certicámara.

Validez y seguridad jurídica electrónica

Componente Firma Digital en JAVA

Documento de especificación y contexto:

Versión 1.0

IDENTIFICADOR	Sign4J.pdf
NOMBRE DEL DOCUMENTO	Documento inicial de documentación.
ESTADO DEL DOCUMENTO	Aprobado
ÁREA	Consultoría
RESPONSABLES	**Andrés Felipe Escobar Fernández** (andres.escobar@certicamara.com) **Jhon Edgar Gonzalez Amortegui** (jhon.gonzalez@certicamara.com)
REVISORES	**Carlos Orlando Peña** (carlos.pena@certicamara.com)

CONTROL DE VERSIONES DEL DOCUMENTO

Versión	Fecha creación	Fecha liberación	Descripción cambio

CONTROL DE REVISIONES Y APROBACIONES

Revisado por	Firma	Fecha

Tabla de Contenidos

Documento de Especificación y contexto del Componente de Firma Digital en Java

1. INTRODUCCIÓN

Certicámara S.A. ha desarrollado un Componente API que permite firmar y/o cofirmar varios documentos en formato PKCS#7 Attached y Detached, así como en formatos XML Enveloped, Enveloping, Detached y archivos en formato PDF.

2. OBJETIVOS

- Documentar el uso de las funcionalidades y el uso del Componente de Firma Digital en Java.

3. VERSIÓN DEL SERVICIO

- 1.0

4. DUEÑOS DEL SERVICIO

Responsable	Certicámara – Ing. Carlos Peña.
Responsable	Área: Consultoría
Informado	

5. FUNCIONALIDADES DEL API.

Este componente API permite firmar digitalmente un documento con un certificado digital en formato p12. Los tipos de documentos que puede firmar este componente API son:

- PKCS#7 Attached
- PKCS#7 Detached
- PKCS#7 Attached Distribuido
- XML Enveloped
- XML Enveloping
- XML Detached
- PDF (PaDES)
- PDF (PaDES) Distribuido

Para firmar un documento se debe especificar en una clase de parámetros toda la configuración relacionada para firma, el certificado con el que se va firmar, bytes del documento a firmar, configuración para validar la revocación de la firma, y demás configuraciones propias para cada firma que se irán describiendo en el presente documento.

Para realizar una firma se debe crear un objeto ***Sign*** que en su contructor recibe una lista de la clase de propiedades, lo que quiere decir que con este componente es posible agrupar en una lista todos los documentos que se van a firmar y firmarlos todos en una iteración. Esta clase tiene un método *signData()* que permite firmar los documentos que están en la lista de parámetros y devuelve una respuesta por cada objeto de parámetros ingresado.

5.1. Creación de un objeto de firma

Para la creación de un objeto firma, se encapsuló la funcionalidad de creación de objetos firma en una clase llamada ***SignFactory*** que tiene un método estático *getSigner* que recibe un String que indica cual tipo de firmante se requiere (PKCS#7 Attached, XML Enveloped, etc.) y recibe un ArrayList de tipo de ***SignatureParameters*** que es en donde se encapsula toda la configuración para firmar un documento. El siguiente es un ejemplo para firmar una lista de signParameters de tipo PKCS#7.

```
Sign s = SignFactory.getSigner(SignFactory.PKCS7, signatureParameters);
s.signData();
```

5.1.1. Método *getSigner* de la clase *SignFactory*.

El método *getSigner* de la clase ***SignFactory*** recibe dos parámetros, es un String que indica el tipo de firma, podemos elegir entre diferentes atributos estáticos de la clase ***SignFactory***, estos atributos son:

Atributo	Valor	Tipo
PKCS7	pkcs7.PKCS7Sign	String
PKCS7Hash	pkcs7.PKCS7SignHash	String
XMLEnveloped	xml.XmlEnveloped	String
XMLEnveloping	xml.XmlEnveloping	String
XMLDetached	xml.XmlDetached	String
PDF	pdf.PdfSign	String

Para la lista de SignatureParameters se debe definir de la siguiente manera:

```
ArrayList<SignatureParameters> lista = new ArrayList<SignatureParameters>();
lista.add(signParameters);
```

En donde la variable *signParameters* es una implementación de la clase ***SignatureParameters***. Existen los siguientes tipos de implementaciones de ***SignatureParameters***:

Tipos de Firma	Implementación SignatureParameters
• **PKCS#7 Attached** • **PKCS#7 Detached** • **PKCS#7 Attached Hash**	PKCS7Parameters
• **XML Enveloped** • **XML Enveloping**	XMLParameters

• **XML Detached**	XMLDetachedParameters
• **PDF**	PDFParameters

Existen 3 atributos comunes para todos los ***SignatureParameters***, que son:

- **Byte[]** *bytesToSign*: Son los bytes del documento que se van a firmar.
- **RevocationVerify** *revocationVerify*: es la configuración para validar el certificado de firma y los certificados con los que ya se firmó el documento (si ya está firmado).
- **CertificateConfiguration** *certificateConf*: es la configuración para identificar el método con el que se va acceder al certificado en formato p12 con el que se va a firmar.

5.1.2. RevocationVerify.

El componente API de firma no solo firma digitalmente un documento sino también verifica el documento al finalizar la operación para validar que las firmas que hay en el documento estén válidas. Para generar un objeto ***RevocationVerify*** se debe ejecutar el siguiente código:

```
RevocationVerify p = new RevocationVerify("CRL_ONLY",new
GregorianCalendar(),"c:/temp/crl","http://oscp.certicamara.com");

p.setKeyStorePath("c:/temp/keystore/Keystore");
```

El constructor recibe 4 parámetros que son los siguientes en orden:

Atributo	Valor	Tipo
Tipo de verificación de Revocación	Tiene 4 posible valores: “CRL_ONLY” “OSCP_ONLY” “OSCP_CRL” “CRL_OSCP”	String
Calendar	Fecha para la cual se quiere validar la validez de la firma, si es null coje la fecha actual del sistema.	Calendar
Dirección CRL	Se indica la ruta absoluta de la carpeta en la cual se encuentran las crl.	String
Dirección OCSP	Se indica la dirección URL en donde se válida la revocación por OCSP.	String

Además de crear el objeto se le debe indicar la ruta absoluta en donde se encuentra el keystore en donde están almacenados los certificados públicos de las entidades de confianza, como el de Certicamara. Con esta configuración ya es posible validar los certificados con los que se realizaron la firma.

5.1.3. CertificateConfiguration

Esta también es una clase abstracta que tiene 3 tipos de implementaciones: cuando se accede al certificado con los bytes de este, cuando el certificado está adentro de un keystore o cuando se

tiene el certificado ya en formato X509. Cada implementación en su constructor recibe todos los parámetros como se presenta a continuación:

Tipos de configuración de certificado	Constructores
CertificateFromBytes	1. Byte[] *bytesCertificado*, String *passwordDelCertificado* 2. Byte[] *bytesCertificado*, String *passwordDelCertificado*, String *aliasDelCertificado*
CertificateFromKeystore	1. KeyStore *keystoreCertificado*, String *passwordDelCertificado* 2. KeyStore *keystoreCertificado*, String *passwordDelCertificado*, String *aliasDelCertificado*
CertificateFromX509	1. X509Certificate *certificadoX509*, PrivateKey *llavePrivada* 2. X509Certificate *certificadoX509*, PrivateKey *llavePrivada*, Certificate[] *cadenaDeCertificacion*

Un ejemplo de implementación:

```
CertificateConfiguration cert = new
CertificateFromBytes(UtilsSign.getBytesFromFile(p12Certificate),certificatePassw
ord);
```

En donde la variable *p12Certificate* es la ruta absoluta en donde se encuentra el archivo del certificado en formato p12, y la variable *certificatePassword* es el password del certificado con el que se va a firmar.

5.1.4. Respuesta del Sign: List<ProcessResponseSign>.

Como al firmar le pasamos una lista de parámetros de configuración de firma, para cada elemento de esa lista debe existir una respuesta. Para cada configuración existe una clase respuesta llamada ProcessResponseSign en donde tiene los siguientes elementos:

Atributo	Valor	Tipo
Exito	Es el booleano que indica si la operación de firma fue exitosa o no.	boolean
MessageResponse	Lista de mensajes que indica cuales fueron los problemas que se presentaron si ocurrió algún error a la hora de firmar. Cada mensaje tiene un código y un mensaje.	List<MessageResponse>
Resultado	Son los bytes del documento firmado si la operación fue exitosa, sino son null.	Byte[]

Un ejemplo sobre como iterar sobre este objeto:

```
Sign s = SignFactory.getSigner(SignFactory.PKCS7, list);

List<ProcessResponseSign> prs = s.signData();
for(ProcessResponseSign pp :prs){
	if(pp.isExito()){
	UtilsSign.saveSignedFile("c:/temp/pkcs.p7z", pp.getResultado());
	}else{
		for(MessageResponse mm : pp.getMessageResponse()){
		System.out.println(mm.getCodigo()+" "+mm.getMensaje());
		}
	}
}
```

El anterior fue un ejemplo en donde recorremos la lista de respuesta y si la repuesta fue satisfactoria, guardamos el archivo firmado generado en una ruta. Si la respuesta es inválida se imprime en consola el código del error y el mensaje con la descripción. Adicionalmente la clase ***MessageResponse*** tiene otro atributo que se llama *Trace*, en donde se indica la traza del error.

A continuación se presentara la forma en cómo se debe invocar el componente para realizar una firma de los tipos actualmente soportados por este componente API.

5.2. FIRMA EN FORMATO PKCS#7

A continuación se explica como se debe realizar implementación de un programa que realice firmas en este formato. En resumidas cuentas para desarrollar un fragmento de código que realice la firma en este formato debemos primero, crear un objeto PKCS7Parameters, en donde se guardan todos los parámetros que necesita la firma, y luego se construye un objeto Sign cuyo constructor recibe como parámetros la lista de objetos PKCS7Parameters y el tipo de firma que se quiere realizar, luego se llama al método ***signData***.

5.2.1. PKCS7Parameters

Esta es la clase de configuración para realizar una firma tanto en PKCS#7 Attached y Detached. El constructor de esta clase, como ya ha sido indicado, recibe:

- **Byte[]** *bytesToSign*: Son los bytes del documento que se van a firmar.
- **RevocationVerify** *revocationVerify*: es la configuración para validar el certificado de firma y los certificados con los que ya se firmó el documento (si ya está firmado para formato Attached).
- **CertificateConfiguration** *certificateConf*: es la configuración para identificar el método con el que se va acceder al certificado en formato p12 con el que se va a firmar.
- **Boolean** *attached*: es el booleano que indica si la firma es attached o detached.

Además tiene un atributo adicionar que se puede configurar; en las firmas Attached se le puede indicar cuál es el nombre del archivo y la extensión del archivo que se va a firmar, esto se indica puesto que a la hora de extraer el contenido de la firma no se sabe en qué formato pueda estar esos bytes que se firmaron. A continuación se presenta un ejemplo de configuración del

PKCS7Parameters:

```
PKCS7Parameters signParameters = new
PKCS7Parameters(UtilsSign.getBytesFromFile(fileToSignPath),
      revocationVerify,certificateConfiguration,true);
signParameters.setFileNameAndExtension(fileToSignName);
```

5.2.2. Firma en formato PKCS#7 Attached

Para realizar el firmado en formato PKCS#7 Attached se colocar le booleano del constructor de PKCS7Parameters en true. Un ejemplo de firma es el siguiente:

```
CertificateConfiguration certificateConfiguration = new
CertificateFromBytes(UtilsSign.getBytesFromFile(pkcs12Path),pkcs12Password);

RevocationVerify revocationVerify = new
RevocationVerify("OCSP_ONLY",null,null,null);
revocationVerify.setKeyStorePath(resourcesPath+"/keystore/Keystore");

TSAAuthentication autenticacion = new
TSACertAuthentication(UtilsSign.getBytesFromFile(certificatePathStamp),certifica
teStampPass);

TSAProperties tsaProperties = new TSAProperties(tsaURL, tsaPolicyOID,
hashAlgorithm, autenticacion);


PKCS7Parameters signParameters = new
PKCS7Parameters(UtilsSign.getBytesFromFile(fileToSignPath),
            revocationVerify,certificateConfiguration,true);
signParameters.setFileNameAndExtension(fileToSignName);
//Añadir estampa
signParameters.setTimeStampSettings(tsaProperties);
ArrayList<SignatureParameters> list = new ArrayList<SignatureParameters>();
list.add(signParameters);

Sign s = SignFactory.getSigner(SignFactory.PKCS7, list);

List<ProcessResponseSign> prs = s.signData();

for(ProcessResponseSign pp :prs){
      if(pp.isExito()){
      UtilsSign.saveSignedFile("c:/temp/pkcs7.p7z",pp.getResultado());
      }else{
      for(MessageResponse mm : pp.getMessageResponse()){
            System.out.println(mm.getCodigo()+" "+mm.getMensaje());
      }
      }
}
```

5.2.3. Firma en formato PKCS#7 Detached

Para realizar el firmado en formato PKCS#7 Attached se debe colocar el booleano del constructor de PKCS7Parameters en false. Un ejemplo de firma es el siguiente:

```
CertificateConfiguration certificateConfiguration = new
CertificateFromBytes(UtilsSign.getBytesFromFile(pkcs12Path),pkcs12Password);

RevocationVerify revocationVerify = new
RevocationVerify("OCSP_ONLY",null,null,null);
revocationVerify.setKeyStorePath(resourcesPath+"/keystore/Keystore");

PKCS7Parameters signParameters = new
PKCS7Parameters(UtilsSign.getBytesFromFile(fileToSignPath),
            revocationVerify,certificateConfiguration,false);
signParameters.setFileNameAndExtension(fileToSignName);

ArrayList<SignatureParameters> list = new ArrayList<SignatureParameters>();
list.add(signParameters);

Sign s = SignFactory.getSigner(SignFactory.PKCS7, list);

List<ProcessResponseSign> prs = s.signData();

for(ProcessResponseSign pp :prs){
      if(pp.isExito()){
      UtilsSign.saveSignedFile("c:/temp/pkcs7D.p7s", pp.getResultado());
      }else{
      for(MessageResponse mm : pp.getMessageResponse()){
            System.out.println(mm.getCodigo()+" "+mm.getMensaje());
      }
      }
}
```

5.2.4. Firma en formato PKCS#7 Attached Distribuido

Para realizar el firmado en formato PKCS#7 Attached Distribuido se debe indicar en el SignFactory que el tipo de firma es PKCS7Hash y hacer uso del Applet de firma. Un ejemplo de firma es el siguiente:

```
String hashesPkcs7 = null;
PKCS7SignHash s2 = null;
try{
      archivoSignPath1 = pathResources + "/dataIn/Archivo_Importante.txt";
      fileToSignName = "ArchivoImportáanñññtesfsdfdfsdf.txt";
```

```
		RevocationVerify revocationVerify = new RevocationVerify("CRL_ONLY", null,
		"C:/CRL/crl", null);
		revocationVerify.setKeyStorePath(pathResources+"/keystore/Keystore");
		CertificateConfiguration cert = new CertificateConfiguration() {
		@Override
			public void validate() throws CertificateInitException {
			}
		};
		ArrayList<SignatureParameters> listaPkcs7 = new
		ArrayList<SignatureParameters>();
		for(int i=0; i<1; i++){
			PKCS7Parameters signParameterspkcs7 = new
		PKCS7Parameters(UtilsSign.getBytesFromFile(archivoSignPath1),
		revocationVerify, cert, true);
			if(isFilenameValid(fileToSignName)){
				signParameterspkcs7.setFileNameAndExtension(fileToSignName);
			}
			else{
				signParameterspkcs7.setFileNameAndExtension("defaultName.dat");
			}
			listaPkcs7.add(signParameterspkcs7);
		}
		s2 = (PKCS7SignHash) SignFactory.getSigner(SignFactory.PKCS7Hash, listaPkcs7);
		hashesPkcs7 = s2.getHashs();
}
catch(Exception e){
		e.printStackTrace();
		request.setAttribute("hash",e.toString());
}
ArrayList<Sign> lista = new ArrayList<Sign>();
lista.add(s2);
session.setAttribute("hash", hashesPkcs7);
session.setAttribute("ListaHashSigned", lista);
```

En donde lo primero que debemos hacer es obtener el hash de los documentos que se van a firmar.

```
function setAppletText() {
		var applet = document.getElementById('appletfirma');
		var timediv = document.getElementById('txtHash');
		applet.Sign(timediv.value, "archivoDist", false);
}

function setSign(value){
		var field = document.getElementById('Firma');
		field.value = value;
}
```

Luego obtenemos la firma de los documentos por medio del Applet de Firma.

```
ArrayList<Sign> lista = (ArrayList<Sign>) session.getAttribute("ListaHashSigned");
PKCS7SignHash s2 = (PKCS7SignHash)lista.get(1);
String[] hashesFirmados = request.getParameter("Firma").split("%");
String hashesFirmadosPkcs7 = "";
for(int i=0;i<s2.signatureParameters.size(); i++){
	hashesFirmadosPkcs7 += hashesFirmados[i]+"%";
}
List<ProcessResponseSign> listaRPkcs7 = new ArrayList<ProcessResponseSign>();
try {
	listaRPkcs7.addAll(s2.joinPKCS7Signatures(hashesFirmadosPkcs7));
}
catch (SignException e) {
	e.printStackTrace();
}
k = 0;
for(ProcessResponseSign processResponseSign: listaRPkcs7){
	if(processResponseSign.isExito()){
	//Ejemplo para descargar el archivo firmado.
		session.removeAttribute("isHash");
		verificarPkcs7(listaRPkcs7.get(k).getResultado());
	//response.setContentType("application/pkcs7-mime");
	//response.setHeader("Content-Disposition","attachment; filename =
	"+"ArchivoFirmado.p7z");
	//ServletOutputStream op = response.getOutputStream();
	//.write(listaRPkcs7.get(k).getResultado());
	//op.flush();
	//op.close();
		UtilsSign.saveSignedFile("C:/Users/jhon.triana/Desktop/prueba/prueba"+k+
	".p7z",listaRPkcs7.get(k).getResultado() );
		k++;
	}
	else{
		k++;
		String error="";
		for(MessageResponse messageResponse:
processResponseSign.getMessageResponse()){
			error += ("Error al firmar documentos: Código:
"+messageResponse.getCodigo()+" Mensaje: "+messageResponse.getMensaje());
		}
		session.setAttribute("error", error);
		response.sendRedirect("index.jsp");
	}
}
```

Como último paso unimos la firma con el documento original

5.3. FIRMA EN FORMATO XML

A continuación se explica cómo se debe realizar implementación de un programa que realice firmas en este formato. En resumidas cuentas para desarrollar un fragmento de código que realice la firma en este formato debemos primero, crear un objeto ***XmlParameters***, en donde se guardan todos los parámetros que necesita la firma, y luego se construye un objeto Sign cuyo constructor recibe como parámetros la lista de objetos ***XmlParameters*** y el tipo de firma que se quiere realizar, luego se llama al método ***signData***.

5.3.1. XMLParameters

Esta es la clase de configuración para realizar una firma tanto en XML enveloped y enveloping. El constructor de esta clase, como ya ha sido indicado, recibe:

- **Byte[]** *bytesToSign*: Son los bytes del documento que se van a firmar.
- **RevocationVerify** *revocationVerify*: es la configuración para validar el certificado de firma y los certificados con los que ya se firmó el documento
- **CertificateConfiguration** *certificateConf*: es la configuración para identificar el método con el que se va acceder al certificado en formato p12 con el que se va a firmar.
- **String** *tagToSign*: es el string que indica cual es el tag con el id a firmar, si el string es vacío se firma todo el documento.
  Si se requiere firmar un tag debe tener la siguiente forma el XML:
  <firma id="tagFirma">Firma XML</firma>
  El String de *tagToSign* debe decir tagFirma puesto que pueden existir varios tag firma y se debe indicar cual tag específicamente se debe firmar.

A continuación se presenta un ejemplo de configuración del XmlParameters:

```
XmlParameters xmlParameters = new XmlParameters
(UtilsSign.getBytesFromFile(fileToSignPath),
      revocationVerify,certificateConfiguration, tagToSign);
```

5.3.2. Firma en formato XML Enveloped

Para realizar el firmado en formato XML Enveloper se debe indicar en el SignFactory que el tipo de firma es XML Enveloped. Un ejemplo de firma es el siguiente:

```
CertificateConfiguration certificateConfiguration = new
CertificateFromBytes(UtilsSign.getBytesFromFile(pkcs12Path),pkcs12Password);

RevocationVerify revocationVerify = new
RevocationVerify("OCSP_ONLY",null,null,null);
revocationVerify.setKeyStorePath(resourcesPath+"/keystore/Keystore");

XmlParameters xmlParameters = new XmlParameters
(UtilsSign.getBytesFromFile(fileToSignPath),
      revocationVerify,certificateConfiguration, tagToSign);

ArrayList<SignatureParameters> list = new ArrayList<SignatureParameters>();
list.add(xmlParameters);
```

```
Sign s = SignFactory.getSigner(SignFactory.XMLEnveloped, list);

List<ProcessResponseSign> prs = s.signData();

for(ProcessResponseSign pp :prs){
	if(pp.isExito()){
	UtilsSign.saveSignedFile("c:/temp/enveloped.xml",pp.getResultado());
	}else{
	for(MessageResponse mm : pp.getMessageResponse()){
		System.out.println(mm.getCodigo()+" "+mm.getMensaje());
	}
	}
}
```

5.3.3. Firma en formato XML Enveloping

Para realizar el firmado en formato XML Enveloping se debe indicar en el SignFactory que el tipo de firma es XML Enveloping. Un ejemplo de firma es el siguiente:

```
CertificateConfiguration certificateConfiguration = new
CertificateFromBytes(UtilsSign.getBytesFromFile(pkcs12Path),pkcs12Password);

RevocationVerify revocationVerify = new
RevocationVerify("OCSP_ONLY",null,null,null);
revocationVerify.setKeyStorePath(resourcesPath+"/keystore/Keystore");

XmlParameters xmlParameters = new XmlParameters
(UtilsSign.getBytesFromFile(fileToSignPath),
	revocationVerify,certificateConfiguration, tagToSign);

ArrayList<SignatureParameters> list = new ArrayList<SignatureParameters>();
list.add(xmlParameters);

Sign s = SignFactory.getSigner(SignFactory.XMLEnveloping, list);

List<ProcessResponseSign> prs = s.signData();

for(ProcessResponseSign pp :prs){
	if(pp.isExito()){
	UtilsSign.saveSignedFile("c:/temp/enveloping.xml",pp.getResultado());
	}else{
	for(MessageResponse mm : pp.getMessageResponse()){
		System.out.println(mm.getCodigo()+" "+mm.getMensaje());
	}
	}
}
```

5.4. FIRMA EN FORMATO XML Detached

A continuación se explica cómo se debe realizar implementación de un programa que realice firmas en este formato. En resumidas cuentas para desarrollar un fragmento de código que realice la firma en este formato debemos primero, crear un objeto ***XmlDetachedParameters***, en donde se guardan todos los parámetros que necesita la firma, y luego se construye un objeto Sign cuyo constructor recibe como parámetros la lista de objetos ***XmlDetachedParameters*** y el tipo de firma que se quiere realizar, luego se llama al método ***signData***.

5.4.1. XMLDetachedParameters

Esta es la clase de configuración para realizar una firma XML Detached. El constructor de esta clase, como ya ha sido indicado, recibe:

- **Byte[]** *bytesToSign*: Son los bytes del documento que se van a firmar.
- **RevocationVerify** *revocationVerify*: es la configuración para validar el certificado de firma y los certificados con los que ya se firmó el documento
- **CertificateConfiguration** *certificateConf*: es la configuración para identificar el método con el que se va acceder al certificado en formato p12 con el que se va a firmar.
- **String** *uriXML*: es el string que indica la ruta absoluta del documento a firmar (de tipo file:///) un ejemplo de una uriXML es:
  "file:///C:/Temp/docs/ArchivoXML.xml"

A continuación se presenta un ejemplo de configuración del XmlDetachedParameters:

```
XmlDetachedParameters xmlParameters = new XmlDetachedParameters
(UtilsSign.getBytesFromFile(fileToSignPath),
      revocationVerify,certificateConfiguration, uriXML);
```

5.4.2. Firma en formato XML Detached

Para realizar el firmado en formato XML Detached se debe indicar en el SignFactory que el tipo de firma es XML Detached. Un ejemplo de firma es el siguiente:

```
CertificateConfiguration certificateConfiguration = new
CertificateFromBytes(UtilsSign.getBytesFromFile(pkcs12Path),pkcs12Password);

RevocationVerify revocationVerify = new
RevocationVerify("OCSP_ONLY",null,null,null);
revocationVerify.setKeyStorePath(resourcesPath+"/keystore/Keystore");

XmlDetachedParameters xmlParameters = new XmlDetachedParameters
(UtilsSign.getBytesFromFile(fileToSignPath),
      revocationVerify,certificateConfiguration, uriXML);

ArrayList<SignatureParameters> list = new ArrayList<SignatureParameters>();
list.add(xmlParameters);
```

```
Sign s = SignFactory.getSigner(SignFactory.XMLDetached, list);

List<ProcessResponseSign> prs = s.signData();

for(ProcessResponseSign pp :prs){
	if(pp.isExito()){
	UtilsSign.saveSignedFile("c:/temp/detached.xml",pp.getResultado());
	}else{
	for(MessageResponse mm : pp.getMessageResponse()){
		System.out.println(mm.getCodigo()+" "+mm.getMensaje());
	}
	}
}
```

5.5. FIRMA EN FORMATO PDF

A continuación se explica cómo se debe realizar implementación de un programa que realice firmas en este formato. En resumidas cuentas para desarrollar un fragmento de código que realice la firma en este formato debemos primero, crear un objeto ***PDFParameters***, en donde se guardan todos los parámetros que necesita la firma, y luego se construye un objeto Sign cuyo constructor recibe como parámetros la lista de objetos ***PDFParameters*** y el tipo de firma que se quiere realizar, luego se llama al método ***signData***.

5.5.1. PDFParameters

Esta es la clase de configuración para realizar una pdf. El constructor de esta clase, como ya ha sido indicado, recibe:

- **Byte[]** *bytesToSign*: Son los bytes del documento que se van a firmar.
- **RevocationVerify** *revocationVerify*: es la configuración para validar el certificado de firma y los certificados con los que ya se firmó el documento
- **CertificateConfiguration** *certificateConf*: es la configuración para identificar el método con el que se va acceder al certificado en formato p12 con el que se va a firmar.

Adicionalmente, en el PDF se puede configurar otros atributos adicionales como para cifrarlo, estamparlo, etc. Que se describirán a continuación.

- Para añadir información de la firma, como lo es la razón de la firma y la localización de la firma, se debe llamar al método *setInformation* e ingresar 2 strings que indican la razón y la localización de firma:

```
signParameters.setInformation(signReason, signLocation);
```

- Para añadir imagen a la firma o un indicativo visual de la firma se debe llamar al método *setImageSettings* de la siguiente manera:

```
signParameters.setImageSettings(signFieldName,
UtilsSign.getBytesFromFile(pdf2SignImagePath),
new Rectangle(lowerLeftX,lowerLeftY, upperRightX,upperRightY),
signPage,
imageValidation,
contentSignature,
```

```
RenderingMode.DESCRIPTION);
```

En donde el orden de los parámetros son:

- ✓ signFielName: es un String con el nombre de la firma. Este campo debe ser único para cada firma.
- ✓ Los bytes de la imagen que puede tener la firma. Si no se quiere ingresar imagen se puede enviar en null este campo.
- ✓ Se debe crear un Rectangle que indica en qué posición del pdf va ir la imagen de la firma. Los datos que recibe esta clase son enteros positivos. Entonces los 2 primeros enteros son la coordenada de la esquina inferior derecha y los siguientes 2 enteros es la coordeanda de la derecha superior izquierda.
- ✓ Se debe también indicar la página en donde va ir la firma. Debe ser un entero positivo y menor al número máximo de páginas del Pdf.
- ✓ imageValidation: es un booleano que indica si en la imagen debe salir la validación de la firma que muestra el Adobe Reader.
- ✓ contentSignature: es un texto, si se requiere si no se deja un String vacio, que sale en la posición de la firma. Este texto se adapta a la dimensión del rectángulo que se indicó.
- ✓ RenderingMode: es el tipo de impresión de la firma. Actualmente exite:
  - DESCRIPTION : pone solo la información en la variable contentSignature.
  - NAME_DESCRIPTION : pone información relevante sobre la firma.

➢ Para firmar un documento por partes se debe realizar de la siguiente forma:

```
signParameters.setImageSettings(signFieldName,
UtilsSign.getBytesFromFile(pdf2SignImagePath),
new Rectangle(lowerLeftX,lowerLeftY, upperRightX,upperRightY),
signPage,
imageValidation,
contentSignature,
RenderingMode.DESCRIPTION);

signParameters.setPdfByParts(true);
```

En donde dicho documento debe contener previamente los campos de firma, la primera función, setImageSettings, se usa para indicar el campo del documento (signFieldName) que se va a firmar y la segunda función, setPdfByParts para que el documento sea firmado por partes.

➢ Para añadir cifrar el documento con una clave se debe realizar de la siguiente forma:

```
signParameters.setEncryption(permissions, password, strench);
```

En donde el orden de parámetros es:

- ✓ Permissions: los permisos del PDF.
- ✓ Password: es el password con el que se va a cifrar el PDF.
- ✓ Strench: es un boolean que indica si el algoritmo para cifrar es de 128bits o no (de 64 bits).

Es de aclarar que no se puede cofirmar un PDF.

➢ Para añadir una estampa al documento se debe realizar de la siguiente forma:

```
TSAAuthentication autenticacion = new
TSACertAuthentication(UtilsSign.getBytesFromFile(certificatePathStamp),
certificateStampPass);

TSAProperties tsaProperties = new TSAProperties(tsaURL, tsaPolicyOID,
hashAlgorithm, autenticacion);

signParameters.setTimeStampSettings(tsaProperties);
```

En donde el objeto ***TSAAuthentication*** es la clase que indica cómo se va autenticar frente a la TSA, existen 3 implementaciones: por Certificado, por Usuario y Password y sin ningún tipo de autenticación. Lo siguiente es crear un ***TSAProperties*** en donde se le indica:

- ✓ La dirección URL en donde esta publicada la TSA.
- ✓ La política OID de la TSA. Puede ser vacía.
- ✓ El tipo de algoritmo con el que se le va sacar el hash (SHA-1, SHA-256, SHA-512, etc.).
- ✓ Y el TSAAuthentication.

Finalmente se agrega al ***PDFParameters*** la configuración de la TSA.

- ➢ Para añadir verificación LTV se debe realizar de la siguiente forma:

```
TSAAuthentication autenticacion = new
TSACertAuthentication(UtilsSign.getBytesFromFile(certificatePathStamp),
certificateStampPass);

TSAProperties tsaProperties = new TSAProperties(tsaURL, tsaPolicyOID,
hashAlgorithm, autenticacion);

RevocationVerify revocationVerify = new
RevocationVerify("CRL_ONLY",null,"C:/Temp/CRLS/",null);

String tsaCertificates = pathResources + "/TSAs/";
signParameters.setLtv(tsaCertificates, tsaProperties, revocationVerify);
```

En donde el objeto ***TSAAuthentication*** es la clase que indica cómo se va autenticar frente a la TSA, existen 3 implementaciones: por Certificado, por Usuario y Password y sin ningún tipo de autenticación. Lo siguiente es crear un ***TSAProperties*** en donde se le indica:

- ✓ La dirección URL en donde esta publicada la TSA.
- ✓ La política OID de la TSA. Puede ser vacía.
- ✓ El tipo de algoritmo con el que se le va sacar el hash (SHA-1, SHA-256, SHA-512, etc.).
- ✓ Y el TSAAuthentication.

Este objeto se utiliza para realizar la estampa de tiempo de la CRL. El objeto ***TSAProperties*** puede ser el mismo objeto que se definió en la estampa del documento.
Además se debe indicar la ruta del acceso de la CRL con el objeto ***RevocationVerify*** que también puede ser el mismo definido en el constructor del ***PDFParameters***.
Finalmente se agrega al ***PDFParameters*** la configuración de la TSA y el tipo de revocación.

- Para añadir la información biométrica para firma electrónica se debe realizar de la siguiente forma:

```
/*--Se recomienda crear un XML con los datos biométricos usados para realizar la
firma electrónica--*/
Arguments listaArgumentos = new Arguments();
listaArgumentos.add(new Argument("Nombres", "Nombres Persona"));
listaArgumentos.add(new Argument("Apellidos", "Apellidos Persona"));
listaArgumentos.add(new Argument("Dedo", "2")); /*Aplica para firma con huella*/
byte[] xmlFirmaElectronica = UtilsSign.CrearXmlInformacionBiometrica("1(tipo de
biometría a usar)", listaArgumentos, "Base64 del dato biométrico (huella, voz,
etc)");

/*--Se recomienda cifrar el XML con los datos biométricos usados para realizar
la firma electrónica--*/
byte[] xmlFirmaElectronicaCifrado = crypt.crypData(xmlFirmaElectronica,
cert.getCertificateX509.getEncoded());

/*--Se debe crear una lista que contendrá la información biométrica usada--*/
ArrayList<byte[]> listaDatosBiometricos = new ArrayList<byte[]>();
listaDatosBiometricos.add(xmlFirmaElectronicaCifrado);
/*--Nota: una firma electrónica puede contener más de un tipo de datos
biométricos, cada tipo puede agregarse a la lista de datos biométricos
listaDatosBiometricos.add(xmlDatosBiometricos); --*/

/*--Finalmente se asigna la lista al objeto de firma signParameters --*/
signParameters.setBiometricData(listBiometricdata);
```

A continuación se presenta un ejemplo de configuración del PdfParameters:

```
PdfParameters signParameters = new
PdfParameters(UtilsSign.getBytesFromFile(pdf2SignPath),revocationVerify,cert);

signParameters.setInformation(signReason, signLocation);

signParameters.setImageSettings(signFieldName,
UtilsSign.getBytesFromFile(pdf2SignImagePath),
new Rectangle(lowerLeftX,lowerLeftY,upperRightX,upperRightY),signPage,
false, contentSignature, RenderingMode.DESCRIPTION);

signParameters.setTimeStampSettings(tsaProperties);

signParameters.setEncryption(permissions, password, strench);

signParameters.setLtv(tsaProperties, revocationVerify);

signParameters.setBiometricData(listBiometricdata);
```

5.5.2. Firma en formato PDF

Para realizar el firmado en formato PDF se debe indicar en el SignFactory que el tipo de firma es PDF. Un ejemplo de firma es el siguiente:

```
CertificateConfiguration cert = new
CertificateFromBytes(UtilsSign.getBytesFromFile(p12Certificate),certificatePassw
ord);

RevocationVerify revocationVerify = new
RevocationVerify("CRL_ONLY",null,"C:/Temp/CRLS/",null);
revocationVerify.setKeyStorePath(pathResources+"c:/Temp/keystore/Keystore");

TSAAuthentication autenticacion = new
TSACertAuthentication(UtilsSign.getBytesFromFile(certificatePathStamp),certifica
teStampPass);

TSAProperties tsaProperties = new TSAProperties(tsaURL, tsaPolicyOID,
hashAlgorithm, autenticacion);

Arguments listaArgumentos = new Arguments();
listaArgumentos.add(new Argument("Nombres", "Nombres Persona"));
listaArgumentos.add(new Argument("Apellidos", "Apellidos Persona"));
listaArgumentos.add(new Argument("Dedo", "2")); /*Aplica para firma con huella*/
byte[] xmlFirmaElectronica = UtilsSign.CrearXmlInformacionBiometrica("1(tipo de
biometría a usar)", listaArgumentos, "Base64 del dato biométrico (huella, voz,
etc)");

byte[] xmlFirmaElectronicaCifrado = crypt.crypData(xmlFirmaElectronica,
cert.getCertificateX509.getEncoded());

ArrayList<byte[]> listaDatosBiometricos = new ArrayList<byte[]>();
listaDatosBiometricos.add(xmlFirmaElectronicaCifrado);

PdfParameters signParameters = new
PdfParameters(UtilsSign.getBytesFromFile(pdf2SignPath),revocationVerify,cert);

signParameters.setInformation(signReason, signLocation);

signParameters.setImageSettings(signFieldName,
UtilsSign.getBytesFromFile(pdf2SignImagePath),
new Rectangle(lowerLeftX,lowerLeftY,upperRightX,upperRightY),signPage,
false, contentSignature, RenderingMode.DESCRIPTION);

signParameters.setPdfByParts(true);

signParameters.setTimeStampSettings(tsaProperties);

signParameters.setEncryption(permissions, password, strench);
```

```
String tsaCertificates = pathResources + "/TSAs/";
signParameters.setLtv(tsaCertificates, tsaProperties, revocationVerify);
signParameters.setBiometricData(listBiometricdata);

ArrayList<SignatureParameters> lista = new ArrayList<SignatureParameters>();

lista.add(signParameters);

Sign s = SignFactory.getSigner(SignFactory.PDF, lista);

List<ProcessResponseSign> listaR = s.signData();

for(ProcessResponseSign pp :listaR){
	if(pp.isExito()){

	UtilsSign.saveSignedFile("C:/temp/pdf.pdf", pp.getResultado());

	}else{
		for(MessageResponse mm : pp.getMessageResponse()){
			System.out.println(mm.getCodigo()+" "+mm.getMensaje());
		}
	}

}
```

5.5.3. Firma en formato PDF Distribuido

Para realizar el firmado en formato PDF Distribuido se debe indicar en el SignFactory que el tipo de firma es PDF y hacer uso del Applet de firma. Un ejemplo de firma es el siguiente:

```
String hashesPdf = null;
PdfSign s1 = null;
try{
	archivoSignPath2 = pathResources + "/dataIn/Documento_SinFirmar.pdf";
	RevocationVerify revocationVerify = new RevocationVerify("CRL_ONLY", null,
	"C:/CRL/crl", null);
	revocationVerify.setKeyStorePath(pathResources+"/keystore/Keystore");
	CertificateConfiguration cert = new CertificateConfiguration() {
	@Override
		public void validate() throws CertificateInitException {
		}
	};
	TSAAuthentication autenticacion = new
	TSACertAuthentication(UtilsSign.getBytesFromFile(certificatePathStamp),
	certificateStampPass);
	TSAProperties tsaProperties = new TSAProperties(tsaURL, tsaPolicyOID,
```

```
	hashAlgorithm, autenticacion);
	ArrayList<SignatureParameters> listaPdf = new
	ArrayList<SignatureParameters>();
	for(int i=0; i<1; i++){
		PdfParameters signParameterspdf = new
	PdfParameters(UtilsSign.getBytesFromFile(archivoSignPath2), revocationVerify,
	cert);
		signParameterspdf.setInformation(signReason, signLocation);
			signParameterspdf.setImageSettings(signFieldName,
		UtilsSign.getBytesFromFile(pdf2SignImagePath), new Rectangle(lowerLeftX,
	lowerLeftY, upperRightX, upperRightY), signPage, false, contentSignature,
	RenderingMode.DESCRIPTION);
		signParameterspdf.setTimeStampSettings(tsaProperties);
		listaPdf.add(signParameterspdf);
	}
	s1 = (PdfSign) SignFactory.getSigner(SignFactory.PDF, listaPdf);
	hashesPdf = s1.getHashs();
}
catch(Exception e){
	e.printStackTrace();
	request.setAttribute("hash",e.toString());
}
ArrayList<Sign> lista = new ArrayList<Sign>();
lista.add(s1);
session.setAttribute("hash", hashesPdf);
session.setAttribute("ListaHashSigned", lista);
```

En donde lo primero que debemos hacer es obtener el hash de los documentos que se van a firmar.

```
function setAppletText() {
	var applet = document.getElementById('appletfirma');
	var timediv = document.getElementById('txtHash');
	applet.Sign(timediv.value, "archivoDist", false);
}

function setSign(value){
	var field = document.getElementById('Firma');
	field.value = value;
}
```

Luego obtenemos la firma de los documentos por medio del Applet de Firma.

```
ArrayList<Sign> lista = (ArrayList<Sign>) session.getAttribute("ListaHashSigned");
PdfSign s1 = (PdfSign)lista.get(0);
String[] hashesFirmados = request.getParameter("Firma").split("%");
String hashesFirmadosPdf = "";
for(int i=0;i<s1.signatureParameters.size(); i++){
	hashesFirmadosPdf += hashesFirmados[i]+"%";
}
```

```
List<ProcessResponseSign> listaRPdf = new ArrayList<ProcessResponseSign>();
try {
	listaRPdf.addAll(s1.joinPDFSignatures(hashesFirmadosPdf));
}
catch (SignException e) {
	e.printStackTrace();
}
int k=0;
//Se recupera la lista en donde esta la configuracion de los hashes
for(ProcessResponseSign processResponseSign: listaRPdf){
	if(processResponseSign.isExito()){
		verificarPdf(listaRPdf.get(k).getResultado());
	//Ejemplo para descargar el archivo firmado.
		session.removeAttribute("isHash");
	//response.setContentType(" application/pdf ");
	//response.setHeader("Content-Disposition","attachment; filename =
	"+"ArchivoFirmado.pdf");
	//ServletOutputStream op = response.getOutputStream();
	//op.write(listaRPdf.get(k).getResultado());
	//op.flush();
	//op.close();

		UtilsSign.saveSignedFile("C:/Users/jhon.triana/Desktop/prueba/prueba" +
k + ".pdf",listaRPdf.get(k).getResultado() );
		k++;
	}
	else{
		k++;
		String error="";
		for(MessageResponse messageResponse:
processResponseSign.getMessageResponse()){
			error += ("Error al firmar documentos: Código:
"+messageResponse.getCodigo()+" Mensaje: "+messageResponse.getMensaje());
		}
		session.setAttribute("error", error);
		response.sendRedirect("index.jsp");
	}
}
```

Como último paso unimos la firma con el documento original

6. Precondiciones

- Se debe ejecutar en una plataforma java 1.6 en adelante.
- Se deben tener permisos de lectura sobre los archivos que se quieren firmar.
- Para realizar la co-firma de un archivo en cualquier formato, todas las firmas anteriores deben ser validas, de lo contrario el componente no dejará realizar la cofirma.
- Se deben contar con certificados válidos para realizar el proceso de firma.

7. Excepciones Presentadas

Errores presentados al momento de la firma:

```
# Rango 100 relacionado con la inicialización del componente:
S0100=Los par\u00E1metros de configuraci\u00F3n del certificado
ingresados son inv\u00E1lidos.
S0101=Error al inicializar la configuraci\u00F3n del certificado\:
S0102=Los par\u00E1metros de configuraci\u00F3n del PKCS7Parameters
ingresados son inv\u00E1lidos.
S0103=Los par\u00E1metros de configuraci\u00F3n del XmlParameters
ingresados son inv\u00E1lidos.
S0104=Los par\u00E1metros de configuraci\u00F3n del
XmlDetachedParameters ingresados son inv\u00E1lidos.
S0105=Los par\u00E1metros de configuraci\u00F3n del PdfParameters
ingresados son inv\u00E1lidos.
S0106=Alg\u00FAn elemento de la lista de par\u00E1metros ingresada, no
corresponde al tipo de firma que se quiere realizar.
S0107=No ha ingresado par\u00E1metros de configuraci\u00F3n de la
firma.


# Rango 300 relacionado la firma PKCS7:
S0300=No existe el proveedor de seguridad BouncyCastle.
S0301=Error al tratar de firmar con el certificado con el proveedor de
seguridad.
S0302=Error al tratar de a\u00F1adir el nombre del archivo a la firma.
S0303=Error al tratar de a\u00F1adir la llave p\u00FAblica a la firma.
S0304=Ocurrio un error al tratar de firmar el documento.
S0305=Ocurrio un error al tratar de obtener los bytes del documento
firmado.

# Rango 400 relacionado la firma XML Enveloped:
S0400=Error al configurar el algoritmo de hash en la firma xml
enveloped.
S0401=Error al configurar la transformaci\u00F3n del xml en formato
enveloped.
S0402=Error al configurar la canocalizaci\u00F3n del xml en formato
enveloped.
S0403=Error al configurar del formato de la firma xml en formato
enveloped.
S0404=Error al configurar el DocumentBuilder para la generaci\u00F3n
del xml.
S0405=Error al en el proceso de generación de la firma enveloped.
S0406=Error al configurar el marshal para realizar el xml.
S0407=Error al transformar la firma en formato xml enveloped.
S0408=Los bytes ingresados no corresponden a un documento xml.
# Rango 500 relacionado la firma XML Enveloping:
S0500=Error al configurar el algoritmo de hash en la firma xml
enveloped.
S0501=Error al configurar la canocalizaci\u00F3n del xml en formato
enveloped.
S0502=Error al configurar del formato de la firma xml en formato
```

```
enveloped.
S0503=Error al configurar el DocumentBuilder para la generaci\u00F3n
del xml.
S0504=Error al en el proceso de generación de la firma enveloped.
S0505=Error al configurar el marshal para realizar el xml.
S0506=Error al transformar la firma en formato xml enveloped.
S0507=Los bytes ingresados no corresponden a un documento xml.

# Rango 600 relacionado la firma XML Detached:


# Rango 700 relacionado la firma PDF:
S0700=Problemas al leer el pdf. Los bytes del documento no
corresponden a un archivo PDF o el archivo esta corrupto.
S0701=No es permitido a\u00F1adirle informaci\u00F3n al archivo PDF a
firmar.
S0702=Error al unir la verificaci\u00F3n LTV con el archivo PDF.
S0703=Error al unir la configuraci\u00F3n ingresada con el archivo
PDF\:
S0704=No se encuentra el algoritmo para extraer el hash al archivo
PDF.
S0705=Error al extraer el hash al archivo PDF\:
S0706=Error al procesar la firma. los bytes de firma no corresponden a
una firma v\u00E1lida.
S0707=Error al generar la estampa\:
S0708=Error al procesar la respuesta de la estampa.
S0709=Error al unir la estampa con el documento PDF.
S0710=Error al generar el archivo PDF con la firma y los
par\u00E1metros ingresados.
S0711=Error al generar los bytes del archivo PDF firmado.
S0712=Error al firmar el hash. No se puede leer el hash por lo que
está formato base64.
S0713=Error al inicializar la firma PKCS7 para firmar el Hash:
S0714=El número de hashs a firmar es diferente al número de
PdfParameters.
S0715=Error al agregar la verificaci\u00F3n CRL/OSCP al documento PDF.
S0716=Error al estampar la verificaci\u00F3n LTV al documento PDF\:
S0717=No se encontraron firmas v\u00E1lidas para estampar o no existe
verificaci\u00F3n CRL/OCSP v\u00E1lida para agregar al documento PDF.
S0718=Error al decodificar la firma en base64.

# Rango 200 relacionado con keystore o certificado

S0200=No es posible hacer lectura del keyStore
S0201=Algoritmo usado para la firma, no es valido
S0202=Problemas en la lectura del certificado digital
S0203=Parámetros invalidos en el algoritmo usado para la firma
S0204=Problemas de reconocimiento del keystore que se esta usando
S0205=Verificación no válida para el certificado digital usado
S0206=Problemas de acceso al certificado digital
S0207=Problema en la firma critografica del mensaje (CMS)
```

```
S0208=El certificado seleccionado no puede ser usado porque no
contiene la llave privada o no es un certificado válido


#-- Rango 300 otros
```

Errores presentados al momento de hacer el estampado cronológico:

```
T101=No ha ingresado el usuario y/o password para la
autenticaci\u00F3n frente a la TSA.
T102=Error al validar la informaci\u00F3n de la autenticaci\u00F3n
frente a la TSA por certificado.
T103=Error al validar la informaci\u00F3n de las propiedades de
configuracion de la TSA.
T104=Error al validar la informaci\u00F3n de las propiedades de
validacion de estampa.
T105=Algoritmo seleccionado inv\u00E1lido\:
T106=Falla al obtener respuesta desde la direcci\u00F3n\:
T107=la URL de la TSA esta mal formada\:
T108=Error al obtener la respuesta de la TSA. No se ha podido conectar
a la direcci\u00F3n\:
T109=Error al leer la respuesta de la TSA.
T110=Respuesta inv\u00E1lida de la TSA.
T1104=La informaci\u00F3n a estampar es de formato incorrecto.
T1101073741824=Error al procesar la solicitud de estampa en la TSA.
T11065536=No est\u00E1 autorizado para solicitar una estampa\:
T110128=Identificador de algoritmo no soportado o no conocido.
T110256=La pol\u00EDtica OID enviada no es soportada por la TSA.
T11064=Integridad de la firma de la TSR es inv\u00E1lida.
T11032=Transacci\u00F3n de para estampar no permitida o soportada.
T11016=La fecha de la estampa es diferente a la definida por la
pol\u00EDtica local.
T1108=Ning\u00FAn certificado se encontr\u00F3 con la informaci\u00F3n
suministrada.
T111=Error al obtener el token de la respuesta de la TSA.
T112=Error al extraer el hash del documento, no se encuentra el
algoritmo SHA1.
T113=Error al generar/codificar la petici\u00F3n TSQ.
T114=Error en la lectura de los bytes de la TSR. TSR inv\u00E1lida o
no tiene un formato v\u00E1lido.
T115=Error al validar la respuesta de la TSA.
T116=Error al decodificar los bytes de la respuesta de la TSA.
T117=Error al analizar el contenido del certificado de la estampa.
T118=Error al obtener los certificados de la estampa.
T119=Existi\u00F3 un problema la firmar el TSQ \:
T120=No se puede validar el estado de la estampa.
T121=No se puede extraer el contenido de la estampa.
T122=El certificado con el que se ha estampado ha expirado.
T123=No se ha podido validar la vigencia del certificado con el que se
ha estampado.
```

Errores presentados al momento de hacer la verificación del certificado:

1 = Problemas al verificar el certificado con Keystore.
2 = Problemas al leer el certificado.
3 = Problemas al leer la información el certificado.
4 = La contraseña del certificado es incorrecta.
5 = El certificado tiene una extención critica no admitida.
6 = El certificado no se encuentra valido para la fecha
7 = El certificado esta revocado desde la fecha
8 = Problemas al acceder al keystore del certificado.
9 = Error al acceder/leer la CRL.
10 = El certificado ha sido revocado
11 = Las CRLs no corresponden al emisor del certificado digital.
12 = Error en la lectura de la cadena de certificados autorizados.
13 = La CRL no firgura en los certificados de confianza.
14 = El certificado no fue emitido por una autoridad de certificación de confianza.
15 = No se encuentran los archivos de certificados autorizados.
16 = La ruta de certificado de confianza no se encuentra dentro del Keystore de Java.
17 = La CRL esta desactualizada
18 = No se encuentra la direccion URL para la autenticacion OCSP.
19 = Error en la verificacion OCSP.

V100 = Error al tratar de extraer las firmas.
V101 = La firma digital no tiene un formato válido.
V102 = No se pueden extraer los bytes del texto firmado.
V103 = El algoritmo de firma digital del certificado no está soportado.
V104 = No existe un proveedor de seguridad definido por defecto.
V105 = No es posible obtener el certificado digital: #.
V106 = Error al leer el contenido de la firma.
V107 = Error al extraer los bytes del pdf.
V200 = Error: El archivo xml está mal formado.
V201 = Error en la integridad de La firma digital.
V203 = El algoritmo de firma digital del certificado no está soportado.
V204 = Ha fallado la validación de la integridad de la firma con el certificado digital emitido a
V205 = No se ha podido validar integridad # ,Error al validar la integridad : %;
V300 = Error al verificar los certificados del documento.
V250 = Error al iniciar el verificador de certificados.

V251 = Error al leer algún certificado.
V252 = Error al leer la información de la respuesta OCSP suministrada.
V253 = Error al leer la información de la solicitud OCSP suministrada.
V254 = Error al escribir la respuesta OCSP en la ruta suministrada.
V255 = Error al escribir la solicitud OCSP en la ruta suministrada.
V256 = La URL del servidor OCSP no es válida.

```
V257 = Proveedor de seguridad incorrecto.
V258 = Error obteniendo el identificador del certificado que se desea
verificar.
V259 = Error leyendo o escribiendo datos en la conexión con el
servidor OCSP.
V260 = Error: Petición inválida.
V261 = Error: Problema interno en el servidor OCSP.
V262 = Error: Intente de nuevo más tarde.
V263 = Error: La petición debe estar firmada.
V264 = Error: Debe autenticarse con el servidor OCSP.
V265 = El identificador del certificado que se obtuvo en la respuesta
no corresponde con el certificado que se desea verificar.
V266 = Se produjo un error en el procesamiento de la respuesta OCSP.
V267 = Error al leer el archivo del certificado.
V268 = Error creando el objeto certificado en el sistema.
V269 = Error leyendo el archivo del certificado para obtener la URL
del servidor OCSP.
V270 = Error obteniendo el Acceso a la información de autoridad.

V1 = Problemas al verificar el certificado con Keystore.
V2 = Problemas al leer el certificado.
V3 = Problemas al leer la información el certificado.
V4 = Problemas al decodificar el certificado con la contraseña.
V5 = El certificado tiene una extención critica no admitida.
V6 = El certificado no se encuentra valido para la fecha
V7 = El certificado esta revocado desde la fecha
V8 = Problemas al acceder al keystore del certificado.
V9 = Error al acceder/leer la CRL.
V10 = El certificado ha sido revocado
V11 = Las CRLs no corresponden al emisor del certificado digital.
V12 = Error en la lectura de la cadena de certificados autorizados.
V13 = La CRL no firgura en los certificados de confianza.
V14 = El certificado no figura en los certificados de confianza.
V15 = No se encuentran los archivos de certificados autorizados.
V16 = La ruta de certificado de confianza no se encuentra dentro del
Keystore de Java.
V17 = La CRL esta desactualizada
V18 = No se encuentra la direccion URL para la autenticacion OCSP.
V19 = Error en la verificacion OCSP:
V20 = No se encuentra el elemento Signature para verificar la firma
V300 = Error obteniendo el identificador del certificado que se desea
verificar.
V301 = Error leyendo o escribiendo datos en la conexión con el
servidor OCSP.
V302 = Error: Petición inválida.
V303 = Error: Problema interno en el servidor OCSP.
V304 = Error: Intente de nuevo más tarde."
V305 = Error: La petición debe estar firmada.
V306 = Error: Debe autenticarse con el servidor OCSP.
V307 = La firma de la respuesta no es válida.
V308 = La firma del certificado a verificar no es válida.
```

```
V309 = La llave pública del emisor no es válida.
V310 = El algoritmo utilizado para la firma del certificado no es
válido.
V311 = El identificador del certificado que se obtuvo en la respuesta
no corresponde con el certificado que se desea verificar.
V312 = Se produjo un error en el procesamiento de la respuesta OCSP.
V313 = Proveedor incorrecto.
V314 = Error creando el objeto certificado en el sistema.
V315 = Error al leer el archivo del certificado.
V316 = Error leyendo el archivo del certificado para obtener la URL
del servidor OCSP.
V317 = Error obteniendo el Acceso a la información de autoridad.
V318 = Alguno de los certificados ha expirado.
V319 = Alguno de los certificados todavía no es válido.
V320 = Error al leer algún certificado.
V400 = No hay archivos validos de CRL en el directorio
V401 = Hubo un error al leer las CRLs en el directorio
V402 = La URL de la CRL no es válida
V403 = No ha seleccionado ningun tipo de verificación correcto.
V404 = No se encuentra el elemento Signature
V405 = validateContext is null


9999 = Excepción no controlada.
```