Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Dipartimento federale delle finanze DFF

Ufficio federale dell'informatica e della telecomunicazione
Centro soluzioni
Soluzioni Berna
Sviluppi 1

Matthias Rüedlinger

Verifica della firma XML e-dec

Raccomandazioni per la verifica delle firme digitali (WS security)

Nome del progetto: **e-dec**
Versione: **0.3**
Data: **19.5.2009**

Stato	in elaborazione	in esame	approvato per l'utilizzazione
	☐	☐	☑

Cerchia delle persone interessate	
Autore:	Matthias Rüedlinger (mru)
Approvazione:	gruppo di progetto e-dec IDEA
Utenti:	AFD, responsabile del progetto
Per informazione:	AFD

Controllo delle modifiche, verifica e approvazione			
Quando	**Versione**	**Chi**	**Descrizione**
24.4.2009	0.1	mru	Prima versione
14.5.2009	0.2	mru	Aggiunta dell'esempio di codice Java
15.5.2009	0.3	mru, shu	Adeguamenti nel paragrafo CRL e adeguamento del titolo

Indice

1 Introduzione

Il presente documento è rivolto ai clienti e-dec della dogana e ai fornitori di software che intendono verificare la firma XML delle decisioni d'imposizione elettroniche (IMe).

Il documento spiega come attuare le disposizioni per EdecReceiptService figuranti nella descrizione dell'interfaccia [1] e nel contratto di servizio [2].

La verifica della firma di documenti XML è stata specificata dall'organizzazione W3C. La specificazione **XML Signature Syntax and Processing *(*XMLDsig)** può essere consultata al link seguente:

http://www.w3.org/TR/xmldsig-core/.

Per procedere alla verifica della firma di un documento XML, il linguaggio di programmazione o il framework corrispondenti devono attuare lo standard W3C **XML Signature Syntax and Processing *(*XMLDsig)**.

La firma XML è contenuta in una busta SOAP. L'inclusione della firma nell'intestazione SOAP avviene in base allo standard WS security.

1.1 Osservazione

Gli esempi di codici riportati servono solo quale guida nella procedura di verifica della firma e della catena di fiducia (*chain of trust*) del certificato in una risposta IMe. La resa del codice non è stata ottimizzata.

1.2 Riferimenti

I documenti riportati qui appresso contengono informazioni relative alla firma digitale nell'ambito e-dec.

Rif.	Titolo	Versione
[1]	Descrizione dell'interfaccia e-dec decisione d'imposizione (descrizione dei messaggi in entrata e in uscita per il servizio).	1.5
[2]	Service Contract EdecReceiptService (disponibile solo in tedesco; descrizione dei canali di comunicazione – servizio Internet e e-mail).	1.3

2 Strumenti / framework

L’elenco sottostante rappresenta una selezione di strumenti e framework utilizzabili per la verifica della firma XML.

Strumenti / framework	**Descrizione**	**URL**
Java SE 6	Permette di verificare le firme mediante le API XML fornite.	http://java.sun.com/javase/
Apache XML Security	Permette di verificare le firme XML per Java o C++ grazie a un’apposita biblioteca.	http://santuario.apache.org/
IAIK XML Security Toolkit (XSECT)	Si tratta di una biblioteca Java commerciale per la verifica delle firme XML.	http://jce.iaik.tugraz.at

3 Verifica della firma XML con Java

Si consiglia di utilizzare la firma digitale XML API (JSR 105) specificata da Java Community Process (JCP). Qui appresso sono elencate alcune implementazioni.

- A partire da Java 6, la firma digitale XML API è integrata nell'edizione Java standard.
- Apache XML Security è un'implementazione libera della firma digitale XML API.
- L'IAIK XML Security Toolkit (XSECT) è un'implementazione commerciale della firma digitale XML API.

3.1 Firma digitale XML API

L'esempio sottostante utilizza l'interfaccia standard della firma digitale XML API ed è dunque irrilevante quale implementazione si decida di impiegare. Vi sono alcune differenze solo per quanto riguarda la registrazione del provider di sicurezza.

Osservazione: in questo esempio la CRL (*certificate revocation list*, lista dei certificati revocati) della PKI (*public key infrastructure*, infrastruttura per le chiavi pubbliche) Admin non viene ancora controllata.

3.1.1 Esempio: verifica della firma XML

Dapprima viene letto il documento XML. È importante che il DocumentBuilder sia **NamespaceAware**.

```
InputStream is = XMLDsigClient.class.getResourceAsStream("eVVResponse.xml");

// create DocumentBuilderFactory which is Namespace aware
DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
builderFactory.setNamespaceAware(true);

DocumentBuilder builder = builderFactory.newDocumentBuilder();
logger.info("Is DocumentBuilder NamespaceAware: " + builder.isNamespaceAware());

// parse xml file (dumped soap request)
Document xmldoc = builder.parse(is);
```

XPath deve essere inizializzato con un proprio **NamespaceContext** affinché si possa utilizzare il Namespace nell'interrogazione XPath. Il **NamespaceContextImpl** implementa il NamespaceContext dell'interfaccia e deve essere creato dall'utente stesso (v. esempio relativo al NamespaceContext).

```
// create XPath object which has own NamespaceContext
XPath xpath = XPathFactory.newInstance().newXPath();
// with a custom NamespaceContext we can use Namespaces in our XPath query
NamespaceContext nsc = new NamespaceContextImpl();
xpath.setNamespaceContext(nsc);
```

Con XPath è possibile estrarre il **X509 Token**. Esso contiene un certificato X509 codificato in Base64.

e-dec

```
// extract x509 Token --> xml element wsse:BinarySecurityToken
XPathExpression expr = xpath.compile("//wsse:Security/wsse:BinarySecurityToken");
Node x509Node = (Node) expr.evaluate(xmldoc, XPathConstants.NODE);
```

Si possono utilizzare i dati del X509 Token per creare un certificato X509. Tuttavia, nel certificato occorre marcare l'inizio con **-----BEGIN CERTIFICATE-----** e la fine con **-----END CERTIFICATE-----**; in caso contrario la lettura del certificato non è possibile.

```
// X509 Token is encoded in base64
String header = "-----BEGIN CERTIFICATE-----\n";
String footer = "\n-----END CERTIFICATE-----";
// so we need a to add the certificate header and footer
// to the raw x509 data
byte[] x509data = (header + x509Node.getTextContent() + footer).getBytes();
ByteArrayInputStream bis = new ByteArrayInputStream(x509data);

// create certificate
CertificateFactory cf = CertificateFactory.getInstance("X.509");
X509Certificate cert = (X509Certificate) cf.generateCertificate(bis);
```

A questo punto, il certificato X509 e la catena di fiducia vengono verificati mediante il certificato CA. Se il certificato o la catena di fiducia non sono validi, il **CertPathValidator.validate(…)** lancia un'eccezione.

```
// verify ca chain
// read in ca cert
is = XMLDsigClient.class.getResourceAsStream("adminca-cd-t01_BIT_CA_certificate.crt");
X509Certificate caCert = (X509Certificate) cf.generateCertificate(is);

// trusted ca cert
Set<TrustAnchor> trust = Collections.singleton(new TrustAnchor(caCert, null));
PKIXParameters params = new PKIXParameters(trust);

// Disable CRL checking since we are not supplying any CRLs
params.setRevocationEnabled(false);
// sets the time for which the validity of the certification
// path should be determined
params.setDate(new Date());

CertPath certPath = cf.generateCertPath(Collections.singletonList(cert));
CertPathValidator certPathValidator = CertPathValidator.getInstance("PKIX");
PKIXCertPathValidatorResult result = (PKIXCertPathValidatorResult) certPathValidator
    .validate(certPath, params);
```

L'elemento di firma XML viene estratto mediante XPath e viene inizializzato un JSR 105 provider. In questo caso il provider viene inizializzato esplicitamente; ciò è necessario se, ad esempio, si lavora con Apache XML Security. Nel caso di Java 6, è già registrato un provider di firma digitale XML API (JSR 105) e questo passaggio non dovrebbe dunque essere necessario.

```
// extract xml element ds:Signature
expr = xpath.compile("//wsse:Security/ds:Signature");
Node dsSignature = (Node) expr.evaluate(xmldoc, XPathConstants.NODE);

DOMValidateContext context = new DOMValidateContext(cert.getPublicKey(), dsSignature);

String providerName = System.getProperty("jsr105Provider",
    "org.jcp.xml.dsig.internal.dom.XMLDSigRI");

logger.info("jsr 105 provider: " + providerName);

XMLSignatureFactory factory = XMLSignatureFactory.getInstance("DOM", (Provider) Class
    .forName(providerName).newInstance());

XMLSignature signature = factory.unmarshalXMLSignature(context);
```

La firma XML viene verificata con il **DOMValidateContext**. Quest'ultimo possiede la chiave pubblica e un riferimento all'elemento di firma XML.

```
      // Check core validation status
      boolean coreValidity = signature.validate(context);

      if (coreValidity == false) {

        logger.error("Signature failed core validation!");
        boolean sv = signature.getSignatureValue().validate(context);
        logger.info("Signature validation status: " + sv);

        // Check the validation status of each Reference
        Iterator<Reference> i = signature.getSignedInfo().getReferences().iterator();

        for (int j = 0; i.hasNext(); j++) {
          // signature was not valid so try to find out which refrence was invalid
          Reference ref = i.next();
          boolean refValid = ref.validate(context);
          String id = ref.getURI();
          logger.info("Reference (" + j + ") with URI [" + id + "] validation status: "
              + refValid);
        }
      }
      else {
        logger.info("Signature passed core validation!");
      }
```

3.1.2 Esempio: NamespaceContext Implementation

Questa è l'implementazione NamespaceContext che riconosce tutti i Namespace necessari della risposta IMe. Il NamespaceContext permette di effettuare le interrogazioni XPath con i prefissi corrispondenti. In tal modo, si garantisce il mapping tra i prefissi e i Namespace.

```
public class NamespaceContextImpl implements NamespaceContext{

  public static final String NS_URI_WSSE = "http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd";
  public static final String PREFIX_WSSE = "wsse";
  public static final String NS_URI_SOAP_ENV = "http://schemas.xmlsoap.org/soap/envelope/";
  public static final String PREFIX_SOAP_ENV = "soap";
  public static final String NS_URI_XMLDSIG = "http://www.w3.org/2000/09/xmldsig#";
  public static final String PREFIX_XMLDSIG = "ds";
  public static final String NS_URI_EVV = "http://www.e-
dec.ch/xml/schema/edecReceiptResponse/v1";
  public static final String PREFIX_EVV = "evv";
  private Map<String, String> value = new HashMap<String, String>();

  public NamespaceContextImpl() {
    value.put(PREFIX_EVV, NS_URI_EVV);
    value.put(PREFIX_SOAP_ENV, NS_URI_SOAP_ENV);
    value.put(PREFIX_WSSE, NS_URI_WSSE);
    value.put(PREFIX_XMLDSIG, NS_URI_XMLDSIG);
  }

  public String getNamespaceURI(String prefix) {
    return value.get(prefix);
  }

  public String getPrefix(String uri) {
     throw new UnsupportedOperationException();
  }

  public Iterator<String> getPrefixes(String uri) {
    throw new UnsupportedOperationException();
  }
}
```

3.2 CRL – Certificate Revocation List (lista dei certificati revocati)

In questo esempio di codice, la CRL non viene ancora verificata. Una lista dei certificati revocati è ottenibile alla pagina iniziale del sito Admin PKI.

http://www.pki.admin.ch/crl.php

4 Fonti

Specificazione XML Signature Syntax and Processing (XMLDsig)
http://www.w3.org/TR/xmldsig-core/

XMl Digital Signature API (JSR 105)
http://jcp.org/en/jsr/detail?id=105

Apache XML Security
http://santuario.apache.org/

IAIK XML Security Toolkit (XSECT)
http://jce.iaik.tugraz.at/sic/products/xml_security/xsect

Articolo: XML Signature with JSR-105 in Java SE 6
http://today.java.net/pub/a/today/2006/11/21/xml-signature-with-jsr-105.html?page=1

Articolo: Using JSR 105 with JDK 1.4 or 1.5
http://weblogs.java.net/blog/mullan/archive/2008/02/using_jsr_105_w_1.html

Presentazione: XML Security and JSR 105-106
http://www.parleys.com/display/PARLEYS/XML+Security+and+JSR+105-106