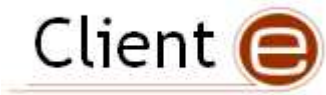
Manual de Migración del



Índice

1 Introducción

El Cliente de Firma es una herramienta de Firma Electrónica que funciona en forma de Applet de Java integrado en una página Web mediante JavaScript.

El Cliente hace uso de los certificados digitales X.509 y de las claves privadas asociadas a los mismos que estén instalados en el repositorio o almacén de claves y certificados (*keystore*) del navegador web (*Internet Explorer, Mozilla, Firefox*) o el sistema operativo así como de los que estén en dispositivos (tarjetas inteligentes, dispositivos *USB*) configurados en el mismo (el caso de los DNI-e).

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

El Cliente de Firma contiene las interfaces y componentes web necesarios para la realización de los siguientes procesos (además de otros auxiliares como cálculos de hash, lectura de ficheros, etc…):

- Firma de formularios Web.
- Firma de datos y ficheros.
- Multifirmamasiva de datos y ficheros.
- Cofirma (CoSignature)→Multifirma al mismo nivel.
- Contrafirma (CounterSignature)→Multifirma en cascada.

Como complemento al cliente de firma, se encuentra un cliente de cifrado que nos permite realizar las funciones de encriptación y desencriptación de datos atendiendo a diferentes algoritmos y configuraciones. Además permite la generación de sobres digitales.

2 Objeto

El presente documento describe el procedimiento de la migración de las aplicaciones Web que integren el Cliente @firma para incorporar su última versión.

Este manual expone el procedimiento de migración desde la versión 2.4 del Cliente en adelante.

3 Alcance

Este manual se ha realizado tomando como base el que se ha realizado una integración estándar de una versión del cliente @firma distinta a la última, esto es:

- Se hace uso de las bibliotecas JavaScript que se distribuyen con esa versión del Cliente.
- Se hace uso de los métodos publicados en el Applet.

Los pasos detallados en este manual sirven para adaptar los despliegues existentes del Cliente @firma a su última versión.

Aquí sólo se describe la migración de las funcionalidades del Cliente implementadas en versiones anteriores. Para la incorporación de las nuevas funcionalidades disponibles en la última versión del Cliente será necesario dirigirse al “Manual del Integrador”.

4 Migración alaversión 3.3.1del Cliente @firma

La redacción de este manual viene motivada principalmente por el cambio metodológico y de arquitectura que se ha ido realizado a lo largo de las distintas versiones del Cliente @firma, en buena parte motivadas por los cambios de las nuevas versiones de Java.

Acceda al apartado correspondiente a la versión del Cliente @firma que tenga desplegada y realice los cambios necesarios para la adaptación a la última versión.

Si va a realizar un nuevo despliegue del Cliente @firma, ignore este documento y despliegue la última versión del Cliente valiéndose de su Manual del integrador.

4.1 Migración desde el Cliente 2.4

Mientras que en la versión 2.4 del cliente y anteriores se instalaban siempre las dependencias del cliente y este debía cargarse en cada ejecución, en la nueva versión del Cliente las dependencias sólo se instalan en caso de ser necesarias y el Cliente se despliega mediante JNLP. Durante la carga del Cliente, si se detecta que el entorno puede requerir la instalación de dependencias del Cliente, se cargará un Bootloader encargado de comprobar si estas dependencias están cubiertas o es necesario instalarlas. La carga del Bootloader, las comprobaciones y la instalación de dependencias son automáticas y el integrador no tiene que hacer nada al respecto.

Cuando el sistema del usuario no es susceptible de requerir dependencias adicionales, se cargará directamente el Cliente y no el Bootloader.

Adicionalmente, la nueva versión del cliente cuenta con una nueva arquitectura que divide sus funcionalidades en 3 construcciones distintas:

- **Construcción LITE**: Soporta firmas PKCS#1, CMS/PKCS#7 y CADES, e incorpora todas las capacidades actuales del cliente (firmas, cifrados, acceso a repositorios…).
- **Construcción MEDIA**: Soporta firmas XMLdSig, XAdES, ODF y OOXML, más las funcionalidades de la construcción LITE.
- **Construcción COMPLETA**: Soporta firmas PDF, además de disponer de las funcionalidades de la construcción MEDIA.

En el momento de cargar el Cliente el integrador podrá indicar que construcción se desea, según las funcionalidades que requiera su sistema.

El nuevo Cliente @firma se despliegade forma similara las versiones anteriores. Para actualizar el Cliente @firma a la última versión y adaptar nuestra aplicación Web deberemos seguir los siguientes pasos:

1. Sustituir la totalidad de ficheros de despliegue (bibliotecas JavaScript, ficheros JNLP, archivos JAR, ficheros ZIP, ficheros de propiedades y cualquier otro ficheros distribuido con el Cliente @firma) del cliente por las de la última versión. Durante este proceso, al sustituir el fichero "*constantes.js*", deberemos asegurarnos de que las constantes del nuevo cliente tienen asignadas el mismo valor que el del cliente desplegado. Consulte el apartado Despliegue del Cliente @firma para más detalle.
2. Adaptar, si procede, los HTML que cargan el cliente según se explica en el apartadoCarga del *Applet* de firma.

3. Revisar si alguno de los métodos utilizados ha cambiado su comportamiento según se indica en el apartado Cambios en los procedimientos para asegurarnos de que no afecta a nuestra aplicación. En caso de afectarnos, proceder tal como se indica.
4. Comprobar que no hacemos uso de ninguno de los métodos eliminados del Cliente.De hacerlo, lo sustituiremos según se indica en el apartado Antes existía un solo método para configurar los destinatarios de los sobres electrónicos:

- `setRecipientsToCMS(String):` Este método definía todos los destinatarios del sobre de una sola vez. Como parámetro recibía el listado de rutas locales de los certificados de los destinatarios separadas por '\n'.

En la nueva versión, existen dos métodos adicionales para indicar los destinatarios:

- `addRecipientToCMS(String):` Permite agregar un certificado al listado de destinatarios del sobre. Recibe el certificado codificado en base 64.
- `removeRecipientToCMS(String):` Permiteeliminar un certificado al listado de destinatarios del sobre. Recibe el certificado codificado en base 64.

El método `removeRecipientToCMS` elimina cualquier certificado de la lista, ya haya sido cargado mediante `addRecipientToCMS` o `setRecipientsToCMS.` Si deseamos eliminar todos los certificados podemos utilizar `setRecipientsToCMS(null).`

5. Restricciones, o por un mecanismo alternativo si no existe alternativa.

4.1.1 Despliegue del Cliente @firma

4.1.1.1 Importación de librerías JavaScript

Las librerías JavaScript del nuevo cliente @firma han sufrido ciertos cambios desde sus versiones anteriores, por lo que deberá revisarse el modo de uso desde sus páginas Web HTML. Estos cambios se han realizado para mejorar la compatibilidad con ciertas versiones del entorno de ejecución de Java (JRE), pero sobre todo para garantizar el funcionamiento correcto con versiones futuras de JRE, navegadores Web y sistemas operativos, especialmente en las nuevas arquitecturas de 64 bits.

- El integrador ya no necesita gestionar la instalación y actualización del Cliente, dadoque estos procesos se gestionan de forma completamente automática. Debe retirar cualquier referencia en su código HTML a los siguientes métodos JavaScript o parámetros:
  - instalar()
  - desinstalar()
  - isActualizado()
  - actualizar()
  - isInstalado()
  - getDirectorioInstalacion()
  - VariableinstallDirectory en *constantes.js*
  - Variable instalador en *instalador.js*

- Los JavaScript que necesita importar en sus páginas HTML pueden igualmente haber cambiado. Asegúrese de que no importa ningún JavaScript que no exista en el directorio de despliegue o en su subdirectorio `common-js`.
  - Consulte las páginas Web de ejemplo para determinar que ficheros JavaScript es necesario importar para las distintas operaciones de criptografía y firma electrónica.

Note especialmente que se han eliminado las librerías JavaScript "*runApplet.js*" y "*time.js*".Es obligatorio eliminar las esperas explícitas a la carga del *Applet*. Esto es, las sentencias que hacen uso de los métodos de la biblioteca *"time.js"* y la variable `clienteFirmaCargado`. Por ejemplo:

```
whenTry("clienteFirmaCargado == true", "clienteFirma.setCipherAlgorithm('"
+cipherAlgorithm+"')", "No se ha podido iniciar el Applet de firma.");
```

Por regla general, el navegador Web no termina la carga de la página hasta que no se finaliza la carga del Cliente, por lo que no suele ser necesario agregar sentencias de este tipo.

4.1.2 Carga del *Applet* de firma

La nueva arquitectura del cliente elimina el sistema de módulos (*plugins*) con el que se contaba en versiones V2.x(debido a nuevas restricciones de seguridad de la JRE1.6u17 y superiores) y establece 3 construcciones distintas que incorporan diferentes funcionalidades (cada una incorporando las funcionalidades de las anteriores). Esta nueva arquitectura requiere que cada vez que se cargue el *Applet*mediante el método *cargarAppletFirma()* se indique la construcción mínima que exija nuestra aplicación para funcionar correctamente. Esto lo haremos pasándole los parámetros 'LITE', 'MEDIA' o 'COMPLETA' al método según sea la construcción que necesitemos. Si no se indica nada, se interpretará que se desea la construcción por defecto, que será la 'LITE' salvo que se indique lo contrario mediante la variable `defaultBuild` del fichero "*constantes.js*". El integrador deberá consultar el listado de funcionalidades incorporado en cada construcción del usuario para indicar cual debe utilizar.

Para cargar el *applet* de firma exigiendo que se disponga de al menos la construcción MEDIA, por ejemplo, usaríamos la sentencia:

```
<script type="text/javascript">
    cargarAppletFirma('MEDIA');
</script>
```

También podríamos establecer la variable`defaultBuild` del fichero "*constantes.js*" con el valor `MEDIA` y hacer:

```
<script type="text/javascript">
    cargarAppletFirma();
</script>
```

4.1.2.1 Localización de la llamada al método de carga del Applet

En la versión 3 del Cliente @firma se ha cambiado el modo de despliegue de los Applets para seguir las últimas recomendaciones de Oracleal respecto.

Debido a estos cambios, la llamada al método `cargarAppletFirma()` no puede ser realizada dentro de una etiqueta XML, y debe situarse dentro del cuerpo de una sección delimitada por etiquetas. La implicación práctica más directa de esta restricción es que ahora no es posible realizar la llamada de carga en la propiedad `onLoad()` de la etiqueta HTML `<body>`, siendo la opción recomendada situar esta llamada en cualquier lugar entre las etiquetas `<body>` y `</body>`.

Cualquier otra llamada al Applet (comprobar si está instalado, obtener la versión, etc.) sigue pudiéndose invocar desde `onLoad()` o cualquier otro gestor de eventos interno a una etiqueta HTML.

Esto aplica igualmente a cualquier proceso de carga del cliente desde un disparador de evento: `onClick(), onMouseOver()`…

4.1.3 Cambios en los procedimientos

4.1.3.1 Formato de firma por defecto

En las nuevas versiones del Cliente @firma el formato de firma por defecto es CAdES.

Si su herramienta no configurase el formato de firma explícitamente mediante el método `setSignatureFormat(String)`, deberá introducir esta sentencia indicando el formato que desee utilizar o modificar el formato de firma por defecto por medio de la variable `signatureFormat`, del fichero de configuración "*constantes.js".*

4.1.3.2 Cadena de certificación en firmas XAdES

En las nuevas versiones del Cliente @firma se inserta la cadena de certificación completa en las firmas XAdES generadas. Este es el comportamiento recomendado para firmas XAdES, pero impide la correcta validación de las firmas con versiones de la Plataforma @firma anteriores a la 5.5. Para desactivar este comportamiento es necesario establecer el parámetro extra "`includeOnlySignningCertificate`" al valor "`true`". Podemos hacer esto mediante la siguiente llamada:

- `clienteFirma.addExtraParam("includeOnlySignningCertificate", "true");`

4.1.3.3 Ensobrado de datos

En la versión 2.4 y anteriores del cliente, por un error en la implementación, se establecía el texto plano que se deseaba ensobrar mediante el método `setData(String)`. En la nueva versión del cliente, tal como se indicaba en la especificación del método, el texto que se le debe pasar a este método debe estar codificado en base 64. Podemos realizar el paso intermedio de pasar de texto plano a texto en base 64 mediante el método: `getBase64FromText(String, String)`.

Así obtenemos que:

- **Versión 2.4**: `clienteFirma.setData("texto");`

- **Ahora**: `clienteFirma.setData(clienteFirma.getBase64FromText("texto", "utf-8"));`

4.1.3.4 Cifrado de datos

En versiones anteriores del Cliente, el comportamiento de los métodos de entrada y salida de las funciones de cifrado tenían un comportamiento errático. Por ejemplo, el método `setPlainData(String)` para indicar los datos que se desean cifrar recibía el texto en claro para cifrar, mientras que `getPlainData()` devolvía este texto en base 64.

Ahora todas las entradas y salidas de datos realizadas con los siguientes métodos se realizarán en base 64:

- `setPlainData(String)`
- `setCipherData(String)`
- `getPlainData()`
- `getCipherData()`

Si actualmente invoca a alguno de estos métodos pasándole o recibiendo los datos en texto claro utilice los métodos `getBase64FromText(String, String)` y `getTextFromBase64(String, String)` del cliente para realizar las transformaciones oportunas.

4.1.3.5 Devolución de nulos

Las anteriores versiones del cliente disponían de métodos que debían devolver una cadena de texto y, en caso de error o no disponer de datos para su devolución, devolvían cadena vacía. Esta práctica, que si bien evitaba comprobar que el valor de retorno fuese nulo, llevaban a no poder distinguir cuando la operación había finalizado correctamente o si se habían devuelto datos significativos. La nueva versión del cliente, devuelve nulo en estos métodos en los que puede malinterpretarse el resultado si se devolviese cadena vacía. El comportamiento explicado se refleja en el JavaDoc de la nueva versión del cliente y los métodos afectados son:

- getCipherData()
- getPlainData()
- getData()
- getBase64Data()
- getPassword()
- getSignatureBase64Encoded()
- getSignatureText()

En caso de que utilizar alguno de estos métodos en nuestra aplicación, deberemos consultar que el resultado no sea nulo antes de utilizar el valor devuelto. Por ejemplo:

```
varplainText = clienteFirma.getPlainData();
if(plainText == null) {
     alert("No se ha podido recuperar el texto plano");
```

```
} else {
      alert(plainText);
}
```

4.1.3.6 Configuración del fichero de entrada

Existen una serie de métodos de operación que especifican por parámetro el fichero que se desea procesar, en lugar de tomar el fichero configurado mediante setFileuri(String) o setFileuriBase64Encoded(String). Adicionalmente, estos métodos modificaban la configuración del Cliente de tal forma que los ficheros especificados quedaban establecidos como ficheros de entrada para el resto de operaciones. Los métodos en cuestión son:

- getFileBase64Encoded(String strUri, booleanshowProgress)
- cipherFile(StringstrUri)
- decipherFile(StringstrUri)
- signAndPackFile(Stringuri)

En la nueva versión del Cliente @firma, los métodos mencionados no alteran la configuración del fichero de entrada establecido en el Cliente.

Por ejemplo, dado el siguiente código:

```
…
clienteFirma.setFileuri(“foo.txt”);
clienteFirma.getFileBase64Encoded(“bar.txt”, false);
clienteFirma.sign();
…
```

El nuevo Cliente @firma firmaría el fichero “foo.txt”, mientras que las versiones anteriores firmarían “bar.txt”.

4.1.3.7 Configuración de los destinatarios de los sobres electrónicos

Antes existía un solo método para configurar los destinatarios de los sobres electrónicos:

- `setRecipientsToCMS(String)`: Este método definía todos los destinatarios del sobre de una sola vez. Como parámetro recibía el listado de rutas locales de los certificados de los destinatarios separadas por ‘\n’.

En la nueva versión, existen dos métodos adicionales para indicar los destinatarios:

- `addRecipientToCMS(String)`: Permite agregar un certificado al listado de destinatarios del sobre. Recibe el certificado codificado en base 64.
- `removeRecipientToCMS(String):` Permiteeliminar un certificado al listado de destinatarios del sobre. Recibe el certificado codificado en base 64.

El método `removeRecipientToCMS` elimina cualquier certificado de la lista, ya haya sido cargado mediante `addRecipientToCMS` o `setRecipientsToCMS`. Si deseamos eliminar todos los certificados podemos utilizar `setRecipientsToCMS(null)`.

4.1.4 Restricciones

4.1.4.1 Funciones y métodos eliminados en el Applet Java

public void signHTML(java.io.InputStream is)

JavaScript no soporta el tipo de datos Java `InputStream`, por lo que su uso desde el cliente es imposible, y exponer la función puede llevar a equívocos o causar un uso inapropiado.

La firma realizada por este método es una firma simple, con la configuración establecida, sobre los datos extraídos del flujo de entrada. Podemos emular su comportamiento siguiendo los siguientes pasos:

1. Leyendo los datos del flujo de entrada en cuestión desde la aplicación que utiliza el cliente.
2. Convirtiendo los datos leídos a Base64.
3. Estableciéndolos como entrada del cliente con el método `setData(String)`.
4. Ejecutando la operación de firma mediante el método `sign()` del cliente.

public byte[] getSignature()

JavaScript no soporta el tipo de datos de Java `byte[]`, por lo que su uso es imposible, y exponer la función puede llevar a equívocos o causar un uso inapropiado.

Para recuperar la información de firma puede utilizarse:

- `getSignatureText()` para las firmas XML. Obtiene la cadena de texto que representa el XML de firma. Puede obtenerse el mismo resultado que con el método getSignature() utilizando el método getBytes() sobre su salida.
- `getSignatureBase64Encoded()` para cualquier tipo de firma. Devuelve la firma en forma de cadena en base 64. Puede obtenerse el mismo resultado que con el método getSignature() decodificando la cadena en base 64 obtenida.

public String Firma(String datos)

Se elimina el método deprecado `Firma(String)`. Este método permitía la compatibilidad del Cliente con la Plataforma @firma versión 4.

Este método recibía una cadena en base 64 compuesta por un código de operación, el valor de transacción y hasta un par de parámetros, todos concatenados por el separador almohadilla ('#'):

OP#TRANS#PARAM1#PARAM2

Los valores permitidos para el parámetro de operación (OP) eran:

- `0`: Operación de Firma. Para esta operación PARAM1 es el hash de los datos que deseamos firmar y se ignora PARAM2.
- `1`: Operación de Cofirma. Para esta operación PARAM1 es el hash de los datos que deseamos cofirmar y PARAM2 es la firma electrónica original.

- 2: Operación de Contrafirma de nodos. Para esta operación PARAM1 es la firma que deseamos contrafirmar y PARAM2 es el listado de índices tal como los recibe el método `setSignersToCounterSign(String)`.

El resultado de la operación es la cadena:

cert=CERT;enc=SIGN

En esta cadena CERT es el certificado en base 64 utilizado para la firma y SIGN es el resultado de la operación en base 64.

En cada operación se declaraba un atributo firmado adicional con resto a los que aparecen por regla general en las firmas CMS. Este sería el atributo "`2.5.4.45`" con el valor "TRANS".

Para imitar este comportamiento con el Cliente, se deberían realizar las siguientes acciones:

- Descodificar el Base64 del parámetro que recibía el método y obtener cada uno de sus componentes (OP, TRANS, PARAM1 y PARAM2).
- Configurar el formato de firma CMS, el algoritmo de firma SHA1withRSA y el modo de firma explícito, mediante las sentencias:
  - `clienteFirma.setSignatureFormat("CMS");`
  - `clienteFirma.setSignatureAlgorithm("SHA1withRSA");`
  - `clienteFirma.setSignatureMode("explicit);`
- Configurar el número de transacción como atributo firmado adicional:
  - `clienteFirma.addSignedAttribute("2.2.4.45", TRANS);`
- Configurar los datos de entrada, hash, firma y firmantes según la operación inidicada (OP) y ejecutar la operación:

```
switch(OP) {
// Firma
case 0:
     // Establecemos el hash en base 64 para la firma
     clienteFirma.setHash(PARAM1);
     clienteFirma.sign();
     break;

// Cofirma (Firma en paralelo)
case 1:
     // Establecemos el hash en base 64 para la firma
     clienteFirma.setHash(PARAM1);
     //Establecemos la firma
     clienteFirma.setElectronicSignature(PARAM2);
     //Cofirmamos
     clienteFirma.coSign();
     break;

// Contrafirma (Firma en cascada)
case 2:
     //Establecemos la firma
     clienteFirma.setElectronicSignature(PARAM1);
     //Establecemos los firmantes que se desean contrafirmar
     clienteFirma.setSignersToCounterSign(PARAM2);
```

```
            // Contrafirmamos
            clienteFirma.counterSignIndexes();
            break;
        }
```

- Recuperar el resultado de la operación y el certificado utilizado para, si se desea, componer la cadena de salida.
  - `clienteFirma.getSignCertificateBase64Encoded();`
  - `clienteFirma.getSignatureBase64Encoded();`

public String getCMSData()

Se elimina el método en favor del equivalente `getB64Data()`.

4.1.4.2 Algoritmos de cifrado eliminados

Se han eliminado los siguientes algoritmos de cifrado, por considerarse obsoletos o en desuso:

- CAST5
- IDEA
- Twofish
- Serpent

Aun cuando haya algoritmos de cifrado que se mantengan desde la versión anterior del cliente, es posible que haya cambiado su configuración por defecto. Por regla general, siempre debería descifrarse con la misma versión del cliente con la que se cifró.

4.2 Migración desde el Cliente 3.0.2/3.0.3

Mientras que en las versión 3.0.2/3.0.3 del Cliente y anteriores se instalaban siempre las dependencias del cliente y este debía cargarse en cada ejecución, en la nueva versión del Cliente las dependencias sólo se instalan en caso de ser necesarias y el Cliente se despliega mediante JNLP, por lo que puede quedarse almacenado en la caché de Java. Durante la carga del Cliente, si se detecta que el entorno puede requerir la instalación de dependencias del Cliente, se cargará un Bootloader encargado de comprobar si estas dependencias están cubiertas o es necesario instalarlas. La carga del Bootloader, las comprobaciones y la instalación de dependencias son automáticas y el integrador no tiene que hacer nada al respecto.

Cuando el sistema del usuario no es susceptible de requerir dependencias adicionales, se cargará directamente el Cliente y no el Bootloader.

El nuevo Cliente @firma se despliega de forma similar a las versiones anteriores. Para actualizar el Cliente @firma a la última versión y adaptar nuestra aplicación Web deberemos seguir los siguientes pasos:

1. Sustituir la totalidad de ficheros de despliegue (bibliotecas JavaScript, ficheros JNLP, archivos JAR, ficheros ZIP, ficheros de propiedades y cualquier otro ficheros distribuido con el Cliente @firma) del cliente por las de la última versión. Durante este proceso, al sustituir el fichero "*constantes.js*", deberemos asegurarnos de que las constantes del nuevo cliente tienen asignadas el mismo valor que el del cliente desplegado. Consulte el apartado Despliegue del Cliente @firmapara más detalle.
2. Revisar si alguno de los métodos utilizados ha cambiado su comportamiento según se indica en el apartado Cambios en los procedimientos para asegurarnos de que no afecta a nuestra aplicación. En caso de afectarnos, proceder tal como se indica.
3. Comprobar que no hacemos uso de ninguno de los métodos eliminados del Cliente. De hacerlo, lo sustituiremos según se indica en el apartado Restricciones, o por un mecanismo alternativo si no existe alternativa.

4.2.1 Despliegue del Cliente @firma

4.2.1.1 Importación de librerías JavaScript

Las librerías JavaScript del nuevo cliente @firma han sufrido ciertos cambios desde sus versiones anteriores, por lo que deberá revisarse el modo de uso desde sus páginas Web HTML. Estos cambios se han realizado para mejorar la compatibilidad con ciertas versiones del entorno de ejecución de Java (JRE), pero sobre todo para garantizar el funcionamiento correcto con versiones futuras de JRE, navegadores Web y sistemas operativos, especialmente en las nuevas arquitecturas de 64 bits.

- El integrador ya no necesita gestionar la instalación y actualización del Cliente, dado que estos procesos se gestionan de forma completamente automática. Debe retirar cualquier referencia en su código HTML a los siguientes métodos JavaScript o parámetros:
  - instalar()

- desinstalar()
- isActualizado()
- actualizar()
- isInstalado()
- getDirectorioInstalacion()
- Variable installDirectory en *constantes.js*
- Variable instalador en *instalador.js*

- Los JavaScript que necesita importar en sus páginas HTML pueden igualmente haber cambiado. Asegúrese de que no importa ningún JavaScript que no exista en el directorio de despliegue o en su subdirectorio `common-js`.
  - Consulte las páginas Web de ejemplo para determinar que ficheros JavaScript es necesario importar para las distintas operaciones de criptografía y firma electrónica.

Note especialmente que se han eliminado las librerías JavaScript "*runApplet.js*" y "*time.js*".Es obligatorio eliminar las esperas explícitas a la carga del *Applet*. Esto es, las sentencias que hacen uso de los métodos de la biblioteca *"time.js"* y la variable `clienteFirmaCargado`. Por ejemplo:

```
whenTry("clienteFirmaCargado == true", "clienteFirma.setCipherAlgorithm('"
+cipherAlgorithm+"')", "No se ha podido iniciar el Applet de firma.");
```

Por regla general, el navegador Web no termina la carga de la página hasta que no se finaliza la carga del Cliente, por lo que no suele ser necesario agregar sentencias de este tipo.

4.2.2 Cambios en los procedimientos

4.2.2.1 Confirmaciones de acceso a disco

A partir de la versión 3.3, el Cliente @firma pide confirmación al usuario mediante un diálogo modal por cada acceso a disco que se realice sin que este lo haya indicado explícitamente. Este comportamiento no debe afectar a las aplicaciones desplegadas salvo en caso particulares en el uso del modo de firma masiva programática.

Si su aplicación utiliza el modo de firma masiva programática para la firma masiva de ficheros, asegúrese de utilizar el método `massiveSignatureSign(String)` para la ejecución de las firmas.

Si su aplicación utiliza el modo de firma masiva programática y almacena las firmas resultantes en disco, evalúe el uso del mecanismo de firma de directorios o el usuario se verá obligado a aceptar el guardado individual de cada una de las firmas resultantes.

4.2.2.2 Formato de firma por defecto

En las nuevas versiones del Cliente @firma el formato de firma por defecto es CAdES.

Si su herramienta no configurase el formato de firma explícitamente mediante el método `setSignatureFormat(String)`, deberá introducir esta sentencia indicando el formato que

desee utilizar o modificar el formato de firma por defecto por medio de la variable `signatureFormat`, del fichero de configuración "*constantes.js".*

4.2.2.3 Cadena de certificación en firmas XAdES

En las nuevas versiones del Cliente @firma se inserta la cadena de certificación completa en las firmas XAdES generadas. Este es el comportamiento recomendado para firmas XAdES, pero impide la correcta validación de las firmas con versiones de la Plataforma @firma anteriores a la 5.5. Para desactivar este comportamiento es necesario establecer el parámetro extra "`includeOnlySignningCertificate`" al valor "`true`". Podemos hacer esto mediante la siguiente llamada:

- `clienteFirma.addExtraParam("includeOnlySignningCertificate", "true");`

4.2.2.4 Configurar política de firma

La última versión del Cliente @firma amplia las opciones para la configuración de la política de firma de las firmas avanzadas generadas.

En versiones anteriores el método `setPolicy(String, String, String)`, recibía el identificador, la descripción y el cualificador de la política de firma que se desease establecer. En su última versión, el Cliente @firma agrega un cuarto parámetro que es la huella digital (hash) SHA1 de la política de firma. Este hash se debe introducir en base 64.

Ahora el método `setPolicy` tiene la forma:

```
voidsetPolicy(String identifier,String description,String qualifier, String hashB64);
```

Un ejemplo de establecimiento de política es:

```
clienteFirma.setPolicy("urn:oid:2.16.724.1.3.1.1.2.1.8",“Politica de la AGE”,“http://administracionelectronica.gob.es/es/ctt/politicafirma/politica_firma_AGE_v1_8.pdf”,“7SxX3erFuH31TvAw9LZ70N7p1vA=”);
```

Así mismo, ahora también es posible indicar como identificador de la política una URL, URN u OID.

4.2.2.5 Cifrado de datos

En versiones anteriores del Cliente, el comportamiento de los métodos de entrada y salida de las funciones de cifrado tenían un comportamiento errático. Por ejemplo, el método `setPlainData(String)` para indicar los datos que se desean cifrar recibía el texto en claro para cifrar, mientras que `getPlainData()` devolvía este texto en base 64.

Ahora todas las entradas y salidas de datos realizadas con los siguientes métodos se realizarán en base 64:

- `setPlainData(String)`

- `setCipherData(String)`
- `getPlainData()`
- `getCipherData()`

Si actualmente invoca a alguno de estos métodos pasándole o recibiendo los datos en texto claro utilice los métodos `getBase64FromText(String, String)` y `getTextFromBase64(String, String)` del cliente para realizar las transformaciones oportunas.

4.2.2.6 Configuración del fichero de entrada

Existen una serie de métodos de operación que especifican por parámetro el fichero que se desea procesar, en lugar de tomar el fichero configurado mediante setFileuri(String) o setFileuriBase64Encoded(String). Adicionalmente, estos métodos modificaban la configuración del Cliente de tal forma que los ficheros especificados quedaban establecidos como ficheros de entrada para el resto de operaciones. Los métodos en cuestión son:

- getFileBase64Encoded(String strUri, booleanshowProgress)
- cipherFile(StringstrUri)
- decipherFile(StringstrUri)
- signAndPackFile(Stringuri)

En la nueva versión del Cliente @firma, los métodos mencionados no alteran la configuración del fichero de entrada establecido en el Cliente.

Por ejemplo, dado el siguiente código:

```
…
clienteFirma.setFileuri(“foo.txt”);
clienteFirma.getFileBase64Encoded(“bar.txt”, false);
clienteFirma.sign();
…
```

El nuevo Cliente @firma firmaría el fichero “foo.txt”, mientras que las versiones anteriores firmarían “bar.txt”.

4.2.2.7 Configuración de los destinatarios de los sobres electrónicos

Antes existía un solo método para configurar los destinatarios de los sobres electrónicos:

- `setRecipientsToCMS(String)`: Este método definía todos los destinatarios del sobre de una sola vez. Como parámetro recibía el listado de rutas locales de los certificados de los destinatarios separadas por ‘\n’.

En la nueva versión, existen dos métodos adicionales para indicar los destinatarios:

- `addRecipientToCMS(String)`: Permite agregar un certificado al listado de destinatarios del sobre. Recibe el certificado codificado en base 64.
- `removeRecipientToCMS(String):` Permiteeliminar un certificado al listado de destinatarios del sobre. Recibe el certificado codificado en base 64.

El método `removeRecipientToCMS` elimina cualquier certificado de la lista, ya haya sido cargado mediante `addRecipientToCMS` o `setRecipientsToCMS`. Si deseamos eliminar todos los certificados podemos utilizar `setRecipientsToCMS(null)`.

4.2.2.8 Codificación de textos

Los métodos `getBase64FromText(String)` y `getTextFromBase64(String)` se sustituyen por una versión que permite seleccionar la codificación de texto a utilizar. La declaración de los nuevos métodos es:

- `String getBase64FromText(String plainText, String charsetName);`
- `String getTextFromBase64(String b64, String charsetName);`

Si no se establece el parámetro `charsetName` se tomará la codificación por defecto del sistema. En caso de no saber con certeza la codificación del texto, se deberá indicar `null`. En el caso de tratarse un XML debe indicarse siempre `null` para que se autodetecte la codificación del XML.

4.2.3 Restricciones

public String Firma(String datos)

Se elimina el método deprecado `Firma(String)`. Este método permitía la compatibilidad del Cliente con la Plataforma @firma versión 4.

Este método recibía una cadena en base 64 compuesta por un código de operación, el valor de transacción y hasta un par de parámetros, todos concatenados por el separador almohadilla ('#'):

OP#TRANS#PARAM1#PARAM2

Los valores permitidos para el parámetro de operación (OP) eran:

- `0`: Operación de Firma. Para esta operación PARAM1 es el hash de los datos que deseamos firmar y se ignora PARAM2.
- `1`: Operación de Cofirma. Para esta operación PARAM1 es el hash de los datos que deseamos cofirmar y PARAM2 es la firma electrónica original.
- `2`: Operación de Contrafirma de nodos. Para esta operación PARAM1 es la firma que deseamos contrafirmar y PARAM2 es el listado de índices tal como los recibe el método `setSignersToCounterSign(String)`.

El resultado de la operación es la cadena:

cert=CERT;enc=SIGN

En esta cadena CERT es el certificado en base 64 utilizado para la firma y SIGN es el resultado de la operación en base 64.

En cada operación se declaraba un atributo firmado adicional con resto a los que aparecen por regla general en las firmas CMS. Este sería el atributo "`2.5.4.45`" con el valor "TRANS".

Para imitar este comportamiento con el Cliente, se deberían realizar las siguientes acciones:

- Descodificar el Base64 del parámetro que recibía el método y obtener cada uno de sus componentes (OP, TRANS, PARAM1 y PARAM2).

- Configurar el formato de firma CMS, el algoritmo de firma SHA1withRSA y el modo de firma explícito, mediante las sentencias:
  - `clienteFirma.setSignatureFormat("CMS");`
  - `clienteFirma.setSignatureAlgorithm("SHA1withRSA");`
  - `clienteFirma.setSignatureMode("explicit);`
- Configurar el número de transacción como atributo firmado adicional:
  - `clienteFirma.addSignedAttribute("2.2.4.45", TRANS);`
- Configurar los datos de entrada, hash, firma y firmantes según la operación inidicada (OP) y ejecutar la operación:

```
switch(OP) {
// Firma
case 0:
     // Establecemos el hash en base 64 para la firma
     clienteFirma.setHash(PARAM1);
     clienteFirma.sign();
     break;

// Cofirma (Firma en paralelo)
case 1:
     // Establecemos el hash en base 64 para la firma
     clienteFirma.setHash(PARAM1);
     //Establecemos la firma
     clienteFirma.setElectronicSignature(PARAM2);
     //Cofirmamos
     clienteFirma.coSign();
     break;

// Contrafirma (Firma en cascada)
case 2:
     //Establecemos la firma
     clienteFirma.setElectronicSignature(PARAM1);
     //Establecemos los firmantes que se desean contrafirmar
     clienteFirma.setSignersToCounterSign(PARAM2);
     // Contrafirmamos
     clienteFirma.counterSignIndexes();
     break;
}
```

- Recuperar el resultado de la operación y el certificado utilizado para, si se desea, componer la cadena de salida.
  - `clienteFirma.getSignCertificateBase64Encoded();`
  - `clienteFirma.getSignatureBase64Encoded();`

public String getCMSData()

Se elimina el método en favor del equivalente `getB64Data()`.

4.3 Migración desde el Cliente 3.1/3.1.1

Mientras que en estas versiones existía la posibilidad de seleccionar el modo de despliegue del Cliente: tradicional y JNLP. En la nueva versión del Cliente el despliegue siempre se realiza mediante JNLP y las dependencias sólo se instalan en caso de ser necesarias. Durante la carga del Cliente, si se detecta que el entorno puede requerir la instalación de dependencias del Cliente, se cargará un Bootloader encargado de comprobar si estas dependencias están cubiertas o es necesario instalarlas. La carga del Bootloader, las comprobaciones y la instalación de dependencias son automáticas y el integrador no tiene que hacer nada al respecto.

Cuando el sistema del usuario no es susceptible de requerir dependencias adicionales, se cargará directamente el Cliente y no el Bootloader. En la nueva versión del Cliente no existe el appletArchDetector que existía en la versión 3.1.

Para actualizar el Cliente @firma a la última versión y adaptar nuestra aplicación Web deberemos seguir los siguientes pasos:

1. Sustituir la totalidad de ficheros de despliegue (bibliotecas JavaScript, ficheros JNLP, archivos JAR, ficheros ZIP, ficheros de propiedades y cualquier otro ficheros distribuido con el Cliente @firma) del cliente por las de la última versión. Durante este proceso, al sustituir el fichero "*constantes.js*", deberemos asegurarnos de que las constantes del nuevo cliente tienen asignadas el mismo valor que el del cliente desplegado. Consulte el apartado Despliegue del Cliente @firmapara más detalle.
2. Revisar si alguno de los métodos utilizados ha cambiado su comportamiento según se indica en el apartado Cambios en los procedimientospara asegurarnos de que no afecta a nuestra aplicación. En caso de afectarnos, proceder tal como se indica.
3. Comprobar que no hacemos uso de ninguno de los métodos eliminados del Cliente. De hacerlo, lo sustituiremos según se indica en el apartadoRestricciones, o por un mecanismo alternativo si no existe alternativa.

4.3.1 Despliegue del Cliente @firma

4.3.1.1 Importación de librerías JavaScript

Las librerías JavaScript del nuevo cliente @firma han sufrido ciertos cambios desde sus versiones anteriores, por lo que deberá revisarse el modo de uso desde sus páginas Web HTML. Estos cambios se han realizado para mejorar la compatibilidad con ciertas versiones del entorno de ejecución de Java (JRE), pero sobre todo para garantizar el funcionamiento correcto con versiones futuras de JRE, navegadores Web y sistemas operativos, especialmente en las nuevas arquitecturas de 64 bits.

- El integrador ya no necesita gestionar la instalación y actualización del Cliente, dado que estos procesos se gestionan de forma completamente automática. Debe retirar cualquier referencia en su código HTML a los siguientes métodos JavaScript o parámetros:
  - instalar()

- desinstalar()
- isActualizado()
- actualizar()
- isInstalado()
- getDirectorioInstalacion()
- Variable installDirectory en *constantes.js*
- Variable instalador en *instalador.js*

- Los JavaScript que necesita importar en sus páginas HTML pueden igualmente haber cambiado. Asegúrese de que no importa ningún JavaScript que no exista en el directorio de despliegue o en su subdirectorio `common-js`.
  - Consulte las páginas Web de ejemplo para determinar que ficheros JavaScript es necesario importar para las distintas operaciones de criptografía y firma electrónica.

Note especialmente que se han eliminado las librerías JavaScript "*runApplet.js*" y "*time.js*".Es obligatorio eliminar las esperas explícitas a la carga del *Applet*. Esto es, las sentencias que hacen uso de los métodos de la biblioteca *"time.js"* y la variable `clienteFirmaCargado`. Por ejemplo:

```
whenTry("clienteFirmaCargado == true", "clienteFirma.setCipherAlgorithm('"
+cipherAlgorithm+"')", "No se ha podido iniciar el Applet de firma.");
```

Por regla general, el navegador Web no termina la carga de la página hasta que no se finaliza la carga del Cliente, por lo que no suele ser necesario agregar sentencias de este tipo.

4.3.2 Cambios en los procedimientos

4.3.2.1 Confirmaciones de acceso a disco

A partir de la versión 3.3, el Cliente @firma pide confirmación al usuario mediante un diálogo modal por cada acceso a disco que se realice sin que este lo haya indicado explícitamente. Este comportamiento no debe afectar a las aplicaciones desplegadas salvo en caso particulares en el uso del modo de firma masiva programática.

Si su aplicación utiliza el modo de firma masiva programática para la firma masiva de ficheros, asegúrese de utilizar el método `massiveSignatureSign(String)` para la ejecución de las firmas.

Si su aplicación utiliza el modo de firma masiva programática y almacena las firmas resultantes en disco, evalúe el uso del mecanismo de firma de directorios o el usuario se verá obligado a aceptar el guardado individual de cada una de las firmas resultantes.

4.3.2.2 Formato de firma por defecto

En las nuevas versiones del Cliente @firma el formato de firma por defecto es CAdES.

Si su herramienta no configurase el formato de firma explícitamente mediante el método `setSignatureFormat(String)`, deberá introducir esta sentencia indicando el formato que

desee utilizar o modificar el formato de firma por defecto por medio de la variable `signatureFormat`, del fichero de configuración "*constantes.js".*

4.3.2.3 Cadena de certificación en firmas XAdES

En las nuevas versiones del Cliente @firma se inserta la cadena de certificación completa en las firmas XAdES generadas. Este es el comportamiento recomendado para firmas XAdES, pero impide la correcta validación de las firmas con versiones de la Plataforma @firma anteriores a la 5.5. Para desactivar este comportamiento es necesario establecer el parámetro extra "`includeOnlySignningCertificate`" al valor "`true`". Podemos hacer esto mediante la siguiente llamada:

- `clienteFirma.addExtraParam("includeOnlySignningCertificate", "true");`

4.3.2.4 Configurar política de firma

La última versión del Cliente @firma amplia las opciones para la configuración de la política de firma de las firmas avanzadas generadas.

En versiones anteriores el método `setPolicy(String, String, String)`, recibía el identificador, la descripción y el cualificador de la política de firma que se desease establecer. En su última versión, el Cliente @firma agrega un cuarto parámetro que es la huella digital (hash) SHA1 de la política de firma. Este hash se debe introducir en base 64.

Ahora el método `setPolicy` tiene la forma:

```
voidsetPolicy(String identifier,String description,String qualifier, String hashB64);
```

Un ejemplo de establecimiento de política es:

```
clienteFirma.setPolicy("urn:oid:2.16.724.1.3.1.1.2.1.8",“Politica de la AGE”,“http://administracionelectronica.gob.es/es/ctt/politicafirma/politica_firma_AGE_v1_8.pdf”,“7SxX3erFuH31TvAw9LZ70N7p1vA=”);
```

Así mismo, ahora también es posible indicar como identificador de la política una URL, URN u OID.

4.3.2.5 Cifrado de datos

En versiones anteriores del Cliente, el comportamiento de los métodos de entrada y salida de las funciones de cifrado tenían un comportamiento errático. Por ejemplo, el método `setPlainData(String)` para indicar los datos que se desean cifrar recibía el texto en claro para cifrar, mientras que `getPlainData()` devolvía este texto en base 64.

En la nueva versión del Cliente @firma todas las entradas y salidas de datos realizadas con los siguientes métodos se realizarán en base 64:

- `setPlainData(String)`

- `setCipherData(String)`
- `getPlainData()`
- `getCipherData()`

Si actualmente invoca a alguno de estos métodos pasándole o recibiendo los datos en texto claro utilice los métodos `getBase64FromText(String, String)` y `getTextFromBase64(String, String)` del cliente para realizar las transformaciones oportunas.

4.3.2.6 Configuración de los destinatarios de los sobres electrónicos

Antes existía un solo método para configurar los destinatarios de los sobres electrónicos:

- `setRecipientsToCMS(String)`: Este método definía todos los destinatarios del sobre de una sola vez. Como parámetro recibía el listado de rutas locales de los certificados de los destinatarios separadas por '\n'.

En la nueva versión, existen dos métodos adicionales para indicar los destinatarios:

- `addRecipientToCMS(String)`: Permite agregar un certificado al listado de destinatarios del sobre. Recibe el certificado codificado en base 64.
- `removeRecipientToCMS(String):` Permiteeliminar un certificado al listado de destinatarios del sobre. Recibe el certificado codificado en base 64.

El método `removeRecipientToCMS` elimina cualquier certificado de la lista, ya haya sido cargado mediante `addRecipientToCMS` o `setRecipientsToCMS`. Si deseamos eliminar todos los certificados podemos utilizar `setRecipientsToCMS(null)`.

4.3.2.7 Codificación de textos

Los métodos `getBase64FromText(String)` y `getTextFromBase64(String)` se sustituyen por una versión que permite seleccionar la codificación de texto a utilizar. La declaración de los nuevos métodos es:

- `String getBase64FromText(String plainText, String charsetName);`
- `String getTextFromBase64(String b64, String charsetName);`

Si no se establece el parámetro `charsetName` se tomará la codificación por defecto del sistema. En caso de no saber con certeza la codificación del texto, se deberá indicar `null`. En el caso de tratarse un XML debe indicarse siempre `null` para que se autodetecte la codificación del XML.

4.3.3 Restricciones

4.3.3.1 Funciones y métodos eliminados en el Applet Java

public String Firma(String datos)

Se elimina el método deprecado `Firma(String)`. Este método permitía la compatibilidad del Cliente con la Plataforma @firma versión 4.

Este método recibía una cadena en base 64 compuesta por un código de operación, el valor de transacción y hasta un par de parámetros, todos concatenados por el separador almohadilla ('#'):

OP#TRANS#PARAM1#PARAM2

Los valores permitidos para el parámetro de operación (OP) eran:

- `0`: Operación de Firma. Para esta operación PARAM1 es el hash de los datos que deseamos firmar y se ignora PARAM2.
- `1`: Operación de Cofirma. Para esta operación PARAM1 es el hash de los datos que deseamos cofirmar y PARAM2 es la firma electrónica original.
- `2`: Operación de Contrafirma de nodos. Para esta operación PARAM1 es la firma que deseamos contrafirmar y PARAM2 es el listado de índices tal como los recibe el método `setSignersToCounterSign(String)`.

El resultado de la operación es la cadena:

cert=CERT;enc=SIGN

En esta cadena CERT es el certificado en base 64 utilizado para la firma y SIGN es el resultado de la operación en base 64.

En cada operación se declaraba un atributo firmado adicional con resto a los que aparecen por regla general en las firmas CMS. Este sería el atributo "`2.5.4.45`" con el valor "TRANS".

Para imitar este comportamiento con el Cliente, se deberían realizar las siguientes acciones:

- Descodificar el Base64 del parámetro que recibía el método y obtener cada uno de sus componentes (OP, TRANS, PARAM1 y PARAM2).
- Configurar el formato de firma CMS, el algoritmo de firma SHA1withRSA y el modo de firma explícito, mediante las sentencias:
  - `clienteFirma.setSignatureFormat("CMS");`
  - `clienteFirma.setSignatureAlgorithm("SHA1withRSA");`
  - `clienteFirma.setSignatureMode("explicit);`
- Configurar el número de transacción como atributo firmado adicional:
  - `clienteFirma.addSignedAttribute("2.2.4.45", TRANS);`
- Configurar los datos de entrada, hash, firma y firmantes según la operación inidicada (OP) y ejecutar la operación:

```
switch(OP) {
// Firma
case 0:
     // Establecemos el hash en base 64 para la firma
     clienteFirma.setHash(PARAM1);
     clienteFirma.sign();
     break;

// Cofirma (Firma en paralelo)
case 1:
     // Establecemos el hash en base 64 para la firma
     clienteFirma.setHash(PARAM1);
     //Establecemos la firma
```

```
            clienteFirma.setElectronicSignature(PARAM2);
            //Cofirmamos
            clienteFirma.coSign();
            break;

        // Contrafirma (Firma en cascada)
        case 2:
            //Establecemos la firma
            clienteFirma.setElectronicSignature(PARAM1);
            //Establecemos los firmantes que se desean contrafirmar
            clienteFirma.setSignersToCounterSign(PARAM2);
            // Contrafirmamos
            clienteFirma.counterSignIndexes();
            break;
        }
```

- Recuperar el resultado de la operación y el certificado utilizado para, si se desea, componer la cadena de salida.
  - `clienteFirma.getSignCertificateBase64Encoded();`
  - `clienteFirma.getSignatureBase64Encoded();`

public String getCMSData()

Se elimina el método en favor del equivalente `getB64Data()`.

publicvoidchangeLanguage(String)

Se elimina el método. En su lugar, para establecer el idioma, podemos proporcionar como parámetros del applet:

- `language`: Código de idioma conforme a la ISO 639. Por ejemplo: “es”, “en”, “ar”, “de”…
- `country`: Código de país o región conforme a la ISO 3166. Por ejemplo, “ES”, “UK”, “US”, “DE”…
- `variant`: Código de variante de libre uso.

Los tres parámetros son opcionales. Sólo se tendrá en cuenta el parámetro `country` si también se ha proporcionado el parámetro `language`; y el parámetro `variant` si se han indicado los otros dos.

4.4 Migración desde el Cliente 3.2

En la nueva versión del Cliente sólo se instalan las dependencias en caso de ser necesarias. Durante la carga del Cliente, si se detecta que el entorno puede requerir la instalación de dependencias del Cliente, se cargará un Bootloader encargado de comprobar si estas dependencias están cubiertas o es necesario instalarlas. La carga del Bootloader, las comprobaciones y la instalación de dependencias son automáticas y el integrador no tiene que hacer nada al respecto.

Cuando el sistema del usuario no es susceptible de requerir dependencias adicionales, se cargará directamente el Cliente y no el Bootloader.

Para actualizar el Cliente @firma a la última versión y adaptar nuestra aplicación Web deberemos seguir los siguientes pasos:

1. Sustituir la totalidad de ficheros de despliegue (bibliotecas JavaScript, ficheros JNLP, archivos JAR, ficheros ZIP, ficheros de propiedades y cualquier otro ficheros distribuido con el Cliente @firma) del cliente por las de la última versión. Durante este proceso, al sustituir el fichero "*constantes.js*", deberemos asegurarnos de que las constantes del nuevo cliente tienen asignadas el mismo valor que el del cliente desplegado. Consulte el apartado Despliegue del Cliente @firmapara más detalle.
2. Revisar si alguno de los métodos utilizados ha cambiado su comportamiento según se indica en el apartado Cambios en los procedimientos para asegurarnos de que no afecta a nuestra aplicación. En caso de afectarnos, proceder tal como se indica.
3. Comprobar que no hacemos uso de ninguno de los métodos eliminados del Cliente. De hacerlo, lo sustituiremos según se indica en el apartado Restricciones, o por un mecanismo alternativo si no existe alternativa.

4.4.1 Despliegue del Cliente @firma

4.4.1.1 Importación de librerías JavaScript

Las librerías JavaScript del nuevo cliente @firma han sufrido ciertos cambios desde sus versiones anteriores, por lo que deberá revisarse el modo de uso desde sus páginas Web HTML. Estos cambios se han realizado para mejorar la compatibilidad con ciertas versiones del entorno de ejecución de Java (JRE), pero sobre todo para garantizar el funcionamiento correcto con versiones futuras de JRE, navegadores Web y sistemas operativos, especialmente en las nuevas arquitecturas de 64 bits.

- El integrador ya no necesita gestionar la instalación y actualización del Cliente, dado que estos procesos se gestionan de forma completamente automática. Debe retirar cualquier referencia en su código HTML al Bootloader y a los siguientes métodos JavaScript o parámetros:
  - instalar()
  - desinstalar()
  - isActualizado()

- actualizar()
- isInstalado()
- getDirectorioInstalacion()
- Variable installDirectory en *constantes.js*
- Variable instalador en *instalador.js*

- Los JavaScript que necesita importar en sus páginas HTML pueden igualmente haber cambiado. Asegúrese de que no importa ningún JavaScript que no exista en el directorio de despliegue o en su subdirectorio `common-js`.
  - Consulte las páginas Web de ejemplo para determinar que ficheros JavaScript es necesario importar para las distintas operaciones de criptografía y firma electrónica.

Note especialmente que se han eliminado las librerías JavaScript "*runApplet.js*" y "*time.js*".Es obligatorio eliminar las esperas explícitas a la carga del *Applet*. Esto es, las sentencias que hacen uso de los métodos de la biblioteca *"time.js"* y la variable `clienteFirmaCargado`. Por ejemplo:

```
whenTry("clienteFirmaCargado == true", "clienteFirma.setCipherAlgorithm('"
+cipherAlgorithm+"')", "No se ha podido iniciar el Applet de firma.");
```

Por regla general, el navegador Web no termina la carga de la página hasta que no se finaliza la carga del Cliente, por lo que no suele ser necesario agregar sentencias de este tipo.

4.4.2 Cambios en los procedimientos

4.4.2.1 Confirmaciones de acceso a disco

A partir de la versión 3.3, el Cliente @firma pide confirmación al usuario mediante un diálogo modal por cada acceso a disco que se realice sin que este lo haya indicado explícitamente. Este comportamiento no debe afectar a las aplicaciones desplegadas salvo en caso particulares en el uso del modo de firma masiva programática.

Si su aplicación utiliza el modo de firma masiva programática para la firma masiva de ficheros, asegúrese de utilizar el método `massiveSignatureSign(String)` para la ejecución de las firmas.

Si su aplicación utiliza el modo de firma masiva programática y almacena las firmas resultantes en disco, evalúe el uso del mecanismo de firma de directorios o el usuario se verá obligado a aceptar el guardado individual de cada una de las firmas resultantes.

4.4.2.2 Formato de firma por defecto

En las nuevas versiones del Cliente @firma el formato de firma por defecto es CAdES.

Si su herramienta no configurase el formato de firma explícitamente mediante el método `setSignatureFormat(String)`, deberá introducir esta sentencia indicando el formato que desee utilizar o modificar el formato de firma por defecto por medio de la variable `signatureFormat`, del fichero de configuración "*constantes.js".*

4.4.2.3 Cadena de certificación en firmas XAdES

En las nuevas versiones del Cliente @firma se inserta la cadena de certificación completa en las firmas XAdES generadas. Este es el comportamiento recomendado para firmas XAdES, pero impide la correcta validación de las firmas con versiones de la Plataforma @firma anteriores a la 5.5. Para desactivar este comportamiento es necesario establecer el parámetro extra "`includeOnlySignningCertificate`" al valor "`true`". Podemos hacer esto mediante la siguiente llamada:

- `clienteFirma.addExtraParam("includeOnlySignningCertificate", "true");`

4.4.2.4 Configurar política de firma

La última versión del Cliente @firma amplia las opciones para la configuración de la política de firma de las firmas avanzadas generadas.

En versiones anteriores el método `setPolicy(String, String, String)`, recibía el identificador, la descripción y el cualificador de la política de firma que se desease establecer. En su última versión, el Cliente @firma agrega un cuarto parámetro que es la huella digital (hash) SHA1 de la política de firma. Este hash se debe introducir en base 64.

Ahora el método `setPolicy` tiene la forma:

```
voidsetPolicy(String identifier,String description,String qualifier, String hashB64);
```

Un ejemplo de establecimiento de política es:

```
clienteFirma.setPolicy("urn:oid:2.16.724.1.3.1.1.2.1.8",“Politica de la AGE”,“http://administracionelectronica.gob.es/es/ctt/politicafirma/politica_firma_AGE_v1_8.pdf”,“7SxX3erFuH31TvAw9LZ70N7p1vA=”);
```

Así mismo, ahora también es posible indicar como identificador de la política una URL, URN u OID.

4.4.2.5 Cifrado de datos

En versiones anteriores del Cliente, el comportamiento de los métodos de entrada y salida de las funciones de cifrado tenían un comportamiento errático. Por ejemplo, el método `setPlainData(String)` para indicar los datos que se desean cifrar recibía el texto en claro para cifrar, mientras que `getPlainData()` devolvía este texto en base 64.

En la nueva versión del Cliente @firma todas las entradas y salidas de datos realizadas con los siguientes métodos se realizarán en base 64:

- `setPlainData(String)`
- `setCipherData(String)`
- `getPlainData()`

- `getCipherData()`

Si actualmente invoca a alguno de estos métodos pasándole o recibiendo los datos en texto claro utilice los métodos `getBase64FromText(String, String)` y `getTextFromBase64(String, String)` del cliente para realizar las transformaciones oportunas.

4.4.2.6 Codificación de textos

Los métodos `getBase64FromText(String)` y `getTextFromBase64(String)` se sustituyen por una versión que permite seleccionar la codificación de texto a utilizar. La declaración de los nuevos métodos es:

- `String getBase64FromText(String plainText, String charsetName);`
- `String getTextFromBase64(String b64, String charsetName);`

Si no se establece el parámetro `charsetName` se tomará la codificación por defecto del sistema. En caso de no saber con certeza la codificación del texto, se deberá indicar `null`. En el caso de tratarse un XML debe indicarse siempre `null` para que se autodetecte la codificación del XML.

4.4.3 Restricciones

4.4.3.1 Funciones y métodos eliminados en el Applet Java

public String Firma(String datos)

Se elimina el método deprecado `Firma(String)`. Este método permitía la compatibilidad del Cliente con la Plataforma @firma versión 4.

Este método recibía una cadena en base 64 compuesta por un código de operación, el valor de transacción y hasta un par de parámetros, todos concatenados por el separador almohadilla ('#'):

OP#TRANS#PARAM1#PARAM2

Los valores permitidos para el parámetro de operación (OP) eran:

- `0`: Operación de Firma. Para esta operación PARAM1 es el hash de los datos que deseamos firmar y se ignora PARAM2.
- `1`: Operación de Cofirma. Para esta operación PARAM1 es el hash de los datos que deseamos cofirmar y PARAM2 es la firma electrónica original.
- `2`: Operación de Contrafirma de nodos. Para esta operación PARAM1 es la firma que deseamos contrafirmar y PARAM2 es el listado de índices tal como los recibe el método `setSignersToCounterSign(String)`.

El resultado de la operación es la cadena:

cert=CERT;enc=SIGN

En esta cadena CERT es el certificado en base 64 utilizado para la firma y SIGN es el resultado de la operación en base 64.

En cada operación se declaraba un atributo firmado adicional con resto a los que aparecen por regla general en las firmas CMS. Este sería el atributo "2.5.4.45" con el valor "TRANS".

Para imitar este comportamiento con el Cliente, se deberían realizar las siguientes acciones:

- Descodificar el Base64 del parámetro que recibía el método y obtener cada uno de sus componentes (OP, TRANS, PARAM1 y PARAM2).
- Configurar el formato de firma CMS, el algoritmo de firma SHA1withRSA y el modo de firma explícito, mediante las sentencias:
  - `clienteFirma.setSignatureFormat("CMS");`
  - `clienteFirma.setSignatureAlgorithm("SHA1withRSA");`
  - `clienteFirma.setSignatureMode("explicit);`
- Configurar el número de transacción como atributo firmado adicional:
  - `clienteFirma.addSignedAttribute("2.2.4.45", TRANS);`
- Configurar los datos de entrada, hash, firma y firmantes según la operación inidicada (OP) y ejecutar la operación:

```
switch(OP) {
// Firma
case 0:
     // Establecemos el hash en base 64 para la firma
     clienteFirma.setHash(PARAM1);
     clienteFirma.sign();
     break;

// Cofirma (Firma en paralelo)
case 1:
     // Establecemos el hash en base 64 para la firma
     clienteFirma.setHash(PARAM1);
     //Establecemos la firma
     clienteFirma.setElectronicSignature(PARAM2);
     //Cofirmamos
     clienteFirma.coSign();
     break;

// Contrafirma (Firma en cascada)
case 2:
     //Establecemos la firma
     clienteFirma.setElectronicSignature(PARAM1);
     //Establecemos los firmantes que se desean contrafirmar
     clienteFirma.setSignersToCounterSign(PARAM2);
     // Contrafirmamos
     clienteFirma.counterSignIndexes();
     break;
}
```

- Recuperar el resultado de la operación y el certificado utilizado para, si se desea, componer la cadena de salida.
  - `clienteFirma.getSignCertificateBase64Encoded();`
  - `clienteFirma.getSignatureBase64Encoded();`

public String getCMSData()

Se elimina el método en favor del equivalente `getB64Data()`.

publicvoidchangeLanguage(String)

Se elimina el método. En su lugar, para establecer el idioma, podemos proporcionar como parámetros del applet:

- `language`: Código de idioma conforme a la ISO 639. Por ejemplo: “es”, “en”, “ar”, “de”…
- `country`: Código de país o región conforme a la ISO 3166. Por ejemplo, “ES”, “UK”, “US”, “DE”…
- `variant`: Código de variante de libre uso.

Los tres parámetros son opcionales. Sólo se tendrá en cuenta el parámetro `country` si también se ha proporcionado el parámetro `language`; y el parámetro `variant` si se han indicado los otros dos.

4.5 Migración desde el Cliente 3.3

En la nueva versión del Cliente es una revisión menor desde el Cliente 3.3 que corrige algunos bugs del mismo, añade la compatibilidad con las últimas versiones de Firefox y agrega nuevas opciones de configuración. Así pues, la migración del Cliente 3.3 al 3.1 es prácticamente instantánea y sólo es necesario, si lo deseamos, hacer cambios para la corrección de su comportamiento mediante las nuevas propiedades que incorpora.

Para actualizar el Cliente @firma a la última versión y adaptar nuestra aplicación Web deberemos seguir los siguientes pasos:

1. Sustituir la totalidad de ficheros de despliegue (bibliotecas JavaScript, ficheros JNLP, archivos JAR, ficheros ZIP, ficheros de propiedades y cualquier otro ficheros distribuido con el Cliente @firma) del cliente por las de la última versión. Durante este proceso, al sustituir el fichero "*constantes.js*", deberemos asegurarnos de que las constantes del nuevo cliente tienen asignadas el mismo valor que el del cliente desplegado.
2. Revisar si alguno de los métodos utilizados ha cambiado su comportamiento según se indica en el apartado Cambios en los procedimientospara asegurarnos de que no afecta a nuestra aplicación. En caso de afectarnos, proceder tal como se indica.

4.5.1 Cambios en los procedimientos

4.5.1.1 Cadena de certificación en firmas XAdES

En las nuevas versiones del Cliente @firma se inserta la cadena de certificación completa en las firmas XAdES generadas. Este es el comportamiento recomendado para firmas XAdES, pero impide la correcta validación de las firmas con versiones de la Plataforma @firma anteriores a la 5.5. Para desactivar este comportamiento es necesario establecer el parámetro extra "`includeOnlySignningCertificate`" al valor "`true`". Podemos hacer esto mediante la siguiente llamada:

- `clienteFirma.addExtraParam("includeOnlySignningCertificate", "true");`

5 Glosario de términos

Firma electrónica

Es el conjunto de datos, en forma electrónica, anejos a otros datos electrónicos o asociados funcionalmente con ellos, utilizados como medio para identificar formalmente al autor o a los autores del documento que la recoge.

XML Digital Signature (XMLDSig)

Es una recomendación del W3C que define una sintaxis XML para la firma digital

XML AdvancedSignature (XAdES)

Es un conjunto de extensiones a las recomendaciones XML-DSig haciéndolas adecuadas para la firma electrónica avanzada.

RSA

Es un sistema criptográfico de clave pública desarrollado en 1977. En la actualidad, RSA es el primer y más utilizado algoritmo de este tipo y es válido tanto para cifrar como para firmar digitalmente.

XML

Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

Office Open XML (OOXML)

Es un formato de archivo abierto y estándar cuyas extensiones más comunes son .docx, .xlsx y .pptx. Se le utiliza para representar y almacenar hojas de cálculo, diagramas, presentaciones y documentos de texto. Un archivo Office Open XML contiene principalmente datos basados en el lenguaje de marcado XML, comprimidos en un contenedor .zip específico.

Open DocumentFormat (ODF)

Es un formato de fichero estándar para el almacenamiento de documentos ofimáticos tales como hojas de cálculo, memorandos, gráficas y presentaciones. Aunque las especificaciones fueron inicialmente elaboradas por Sun, el estándar fue desarrollado por el comité técnico para Open Office XML de la organización OASIS y está basado en un esquema XML inicialmente creado e implementado por la suite ofimática OpenOffice.org (ver OpenOffice.org XML).

ZIP

Es un formato de almacenamiento sin pérdida, muy utilizado para la compresión de datos como imágenes, programas o documentos.

PDF

Es un formato de almacenamiento de documentos, desarrollado por la empresa Adobe Systems. Este formato es de tipo compuesto (imagen vectorial, mapa de bits y texto).

SHA

Es un sistema de funciones hash criptográficas relacionadas de la Agencia de Seguridad Nacional de los Estados Unidos y publicadas por el NationalInstitute of Standards and Technology (NIST). El primer miembro de la familia fue publicado en 1993 es oficialmente llamado SHA. Sin embargo, hoy día, no oficialmente se le llama SHA-0 para evitar confusiones con sus sucesores. Dos años más tarde el primer sucesor de SHA fue publicado con el nombre de SHA-1. Existen cuatro variantes más que se han publicado desde entonces cuyas diferencias se basan en un diseño algo modificado y rangos de salida incrementados: SHA-224, SHA-256, SHA-384, y SHA-512 (llamándose SHA-2 a todos ellos).

PKCS

Se refiere a un grupo de estándares de criptografía de clave pública concebidos y publicados por los laboratorios de RSA en California. A RSA Security se le asignaron los derechos de licenciamiento para la patente de algoritmo de clave asimétrica RSA y adquirió los derechos de licenciamiento para muchas otras patentes de claves.

W3C

Es un consorcio internacional que produce recomendaciones para la World Wide Web. Está dirigida por Tim Berners-Lee, el creador original de URL (UniformResourceLocator, Localizador Uniforme de Recursos), HTTP (HyperText Transfer Protocol, Protocolo de Transferencia de HiperTexto) y HTML (Lenguaje de Marcado de HiperTexto) que son las principales tecnologías sobre las que se basa la Web.

OpenOffice.org

Es una suite ofimática libre (código abierto y distribución gratuita) que incluye herramientas como procesador de textos, hoja de cálculo, presentaciones, herramientas para el dibujo vectorial y base de datos. Está disponible para varias plataformas, tales como Microsoft Windows, GNU/Linux, BSD, Solaris y Mac OS X. Soporta numerosos formatos de archivo, incluyendo como predeterminado el formato estándar ISO/IEC OpenDocument (ODF), entre otros formatos comunes. A febrero de 2010, OpenOffice.org soporta más de 110 idiomas.

Base64

Es un sistema de numeración posicional que usa 64 como base. Es la mayor potencia de dos que puede ser representada usando únicamente los caracteres imprimibles de ASCII. Esto ha propiciado su uso para codificación de correos electrónicos, PGP y otras aplicaciones. Todas las variantes famosas que se conocen con el nombre de Base64 usan el rango de caracteres A-Z, a-z y 0-9 en este orden para los primeros 62 dígitos, pero los símbolos escogidos para los últimos dos dígitos varían considerablemente de unas a otras. Otros métodos de codificación como UUEncode y las últimas versiones de binhex usan un conjunto diferente de 64 caracteres para representar 6 dígitos binarios, pero éstos nunca son llamados Base64.

ASN.1

Es una norma para representar datos independientemente de la máquina que se esté usando y sus formas de representación internas. Es un protocolo de nivel de presentación en el modelo OSI.

Autoridad de Certificación (CA)

Es una entidad de confianza, responsable de emitir y revocar los certificados digitales o certificados, utilizados en la firma electrónica, para lo cual se emplea la criptografía de clave pública. Jurídicamente es un caso particular de Prestador de Servicios de Certificación.

Certificado Digital

Es un documento digital mediante el cual un tercero confiable (una autoridad de certificación) garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública.

Infraestructura de Clave Pública (PKI)

Es una combinación de hardware y software, políticas y procedimientos de seguridad que permiten la ejecución con garantías de operaciones criptográficas como el cifrado, la firma digital o el no repudio de transacciones electrónicas.