EbXML MS Processing Model with Signatures Proposal

Suresh Damodaran
Senior Software Architect
Sterling Commerce

Ver 0.2, Nov 06, 2001

Overview

We discuss the use of XML Signature and related issues in this document. We use the term "payload" to refer to any document that an application sends to another application. The payload could be an XML document, EDI document, or any other type of document.
Payload encryption is not addressed in this document.

Security Considerations

We discuss in this section the security considerations that led to the processing model in this document.

Envelope Signing vs. Payload Signing

It is helpful to differentiate between envelope signing and payload signing while considering ebXML MS. Envelope signing denotes "signing the message" that is sent from a MSH to another MSH for the purpose of data integrity[1]. The steps involved are the following.

1. Create digests for the SOAP envelope and each payload after the specified transforms are applied to the SOAP envelope and payload
2. Insert the digests calculated in step 1 into a ds:SignedInfo element. Canonicalize, and create the digest of the ds:SignedInfo element. Encrypt the ds:SignedInfo digest.

The envelope signing allows verification of the data integrity of the whole message. Note that while doing envelope signing, individual payloads are not signed individually. On the other hand, payload signing is to verify source authenticity of the payload after it has arrived and is stored at the destination. Payload signing is not a necessity, though could be useful in the following circumstances.

1. The sender of the payload may not be the creator or the "source" of the payload, and therefore, it may be useful to sign the payload separately under such circumstances.
2. The receiver of the payload does not wish to store the message envelope to which the payload is attached. Note that it is advisable to store the message envelope for non-repudiation purposes. If the envelope is stored and it is safe to assume that the sender of the payload is always identical to creator of the payload, then the receiver may do source authentication of the payload by creating the digest of the payload and matching it to the digest value in the signed ds:SignedInfo in the stored envelope.

Securing Payload Processing

Note that either signing the payload, or signing the digest of the payload (in ds:SignedInfo) secures the processing that a payload has to go through. There are two types of processing that need to be secured. These are (a) the transformations that need to be applied to a payload before a digest is created, (b) the package processing information, such as MIME Content-ID, and Content-Type. The first type of transformation are already secured as part of ds:Signature processing, whereas the second type of processing needs to be secured by entering them in eb:Manifest and signing them[2].

---

[1] EbXML MS specs uses the term "non-persistent security," which I believe is a misnomer because the receiver may persist the message signature for a long time along with the payloads for non-repudiation and source authentication of the payload under some circumstances.

[2] A potential extension of the notion of PayloadProcessing is to include other types of payload processing information to be communicated to the application. I believe MS already has this thought as evidenced by `<any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>` in `<eb:Reference>`

Canonicalization of the payload processing info is another important requirement. The ds:Transforms are canonicalized as part of canonicalization of ds:SignedInfo as they are expressed in XML. However, the packaging information, though expressed in XML as part of eb:Manifest, has the Content-Type and Content-ID values which need to be in a “normalized form” that can be understood by all MSHs. Note that the values are used for processing by the MIME Unpacker (see Section “Inbound”), and any valid MIME Header type and value would be in normalized form[3].

Outbound

Figure 1 describes the processing model when messages are constructed and sent out. The green boxes describe essential MSH processing, yellow boxes describe envelope signature related processing, and the pink box describes the optional processing. The payload could be signed or unsigned.

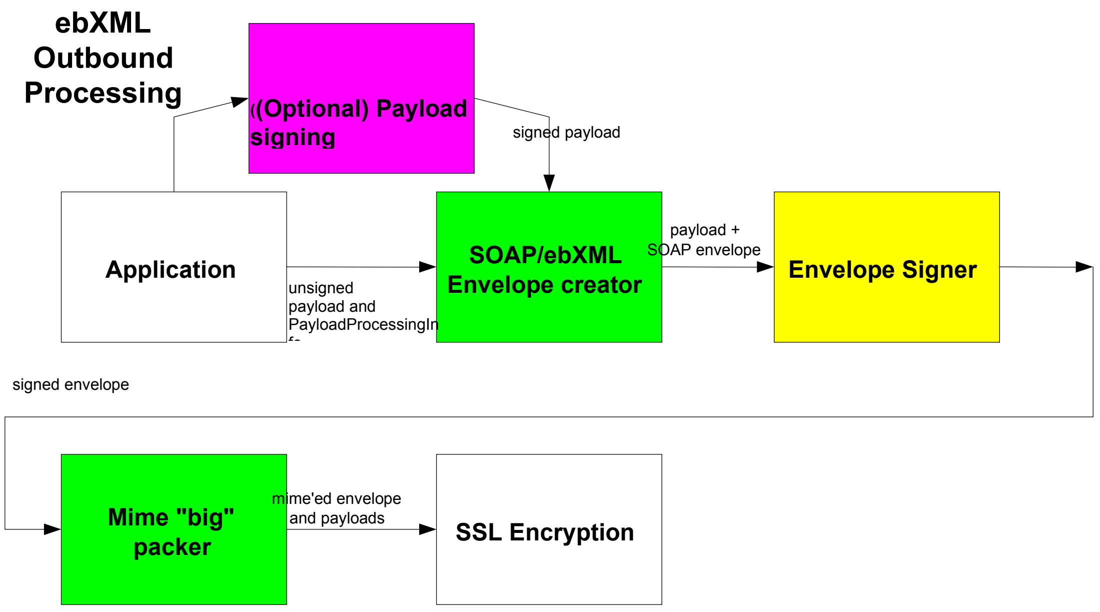
ebXML Outbound Processing
((Optional) Payload signing
signed payload
Application
unsigned payload and PayloadProcessingInfo
SOAP/ebXML Envelope creator
payload + SOAP envelope
Envelope Signer
signed envelope
Mime "big" packer
mime'ed envelope and payloads
SSL Encryption

Figure 1

The application creates one or more payloads and PayloadProcessingInfo for each payload. The payloads and PayloadProcessingInfo are passed to the SOAP/ebXML Envelope creator. Note that the payload is NOT required to be MIME encoded, though if encoded, the payload will be treated as non-XML. The PayloadProcessingInfo includes the following:

1. The desired MIME headers such as Content-ID, Content-Type
2. XML Signature transforms for each payload

The MS spec does not have to specify the structure and behavior of the PayloadProcessingInfo. The SOAP/ebXML Envelope creator does the packaging to create the envelope and payloads. The MIME headers can go into the eb:Reference element in eb:Manifest (the eb schema will need change to include the MIME headers). The transforms are stored in the ds:Reference element for each payload.

[3] We need to create another canonicalization algorithm for normalizing the MIME Header values (such as Multipart/related). We can describe in a separate document and use the transforms described in that document for our purposes. I do see that when a payload is MIME encoded by the application (before entering MSH) we could require canonicalization of the MIME message prior to signing. S/MIME may provide the right solution for this purpose.

The envelope and payloads are sent to a Signer[4], which signs the envelope using XML Signature. The envelope, after this processing, includes the ds:Signature element that contains the signature of the envelope. The signed envelope and payloads are sent to the MIME encoder that does the MIME packing. The MIME encoded document,"a single big document," is sent to the SSL encryptor that encrypts the message and sends the message over the wire (http/smtp).

Comments: There is a concern that the MIME header is repeated in the envelope and in the actual message. In response to this concern, consider the following points.

1. If the envelope is not signed, it is not REQUIRED to put the MIME headers in the manifest.
2. When the manifest is signed, it MUST include all headers that would appear in the MIME message transmitted, except those that could change en-route.
3. The duplicate headers (in eb:Reference and in MIME message) can help as a check for the message receiver to make sure that the MIME headers recovered from the message are not corrupted.

Inbound

Figure 2 below describes the inbound processing with signature.

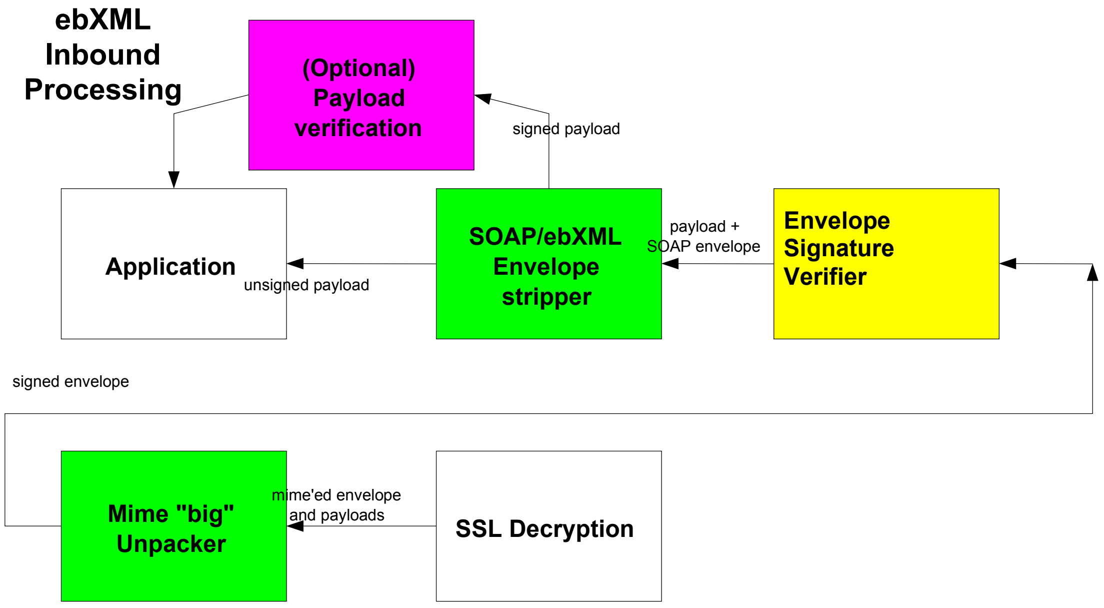

Figure 2

The SSL Decryption stage takes in the incoming document and presents the MIME encoded message that includes the envelope and payloads to the MIME Unpacker. MIME Unpacker unpacks the first MIME body, which is the envelope. MIME Unpacker unpacks the rest of the message using the payload MIME header information available in eb:Manifest in the envelope. The envelope signature is verified by the Envelope Verifier. The SOAP/ebXML Envelope stripper takes the verified message from the Envelope verifier, and delivers PayloadProcessingInfo as well as the payloads to the application.

[4] An alternate implementation may actually create multiple MIME messages (one for each payload) that includes the payload and mime header to the Signer. Yet another implementation may send a single MIME'd message that has all the payloads to the Signer. In both of these cases, the Signer has the onus of taking apart the payload and digesting and creating the Signature in ds:SignInfo

Proposed Addition to eb Schema

Insert the following in <eb:Reference> ("tns" qualification needs to be added in eb Schema)

```
    <xsd:element name="MIMEHeaders">
        <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="1">
                <xsd:element ref="MIMEHeader"/>
            </xsd:sequence>
            <xsd:attribute name="CanonicalizationMethod" type="xsd:string"
value="http://oasis.org#mimeManifest"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="MIMEHeader">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="parameter" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name=":comment" minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="fieldName" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:string" use="optional"/>
        </xsd:complexType>
    </xsd:element>
```

Canonicalization of MIME Header Values

These are early thoughts. RFC 2822, sections 2.1 and 2.2 and related sections form the basis of the canonicalization. Excluded from the canonicalization are Header fields whose values may change while in transit.