Use of XML DSIG in ebXML Registry

Author: Sekhar Vajjhala

Date : 10/03/01

Version: 0.2

Status of this document

This document outlines a proposal for using XML signatures for ebXML registry. This document is work in progress.

Acknowledgements

Suresh Damodaran, Sanjay Patil and Farrukh Najmi provided invaluable feedback in the design of this document.

Introduction

This document specifies the use of XML signatures [XMLDSIG] by ebXML Registry Clients and ebXML Registry. The specification is targeted for Version 2.0 of the ebXML Registry Services Specification.

Related Documents

The following documents provide the necessary background and additional background to the reader:

- SOAP Messages with Attachments [SOAPATTACH]
- XML-Signature Syntax and Processing [XMLDSIG]

Use Cases

The use of XML signatures is intended to cover the following use cases. The Use Cases (indicated in bold) refer to the use case numbers in the Security Risks Document [SECRISK].

Use Case 9: Registry Client wants to ensure that the Registry Content he is publishing to Registry is not changed on the network.

Use Case 11: Registry Client wants to ensure that the Registry Content he has published to the Registry is not changed by the Registry Administrator.

Use Case 13: Registry Client wants to ensure that the Registry Content sent to him by Registry is not changed on the network.

Use Case 14: Registry Client wants to ensure that the Registry Content received from Registry is legitimate. For example, Registry Client wants to verify that the information claiming to have been published by a company XYZ was really published by company XYZ.

Caveats and Assumptions

The following assumptions are made by this specification:

1. The communication between a Registry Client and a Registry is assumed to either ebXML Messaging Service [ebMS] or just SOAP with Attachments [SOAPATTACH] (referred to as vanillla SOAP with Attachments in this document).
2. Registry Content can include payloads consisting of arbitrary digital content (i.e. payloads need not necessarily be XML documents). Payloads are carried in a SOAP Message with Attachments [SOAPATTACH].
3. ebXML message data (which contain data other than payload data in request/response messages) is carried in the SOAP body within a SOAP envelope.

Overview

This section provides an overview of how the XML signatures are used by Registry Clients and Registry Operator. Specific requirements are stated elsewhere in the document.

This document specifies the use of XML signatures for:

- Signing of information in the SOAP envelope. This is referred to as header signature.
- Signing of payload (i.e. information which is carried in a SOAP attachment). This is referred to as a payload signature.

Payload and header signatures are each represented by a distinct, separate ds:Signature element. Either both, one or none of the payload and header signature may be present as illustrated by the following table:

Header Signature	Payload Signature	Usage
No	No	Non secure client access
No	Yes	--- No Use Case ----
Yes	No	Delete object
Yes	Yes	Submit Objects with payload

Use Case 14 below illustrates the use of header and payload signatures:

- RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the content along with the payload signature to the Registry.
- RC2 (Registry Client 2) retrieves RC1's content from the Registry.
- RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves the content, the response from the Registry Operator to RC2 contains the following:

- Payload containing the content that has been published by RC1.
- RC1's payload signature (represented by a ds:Signature element) over RC1's published content.
- Either the key for validating RC1's payload signature in ds:Signature element (using the KeyInfo element as specified in [XMLDSIG]) or RC1's identity so RC2 can obtain the validation key for signature itself (e.g. retrieve a certificate containing the public key for RC1).
- A ds:Signature element containing the header signature. Note that the Registry Operator not RC1 generates this signature.

Header Signature Requirements

This section specifies the requirements for generation, packaging and validation of a header signature. These requirements apply when the Registry Client and Registry Operator communicate using vanilla SOAP with Attachments. When ebXML MS is used for communication, then the [ebMS] specifies the generation, packaging and validation of XML signatures in the SOAP header. Therefore the header signature requirements do not apply when the ebXML MS is used for communication. However, payload signature generation requirements (specified elsewhere in this document) do apply whether vanilla SOAP with Attachments or ebXML MS is used for communication.

Packaging Requirements

A header signature is represented by a ds:Signature element. The ds:Signature element generated must be packaged in a <SOAP-ENV:Header> element.

The packaging of the ds:Signature element in the SOAP header field is shown below.

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
Content-Description: ebXML Message

-- MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: http://claiming-it.com/claim061400a.xml

<?xml version='1.0' encoding="utf-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
         …signature over soap envelope
      </ds:Signature>
    </SOAP-ENV: Header>
    <SOAP-ENV: Body>
```

```
        …
    </SOAP-ENV: Body>
</SOAP-ENV: Envelope>
```

Header Signature Generation Requirements

The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in this section.

A ds:Signature element contains:

- ds:SignedInfo
- ds:SignatureValue
- ds:KeyInfo

The ds:SignedInfo element must be generated as follows:

1. ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute, a client must be capable of signing using only the following Algorithm attribute:

   http://www.w3.org/2000/09/xmldsig/#dsa-sha1

   The above algorithm is being chosen because any XMLDSIG implementation conforming to the [XMLDSIG] specification supports it.

2. an optional ds:CanonicalizationMethod element. If it is not specified, then the canonicalization method must default to [XMLC14N] , which is the default specified by [XMLDSIG].

3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation. This signs the entire ds:Reference element:
   - Must include the following ds:Transform

     http://www.w3.org/2000/09/xmldsig#enveloped-signature

     This ensures that the signature (which is embedded in the <SOAP-ENV:Header> element) is not included in the signature calculation.

   - Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the ds:Reference element (The URI attribute is optional in the [XMLDSIG] specification.) . The URI attribute must be “”.
   - Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support the following digest algorithm:

     http://www.w3.org/2000/09/xmldsig/#sha1

- Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].

The ds:SignedValue must be generated as specified in [XMLDSIG].

The ds:KeyInfo element must be present and is subject to the requirements stated in the "KeyDistrbution and KeyInfo element" section of this document.

Header Signature Validation Requirements

The ds:Signature element for the ebXML message header must be validated by the recipient as specified by [XMLDSIG].

Header Signature Example

The following example shows the format of a header signature:

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
                <ds:CanonicalizationMethod>
                    Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
                </ds:CanonicalizationMethod>
                <ds:Reference URI= "">
                        <ds:Transform>
                                http://www.w3.org/2000/09/xmldsig#enveloped-signature
                        </ds:Transform>
                        <ds:DigestMethod DigestAlgorithm="./xmldsig#sha1">
                        <ds:DigestValue> ... </ds:DigestValue>
                </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>  ...  </ds:SignatureValue>
</ds:Signature>
```

Payload Signature Requirements

This section specifies the requirements for generation, packaging and validation of payload signatures. Unlike header signature, payload signature is packaged with the payload. Therefore the requirements apply regardless of whether the Registry Client and the Registry Operator communicate over vanilla SOAP with Attachments or ebMS.

[ebMS] does not specify the generation, validation and packaging of payload signatures. The specification is left upto the application (such as [ebMS]). So the requirements on the payload signatures augment the [ebMS] specification.

Packaging Alternatives

A payload signature is represented by a ds:Signature element and must be packaged as specified in this section.

There are several ways to package the payload signature with the payload. The following three options were considered.

1. A payload signature and the payload could be packaged as two separate SOAP attachments with an implicit ordering. The implicit order would have to be understood and followed by both the signer and the recipient.
2. Alternately, payload signature and payload could be packaged as a single SOAP attachment [SOAPATTACH], which is a MIME multipart/Related or multipart/mixed message. The payload can be packaged in the first body part and the signature can be packaged in the second body part. Instead of relying on ordering, the signature could be identified by a specific Content-Type.

3. The payload signature and the payload could be packaged as a single SOAP atatchment [SOAPATTACH], which is a MIME multipart/signed message as specified by [RFC1847].

   [RFC1847] specifies a framework within which security services may be applied to MIME body parts. The framework defines two new content subtypes:
   - multipart/signed
   - multipart/encrypted

   A multipart/signed type contains two MIME body parts:
   – the first body part contains content to be signed
   – second body part contains control information which can be used to verify the digital signature over the first body part

   [RFC1847] itself does not specify the control information in the second body part. Instead, [RFC1847] specifies a protocol parameter for the Content-Type. The protocol parameter determines the type and contents of the control information in the second body part. Other RFCs register the protocol and specify the contents. Such protocols include:

- application/pkcs-7 defined by S/MIME
- application/pgp-mime defined by PGP/MIME

There is however, currently no protocol (e.g. application/xmldsig) registered and specified for use within a multipart/signed message.

Note ToReviewers: *I am going to be investigating the possibility of a registering and XML signature as a protcol for [RFC1847].*

Option 2 is being chosen as the way to package the payload signature for the following reasons:

- Option 3 is consistent and works within the framework specified by [RFC1847] and is hence appears to be the right choice. However, the current expectation is that mulitpart/signed is not as widely supported as mulitpart/Related or multipart/mixed Content-Type. So option 2 is preferred over option 3.
- Option 2 provides aggregation at the level of the payload and is therefore preferrable to option 1.

Payload Signature Packaging Requirements

The payload signature must be packaged with the payload as specified here.

1. The payload and its signature must be enclosed in a MIME multipart message with a Content-Type of multipart/Related.
2. The first body part must be the content
3. The second body part must be the XML signature as specified in the “Payload Signature Generation Requirements”.

The above packaging assumes that the payload is always signed.

The packaging of the payload signature is shown below:

```
MIME-Version: 1.0
Content-Type: multipart/Related; boundary=MIME_boundary; type=text/xml;
Content-Description: ebXML Message

-- MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: http://claiming-it.com/claim061400a.xml

<?xml version='1.0' encoding="utf-8"?>
<SOAP-ENV: Envelope>
        …
 SOAP-ENV: Envelope>
```

--MIME_boundary
Content-Type: multipart/Related; boundary=PAYLOAD_boundary

--PAYLOAD_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: payload1
<SubmitObjectsRequest>…</SubmitObjectsRequest>

--PAYLOAD_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: payload2
<ds:Signature>
….. Payload signature
</ds: Signature>
--MIME_boundary

Payload Signature Generation Requirements

The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in this section.

1. ds:SignatureMethod must be present. For same reasons as noted in the Message Header Requirements", the client must be capable of signing using only the following Algorithm attribute:

   http://www.w3.org/2000/09/xmldsig/#dsa-sha1

2. An optional ds:CanonicalizationMethod element. If it is not specified, then the canonicalization method must default to [XMLC14N], which is the default specified by [XMLDSIG].

3. One Reference element to reference the payload that needs to be signed. The Reference element:
   - Must identify the payload to be signed using the URI attribute of the ds:Reference element. (The URI attribute is optional in the XMLDSIG specification.)
   - Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must be support the following digest algorithm:
     http://www.w3.org/2000/09/xmldsig/#sha1
   - Must contain a <ds:DigestValue> which is computed as specified in [XMLDSIG].

The ds:SignedValue must be generated as specified in [XMLDSIG].

The ds:KeyInfo element must be present and is subject to the requirements stated in the "KeyDistrbution and KeyInfo element" section of this document.

Message Payload Signature Validation

The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].

Payload Signature Example

The following example shows the format of the payload signature:

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
                <ds:CanonicalizationMethod>
                    Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
                </ds:CanonicalizationMethod>
                <ds:Reference URI=#Payload1>
                        <ds:DigestMethod DigestAlgorithm="./xmldsig#sha1">
                        <ds:DigestValue> ... </ds:DigestValue>
                </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>  ...  </ds:SignatureValue>
</ds:Signature>
```

Key Distribution and KeyInfo Element

To validate a signature, the recipient of the signature needs the validation key corresponding to the signer's key. The following use cases need to be handled:

- Registry Operator needs the validation key of the Registry Client to validate the signature
- Registry Client needs the validation key of the Registry Operator to validate the Registry's signature.
- Registry Client RC1 needs the validation key of Registry Client (RC2) to validate the content signed by RC1.

[XMLDSIG] provides a ds:*KeyInfo* element, which can be used to pass the recipient information for retrieving the validation key. ds:*KeyInfo* is an optional element as specified in [XMLDSIG]. This field together with the procedures outlined in this section is used to securely pass the validation key to a recipient.

ds:Keyinfo can be used to pass information such as keys, certificates, names etc. The intended usage of KeyInfo field for ebXML is as follows:

- Pass a DN (Distinguished Name). The recipient extracts the certificate corresponding to the DN name from its own key store location. The public key is then obtained from the certificate.

- Pass a X509 Certificate. This recipient extracts the X509 Certificate and the public key from the certificate.

The following assumptions are also made:

1. A Certificate is associated both with the Registry Operator and a Registry Client.
2. A Registry Client registers its certificate with the Registry Operator. The mechanism used for this is not specified here. The certificate registered with the Registry Operator must match the DN name, which is passed in KeyInfo.
3. A Registry Client obtains the Registry Operator's certificate and stores it in its own local key store. The mechanism is not specified here. The DN name in the Registry's certificate must match Registry's certificate has a DN name, which is passed in KeyInfo.

The usage of ds:KeyInfo field for different use cases is illustrated below:

- Use Case 9 and 11.
    1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.
    2. The DN name of RC is passed to the Registry in KeyInfo field of the header signature.
    3. The DN name of RC is passed to the Registry in KeyInfo field of the payload signature.
    4. Registry Operator retrieves the certificate corresponding to the DN name in the KeyInfo field in the header signature(since RC must have already registered its certificate).
    5. Registry Operator validates the header signature using the public key from the certificate.
    6. Registry Operator validates the payload signature by repeating steps 4 and 5 using the DN name from the KeyInfo field of the payload signature. This is only required if the Registry Operator wants to ensure that contents have not been modified on the network.

- Use Case 13 and 14
    1. RC1 signs the payload and SOAP envelope using its private key and publishes to the Registry.
    2. The DN name of RC1 is passed to the Registry in the KeyInfo field of the header signature.
    3. The DN name of RC2 is passed to the Registry in the KeyInfo field of the payload signature.
    4. RC2 retrieves content from the Registry.
    5. Registry Operator signs the SOAP envelope using its private key. Registry Operator sends RC1's content and the RC1's signature (signed by RC1).
    6. Registry Operator sends its own DN name in the KeyInfo field of the header signature. It also sends RC1's certificate in the KeyInfo field of the payload signature. Note that if PKI infrastructure were assumed, then it would have been

sufficient to send DN of RC1. RC2 would then have obtained the certificate from the PKI infrastructure (for e.g. using XKMS).

7. RC2 obtains Registry Operator's certificate using Registry Operator's DN name in the header signature and verifies Registry's signature.
8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates the signature on the payload.

Issue1: *The mechanism outlined for use case 13 and 14 requires the Registry Operator to insert the RC1's certificate in the payload signature. However, this may not be valid use model to assume. A typical model may be that the Registry Operator simply stores the payload and payload signature when submitted by a Registry Client without ever having to verify the payload signature. Then when the content is retrieved, the Registry Operator returnst the payload and payload signature to the Registry Client without any manipulation. In this scenario, it would not be possible for the Registry Operator to insert a certificate in the KeyInfo field. This scenario can be handled by requiring that the Registry Client to include a certificate instead of a DN name in the KeyInfo field of the payload signature when content is being submitted. The current expectation is that the Registry Client would be required to send a certificate in the payload signature instead of the DN name. The above use cases would change accordingly.*

Issue2: *In a header signature, sending of a DN name is not necessary if a reference to the contract between the Registry Client and the Registry Operator exists in the ebXML message (for e.g. CPAId). The certificate could be retrieved based on the reference contract id.*

To handle the above use cases, a Registry Client and Registry could be required to support the KeyInfo field. However, KeyInfo field has many different elements. So to make it simpler to conform to this specification, only a subset of the KeyInfo field is required to be supported (as specified below):

Based on the above use cases, a Registry Client and Registry must support the following:

- X509Subject element. This is a child element of X509Data which in turn is a child element of KeyInfo. X509SubjectName element can be used to indicate a X.509 subject distinguished name. [XMLDSIG] states that:

  "subject distinguished name SHOULD be compliant with RFC 2253[LDAP-DN]"

- X509Certificate element. This is a child element of X509Data which in turn is a child element of KeyInfo. This can be used to pass the certificate to the recipient. X509Certificate element contains a base64-encoded certificate.

The following table illustrates what is possible in the KeyInfo field.

	Header Signature		Payload Signature	
	DN Name	Certificate	DN Name	Certificate

RC to Registry	Yes	No	Yes	No (yes ok too)
Registry to RC	Yes	No	No	Yes

Relationship to XKMS

XKMS defines a protocol for key registration and distribution. It consists of two parts:

- X-KISS (XML Key Information Service Specification) for processing key information in an XML signature. This allows a client of XKMS to delegate processing (in part or whole) of KeyInfo field to XKMS.
- X-KRSS (XML Key Registration Service Specification) for registration of keys.

XKMS is not a requirement for this specification. However, a Registry Client or a Registry implementation may use XKMS for digital signature key registration and distribution.

Alignment with SOAP-SEC

SOAP Security Extensions [SOAPSEC] proposes a standard to use the XML Signatures to SOAP 1.1 messages with Attachments [SOAPATTACH]. The signature is packaged in a SOAP header entry <SOAP-SEC:Signature>.

The SOAP security extensions package a digital signature in a SOAP header entry. It is not applicable to payload signatures since the requirements for ebXML registries necessitate the payload signature to be packaged with the payload itself rather than in the SOAP header.

An additional reason that the SOAP Security Extensions are not used by this specification is that it appears to be work in progress.

Relationship to S/MIME

S/MIME is currently not a requirement for signing payloads in version 2.0. However, they can be accomodated in future versions of Registry Specifications.

Note To Reviewers: *I can add more information on use cases and how I expect S/MIME to be handled. But I ran out of time. I will add this for the next version of the document.*

Versioning Information

If a standard for signing SOAP Messages with Attachments is specified and that satisfies the use cases outlined earlier in the beginning of this document, then that standard could be adopted by future versions of ebXML Registry Specifications. This document assumes that there is sufficient versioning information which would allow future versions of Registry Clients and the Registry to distinguish between the different standards for signing.

References

[ebMS] Message Service Specification ebXML Transport, Routing & Packaging
http://www.ebxml.org/specs/ebMS.pdf

[SECRISK] Security Proposal for ebXML Registry V2

[RFC1847] Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted

[SOAPSEC] SOAP Security Extensions: Digital Signature
http://www.w3.org/TR/2001/NOTE-SOAP-dsig-20010206/

[SOAP]

[SOAPATTACH] SOAP Messages with Attachments
http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211/

[XMLDSIG] XML-Signature Syntax and Processing,
http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/