XML Signature 2.0
Strawman Proposal

Ed Simon, XMLsec Inc.
2007 November 6
Overview

- Single-pass Creation and Validation
- Direct Support for Multiple Signers
- Remove Need for Generalized Canonicalization
- Signature Profiles (Including Profiles for Algorithms)
V2.0 Structure

- Two child elements: 
  - `<SignedInfo>` and `<Signers>`
  - `<SignedInfo>`, like in V1.0, specifies the objects to be signed
  - `<Signers>` indicates the entities signing the signed info
<SignedInfo> Structure
<SignedInfo>  Example

</ds:Signature>

<ds:SignedInfo>
  <ds:Object Reference="http://xmlsec.com/logo.gif">
    <ds:DigestMethod Algorithm="...#sha1"/>
  </ds:Object>

  <ds:Object>
    <ds:DigestMethod Algorithm="...#md9"/>
    <ds:Value>Some text or XML</ds:Value>
    <ds:DigestValue>BaSe64DiGeSt</ds:DigestValue>
  </ds:Object>
</ds:SignedInfo>
<SignedInfo> Description

- Unlike v1.0, signed objects can be inside <SignedInfo>
- Allows Creator to specify the digest algorithm, specify the object, digest it and specify the digest value
- Both the digest algorithm and digest value are optional
- Objects may be specified by either value or reference
- Q: Do we also need to support enveloped objects outside of <SignedInfo>?
<SignedInfo> Description

- Support for transforms like in V1.0
- Signature properties, also like in V1.0, can be specified
- Q: Is <Manifest> support needed?
<Signers>/<Signer>

Structure
<Signers> Description

- Identifies one or more signers
- Identity of a signer specified using SAML <Subject> element (removing some of the SAML-specific attributes and child elements)
  - Allows encrypted identifier for signer
- <KeyInfo> can be specified for a Signer (optional)
- Signature algorithm (optional) and signature value (required) specified for each signer
<Signers> Example

... 
<ds:Signers>
  <ds:Signer>
    <saml:Subject>
      <saml:NameID Format="urn:oasis:names:...:emailAddress">
        edsimon@xmlsec.com
      </saml:NameID>
    </saml:Subject>
    <ds:SignatureMethod Algorithm="...#suite-b-elliptic-curve"/>
    <ds:SignatureValue>BaSe64SiGvAlUe01</ds:SignatureValue>
  </ds:Signer>
...
<Signers> Example

... <ds:Signer>
   <saml:Subject>
      <saml:EncryptedID>
         <xenc:EncryptedData>
            <xenc:CipherData>
               <xenc:CipherValue>encryptedxID
                  encryptedxID
               </xenc:CipherValue>
            </xenc:CipherData>
         </xenc:EncryptedData>
      </saml:EncryptedID>
   </saml:Subject>
   <ds:SignatureValue>BaSe64SiGvAlUe02</ds:SignatureValue>
</ds:Signer>
</ds:Signers>
Canonicalization

• To robustly hash the `<SignedInfo>` element, one needs to change its textual XML representation into octets

• The process of creating a standard textual XML representation is called canonicalization

• Generalized canonicalization's problem domain includes all the things one could do in XML, for every type of application, that may affect its text representation but not its infoset

• Therefore it is, inter alia, resource-intensive
Canonicalization

- Often better to use namespace-specific canonicalization
- Namespace-specific canonicalization is canonicalization that, rather than trying to encompass all use cases, is tailored and optimized for a particular namespace
<SignedInfo> Canonicalization

• In XML Signature, we need a reliable hash of <SignedInfo>,

• So we define – in the XML Signature specification – how to reliably hash <SignedInfo> which means we (the XML Signature group) need to define a canonical text form of <SignedInfo> for hashing

• Note: We can define the text form so precisely that the term canonicalization need not even be used
<SignedInfo> Canonicalization

- The canonical form of <SignedInfo> is:

  - '<SignedInfo' + ' Id="var"'{{?}
  + ' xmlns="../2008/xmldsig#">'
  + ('<Object' + ' Encoding="var"'{{?} '}
  + ' Id="var"'{{?} + ' MimeType="var"'{{?}
  + ' Reference="var"'{{?} + ' Type="var"'{{?}
  + '>
  + ('<Transforms>'
  + (('<Transform'
  + ' Algorithm="var"'{{?} '}
  + '>
  + (var + '</Transform>'){1..*}
  + '</Transforms>'){{?}
  + (('<Value>' + var + '</Value>'){{?}
  + (('<DigestMethod'
  + ' Algorithm="var"'{{?} '}
  + '>
  + (var + '</DigestMethod>'){{?}
  + (('<DigestValue'
  + var + '</DigestValue>'){{?}
  + '</Value>'){{?}
  + '</Object>'){1..*}
  + '</SignedInfo>'

XMLsec Inc. -- www.xmlsec.com

15
<SignedInfo> Canonicalization

• For canonicalizing the <SignedInfo> element:
  – For elements defined in the Signature schema, only the attributes defined in that schema for it – except for the @xmlns attribute on <SignedInfo> which points to the Signature namespace – are included in the canonicalized output
  – Those attributes are set as per general canonicalization (lexicographical order, double quotation marks, etc.)
<SignedInfo> Canonicalization

- The canonicalization of the content of the <Transform>, <DigestMethod>, and <Value> elements will be specified by XML Signature when that content is defined by XML Signature.
- For non-XML-Signature-defined XML content, XML Signature can specify a default of one of the existing XML-generic W3C canonicalization algorithms.
<SignedInfo> Canonicalization

- The default canonicalization of non-XML-Signature-specified content can be overridden by specifying that canonicalization in the transforms (or signature profile, if any)
- Note: A signature profile CANNOT replace the <SignedInfo> canonicalization algorithm illustrated earlier
- Canonicalization of URIs -- wrt xml:base -- still thinking about that ;)

...
Processing

• Single-pass on both creation and validation (correct me if I am wrong!)

• Order of element processing is `<SignedInfo>`, `<Object>`, `<DigestMethod>`, `<Value>`, `<DigestValue>`, (<`Object`> repeated as necessary), get hash of `<SignedInfo>`, `<Signer>`, `<SignatureMethod>`, `<SignatureValue>`, (<`Signer`> repeated as necessary)
Profiles

- The Signature element has a @ Profile attribute for specifying a profile to be used when interpreting the signature.
- The profile refines the generic XML Signature specification into one specifically suited to the application of the signature.
- Potential to have a list of profiles, each refining a subset of the prior profile (e.g. First profile is an XBRL profile, second profile restricts crypto algorithms).
Profiles

• A profile may specify (for example):
  – Allowance of attributes and specific values; for example, "Specify the Id attribute of the first SignedInfo Object element as 'object1'"
  – Use of transforms; e.g. "only Transform X and Transform Y are allowed"
  – Digest and Signature algorithms; use MD5 for hashing and DSA-SHA1 for signing

• Verification of compliance with signature profiles is up to applications; XML Signature only defines how to specify profiles