SeguriSign versión 8.6

Manual de Applet

Revisión 1.0

SeguriData

SeguriData Privada, S.A. de C.V.

Av. Insurgentes Sur #2375, 3er. piso,
Col. Tizapán, Del. Alvaro Obregón,
C.P. 01000, México, D.F.

Tel. +52 (55) 3098-0700
Fax. +52 (55) 3098-0702

http://www.seguridata.com

Número de Parte: SSIGApplet_8.6

Contenido

CAPÍTULO 1

¿Cómo utilizar este manual?

1.1 Simbología y convenciones

En todo el manual se hace uso de una simbología específica para hacer más sencilla la identificación del tipo de información que se expone, así como de convenciones tipográficas, para hacer más clara la documentación.

1.1.1 Recomendaciones y Advertencias

En los lugares que resulte mas oportuno, se insertarán comentarios sobre el contenido del texto.

Importante

Este tipo de anotaciones contienen sugerencias y aclaraciones que facilitan el uso de la aplicación.

Precaución

Este tipo de anotaciones advierten sobre posibles riesgos en las operaciones descritas

1.2 Objetivo de este manual

Servir como guía para generar mensajes criptográficos firmados utilizando el Applet SeguriSign en aplicaciones existentes o por desarrollar.

Este manual está orientado al personal que actuará en etapas de diseño y/o desarrollo de aplicaciones que integran firma digital a través de SeguriSign.

1.3 Control de Cambios

En la siguiente tabla se actualizan los cambios de este manual. (Tabla 1.1)

Tabla 1.1

Versión	Fecha de la modificación	Modificación
8.6	28-Jun-2011	* Ya no es necesario habilitar los permisos que normalmente se debían definir en el archivo .java.policy. * La clase signer, ahora es Signer. * Se habilita un método para disponer pares de llaves en archivos separados (.key y .cer) en formato PKCS#12. * Firmado con certificado de Publisher vigente e incluye estampa de tiempo de modo que pueda validarse el tiempo de firma, aún después de expirado el certificado de SeguriData.

CAPÍTULO 2

Uso del Applet de SeguriSign para Firma Digital

Como parte de los servicios que proporciona SeguriSign Cliente, se distribuye un componente (Applet) el cual permite realizar firmas digitales desde diversos navegadores y plataformas. El usuario puede firmar un archivo o una cadena de caracteres utilizando pares de llaves dispuestas en formato PKCS#12, en el repositorio de Windows o en Firefox.

Sin embargo, deben cumplirse ciertos requisitos en el equipo del usuario para habilitar su uso.

2.1 Requerimientos

El equipo del usuario debe tener instalado:

- el Plug-In de Java 6

2.2 Otros Requerimientos de Software

Los requerimientos adicionales de Software se enumeran a continuación:

- JRE 1.6.0_20 o superior

2.2.1 El Plug-In de Java

El equipo del usuario debe contar con un navegador al que pueda asociarse el Plug-In de Java (la mayoría de los navegadores actuales cumplen con este requisito) y verificar la versión del mismo. Si ésta no está dentro del rango indicado, debe instalarse el JRE (Java Runtime Environment) proporcionado gratuitamente por *Oracle* (`http://www.java.com`).

2.3 Uso del Applet

2.3.1 Inicialización

Al accesar la página de internet que hace referencia al Applet Cliente de SeguriSign, se desplegará una ventana similar a la de la Figura 2.1, (dependiendo de la versión del Plug-In de Java puede variar).

Figura 2.1 Solicitud de Confirmación

Este cuadro de diálogo, solicita la autorización del usuario para habilitar la funcionalidad del applet. Las opciones que proporciona son:

- *Ejecutar*. Autoriza la ejecución del applet. Si no se encuentra seleccionada la opción "Confiar siempre en el contenido de este editor" se indica que la ejecución se autoriza por única ocasión, en caso contrario se ejecutará siempre.
- *Cancelar*. Niega por única ocasión la ejecución del applet

Importante

Las etiquetas de los botones pueden variar según el idioma.

Importante

El applet no requiere instalación previa, pues se descarga automáticamente al momento de accesar la página de internet que hace referencia a él.

Al dar clic en la opción “Más información...” se obtienen detalles pertinentes al proceso de validación del applet, la confianza heredada para el mismo así como la fecha de firma del componente. (Figura 2.2).

Figura 2.2 Información del Proceso de validación

La opción *Detalles del certificado...* presenta particularidades del certificado del Emisor del applet, en este caso SeguriData. (Figura 2.3).

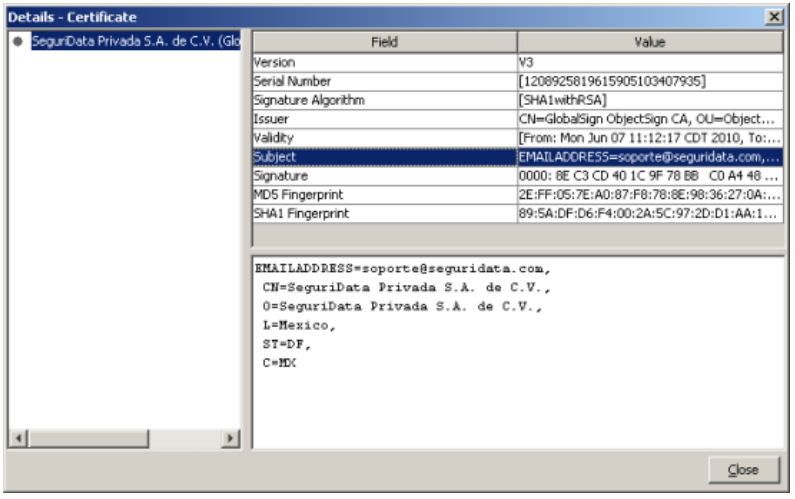

Figura 2.3 Detalle del Certificado

Una vez otorgado el permiso, (haciendo clic en el botón *Ejecutar*) se podrá emplear la funcionalidad de firma del applet.)

> **Importante**
>
> Al presionar el botón Run y dejar habilitado el checkbox “Always trust content from this publisher” se evita que se continúe solicitando permiso de ejecución cada vez que se ejecute un ejemplo.

En caso de visualizar la siguiente advertencia, dar clic derecho sobre la misma e indicar que se permite la ejecución:

2.3.2 Crear Archivo PKCS#12 en Applet SeguriSign (SeguriSign_Applet_Ej_CodeP12.html)

El siguiente ejemplo ilustra una opción de cómo integrar el Applet SeguriSign para conformar un archivo PKCS#12 a partir de archivos de llave y certificado digital.

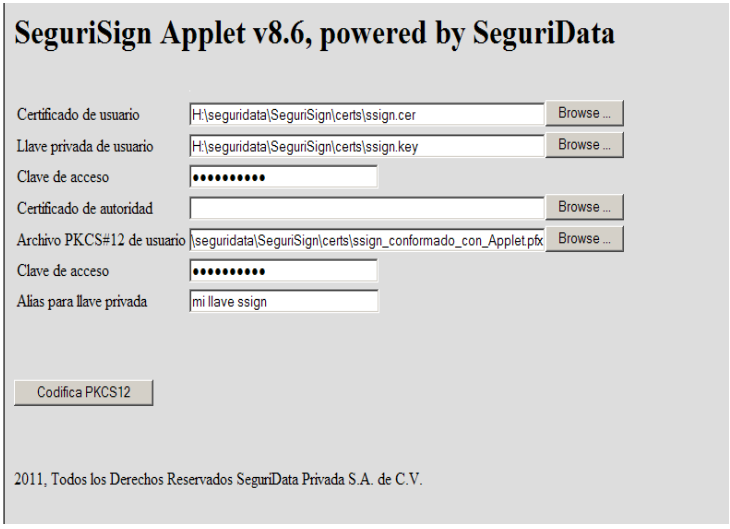

Figura 2.4 Crear archivo PKCS#12

Para conformar un archivo PKCS#12 a partir de archivo de llave privada y certificado digital, se realizan los siguientes pasos:

1- Seleccionar los archivos de llave y certificado e indicar la clave de acceso para la llave privada.

2- Definir el certificado de autoridad (opcional)

3- Definir archivo destino para PKCS#12.

4- Ingresar la clave de acceso.

5- Definir alias o identificador para la llave privada y presionar el botón “Codifica PKCS12”

Cuando se realice una ejecución correcta, saldrá el siguiente mensaje:(Figura 2.5)

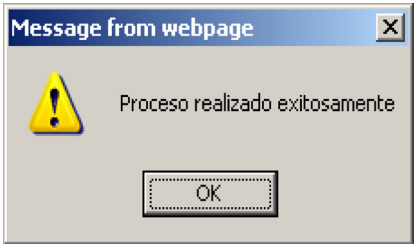

Figura 2.5 Ejecución correcta.

Cuando exista un error durante la ejecución, saldrá un mensaje similar al siguiente según la naturaleza del error: (Figura 2.6)

Figura 2.6 Mensaje de error cuando la clave de acceso para el archivo .key no es correcta.

2.3.3 Ejemplo de Firma (SeguriSign_Applet_Ej_FF.html)

El siguiente ejemplo muestra cómo firmar o firmar y ensobretar cadenas o archivos utilizando pares de llaves contenidas en el repositorio de llaves de Firefox. De inicio se solicitará la clave del contenedor de certificados de Firefox si se encuentra definida alguna, en caso contrario no será solicitada. (Figura 2.7).

Figura 2.7 Proporcionar la clave de acceso.

Una vez ingresada la clave de acceso, será desplegados los certificados contenidos en el repositorio de Firefox. El ejemplo utiliza un componente **select** de HTML, pero puede integrarse de manera distinta, por ejemplo, una tabla. (Figura 2.8)

Figura 2.8 Ejemplo de Firma

Para aplicar la firma de un mensaje o archivo se deberán completar los siguientes pasos:

1- Si la clave es correcta se populará el objeto etiquetado como Llaves.

2- Seleccionar opción a firmar “Archivo” ó “Mensaje”.

3- Seleccionar archivo o ingresar Mensaje de acuerdo al punto número 2.

4- Pueden seleccionarse las opciones “Generar PKCS#1” ó “Ensobreta”. Para esta última deberá ingresarse un certificado en base 64 en la caja de texto “Destinatario”.

5- Presionar botón “Asegurar”.

Importante

Los ejemplos ilustrados despliegan elementos que no necesariamente se espera estén presentes en una interfaz para usuario final. Se lleva a cabo de esta manera a fin de ilustrar la integración del componente para tener acceso a la funcionalidad que hace disponible

2.3.3.1 Mensajes firmados a partir de digestiones previamente calculadas (SeguriSign_Applet_Ej_FF_digest(tabla).html).

El presente ejemplo ilustra cómo conformar mensajes criptográficos firmados a partir de una digestión previamente calculada. Las llaves de firma se encuentran contenidas en el repositorio de Firefox. Como en el ejemplo anterior, en caso de que haya una contraseña definida se solicitará durante la carga de la página ejemplo, cuando no es así no será solicitada clave de acceso alguna.(Figura 2.9)

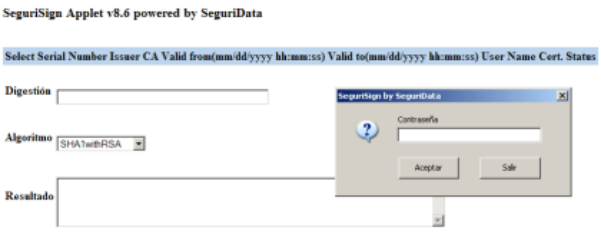

Figura 2.9 Proporcionar la clave de acceso.

Una vez proporcionada la clave se desplegará a manera de tabla, la lista de certificados contenidos. Dicha tabla puede ser susceptible de cambios, ya sea se aplique un estilo diferente, se pagine, etc.

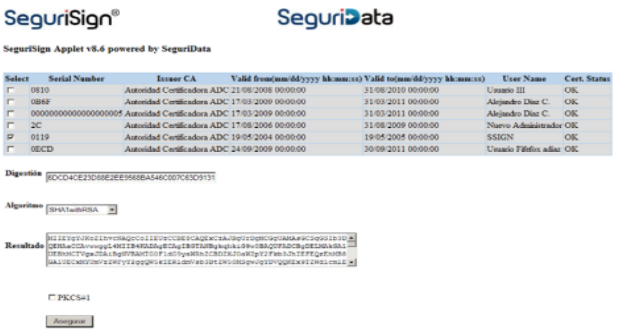

Figura 2.10 Tabla.

Para aplicar la firma se deberán considerar los siguientes pasos:

1- Seleccionar un certificado de la tabla.

2- Ingresar un valor para la digestión.

3- Presionar el botón de “Asegurar”.

2.3.3.2 Mensajes firmados a partir de digestiones utilizando llaves de Firefox (SeguriSign_Applet_Ej_FF_digest.html).

El presente ejemplo es útil para ilustrar cómo integrar firma digital y conformar mensajes a partir de digestiones previamente calculadas, como puede ser el caso de procesos de firma multilateral iniciados en el servidor SeguriSign (Figura 2.11).

Figura 2.11 Ejemplo.

Para aplicar la firma se deberán completar los siguientes pasos:

1- Proporcionar la clave de acceso para repositorio Firefox.

2- Elegir certificado.

3- Ingresar digestión.

4- Presionar botón “Asegurar”

Importante

La digestión que se proporciona en este ejemplo, no debe ser solicitada al usuario final, sino manejada de modo tal que el usuario no sea responsable de la obtención

y/o manipulación de la misma ya que es tarea de la aplicación WEB que integre la firma digital el llevar a cabo dicho tratamiento.

2.3.3.3 Firmar y ensobretar cadenas o archivos con par de llaves en formato PKCS#12 (SeguriSign_Applet_Ej_P12.html).

A fin de ilustrar un proceso mediante el que se obtengan mensajes firmados o mensajes firmados y ensobretados utilizando pares de llaves en archivo bajo formato PKCS#12, se presenta el siguiente ejemplo donde de manera ilustrativa se despliegan los identificadores de pares de llaves contenidos en el PKCS#12 (que normalmente es uno) en un objeto HTML de tipo select.

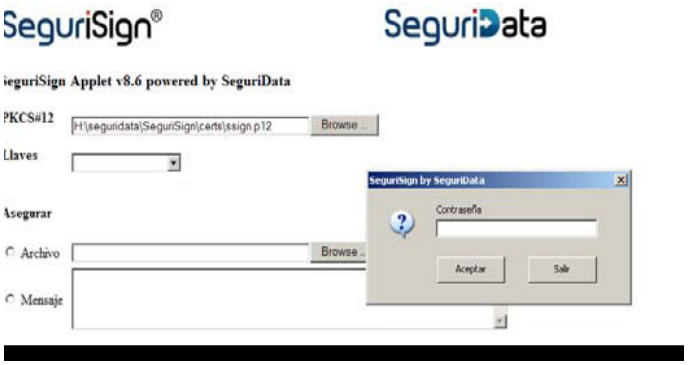

Figura 2.12 Proporcionar la clave.

Seleccionar ruta a archivo de par de llaves y proporcionar la clave cuando se solicite. De acuerdo a como fue programado este ejemplo, al seleccionar el archivo del par de llaves se solicitará la clave de acceso del mismo. En caso de éxito se populará el objeto select identificado por la etiqueta "Llaves" (Figura 2.12)

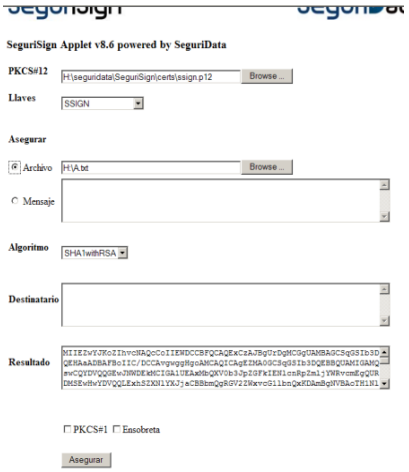

Figura 2.13 Ejemplo.

Para aplicar la firma se deberán completar los siguientes pasos:

1- Elegir opción a firmar: archivo o mensaje

2- Seleccionar archivo o ingresar mensaje. Dependiendo del punto número 2.

3- Presionar botón “Asegurar”.

2.3.3.4 Mensajes firmados a partir de digestiones utilizando llaves contenidas en PKCS#12 (SeguriSign_Applet_Ej_P12_digest.html).

Para integrar firmas para un proceso de firma multilateral en SeguriSign, es decir, sin contenido, puede utilizarse como referencia de integración este ejemplo donde el par de llaves reside en un archivo bajo el estándar PKCS#12 (Figura 2.14).

Figura 2.14 Ejemplo.

Para aplicar la firma se deberán completar los siguientes pasos:

1- Seleccionar archivo PKCS#12 y proporcionar la clave de acceso cuando se solicite.

2- Ingresar digestión.

3- Presionar botón “Asegurar”

2.3.3.5 Firmar y ensobretar mensajes o archivos utilizando pares de llaves del repositorio Windows. (SeguriSign_Applet_Ej_Win.html)

Si el par de llaves a utilizar se encuentra en el repositorio de Windows, es recomendable verificar el código del presente ejemplo a fin de obtener la funcionalidad necesaria y los componentes visuales que sean necesarios, mas no todos. (Figura 2.15)

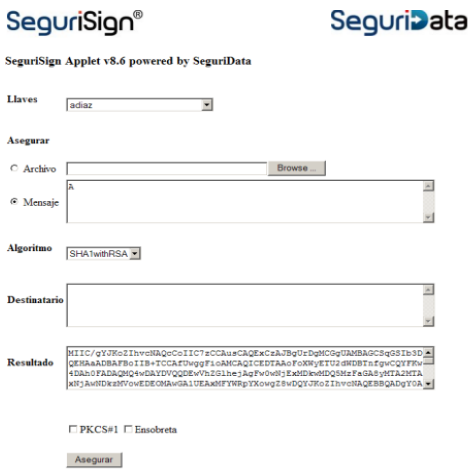

Figura 2.15 Ejemplo.

Para aplicar la firma de un mensaje o archivo se deberán aplicar los siguientes pasos:

1- Seleccionar par de llaves de firma.

2- Seleccionar opción a procesar: Mensaje o archivo.

3- De acuerdo a lo elegido anteriormente, seleccionar archivo o ingresar mensaje a firmar.

4- Presionar botón “Asegurar”.

CAPÍTULO 3

Integración del Cliente SeguriSign en Páginas de Internet

La ubicación del applet que forma parte de SeguriSign Cliente es: **<Ruta_de_instalación>\seguridata\SeguriSignv8.6\Java\applet\SeguriSignClient.jar** este componente no tiene interfaz propia, es por ello que debe integrarse en las páginas HTML de la aplicación donde será utilizado. Para su integración se requiere de:

- Una referencia al applet de SeguriSign (declarada con la etiqueta *<applet>*)
- Una forma HTML (declarada con la etiqueta *<form>*)
- Código JavaScript que permite interactuar con el applet

3.1 Requerimientos

3.1.1 Referencia al Applet

Para hacer referencia al applet desde el código HTML, es necesario incluir dentro del bloque *<BODY>*, la etiqueta *<applet>* tal como se describe a continuación:

```
[1]  <applet
[2]
[3]  code="seguridata.segurisign.Signer"     // Indica la clase principal del applet
[4]
[5]  archive="SeguriSignClient.jar"       // Indica el nombre del archivo jar que
[6]                                                    contiene la clase principal del applet
[7]
[8]  width="0"                    // El applet no tiene interfaz gráfica
[9]
[10] height="0"                   // El applet no tiene interfaz gráfica
[11]
```

```
[12] id="theSigner"                    // Identificador de la clase.
[13]
[14] name="theSigner"                  // Alias con el que se alude a este componente
[15]
[16] mayscript="false"                 // Permite la interacción del applet con JavaScript
[17]
```

Importante

Cambiar el valor del atributo *Id* puede afectar la ejecución corrrecta de los ejemplos si no se hace extensivo el cambio cuando se utilice el *Id* para obtener una referencia al Applet en líneas de programación Javascript.

Importante

Como cualquier otro componente que se integra al código HTML (por ejemplo, una imagen), la ruta donde se indica que reside el applet, debe ser (en el caso de rutas absolutas) o convertirse (en el caso de rutas relativas) en una URL válida, de lo contrario el componente no podrá descargarse ni realizar operación alguna.

3.1.2 Los Objetos HTML

Para llevar a cabo el proceso de firma, se requiere proporcionar la siguiente información, como se indica en la Tabla 3.1.

Tabla 3.1 Datos Solicitados para la Forma HTML

Dato	Descripción
`Archivo PKCS12`	Contiene el par de llaves con las que el usuario firmará.
`Llaves`	Llaves contenidas en el repositorio de Firefox
`Asegurar`	Hace referencia al archivo o mensaje firmar.
`Algoritmo`	Algoritmo utilizado en el proceso de la firma.
`Destinatario`	Certificado digital en base 64 para el destinatario del mensaje, en caso de indicar aplicar Sobre Digital mediante la opción Ensobreta.
`Resultado`	Muestra el resultado de la firma.

Con la finalidad de solicitar la información al usuario, e incluir los datos adicionales requeridos, se emplea el siguiente código HTML:

```
[1] Archivo PKCS12: <input type="text"     name="p12File">
[2] Llaves:  <name="theAliases" id="theAliases">
[3] Asegurar: <input name=”myradio” type="radio" >
[4] Algoritmo:   <tname="signAlgorithms" id="signAlgorithms">
[5] Destinatario: <textarea name="destinatarios" id="destinatarios" rows="4" cols="65"></textarea>
[6] Resultado: <textarea name="p7" id="p7" rows="4" cols="65"></textarea>
[7]
[8] </form>
```

3.1.3 Código JavaScript

Mediante este código se permite interactuar con el applet y procesar los datos necesarios antes de enviarlos al Servidor SeguriSign.

```
[1] <script language="Javascript">
[2] var theAppSigner
[3] var theAliases = document.getElementById("theAliases")
[4]
[5] var p7 = document.getElementById("p7")
[6] var opcion=0
[7]
[8] function initValues()
[9] {
[10] theAppSigner = document.getElementById("theSigner")
[11] theAppSigner.setDoJS(true)
[12] theAppSigner.setDoPKCS12(true)
[13] theAliases.options.add(new Option("No hay entradas ..."))
[14] document.getElementById("destinatarios").disabled = true
[15] }
[16]
[17] function f2(x)
[18] {
[19] if(x == 0)
[20] {
[21] document.getElementById("TEXT_TO_SIGN").disabled = true
[22] document.getElementById("FILE_NAME").disabled = false
[23] opcion=1
[24] }
[25] if(x == 1)
[26] {
[27] document.getElementById("FILE_NAME").disabled = true
[28] document.getElementById("TEXT_TO_SIGN").disabled = false
[29] opcion=2
[30] }
[31] }
```

```
[32]
[33] function activa()
[34] {
[35] if (document.getElementById("doEnvelope").checked)
[36]    document.getElementById("destinatarios").disabled = false
[37] else
[38]    document.getElementById("destinatarios").disabled = true
[39] }
[40]
[41] function procesa_excepcion(exc)
[42] {
[43] msg = exc
[44]
[45] if (exc.indexOf("java.io.IOException: DER input, Integer tag error") != -1 || exc.indexOf("java.io.IOException:
       toDerInputStream") != -1)
[46]    msg = "El archivo PKCS12 está corrupto o no está codificado bajo dicho formato";
[47] else
[48] if (exc.indexOf("java.io.IOException: failed to decrypt safe contents entry") != -1 || exc.indexOf("Given final block not properly
       padded") != -1)
[49] msg = "La contraseña del PKCS#12 no es correcta, intente de nuevo"
[50] else
[51] if (exc.indexOf("java.security.NoSuchAlgorithmException") != -1)
[52] msg = "El PKCS#12 se encuentra cifrado con algún algoritmo no soportado por los proveedores existentes"
[53] else
[54]    if (exc.indexOf("CKR_PIN_INCORRECT") != -1 || exc.indexOf("CKR_PIN_LEN_RANGE") != -1)
[55]       msg = "No fue posible tener acceso al dispositivo debido a un NIP incorrecto"
[56] else
[57]    if (exc.indexOf("CKR_FUNCTION_CANCELED") != -1)
[58]       msg = "La operación fue cancelada por el usuario"
[59] else
[60]    if (exc.indexOf("javax.crypto.BadPaddingException: Given final block not properly padded") != -1)
[61]       msg = "La contraseña del PKCS#12 no es correcta, intente de nuevo"
[62] else
[63]    if (exc.indexOf("by zero") != -1)
[64]       msg = "La contraseña del PKCS#12 no es correcta, intente de nuevo"
[65] else
[66]    if (exc.indexOf("java.security.cert.CertificateException") != -1 || exc.indexOf("java.io.IOException: DerInputStream") != -1)
[67]       msg = "El certificado digital parametrizado es incorrecto"
[68] else
[69]    if (exc.indexOf("file does not exist") != -1)
[70]       msg = "La ruta de archivo recibida no es correcta"
[71]    else
[72]    if (exc.indexOf("java.io.EOFException") != -1)
[73]       msg = "Los parámetros recibidos no son correctos"
[74]    else
[75]    if (exc.indexOf("PKCS11 not found") != -1)
[76]       msg = "La firma digital no fue bien integrada a esta aplicación o FF no se encuentra instalado"
[77]
[78] alert(msg)
[79] }
```

```
[80]
[81] function selectTheFile(obj, title, filter, ext, read) {
[82] theAppSigner.getFilePath(title, filter, ext, read)
[83]
[84] if (theAppSigner.isSuccess()) {
[85] var thePath = theAppSigner.getSelectedFileAndPath()
[86]
[87] if (thePath != "")
[88] obj.value = thePath
[89] } else {
[90]     alert(theAppSigner.getErrorMessage())
[91]     return
[92]     }
[93]
[94] var aliasesR
[95]
[96] if (thePath != "" && ext == "p12")
[97] {
[98] try
[99] {
[100] theAliases.options.length = 0
[101] theAppSigner.getAliases(thePath)
[102]
[103] aliasesR = theAppSigner.getStoreAliases()
[104]
[105] if (aliasesR.indexOf("Error") == 0)
[106] {
[107] procesa_excepcion(aliasesR)
[108] obj.value = ""
[109] return
[110] }
[111]
[112] } catch(err)
[113] {
[114] procesa_excepcion(err.description == null ? err.message : err.description)
[115] document.theForm.reset()
[116] }
[117] }
[118] }
[119]
[120] function firma()
[121] {
[122] var p7var = ""
[123] var current = null
[124]
[125] try
[126] {
[127] if (document.getElementById("p12File").value == "")
[128] {
[129] alert("Debe seleccionar el par de llaves para proceder")
```

```
[130] return
[131] }
[132]
[133] if (document.getElementById("doEnvelope").checked)
[134] {
[135] if (document.getElementById("destinatarios").value == "")
[136] {
[137] alert("No ha proporcionado el destinatario para el mensaje ensobretado")
[138] return
[139] }
[140] }
[141]
[142] if (opcion == 1)
[143] {
[144] current = document.getElementById("FILE_NAME")
[145]
[146] if (current.value != "")
[147] {
[148] theAppSigner.setDoFile(true)
[149] theAppSigner.setToSign(current.value)
[150] }
[151] else
[152] {
[153] alert("Por favor seleccione el archivo que desea firmar.")
[154] current.focus()
[155] return
[156] }
[157] }
[158] else if (opcion == 2)
[159] {
[160] current = document.getElementById("TEXT_TO_SIGN")
[161]
[162] if (current.value != "")
[163] {
[164] theAppSigner.setDoFile(false)
[165] theAppSigner.setToSign(current.value)
[166] }
[167] else
[168] {
[169] alert("Por favor ingrese el mensaje que desea firmar")
[170] return
[171] }
[172] }
[173] else
[174] {
[175] alert("Por favor elija la opcion que desea firmar.")
[176] return
[177] }
[178]
[179] theAppSigner.setSigAlg(document.getElementById("signAlgorithms").value)
```

```
[180]
[181]  theAppSigner.setSignerAlias(theAliases.value)
[182]
[183]  if (document.getElementById("doPKCS1").checked && !document.getElementById("doEnvelope").checked)
[184]  theAppSigner.setCodePKCS1()
[185]  else
[186]  theAppSigner.setCodePKCS7()
[187]
[188]  if (document.getElementById("doEnvelope").checked) {
[189]     theAppSigner.envelope(document.getElementById("destinatarios").value)
[190]     }
[191]  else {
[192]  theAppSigner.sign()
[193]    }
[194]
[195]    if (theAppSigner.isSuccess())
[196]       p7var = theAppSigner.getSignature()
[197]    else
[198]       p7var = theAppSigner.getErrorMessage()
[199]
[200]    if (p7var.indexOf("Error") == 0)
[201]    procesa_excepcion(p7var)
[202]    else
[203]  p7.value = p7var
[204]  }catch(err) {
[205]  procesa_excepcion(err.description == null ? err.message : err.description)
[206]  }
[207]  }
[208]
[209]  </script>
```

3.2 Métodos Disponibles en el Applet

Para utilizar la funcionalidad del Applet Cliente de SeguriSign es necesario conocer los distintos métodos disponibles, los cuales se describen a continuación.

- **encodeB64**

Sintaxis: public void encodeB64(java.lang.String filePath)

Descripción: Convierte un archivo a una cadena base 64

Parameters:

filePath - Ruta y nombre del archivo a codificar

- **envelope**

Sintaxis: public void envelope(java.lang.String dest)
throws java.lang.Exception

Descripción: Ensobreta el archivo o cadena de texto definido

Parameters:

dest - Certificado(s) de destinatario(s) en base 64, separados por |

- **exportKeys2PKCS12**

Sintaxis: public void exportKeys2PKCS12(java.lang.String pKeyPath,
java.lang.String certPath,
java.lang.String pKeyPassw,
java.lang.String p12FmtPath,
java.lang.String acCertPath,
java.lang.String alias,
java.lang.String p12FmtPassw)
throws java.lang.IllegalArgumentException,
java.io.IOException,
java.lang.Exception

Descripción: Exporta un par de llaves dispuestas en PKCS#8 y certificado digital, a formato PKCS#12

Parameters:

pKeyPath - Ruta al archivo de llave privada a exportar

certPath - Ruta al certificado digital a exportar

pKeyPassw - Clave de acceso para la llave privada

p12FmtPath - Archivo destino para el formato PKCS#12 a generar

acCertPath - Ruta al certificado digital de autoridad

alias - Alias para la llave privada

p12FmtPassw - Clave de acceso para el formato PKCS#12 a generar

Throws:

IOException, - Exception

java.lang.IllegalArgumentException

java.io.IOException

java.lang.Exception

- **getAliases**

Sintaxis: public void getAliases(java.lang.String provider)

throws java.lang.Exception

Descripción: Obtiene el conjunto de aliases para el proveedor o archivo PKCS#12 parametrizado

Parameters:

provider - Corresponde al nombre del proveedor PKCS#11 o ruta al archivo PKCS#12. Ver setDoPKCS12

Throws:

java.lang.Exception

- **getCertificateDataStr**

Sintaxis: public java.lang.String getCertificateDataStr()

Descripción: Obtiene los detalles de los certificados de un proveedor del que previamente se obtuvieron sus aliases.

- **getCertificateWhoseHashIs**

Sintaxis: public java.lang.String getCertificateWhoseHashIs(java.lang.String theStrHash)

Descripción: Obtiene el certificado digital cuya digestión es la parametrizada. Previamente debieron haberse obtenido los aliases para algún proveedor determinado

Parameters:

theStrHash - Cadena hexadecimal que representa la digestión del certificado a localizar

Returns:

Certificado digital correspondiente a la digestión hexadecimal parametrizada. Se encuentra codificado en base 64

- **getFilePath**

Sintaxis: public void getFilePath(java.lang.String title,
java.lang.String filter,
java.lang.String extension,
boolean read)

Descripción: Función para obtener la ruta a un archivo

Parameters:

title - Etiqueta a agregar al diálogo para búsqueda de archivos

filter - Nombre descriptivo para los archivos a filtrar

extension - Extensión asociada a los archivos a listar

read - Valor booleano que indica si la ruta es para un archivo a leer o a crear

---

- **listProviders**

Sintaxis: public java.lang.String listProviders()

Descripción: Obtiene la lista de proveedores configurados en el sistema y que soportan PKCS#11

Returns:

Lista de nombres de proveedores separada por el delimitador definido de acuerdo al método setSeparator

---

- **look4FF**

Sintaxis: public void look4FF()

Descripción: Indica que FF debe ser localizado y registrado

- **setCodePKCS1**

Sintaxis: public void setCodePKCS1()

Descripción: Indica la codificación de firmas PKCS#1 para el proceso de firma. Ver firma

---

- **setCodePKCS7**

Sintaxis: public void setCodePKCS7()

Descripción: Indica la codificación de mensajes criptográficos PKCS#7 para el proceso de firma. Ver firma

---

- **setDDMMYYYY**

Sintaxis: public void setDDMMYYYY(boolean value)

Descripción: Determina el formato de fecha a utilizar para el periodo de validez de los certificados localizados en los contenedores de llaves

Parameters:

value - Si se parametriza false, se establece que la fecha sea formateada como MM/dd/yyyy en vez de dd/MM/yyyy

---

- **setDigAlg**

Sintaxis: public void setDigAlg(java.lang.String value)

Descripción: Establece el algoritmo de digestión a utilizar para calcular la huella digital de los certificados del contenedor

Parameters:

value - Cadena que define un algoritmo de digestión. P. ej. "SHA1"

- **setDoCAPI**

Sintaxis: public void setDoCAPI(boolean value)

Descripción: Habilita el acceso al contenedor de certificados personales de Windows

Parameters:

value - Valor booleano que indica si se habilita o no la funcionalidad

---

- **setDoDetached**

Sintaxis: public void setDoDetached(boolean value)

Descripción: Habilita o deshabilita la codificación de mensajes firmados PKCS#7 sin contenido. Ver firma

Parameters:

value - Valor booleano que habilita o deshabilita la funcionalidad

---

- **setDoFile**

Sintaxis: public void setDoFile(boolean value)

Descripción: Establece que la información a firmar corresponde a la ruta y nombre de archivo a procesar. Ver setToSign

Parameters:

value - Valor booleano que indica si se habilita o no la funcionalidad

---

- **setDoJS**

Sintaxis: public void setDoJS(boolean value)

Descripción: Establece que además de manejar el conjunto de aliases como valor de retorno en una sola cadena, se populen objetos javascript con los valores encontrados.

Parameters:

value - Valor booleano que indica si se habilita o no la funcionalidad

---

- **setDoPKCS12**

Sintaxis: public void setDoPKCS12(boolean value)

Descripción: Indica que se va a firmar a partir de un par de llaves contenidas en un archivo codificado bajo el formato PKCS#12

Parameters:

value - Valor booleano que habilita o deshabilita la funcionalidad. Ver getAliases

---

- **setDoRegister**

Sintaxis: public void setDoRegister(boolean value)

Descripción: Especifica que debe registrarse el proveedor para acceso a certificados de firefox

Parameters:

value - Valor booleano que habilita o deshabilita la funcionalidad

---

- **setSeparator**

Sintaxis: public void setSeparator(java.lang.String value)

Descripción: Define una cadena que se utiliza como delimitador para los aliases encontrados por par de llaves para un proveedor PKCS#11 en particular o almacén PKCS#12

Parameters:

value - Valor del separador a utilizar

- **setSigAlg**

Sintaxis: public void setSigAlg(java.lang.String value)

Descripción: Define el algoritmo a utilizar en el proceso de generación de mensajes o firmas

Parameters:

value - Cadena que define el algoritmo de firma a aplicar, p. ej. SHA1withRSA

- **setSignerAlias**

Sintaxis: public void setSignerAlias(java.lang.String value)

Descripción: Indica el alias del usuario que va a firmar. Ver getAliases

Parameters:

value - Alias del usuario firmante

- **setToSign**

Sintaxis: public void setToSign(java.lang.String value)

Descripción: Define la información a firmar, ya sea una cadena o ruta y nombre de archivo

Parameters:

value - Cadena o ruta y nombre de archivo

- **setToSignIsDigest**

Sintaxis: public void setToSignIsDigest(boolean value)

Descripción: Especifica que la cadena a firmar corresponde a la representación hexadecimal de una digestión previamente calculada. Ver setToSign

Parameters:

value - Valor booleano que habilita o deshabilita la funcionalidad

---

- **setVerboseOFF**

Sintaxis: public void setVerboseOFF()

Descripción: Deshabilita el despliegue de mensajes de depuración a la consola

---

- **setVerboseON**

Sintaxis: public void setVerboseON()

Descripción: Habilita el despliegue de mensajes de depuración a la consola

---

- **sign**

Sintaxis: public void sign()

throws java.lang.Exception

Descripción: Genera la firma PKCS#1 o mensaje criptográfico PKCS#7 para el archivo o cadena de texto definido

Throws:

java.lang.Exception - El algoritmo de firma definido no es consistente con la digestión parametrizada

3.3 Ejemplo

La integración de las partes antes descritas para la inclusión del applet en una página HTML se muestra en el siguiente ejemplo:

```
[210] <html>
[211] <head>
[212] <title>SeguriSign by SeguriData, firma con archivo PKCS#12</title>
[213] </head>
[214] <body onLoad="initValues()">
[215] <table width="55%">
[216] <tr>
[217]   <td><p align=left><img src="segurisign.jpg" width="200" height="50" alt="" border="0"></p></td>
[218]   <td><p align=right><img src="sgdata.jpg" width="200" height="42" alt="" border="0"></p></td>
[219] </tr>
[220] <tr></tr>
[221] </table>
[222]
[223] <h3>SeguriSign Applet v8.6 powered by SeguriData</h3>
[224]
[225] <form name="theForm">
[226] <table>
[227]  <tr>
[228]  <td><h4>PKCS#12</h4></td>
[229]  <td><p align="left"><input type="text" id="p12File" name="p12File" size="45"><input type="button" id="p12FileButton"
         name="p12FileButton" value="Browse ..." onClick="selectTheFile(document.getElementById('p12File'), 'Archivo
         PKCS12', 'Archivo PKCS12(*.p12)', 'p12', true)"></p></td>
[230]  </tr>
[231]  <tr>
[232]  <td><h4>Llaves</h4></td>
[233]  <td><p align="left"><select size="1" name="theAliases" id="theAliases"></select></p></td>
[234]  </tr>
[235]  <tr>
[236]  <td> </td>
[237]  <td> </td>
[238]  </tr>
[239]  <tr>
[240]  <td><h4>Asegurar</h4></td>
[241]  <td></td>
[242]  </tr>
[243]  <tr>
[244]  <td><input name="myradio" id="myradio" type="radio"  onClick="f2(0)" > Archivo </td>
[245]  <td ><p align="left"><input type="text" id=FILE_NAME name=FILE_NAME size="45"><input type="button"
         id="file2SignButton" name="file2SignButton" value="Browse ..."
         onClick="selectTheFile(document.getElementById('FILE_NAME'), 'Archivo a firmar', 'Todos los archivos(*.*)', '*',
         true)"></p></td>
[246]  </tr>
[247]  <tr>
[248]  <td><input name="myradio" id="myradio" type="radio"  onClick="f2(1)" > Mensaje </td>
[249]  <td><textarea id=TEXT_TO_SIGN name=TEXT_TO_SIGN rows="4" cols="65"  ></textarea></td>
[250]  </tr>
[251]  <tr>
[252]  <td> </td>
```

```
[253]    <td> </td>
[254]   </tr>
[255]   <tr>
[256]     <td><h4>Algoritmo</h4></td>
[257]     <td>
[258]       <p align="left">
[259]         <select size="1" name="signAlgorithms" id="signAlgorithms">
[260]         <option value="SHA1withRSA" selected=true>SHA1withRSA</option>
[261]         </select>
[262]       </p>
[263]     </td>
[264]   </tr>
[265]   <tr>
[266]    <td> </td>
[267]    <td> </td>
[268]   </tr>
[269]   <tr>
[270]    <td><h4>Destinatario</h4></td>
[271]    <td><textarea name="destinatarios" id="destinatarios" rows="4" cols="65"></textarea></td>
[272]   </tr>
[273]   <tr>
[274]    <td> </td>
[275]    <td> </td>
[276]   </tr>
[277]   <tr>
[278]    <td><h4>Resultado</h4></td>
[279]    <td><textarea name="p7" id="p7" rows="4" cols="65"></textarea></td>
[280]   </tr>
[281]   <tr>
[282]    <td> </td>
[283]    <td> </td>
[284]   </tr>
[285]   <tr>
[286]    <td> </td>
[287]    <td> </td>
[288]   </tr>
[289]   <tr>
[290]    <td></td>
[291]    <td><input type="checkbox" name="doPKCS1" id="doPKCS1">PKCS#1    <input type="checkbox" name="doEnvelope"
           id="doEnvelope" onChange="activa()">Ensobreta</td>
[292]   </tr>
[293]   <tr>
[294]    <td> </td>
[295]    <td> </td>
[296]   </tr>
[297]   <tr>
[298]    <td></td>
[299]    <td><input type="button" id="elBoton" onClick="firma()" value="Asegurar"></td>
[300]   </tr>
[301]  </table>
```

```
[302] </form>
[303]
[304] <script language="Javascript">
[305] var theAppSigner
[306] var theAliases = document.getElementById("theAliases")
[307]
[308] var p7 = document.getElementById("p7")
[309] var opcion=0
[310]
[311] function initValues()
[312] {
[313] theAppSigner = document.getElementById("theSigner")
[314] theAppSigner.setDoJS(true)
[315] theAppSigner.setDoPKCS12(true)
[316] theAliases.options.add(new Option("No hay entradas ..."))
[317] document.getElementById("destinatarios").disabled = true
[318] }
[319]
[320] function f2(x)
[321] {
[322] if(x == 0)
[323] {
[324] document.getElementById("TEXT_TO_SIGN").disabled = true
[325] document.getElementById("FILE_NAME").disabled = false
[326] opcion=1
[327] }
[328] if(x == 1)
[329] {
[330] document.getElementById("FILE_NAME").disabled = true
[331] document.getElementById("TEXT_TO_SIGN").disabled = false
[332] opcion=2
[333] }
[334] }
[335]
[336] function activa()
[337] {
[338] if (document.getElementById("doEnvelope").checked)
[339]    document.getElementById("destinatarios").disabled = false
[340] else
[341]    document.getElementById("destinatarios").disabled = true
[342] }
[343]
[344] function procesa_excepcion(exc)
[345] {
[346] msg = exc
[347]
[348] if (exc.indexOf("java.io.IOException: DER input, Integer tag error") != -1 || exc.indexOf("java.io.IOException:
       toDerInputStream") != -1)
[349]    msg = "El archivo PKCS12 está corrupto o no está codificado bajo dicho formato";
[350] else
```

```
[351] if (exc.indexOf("java.io.IOException: failed to decrypt safe contents entry") != -1 || exc.indexOf("Given final block not properly
        padded") != -1)
[352] msg = "La contraseña del PKCS#12 no es correcta, intente de nuevo"
[353] else
[354] if (exc.indexOf("java.security.NoSuchAlgorithmException") != -1)
[355] msg = "El PKCS#12 se encuentra cifrado con algún algoritmo no soportado por los proveedores existentes"
[356] else
[357]   if (exc.indexOf("CKR_PIN_INCORRECT") != -1 || exc.indexOf("CKR_PIN_LEN_RANGE") != -1)
[358]     msg = "No fue posible tener acceso al dispositivo debido a un NIP incorrecto"
[359] else
[360]   if (exc.indexOf("CKR_FUNCTION_CANCELED") != -1)
[361]     msg = "La operación fue cancelada por el usuario"
[362] else
[363]   if (exc.indexOf("javax.crypto.BadPaddingException: Given final block not properly padded") != -1)
[364]     msg = "La contraseña del PKCS#12 no es correcta, intente de nuevo"
[365] else
[366]   if (exc.indexOf("by zero") != -1)
[367]     msg = "La contraseña del PKCS#12 no es correcta, intente de nuevo"
[368] else
[369]   if (exc.indexOf("java.security.cert.CertificateException") != -1 || exc.indexOf("java.io.IOException: DerInputStream") != -1)
[370]     msg = "El certificado digital parametrizado es incorrecto"
[371] else
[372]   if (exc.indexOf("file does not exist") != -1)
[373]     msg = "La ruta de archivo recibida no es correcta"
[374]   else
[375]   if (exc.indexOf("java.io.EOFException") != -1)
[376]     msg = "Los parámetros recibidos no son correctos"
[377]   else
[378]   if (exc.indexOf("PKCS11 not found") != -1)
[379]     msg = "La firma digital no fue bien integrada a esta aplicación o FF no se encuentra instalado"
[380]
[381] alert(msg)
[382] }
[383]
[384] function selectTheFile(obj, title, filter, ext, read) {
[385] theAppSigner.getFilePath(title, filter, ext, read)
[386]
[387] if (theAppSigner.isSuccess()) {
[388] var thePath = theAppSigner.getSelectedFileAndPath()
[389]
[390] if (thePath != "")
[391] obj.value = thePath
[392] } else {
[393]    alert(theAppSigner.getErrorMessage())
[394]    return
[395]    }
[396]
[397] var aliasesR
[398]
[399] if (thePath != "" && ext == "p12")
```

```
[400] {
[401] try
[402] {
[403] theAliases.options.length = 0
[404] theAppSigner.getAliases(thePath)
[405]
[406] aliasesR = theAppSigner.getStoreAliases()
[407]
[408] if (aliasesR.indexOf("Error") == 0)
[409] {
[410] procesa_excepcion(aliasesR)
[411] obj.value = ""
[412] return
[413] }
[414]
[415] } catch(err)
[416] {
[417] procesa_excepcion(err.description == null ? err.message : err.description)
[418] document.theForm.reset()
[419] }
[420] }
[421] }
[422]
[423] function firma()
[424] {
[425] var p7var = ""
[426] var current = null
[427]
[428] try
[429] {
[430] if (document.getElementById("p12File").value == "")
[431] {
[432] alert("Debe seleccionar el par de llaves para proceder")
[433] return
[434] }
[435]
[436] if (document.getElementById("doEnvelope").checked)
[437] {
[438] if (document.getElementById("destinatarios").value == "")
[439] {
[440] alert("No ha proporcionado el destinatario para el mensaje ensobretado")
[441] return
[442] }
[443] }
[444]
[445] if (opcion == 1)
[446] {
[447] current = document.getElementById("FILE_NAME")
[448]
[449] if (current.value != "")
```

```
[450]  {
[451]  theAppSigner.setDoFile(true)
[452]  theAppSigner.setToSign(current.value)
[453]  }
[454]  else
[455]  {
[456]  alert("Por favor seleccione el archivo que desea firmar.")
[457]  current.focus()
[458]  return
[459]  }
[460]  }
[461]  else if (opcion == 2)
[462]  {
[463]  current = document.getElementById("TEXT_TO_SIGN")
[464]
[465]  if (current.value != "")
[466]  {
[467]  theAppSigner.setDoFile(false)
[468]  theAppSigner.setToSign(current.value)
[469]  }
[470]  else
[471]  {
[472]  alert("Por favor ingrese el mensaje que desea firmar")
[473]  return
[474]  }
[475]  }
[476]  else
[477]  {
[478]  alert("Por favor elija la opcion que desea firmar.")
[479]  return
[480]  }
[481]
[482]  theAppSigner.setSigAlg(document.getElementById("signAlgorithms").value)
[483]
[484]  theAppSigner.setSignerAlias(theAliases.value)
[485]
[486]  if (document.getElementById("doPKCS1").checked && !document.getElementById("doEnvelope").checked)
[487]  theAppSigner.setCodePKCS1()
[488]  else
[489]  theAppSigner.setCodePKCS7()
[490]
[491]  if (document.getElementById("doEnvelope").checked) {
[492]     theAppSigner.envelope(document.getElementById("destinatarios").value)
[493]     }
[494]  else {
[495]  theAppSigner.sign()
[496]    }
[497]
[498]    if (theAppSigner.isSuccess())
[499]       p7var = theAppSigner.getSignature()
```

```
[500]    else
[501]       p7var = theAppSigner.getErrorMessage()
[502]
[503]    if (p7var.indexOf("Error") == 0)
[504]    procesa_excepcion(p7var)
[505]    else
[506]  p7.value = p7var
[507]  }catch(err) {
[508]  procesa_excepcion(err.description == null ? err.message : err.description)
[509]  }
[510]  }
[511]
[512]  </script>
[513]
[514]  </body>
[515]  <applet code="seguridata.segurisign.Signer" id="theSigner" name="theSigner" scriptable="false" mayscript="false"
          height="0" width="0" style="xdisplay: none; width:0; height:0; padding:0; margin:0;" archive="SeguriSignClient.jar">
[516]  </applet>
[517]  </html>
```

Puede copiar el ejemplo anterior en un archivo HTML y guardarlo en una carpeta que contenga también el archivo *SeguriSignClient.jar* de tal forma que pueda ejecutar el ejemplo.