Guidelines for using XML Signatures with the OASIS Security Assertion Markup Language (SAML)

Draft 03, 27 October 2002

Document identifier:

draft-sstc-xmlsig-guidelines-03

Location:

http://www.oasis-open.org/committees/security/docs/

Editor:

Scott Cantor, The Ohio State University and Internet2 (cantor.2@osu.edu)

Contributors:

Phillip Hallam-Baker, Verisign
Christian Geuer-Pollmann, Apache XML Security
Merlin Hughes, Baltimore Technologies
Juergen Kremp, SAP

Abstract:

This document provides suggestions and best practices for using the XML Signature standard with SAML messages to fulfill the requirements of existing and future SAML profiles and bindings.

Status:

This is a draft document that supplements the SAML 1.0 committee specification and does not supersede or override it.

If you are on the security-services@lists.oasis-open.org list for committee members, send comments there. If you are not on that list, subscribe to the security-services-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to security-services-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Security Services TC web page (http://www.oasis-open.org/committees/security/).

Table of Contents

1 Introduction

This non-normative document describes the issues that one must consider when attaching digital signatures to SAML messages using the XML Signature standard **[XMLSig]**, and provides suggested best practices for the application of the standard to SAML 1.0 bindings and profiles, based on SAML and XML Signature implementation experience.

While this document does not supersede or contradict section 5 of the core SAML specification **[SAMLCore]**, section 5 lacks guidance in certain aspects of signature processing that insure interoperability, and was written in advance of the completion of new standards for signature formation that improve the robustness and efficiency of signature processing in SAML applications.

To the extent that SAML 1.0 implementations follow the guidelines in this document, future revisions of the SAML specification will be able to incorporate them normatively without sacrificing backward compatibility.

The following signature processing issues are discussed:

- Canonicalization
- Signature Coverage
- Signature Verification

Note that terms used in this document are as defined in the SAML glossary **[SAMLBind]** Prateek Mishra et al., *Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)*, http://www.oasis-open.org/committees/security/, OASIS, May 2002.

[SAMLGloss] unless otherwise noted.

2 Canonicalization

In XML Signature, canonicalization is the process of transforming a piece of content (formally, an octet stream or an XML node set) into an octet stream for input into a digest algorithm. The SAML 1.0 specification recommends, but does not require, the use of Inclusive Canonicalization **[InclC14N]**, the algorithm that is required of XML Signature implementations to support.

During the SAML specification process, a new Exclusive Canonicalization algorithm **[InclC14N]**
John Boyer, *Inclusive XML Canonicalization Version 1.0*,
http://www.w3.org/TR/xml-c14n/, World Wide Web Consortium.

[ExclC14N] was under development by the W3C Signature working group, and has since moved to Recommendation status. The purpose of the new algorithm is to correct certain deficiencies in namespace processing that arise when a signed XML fragment is placed within an XML context, such as a SOAP envelope, and then verified by a relying party while within that context. When the standard algorithm is used, namespaces from the surrounding context "bleed into" the canonicalized XML of the signed fragment, and invalidate the signature.

Since SAML assertions, responses, and requests are by their nature designed to be embeddable in other XML messages, the use of Exclusive Canonicalization is highly advantageous for many SAML applications, and this algorithm is therefore strongly suggested for use when signing SAML content.

Note that canonicalization algorithms are used with XML Signatures in two ways. They can be specified as the CanonicalizationMethod for an entire Signature (in which case canonicalization is applied specifically to the SignedInfo element). They can also be applied as a Transform within a Reference, in which case canonicalization applies to the specific data being signed for a given Reference. To avoid namespace problems, Exclusive Canonicalization must be used in both places.

2.1 Namespace Prefixes in Values

Exclusive Canonicalization can only insure that the necessary namespace prefixes are declared in the resulting octet stream when the prefixes are used in element and attribute names. When namespace prefixes are used in element or attribute values, as commonly occurs when using the QName schema type, any prefixes that would not otherwise be "visibly used" in the document must be declared in the "InclusiveNamespaces PrefixList" parameter to the canonicalization algorithm.

Since in most cases both the SAML assertion and SAML protocol namespaces will be bound to prefixes in the signed message, those prefixes should be included on the InclusiveNamespaces PrefixList when Exclusive Canonicalization is used as a transform. Other prefixes may also be needed if they are not "visibly used".

Furthermore, if either namespace is bound and/or used within the SignedInfo element itself, possibly as part of an XPath expression, then it must also be included on the InclusiveNamespaces PrefixList when Exclusive Canonicalization is used as a CanonicalizationMethod.

2.2 Best Practices

- **When possible, use the Exclusive Canonicalization algorithm when signing SAML assertions, requests, or responses, especially if the SAML object may be signed before insertion into a larger XML context.**
- **When used, the algorithm should be applied at both the Signature level, and as a Transform within the SAML Reference.**

- **Bind the SAML protocol and assertion namespaces (and any others used) to prefixes and include those prefixes in the InclusiveNamespaces PrefixList parameter to Exclusive Canonicalization.**

3 Signature Coverage

The XML Signature specification provides a plethora of techniques for embedding signatures in XML documents and for specifying what content (XML and otherwise) is to be signed. The SAML 1.0 specification mandates the use of the "enveloped signature" syntax, in which the Signature element is placed within the XML fragment that is being signed; the SAML 1.0 schema provides for the placement of optional Signature elements within the Assertion, Request and Response elements. The SAML 1.0 specification also makes explicit that such a signature must cover (thus include in its SignedInfo) all of the attributes and elements within the SAML element being signed, including any nested assertions and their Signatures.

The SAML specification does not, however, specify in detail how that signature coverage is to be expressed in the Signature element. As section 4 describes, one of the ways that an application can determine the content being signed is to check for specific references and transforms in the Signature; this makes it advantageous for SAML implementations to be consistent in their use of such transforms to express what is being signed. There are also efficiency advantages to certain approaches as well.

In the general case, any SAML signature should explicitly specify the containing SAML element (Assertion, Response, or Request) being signed. The following sections discuss various ways in which signatures can meet this goal. Exceptions to this rule are profile-specific (see section 5 for an example) because outside of a profile, there can be few assumptions about how a SAML object will be used. Recall also that a SAML Assertion can be signed and placed within a signed SAML Response, which illustrates the potential complexity.

Unfortunately, there is no mandatory reference syntax or transform algorithm in **[XMLSig]** that can, in general, isolate a subset of a document unless XML ID attributes on those elements are permitted, which SAML does not allow. Therefore, the methods presented below are a set of options that may be possible or impossible for different implementations depending on the features available.

3.1 References

The first step in specifying coverage with an enveloped signature is to include a single Reference element with a URI that directs the signature processor to include XML content from within the document containing the signature. This can be accomplished either with an empty URI ("") or with a fragment identifier ("#1234"). The latter syntax requires that it be possible to include special ID attributes in the signed element content, but SAML 1.0 does not permit this. Therefore an empty reference URI is the only mandatory syntax that can be used to indicate the "current document" as the source of data.

An additional optional syntax involves the use of an empty base URI with a fragment identifier containing other non-ID-based XPointer expressions such as "#xpointer(/)", which also represents the entire document, or a more complex expression that declares the specific element sub-tree to sign by referencing the root element. An example of this would be:

"#xmlns(samlp=urn:oasis:names:tc:SAML:1.0:protocol)xpointer(ancestor::samlp::Response[1])"

This a good way to isolate the object being signed without using extra Transforms (see below), but may not be supported by some libraries. If it is supported, it is recommended as the most straightforward method to use.

3.2 Transforms

The second step in specifying coverage, with any signature, is to include zero or more Transform elements that specify how to turn the results of evaluating the Reference URI into a final node set or octet stream for input into canonicalization and digest computation. For example, a special transform (http://www.w3.org/2000/09/xmldsig#enveloped-signature) is provided for specifying that a signature is enveloped, and is thus excluded from the node set containing it.

If the optional Reference syntax is used, or if the document contains only the content being signed, then the enveloped transform (with suitable canonicalization) is sufficient to complete the specification of a signature. If not, then additional transforms must be applied first. There are two primary XML subsetting algorithms defined at the present time, the original XPath Filter Transform described in http://www.w3.org/TR/xmldsig-core/#sec-XPath and the new version 2.0 transform defined in **[IncIC14N]** John Boyer, *Inclusive XML Canonicalization Version 1.0*, http://www.w3.org/TR/xml-c14n/, World Wide Web Consortium.

[ExcIC14N] John Boyer et al., *Exclusive XML Canonicalization Version 1.0*, http://www.w3.org/TR/xml-exc-c14n/, World Wide Web Consortium.

[XPath2]. Both are optional, and may not be available in some libraries.

While the version 2.0 specification is currently only a proposed recommendation by the W3C, it offers a tremendous advantage over the original in terms of both performance and clarity, and is highly suggested over its predecessor. The original transform is complex to implement efficiently, and forming accurate filter expressions with it is somewhat difficult, even for experienced developers. The new version is more straightforward to understand and is typically much faster to process, both important for a typical SAML application. The enveloped signature transform can also be carried out as part of a single compound XPath Filter 2 expression set, which further improves efficiency in some cases.

If signature coverage requires the use of an XPath transform, it is therefore suggested that it be specified using a single XPath Filter 2.0 Transform element containing two XPath filter expressions:

```
<ds:Transform Algorithm="http://www.w3.org/2002/06/xmldsig-filter2">
  <dsig-xpath:XPath Filter="intersect">
    here()/ancestor::samlp:Response[1]
  </dsig-xpath:XPath>
  <dsig-xpath:XPath Filter="subtract">
    here()/ancestor::ds:Signature[1]
  </dsig-xpath:XPath>
</ds:Transform>
```

The example above would apply when signing a Response. Requests and Assertions would be identical but for the substitution of "samlp:Request" or "saml:Assertion" in the first expression.

Finally, as described in section 2, the final Transform should usually be Exclusive Canonicalization to protect the signed content from namespace contamination. This is unnecessary if there is no surrounding context.

3.3 Best Practices

- **SAML signatures should include a single Reference element with an empty URI, a fragment identifier of "#xpointer(/)" or an XPointer expression such as the one described in section 3.1.**
- **If Transforms must be used to subset the document being signed, use of a compound XPath Filter 2.0 Transform, as described above, is the most efficient way to isolate the containing element for signature input and exclude the enveloped signature.**

- **Exclusive Canonicalization should be used as the final Transform unless the object will never be verified in an XML context other than the one in which it was signed.**

4 Signature Verification

When a signed message is received by a relying party, there are three main steps in the verification process: verifying that the message has not been tampered with in transit, evaluating the legitimacy of the signer (via certificate validation or other key verification techniques), and determining what portions of the message have been signed. The first two steps are well-defined by **[XMLSig]** and out of scope for SAML, respectively. The latter step is a subtle consideration that is expressed as "only what is signed is secure", and simply means that an XML Signature can expressively exclude portions of a message using transforms, and without examining those transforms (or at least their output) a relying party can be tricked by a signer into trusting data that has not been signed.

There are three primary methods an application can use to determine what has been signed, discussed in the following sections.

4.1 Parse the Octet Stream

The input to the digest algorithm is an octet stream derived by dereferencing the Reference URI, applying the Transforms, and performing canonicalization. While in general those bytes do not have to consist of well-formed or valid XML, in the case of SAML, they should represent exactly the containing element being signed, minus the enveloped signature. Therefore, the bytes can be fed back into a parser for reconstruction of the unsigned message. The message can then be validated (with the parser or by hand), insuring that only the signed data is consumed by the SAML application.

This method has the advantage of being easy to implement in most cases, provided the XML Signature implementation provides access to the octet stream that is the result of digest input processing. The disadvantage is that it may result in extra parsing if the application has already parsed the message to locate the Signature in the first place.

4.2 Node Set Comparison

When the result of applying transforms to a Reference is an XML node set, the relying party can apply the Transforms to the source material, and then compare the resulting node set against the nodes that are to be viewed as "secure". This can be a one time comparison or an ongoing filtering process.

The advantage to this approach is that it doesn’t require a full reparse of the resulting data, but the disadvantage is a certain degree of complexity above and beyond typical XML processing requirements.

However, if Exclusive Canonicalization is used as a final Transform to prevent namespace contamination, as this document recommends in many cases, then the output is an octet stream, and not a node set, which precludes this method.

4.3 Profiling Transforms

The final method requires that a pair of cooperating implementations at the sending and receiving ends agree on the Reference URI and the set of Transforms to be used. This allows a relying party to examine the Reference URI and Transform elements in the document after parsing, and compare its expectations to what the signer has provided.

This method is by far the most efficient, since no extra parsing is involved, but it requires agreement on the transforms to be used, which compromises interoperability if the specification does not mandate a specific profile. This is may be an acceptable tradeoff if performance trumps interoperability for an application. Since the SAML 1.0 specification does not outline conformance requirements in the area of digital signature interoperability, this method does not preclude conformance, though it does compromise interoperability.

4.4 Best Practices

- **As a matter of security, relying parties must determine that the correct portions of a signed SAML message have been included in the digested bytes.**
- **If interoperability is the paramount concern, then one of the methods described in sections 4.1 and 4.2 can be used to make this determination. Only 4.1 can be used if Exclusive Canonicalization is used as a transform.**
- **If performance is critical and interoperability is not a consideration, then the approaches described in section 3 can form the basis of an efficient profile between cooperating endpoints.**

5 SAML Profile Considerations

A SAML profile is an application of SAML messages and bindings to solve a specific technical problem, often including constraints on the messages and their contents and the methods of exchange. Some profiles may require the use of digital signatures to insure message integrity, for example when the message must be passed through an untrusted intermediary. Because profiles can include a less general set of assumptions than the SAML specification as a whole, there can be implications toward the use of digital signatures within a profile. This may suggest specific optimizations or additional constraints to simplify profile implementation and facilitate interoperability.

5.1 Browser/POST Profile

The Browser/POST profile, described in **[SAMLBind]**, is a mechanism for establishing an authenticated session between a browser and a web server by issuing a SAML authentication assertion within a signed SAML response from one web server in an HTML form, and posting it from the browser to the target web server. Because the response must travel in the clear through the browser (and possibly over the network, though use of SSL is recommended), it must be digitally signed by the asserting server and verified by the target server.

What makes this profile more restrictive than SAML in general is that there is no surrounding XML context for the SAML Response message. If the enclosed assertion is not signed (and this is not a requirement of the profile), then many of the issues that complicate canonicalization and the specification of signature coverage disappear. In the interest of maximizing the usability of libraries that do not support some of the optional features of **[XMLSig]**, a more restricted signature profile can be used to insure both security and interoperability.

With respect to canonicalization, since there can be no namespace declarations outside the message being signed, the original SAML recommendation of Inclusive Canonicalization can be followed if an implementation of Exclusive Canonicalization is not available for some reason. In addition, there is no need to specify a canonicalization algorithm in the transform step.

With respect to coverage, by profile definition, the SAML response signature must apply to the entire message. Since it is unnecessary to isolate a specific element in the message, an empty reference URI and the enveloped signature transform is sufficient to specify what is signed. This is advantageous because it relies solely on mandatory features of the signature specification and should be possible with any signature implementation.

The assertion that carries the basic authentication payload is specified by the profile as a short-lived assertion. This makes signing it a waste of resources. If however an additional, longer-lived, assertion is enclosed in the response (a legal though unspecified addition to the profile), it may be signed for some application-specific purpose. In that event, the issues of namespace contamination and signature coverage discussed in this document are relevant and these simplifications cannot be employed.

5.1.1 Profile Recommendations

- **In the signature over the SAML Response, use an empty ("") Reference URI with the Enveloped Signature Transform, and specify any appropriate Canonicalization Method.**
- **If an additional, enclosed SAML Assertion is to be signed, review the other options discussed in this document for canonicalization and signature coverage.**

6 Futures

XML represents an evolving set of specifications that will continue to advance in new directions in the future. **[XMLSig]** and related specifications are no exception. Since useful new canonicalization and transform algorithms are likely to appear with relevance to SAML and its profiles, these guidelines must be viewed as a snapshot of current practice only.

A particularly important area of developing work is in better accommodating schema validation during signature verification, since SAML currently defines only XML Schema documents as a normative description of SAML XML messages. For example, work has been done outside the W3C on a more schema-aware canonicalization algorithm that may be well suited to SAML applications (http://www.uddi.org/pubs/SchemaCentricCanonicalization-20020710.htm).

One particular problem SAML implementations that rely on schema validation must guard against is the presence of base64-encoded data inside signed SAML messages. Schema validation imposes certain normalization steps on schema processors that will result in invalidation of signatures in such cases. One example that may be common is the case in which a SAML assertion is signed, and placed within a SAML response that is also signed. Unless schema normalization is disabled, the values exposed in the resulting, parsed XML will not be the same as the values originally signed, though not in ways that are semantically different. There are imperfect workarounds, but this is an example of how future work will be important to insuring the robustness of future SAML implementations.

7 References

The following are cited in the text of this document:

[SAMLCore] Phillip Hallam-Baker et al., *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)*, http://www.oasis-open.org/committees/security/, OASIS, May 2002.

[SAMLBind] Prateek Mishra et al., *Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)*, http://www.oasis-open.org/committees/security/, OASIS, May 2002.

[SAMLGloss] Jeff Hodges et al., *Glossary for the OASIS Security Assertion Markup Language (SAML)*, http://www.oasis-open.org/committees/security/, OASIS, May 2002.

[XMLSig] Donald Eastlake et al., *XML-Signature Syntax and Processing*, http://www.w3.org/TR/xmldsig-core/, World Wide Web Consortium.

[InclC14N] John Boyer, *Inclusive XML Canonicalization Version 1.0*, http://www.w3.org/TR/xml-c14n/, World Wide Web Consortium.

[ExclC14N] John Boyer et al., *Exclusive XML Canonicalization Version 1.0*, http://www.w3.org/TR/xml-exc-c14n/, World Wide Web Consortium.

[XPath2] Joseph Reagle et al., *XML-Signature XPath Filter 2.0*, http://www.w3.org/2002/06/xmldsig-filter2/, World Wide Web Consortium.

Appendix A. Acknowledgments

The editors would like to acknowledge the contributions of the OASIS SAML Technical Committee, whose voting members at the time of publication were:

Allen Rogers, Authentica

Irving Reid, Baltimore Technologies

Krishna Sankar, Cisco Systems

Ronald Jacobson, Computer Associates

Hal Lockhart, Entegrity

Carlisle Adams, Entrust Inc.

Robert Griffin, Entrust Inc.

Robert Zuccherato, Entrust Inc.

Don Flinn, Hitachi

Joe Pato, Hewlett-Packard (co-chair)

Jason Rouault, Hewlett-Packard

Marc Chanliau, Netegrity

Chris McLaren, Netegrity

Prateek Mishra, Netegrity

Charles Knouse, Oblix

Steve Anderson, OpenNetwork

Rob Philpott, RSA Security

Jahan Moreh, Sigaba

Bhavna Bhatnagar, Sun Microsystems

Jeff Hodges, Sun Microsystems (co-chair)

Eve Maler, Sun Microsystems (former chair)

Aravindan Ranganathan, Sun Microsystems

Emily Xu, Sun Microsystems

Bob Morgan, University of Washington and Internet2

Scott Cantor, The Ohio State University and Internet2

Phillip Hallam-Baker, VeriSign

Appendix B. Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.