

SeguriSign para desarrolladores



I.S.C. Henry Garcia
hgarcia@seguridata.com

¿ Qué es SeguriSign ?

Una solución de firma electrónica para,

- Generación, validación y registro de mensajes criptográficos firmados o firmados y ensobretados
- Gestión y administración de evidencias criptográficas asociadas a una transacción de firma electrónica

¿ Qué aplicación práctica tiene ?

- Habilitar firma electrónica a toda aplicación propietaria que requiera integrarla en cualquier parte de sus procesos
- Proporcionar pistas de auditoría o evidencias criptográficas generadas de acuerdo a estándares y algoritmos criptográficos seguros e internacionales.

Diagrama Básico de Bloques

SeguriSign 8.0

API

Servidor

Módulos de
Firma y
Sobre

Java

Servicio Windows
Proceso Linux

Librerías
dinámicas para
VS: VB, C#:

Alta DB

Visor: CMS

Applet

Aplicación p/ Consulta
de Transacciones
(WEB)

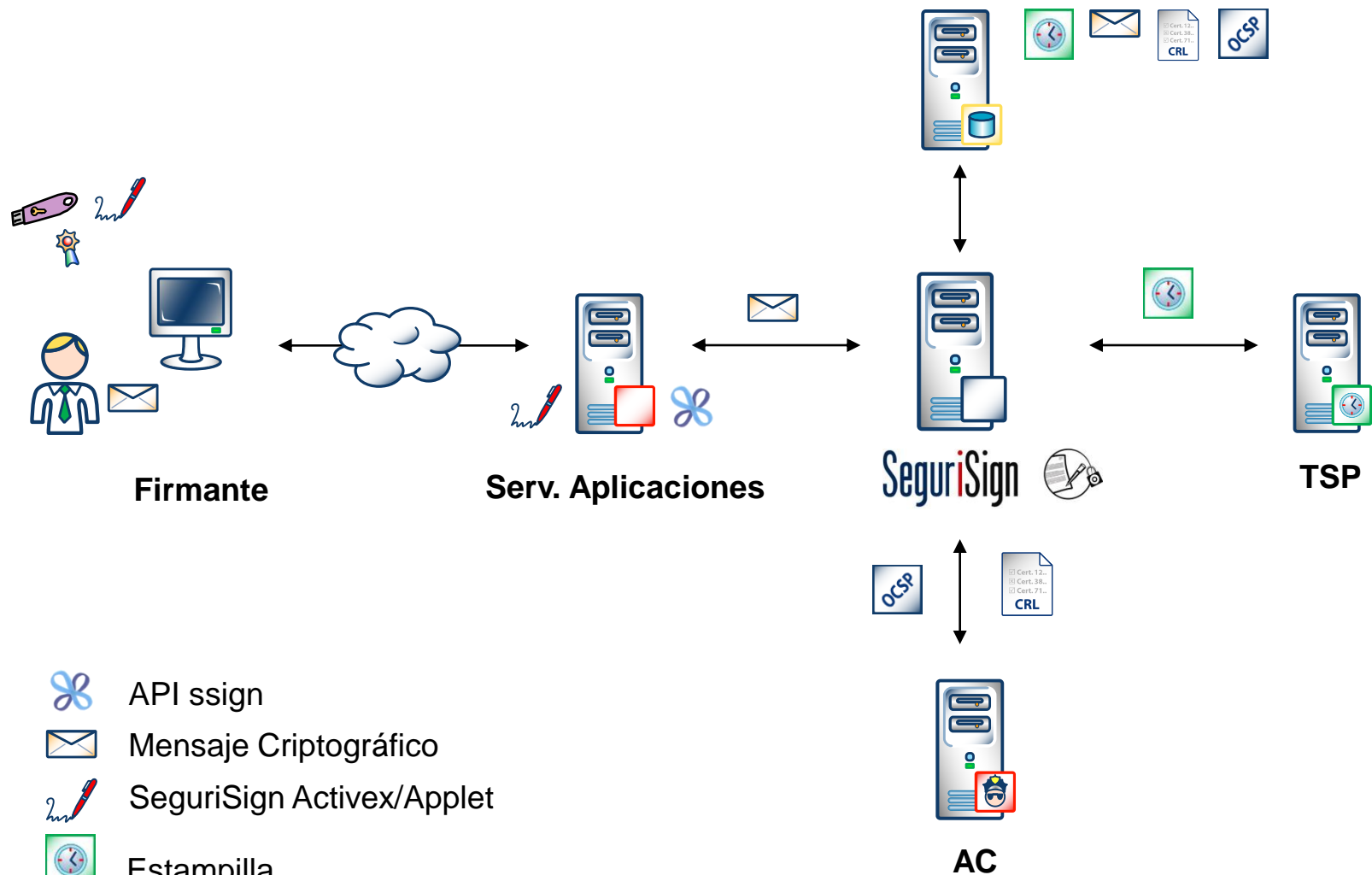
Ejemplos

Configurador
(WEB)

ActiveX

Motor Criptográfico: SeguriLib v3.2

Modelo conceptual

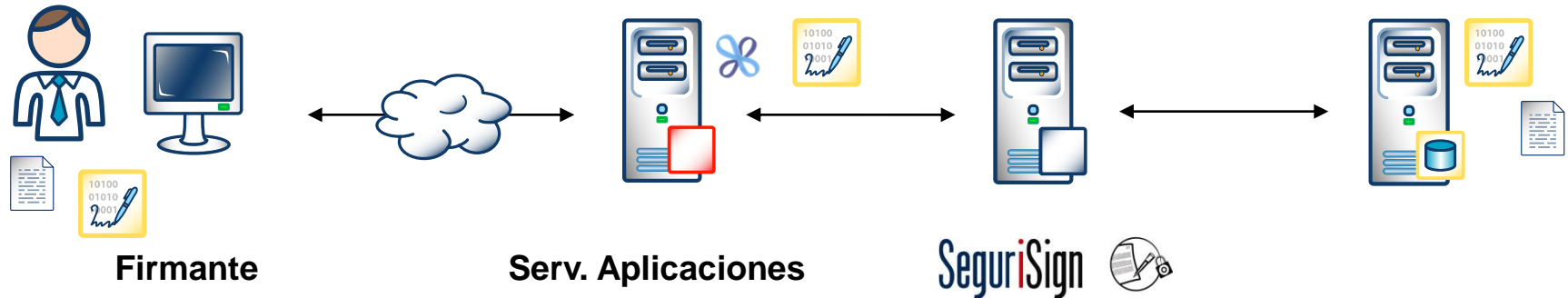


Modelo Conceptual

1. El aplicativo solicita al usuario, firmar o ensobretar un mensaje; si SeguriSign ActiveX no se encuentra instalado en el equipo del usuario, es descargado desde el Servidor de Aplicaciones. SeguriSign Applet es descargado cada vez que se aplica una operación criptográfica
2. El mensaje generado es remitido al aplicativo y recuperado por éste
3. Para procesar el mensaje se envía a SeguriSign Servidor, a través de la API
4. SeguriSign Servidor analiza el mensaje criptográfico y solicita una stampa de tiempo cuando éste procede de una Autoridad configurada en su entorno: la stampa ampara la firma del mensaje recibido
5. Con respecto a la stampa de tiempo se analiza el certificado digital: vigencia
6. Se verifica el estatus de revocación del certificado digital, de acuerdo a los parámetros definidos para el emisor en cuestión
7. La revocación se evalúa basándose en el instante de tiempo de la estampilla recibida (si aplica)
 - Se solicita el estatus de revocación a la Autoridad: OCSP (si aplica)
 - Se verifica el estatus de revocación: CRL's (si aplica)
8. El mensaje criptográfico y sus evidencias son dados de alta en base de datos
9. Se reporta el estatus obtenido al aplicativo (API)

Tipos de procesos de firma

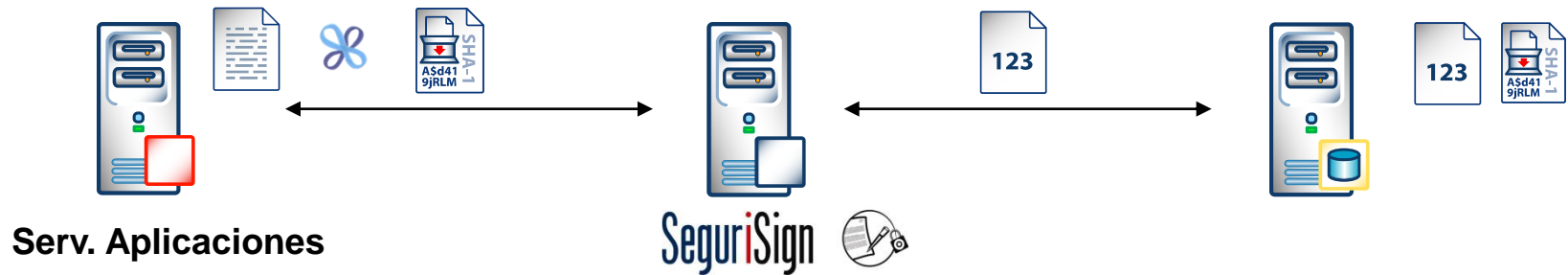
Proceso de firma unilateral



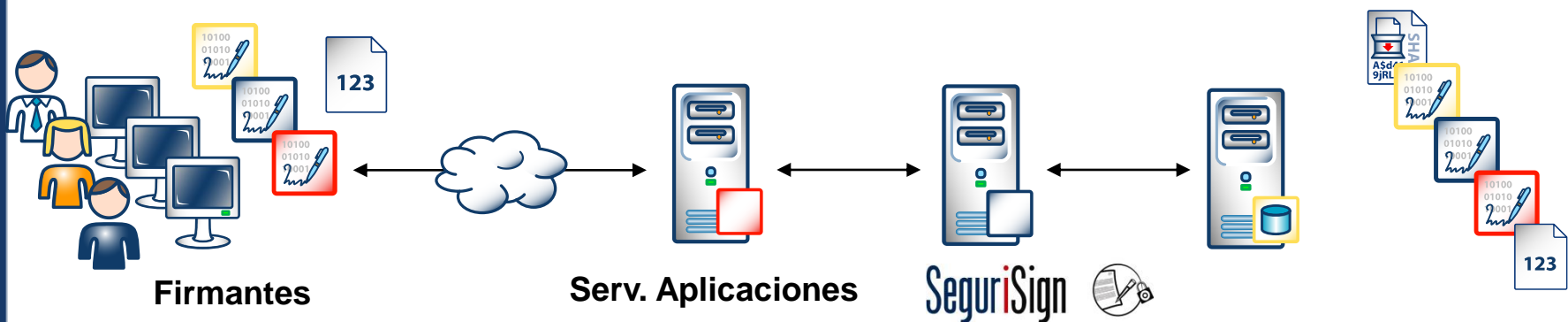
- Se genera un mensaje firmado, donde participa únicamente un firmante
- El documento del mensaje firmado unilateralmente, puede o no ser el mismo para otros usuarios
- El documento puede o no formar parte del mensaje criptográfico, pero siempre es almacenado en la base de datos de SeguriSign

Proceso de firma multilateral

- Es iniciado por el aplicativo a partir del documento a firmar

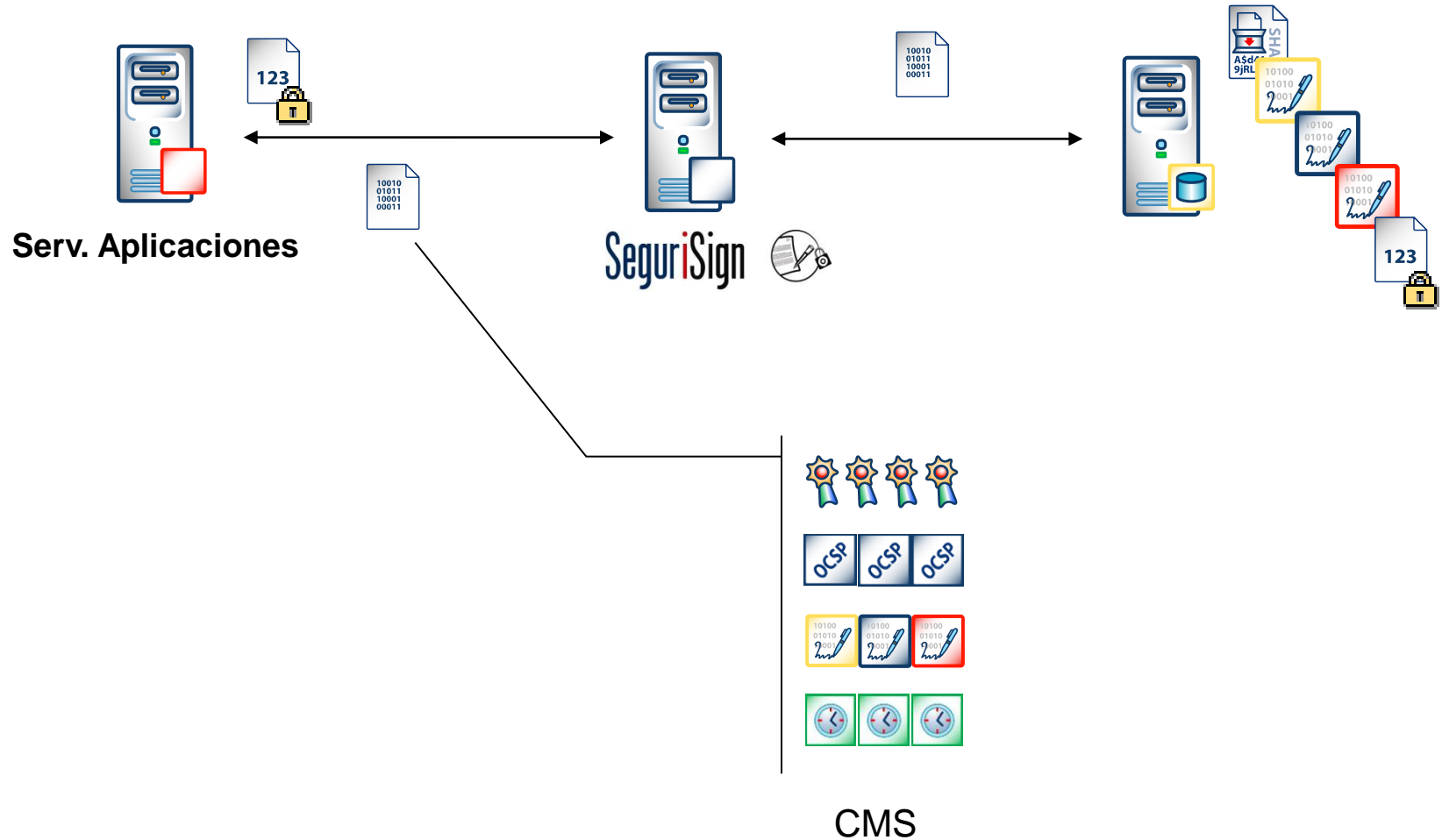


- El mensaje firmado es de un solo firmante, pero se autentican varias firmas con referencia al ID resultante del proceso de alta del documento



123 ID de proceso Archivo asegurado API Digestión Firmas


- El aplicativo solicita el estado actual del proceso de firma multilateral o el cierre del mismo: CMS



- El documento a asegurar,
 1. debe ser 'dado de alta' como punto inicial del proceso
 2. necesariamente es el mismo para todos los usuarios
 3. nunca reside en la base de datos del producto
- El resultado final es un CMS conformado con tantos firmantes como hayan participado en el proceso, mas las evidencias criptográficas y certificados involucrados
- El CMS final puede ser visualizado en el equipo del firmante

Validación de transacciones de firma electrónica

Criterios para determinar el estatus de una transacción de autenticación de firma

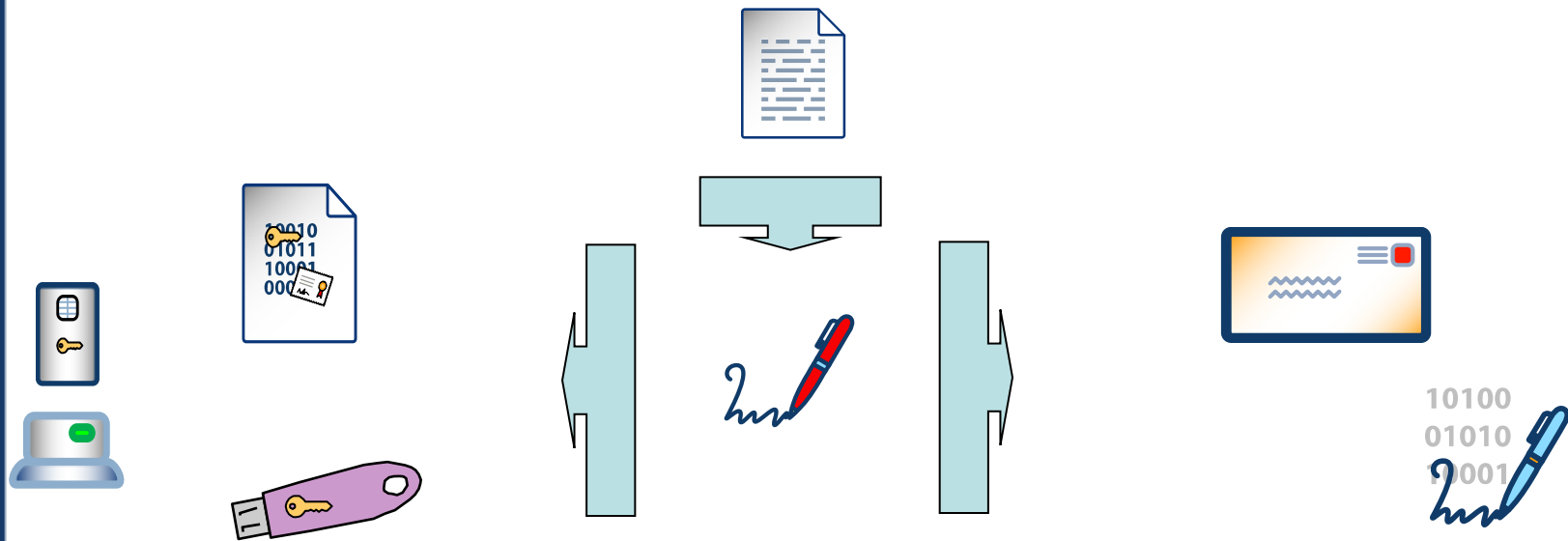
- Integridad del mensaje criptográfico
- La transacción es confidencial (Aplica sólo en Sobre Digital)
- El Certificado del Firmante:
 - Procede de un emisor de confianza configurado (Auténtico)
 - Debe ser vigente 
 - Es íntegro
 - No está revocado: CRL, OCSP, IES
- El mensaje criptográfico contiene una Firma Digital Auténtica

De este modo se asegura la integridad, confidencialidad, autenticidad y no-repudio de la transacción



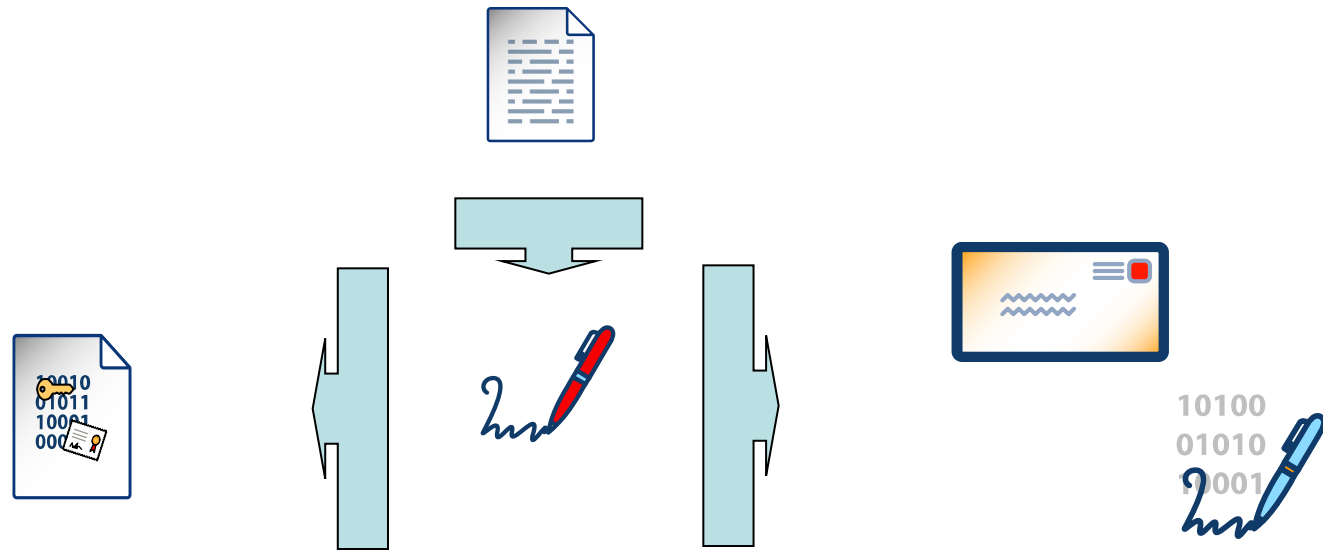
Módulos para generación y envío de
firmas o sobres digitales

SeguriSign Cliente (ActiveX)



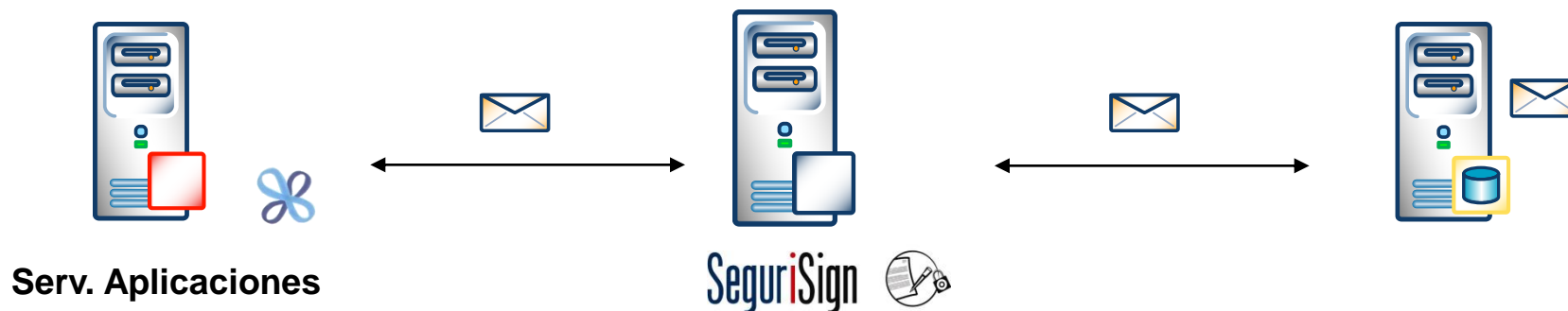
- 1.- El par de llaves del firmante reside en un token que puede ser físico o lógico
- 2.- SeguriSign ActiveX aplica operaciones criptográficas sobre un mensaje plano y obtiene un mensaje criptográfico: firmado o firmado y ensobretado

SeguriSign Cliente (Applet)



- 1.- El par de llaves del firmante reside en un token lógico
- 2.- SeguriSign Applet aplica operaciones criptográficas sobre un mensaje plano y obtiene un mensaje criptográfico: firmado o firmado y ensobretado

API SeguriSign



Mensaje Criptográfico



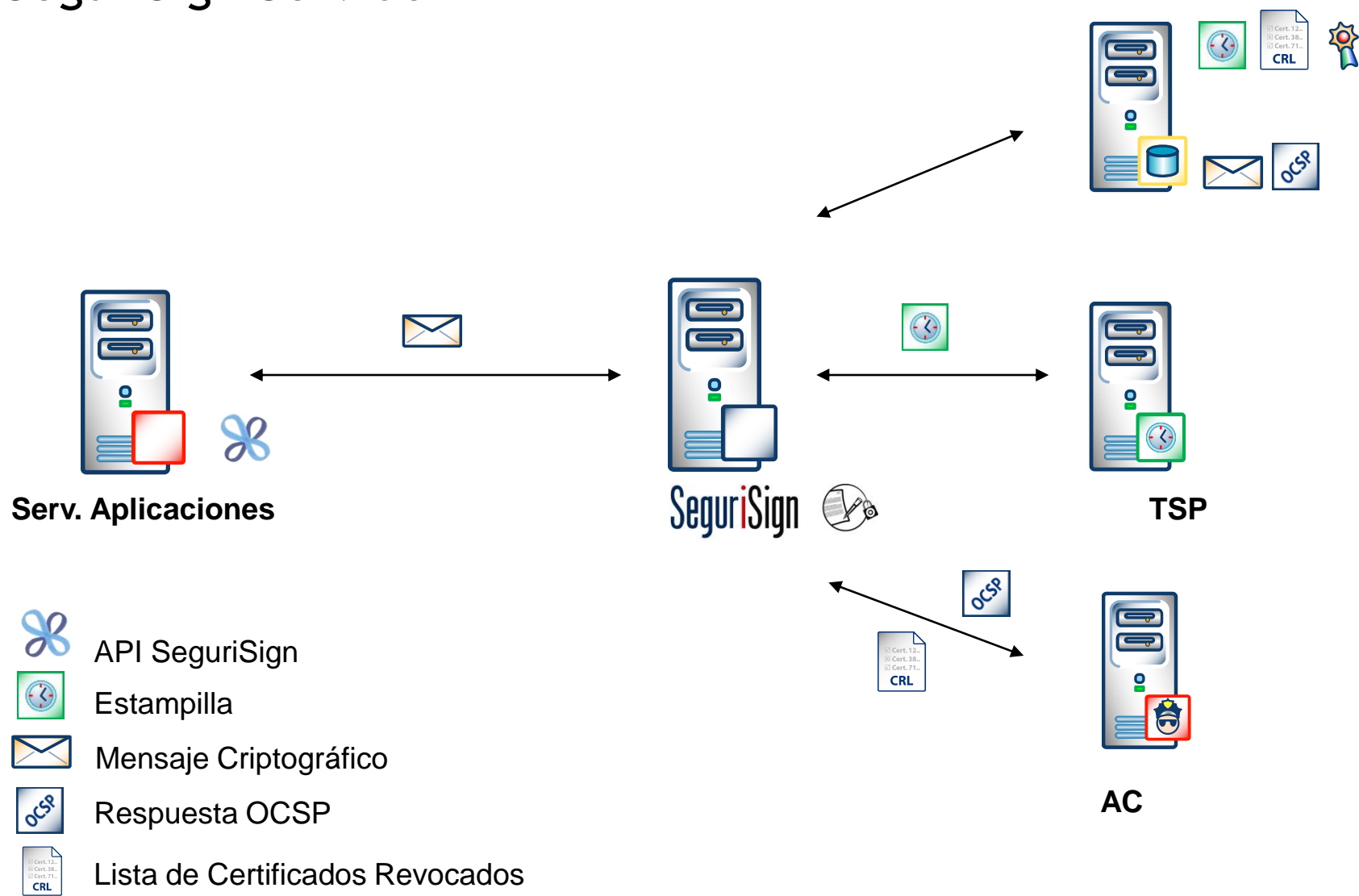
API SeguriSign

1.- El aplicativo que solicita al firmante aplicar su firma a una transacción, recupera el mensaje criptográfico resultante;

2.- SeguriSign API es instanciada para obtener un estatus del mensaje ante el Servidor SeguriSign

Módulo para autenticación de firmas

SeguriSign Servidor



Desarrollo de aplicaciones con SeguriSign

Prerrequisitos para uso de SeguriSign ActiveX en navegadores ActiveX enabled

*Plataformas

Windows 98 SE, Me, 2000, NT, XP, 2003, 2008, Vista, W7

*Internet Explorer 5.0+

- La propiedad Secuencias de Comandos ActiveX debe estar habilitada para la Zona de Seguridad Internet del navegador o la que aplique de acuerdo a donde resida la aplicación Web y de donde se acceda.
- La propiedad descargar los controles firmados para ActiveX debe estar habilitada o en modo pedir datos para la Zona de Seguridad Internet del navegador o la que aplique de acuerdo a donde resida la aplicación Web y de donde se acceda.
- La propiedad ejecutar los controles y complementos para ActiveX debe estar habilitada en la Zona de Seguridad Internet del navegador o la que aplique de acuerdo a donde resida la aplicación Web y de donde se acceda.
- La propiedad activar la secuencia de comandos de los controles ActiveX marcados como seguros debe estar habilitada en la Zona de Seguridad de Internet o la que aplique de acuerdo a donde resida la aplicación Web y de donde se acceda.
- Los controles ActiveX no deben estar restringidos a nivel dominio para el perfil de Windows del usuario que generará su par de llaves.
- Para Windows Vista es necesario dar de alta como sitio de confianza, el sitio que incluye SeguriSign ActiveX

*Netscape Navigator 7.1 (AX) / Mozilla FireFox (Plugin para ejecución de ActiveX)

- Aplican algunas consideraciones para Internet Explorer 5.0+; además el archivo de configuración **activex.js** ubicado bajo el directorio donde instaló Netscape 7.1/defaults/pref/ deberá contener la línea:

```
pref("capability.policy.default.ClassID.CID91006BE5-1C6B-11D5-BCD5-0050DA23D78F", "AllAccess");
```

para que sea posible la ejecución del componente.

Cómo incluir SeguriSign ActiveX en una página HTML

```
<html>

<head>

<title>SeguriData, SeguriSign - Firma Digital.</title>

<OBJECT
  classid="clsid:91006BE5-1C6B-11D5-BCD5-0050DA23D78F"
  codebase="SeguriSignv8.6.1.cab"
  id="SeguriSign"
  width="0"
  height="0">
</OBJECT>

</head>

.....

.....

.....

</html>
```

El classid define el identificador del objeto, tal como será registrado en el equipo cliente.

El codebase indica el directorio o ruta relativa al Servidor Web bajo la que está disponible el componente para su instalación en línea.

La propiedad id indica el alias con el que se manipulará el objeto en las funciones vbscript o javascript definidas.



Los errores que se presentan al utilizar SeguriSign ActiveX tienen que ver con permisos del usuario en el Sistema, una mala integración del producto o una mala operación de la aplicación por parte del usuario final.

SeguriSign ActiveX - Propiedades

Propiedad	Tipo	Descripción
b64Enters	[in]	Indica al objeto SeguriSign que al disponer en base 64 la información genere o no líneas de 64 caracteres separadas por enters. Al asignar 1 a esta propiedad se indica que se incluyan los saltos de línea. El valor por omisión es que no se generen saltos de línea para el formato base 64
CA	[in]	Criterio de búsqueda alternativo para localizar las llaves del Firmante. Su valor debe ser el "Common Name" de la Autoridad Certificadora que emitió el Certificado del Firmante. Se utiliza, para de algún modo; contrarrestar la imprecisión de localización de la propiedad Serial. De esta manera se localiza un certificado con el Número de Serie indicado por Serial y emitido por la Autoridad cuyo "Common Name" es el indicado por esta propiedad
Compress	[in]	Permite indicar que la información a firmar y/o ensobretar sea comprimida antes de aplicar las operaciones criptográficas. Por omisión SeguriSign no se comprime la información antes de ser procesada. De desear aplicar compresión deberá asignarse 1 a esta propiedad.
Empty	[in]	Indica si el Mensaje Criptográfico firmado a generar contendrá o no la información asegurada, es decir, que se conformará un mensaje criptográfico cuyo contenido asegurado no está asociado o incluido en el mismo. Por default todos los mensajes firmados generados por SeguriSign incluyen la información firmada, a menos que se exprese lo contrario asignando el valor 1 a esta propiedad
EncAlg2Envelope	[in]	Por default los sobres digitales generados por SeguriSign se conforman utilizando RC4 como algoritmo de encriptación simétrica. Para utilizar DESEDE3CBC como otra alternativa asigne el valor 0 a esta propiedad
Ext	[in]	Se utiliza para definir la extensión de archivo por default para el selector de archivos a desplegar con la ejecución de los métodos GetFile y SetFile. Por ejemplo, la cadena ".key" establecería dicha extensión como default para el selector de archivos.
File	[out]	Contiene el nombre del archivo seleccionado al finalizar con éxito un proceso de Firma o Ensobretado de Archivos
Filter	[in]	Da como entrada el filtro a aplicar por el selector de archivos a desplegar con la ejecución de los métodos GetFile y SetFile. Por ejemplo la cadena "Archivo de Llave Privada (*.key) *.key" indica que se filtrarán aquellos archivos con extensión ".key" y cuya descripción sería Archivo de Llave Privada
High	[in]	Cuando se le asigna como valor 0, se configura que no se desea mostrar al usuario el aviso de creación de elemento protegido. Por consiguiente la advertencia de acceso a dicho elemento, también quedará inhabilitada

SeguriSign ActiveX - Propiedades (II)

Propiedad	Tipo	Descripción
InFileName	[in]	Deberá contener la ruta al archivo a Firmar y Ensobretar
Lang	[in]	Indica el Idioma de la interfaz y de los mensajes de error no dependientes de Sistema Operativo. El valor por default es 1 o Español. Si a esta propiedad se le asigna como valor 0, los mensajes de la interfaz y de error serán en inglés
OutFileName	[in]	Indica la ruta y nombre de archivo para un mensaje criptográfico a generar en archivo en vez del mensaje criptográfico en base 64 como valor de retorno. El mensaje Criptográfico a archivo se generará en binario
RejectExpiredCerts	[in]	Permite indicar que se desea generar una firma o sobre digitales sin que se tome en cuenta el periodo de validez del certificado digital, es decir, que aunque éste esté expirado se realice el proceso
Serial	[in]	Criterio de búsqueda alternativo a ThumbPrint para localizar las llaves del Firmante. Se parametriza como una cadena hexadecimal que representa el Número de Serie del Certificado del Firmante
Status	[out]	Devuelve el estatus de Ejecución, donde 2000 indica un proceso exitoso y 0 que el usuario canceló el proceso de Firma o Encriptación. Cualquier valor distinto a los anteriores corresponde a un error de ejecución
Thumbprint	[in]	Huella Digital del Certificado de usuario a utilizar para la firma. Se parametriza como una cadena hexadecimal que representa la digestión MD5 o SHA1 del Certificado. Es útil para integrar firmas con SeguriSign en modo faceless o sin interfaz
UserKeys	[in]	Recibe un string que contiene en Base 64, la codificación del PKCS12 del par de llaves del usuario a utilizar en el proceso de firma digital
UserPwd	[in]	Recibe la contraseña para descifrar el par de llaves del usuario parametrizadas en UserKeys
Viewer	[in]	Indica que se desea o no desplegar como parte de la interfaz, un visor de textos para los archivos o transacciones, a fin de presentarlos al usuario antes de su firma o firma y ensobretado. Deberá tomar como valor 0 o 1. El valor por default es 1 o habilitado
Warning	[in]	Despliega un aviso al usuario de que se va a conformar una firma digital utilizando una digestión previamente calculada de un documento y si desea o no realizar el proceso
Who	[in]	Se utiliza para configurar el nombre del Servidor que solicita la Firma Digital como parte de la interfaz de Firma

SeguriSign ActiveX - Propiedades (III)

Propiedad	Tipo	Descripción
Filename4TextInput	[in]	<p>Indica que cuando se generen mensajes firmados a partir de cadenas parametrizadas y se ha indicado previa compresión de los datos asegurados, debe utilizarse el valor de esta propiedad como "nombre de archivo" para la cadena de texto parametrizada.</p> <p>Sintaxis SeguriSign.Filename4TextInput = "Detalle.xml"</p> <p>Tipo String - [Entrada]</p> <p>Notas Esta propiedad debe utilizarse cuando se generen mensajes firmados a partir de cadenas de texto y donde se aplique compresión de los datos asegurados</p>
b64Columns	[in]	<p>Descripción Define el tamaño en caracteres para la línea resultante base64 (Será múltiplo de 4 y no mayor a 1024 bytes) y sólo aplica cuando la propiedad OutFileName ha sido definida con un valor.</p> <p>Sintaxis SeguriSign.b64Columns = 64</p> <p>Tipo Decimal - [Entrada]</p>
b64Output	[in]	<p>Descripción Indica que cuando el mensaje criptográfico se destina a archivo, la salida será en base 64 (valor 1); sólo aplica cuando la propiedad OutFileName ha sido definida con un valor</p> <p>Sintaxis SeguriSign.b64Output = 1</p> <p>Tipo Decimal - [Entrada]</p>

SeguriSign ActiveX - Métodos

Método	Descripción
String SeguriSign.Firma(<i>Arg1</i>)	<p>Método que Firma Digitalmente un texto recibido como parámetro o un Archivo.</p> <p>Si el valor de <i>Arg1</i> es una cadena vacía "" se indica que se desea firmar un archivo a seleccionar desde la interfaz de firma.</p> <p>Si el valor de <i>Arg1</i> es distinto de "", como por ejemplo "Fírmame" se firmará digitalmente el texto recibido.</p> <p>Su valor de retorno, si SeguriSign.status es igual a 2000, corresponde a un String que contiene un mensaje criptográfico codificado según el estándar PKCS7</p>
String SeguriSign.Sobre(<i>Arg1</i> , <i>Arg2</i>)	<p>Método que firma y ensobrete un texto o archivo.</p> <p><i>Arg1</i> corresponde a la información a asegurar. Si su valor es "" se procesará un archivo a seleccionar. Si tiene valor distinto de "" se firmará y ensobretará el string parametrizado.</p> <p><i>Arg2</i> corresponde al certificado del destinatario o entidad a la que se le ensobretará la información. Si su valor es "" el certificado podrá seleccionarse entre los existentes en el Windows, desde la interfaz de firma. Por el contrario, cuando tiene valor; éste deberá ser el Certificado del destinatario en formato texto (Base64)</p> <p>Su valor de retorno, si SeguriSign.status vale 2000, es una cadena que corresponde a un mensaje criptográfico codificado según el estándar PKCS7.</p>

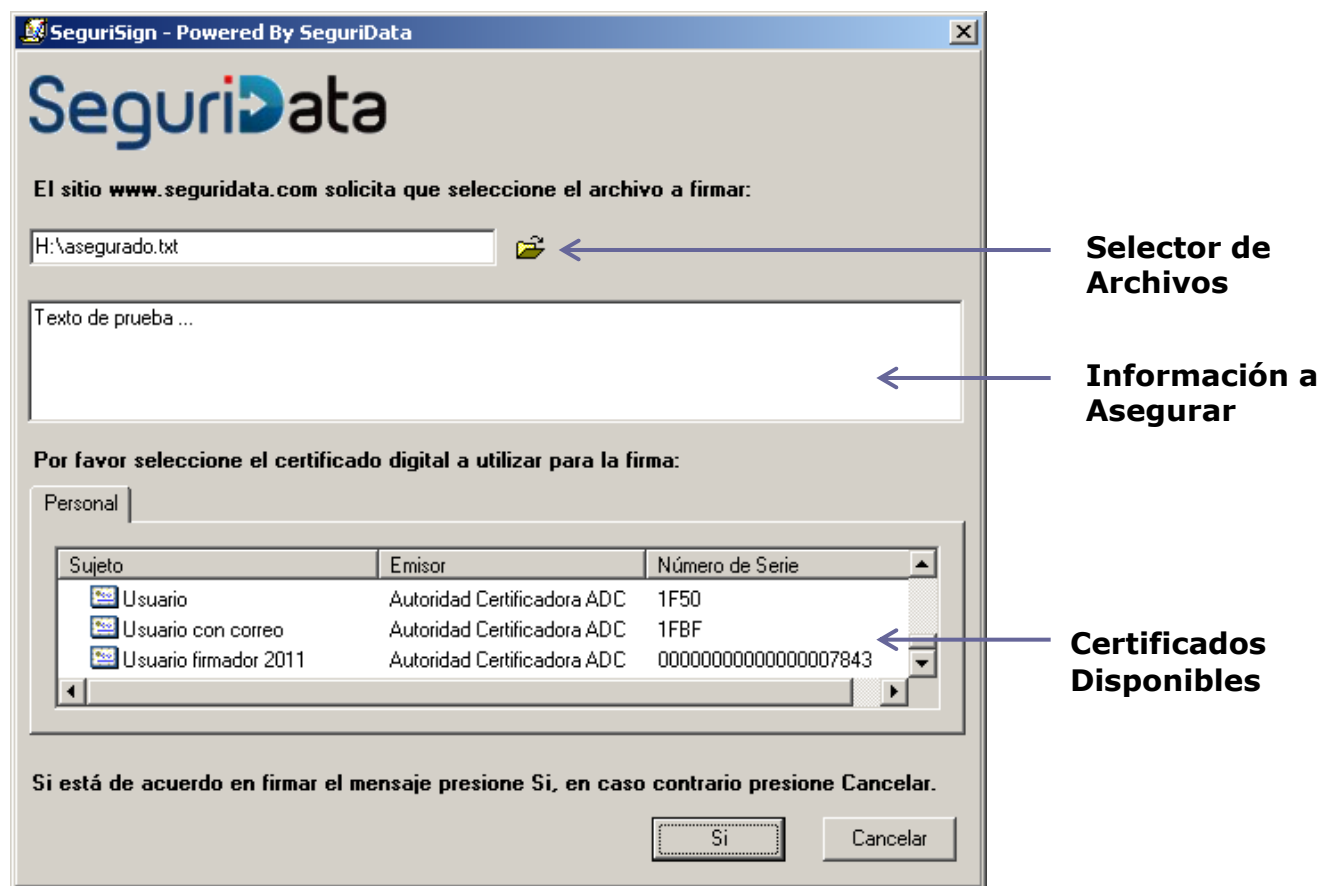
SeguriSign ActiveX - Métodos (II)

Método	Descripción
String SeguriSign.signHash (Arg1, Arg2)	<p>Conforma una firma en modo faceless o sin interfaz, utilizando la cadena de digestión parametrizada.</p> <p><i>Arg1</i> Cadena hexadecimal de la digestión del mensaje a firmar según los algoritmos de digestión MD5 o SHA1</p> <p><i>Arg2</i> Indica el tipo de mensaje a conformar. Si su valor es 1 se conformará</p>
String SeguriSign.retrieveMyCerts(Arg1)	<p>Método que analiza el contenedor de certificados personales y devuelve detalles de los certificados contenidos y que tienen una llave privada asociada</p> <p><i>Arg1</i> corresponde al algoritmo de digestión a utilizar para el cálculo del ThumbPrint del certificado de usuario. Los algoritmos posibles son "MD5" y "SHA1"</p> <p>Notas</p> <p><i>Cuando no hay Certificados la cadena del valor de retorno tiene por valor: "No valid certificates were found." para inglés y "No hay Certificados Digitales Válidos." para Español</i></p> <p>Valor de Retorno</p> <p><i>Si Status es igual a 2000, se devuelve una cadena con el siguiente formato:</i></p> <p><i>"<Serial_1> <Issuer's_CN_1> <Valid_From_1> <Valid_To_1> <Subject's_CN_1> <Cert's_ThumbPrint_1>\r\n... \r\n<Serial_n> <Issuer's_CN_n> <Valid_From_n> <Valid_To_n> <Subject's_CN_n> <Cert's_ThumbPrint_n>\r\n"</i></p>
String SeguriSign.setPfx(Arg1)	<p>Dispone un archivo de intercambio de información personal o PKCS12 en base 64 como una variable de tipo cadena</p> <p><i>Arg1</i> ruta al archivo de intercambio a disponer en base 64</p> <p>Valor de Retorno</p> <p>En una ejecución de éxito, retorna una cadena con el archivo dispuesto en base 64</p>

SeguriSign ActiveX - Métodos (III)

Método	Descripción
SeguriSign.GetFiles()	<p>Despliega un selector de archivos y obtiene la ruta y nombre del archivo seleccionado. Es útil para seleccionar un archivo en modo apertura.</p> <p>Sintaxis</p> <p>selectedFile = SeguriSign.GetFiles()</p>
SSeguriSign.SetFile()	<p>Descripción</p> <p>Despliega un selector de archivos, obteniendo la ruta y nombre del archivo seleccionado. Es útil para seleccionar un archivo en modo escritura.</p> <p>Sintaxis</p> <p>selectedFile = SeguriSign.SetFile()</p>
String SeguriSign.setPfx(Arg1)	<p>Dispone un archivo de intercambio de información personal o PKCS12 en base 64 como una variable de tipo cadena</p> <p><i>Arg1</i> ruta al archivo de intercambio a disponer en base 64</p> <p>Valor de Retorno</p> <p>En una ejecución de éxito, retorna una cadena con el archivo dispuesto en base 64</p>

SeguriSign ActiveX - Interfaz para firma de archivos de texto



Generando una lista de certificados de firmante (ListaDeFirmantes.html)

```
function listMyCerts()
{
    var theListOfCerts = ""
    var thisSeparator = "|"

    mainSeparator = thisSeparator.charAt(0)
    SeguriSign.separator = thisSeparator.charCodeAt(0)

    theListOfCerts = SeguriSign.retrieveMyCerts("SHA1")

    return theListOfCerts;
}
```

Firma digital de archivos (FirmaArchivos.html)

```
function Sign()
{
    var TextToSign
    var pkcs7_ = ""
    var estatus
    var FileName

    TextToSign = ""
    if (document.Firma_Digital.doCompress.checked == true)
        SeguriSign.Compress = 1
    else
        SeguriSign.Compress = 0
    SeguriSign.RejectExpiredCerts = 0
    SeguriSign.Thumbprint = document.Firma_Digital.theCerts.value

    SeguriSign.InFileName = InFileName_
    SeguriSign.OutFileName = OutFileName_

    SeguriSign.b64Columns = 1024
    SeguriSign.b64Output = 1

    pkcs7_ = SeguriSign.firma(TextToSign)

    if (SeguriSign.status == 2000)
    {
        alert("Se realizó con éxito el proceso de firma")
        document.Firma_Digital.FileName.value = SeguriSign.File
    }
}
```


Ensobretado de cadenas de texto (SobreCadenas.html)

```
<html>
...
<script language=javascript>
function sobreCadena()
{
    var ensobretado = ""
    var destinatario = ""

    if (rutaArchivo.value == "")
        seleccionaArchivo()

    if (cadenaAEnsobretar.value == "") {
        alert("No ha proporcionado la cadena a ensobretar")
        cadenaAEnsobretar.focus()
        return
    }

    destinatario = SeguriSign.setPfx(rutaArchivo.value)

    if (SeguriSign.status == 2000) {
        SeguriSign.Thumbprint = theCerts.value
        ensobretado = SeguriSign.Sobre(cadenaAEnsobretar.value, destinatario)
        if (SeguriSign.status == 2000)
            sobre.value = ensobretado
    }

}</ script >
...
```

Firmado de archivo en modo 'detached' (FirmaArchivosDetached.html)

```
<html>
...
<script language=javascript>
function firmaArchivo()
{
    var firmado = ""

    if (rutaArchivo.value == "")
        seleccionaArchivo()

    SeguriSign.Thumbprint = theCerts.value
    SeguriSign.InFileName = rutaArchivo.value
    SeguriSign.empty = 1

    firmado = SeguriSign.Firma("")

    if (SeguriSign.status == 2000)
        firma.value = firmado
}
</ script >
...
```

Firma a partir de una cadena de digestión - Modo detached (FirmaHash.html)

```
<html>
...
<script language=javascript>
function firmaHash()
{
    var firmado = ""

    if (hashParaFirma.value == "")
    {
        alert("No ha proporcionado la digestión para conformar la firma")
        hashParaFirma.focus()
        return
    }

    SeguriSign.Thumbprint = theCerts.value

    firmado = SeguriSign.signHash(hashParaFirma.value, 1)
    if (SeguriSign.status == 2000)
        firma.value = firmado

}
</ script >
...
```

Firma de transacciones utilizando PKCS#12 - (FirmaPKCS12.html)

```
...
<script language=VBS>
  Sub CertInst_OnClick
  ...

  thePfx = SeguriSign.setPfx(document.Instalacion.LLAVE_PRIVADA.value)

  if SeguriSign.Status <> 2000 then
    Exit Sub
  end if

  SeguriSign.userkeys = thePfx
  SeguriSign.userpwd = document.Instalacion.CLAVE_PRIVADA.value

  if document.Instalacion.doWarning.checked = true then
    SeguriSign.high = 1
  else
    SeguriSign.high = 0
  end if

  pkcs7_ = SeguriSign.firma(document.Instalacion.text2Sign.value)

  if SeguriSign.Status = 2000 then
    document.Instalacion.PKCS7.value = pkcs7_
    document.Instalacion.submit()
  end if

  End Sub
</ script >
...
```

Prerrequisitos para uso de SeguriSign Applet en navegadores Java enabled

*Plataformas

Multiplataforma: Windows 98 SE, Me, 2000, NT, XP, 2003, 2008, Vista, W7. Mac OS X versión 10.3.5, Solaris 10

*Consideraciones Generales

- El Navegador de la máquina cliente debe tener asociado el Plugin de Java 2 versión 1.4 o superior. La versión recomendada es la 1.4.2_05.
- El usuario debe contar con un archivo de permisos en su directorio HOME. El archivo se llama .java.policy y su contenido es el siguiente:

```
grant { permission java.io.FilePermission "<<ALL FILES>>", "read", signedBy "SeguriData"; };
```

Si el archivo no existe, el usuario puede crearlo y si ya existe, sólo hay que agregarle dichas líneas.

Estos requisitos son necesarios para cualquier plataforma y se deben a que la infraestructura criptográfica de Java se convirtió en parte de la distribución estándar desde la versión mencionada y a que este lenguaje maneja un sandbox que limita las operaciones que un applet puede ejecutar en la máquina cliente, de tal forma que no puedan ejecutarse acciones maliciosas y/o no autorizadas por parte del usuario.

Cómo incluir SeguriSign Applet en una página HTML

```
<html>
.....
<body>
.....
</form>
<APPLET
  CODE= "seguridata.segurisign.Signer"
  JAVA_CODEBASE= "..\applets\"
  id="theSigner"
  ARCHIVE= "SeguriSignClient.jar"
  WIDTH=0
  HEIGHT=0
  NAME=theSigner
  MAYSCRIPT=false
  type= "application/x-java-applet;version=1.4"
  scriptable=false>

</body>
</html>
```

CODE define la clase principal a utilizar

JAVA_CODEBASE indica dónde localizar en el servidor el archivo del applet de acuerdo a la propiedad ARCHIVE

WIDTH y HEIGHT definen la dimensión del applet en el navegador, que por ser faceless carece de interfaz.

NAME define el alias con el que se hará referencia al applet en las funciones Javascript

TYPE indica la versión 1.4 de plug-in como necesaria y detecta si está o no presente



Los errores que se presentan al utilizar SeguriSign Applet se deben a permisos del navegador, una mala integración del producto o mala operación del aplicativo por parte del usuario final. La versión del JRE puede ocasionar errores de parseo en certificados y/o bolsas de certificados (PKCS12)

Firma digital de cadenas de texto (FirmaCadenas(Applet).html)

```
var p7var = ""
var current = null

current = document.getElementById("TEXT_TO_SIGN")

if (current.value != "")
{
    theAppSigner.setDoFile(false)
    theAppSigner.setToSign(current.value)
}
else
{
    alert("Por favor ingrese el mensaje que desea firmar")
    return
}

theAppSigner.setSigAlg(document.getElementById("signAlgorithms").value)

theAppSigner.setSignerAlias(theAliases.value)

theAppSigner.setCodePKCS7()

theAppSigner.sign()

if (theAppSigner.isSuccess())
    p7var = theAppSigner.getSignature()
else
    p7var = theAppSigner.getErrorMessage()

if (p7var.indexOf("Error") == 0)
    procesa_excepcion(p7var)
else
    p7.value = p7var
```

Mensajes firmados y Ensobretados - (FirmaSobreArchivos(Applet).html)

```
var p7var = ""
var current = null

current = document.getElementById("TEXT_TO_SIGN")

if (current.value != "")
{
    theAppSigner.setDoFile(false)
    theAppSigner.setToSign(current.value)
}
else
{
    alert("Por favor ingrese el mensaje que desea firmar")
    return
}

theAppSigner.setSigAlg(document.getElementById("signAlgorithms").value)

theAppSigner.setSignerAlias(theAliases.value)

theAppSigner.setCodePKCS7()

theAppSigner.envelope(document.getElementById("destinatarios").value)

if (theAppSigner.isSuccess())
    p7var = theAppSigner.getSignature()
else
    p7var = theAppSigner.getErrorMessage()

if (p7var.indexOf("Error") == 0)
    procesa_excepcion(p7var)
else
    p7.value = p7var
```


Firmado de archivo - Modo “detached” (FirmaArchivosDetached(Applet).html)

```
function firma()
{
    var p7var = ""
    var current = null
    try
    {

        theAppSigner.setSigAlg(document.getElementById("signAlgorithms").value)

        theAppSigner.setSignerAlias(theAliases.value)
        theAppSigner.setToSignIsDigest(true)
        theAppSigner.setToSign(document.getElementById("theDigest").value)

        theAppSigner.setCodePKCS7()

        theAppSigner.sign()

        p7var = theAppSigner.getSignature()

        if (p7var.indexOf("Error") == 0)
            procesa_excepcion(p7var)
        else
            p7.value = p7var

    }catch(err)
    {
        procesa_excepcion(err.description == null ? err.message : err.description)
    }
}
```

Applet versus ActiveX

- Plataforma

- Si se requiere una solución que sea multiplataforma el componente a utilizar será SeguriSign Applet. SeguriSign ActiveX sólo funciona para plataformas Windows.

- Navegador

- SeguriSign ActiveX es sólo integrable en Navegadores ActiveX - Enabled: IE 5.0+ y NN7.1+, mientras que SeguriSign Applet es soportado por una variedad de Navegadores que sean Java Enabled.

- SeguriSign ActiveX ofrece como opciones para firma y sobre: interfaz gráfica de usuario; modo faceless o sin interfaz: localización de certificados a partir de huella digital o número de Serie y nombre de Autoridad; firma a partir de cadena Pkcs12; firma de cadenas o archivos

- Si se desea o requiere movilidad para los firmantes de transacciones y que las llaves privadas no residan en el equipo la opción sería Applet dado a su capacidad de firma a partir de archivo Pkcs12

- Si el utilizar tokens criptográficos da un valor agregado a la solución o es requisito del aplicativo el módulo de firma ideal para el soporte de estos es SeguriSign ActiveX por su fácil conexión con cualquier CSP que sea “MS CAPI compliant”

- SeguriSign Applet ofrece funciones de firma y sobre digital: a partir de archivo de certificado para el destinatario del sobre y archivo pkcs12 para las llaves del firmante; opción de ensobretado utilizando como parámetro una cadena de certificado x.509

Applet versus ActiveX (II)

- Permisos de Ejecución
 - SeguriSign Applet requiere en la mayoría de los navegadores editar un archivo de permisos y crear una entrada especial para que tenga permisos de ejecución y no sea restringido.
 - SeguriSign ActiveX puede llegar a necesitar en algunos casos que se le proporcionen permisos de ejecución, pero estos son aislados y muchas veces controlables a partir de los perfiles del Sistema Operativo o mucho más amigables de definir que para el Applet.
- Interoperabilidad con otros productos SeguriData
 - Los mensajes criptográficos firmados conformados por SeguriSign ActiveX y SeguriSign Applet son compatibles con SeguriLib y SeguriDoc. Es posible que versiones no recientes de dichos productos indiquen errores de formato inválido al procesarlos.
 - Un mensaje firmado y ensobretado por SeguriSign ActiveX es procesado por SeguriDoc y SeguriLib como simple sobre dado a que se conforma en capas: primero se genera un mensaje firmado y éste a su vez es ensobretado. SeguriDoc al abrir el sobre no dispone en la lista de firmantes certificado digital alguno ya que desconoce que al abrir el sobre se necesita procesar el contenido como mensaje firmado. De igual forma es posible que versiones no recientes de los productos mencionados indiquen errores de formato en estos mensajes.
 - Los mensajes firmados y ensobretados generados por SeguriSign Applet son estándar a SeguriLib y por consiguiente para SeguriDoc estándar, mas no para SeguriDoc CAPI debido a la combinación de firmado y ensobretado en un mismo mensaje.

Java API

Método

SSignEvidence **authenticateCryptographicMessage** (SSignDocument cryptographicMessage, SSignDocument externContent, java.lang.String folio, java.lang.String serial)

Solicita la autenticación de un mensaje criptográfico

Argumento	Tipo	Detalle
server	String	Nombre de servidor o ip SeguriSign
Port	int	Puerto del servicio SeguriSign
signedMessage	Byte[]	Arreglo de bytes con el documento firmado que se desea autenticar.
Folio	String	Número de Folio para la transacción y que puede ser relevante para el aplicativo que integra Firma Digital
Serial	String	Número de Serie del Certificado que se espera utilice el Firmante
FileName	String	Nombre del archivo firmado procedente de la ejecución de algún módulo de Cliente de Firma (Applet o ActiveX). En caso que no se firmen archivos sino transacciones de texto, se recomienda ampliamente enviar un valor de nombre de archivo arbitrario con extensión txt.
ExternContent	byte[]	Información Firmada para el caso de transacciones procedentes de un navegador Netscape 4.x, en los casos en que la firma provenga de otro navegador, su valor será "NONE"

Returns: El objeto SSignEvidence obtenido contiene la secuencia de firma en caso de que el proceso haya sido exitoso.

Autenticación de mensajes criptográficos unilaterales

```
// Ejemplo de invocación de método AuthenticateCryptographicMessage
import com.seguridata.segurisign.api.UnilateralSignature;
import com.seguridata.segurisign.beans.*;

//Se crea instancia de gestor de procesos de firma unilateral
UnilateralSignature securisign = new UnilateralSignature();

//Se asignan los valores del servidor SeguriSign (nombre y puerto)
securisign.setServer(server);
securisign.setPort(Integer.parseInt(port));

//Se crea instancia de clase SSignDocument y asigna el contenido del archivo del mensaje firmado
SSignDocument signedMessageDoc = new SSignDocument();

signedMessageDoc.setData(signedMessage);
signedMessageDoc.setName(fileName);

//Se crea instancia de clase SSignDocument para documento externo de documento firmado no
'detached'
SSignDocument externContent = new SSignDocument();

externContent.setData("NONE".getBytes());
externContent.setName(externFileName);

//Se solicita que se autentique el mensaje de firma unilateral
SSignEvidence tspStamp = securisign.authenticateCryptographicMessage(signedMessageDoc,
externContent, folio, serial);

//Despliegue de identificador de proceso de firma unilateral
unilateralID = tspStamp.getSequence();
System.out.println("Identificador de proceso de firma unilateral: " + unilateralID);
```

Método

[SSignDocument](#) **getOriginalDocument** (java.lang.String **Sequence**)

Solicita al servidor el documento original de la transacción que corresponde con el número de secuencia (Sequence) proporcionado como parámetro. El documento en caso de existir estará contenido en la instancia de la clase SSignDocument que devuelve el método [getOriginalDocument\(\)](#). A partir del resultado se puede obtener el nombre del documento mediante el método [getName\(\)](#) y su contenido con el método [getData\(\)](#)

Argumento	Tipo	Detalle
Sequence	String	Secuencia de la Transacción de la que se obtendrá la información original asegurada

Solicitud de información original asegurada al servidor SeguriSign

```
// Ejemplo de invocación de método getOriginalDocument
import com.seguridata.segurisign.beans.SSignDocument;

try{
    SSignDocument signedMessageRecovered = new SSignDocument();
    signedMessageRecovered = securisign.getOriginalDocument(Sequence);
    byte bytesDocto[] = signedMessageRecovered.getData();

    System.out.println("Documento firmado:");
    System.out.println("Nombre:" + signedMessageRecovered.getData());
    for (int i : bytesDocto){
        System.out.print((char)bytesDocto[i]);
    }
} catch (Exception securisignExc) {
    System.out.println("Error de gesti\u00f3n de proceso unilateral"
        + lineSeparator + securisignExc.getMessage());
}
```


Método

[SSignEvidence](#).**getCryptographicEvidence**(java.lang.String **sequence**, ParameterTypes.EvidenceType **requestedEvidence**)

Solicita una evidencia criptográfica asociada a la Secuencia SeguriSign especificada

Argumento	Tipo	Detalle
Sequence	String	Secuencia SeguriSign de la que se solicita una evidencia criptográfica en particular
requestedEvidence	ParameterTypes.EvidenceType	Tipo de evidencia solicitada, los valores posibles se encuentran definidos en la clase ParameterTypes.

Returns:

Evidencia Criptográfica asociada a la Secuencia SeguriSign especificada,

Solicitud de evidencias criptográficas asociadas a una transacción SeguriSign

```
// Ejemplo de invocación de método GetCryptographicData4Sequence
import com.seguridata.segurisign.api.ParameterTypes.EvidenceType;
import com.seguridata.segurisign.beans.SSignEvidence;
import com.seguridata.segurisign.api.UnilateralSignature;
import com.seguridata.segurisign.api.TSPStampDecoder;
import com.seguridata.segurisign.api.ParameterTypes.DigestAlgorithm;

try{
    UnilateralSignature securisign = new UnilateralSignature();
    for (EvidenceType c : EvidenceType.values()) {
        SSignEvidence actualEvidence = null;
        actualEvidence = securisign.getCryptographicEvidence(Sequence, evidenceType);
        switch (c) {
            case PKCS7:
                System.out.println(securisign.getSignedMessageData(actualEvidence));
                break;
            case OCSP:
                System.out.println(actualEvidence);
                break;
            case TSP:
                TSPStampDecoder tspDecoder = new TSPStampDecoder();
                System.out.println(tspDecoder.decode(actualEvidence,
                    DigestAlgorithm.SHA1));
                break;
        }
    }
} catch (Exception securisignExc) {
    System.out.println("Error de gesti\u00f3n de proceso unilateral"
        + lineSeparator + securisignExc.getMessage());
}
```

Método

MultilateralProcess **init**(byte[] data, Process.DataType dataType, ParameterTypes.DigestAlgorithm digestAlgorithm, java.lang.String dataInfo, Process.ProcessType processType)
throws java.lang.IllegalArgumentException,
java.lang.Exception

Inicia el proceso de firma Multilateral

Argumento	Tipo	Detalle
data	byte[]	Información a asociar al proceso de firma multilateral
dataType	ParameterTypes.DigestAlgorithm	Especifica cómo interpretar los datos parametrizados
digestAlgorithm	String	Define el algoritmo de digestión involucrado en el proceso de firma (SHA1)
dataInfo	String	Metadato para el proceso de firma
processType	Process.ProcessType	Indica si se genera un paquete firmado CMS o XML.

Returns:

Detalles del proceso multilateral iniciado

Throws:

java.lang.IllegalArgumentException - Alguno de los argumentos es nulo o no tiene valor
java.lang.Exception - Error durante el procesamiento del mensaje

Método

```
java.lang.String update(SSignDocument signedMessage,  
                        java.lang.String id,  
                        java.lang.String serial)
```

throws

java.lang.IllegalArgumentException, java.lang.Exception

Autentica un mensaje firmado correspondiente a un proceso de firma multilateral iniciado o abierto

Argumento	Tipo	Detalle
signedMessage	SSignDocument	Mensaje criptográfico a asociar al proceso de firma multilateral
Id	String	Identificador del proceso de firma multilateral ya iniciado
serial	String	Habilita el solicitar que cierto usuario haya firmado el mensaje (serie de certificado)

Returns:

Identificador asociado a la firma autenticada

Throws:

java.lang.IllegalArgumentException - Alguno de los argumentos es nulo o no tiene valor

java.lang.Exception - Error durante el procesamiento del mensaje

Método

MultilateralProcess **finalize**(java.lang.String id,
ParameterTypes.CMSType reportType)
throws java.lang.IllegalArgumentException,
java.lang.Exception

Finaliza el proceso de firma multilateral

Argumento	Tipo	Detalle
Id	String	Identificador del proceso de firma multilateral ya iniciado
reportType	ParameterTypes .CMSType	Indica qué acción tomar en el proceso y qué firmas incluir en el mismo

Throws:

java.lang.IllegalArgumentException - Alguno de los argumentos es nulo o no tiene valor
java.lang.Exception - Error durante el procesamiento del mensaje

Inicio de un proceso para firma multilateral

```
// Ejemplo de invocación de método multiSignedMessage_Init
import SSign.Process.DataType;
import SSign.Process.ProcessType;
import com.seguridata.segurisign.api.MultilateralSignature;
import com.seguridata.segurisign.api.ParameterTypes.DigestAlgorithm;
import com.seguridata.segurisign.beans.MultilateralProcess;
    try {
        //Se crea instancia de gestor de procesos de firma multilateral
        MultilateralSignature securisign = new MultilateralSignature();
        //Se asignan los valores del servidor SeguriSign (nombre y puerto)
        securisign.setServer(args[0]);
        securisign.setPort(Integer.parseInt(args[1]));
        //Se dispone la ruta del archivo a dar de alta como archivo a firmar
        //en el proceso de firma multilateral a iniciar
        byte[] data = args[2].getBytes();
        //Para este ejemplo el metadato a asignar al proceso ser\u00e9 el
        //nombre del archivo dado de alta
        String dataInfo = new File(args[2]).getName();
        //Se inicia un proceso de firma multilateral con la informaci\u00f3n
        //contenida en el archivo parametrizado. Se van a generar firmas a
        //partir de un hash SHA1 para un proceso basado en CMS
        MultilateralProcess multilateralProcess = securisign.init(data,
            DataType.FILE, DigestAlgorithm.SHA1, dataInfo,
            ProcessType.CMS);
        multilateralId = multilateralProcess.getId();
        hash2Sign = multilateralProcess.getThumbPrint();
        System.out.println("Proceso iniciado con id:" + lineSeparator +
            multilateralId);
        System.out.println("Hash a utilizar para generar la firma:" +
            lineSeparator + hash2Sign);
    } catch (Exception e) {
        System.out.println("Error de gesti\u00f3n de proceso multilateral" +
            lineSeparator + e.getMessage());
    }
```

Autenticación de mensajes para firma multilateral

```
// Ejemplo de invocación de método multiSignedMessage_Update
import com.seguridata.segurisign.api.MultilateralSignature;
import com.seguridata.segurisign.beans.SSignDocument;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
    try {
        //Se crea instancia de gestor de procesos de firma multilateral
        MultilateralSignature securisign = new MultilateralSignature();
        //Se asignan los valores del servidor SeguriSign (nombre y puerto)
        securisign.setServer(args[0]);
        securisign.setPort(Integer.parseInt(args[1]));
        //Se genera instancia de clase SSignDocument para primer mensaje
        //firmado y se asignan sus propiedades
        SSignDocument signedMessage1 = new SSignDocument();
        signedMessage1.setBase64(false);
        signedMessage1.setData(getBytesFromFile(new File(args[3])));
        //Se solicita autenticar la firma
        String sequence1 = securisign.update(signedMessage1, multilateralId,
serial);

        //Despliegue de resultados
        System.out.println("Firma autenticada con id: " + sequence1);
        System.out.println("Firma autenticada con id: " + sequence2);
    } catch (Exception e) {
        System.out.println("Error de gesti\u00f3n de proceso multilateral"
            + lineSeparator + e.getMessage());
    }
```

Cierre de procesos para firma multilateral

```
// Ejemplo de invocación de método multiSignedMessage_Final
import com.seguridata.segurisign.api.MultilateralSignature;
import com.seguridata.segurisign.api.ParameterTypes.CMSType;
import com.seguridata.segurisign.api.ParameterTypes.DigestAlgorithm;
import com.seguridata.segurisign.api.ParameterTypes.ProcessStatus;
import com.seguridata.segurisign.beans.MultilateralProcess;
    try {
        MultilateralSignature securisign = new MultilateralSignature();
        //Se asignan los valores del servidor SeguriSign (nombre y puerto)
        securisign.setServer(args[0]);
        securisign.setPort(Integer.parseInt(args[1]));
        //Previo al cierre del proceso se solicita su estatus
        ProcessStatus status = securisign.getStatus(multilateralId);
        //Se despliega estatus actual del proceso
        System.out.println("Estatus antes del cierre " + status);
        //Se solicita el cierre del proceso
        MultilateralProcess multilateralProcess =
securisign.finalize(multilateralId,
                        CMSType.AllSignaturesWithClose);
        //Previo al cierre del proceso se solicita su estatus
        status = securisign.getStatus(multilateralId);
        //Se despliega estatus actual del proceso
        System.out.println("Estatus luego del cierre " + status);
    } catch (Exception e) {
        System.out.println("Error de gesti\u00f3n de proceso multilateral"
                           + lineSeparator + e.getMessage());
    }
```


Revisión de firmantes de un proceso de firma multilateral

```
// Ejemplo de invocación de método getMultilateralSignedMessageData
import com.seguridata.segurisign.api.MultilateralSignature;
import com.seguridata.segurisign.api.ParameterTypes.CMSType;
import com.seguridata.segurisign.api.ParameterTypes.DigestAlgorithm;
import com.seguridata.segurisign.api.ParameterTypes.ProcessStatus;
import com.seguridata.segurisign.beans.MultilateralProcess;
import com.seguridata.segurisign.api.TSPStampDecoder;
import com.seguridata.segurisign.beans.CMSSignedMessage;

try {
    MultilateralSignature securisign = new MultilateralSignature();

    //Se asignan los valores del servidor SeguriSign (nombre y puerto)
    securisign.setServer(args[0]);
    securisign.setPort(Integer.parseInt(args[1]));

    CMSSignedMessage cMSSignedMessage =
securisign.getMultilateralSignedMessageData(multilateralProcess.getCms().getData(),
        multilateralId, DigestAlgorithm.SHA1);

    //Se despliegan los detalles del CMS final
    System.out.println(cMSSignedMessage);
    //Se crea instancia de clase para parsear estampillas TSP
    TSPStampDecoder parser = new TSPStampDecoder();

    //Despliegue de detalles de estampa de cierre del proceso de firma multilateral
    System.out.println(parser.decode(
        multilateralProcess.getTimeStamp(), DigestAlgorithm.SHA1));
} catch (Exception e) {
    System.out.println("Error de gesti\u00f3n de proceso multilateral"
        + lineSeparator + e.getMessage());
}
```

API .NET

Método

Int **AuthenticatePKCS7** (String securiSignIP, int securiSignPort, string folio, int folioLen, string serial, int serialLen, string fileName, string isBase64, string pkcs7, int pkcs7Len, string externContent, int externContentLen, char getReceipt, StringBuilder Sequence, StringBuilder Receipt, ref int receiptLen, StringBuilder errorMsg)

Solicita la autenticación de un PKCS7

Argumento	Tipo	Detalle
securiSignIP	String	Nombre de servidor o ip SeguriSign
securiSignPort	int	Puerto del servicio SeguriSign
folio	String	Asignado por la aplicación si así lo desea, en caso contrario pudiera ser N/A ; MAX(40)
folioLen	String	Longitud de "Folio"
Serial	String	indica el numero de serie del certificado del firmante q se espera haya sido utilizado, en caso contrario colocar "11111111111111111111"
serialLen	String	Longitud de "Serial"
fileName	String	Nombre del archivo original
isBase64	String	Indica si la información firmada estaba codificada en base 64 antes de procesarse o no; por lo general aplica "FALSE"
pkcs7	String	Firma pkcs7

Método *AuthenticatePKCS7 II*

Int **AuthenticatePKCS7** (String securiSignIP, int securiSignPort, string folio, int folioLen, string serial, int serialLen, string fileName, string isBase64, string pkcs7, int pkcs7Len, string externContent, int externContentLen, char getReceipt, StringBuilder Sequence, StringBuilder Receipt, ref int receiptLen, StringBuilder errorMsg)

Solicita la autenticación de un PKCS7

Argumento	Tipo	Detalle
pkcs7Len	Int	Longitud del pkcs7
externContent	String	Aplica a mensajes firmados q no contienen la informacion firmada en el pkcs7; cuando lo contienen se parametriza "NONE"
externContentLen	int	Longitud de "externContent"
getReceipt	Char	Indica si se desea obtener una copia del recibo criptografico generado para la transaccion 'Y' o 'N'
Sequence	StringBuilder	Número de secuencia asignado por SeguriSign MAX(40)
Receipt	StringBuilder	recibo criptográfico obtenido MAX(1024*5)
receiptLen	Int	Longitud del recibo criptográfico obtenido
errorMsg	String	Devuelve un mensaje de error en caso de existir.

Returns: Valor numérico "2000" en caso de ser exitoso.

Autenticación de mensajes criptográficos unilaterales

```
// Importar funcion AuthenticatePKCS7 de librería SeguriSIGN.dll
[DllImport("SeguriSIGN.dll", CharSet = CharSet.Ansi)]
public static extern int AuthenticatePKCS7(
    String securisignIP,
    int securisignPort,
    String folio,
    int folioLen,
    String serial,
    int serialLen,
    String fileName,
    String isBase64,
    String pkcs7,
    int pkcs7Len,
    String externContent,
    int externContentLen,
    System.Char getReceipt,
    StringBuilder Sequence,
    StringBuilder receipt,
    ref int receiptLen,
    StringBuilder errorMsg);
```

}

[illegible]

Método

Int GetOriginalDocumentI (String IP,int Port,String TransactionNumber,int InBase64,String FileName,String OriginalDocument,String OriginalDocumentLen,String ErrorMsg)

Solicita al servidor el documento original de la transacción que corresponde con el número de secuencia (Sequence) proporcionado como parámetro.

Argumento	Tipo	Detalle
IP	String	Secuencia de la Transacción de la que se obtendrá la información original asegurada
Port	int	Puerto en el que escucha SeguriSign
TransactionNumber	String	Numero de Secuencia
InBase64	Int	Indica si está codificado en base64
FileName	String	Nombre del archivo
OriginalDocument	StringBuilder	Devuelve el Documento original
OriginalDocumentLen	String	Devuelve la Longitud del documento original
ErrorMsg	StringBuilder	Mensaje de error en caso de existir

Solicitud de información original asegurada al servidor SeguriSign

```
// Importar funcion GetOriginalDocumentI de Librería SeguriSIGN.dll
[System.Runtime.InteropServices.DllImport("SeguriSIGN.dll", CharSet =
    System.Runtime.InteropServices.CharSet.Ansi)]
    public static extern int GetOriginalDocumentI(
        String IP,
        int Port,
        Strig TransactionNumber,
        int InBase64,
        String FileName,
        String OriginalDocument,
        String OriginalDocumentLen,
        String ErrorMsg);
```

Returns

1 = Correcto, <> 1 = Incorrecto

Solicitud de información original asegurada al servidor SeguriSign II

```
// Importar funcion SaveToFile de Librería SeguriSIGN.dll
[System.Runtime.InteropServices.DllImport("SeguriSIGN.dll", CharSet =
    System.Runtime.InteropServices.CharSet.Ansi)]
    public static extern int SaveToFile(
        string Buffer,
        int BufferLen,
        string FileName);
```

Returns

1 = Correcto, <> 1 = Incorrecto

Solicitud de información original asegurada al servidor SeguriSign II

```
// Ejemplo de invocación de método getOriginalDocument
IP = TB_IP.Text;
Port = Val(ssPort.Text);
TransactionNumber = TB_NRecibo2.Text;
InBase64 = "FALSE";
FileName = Space(255);
// Depende del tamaño del archivo original. Puede dimensionarse del tamaño máximo de la transacción
OriginalDocument = Space(1024 * 10);
OriginalDocumentLen = 0;
ErrorMsg = Space(512);

Status = GetOriginalDocumentI(IP, Port, TransactionNumber, InBase64, FileName, OriginalDocument,
                              OriginalDocumentLen, ErrorMsg);

if (Status == 1) {
    FileName = "NombreArchivo.txt";
    Status = SaveToFile(OriginalDocument, OriginalDocumentLen, FileName);
    if (Status == 1) {
        MessageBox.Show( "Documento Guardado en " + FileName);
    } else {
        MessageBox.Show( "Error al salvar documento original");
    }
}

// Error
} else {
    MessageBox.Show("Error al obtener documento original: "+ Trim(ErrorMsg));
}
```

Método

Int **MultiSignedMessage_Init**(String securisignIP, int securisignPort, StringBuilder data, int dataLen, System.Char dataType, String info, StringBuilder theID, StringBuilder theDigest, ref int theDigestLen, StringBuilder errorMsg)

Inicial el proceso de firma multilateral

Argumento	Tipo	Detalle
securiSignIP	String	IP del Servidor de SeguriSign
securisignPort	Int	Puerto en donde escucha el SeguriSign
Data	StringBuilder	ID que servirá para identificar las transacciones asociadas a un mismo proceso de firma multilateral
dataLen	Int	Longitud de “Data”
DataType	Char	0 - Información a registrar; 1 - Ruta del archivo a registrar; 2 - Digestión del archivo (SHA1)
Info	String	Nombre del archivo
theID	StringBuilder	Identificador único para el proceso de firma multilateral iniciado
theDigest	StringBuilder	Digestión del Documento

Método

Argumento	Tipo	Detalle
theDigestLen	Int	Longitud de la Digestión
errorMsg	StringBuilder	Mensaje de error en caso de existir

Returns:

Digito 1 en caso de ser correcto, <> de 1 en caso de existir un error

Método

```
public static extern int multiSignedMessage_Update(String securisignIP,int securisignPort,StringBuilder p7,int p7Len,String id,String serial,StringBuilder numSeq,StringBuilder errorMsg) throws  
    java.lang.IllegalArgumentException, java.lang.Exception
```

Autentica un mensaje firmado correspondiente a un proceso de firma multilateral iniciado o abierto

Argumento	Tipo	Detalle
securiSignIP	String	IP del Servidor de SeguriSign
securisignPort	Int	Puerto en donde escucha el SeguriSign
P7	StringBuilder	Pcks7 de resultado de la firma del hash del documento sometido a firma
P7len	Int	Longitud de P7
Id	String	Identificador único para el proceso de firma multilateral
Serial	String	Numero de serie del certificado.
numSeq	StringBuilder	Devuelve un número de secuencia resultado de la autenticación
errorMsg	StringBuilder	Devuelve un mensaje de error en caso de existir

Returns:

Digito 1 en caso de ser correcto, <> de 1 en caso de existir un error

Método

```
public static extern int multiSignedMessage_Final(String securisignIP,int securisignPort,String securisignID,System.Char reportType, ref System.IntPtr cms,ref int cmsLen,ref System.IntPtr tsp4WholeCMS,ref int tsp4WholeCMSLen,StringBuilder additionalData,ref int additionalDataLen,StringBuilder errorMsg)
```

Finaliza el proceso de firma multilateral

Argumento	Tipo	Detalle
securiSignIP	String	IP del Servidor de SeguriSign
securisignPort	Int	Puerto en donde escucha el SeguriSign
Id	String	Identificador único para el proceso de firma multilateral
reportType	Char	Tipo de reporte 0=all_no_close; 1=allAndClose; 2=local_no_close; 3=foreign_no_close
Cmd	IntPtr	Devuelve el CMS del proceso
cmsLen	Int	Longitud del CMS
tsp4WholeCMS	IntPtr	Devuelve el TSP
tsp4WholeCMSLen	Int	Devuelve la longitud del TSP

Método

Argumento	Tipo	Detalle
additionalData	StringBuilder	attached data - (size 256)
additionalDataLen	Int	Longitud para “additionalData”
errorMsg	StringBuilder	Devuelve un mensaje de error en caso de existir

Returns:

Digito 1 en caso de ser correcto, <> de 1 en caso de existir un error

Inicio de un proceso para firma multilateral

```
//Inicia el Proceso de firma multilateral
SeguriSIGNP.SeguriSIGNClass SSign = new SeguriSIGNP.SeguriSIGNClass();
StringBuilder archivoFuente = new StringBuilder(1024);
archivoFuente.Append("D:\\1.txt");
StringBuilder theID = new StringBuilder(64);
theID.Append(1);
StringBuilder theDigest = new StringBuilder(256);
StringBuilder errorMessage = new StringBuilder(1024);
string info = "1.txt";
int lenDigest = theDigest.Capacity;
int success;
bool flag = false;
success = multiSignedMessage_Init(
    textBox1.Text,
    int.Parse(numericUpDown1.Value.ToString()),
    archivoFuente,
    archivoFuente.Length, //longitud de archivo aislado
    '1',
    info, //nombre del archivo
    theID,
    theDigest,
    ref lenDigest,
    errorMessage);

SSign.b64Enters = 0;
if (success == 1)
{
    txtID.Text = theID.ToString();
    txtDigest.Text = theDigest.ToString();
}
else
    MessageBox.Show(this, errorMessage.ToString(), "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
```


Autenticación de mensajes para firma multilateral

```
// Ejemplo de invocación de método multiSignedMessage_Update
//inicializar componente
SeguriSIGNP.SeguriSIGNClass SSign = new SeguriSIGNP.SeguriSIGNClass();
    int resultUpdate;
    myListObj element = null;
    StringBuilder errorMessage = new StringBuilder(1024);
    StringBuilder numSeq = new StringBuilder(64);
    StringBuilder numSeque = new StringBuilder(1024);

    SSign.ThumbPrint = "ThumbPrint";

//Firmar HASH
    SSign.Viewer = 1;
    string firma = SSign.signHash(txtDigest.Text.ToString(), 1);
    if (SSign.status == 2000)
    {

//Autenticar
        StringBuilder sbFirma = new StringBuilder(firma);
        resultUpdate = multiSignedMessage_Update(textBox1.Text,
int.Parse(numericUpDown1.Value.ToString()), sbFirma, sbFirma.Length, txtID.Text,
"11111111111111111111111111111111", numSeque, errorMessage);
        if (resultUpdate == 1)
            txtSecuencia.Text = numSeque.ToString().Trim();
        else
            MessageBox.Show(errorMessage.ToString());
    }
    else
    {
        MessageBox.Show("Error :" + errorMessage);
    }
}
```

Cierre de procesos para firma multilateral

```
// Ejemplo de invocación de método multiSignedMessage_Final
IntPtr cmss = new IntPtr();
IntPtr tsp4 = new IntPtr();
string info = "1.txt";
StringBuilder infobuilder = new StringBuilder(info);
int infoLen = infobuilder.Capacity;
int cmslen = 0;
int tsp4len = 0;
StringBuilder errorMessage = new StringBuilder(1024);
//Finalizar proceso de firma multilateral
int final = multiSignedMessage_Final(textBox1.Text,
int.Parse(numericUpDown1.Value.ToString()), txtID.Text, '1', ref cmss, ref cmslen,
ref tsp4, ref tsp4len, infobuilder, ref infoLen, errorMessage);
if (final == 0)
    MessageBox.Show(this, errorMessage.ToString(), "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
else{
    MessageBox.Show(this, "El proceso de firma multilateral fue cerrado", "Aviso",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
    MessageBox.Show(this, "Dato Asociado " + info, "Aviso", MessageBoxButtons.OK,
    MessageBoxIcon.Information);
    Byte[] biteCMS = new Byte[cmslen];
    System.Runtime.InteropServices.Marshal.Copy(cmss, biteCMS, 0, cmslen);
    Byte[] biteTSP = new Byte[tsp4len];
    System.Runtime.InteropServices.Marshal.Copy(tsp4, biteTSP, 0, tsp4len);
    //Guardar CMS y TSP Generado
    EscribirArchivo(biteCMS, biteTSP, txtID.Text.ToString());
    ssign_api_freeMem(ref cmss);
    ssign_api_freeMem(ref tsp4);
}
```

WebServices - Java

Mensaje

sgdata.SignDocumentResponse **SignDocumentRequest**

Solicita al webService la firma de un documento. Los elementos que requiere para su funcionamiento son:

Argumento	Tipo	Detalle
documentoAFirmar	sgdata.Document	Documento que se firmará, se asigna el contenido mediante los métodos setData(content) y setFilename(fileName) de la clase sgdata.Document.
keyID	String	Identificador de las llaves usadas por el usuario. El usuario se determina mediante los métodos setUsername(userName) y setPassword(password) de la clase SgSignToolsWSStub. Para definir el servidor se envía la dirección en el constructor de la clase SgSignToolsWSStub(direccion, null)

Returns:

En una ejecución exitosa la instancia de la clase SignDocumentResponse que resulta contiene el documento firmado y se puede obtener con el método getSignedDoc() de dicha clase.

```
// Ejemplo de invocación de mensaje SignDocumentRequest
SgSignToolsWSStub puertoWebService = new SgSignToolsWSStub(direccion, null);
puertoWebService.setUsername(userName);
puertoWebService.setPassword(password);
sgdata.Document documentoAFirmar = new sgdata.Document();
documentoAFirmar.setData(content);
documentoAFirmar.setFilename(fileName));

//Requerimiento de Firma
sgdata.SignDocumentRequest requerimientoFirma = new sgdata.SignDocumentRequest();
requerimientoFirma.setKeyid(keyID);
requerimientoFirma.setDocToSign(documentoAFirmar);
requerimientoFirma.setWithContent(true);

// envío de la petición de firma
sgdata.SignDocumentResponse respuestaFirma = new sgdata.SignDocumentResponse();
respuestaFirma =
    (sgdata.SignDocumentResponse)puertoWebService.processMessage(requerimientoFirma);
sgdata.Document documentoFirmado = respuestaFirma.getSignedDoc();
```

Método

sgdata.VerifyResponse **VerifyRequest**

Solicita al webService la autenticación de un documento. Los elementos que requiere para su funcionamiento son:

Argumento	Tipo	Detalle
documentoFirmado	sgdata.Document	Documento firmado que se desea autenticar.
folio	String	Número de folio para el documento. Este campo lo define el usuario de acuerdo a la lógica de sus aplicaciones. Para definirlo se puede utilizar el método setFolio() de la clase VerifyRequest.
documentoOriginal	sgdata.Document	Documento que fue firmado. En el caso de firmas sin contenido es necesario definir este dato. Para definirlo se puede utilizar el método setOriginalDoc() de la clase VerifyRequest.

Returns:

En una ejecución exitosa la instancia de la clase VerifyResponse que resulta contiene la secuencia del documento autenticado y se puede obtener con el método getSequence() de dicha clase.

Autenticación de una firma mediante el Web Service.

```
// Ejemplo de invocación del mensaje VerifyRequest

SgSignToolsWSStub puertoWebService = new SgSignToolsWSStub(direccion, null);
puertoWebService.setUsername(userName);
puertoWebService.setPassword(password);
sgdata.Document documentoFirmado = respuestaFirma.getSignedDoc();
sgdata.VerifyRequest verificacionDocumento = new sgdata.VerifyRequest();
verificacionDocumento.setSignedDoc(documentoFirmado);
verificacionDocumento.setFolio(folio);
verificacionDocumento.setOriginalDoc(documentoOriginal);

sgdata.VerifyResponse respuestaVerificacion =
(sgdata.VerifyResponse) puertoWebService.processMessage(verificacionDocumento);
System.out.println("Secuencia: " + respuestaVerificacion.getSequence());
```

Método

sgdata.SignDocumentResponse **SignDocumentByGroupRequest**

Solicita al webService la firma de un documento utilizando el firmante por defecto para un grupo del portal. Los elementos que requiere para su funcionamiento son:

Argumento	Tipo	Detalle
documentoAFirmar	sgdata.Document	Documento que se firmará, se asigna el contenido mediante los métodos setData(content) y setFilename(fileName) de la clase sgdata.Document.
groupID	String	Identificador del grupo cuyo certificado del usuario principal será utilizado para realizar el proceso de firmado. El usuario que se usa para la conexión debe ser un usuario administrador, se determina mediante los métodos setUsername(userName) y setPassword(password) de la clase SgSignToolsWSSStub. Para definir el servidor se envía la dirección en el constructor de la clase SgSignToolsWSSStub(direccion, null)

Returns:

En una ejecución exitosa la instancia de la clase SignDocumentResponse que resulta contiene el documento firmado y se puede obtener con el método getSIGNEDoc() de dicha clase.

Firma de documentos utilizando al firmante principal de un grupo.

```
// Ejemplo de uso del mensaje SignDocumentByGroupRequest

SgSignToolsWSStub puertoWebService = new SgSignToolsWSStub(direccion, null);
puertoWebService.setUsername(userName);
puertoWebService.setPassword(password);
sgdata.Document documentoAFirmar = new sgdata.Document();

sgdata.SignDocumentByGroupRequest requerimientoFirmaByGroup = new
sgdata.SignDocumentByGroupRequest();
sgdata.SignDocumentResponse respuestaFirma = new sgdata.SignDocumentResponse();

// Asignación de documento a firmar
documentoAFirmar.setData(content);
if (sFileName != null)
    documentoAFirmar.setFilename(fileName);
Else
    documentoAFirmar.setFilename("bufferSign");

requerimientoFirmaByGroup.setDocToSign(documentoAFirmar);
requerimientoFirmaByGroup.setGroupId(groupId);
respuestaFirma = (sgdata.SignDocumentResponse) puertoWebService
    .processMessage(requerimientoFirmaByGroup);
sgdata.Document documentoFirmado = respuestaFirma.getSignedDoc();
```

WebServices - .NET

Mensaje

sgdata.SignDocumentResponse **SignDocumentRequest**

Solicita al webService la firma de un documento (firma unilateral). Los elementos que requiere para su funcionamiento son:

Argumento	Tipo	Detalle
documentoAFirmar	sgdata.Document	Documento que se firmará, se asigna el contenido mediante los métodos setData(content) y setFilename(fileName) de la clase sgdata.Document.
keyID	String	Identificador de las llaves usadas por el usuario. El usuario se determina mediante los métodos setUsername(userName) y setPassword(password) de la clase SgSignToolsWSStub. Para definir el servidor se envía la dirección en el constructor de la clase SgSignToolsWSStub(direccion, null)

Returns:

En una ejecución exitosa la instancia de la clase SignDocumentResponse que resulta contiene el documento firmado y se puede obtener con el método getSignedDoc() de dicha clase.

```

// Ejemplo de invocación de mensaje SignDocumentRequest
//Instancia del Webservice
Uri myuri = new Uri(txt_url.Text);
SgSignToolsWSDemo.SgSingToolsWS.SgSignToolsWS ws = new
SgSignToolsWSDemo.SgSingToolsWS.SgSignToolsWS();
ws.Url = txt_url.Text;
ws.Credentials = new NetworkCredential(txt_user.Text, txt_pwd.Text);

//requerimiento de firma
    SgSingToolsWS.SignDocumentRequest request = new
    SgSignToolsWSDemo.SgSingToolsWS.SignDocumentRequest();
    request.keyid = txt_id.Text;
    request.withContent = false;

//Documento a Firmar
    request.docToSign = new SgSignToolsWSDemo.SgSingToolsWS.Document();
    request.docToSign.filename = Path.GetFileName(txt_filename.Text);
    request.docToSign.compressed = false;
    request.docToSign.data = File.ReadAllBytes(txt_filename.Text);

//Firma
    try {
        SgSingToolsWS.SignDocumentResponse response =
        (SgSingToolsWS.SignDocumentResponse)ws.ProcessMessage(request);
        File.WriteAllBytes(txt_filename.Text +
        ".fir",response.signedDoc.data);
    } catch (SoapException ex) {
        ...
        ...
    }

```

Método

sgdata.VerifyResponse **VerifyRequest**

Solicita al webService la autenticación de un documento. Los elementos que requiere para su funcionamiento son:

Argumento	Tipo	Detalle
documentoFirmado	sgdata.Document	Documento firmado que se desea autenticar.
folio	String	Número de folio para el documento. Este campo lo define el usuario de acuerdo a la lógica de sus aplicaciones. Para definirlo se puede utilizar el método setFolio() de la clase VerifyRequest.
documentoOriginal	sgdata.Document	Documento que fue firmado. En el caso de firmas sin contenido es necesario definir este dato. Para definirlo se puede utilizar el método setOriginalDoc() de la clase VerifyRequest.

Returns:

En una ejecución exitosa la instancia de la clase VerifyResponse que resulta contiene la secuencia del documento autenticado y se puede obtener con el método getSequence() de dicha clase.

Autenticación de una firma mediante el Web Service.

```
// Ejemplo de invocación del mensaje VerifyRequest

//Instancia al WebService
Uri myuri = new Uri(txt_url.Text);
bool ssl = myuri.Scheme.ToUpper() == "HTTPS";
SgSignToolsWSDemo.SgSingToolsWS.SgSignToolsWS ws = new
    SgSignToolsWSDemo.SgSingToolsWS.SgSignToolsWS();

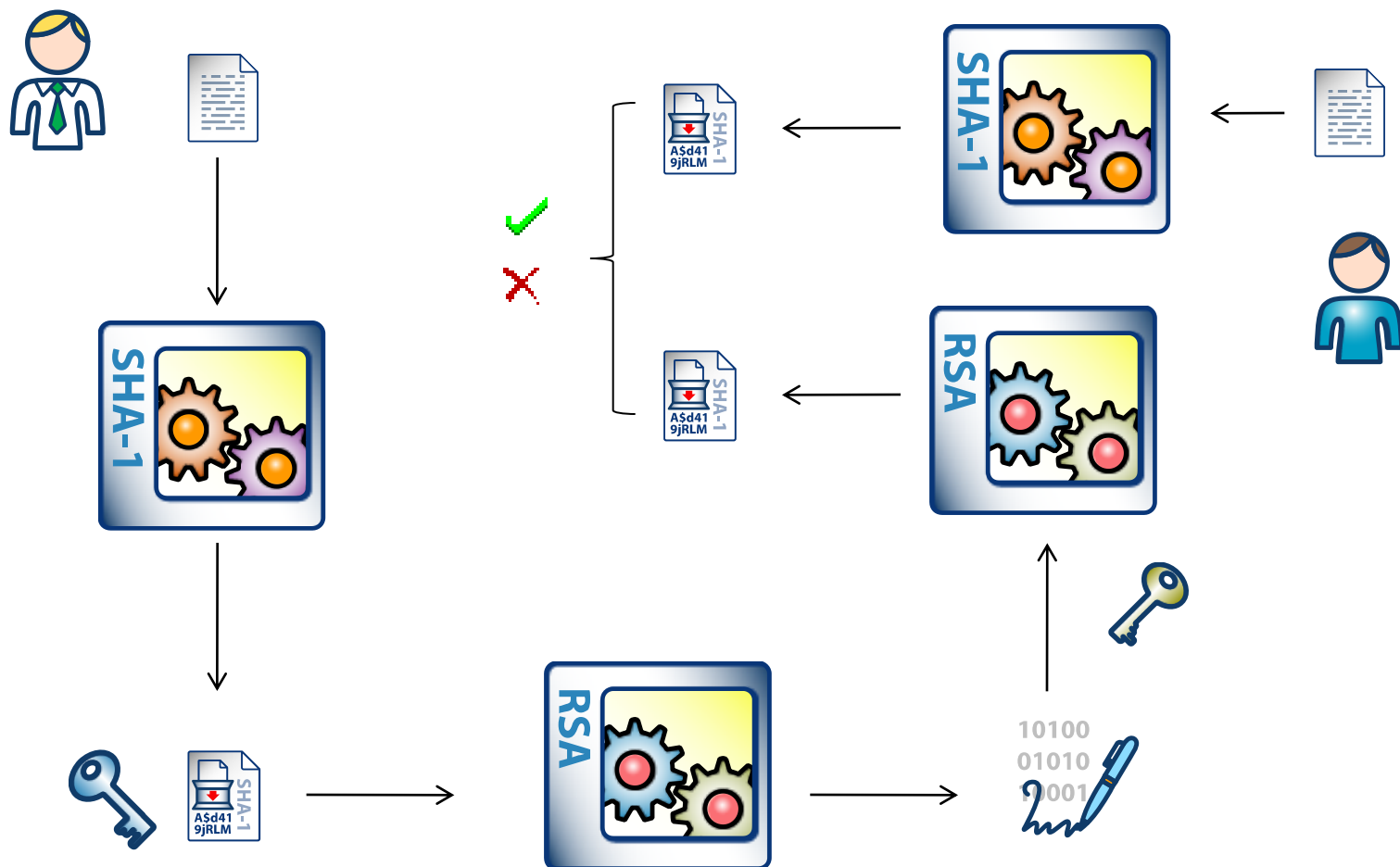
ws.Url = txt_url.Text;
ws.Credentials = new NetworkCredential(txt_user.Text, txt_pwd.Text);

//Requerimiento de verificación
SgSingToolsWS.VerifyRequest request = new SgSignToolsWSDemo.SgSingToolsWS.VerifyRequest();
request.signedDoc = new SgSignToolsWSDemo.SgSingToolsWS.Document();
request.signedDoc.compressed = false;
request.signedDoc.filename = Path.GetFileName(txt_filename.Text);
request.signedDoc.filename = request.signedDoc.filename.Substring(0,
    request.signedDoc.filename.Length-4);
request.signedDoc.data = File.ReadAllBytes(txt_filename.Text);

//Verificación
try
{
    SgSingToolsWS.VerifyResponse response =
        (SgSingToolsWS.VerifyResponse)ws.ProcessMessage(request);
    MessageBox.Show("ID=" + response.sequence);
} catch (SoapException ex){
    ...
}
}
```

Apéndice

Proceso de firma digital



PKCS#7 con contenido

```
30 82 05AE
 06 09
 2A 86 48 86 F7 0D 01 07 02          PKCS7 - SIGNED DATA
A0 82 059F
 30 82 059B
  02 01
  01
 31 0B
  30 09
    06 05
    2B 0E 03 02 1A          SHA-1 Hash Algorithm
    05 00
30 18
 06 09
 2A 86 48 86 F7 0D 01 07 01          PKCS7 - DATA
A0 0B
 04 09
  46 69 72 6D 61 64 6F 2E 2E      Firmado...
A0 82 03BA
 30 82 03B6
 30 82 029E
  A0 03
    02 01
    02
  02 14
    30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
    30 31 32 31
...
...
D0 77 9A CA 4F F4 DE 59 99 26 9F BE 45 8D AC AF
3A 86 96 95 1D 91 A4 07 74 FA AB 8D 98 26 1F FF
C7 09 EC 65 FB 2C 7D 6B 5E 3B 76 31 3F D8 0F 3A
CD C9 95 B9 A3 31 0D 61 C0 A8 8E C1 32 59 9D 19
51 D6 A9 F0 51 C6 9C 83 1F 7D E9 DE AC 99 5E 64
D1 D9 8B DB A9 5A 4E 89 FA A8 9D 15 FC BD 9A
```

PKCS#7 detached

```
30 82 05A1
 06 09
 2A 86 48 86 F7 0D 01 07 02          PKCS7 - SIGNED DATA
A0 82 0592
 30 82 058E
  02 01
  01
 31 0B
  30 09
    06 05
    2B 0E 03 02 1A          SHA-1 Hash Algorithm
    05 00
 30 0B
  06 09
  2A 86 48 86 F7 0D 01 07 01          PKCS7 - DATA
A0 82 03BA
 30 82 03B6
  30 82 029E
  A0 03
  02 01
  02
  02 14
  30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
  30 31 32 31
...
...
27 52 DD 15 64 E3 ED 2F 01 E7 70 CC E0 EA 12 9E
3B 62 C7 3F 6C EF 50 DF 9D 09 64 E7 C4 B8 61 43
6C 86 62 5C 03 FC C6 5E 50 08 21 CC 1E 75 14 84
DC 3E A8 A6 26 A7 F6 45 F1 2A 69 C3 1E D5 35 DF
38 18 43 7B F4 0C C6 7C F4 F9 D7 91 F8 22 4B 64
0E 60 64 9E 34 FB 03 97 D7 67 D0 1C A7 59 9A
```