
Experience with XML Signature and Recommendations for future Development

Version 01, 1 August 2007

Document identifier:

Experience-Recommendation-Oracle-01

Contributors:

Pratik Datta, Oracle Corporation
Rich Levinson, Oracle Corporation
Prateek Mishra, Oracle Corporation
Vamsi Motukuru, Oracle Corporation
Ramana Turlapati, Oracle Corporation

Table of Contents

1 Introduction.....	2
2 Streaming XML-SIG.....	3
2.1 Why current XML Signature can't be streamed ?.....	3
2.1.1 What is a nodeset ?.....	3
2.1.2 Alternative to the nodeset – a SAX/StAX/XMLReader stream.....	3
2.2 Changes needed to the XML signature spec.....	4
2.2.1 Section 4.3.3.2 The reference processing model.....	4
2.2.2 Section 6.6.3 XPath filtering.....	6
2.2.3 XPath Filter 2.0.....	6
2.2.4 C14n.....	7
3 References.....	8

1 Introduction

Since the publication of [XML-SIG] there has been substantial implementation of the standard throughout the community. The standard has been applied to many different higher-level protocols, including SAML [SAML2.0] and Web Services Security [WSS1.1].

Based upon these experiences, there appear to be a number of areas where enhancements or simplifications would aid in broader use of the standard. This note proposes two such enhancements based on our implementation experience.

In addition to these enhancements, we would also suggest examination of the relationship between XML-SIG and specifications published after it was finalized (XPath 2.0, XML InfoSet 1.1, xml:id, DOM 3.0).

Finally, the proposals described in this document are the opinions of the authors of this note and not of their employer.

There are two main parts to our recommendation, each independent of the other:

1. Enhancements or profiling to ensure that [XML-SIG] can be used straightforwardly by a developer, with only modest knowledge of XML fundamentals, for signing an existing well-formed XML document. This would address the issue of perceived complexity of XML-SIG often referenced in discussions on the web and elsewhere.
 - There should be no requirement that schema information be available for the document or for any utility classes (e.g., wsu in [WSS1.1]).
 - Use of XPath, except when required for message protection, should be avoided wherever possible, using for example, attributes of type "xml:id" to reference signed XML elements within the document. If such an attribute is not present, it should be possible to add it to the signed element.
 - Ability to extract keys and signatures while making one-pass over the signed document. This type of message structure is found in [WSS1.1], for example, where a message should at first have the key, then the signature and then the elements that are signed. It allows developers to work with a simple interaction model that does not require them to reason about details of DOM or other XML representations
- 2) A profile that allows the use of [XML-SIG] to be used in streaming implementations. [XML-SIG] is currently written in way that is not amenable to streaming. The algorithms mentioned in the specification require an implementation to have the complete xml document in memory as a tree structure. As a result, this makes it very expensive for xml routers and gateways to scale as load increases. Also the current algorithms are not suitable for hardware based XML acceleration techniques, because hardware boards typically have little on board memory, so they work best when the job has a lot of processing and very not much main memory access.

The remainder of this document describes the proposal for streaming in greater detail, focusing on changes that would be required or areas of difficulty.

2 Streaming XML-SIG

In this section we outline a profile of XML Signature, which would cover many practical use cases. A producer of signatures that complies with this standard would produce this subset. A consumer which supports the existing XML signature standard will definitely be able to consume this signature, however a consumer that only supports this new proposal will not be able to process signature that are outside of this subset.

2.1 Why current XML Signature can't be streamed ?

- **References can go to any part of the document:** An XML Signature implementation, whether verifying or signing, needs to at first resolve references. One signature can have many references and each reference can be to any part of the document. This requires the entire document to be available.
- **Verification key can be anywhere in the document.** The key required to verify a signature can be anywhere in the document.
- **Each transform produces a large “nodeset” or “binary data”.** This whole nodeset or binary data must be kept in memory to be fed to the next transform.

The first two problems are easier to solve. A streaming XML Signature can work in two passes, in the first pass it can collect all the keys and all the signatures elements, and in the second pass it can perform the actual reference resolution, transforms, digesting and signing. The third problem is discussed in greater detail below.

2.1.1 What is a nodeset ?

The first step in a signature is to identify what is to be signed. There are many ways to do this – the simplest is to put an ID on an element, and mention that ID in the reference, this way the entire subtree under that element is marked for signing. Another way is to identify an element using an XPATH expression – this xpath expression can choose one complete subtree, or many disjoint subtrees or even parts of subtrees.. E.g you might want to sign all the credit card elements in the document, but exclude the merchant name from each of the elements.

The result of a selection is expressed as a nodeset.

A nodeset contains all the DOM nodes of the selection, including all attribute, text, element, comment nodes. It even has all the namespace nodes for all the elements including which explicitly specified or inherited (some implementations optimize away the namespace nodes) So nodeset has hundred of nodes for small document, and ten of thousands for large ones. A node set is like mathematical “set” – it is not ordered in any way. So the nodeset does not have information about parents and children, so an implementation needs to have the DOM tree as well as the nodeset.

2.1.2 Alternative to the nodeset – a SAX/StAX/XMLReader stream

We are proposing that we replace the nodeset with a SAX/StAX/XMLReader stream. A SAX stream consists of events like StartElement, EndElement, Text, etc. StAX (JSR 173) and XMLReader (.NET) have similar events but follow a pull model.

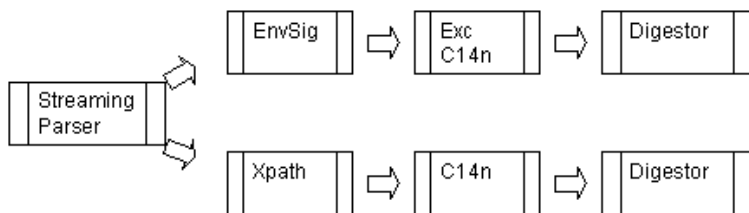
The set of events emitted by a streaming parser is somewhat similar to a nodeset in that it contains all elements and text nodes. But unlike the nodeset, this is ordered, and also for each element there is a begin element and an end element, which means that it has complete information and the XML signature implementation doesn't need the DOM tree to infer parent/child relationships.

102

103 When processing a nodeset the entire nodeset and the entire DOM tree need to be in memory, but not for
104 a streaming parser. In this case it sufficient to only store a stack with all the ancestors of the current
105 element. However some constraints need to be put on the XML Signature spec for this.

106 **“Pipeline” model for processing**

107 To sign/verify using the streaming parser the following processing model will be used. There will be
108 “pipeline” set up for each reference, and each pipeline will have one “processing node” for each transform.
109 As the streaming parser is going over the whole document, it will send each xml event to each pipeline.
110 The individual pipelines will process in parallel, but inside each pipeline, events go sequentially from one
111 processing node to the next one. The end result of each pipeline will be a digest value.



112 For example, if the signature has two references, where the first reference has an Enveloped Signature
113 transform, and another ExcC14n transform, and the second references has an XPath transform and a
114 C14n transform, then the pipe line will be setup as follows.

115 The Streaming Parser will generate XML events and send them to both pipelines, The EnvSig and Xpath
116 processing nodes will act as filters, they prevent some xml events from going to the next processing node.
117 The ExcC14n and C14n will convert from xml event stream to a binary stream, and the digester will
118 compute digests over the binary stream. The end result will be a set of digests, one for each reference.

2.2 Changes needed to the XML signature spec

2.2.1 Section 4.3.3.2 The reference processing model

This section talks about Xpath Nodesets. It says that Xpath is not necessary and a sufficiently functional replacement can be used.

A nodeset is more generic, so we want to define a more constrained nodeset as follows

1. If an attribute node exists in the nodeset its parent element must exist in the nodeset.
2. If an element or attribute that is in the nodeset, uses a namespace prefix, the corresponding namespace node that defines the prefix must exist in the nodeset.

These constraints enable the nodeset to be transformed into an valid XML document, with only one exception – the XML can have multiple root elements. We are not adding a third constraint to remove multiple root elements, because it is a common use case for signing, and secondly multiple root elements can be easily represented in SAX/StaX or other streaming parsers.

Example for the first constraint above:

Consider the following XML document (whitespace is only for legibility, assume there are no whitespace text nodes)

```
<A>text
  <B c="1">
    <D>hello</D>
    <E>world</E>
    <F>
      <G h="2"/>
    </F>
  </B>
</A>
```

A regular nodeset can have the following nodes

A, text, c.

But this nodeset cannot be represented as an xml event stream, because the attribute c's owner element (i.e. B) is not in the nodeset, and in SAX/Stax/XMLReader attributes can only be in the context of an element.

However it is possible for a nodeset to have the following nodes

text, D, hello, G, h.

This can be represented as the following xml event stream

```
text('text')
beginElement('D')
text('hello')
endElement('D')
beginElement('G')
attribute('h')
endElement('G')
```

Notice how this is not valid XML because it has multiple root elements, but it still streamable.

Example for the second constraint above:

Suppose the XML document is:

```
<n:A xmlns:n="http://foo" xmlns:n2="http://bar" >
<n:B>
```

```
163         <n:C/>
164     </n:B>
165 </n:A>
```

166 Even though the namespaces n and n2 are defined only once, in a nodeset they are present for every
167 descendant. i.e. the nodeset is actually

168 A, n, n2, B, n, n2, C, n, n2

169 Now a subset of this nodeset could have some namespace nodes missing e.g.

170 A, n, n2, B, n2, C, n, n2

171 In the above nodeset, B's n is missing. This is not valid XML, because B uses n. So it cannot be
172 represented in an xml event stream. This motivates the second constraint.

173 In SAX/StaX/XMLreader, namespaces are automatically inherited. The only way to remove a namespace
174 that is already defined in an ancestor is to explicitly redefine the prefix to an empty string.

175 **2.2.2 Section 6.6.3 XPath filtering**

176 XPath filtering involves evaluating the given XPath expression for every node in the nodeset.

177 XPath expressions can be very generic, so we want to limit xpath expressions as follows

- 178 1. Can only use the "ancestor", "parent", "self", "attribute" and "namespace" axes. (cannot use
179 "child", "descendant", "following", "preceding", "following-sibling", "preceding-sibling" axes). also
180 attributes and namespaces can only be off the self axis, not parent or ancestor.
- 181 2. Cannot use "string-value" of element nodes or root nodes as "string-value" involves looking at all
182 the descendant nodes and concatenating their text values. If an element has only one text node
183 child, a streaming parser might return the text as separate chunks, so it makes it very difficult
184 for implementations to compute this.
- 185 3. If the XPath transform is the first transform in a reference and the Reference URI is empty, then
186 the input to the XPath transform is the entire document. Otherwise the input may be a nodeset
187 that is a subset of the document. In this case Xpath expression may need to consider nodes that
188 are not in the input nodeset. This is not possible in a streaming model. So we put a constraint that
189 an XPath transform will only work on the nodes in the input nodeset.

190

191 These constraints will allow an implementation to evaluate the XPATH expresion keeping only the
192 following in memory

- 193 ● the current node,
- 194 ● all the attributes and namespaces of the current node
- 195 ● all the ancestor elements of the current node

196 which is typical for a streaming parser.

197 **2.2.3 XPath Filter 2.0**

198 XPath Filter 2.0 makes it easier for a DOM based XPath implementation, because the Xpath expressions
199 need to be evaluated only one, unlike XPath Filter 1.0, in which the need to be executed for every
200 nodeset.

201 However for a streaming implementation, the entire document is never available at once, so it is not
202 possible to evaluate the XPath expression and get a complete nodeset. Instead the implementation would
203 need to convert the XPath Filter 2.0 to an XPath Filter 1.0, and evaluate that.

204 Conversion from XPath Filter 2.0 to XPath Filter 1.0 is extremely difficult, so we need a very constrained

205 Xpath definition

- 206 ● Must use locationPath or a union of location paths
- 207 ● Must use self, child, descendant, attribute and namespace axes
- 208 ● Cannot use context size and context position

209

210 E.g. suppose the XPath 2.0 Filter contains

211 `//soap:Body//ns1:Echo[@a='23']`

212 to convert it we need to reverse it

213 `ancestor-or-self::ns1:Echo[@a='23']/ancestor-or-self::soap:Body`

214 **2.2.4 C14n**

215 Special attributes like `xml:base`, `xml:lang` and `xml:space` need special treatment because they affect the
216 output even if they are not part of the nodeset. . In the pipeline processing model, if a processing node
217 filters out one of these attributes, and if the attribute is inheritable, then the processing node should emit it
218 with the descendants.

3 References

- 220 [XML-SIG] D. Eastlake et al., *XML-Signature Syntax and Processing*, World Wide Web
221 Consortium, February 2002. <http://www.w3.org/TR/xmlsig-core/>.
- 222 [SAML2.0] S. Cantor et al, Assertions and Protocol for SAML 2.0, March 2005, [http://www.oasis-](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#samlv20)
223 [open.org/committees/tc_home.php?wg_abbrev=security#samlv20](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#samlv20)
- 224 [WSS1.1] A. Nadalin, et al., Web Services Security: SOAP Message Security 1.1, Sep 2005,
225 [http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-](http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf)
226 [SOAPMessageSecurity.pdf](http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf)