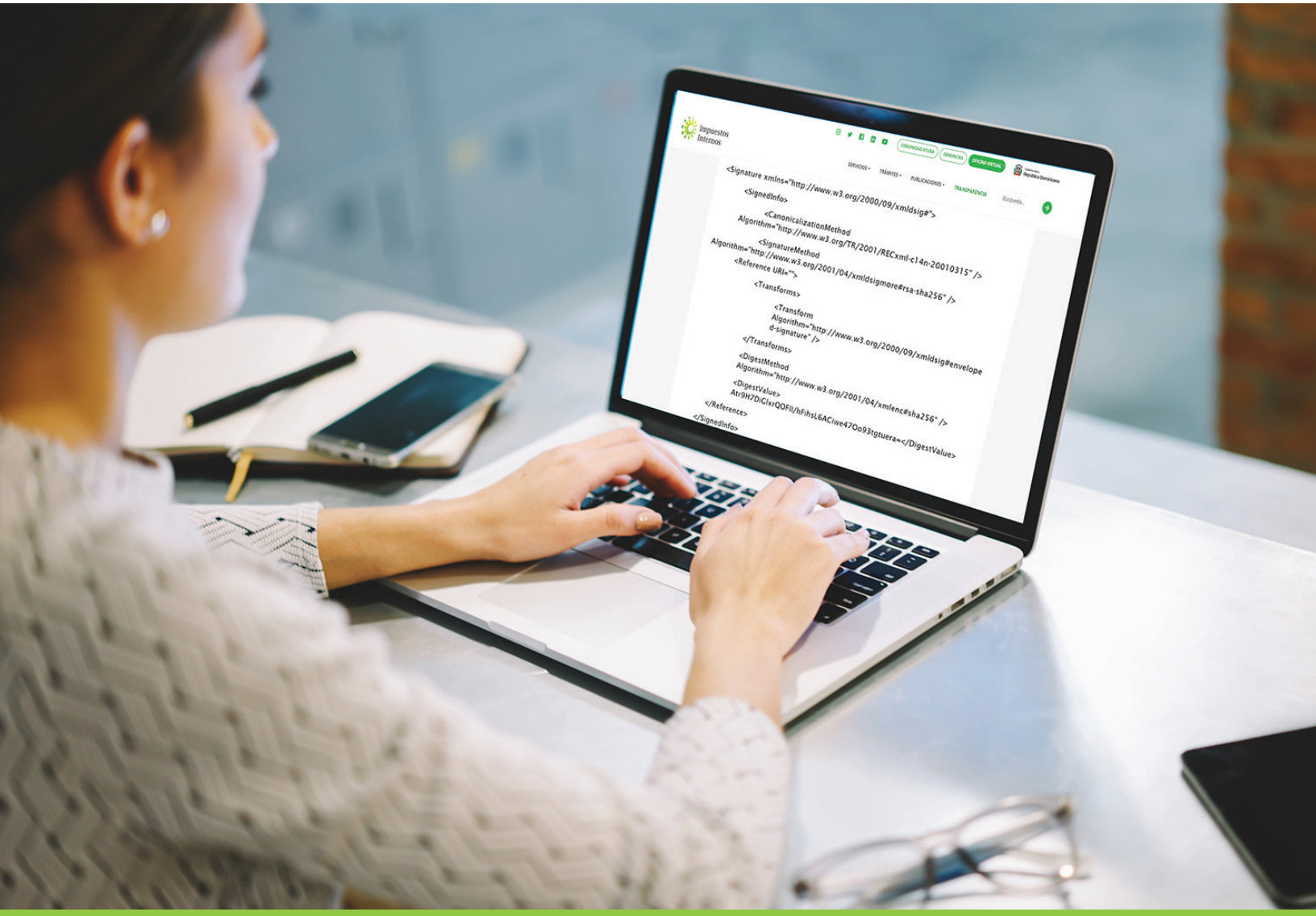
IMPUESTOS INTERNOS



Firmado Comprobantes Fiscales Electrónicos (e-CF)

Contenido

Propósito

El presente documento tiene por finalidad proveer varios ejemplos demostrativos para firmar un e-CF por medio de un certificado digital, el cual, sirve para identificar un usuario en Internet, asociando a la persona física o jurídica a una serie de datos, siendo necesario un tercero de confianza o autoridad certificadora que autentifique dicha asociación.

Los ejemplos que presentamos se tratan de una segmentación de códigos generados en las tecnologías .net(c#), VB.Net, TypeScript, Java y PHP, mediante los cuales es posible firmar el XML base de los e-CF, no obstante, los mismos son demostrativos y no representan la única forma de firmar un XML digitalmente.

Descripción

De acuerdo con el RFC2828 una firma digital se define como "un valor calculado mediante un algoritmo criptográfico y añadido a la estructura de un objeto que contiene información de manera que cualquier receptor de la información pueda usar la firma para verificar su origen e integridad".

De los elementos que componen un XML Signature (componente de firma de una XML), tenemos que tomar en cuenta las siguientes etiquetas:

- **SignatureMethod:** Define el algoritmo usado para la firma digital.
- **DigestMethod:** Identifica el algoritmo usado para calcular el valor del elemento DigestValue.
- **DigestValue:** Propiedad que contiene el valor codificado en formato Base 64 de la síntesis del objeto descrito por la propiedad Uri en la etiqueta Reference Ej.: <Reference URI="">.

El valor de la propiedad URI dentro de la etiqueta Reference tiene que estar en blanco (esto para referencia de que la firma se aplica a todo el documento).

El SignatureMethod y el DigestMethod usan como algoritmo de encriptación/desencriptación la función criptográfica SHA256 para la firma, es obligatorio usar este tipo de función al firmar el XML de la e-CF.

URL de referencias.

Descripción de firma digital, reglas y sintaxis: https://www.w3.org/TR/xmldsig-core2/

Ejemplo de la estructura de una firma: dentro de un XML

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
 <SignedInfo>
 <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/RECxml-c14n-20010315" />
 <SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsigmore#rsa-sha256" />
 <Reference URI="">
  <Transforms>
   <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#envelope d-signature" />
  </Transforms>
  <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
  <DigestValue>Atr9H7DiGlxrQOFII/hFihsL6ACiwe47Oo93tgtuera=</DigestValue>
 </Reference>
 </SignedInfo>
  <SignatureValue>gQyXO0FFDGIITESTpP5xZjLIRtv/Q7/ixe1lNDLDA5aw...</SignatureValue>
 <KeyInfo>
  <X509Data>
<X509Certificate>DGIITESTBFagAwIBAgIInYGQUX9q0IwDQYJKoZIhvcNAQ...</X509Certificate>
  </X509Data>
 </KeyInfo>
</Signature>
```

Método de firmado en .net (c#),

```
/// <summary>
/// Método utilizado para el firmado de XML con un certificado digital.
/// </summary>
/// <param name="xmlDoc">XML a firmar.</param>
/// <param name="pathCert">Ubicacion del certificacion digital</param>
/// <param name="passCert">Contraseña del certificado digital</param>
/// <returns></returns>
static XmlDocument Signed(XmlDocument xmlDoc, string pathCert, string passCert)
{
    try
    {
        if (!File.Exists(pathCert)) throw new Exception("El certificado para firma no existe");
        var cert = new X509Certificate2(pathCert, passCert, X509KeyStorageFlags.Exportable);
        var exportedKeyMaterial = cert.PrivateKey.ToXmlString(true);
        var key = new RSACryptoServiceProvider(new CspParameters(24));
        key.PersistKeyInCsp = false;
        key.FromXmlString(exportedKeyMaterial);
        SignedXml signedXml = new SignedXml(xmlDoc);
        signedXml.SigningKey = key;
        signedXml.SignedInfo.SignatureMethod =
"http://www.w3.org/2001/04/xmldsig-more#rsa-sha256";
```

```
        // Se agrega la referencia del algoritmo de firma utilizado.
        Reference reference = new Reference();
        reference.AddTransform(new XmlDsigEnvelopedSignatureTransform());
        reference.DigestMethod = "http://www.w3.org/2001/04/xmlenc#sha256";
        reference.Uri = "";
        signedXml.AddReference(reference);

        // Se agrega la información del certificado utilizado para la firma.
        KeyInfo keyInfo = new KeyInfo();
        keyInfo.AddClause(new KeyInfoX509Data(cert));
        signedXml.KeyInfo = keyInfo;

        // Generate the signature.
        signedXml.ComputeSignature();

        //Obtenemos la representación del XML firmado y la guardamos en un XmlElement object
        XmlElement xmlFirmaDigital = signedXml.GetXml();

        //Adicionamos el elemento de la firma al documento XML.
        xmlDoc.DocumentElement.AppendChild(xmlDoc.ImportNode(xmlFirmaDigital, true));
        return xmlDoc;
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

Método de firmado en VB.Net

```
''' <summary>
    ''' Método para el firmado de XML con certificado digital (*.p12)
    ''' </summary>
    ''' <param name="xmlDoc"> documento a firmar</param>
    ''' <param name="pathCert">ubicacion del certificado a utilizar para firmado</param>
    ''' <param name="passCert">contraseña del certificado digital</param>
    ''' <returns>XML firmado digitalmente</returns>
    Private Function Signed (ByVal xmlDoc As XmlDocument, ByVal pathCert As String, ByVal
passCert As String) As XmlDocument
        Try
            If Not File.Exists(pathCert) Then Throw New Exception("El certificado para firma no existe")
            Dim cert = New X509Certificate2(pathCert, passCert, X509KeyStorageFlags.Exportable)
            Dim exportedKeyMaterial = cert.PrivateKey.ToXmlString(True)
            Dim key = New RSACryptoServiceProvider(New CspParameters(24))
            key.PersistKeyInCsp = False
            key.FromXmlString(exportedKeyMaterial)
            Dim signedXml As SignedXml = New SignedXml(xmlDoc)
```

```
            signedXml.SigningKey = key
            signedXml.SignedInfo.SignatureMethod =
"http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"
            Dim reference As Reference = New Reference()
            reference.AddTransform(New XmlDsigEnvelopedSignatureTransform())
            reference.DigestMethod = "http://www.w3.org/2001/04/xmlenc#sha256"
            reference.Uri = ""
            signedXml.AddReference(reference)
            Dim keyInfo As KeyInfo = New KeyInfo()
            keyInfo.AddClause(New KeyInfoX509Data(cert))
            signedXml.KeyInfo = keyInfo
            signedXml.ComputeSignature()
            Dim xmlFirmaDigital As XmlElement = signedXml.GetXml()
            xmlDoc.DocumentElement.AppendChild(xmlDoc.ImportNode(xmlFirmaDigital, True))
            Return xmlDoc
        Catch ex As Exception
            Throw ex
        End Try
    End Function
```

Método de firmado en TypeScript

```
import { Injectable } from '@angular/core';
import * as forge from 'node-forge';
import * as xmldom from  'xmldom';

/*Referencias de librerias
 Node-forge: https://www.npmjs.com/package/node-forge#pkcs12

Xmldom: https://www.npmjs.com/package/xmldom

P12-pem:https://www.npmjs.com/package/p12-pem

  // Special character normalization:
  // - https://www.w3.org/TR/xml-c14n#ProcessingModel (Attribute / Text)
  // - https://www.w3.org/TR/xml-c14n#Example-Chars

Estas librerias fueron probadas Angular CLI v.10.0.8.
Note.js v.14.16.0
*/

@Injectable({
  providedIn: 'root'
})
export class FirmaXMLService {
```

```
  constructor() { }
/* El parametro xml:un string formateado en xml:
("<ECF><eNCF>E310000000001</eNCF></ECF>").
   El parametro certificado:  tipo ArrayBuffer, puede provenir de un control fileUpload.
   El Password:es el valor de la clave del certificado.
*/
 //Método utilizado para generar la firma del xml.

  public Firmar(xml:any,certifado: ArrayBuffer,password:string) {
    //Se Genera el  PEM & PrivateKey del certificado
    let pem = this.convertToPem(certifado, password);
    let privateKey = forge.pki.privateKeyFromPem(pem.pemKey);
    //--------------------------------------------------------

    //Se prepara la  encriptacion del contenido del xml
    let md = forge.md.sha256.create();

    var xmlData2 = (new xmldom.DOMParser()).parseFromString(xml);

    //Canolizacion del tag SignedInfo para poder generar correctamente la firma del documento.

    var xmlCanolizadoData2=this.c14nCanonicalization(xmlData2.firstChild,null);
    md.update(xmlCanolizadoData2.toString(), 'utf8');
    //--------------------------------------------------------

    /* Se obtiene solo el contenido del en formato encode64
      para luego asignar este valor a el tag DigestValue
    */
    let messageDigest = forge.util.encode64(md.digest().data);
    //--------------------------------------------------------

    //Se construye el xml con el tag de firma integrado en este punto aun no se genera el tag
SignatureValue
    let pemXMLFormat=this.obtenerPEMXMLFormat(pem.pemCertificate);
    let xmlSinFirmado=
this.agregarEstructuraFirma(xmlCanolizadoData2.toString(),pemXMLFormat,messageDigest);

    /*Se convierte a formato XML la transformacion anterior. Esta transformacion es necesaria
para poder obtener la  canolizacion del XML.
    */
    var xmlData = (new xmldom.DOMParser()).parseFromString(xmlSinFirmado);

    //Canolizacion del tag SignedInfo para poder generar correctamente la firma del documento.
    var
xmlCanolizado=this.c14nCanonicalization(xmlData.getElementsByTagName("SignedInfo")[0],{defau
ltNsForPrefix:{ds: 'http://www.w3.org/2000/09/xmldsig#'}});
```

```
//Generación de firma,
    md = forge.md.sha256.create();
    md.update(xmlCanolizado.toString(), 'utf8');
    let signature = privateKey.sign(md);
    //------------------------------------------------------------------------

    //Agregando firma a el xml generado con el formato de firma.
    let signatureValue = '<SignatureValue>' + forge.util.encode64(signature) + '</SignatureValue>';
    let indiceFirma = xmlSinFirmado.search('</Signature>');
    let xmlFirmado = xmlSinFirmado.substring(0, indiceFirma) + signatureValue +
xmlSinFirmado.substring(indiceFirma);

    let resultadoFirma={
      'xmlFirmadoString':xmlFirmado,
      'xmlFirmadoBlob':this.generarBlobArchivoFirmado(xmlFirmado),

    }

    return resultadoFirma;

  }
  // Método que recibe un xml y retorna un archivo blob.
  private generarBlobArchivoFirmado(xmlFirmado:string)
  {
    let blob = new Blob([xmlFirmado],
    { type: "text/xml;charset=utf-8" });

    return blob;
  }

  //Sirve para obtener el formato del Pem
  private obtenerPEMXMLFormat(PEM:any)
  {
    let PEMToXmlFormat = PEM.replace('-----BEGIN CERTIFICATE-----', '');
      PEMToXmlFormat = PEMToXmlFormat.replace('-----END CERTIFICATE-----', '');
      return PEMToXmlFormat;
  }

 //Permite generar la estructura del area de la firma en formato de la estructura del xml
 private agregarEstructuraFirma(xml:string,PEM:any,digestValue:any)
 {
  let signatureXmlFormat = '<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">';
  let keyinfoXmlFormat = '<KeyInfo>' +
  '<X509Data>' +
  '<X509Certificate>' + PEM + '</X509Certificate>' +
  '</X509Data>' +
  '</KeyInfo>';
```

```
  let signedInfoXmlFormat = '<SignedInfo><CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />'+
                  '<SignatureMethod
Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />'+
                  '<Reference URI="">'+
                     '<Transforms>'+
                          '<Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />'+
                      '</Transforms>'+
                      '<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />'+
                      '<DigestValue>' + digestValue + '</DigestValue>'+
                  '</Reference>'+
                 '</SignedInfo>';

  let firmado = signatureXmlFormat + signedInfoXmlFormat + keyinfoXmlFormat + '</Signature>';

  let indice = xml.search ('</ECF>');

  let xmlSinFirmado = xml.substring(0, indice) + firmado + xml.substring(indice);
  return xmlSinFirmado;
 }

  //Permite convertir a Pem
  private convertToPem(p12ArrayBuffer, password) {

    let p12Asn1 = forge.asn1.fromDer(p12ArrayBuffer);
    let p12 = forge.pkcs12.pkcs12FromAsn1(p12Asn1, false, password);
    let pemKey = this.getKeyFromP12(p12, password);
    let _a = this.getCertificateFromP12(p12), pemCertificate = _a.pemCertificate, commonName =
_a.commonName;
    return { pemKey: pemKey, pemCertificate: pemCertificate, commonName: commonName };
  }

  // Obtiene la llave de un p12
  private getKeyFromP12(p12, password) {
    var keyData = p12.getBags({ bagType: forge.pki.oids.pkcs8ShroudedKeyBag },
password);
    var pkcs8Key = keyData[forge.pki.oids.pkcs8ShroudedKeyBag][0];
    if (typeof pkcs8Key === 'undefined') {
      pkcs8Key = keyData[forge.pki.oids.keyBag][0];
    }
    if (typeof pkcs8Key === 'undefined') {
      throw new Error('Unable to get private key.');
    }
var pemKey = forge.pki.privateKeyToPem(pkcs8Key.key);
    pemKey = pemKey.replace(/\r\n/g, '');
    return pemKey;
  }
```

```
// Para obtener el certificado de un p12
  private getCertificateFromP12(p12) {
    var certData = p12.getBags({ bagType: forge.pki.oids.certBag });
    var certificate = certData[forge.pki.oids.certBag][0];
    var pemCertificate = forge.pki.certificateToPem(certificate.cert);
    pemCertificate = pemCertificate.replace(/\r\n/g, '');
    var commonName = certificate.cert.subject.attributes[0].value;
    return { pemCertificate: pemCertificate, commonName: commonName };
  }

  // Utilizado para poder realizar la canonicalización de etiquetas del xml.
  private c14nCanonicalization = function (node, options) {
    options = options || {};
    var defaultNs = options.defaultNs || "";
    var defaultNsForPrefix = options.defaultNsForPrefix || {};
    var ancestorNamespaces = options.ancestorNamespaces || [];

    var res = this.c14nCanonicalizationInterno(node, [], defaultNs, defaultNsForPrefix,
ancestorNamespaces);
    return res;
  };

  private c14nCanonicalizationInterno = function (node, prefixesInScope, defaultNs,
defaultNsForPrefix, ancestorNamespaces) {

    if (node.nodeType === 8) { return this.renderComment(node); }
    if (node.data) { return this.encodeSpecialCharactersInText(node.data); }

    var i, pfxCopy
      , ns = this.renderNs(node, prefixesInScope, defaultNs, defaultNsForPrefix,
ancestorNamespaces)
      , res = ["<", node.tagName, ns.rendered, this.renderAttrs(node, ns.newDefaultNs), ">"];

    for (i = 0; i < node.childNodes.length; ++i) {
      pfxCopy = prefixesInScope.slice(0);
      res.push(this.c14nCanonicalizationInterno(node.childNodes[i], pfxCopy, ns.newDefaultNs,
defaultNsForPrefix, []));
    }

    res.push("</", node.tagName, ">");
    return res.join("");
  };

  private renderComment = function (node) {

    if (!this.includeComments) { return ""; }

    var isOutsideDocument = (node.ownerDocument === node.parentNode),
```

```
      isBeforeDocument = null,
      isAfterDocument = null;

    if (isOutsideDocument) {
      var nextNode = node,
        previousNode = node;

      while (nextNode !== null) {
        if (nextNode === node.ownerDocument.documentElement) {
          isBeforeDocument = true;
          break;
        }

        nextNode = nextNode.nextSibling;
      }

      while (previousNode !== null) {
        if (previousNode === node.ownerDocument.documentElement) {
          isAfterDocument = true;
          break;
        }

        previousNode = previousNode.previousSibling;
      }
    }

    return (isAfterDocument ? "\n" : "") + "<!--" + this.encodeSpecialCharactersInText(node.data) +
"-->" + (isBeforeDocument ? "\n" : "");
  };

  private renderAttrs = function(node, defaultNS) {
    var a, i, attr
      , res = []
      , attrListToRender = [];

    if (node.nodeType===8) { return this.renderComment(node); }
if (node.attributes) {
      for (i = 0; i < node.attributes.length; ++i) {
        attr = node.attributes[i];
        // Para ignorar los atributos de definición del espacio de nombres.
        if (attr.name.indexOf("xmlns") === 0) { continue; }
        attrListToRender.push(attr);
      }
    }
attrListToRender.sort(this.attrCompare);

    for (a in attrListToRender) {
      if (!attrListToRender.hasOwnProperty(a)) { continue; }
```

```
      attr = attrListToRender[a];
      res.push(" ", attr.name, '="', this.encodeSpecialCharactersInAttribute(attr.value), '"');
    }

    return res.join("");
  };

  private renderNs = function(node, prefixesInScope, defaultNs, defaultNsForPrefix,
ancestorNamespaces) {
    var a, i, p, attr
      , res = []
      , newDefaultNs = defaultNs
      , nsListToRender = []
      , currNs = node.namespaceURI || "";

      // Utilizado para manejar el espacio de nombres del propio nodo
    if (node.prefix) {
      if (prefixesInScope.indexOf(node.prefix)==-1) {
        nsListToRender.push({"prefix": node.prefix, "namespaceURI": node.namespaceURI ||
defaultNsForPrefix[node.prefix]});
        prefixesInScope.push(node.prefix);
      }
    }
    else if (defaultNs!=currNs) {
      //nuevo  por defecto ns.
        newDefaultNs = node.namespaceURI;
        res.push(' xmlns="', newDefaultNs, '"');
    }

      // Utilizado para manejar el espacio de nombres de los atributos.
    if (node.attributes) {
      for (i = 0; i < node.attributes.length; ++i) {
        attr = node.attributes[i];

        // Para manejar todos los atributos prefijados que están incluidos en la lista de prefijos y
donde el prefijo aún no está definido.
        // Los nuevos prefijos solo se pueden definir mediante `xmlns:`.

        if (attr.prefix === "xmlns" && prefixesInScope.indexOf(attr.localName) === -1) {
          nsListToRender.push({"prefix": attr.localName, "namespaceURI": attr.value});
          prefixesInScope.push(attr.localName);
        }

        //Manejar todos los atributos prefijados que no son definiciones xmlns.
        //Donde el prefijo no está definido .
        if (attr.prefix && prefixesInScope.indexOf(attr.prefix)==-1 && attr.prefix!="xmlns" &&
attr.prefix!="xml") {
          nsListToRender.push({"prefix": attr.prefix, "namespaceURI": attr.namespaceURI});
```

```
        prefixesInScope.push(attr.prefix);
      }
    }
  }

  if(Array.isArray(ancestorNamespaces) && ancestorNamespaces.length > 0){
    // Eliminar espacios de nombres que ya están presentes en nsListToRender
    for(var p1 in ancestorNamespaces){
      if(!ancestorNamespaces.hasOwnProperty(p1)) continue;
      var alreadyListed = false;
      for(var p2 in nsListToRender){
        if(nsListToRender[p2].prefix === ancestorNamespaces[p1].prefix
          && nsListToRender[p2].namespaceURI === ancestorNamespaces[p1].namespaceURI)
        {
          alreadyListed = true;
        }
      }

      if(!alreadyListed){
        nsListToRender.push(ancestorNamespaces[p1]);
      }
    }
  }

  nsListToRender.sort(this.nsCompare);

   //renderizar espacios de nombres
  for (a in nsListToRender) {
    if (!nsListToRender.hasOwnProperty(a)) { continue; }

    p = nsListToRender[a];
    res.push(" xmlns:", p.prefix, '="', p.namespaceURI, '"');
  }

  return {"rendered": res.join(""), "newDefaultNs": newDefaultNs};
};

 // Obtiene  el texto de caracteres encodeados
private static get_xml_special_to_encoded_text(attributeValue) {
  var xml_special_to_encoded_attribute = {
    '&': '&amp;',
    '<': '&lt;',
    '"': '&quot;',
    '\r': '',
    '\n': '',
    '\t': '&#x9;',
    "'" : '&apos;',
    '>' : '&gt;'
```

```
    }
    return xml_special_to_encoded_attribute[attributeValue];
  };

   // Encodear caracteres especiales en texto
  private encodeSpecialCharactersInText(text) {
    var _text = text;
    return text
      .replace(/([&<>\r])/g, function (str, item) {
        return FirmaXMLService.get_xml_special_to_encoded_text(item)
      })
  };

  // Encodear caracteres especiales en atributos
  private encodeSpecialCharactersInAttribute(attributeValue) {
    return attributeValue
      .replace(/([&<"\r\n\t])/g, function (str, item) {
        return FirmaXMLService.get_xml_special_to_encoded_text(item)
      })
  };
}
```

Método de firmado en Java

```
package dgii.lib;

import java.io.InputStream;
import java.io.Serializable;
import java.security.KeyStore;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import javax.xml.crypto.dsig.dom.DOMSignContext;
import javax.xml.crypto.dsig.keyinfo.KeyInfo;
import javax.xml.crypto.dsig.keyinfo.KeyInfoFactory;
import javax.xml.crypto.dsig.keyinfo.X509Data;
import javax.xml.crypto.dsig.spec.C14NMethodParameterSpec;
import javax.xml.crypto.dsig.spec.TransformParameterSpec;
import org.w3c.dom.Element;
//Ref:
https://repo1.maven.org/maven2/com/oracle/database/xml/xmlparserv2/21.8.0.0/xmlparserv2-
21.8.0.0.jar
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.parser.v2.XMLDocument;
```

```
//Fin Ref
import javax.xml.crypto.dsig.CanonicalizationMethod;
import javax.xml.crypto.dsig.DigestMethod;
import javax.xml.crypto.dsig.Reference;
import javax.xml.crypto.dsig.SignedInfo;
import javax.xml.crypto.dsig.Transform;
import javax.xml.crypto.dsig.XMLSignature;
import javax.xml.crypto.dsig.XMLSignatureFactory;

/*
 * Referencias
 * JDK
https://www.oracle.com/java/technologies/javase/javase8u211-later-archive-downloads.html
 *
 * https://community.oracle.com/tech/developers/discussion/1539728/digital-signing-validating
 *
 * https://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/DocumentBuilderFactory.html
 *
 * https://www.oracle.com/technical-resources/articles/wang-whitespace.html
 *
 * Parser XML v2-21.8.0.0
 * xmlparserv2-21.8.0.0.jar =>
https://repo1.maven.org/maven2/com/oracle/database/xml/xmlparserv2/21.8.0.0/xmlparserv2-
21.8.0.0.jar
 * https://repo1.maven.org/maven2/com/oracle/database/xml/xmlparserv2/
 * Si usa VS Code como entorno para desarrollar estas extensiones serán de utilidad
 *
 * https://code.visualstudio.com/docs/java/extensions
 * https://code.visualstudio.com/docs/java/java-tutorial
 * Solo instalar el extensionPack for Java y el Coding Pack for Java para el JDK recomendamos
Oracle Java SE
 */

public class SignManager{
    private static final String algoritmoSignatureMethod =
"http://www.w3.org/2001/04/xmldsig-more#rsa-sha256";
    /*
     * Método para el firmado un XML utilizando un certificado digital (.p12)
     * @param streamXML InputStream del xml que será firmado.
     * @param pathCerticate ruta del certificado digital (archivo extension .p12)
     * @param passwordCertificate contraseña del certificado digital
     * @param pathFirmado Ruta donde sera almacenado el XML firmado
     * @throws Exception
     */
    public Element SignXML(InputStream streamXML, InputStream streamCerticate,String
passwordCertificate) throws Exception
    {
```

```
try {
        // Creamos un DOM XMLSignatureFactory. Se usará para generar la firma
        XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");

        Reference ref = fac.newReference("", fac.newDigestMethod(DigestMethod.SHA256, null),
                Collections.singletonList(fac.newTransform(Transform.ENVELOPED,
(TransformParameterSpec) null)),
                null,
                null);

        // Creamos el objeto SignedInfo
        SignedInfo si =
fac.newSignedInfo(fac.newCanonicalizationMethod(CanonicalizationMethod.INCLUSIVE,
                (C14NMethodParameterSpec) null),
                fac.newSignatureMethod(algoritmoSignatureMethod, null),
                Collections.singletonList(ref));

        // Cargamos el archivo p12 desde disco
        KeyStore ks = KeyStore.getInstance("PKCS12");
        ks.load(streamCerticate, passwordCertificate.toCharArray());
        String param = ks.aliases().nextElement();

        // Extraemos los datos del archivo p12
        KeyStore.PrivateKeyEntry keyEntry = (KeyStore.PrivateKeyEntry) ks
                .getEntry(param, new
KeyStore.PasswordProtection(passwordCertificate.toCharArray()));

        X509Certificate cert = (X509Certificate) keyEntry.getCertificate();

        KeyInfoFactory kinfoFactory = fac.getKeyInfoFactory();

        // Cargamos nuestro certificado
        List<Serializable> x509Content = new ArrayList<Serializable>();
        x509Content.add(cert);

        // Objetos para extraer la llave privada
        X509Data x509d = kinfoFactory.newX509Data(x509Content);
        KeyInfo kinfo = kinfoFactory.newKeyInfo(Collections.singletonList(x509d));

        DOMParser parser = new DOMParser();
        parser.setPreserveWhitespace(false);
        parser.parse(streamXML);
        XMLDocument xml =parser.getDocument();

        Element xmlRoot = xml.getDocumentElement();

        // Crea el contexto de firmma y especifica las llaves privadas
        DOMSignContext dsc = new DOMSignContext(keyEntry.getPrivateKey(), xmlRoot);
```

```
            // crea los nodos de firmas xml
            XMLSignature signature = fac.newXMLSignature(si, kinfo);

            // en esta etapa dsc modifica el objeto xmlRoot insertándole la firma
            signature.sign(dsc);
            return xmlRoot;

        } catch (Exception e) {
            throw e;
        }
    }
}
```

Método de firmado en PHP

```
<?php

namespace Dgii\Lib;

include './lib/Exception/CertificateException.php';
include './lib/Exception/XmlSignatureValidatorException.php';
include './lib/Exception/XmlSignerException.php';
include './lib/PrivateKeyStore.php';
include './lib/Algorithm.php';
include './lib/CryptoSignerInterface.php';
include './lib/CryptoSigner.php';
include './lib/X509Reader.php';
include './lib/XmlReader.php';
include './lib/XmlSigner.php';

use Selective\XmlDSig\PrivateKeyStore;
use Selective\XmlDSig\Algorithm;
use Selective\XmlDSig\CryptoSigner;
use Selective\XmlDSig\XmlSigner;

/*
Descargar la librería XMLDSIG desde https://github.com/selective-php/xmldsig
Probado en las versiones de php 8.1.12 y 8.1.13
*/

/*
Nota:

**Refactorizaciones al archivo XmlSigner.php
al instanciar la clase DOMDocument coloque la propiedad preserveWhiteSpace a false debido a
que los espacios en blanco no deben ser preservados
```

```
Existe otra función que recibe un DOMDocument recuerde ajustar este valor antes de enviar el
objeto.

 $xml->preserveWhiteSpace = true;  cambiar a   $xml->preserveWhiteSpace = false;

 **Por otro lado

 $canonicalData = $element->C14N(true, false); cambiar a $canonicalData =
$element->C14N(false, false);

 puede dejarlos sin parámetros puesto que sus valores por defecto son false, es decir puede ser
=> $canonicalData = $element->C14N()

 **En la función appendSignature puede comentar las líneas 154 hasta la 170, los tag KeyValue,
RSAKeyValue, Exponent no son necesarios

 **Recuerde habilitar la extensión openssl en su archivo php.ini, en algunas distribuciones esta
deshabilitado por defecto.

*/

final class SignManager
{
      /**
     * The constructor.
     *
     * @param string $cert_store contenido del archivo p12
     * @param string $password contraseña para acceder a la información contenida en el
certificado
     * @param string $xml contenido del archivo xml
     */
    public function sing(string $cert_store, string $password, string $xml): string
    {
        if (!openssl_pkcs12_read($cert_store , $certs, $password)) {

            echo "Error: No fue posible leer el contenido del certificado.\n";
            exit;
        }

        $pem_file_contents = $certs['cert'] . $certs['pkey'];

        $privateKeyStore = new PrivateKeyStore();

        $privateKeyStore->loadFromPem($pem_file_contents, $password);

        $privateKeyStore->addCertificatesFromX509Pem($pem_file_contents);

        $algorithm = new Algorithm(Algorithm::METHOD_SHA256);
```

```
        $cryptoSigner = new CryptoSigner($privateKeyStore, $algorithm);

        $xmlSigner = new XmlSigner($cryptoSigner);

        $xmlSigner->setReferenceUri('');

        $signedXml = $xmlSigner->signXml($xml);

        return $signedXml;
    }

}
```

dgii.gov.do

Facturación Electrónica

Centro de Contacto: (809) 689-3444, opción 4
Contacto Directo: (809) 287-2009
Correo: facturacionelectronica@dgii.gov.do

En el portal web DGII - Comunidad de Ayuda,
categoría “Facturación Electrónica”

IMPUESTOS INTERNOS
Marzo 2023

@DGIIRD