por cada operación de firma
8. Error al cargar el listado de certificados después del cambio en caliente del almacén por defecto
9. AutoFirma no puede comunicarse con el navegador en macOS
10. Sólo se realiza la firma del primer documento de una serie cuando se realizan las firmas desde Google Chrome
11. No se abre la aplicación de firma al realizar la firma desde Google Chrome
12. No se abre la aplicación de firma con Edge Legacy (EdgeHTML)
13. No se abre la aplicación de firma con Firefox cuando el servidor declara una política de seguridad (CSP)

A continuación, se describen los problemas asociados a estos casos de error.

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

Para el acceso al almacén de claves y certificados de Firefox 49 y superiores en Windows necesita se que se tenga instalado el entorno de ejecución redistribuible de Microsoft Visual C++ 2015. Si no consigue acceder a sus certificados y claves privadas desde AutoFirma, 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.

2. 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/

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

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

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.

4. Falla la operación de firma con DNIe o una tarjeta de la FNMT

Se conoce de cierta incompatibilidad de AutoFirma con los controladores de DNIe y las tarjetas de la FNMT. Esta incompatibilidad puede llevar a que no se pueda firmar con estas tarjetas o que sólo se pueda realizar una firma y falle el proceso cuando se intente hacer alguna más.

Para solventar este problema, AutoFirma incorpora la biblioteca JMulticard, un driver Java para DNIe y algunas tarjetas de la FNMT.

Si se produce algún error al generar firmas con DNIe o tarjetas de la FNMT, abra AutoFirma, acceda al panel de “Preferencias”, pestaña “General” y verifique que en el panel “Opciones generales” se encuentra activada la opción “Habilitar JMulticard para el uso de tarjetas de la FNMT y DNIe (requiere reiniciar AutoFirma)”.

Si sigue sin funcionar la operación de firma, es posible que AutoFirma no sea compatible con su tarjeta inteligente. En ese caso, asegúrese de que dispone de la última versión de los controladores de la tarjeta.

5. No se permite la firma de PDF con ciertos certificados

Las firmas de documentos PDF realizadas externamente (que es el método utilizado por el 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.

6. 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.

7. 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.

AutoFirma tratará de mantener abierto el canal de comunicación con el DNI mientras se realizan las operaciones de firma, pero, de no ser posible, volverá a abrirlo solicitando nuevamente el PIN al usuario.

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/

8. 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.

9. 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.

10. 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 la comunicación entre AutoFirma y el navegador se realice mediante servidor intermedio (mediante el uso de la sentencia “`AutoScript.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 problema 11 No se abre la aplicación de firma al realizar la firma desde Google Chrome.

11. 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 la comunicación entre AutoFirma y navegador se realice mediante servidor intermedio (mediante el uso de la sentencia "`AutoScript.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 10 Sólo se realiza la firma del primer documento de una serie cuando se realizan las firmas desde Google Chrome.

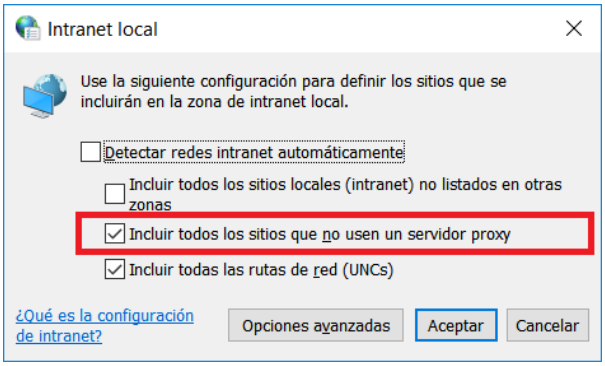
12. No se abre la aplicación de firma con Edge Legacy (EdgeHTML)

Existen determinada configuración de Microsoft Edge Legacy que impide el uso de sockets, lo que lleva a que la aplicación no se pueda comunicar con el JavaScript de despliegue y este termine dando un error pasado un tiempo de espera.

Este error se puede identificar cuando al inciar la operación se muestra la imagen de inicio de AutoFirma (*splash)* pero nunca se llega a mostrar la aplicación. Además, en la consola de Edge aparece recurrentemente un error con el mensaje:

```
SCRIPT12029: SCRIPT12029: WebSocket Error: Network Error 12029, No se pudo establecer una conexión con el servidor
```

Este problema se debe a que el navegador está aplicando ciertas restricciones a la aplicación considerando que se trata de un servidor externo. Esto se produce cuando se encuentra en Windows habilitada la opción de considerar parte de la intranet los sitios a los que no se acceda a través de un servidor proxy. Puede ver esta opción a través del panel "Opciones de Internet" del sistema operativo, pestaña "Seguridad", al pulsar en la zona "Intranet" y seguidamente en el botón "Sitios".



Es probable que esta opción se encuentre habilitada como parte de una configuración de maqueta corporativa. Si es así, NO deshabilite esta opción, ya que podría afectar negativamente a la seguridad de su equipo o el funcionamiento de aplicaciones coorporativas. Con esta opción habilitada no podrá utilizar AutoFirma con Edge Legacy. Si es posible, actualice a Edge Chromium o utilice otro navegador web.

13. No se abre la aplicación de firma con Firefox cuando el servidor declara una política de seguridad (CSP)

Mozilla Firefox no abre por defecto las URL con esquemas personalizados cuando el servidor web declara una política de seguridad. Para permitir las llamadas con protocolo “afirma” utilizadas por el Cliente @firma, será necesario agregar a la política de seguridad el esquema correspondiente. Consulte el apartado 5.4 Configuración del *Content Security Policy* para más información.

ANEXO I. Comunicación JavaScript de despliegue – Cliente @firma

El JavaScript de despliegue del Cliente @firma es el que incluye la lógica de comunicación entre el navegador Web y el cliente de firma. Algunas de las operaciones solicitadas a través del JavaScript de despliegue serán procesadas directamente por el propio JavaScript, mientras que otras sí requerirán que traslade la solicitud al Cliente de firma.

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:

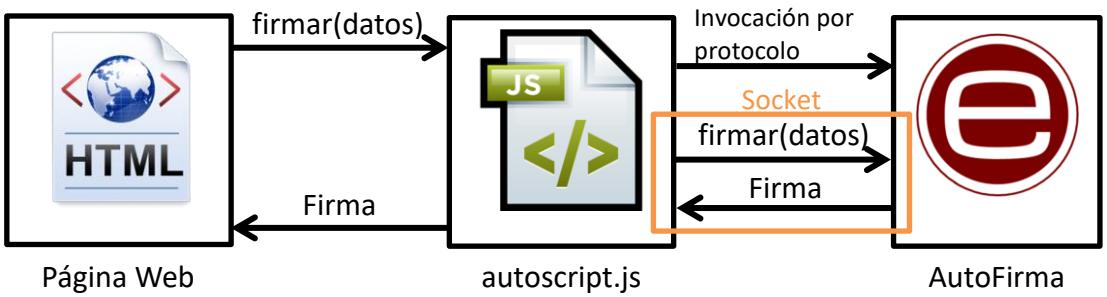
- La comunicación mediante *socket*.
  - Este sistema se utiliza cuando se ejecuta la operación en un equipo sobremesa y navegador Chrome, Firefox, Safari, Internet Explorer 11 y Edge.
- La comunicación mediante un servidor intermedio.
  - Este sistema requiere el despliegue de los dos servicios auxiliares de comunicación y puede utilizarse potencialmente en cualquier navegador. Consulte el apartado Servicios para más información.
  - Dado que este sistema ofrece un peor rendimiento, por defecto sólo se utiliza en entornos de usuario que no soportan la comunicación por sockets o en los que puede dar problemas: dispositivos móviles, Internet Explorer 10 e inferiores, Safari 10….

Por regla general, **siempre se deberían desplegar los servicios auxiliares para la comunicación por servidor intermedio**, ya que hay entornos que dependen de esos servicios, incluso si no son los más utilizados.

I.1. Comunicación por sockets

En la comunicación por *socket*, AutoFirma abre un *socket* local SSL y el JavaScript de despliegue traslada las peticiones de la aplicación a través del mismo. Las respuestas se obtienen a través del propio *socket*.

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



Primeramente, el trámite ordena la operación en cuestión por medio del JavaScript de despliegue del Cliente @firma, por ejemplo, una operación de firma. A continuación, el JavaScript detectará que el entorno del usuario soporta la comunicación por sockets y lanzará AutoFirma mediante una invocación por protocolo junto con las instrucciones para abrirlo. AutoFirma abrirá este *socket* cifrando el canal mediante un certificado SSL y el JavaScript podrá enviar la orden de firma a través del mismo.

Una vez abierta la aplicación, cualquier petición posterior se realizará a través del mismo socket, sin necesidad de relanzarla. AutoFirma se cerrará al detectar que el puerto se ha cerrado o, si no es posible detectarlo, cuando pase un tiempo sin haber recibido más peticiones desde el JavaScript.

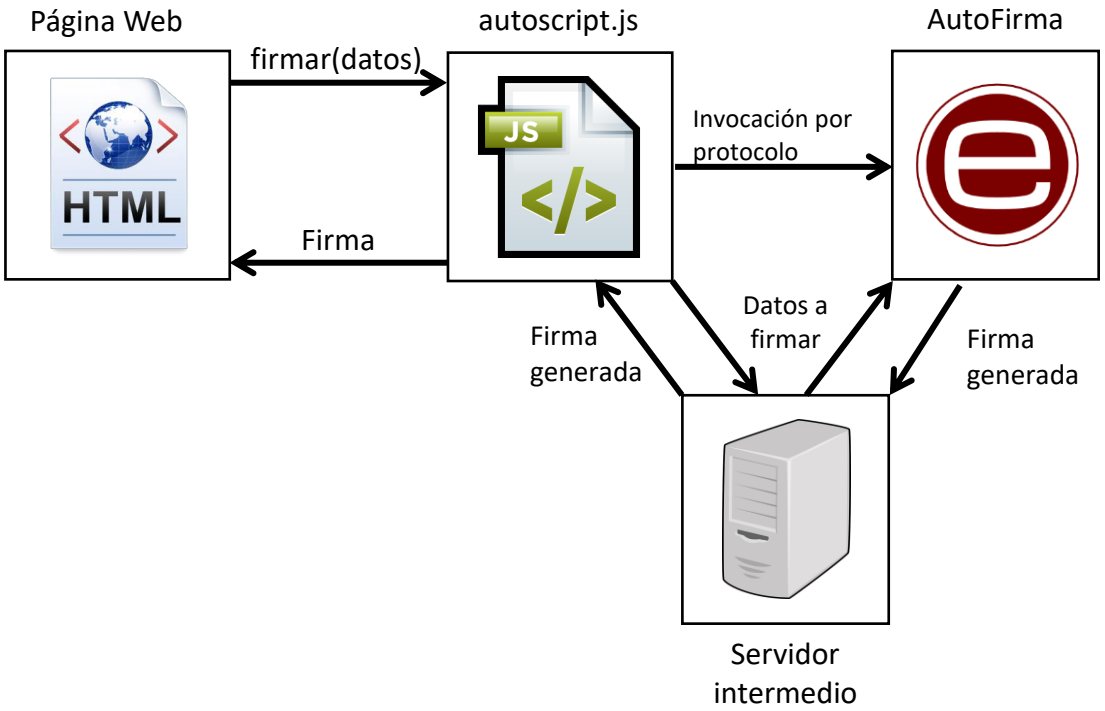
La comunicación a través del socket se cifra mediante el protocolo TLS v1.1/v1.2. Es necesario tener habilitados estos protocolos en el navegador para poder establecer la comunicación con AutoFirma.

I.2. Comunicación por servidor intermedio

Este modo de comunicación se basa en dos servicios disponibles en el mismo dominio que el trámite Web en el que se va a firmar. Cuando el JavaScript de despliegue y la aplicación de firma necesiten comunicarse, subirán la información de la operación a realizar a un servidor, de donde tendrá que descargárselo el destinatario de dichos datos.

La comunicación por servidor intermedio es la utilizada por defecto cuando se accede a la página desde un dispositivo móvil (en el que no podemos abrir un socket local) y cuando se accede desde otro entorno con un navegador no compatible con la comunicación por sockets o en el que se han encontrado problemas.

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



Cuando el JavaScript de despliegue deba enviar a la aplicación de firma la ejecución de una operación, subirá los datos que deba enviarle al servidor intermedio por medio del servicio de guardado, invocará a la aplicación de firma mediante una llamada por protocolo y ésta se descargará los datos por medio del servicio de recuperación. Una vez procesados los datos, la aplicación almacenará el resultado en el servidor intermedio y el JavaScript de despliegue descargará estos datos del servidor para obtener el resultado final.

En este modo de comunicación, el JavaScript de despliegue abre la aplicación de firma por cada operación. Después de finalizar una operación, la aplicación se cierra o queda en segundo plano.

Consulte el apartado 5.3.1 Servicios auxiliares de comunicación para saber más de estos servicios del Cliente @firma y cómo desplegarlos.

ANEXO II. Firma trifásica

El Cliente @firma 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. Compone la firma electrónica en el formato configurado.

En el comportamiento por defecto, AutoFirma realizaría todos estos pasos consecutivamente. Sin embargo, el proceso puede realizarse en tres fases separadas, en donde la primera y la tercera fase, que son las que no requieren la clave privada de firma, se realicen en un servidor remoto, mientras que la segunda seguiría realizándose en el equipo cliente. Este proceso de firma dividido en 3 fases es lo que se denomina 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; el usuario la firmaría en su sistema (Fase II) y el resultado se enviaría de vuelva a servidor; en donde se compondría la firma (Fase III) y podría postprocesarse.
- 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 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 similares, ya que no es necesario programar la lógica del formato para el dispositivo, sólo es necesaria la fase del cifrado de los datos con la clave del usuario.
  - 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 un servicio en el servidor que se encargue de realizar las fases 1 y 3.

Debe recalcarse que el procedimiento de firma trifásica es útil cuando los datos a firmar residen en servidor y la firma generada se necesita también en servidor. En estos casos, se podría indicar al servidor de donde obtener los datos a firmar y que hacer con la firma generada. Si los datos y/o la firma deben estar 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 cliente de firma.

II.1. Servicio de firma trifásica

El servicio de firma trifásica es el que realiza la primera y tercera fase del proceso de firma. Junto al Cliente @firma se distribuye el archivo WAR "`afirma-server-triphase-signer.war`" que despliega un servicio 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 y el apartado 5.3.2 Servicios de firma trifásica y firma de lotes para saber como configurar el servicio.

El servicio de firma trifásica soporta los formatos CAdES, XAdES, PAdES y FacturaE y admite las mismas opciones de configuración que las firmas monofásicas de estos formatos.

Desplegar y configurar el servicio de firma trifásica puede ser necesario para le uso del Cliente de firma móvil, ya que este delega en el servicio trifásico la generación de las firmas en los formatos que no soportan nativamente. Para saber más acerca del Cliente @firma móvil, consulte el apartado 9 Compatibilidad con dispositivos móviles y AutoFirma.

II.1.1. Configuración del gestor de documentos del servicio

La principal ventaja del proceso de firma trifásica es que no es necesario descargar a la página web los datos a firmar para luego pasárselos al método de firma, ni será necesario recoger la firma en un método *callback* para luego enviarla al servidor. Los datos pueden cargarse directamente en servidor y no es necesario que viajen al equipo del usuario y la firma puede guardarse directamente sin necesidad de que la recoja nuestra aplicación. Esta lógica de cargar los datos y guardar la firma la gestiona el servicio de firma trifásica por medio de lo que llamaremos "gestor de documentos" (*Document Manager*).

Nuestro servicio de firma trifásica tendrá configurado un gestor de documentos que le dirá cómo obtener los datos y guardar las firmas. Este gestor de documentos se configura a través de la propiedad "`document.manager`" del fichero de configuración del servicio.

El servicio de firma trifásica se distribuye con dos gestores de documento ya integrados:

- `es.gob.afirma.triphase.server.SelfishDocumentManager`
  - Es el *Document Manager* por defecto y emula el comportamiento de la firma monofásica del cliente.

  - Si no tenemos interés en la firma trifásica, pero sí queremos que nuestra aplicación sea compatible con los clientes móviles de firma, deberemos desplegar el servicio de firma trifásica con este gestor de documentos configurado. Este servicio permitirá completar la firma cuando se utilice un formato no soportado por la aplicación móvil en cuestión:
    - Cliente @firma Android:
      - XAdES y FacturaE.
    - Cliente @firma iOS:
      - XAdES, FacturaE y PAdES.
  - Recibe:
    - Los datos a firmar.
  - Devuelve:
    - La firma generada.
- `es.gob.afirma.triphase.server.FileSystemDocumentManager`
  - Permite cargar los datos a firmar desde un directorio de entrada y guarda las firmas resultantes en un directorio de salida.
  - Recibe:
    - El nombre del fichero de datos a firmar.
  - Devuelve:
    - El nombre con el que se ha guardado el fichero de firma.
  - 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`).
  - **Advertencia:** La inclusión del FileSystemDocumentManager busca servir como ejemplo de gestor de documentos. Este gestor no debería usarse salvo que se ajuste muy concretamente a sus necesidades. Si no es así, lo correcto es que implemente su propio gestor de documentos optimizado para la carga y el guardado de documentos en sus sistemas.

Salvo el caso concreto de los aplicativos móviles, la generación de firmas trifásicas mediante el gestor de documentos por defecto está desaconsejada, ya que este requiere el envío y la descarga de datos adicionales del servidor para hacer las operaciones que podría hacer AutoFirma de forma nativa. Para aprovechar las ventajas del sistema de firma trifásica, configure un gestor de documentos apropiado para su sistema.

Un desarrollador Java podría crear nuevos gestores de documentos e integrarlos en el servicio. Esto le permitiría crear procesos óptimos que accediesen a sus entornos para recoger los datos y guardar las firmas y así se evitase la descarga de los datos redujese la transferencia de datos entre el cliente de firma y los servidores del por red.

II.1.2. Desarrollo de un gestor de documentos personalizado

Solo el responsable de un entorno conoce los requisitos de acceso al mismo, el proceso adecuado para acceder y las medidas de seguridad que es necesario implementar. Es por esto que el servicio trifásico permite que los integradores desarrollen su propia clase gestora de documentos y configurar en el servicio de firma trifásica el que se utilice esta. Por ejemplo, si los documentos a firmar se almacenasen en un repositorio de contenidos remoto y las firmas se deben guardar en base de datos, podremos implementar la lógica de nuestra clase gestora de documentos para que acceda a estos entornos usando las credenciales adecuadas, recupere los documentos y almacene las firmas.

Para implementar un gestor de documentos, se deberá implementar la interfaz `es.gob.afirma.triphase.server.DocumentManager`, disponible en el módulo "`afirma-server-triphase-signer-document`" del proyecto. Si se desea, se puede importar a su proyecto mediante la siguiente referencia de Maven:

```
<dependency>
	<groupId>es.gob.afirma</groupId>
	<artifactId>afirma-server-triphase-signer-document</artifactId>
	<version>1.7.0</version>
</dependency>
```

La interfaz `DocumentManager` define los siguientes métodos que deberemos implementar:

- `byte[] getDocument(byte[] docId, X509Certificate[] certChain, Properties config) throws IOException;`
  - Método para la obtención del documento que se desea firmar.
  - Los parámetros recibidos en este método serán:
    - `id`: Identificador de los datos que se firmaron. Es el mismo valor que se debió recibir en el método getDocument.resultado de decodificar el parámetro que se proporcionó en Base 64 como parámetro de datos a firmar en el método de firma del JavaScript de despliegue.
      - Por ejemplo, si quisieramos recibir en este parámetro el identificador "foo" con el cual poder recuperar los datos, llamaríamos al método de firma del Cliente @firma indicando como parámetro de datos el Base 64 "foo" ("`Zm9v`"). En este parámetro `id` se recibirá ese Base 64.

- `certChain`: Cadena de certificación del certificado utilizado para firmar. Según el tipo de operación, la aplicación utilizada y el almacén del usuario, es posible que este valor sólo contenga el certificado de firma o incluso que el parámetro sea nulo.
- `config`: Parámetros para la configuración de firma. Este es parámetro `extraParams` proporcionado en el método de firma del Cliente @firma. Se podrían incluir aquí parámetros que no entren en conflicto con los de firma y que nos sirvan para proporcionar mayor información a la clase gestora de documentos.

- Esta función debe devolver el binario del documento a firmar.
- En caso de producirse un error al recuperar el documento, la clase gestora deberá lanzar una excepción de tipo `IOException`.

- `String storeDocument(String id, X509Certificate[] certChain, byte[] data, Properties config) throws IOException;`
  - Método para el tratamiento y guardado de la firma generada.
  - Además del guardado, en este método se podrían implementar funciones adicionales de las que se quiera liberar a su sistema. Por ejemplo, se podría implementar aquí el proceso de validación de la firma o la agregación de un sello de tiempo (la agregación de un sello de tiempo ya lleva implícita la validación). De esta forma, si fallase este proceso, el Cliente @firma nortificaría al usuario de forma inmediata.
  - Los parámetros recibidos en este método serán:
    - `id`: Identificador de los datos que se firmaron. Es el mismo valor que se debió recibir en el método `getDocument`.
    - `certChain`: Cadena de certificación del certificado utilizado para firmar. Según el tipo de operación, la aplicación utilizada y el almacén del usuario, es posible que este valor sólo contenga el certificado de firma o incluso que el parámetro sea nulo.
    - `data`: Firma electrónica generada. Son los datos que se deberán tratar y guardar.
    - `config`: Parámetros para la configuración de firma. Este es parámetro `extraParams` proporcionado en el método de firma del Cliente @firma. Se podrían incluir aquí parámetros que no entren en conflicto con los de firma y que nos sirvan para proporcionar mayor información a la clase gestora de documentos.
  - Este método debe devolver una cadena en Base 64, que será la que se reciba como parámetro de datos en el método de éxito de la aplicación que invocó al Cliente @firma. Esta cadena no tiene que ser un dato significativo si no es necesario. Por ejemplo, podría limitarse a devolver la cadena "`OK`" (codificada en base 64), únicamente para que la aplicación sepa que ha finalizado correctamente.

  - En caso de producirse un error al tratar y guardar el documento, la clase gestora deberá lanzar una excepción de tipo `IOException`.

- Adicionalmente, si necesitamos que nuestra clase utilice propiedades que se establecerán en el fichero de configuración del servicio trifásico, se puede implementar en la clase gestora un constructor que reciba un objeto `Properties` por parámetro. El servicio de firma trifásica hará uso de este constructor para inicializar la clase y le proporcionará a través de este parámetro todas las propiedades del fichero de configuración.

II.1.3. Configuración de un gestor de documentos personalizado

Para configurar una clase gestora de documentos personalizada se deberá incluir esta como parte del servicio de firma trifásica. Esto se puede hacer generando una JAR con las clases y recursos necesarios del conector e incluyéndola dentro del WAR del servicio de firma trifásica. Para esto, se debe abrir el WAR "`afirma-server-triphase-signer.war`" con una aplicación de compresión de ficheros y agregar el JAR en el subdirectorio "`\WEB-INF\lib\`". Esta versión modificada del WAR es la que se deberá desplegar en lugar del WAR por proporcionado.

Seguidamente, se deberán configuran en el fichero de configuración del servicio ("`tps_config.properties`") la propiedad "`document.manager`" con el nombre completo de nuestra clase gestora de documentos y todas aquellas propiedades adicionales que quieran tomar de este fichero. Puede consultar más información sobre este fichero en el apartado 5.3.2.1.1 Configuración del servicio de firma trifásica.

II.2. Realizar firma trifásica con el Cliente @firma

Para realizar una firma trifásica con el Cliente @firma es necesario realizar las siguientes acciones:

- **Proporcionar una referencia a los datos**
  - En lugar de los datos, proporcionaremos al método de firma una referencia para que el servicio de firma trifásica encuentre esos datos.
  - La referencia a los datos depende del gestor de documento configurado en el servidor (ver más adelante). En la configuración por defecto del servicio de firma trifásica, seguiremos proporcionando los propios datos como parámetro.
- **Establecer el formato de firma trifásica**
  - Para la firma en los formatos CAdES, PAdES, XAdES y factura electrónica se usará `CAdESTri`, `PAdESTri`, `XAdESTri` y `FacturaEtri`, respectivamente, como identificador del formato.
- **Configurar parámetro con la URL del servidor**
  - En los parámetros adicionales de la operación se deberá configurar la URL del servicio de firma trifásica. Esta información se configurará con la propiedad `serverUrl`.
  - Para saber más de los parámetros adicionales de configuración consulte el apartado 7.1 Paso de parámetros adicionales.

Como resultado, la función de éxito JavaScript devolverá la cadena Base 64 devuelta por el método de guardado de la firma del gestor de documentos.

A continuación, se muestran algunos ejemplos de solicitud de operación trifásica:

- Firma trifásica XAdES:

```
…
  var params = " format=XAdES Detached\nserverUrl=https://miweb.com/afirma-
server-triphase-signer/SignatureService ";

  AutoScript.sign (refDataB64,
                   "SHA512withRSA",
                   "XAdEStri",
                   params,
                   successCalback,
                   errorCallback);
…
```

- Cofirma trifásica CAdES:

```
…
var params = "serverUrl=https://miweb.com/afirma-server-triphase-signer
/SignatureService";

AutoScript.cosign (refSignB64,
                   "SHA512withRSA",
                   "CAdEStri",
                   params,
                   successCalback,
                   errorCallback);
…
```

II.3. Configuración del gestor de documentos "FileSystemDocumentManager"

II.3.1. Parámetros de uso y descripción del funcionamiento

El gestor de documentos `FileSystemDocumentManager` permite gestionar la firma de ficheros almacenados en servidor, lo que sirve para representar de forma básica las capacidades de un gestor de documentos. Este gestor utiliza como identificadores de documento el nombre del fichero en el que se almacena, tanto en la entrada (fichero a firmar) como en la salida (fichero de firma).

Los nombres de fichero que debe recibir se indican codificados en Base64 para evitar problemas de codificación. Por ejemplo:

- El documento "`documento.pdf`" se indicaría con la cadena "`ZG9jdW1lbnRvLnBkZg==`".
- El documento "`firma.xsig`" se indicaría con la cadena "`ZmlybWEueHNpZw==`".

El servicio devuelve el nombre del fichero también codificado en Base64. La ruta del fichero de firma no se proporciona ya que esta es siempre la configurada en la propiedad "`outdir`" del servicio.

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 indicar nunca un nombre de ficheros que contuviese el carácter "dos puntos" (":").

Por ejemplo, si quisiéramos realizar una firma trifásica con el Cliente @firma y nuestro servicio de firma trifásica tuviese configurado el gestor de documentos "FileSystemDocumentManager" podríamos firmar así:

```
…
  var params = "expPolicy=FirmaAGE\nserverUrl=https://miweb.com/afirma-server-
triphase-signer/SignatureService ";

  // Queremos firmar el documento "12345678.pdf" del directorio de entrada
  AutoScript.sign (AutoScript.getBase64FromText("12345678.pdf"),
                   "SHA512withRSA",
                   "PAdESTri",
                   params,
                   successCallback,
                   errorCallback);
…
function successCallback(filenameB64, certB64) {
      // filenameB64 es el nombre del fichero de firma codificado en Base 64
}
```

Advertencia: Este gestor de documentos se incluye a modo ilustrativo. Si se desea utilizar en un entorno productivo, es necesario tener en cuenta algunos requisitos de seguridad. Por ejemplo, los documentos sólo deberían almacenarse en el directorio de entrada del servidor en el que se vaya a iniciar el proceso de firma y debería eliminarse del mismo una vez finalizada. De otra forma, este documento podría ser recuperado por una aplicación malintencionada mediante llamadas al servicio trifásico. Las firmas en el directorio de salida también se deberían retirar una vez generadas.

II.3.2. 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. En este entorno, si se especifica una ruta local, puede ocurrir 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.