OASIS Draft, 279 MarchApril 2004

Document identifier:
sstc-saml-bindings-2.0-draft-098

Location:

Editors:
Frederick Hirsch, Nokia

Contributors:
TBD
Scott Cantor, Individual
Frederick Hirsch, Nokia
John Kemp, Nokia

Abstract:
This specification defines protocol bindings for the use of SAML assertions and request-response messages in communications protocols and frameworks.

Status:
This is a Draft.
Committee members should submit comments and potential errata to the security-services@lists.oasis-open.org list. Others should submit them to the security-services-comment@lists.oasis-open.org list (to post, you must subscribe; to subscribe, send a message to security-services-comment-request@lists.oasis-open.org with "subscribe" in the body) or use other OASIS-supported means of submitting comments. The committee will publish vetted errata on the Security Services TC web page (http://www.oasis-open.org/committees/security/).
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 web page for the Security Services TC (http://www.oasis-open.org/committees/security/ipr.php).
Table of Contents

1 Introduction ........................................................................................................................................... 5
  1.1 Protocol Binding Concepts .................................................................................................................. 5
  1.2 Notation ............................................................................................................................................... 5
2 Specification of Additional Protocol Bindings ...................................................................................... 6
  2.1 Guidelines for Specifying Protocol Bindings ..................................................................................... 6
  2.2 Process Framework for Describing and Registering Protocol Bindings ............................................ 6
3 Protocol Bindings ..................................................................................................................................... 7
  3.1 General Considerations ....................................................................................................................... 7
    3.1.1 Use of SSL 3.0 or TLS 1.0 ............................................................................................................. 7
  3.2 SAML SOAP Binding ......................................................................................................................... 7
    3.2.1 Required Information ................................................................................................................... 7
    3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding ....................................................... 8
      3.2.2.1 Basic Operation ....................................................................................................................... 8
      3.2.2.2 SOAP Headers ....................................................................................................................... 8
      3.2.2.3 Authentication ...................................................................................................................... 9
      3.2.2.4 Message Integrity .................................................................................................................. 9
      3.2.2.5 Confidentiality ..................................................................................................................... 9
    3.2.3 Use of SOAP over HTTP .............................................................................................................. 9
      3.2.3.1 HTTP Headers .......................................................................................................................... 9
      3.2.3.2 Authentication ........................................................................................................................ 9
      3.2.3.3 Message Integrity .................................................................................................................. 10
      3.2.3.4 Message Confidentiality ...................................................................................................... 10
      3.2.3.5 Security Considerations ........................................................................................................ 10
      3.2.3.6 Error Reporting ..................................................................................................................... 10
      3.2.3.7 Metadata Considerations ...................................................................................................... 10
      3.2.3.8 Example SAML Message Exchange Using SOAP over HTTP ........................................... 10
    3.3 Reverse SOAP (PAOS) Binding ........................................................................................................ 11
      3.3.1 Required Information .................................................................................................................. 11
      3.3.2 Overview ................................................................................................................................... 11
      3.3.3 HTTP Headers ........................................................................................................................... 12
    3.4 HTTP Redirect/POST Binding .......................................................................................................... 12
      3.4.1 Required Information ................................................................................................................ 12
      3.4.2 Overview .................................................................................................................................. 12
      3.4.3 Message Encoding .................................................................................................................... 12
        3.4.3.1 RelayState .............................................................................................................................. 13
        3.4.3.2 URL Encodings ................................................................................................................... 13
        3.4.3.3 Form Encoding .................................................................................................................... 15
3.6.5.4 Message Integrity .............................................................. 24
3.6.5.5 Message Confidentiality .................................................. 24
3.6.5.6 Security Considerations ................................................... 24
3.6.5.7 Error Reporting .............................................................. 24
3.6.5.8 Metadata Considerations ................................................ 24
3.6.5.9 Example SAML Message Exchange Using an HTTP URI ... 25
4 SAML Artifact Formats .......................................................... 26
  4.1 SAML 1.x Format .............................................................. 26
    4.1.1 Required Information ................................................... 26
    4.1.2 Format Details ............................................................ 26
  4.2 SAML 1.x Supplemental Format .......................................... 27
    4.2.1 Required Information ................................................... 27
    4.2.2 Format Details ............................................................ 27
5 URL Size Restriction (Non-Normative) .................................... 28
6 References ............................................................................. 29
1 Introduction

This document specifies SAML protocol bindings for the use of SAML assertions and request-response messages in communications protocols and frameworks. [SAMLCore] defines the SAML assertions and request-response messages themselves, and [SAMLProfile] defines specific usage patterns that reference both [SAMLCore] and bindings defined in this specification or elsewhere.

1.1 Protocol Binding Concepts

Mappings of SAML request-response message exchanges onto standard messaging or communication protocols are called SAML protocol bindings (or just bindings). An instance of mapping SAML request-response message exchanges into a specific communication protocol <FOO> is termed a <FOO> binding for SAML or a SAML <FOO> binding.

For example, a SAML SOAP binding describes how SAML request and response message exchanges are mapped into SOAP message exchanges.

The intent of this specification is to specify a selected set of bindings in sufficient detail to ensure that independently implemented products will interoperate.

Unless otherwise specified, a binding should be understood to support the transmission of any SAML protocol message derived from the sampl:RequestAbstractType and sampl:StatusResponseType types. Further, when a binding refers to "SAML requests and responses", it should be understood to mean any protocol messages derived from those types.

For other terms and concepts that are specific to SAML, refer to the SAML glossary [SAMLGloss].

1.2 Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 [RFC2119].

Listings of productions or other normative code appear like this.

Example code listings appear like this.

Note: Non-normative notes and explanations appear like this.

Conventional XML namespace prefixes are used throughout this specification to stand for their respective namespaces as follows, whether or not a namespace declaration is present in the example:

- The prefix saml: stands for the SAML assertion namespace [SAMLCore].
- The prefix sampl: stands for the SAML request-response protocol namespace [SAMLCore].
- The prefix ds: stands for the W3C XML Signature namespace, http://www.w3.org/2000/09/xmldsig# [XMLSig].

This specification uses the following typographical conventions in text: <SAMLElement>, <ns:ForeignElement>, Attribute, Datatype, OtherCode. In some cases, angle brackets are used to indicate non-terminals, rather than XML elements; the intent will be clear from the context.
2 Specification of Additional Protocol Bindings

This specification defines a selected set of protocol bindings, but others will possibly be developed in the future. It is not possible for the OASIS Security Services Technical Committee to standardize all of these additional bindings for two reasons: it has limited resources and it does not own the standardization process for all of the technologies used. The following sections offer guidelines for specifying bindings and a process framework for describing and registering them.

2.1 Guidelines for Specifying Protocol Bindings

This section provides a checklist of issues that MUST be addressed by each protocol binding.

1. Describe the set of interactions between parties involved in the binding. Any restrictions on applications used by each party and the protocols involved in each interaction must be explicitly called out.

2. Identify the parties involved in each interaction, including how many parties are involved and whether intermediaries may be involved.

3. Specify the method of authentication of parties involved in each interaction, including whether authentication is required and acceptable authentication types.

4. Identify the level of support for message integrity, including the mechanisms used to ensure message integrity.

5. Identify the level of support for confidentiality, including whether a third party may view the contents of SAML messages and assertions, whether the binding requires confidentiality, and the mechanisms recommended for achieving confidentiality.

6. Identify the error states, including the error states at each participant, especially those that receive and process SAML assertions or messages.

7. Identify security considerations, including analysis of threats and description of countermeasures.

8. Identify metadata considerations, such that support for a protocol binding for a particular protocol or profile can be advertised in an efficient and interoperable way.

2.2 Process Framework for Describing and Registering Protocol Bindings

For any new protocol binding to be interoperable, it needs to be openly specified. The OASIS Security Services Technical Committee will maintain a registry and repository of submitted bindings titled “Additional Bindings” at the SAML website [SAMLWeb] in order to keep the SAML community informed. The committee will also provide instructions for submission of bindings by OASIS members.

When a protocol binding is registered, the following information MUST be supplied:

1. Identification: Specify a URI that uniquely identifies this protocol binding.

2. Contact information: Specify the postal or electronic contact information for the author of the protocol binding.

3. Description: Provide a text description of the protocol binding. The description SHOULD follow the guidelines described in Section 2.1.

4. Updates: Provide references to previously registered protocol bindings or profiles that the current entry improves or obsoletes.
3 Protocol Bindings

The following sections define SAML protocol bindings sanctioned by the OASIS Security Services Technical Committee. These include the SAML SOAP binding and the SAML reverse SOAP (PAOS) bindings, as well as HTTP bindings that involve a user agent intermediary and URI dereferencing of assertions.

3.1 General Considerations

3.1.1 Use of SSL 3.0 or TLS 1.0

Unless otherwise specified, in any SAML binding's use of SSL 3.0 [SSL3] or TLS 1.0 [RFC2246], servers MUST authenticate to clients using a X.509 v3 certificate. The client MUST establish server identity based on contents of the certificate (typically through examination of the certificate’s subject DN field).

TLS-capable implementations MUST implement the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher suite and MAY implement the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite [AES].

FIPS TLS-capable implementations MUST implement the corresponding TLS_RSA_FIPS_WITH_3DES_EDE_CBC_SHA cipher suite and MAY implement the corresponding TLS_RSA_FIPS_AES_128_CBC_SHA cipher suite [AES] [FIPS].

SSL-capable implementations MUST implement the SSL_RSA_WITH_3DES_EDE_CBC_SHA cipher suite.

FIPS SSL-capable implementations MUST implement the FIPS ciphersuite corresponding to the SSL SSL_RSA_WITH_3DES_EDE_CBC_SHA cipher suite [FIPS].

3.2 SAML SOAP Binding

SOAP (Simple Object Access Protocol) 1.1 [SOAP1.1] is a specification for RPC-like interactions and message communications using XML and HTTP. It has three main parts. One is a message format that uses an envelope and body metaphor to wrap XML data for transmission between parties. The second is a restricted definition of XML data for making strict RPC-like calls through SOAP, without using a predefined XML schema. Finally, it provides a binding for SOAP messages to HTTP and extended HTTP.

The SAML SOAP binding defines how to use SOAP to send and receive SAML requests and responses. Like SAML, SOAP can be used over multiple underlying transports. This binding has protocol-independent aspects, but also calls out the use of SOAP over HTTP as REQUIRED (mandatory to implement).

3.2.1 Required Information


Contact information: security-services-comment@lists.oasis-open.org

Description: Given below.

Updates: urn:oasis:names:tc:SAML:1.0:bindings:SOAP-binding
3.2.2 Protocol-Independent Aspects of the SAML SOAP Binding

The following sections define aspects of the SAML SOAP binding that are independent of the underlying protocol, such as HTTP, on which the SOAP messages are transported.

3.2.2.1 Basic Operation

SOAP messages consist of three elements: an envelope, header data, and a message body. SAML request-response protocol elements MUST be enclosed within the SOAP message body.

SOAP 1.1 also defines an optional data encoding system. This system is not used within the SAML SOAP binding. This means that SAML messages can be transported using SOAP without re-encoding from the "standard" SAML schema to one based on the SOAP encoding.

The system model used for SAML conversations over SOAP is a simple request-response model.

1. A system entity acting as a SAML requester transmits a SAML request element within the body of a SOAP message to a system entity acting as a SAML responder. The SAML requester MUST NOT include more than one SAML request per SOAP message or include any additional XML elements in the SOAP body.

2. The SAML responder MUST return either a SAML response element within the body of another SOAP message or generate a SOAP fault. The SAML responder MUST NOT include more than one SAML response per SOAP message or include any additional XML elements in the SOAP body. If a SAML responder cannot, for some reason, process a SAML request, it MUST generate a SOAP fault. SOAP fault codes MUST NOT be sent for errors within the SAML problem domain, for example, inability to find an extension schema or as a signal that the subject is not authorized to access a resource in an authorization query. (SOAP 1.1 faults and fault codes are discussed in [SOAP1.1] §4.1.)

On receiving a SAML response in a SOAP message, the SAML requester MUST NOT send a fault code or other error messages to the SAML responder. Since the format for the message interchange is a simple request-response pattern, adding additional items such as error conditions would needlessly complicate the protocol.

[SOAP1.1] references an early draft of the XML Schema specification including an obsolete namespace. SAML requesters SHOULD generate SOAP documents referencing only the final XML schema namespace. SAML responders MUST be able to process both the XML schema namespace used in [SOAP1.1] as well as the final XML schema namespace.

3.2.2.2 SOAP Headers

A SAML requester in a SAML conversation over SOAP MAY add arbitrary headers to the SOAP message. This binding does not define any additional SOAP headers.

Note: The reason other headers need to be allowed is that some SOAP software and libraries might add headers to a SOAP message that are out of the control of the SAML-aware process. Also, some headers might be needed for underlying protocols that require routing of messages or by message security mechanisms.

A SAML responder MUST NOT require any headers in the SOAP message to correctly process the SAML message itself, but MAY require additional headers that address underlying routing or message security requirements.

Note: The rationale is that requiring extra headers will cause fragmentation of the SAML standard and will hurt interoperability.
3.2.2.3 Authentication

Authentication of both the SAML requester and the SAML responder is OPTIONAL and depends on the environment of use. Authentication mechanisms available from the underlying substrate protocol MAY be utilized to provide authentication. Section 3.2.3.2 describes authentication in the SOAP over HTTP environment. Authentication mechanisms designed specifically for SOAP message exchange MAY also be utilized.

3.2.2.4 Message Integrity

Message integrity of both SAML requests and SAML responses is OPTIONAL and depends on the environment of use. The security layer in the underlying substrate protocol MAY be used to ensure message integrity. Section 3.2.3.3 describes support for message integrity in the SOAP over HTTP environment. Integrity mechanisms designed specifically for SOAP message exchange MAY also be utilized.

3.2.2.5 Confidentiality

Confidentiality of both SAML requests and SAML responses is OPTIONAL and depends on the environment of use. The security layer in the underlying substrate protocol MAY be used to ensure message confidentiality. Section 3.2.3.4 describes support for confidentiality in the SOAP over HTTP environment. Confidentiality mechanisms designed specifically for SOAP message exchange MAY also be utilized.

3.2.3 Use of SOAP over HTTP

A SAML processor that claims conformance to the SAML SOAP binding MUST implement SAML over SOAP over HTTP. This section describes certain specifics of using SOAP over HTTP, including HTTP headers, error reporting, authentication, message integrity, and confidentiality.

The HTTP binding for SOAP is described in [SOAP1.1] §6.0. It requires the use of a SOAPAction header as part of a SOAP HTTP request. A SAML responder MUST NOT depend on the value of this header. A SAML requester MAY set the value of SOAPAction header as follows:

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

3.2.3.1 HTTP Headers

HTTP proxies MUST NOT cache responses carrying SAML assertions.

Both of the following conditions apply when using HTTP 1.1:

1. If the value of the Cache-Control header field is not set to no-store, then the SAML responder MUST NOT include the Cache-Control header field in the response.

2. If the Expires response header field is not disabled by a Cache-Control header field with a value of no-store, then the Expires field SHOULD NOT be included.

There are no other restrictions on HTTP headers.

3.2.3.2 Authentication

The SAML requester and responder MUST implement the following authentication methods:

1. No client or server authentication.

2. HTTP basic client authentication [RFC2617] with and without SSL 3.0 or TLS 1.0.

3. HTTP over SSL 3.0 or TLS 1.0 (see Section 3.1.1) server authentication with a server-side
4. HTTP over SSL 3.0 or TLS 1.0 mutual authentication with both server-side and a client-side certificate.

If a SAML responder uses SSL 3.0 or TLS 1.0, it MUST use a server-side certificate.

3.2.3.3 Message Integrity

When message integrity needs to be guaranteed, SAML responders MUST implement HTTP over SSL 3.0 or TLS 1.0 (see Section 3.1.1) with a server-side certificate.

3.2.3.4 Message Confidentiality

When message confidentiality is required, SAML responders MUST implement HTTP over SSL 3.0 or TLS 1.0 (see Section 3.1.1) with a server-side certificate.

3.2.3.5 Security Considerations

Before deployment, each combination of authentication, message integrity, and confidentiality mechanisms SHOULD be analyzed for vulnerability in the context of the specific protocol exchange, and the deployment environment. See specific protocol processing rules in [SAMLCore], and the SAML security considerations document [SAMLSec] for a detailed discussion.

[RFC2617] describes possible attacks in the HTTP environment when basic or message-digest authentication schemes are used.

3.2.3.6 Error Reporting

A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD return a "403 Forbidden" response. In this case, the content of the HTTP body is not significant.

As described in [SOAP1.1] § 6.2, in the case of a SOAP error while processing a SOAP request, the SOAP HTTP server MUST return a "500 Internal Server Error" response and include a SOAP message in the response with a SOAP fault element. This type of error SHOULD be returned for SOAP-related errors detected before control is passed to the SAML processor, or when the SOAP processor reports an internal error (for example, the SOAP XML namespace is incorrect, the SAML schema cannot be located, the SAML processor throws an exception, and so on).

In the case of a SAML processing error, the SOAP HTTP server MUST respond with "200 OK" and include a SAML-specified <Status> element in the SAML response within the SOAP body.

For more information about SAML status codes, see the SAML assertion and protocol specification [SAMLCore].

3.2.3.7 Metadata Considerations

Support for the SOAP binding SHOULD be reflected by indicating either a URL endpoint at which requests contained in SOAP messages for a particular protocol or profile are to be sent, or alternatively with a WSDL port/endpoint definition.

3.2.3.8 Example SAML Message Exchange Using SOAP over HTTP

Following is an example of a query that asks for an assertion containing an attribute statement from a SAML attribute authority.

```plaintext
POST /SamlService HTTP/1.1
```
Following is an example of the corresponding response, which supplies an assertion containing the attribute statement as requested.

HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:samlp="http://www.oasis-open.org/committees/security"
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns:ds="http://xml.apache.org/digester">
  <SOAP-ENV:Body>
    <samlp:Response
      xmlns:samlp="http://www.oasis-open.org/committees/security"
      xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
      xmlns:ds="http://xml.apache.org/digester"
      ResponseID='6c34f8b9c2d' MajorVersion='2' MinorVersion='0'
      IssueInstant='2004-03-27T08:42:00Z'>
      <ds:Signature>...</ds:Signature>
      <Status>
        <StatusCode value="samlp:Success"/>
        <Status>
        </Status>
      </Status>
      <saml:Assertion>
        <saml:AttributeStatement>
          ...
        </saml:AttributeStatement>
      </saml:Assertion>
    </samlp:Response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

3.3 Reverse SOAP (PAOS) Binding

This binding leverages the Reverse HTTP Binding for SOAP specification [PAOS]. Implementers MUST comply with the general processing rules specified in [PAOS] in addition to those specified in this document. In case of conflict, [PAOS] is normative.

3.3.1 Required Information

Identification: urn:oasis:names:tc:SAML:2.0:bindings:paos-binding

Contact information: security-services-comment@lists.oasis-open.org

Description: Given below.

Updates: None.

3.3.2 Overview

The reverse SOAP binding is a mechanism by which an HTTP requester can advertise the ability to act as...
a SOAP intermediary to an HTTP responder and handle a SAML protocol message from that responder in a SOAP envelope returned in the body of an HTTP response. The HTTP requester may or may not be the intended recipient of the SAML protocol message.

3.3.3 HTTP Headers

To indicate support for this binding, the HTTP requester includes The HTTP request from the client to the service provider is an HTTP GET request with additional HTTP headers indicating the ability to adhere to the reverse SOAP binding in its request(s). Specifically, the HTTP request meets the requirements outlined in the PAOS specification [PAOS] and this document as follows:

1. The HTTP Accept Header field SHOULD indicate an ability to accept the "application/vnd.paos+xml" content type.
2. The HTTP PAOS Header field SHOULD be present and specify the PAOS version with urn:liberty:paos:2003-08 at a minimum.

Additional PAOS headers such as the service value are specified by profiles that use the PAOS binding.

3.4 HTTP Redirect/POST Binding

Naming issue

Sometimes referred to as the "front-channel", the HTTP Redirect/POST binding defines a mechanism by which SAML protocol messages may be transmitted either within URL parameters or within the base64-encoded content of an HTML form control. Permissible URL length is infinite in the abstract, but quite unpredictably limited in actual practice across user agents. Therefore, specialized encodings must be used to carry XML messages on a URL, and larger or more complex message content should be sent using the POST mechanism.

3.4.1 Required Information

Contact information: security-services-comment@lists.oasis-open.org
Description: Given below.
Updates: Effectively replaces the binding aspects of the Browser/POST profile in [SAML 1.1].

3.4.2 Overview

The HTTP-Redirect/POST binding is intended for cases in which the SAML requester and responder need to communicate using an HTTP user agent (as defined in [RFC2616], HTTP 1.1) as an intermediary. This may be necessary, for example, if the communicating parties do not share a direct path of communication. It may also be needed if the responder requires an interaction with the user agent in order to fulfill the request, such as when the user agent must authenticate to it.

Note that some HTTP user agents may have the capacity to play a more active role in the protocol exchange and may support other bindings that use HTTP, such as the SOAP and Reverse SOAP bindings. This binding assumes nothing apart from the capabilities of a common web browser.

3.4.3 Message Encoding

There are two general ways to encoding messages for use with this binding. One is to encode the message into URL parameters and the other is to encode the XML instance in base-64 and then place the result in an HTML form control. When URL encoding is used, the HTTP GET method is used to deliver
the message, while POST is used with form encoding.

All endpoints that support this binding MUST support form encoding and at least one URL encoding technique. Endpoint URLs MUST NOT include query string parameters independent of the URL encoding of messages.

3.4.3.1 RelayState

An additional, optional piece of data called “RelayState” MAY be included with a SAML protocol message transmitted with this binding. The value MUST NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the message independent of any other protections that may or may not exist during message transmission.

If a SAML request message is accompanied by this optional data element, then the SAML responder MUST return its SAML protocol response using a binding that also supports a “RelayState” mechanism, and it MUST place the exact value it received with the request into the corresponding “RelayState” parameter in the response.

If no such value is included with a SAML request message, or if the SAML response message is being generated without a corresponding request, then the SAML responder MAY include a “RelayState” value to be interpreted by the recipient based on the use of a profile or prior agreement between the parties.

3.4.3.2 URL Encodings

There are many possible ways to encode XML into a URL, depending upon the constraints in effect. This specification defines - such one such methods without precluding others. Binding endpoints SHOULD indicate which encodings they support using metadata, when appropriate. Particular encodings MUST be uniquely identified with a URI when defined. It is NOT a requirement that all possible SAML messages be encodable with a particular set of rules, but the rules MUST clearly indicate which messages or content can or cannot be so encoded.

A URL encoding MUST place the message entirely within the URL query string, and MUST reserve the rest of the URL for the endpoint of the message recipient.

Plain URL Encoding

SAML protocol messages MAY be encoded by taking each specific element and attribute present in the original message, and embedding each of these as individual query parameter components of the URL. Such encoding may take place according to the rules specified for the application/x-www-form-urlencoded content type described in [HTML401]. Specifically, rules apply to the <query> component of an HTTP URL as described in [RFC2396].

The original XML protocol message MUST be encoded as follows:

- If an element or attribute is mandatory in a message, based on the specification of that message in [SAMLCore], then that element or attribute MUST appear in the URL-encoded form of the message.
- If an element or attribute is designated as optional in the specification of that message in [SAMLCore], and does not appear in the protocol message, then the corresponding data item MUST NOT appear in the URL-encoded message.
- Each item specified as part of the <query> component MUST be the value of the XML protocol message element or attribute.
- Several of the elements present in SAML protocol messages may have multiple values (for example, elements with maxOccurs="unbounded" specified in the XML schema). Such elements MUST be encoded as URL-encoded, space-separated lists of values (as shown below):

[TODO]<EXAMPLE>
• In certain cases (such as samlp:StatusCode Value), elements may be drawn from different XML namespaces, and qualified using an XML QName format. In such cases, values SHOULD be URL-encoded including the QName prefix that appears in the XML protocol message, with the prefixes saml and samlp defined to map to the namespaces specified in [SAMLCore] for those prefixes. Other QName prefixes MAY be defined in URL-encoded messages by including query parameters of the form xmlns:<prefix>, where <prefix> MUST be the QName prefix that appears in the protocol message for the element. Thus, xmlns:s might appear in a URL-encoded message as xmlns%3As=http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%3B. If XML QName prefixes are encoded using this method, the prefix definition MUST appear in the encoded URL before the QName value that uses the prefix.

• Elements that may appear with the same name in nested XML structures (such as <samlp:StatusCode>) MUST be encoded by producing a URL-encoded, space-separated string, illustrated in the example below:

  [TODO] <EXAMPLE>

• URL-encoded URL values SHOULD NOT exceed 80 bytes in length. Similarly, the URL-encoded value of the <RelayState> element SHOULD NOT exceed 80 bytes in length.

• Certain XML protocol messages support arbitrary extention, via the presence of an <Extension> element. Messages that contain such content MUST adhere to the following additional rules:

• Attribute values and elements based on complex content models MUST NOT be included in URL-encoded messages.

• All attributes and elements that are included in URL-encoded form MUST have an empty namespace, and unique local names.

• All values included SHOULD NOT exceed 80 bytes in length.

[TODO – SEE IF THERE ARE OTHER SPECIFIC PER-MESSAGE ISSUES FOR URL-ENCODING]

**GZIP Encoding**


SAML protocol messages MAY also be encoded into a URL via the gzip compression method (see [RFC1952]). In such an encoding, the following procedure should be applied to the original SAML protocol message’s XML serialization:

i) Any signature, including the containing `<ds:Signature>` XML element itself, MUST be removed from the SAML protocol message itself. Note that if the content of the message includes another signature, such as for example, a signed SAML assertion, this embedded signature is NOT removed. However, the length of such a message after encoding essentially precludes using this mechanism. Thus SAML protocol messages that contain signed content SHOULD NOT be encoded using this mechanism.

ii) The gzip encoding, as specified in [RFC1952] should be used then applied to the entire XML content of the original SAML protocol message, excepting any signature removed in i). The XML content MUST be encoded as a single “member” of the compressed data set. The OS field of the member header SHOULD be set to 255. The decoder of the data set MAY ignore the OS field and SHOULD treat the result as binary data since the XML self-describes the character encoding used.

iii) The gzip-encoded data MUST be subsequently base-64 encoded according to the rules specified in [RFC2045]. Linefeeds or other whitespace MUST be removed from the result.

iv) The base-64 encoded, gzipped data should be URL-encoded, and added to the URL as a `<query>` component with the string parameter which MUST be named SAMLRequestMessage (if the message is a SAML request) or SAMLResponse (if the message is a SAML response).
If the original SAML protocol message was signed using an XML digital signature, a new signature covering the encoded data as specified above, MUST be applied attached using the rules stated in this specification regarding URL-encoding and digital signatures.

If a “RelayState” value is to accompany the SAML protocol message, it MUST be URL-encoded and placed in an additional query string parameter named RelayState.

**URL-encoding and Digital Signatures**

XML digital signatures are not directly URL-encoded according to the above rules, due to space concerns. If the underlying XML SAML protocol message is signed with an XML signature [XMLDSig], any URL-encoded form of the message MUST be signed as follows:

- The signature algorithm identifier MUST be included as an additional `<query>` component parameter, named SigAlg. The value of this parameter MUST be the algorithm used to sign the URL-encoded SAML protocol message, specified according to [XMLSig][XMLDSig]. The signature itself MUST be omitted from this parameter.

- To construct the signature, a string consisting of the concatenation of the SigAlg and SAMLRequest (or SAMLResponse) query string parameters is constructed as follows:

  ```
  SAMLRequest=value&SigAlg=value
  or
  SAMLResponse=value&SigAlg=value
  ```

- The resulting string of bytes is the octet string to be fed into the signature algorithm. Any other content in the original query string is not included and not signed.

- The signature should cover the encoded content of the `<query>` component of the URL, that is, everything after the `?` character, other than the `Signature` portion of the `<query>` component. The `<query>` component MUST NOT contain any other content than the URL-encoded protocol message and the Signature and SigAlg `<query>` component parameters.

- The signature value MUST be encoded using the base-64 encoding (see [RFC2045]) with any whitespace removed and included as a `<query>` query string component parameter named Signature.

- Note: it should be noted that some characters in the base-64 encoded signature value may themselves require URL-encoding, before being added as a `<query>` component parameter.

- The following signature algorithms (see [XMLDSig]) and their URI representations, MUST be supported in support of this encoding mechanism:

  - DSAwithSHA1 – http://www.w3.org/200/09/xmlsig#dsa-sha1
  - RSAwithSHA1 – http://www.w3.org/200/09/xmlsig#rsa-sha1

[TODO—URL-encoded, signed message]

### 3.4.3.3 Form Encoding

A SAML protocol message is form-encoded by applying the base-64 encoding rules to the XML representation of the message and placing the result in a hidden form control within a form as defined by [HTML401], chapter 17. The HTML document MUST adhere to the XHTML specification, [XHTML]. The base-64 encoded value MAY be line-wrapped at a reasonable length in accordance with common practice.

If the message is a SAML request, then the form control MUST be named `SAMLRequest`. If the message is a SAML response, then the form control MUST be named `SAMLResponse`. Any additional form controls or presentation MAY be included but MUST NOT be required in order for the recipient to
The action attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using this binding to which the SAML message is to be delivered. The method attribute MUST be set to "POST".

If a "RelayState" value is to accompany the SAML protocol message, it MUST be placed in an additional hidden form control named RelayState.

Any technique supported by the user agent MAY be used to cause the submission of the form, and any form content necessary to support this MAY be included, such as submit controls and client-side scripting commands. However, the recipient MUST be able to process the message without regard for the mechanism by which the form submission is initiated.

3.4.4 Message Exchange

The system model used for SAML conversations via this binding is a request-response model, but these messages are sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request. The HTTP interactions before, between, and after these exchanges take place is undefined. Both the SAML requester and responder are assumed to be HTTP responders.

TODO: sequence diagram

1. A system entity acting as a SAML requester responds to an HTTP request from the user agent by returning a SAML request.

2. If URL-encoded, the request is returned encoded into the HTTP response's Location header, and the HTTP status MUST be either 303 or 302. The SAML requester MAY include additional presentation and content in the HTTP response to facilitate the user agent's transmission of the message, as defined in HTTP 1.1. [RFC2616]. The user agent delivers the SAML request by issuing an HTTP GET request to the SAML responder.

3. If form-encoded, then the request is returned in an [XHTML] document containing the form and content defined in section 3.4.3.2. The user agent delivers the SAML request by issuing an HTTP POST request to the SAML responder.

4. In general, the SAML responder MAY respond to the SAML request by immediately returning a SAML response or MAY return arbitrary content to facilitate subsequent interaction with the user agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to indicate the requester's (un)willingness to permit this kind of interaction. Eventually the responder SHOULD return a SAML response to the user agent to be returned to the SAML requester. The SAML response is returned in the same fashion as described for the SAML request.

3.4.4.1 HTTP Considerations

HTTP proxies and the user agent intermediary MUST NOT cache SAML requests or responses. HTTP headers SHOULD be used to protect against such caching behavior.

Both of the following conditions apply when using HTTP 1.1 during message exchange:

1. If the value of the Cache-Control header field is not set to no-store, then the HTTP responder returning a SAML request or response MUST NOT include the value in the Cache-Control header field in the HTTP response.

2. If the Expires HTTP response header field is not disabled by a Cache-Control header field with a value of no-store, then the Expires header SHOULD NOT be included in the HTTP response.

There are no other restrictions on the use of HTTP headers.

3.4.4.2 Authentication

Authentication of both the SAML requester and the SAML responder is OPTIONAL and depends on the
environment of use. The presence of the user agent intermediary means that the requester and responder
cannot rely on the transport layer for authentication, and must authenticate the messages received
instead. SAML provides for a signature on protocol messages for this purpose. Form-encoded messages
MAY be signed before the base-64 encoding is applied. URL-encoded messages MAY be signed if the
encoding method specifies a means for signing.

### 3.4.4.3 Message Integrity

Message integrity of both SAML requests and SAML responses is OPTIONAL and depends on the
environment of use. The presence of the user agent intermediary means that the requester and responder
cannot rely on the transport layer for integrity protection. SAML provides for a signature on protocol
messages for this purpose. Form-encoded messages MAY be signed before the base-64 encoding is
applied. URL-encoded messages MAY be signed if the encoding method specifies a means for signing.

### 3.4.4.4 Confidentiality

This binding MUST NOT be used if the content of the request or response cannot be exposed to the
user agent intermediary. Otherwise, confidentiality of both SAML requests and SAML responses is
OPTIONAL and depends on the environment of use. If confidentiality is necessary, SSL 3.0 or TLS 1.0
SHOULD be used to protect the message in transit between the user agent and the SAML requester and
responder.

### 3.4.5 Security Considerations

Before deployment, each combination of authentication, message integrity, and confidentiality
mechanisms SHOULD be analyzed for vulnerability in the context of the specific protocol exchange, and
the deployment environment. See specific protocol processing rules in [SAMLCore], and the SAML
security considerations document [SAMLSec] for a detailed discussion.

In general, this binding relies on message-level authentication and integrity protection via signing and
does not support confidentiality of messages from the user agent intermediary.

### 3.4.6 Error Reporting

A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD
return a `<StatusResponse>` response message with a second level `<StatusCode>` value of
`RequestDenied`.

HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate
failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

For more information about SAML status codes, see the SAML assertion and protocol specification
[SAMLCore].

### 3.4.7 Metadata Considerations

Support for the HTTP-Redirect/POST binding SHOULD be reflected by indicating URL endpoints at which
requests and responses for a particular protocol or profile should be sent. Either a single endpoint or
distinct request and response endpoints should be possible.

### 3.4.8 Example SAML Message Exchange Using HTTP-Redirect/POST

TBD
3.5 HTTP Artifact Binding

Similar to the Redirect/POST binding, the artifact binding provides a composable substitute in which the SAML request, response, or both are transmitted by reference using a small stand-in called an artifact. A separate, synchronous binding, such as the SAML SOAP binding, is used to exchange the artifact for the actual protocol message using the Artifact Protocol defined in [SAMLCore]. It is composable with the Redirect/POST binding in that they can easily be combined such that the request can be transmitted with one binding and the response with the other because they use the same transport, use the same kind of user agent intermediary, and have similar general processing rules.

3.5.1 Required Information

Contact information: security-services-comment@lists.oasis-open.org
Description: Given below.
Updates: Effectively replaces the binding aspects of the Browser/Artifact profile in [SAML 1.1].

3.5.2 Overview

The HTTP Artifact binding is intended for cases in which the SAML requester and responder need to communicate using an HTTP user agent as an intermediary, but the intermediary's limitations preclude or discourage the transmission of an entire message (or message exchange) through it. This may be for technical reasons or because of a reluctance to expose the message content to the intermediary (and if the use of encryption is not practical).

Note that because of the need to subsequently dereference the artifact using another more direct binding, such as SOAP, a direct communication path must exist between the SAML message sender and recipient in the reverse direction of the artifact's transmission (the receiver of the message and artifact must be able to send an artifact request back to the artifact sender). The message sender must also maintain state while the artifact is pending, which has implications for load-balanced environments.

3.5.3 Message Encoding

There are two methods of encoding an artifact for use with this binding. One is to encode the artifact into a URL parameter and the other is to place the artifact in an HTML form control. When URL encoding is used, the HTTP GET method is used to deliver the message, while POST is used with form encoding.

All endpoints that support this binding MUST support both techniques. Endpoint URLs MUST NOT include query string parameters independent of the URL encoding of messages.

3.5.3.1 RelayState

An additional, optional piece of data called “RelayState” MAY be included with a SAML artifact transmitted with this binding. The value MUST NOT exceed 80 bytes in length and SHOULD be integrity protected by the entity creating the message independent of any other protections that may or may not exist during message transmission.

If an artifact that represents a SAML request is accompanied by this optional data element, then the SAML responder MUST return its SAML protocol response using a binding that also supports a “RelayState” mechanism, and it MUST place the exact value it received with the artifact into the corresponding “RelayState” parameter in the response.

If no such value is included with an artifact representing a SAML request, or if the SAML response message is being generated without a corresponding request, then the SAML responder MAY include a
“RelayState” value to be interpreted by the recipient based on the use of a profile or prior agreement between the parties.

### 3.5.3.2 URL Encoding

To encode an artifact into a URL, the artifact value is **URL-encoded** and placed in a query string parameter named `SAMLart`. If a “RelayState” value is to accompany the SAML artifact, it **MUST** be URL-encoded and placed in an additional query string parameter named `RelayState`. In addition, if the artifact represents a SAML response message that contains a `<RelayState>` value, that value **MUST** be placed in a second query string parameter named “RelayState”. This is to ensure that any relevant state is included in case the artifact cannot be dereferenced.

### 3.5.3.3 Form Encoding

A SAML artifact is form-encoded by placing it in a hidden form control within a form as defined by [HTML401], chapter 17. The HTML document **MUST** adhere to the XHTML specification, [XHTML]. The form control **MUST** be named `SAMLart`. In addition, if the artifact represents a SAML response message that contains a `<RelayState>` value, that value **MUST** be placed in a second hidden form control named “RelayState”. This is to ensure that any relevant state is included in case the artifact cannot be dereferenced. Any additional form controls or presentation **MAY** be included but **MUST NOT** be required in order for the recipient to process the artifact.

The action attribute of the form **MUST** be the recipient's HTTP endpoint for the protocol or profile using this binding to which the artifact is to be delivered. The method attribute **MUST** be set to "POST".

If a “RelayState” value is to accompany the SAML artifact, it **MUST** be placed in an additional hidden form control named `RelayState`.

Any technique supported by the user agent **MAY** be used to cause the submission of the form, and any form content necessary to support this **MAY** be included, such as submit controls and client-side scripting commands. However, the recipient **MUST** be able to process the artifact without regard for the mechanism by which the form submission is initiated.

### 3.5.4 Artifact Types

With respect to this binding, an artifact is a short, opaque string. Different types can be defined and used without affecting the binding. The important characteristics are the ability of an artifact receiver to identify the issuer of the artifact, resistance to tampering and forgery, uniqueness, and compactness.

TODO: define a normative artifact format for 2.0. SAML 1.x and ID-FF 1.2 define the following artifact formats:

- **SourceID** as a fixed-length opaque value
- **SourceID** as a URL location at which to dereference the artifact
- **SourceID** as a hash of the Liberty ProviderID, the unique key that references metadata

The most interesting answer may be a fourth: using the ProviderID directly as the SourceID, and using metadata to publish the actual location. The downside is the loss of fixed length, but the value of that is unclear.

### 3.5.5 Message Exchange

The system model used for SAML conversations via this binding is a request-response model in which an artifact reference takes the place of the actual message content, and the artifact reference is sent to the user agent in an HTTP response and delivered to the message recipient in an HTTP request. The HTTP interactions before, between, and after these exchanges take place is undefined. Both the SAML requester and responder are assumed to be HTTP responders.
Additionally, it is assumed that upon receipt of an artifact by way of the user agent, the recipient invokes a separate, direct exchange with the artifact issuer using the Artifact Protocol defined in [SAMLCore]. This exchange MUST use a binding that does not use the HTTP user agent as an intermediary, such as a SOAP binding. Upon the successful acquisition of a SAML protocol message, the artifact is discarded and the processing of the primary SAML protocol exchange resumes (or ends, if the message is a response).

Issuing and delivering an artifact, along with the subsequent dereference, constitutes half of the overall SAML protocol exchange. This binding can be used to deliver either or both halves of a SAML protocol exchange. A binding composable with it, such as the HTTP-Redirect/POST binding, MAY be used to carry the other half of the exchange. The following sequence assumes that the artifact binding is used for both halves.

TODO: sequence diagram

1. A system entity acting as a SAML requester responds to an HTTP request from the user agent by returning an artifact representing a SAML request.

2. If URL-encoded, the artifact is returned encoded into the HTTP response's Location header, and the HTTP status MUST be either 303 or 302. The SAML requester MAY include additional presentation and content in the HTTP response to facilitate the user agent's transmission of the message, as defined in HTTP 1.1, [RFC2616]. The user agent delivers the artifact by issuing an HTTP GET request to the SAML responder.

3. If form-encoded, then the artifact is returned in an [XHTML] document containing the form and content defined in section 3.5.3. The user agent delivers the artifact by issuing an HTTP POST request to the SAML responder.

4. The SAML responder determines the SAML requester by examining the artifact (the exact process depends on the type of artifact), and issues an <ArtifactRequest> containing the artifact to the SAML requester using a direct SAML binding, temporarily reversing roles. Assuming the necessary conditions are met, the SAML requester returns an <ArtifactResponse> containing the original SAML request message it wishes the responder to process.

5. In general, the SAML responder MAY respond to the SAML request by immediately returning a SAML artifact or MAY return arbitrary content to facilitate subsequent interaction with the user agent necessary to fulfill the request. Specific protocols and profiles may include mechanisms to indicate the requester's (un)willingness to permit this kind of interaction. Eventually the responder SHOULD return a SAML artifact to the user agent to be returned to the SAML requester. The SAML response artifact is returned in the same fashion as described for the SAML request artifact.

6. The SAML requester determines the SAML responder by examining the artifact, and issues an <ArtifactRequest> containing the artifact to the SAML responder using a direct SAML binding. Assuming the necessary conditions are met, the SAML responder returns an <ArtifactResponse> containing the SAML response message it wishes the requester to process.

3.5.5.1 HTTP Considerations

HTTP proxies and the user agent intermediary MUST NOT cache SAML artifacts. HTTP headers SHOULD be used to protect against such caching behavior.

Both of the following conditions apply when using HTTP 1.1 during message exchange:

1. If the value of the Cache-Control header field is not set to no-store, then the HTTP responder returning a SAML artifact MUST NOT include the Cache-Control header field in the HTTP response.

2. If the Expires HTTP response header field is not disabled by a Cache-Control header field with a value of no-store, then the Expires header SHOULD NOT be included in the HTTP response.

There are no other restrictions on the use of HTTP headers.
3.5.2 Authentication

This binding uses a combination of indirect transmission of a message reference followed by a direct exchange to return the actual message. As a result, the message reference (artifact) need not itself be authenticated, but the callback request/response exchange that returns the actual message MAY be mutually authenticated, depending on the environment of use.

If the actual SAML protocol message is intended for a specific recipient, then the artifact’s sender issuer MUST authenticate the sender of the subsequent <ArtifactRequest> message or sender before returning the actual message.

3.5.3 Message Integrity

This binding uses a combination of indirect transmission of a message reference followed by a direct exchange to return the actual message. As a result, the message reference (artifact) need not itself be integrity protected, but the callback request/response exchange that returns the actual message MAY be protected, depending on the environment of use.

3.5.4 Confidentiality

The transmission of an artifact to and from the user agent MUST be protected with confidentiality; SSL 3.0 or TLS 1.0 SHOULD be used. The callback request/response exchange that returns the actual message MAY be protected, depending on the environment of use.

3.5.6 Security Considerations

Before deployment, each combination of authentication, message integrity, and confidentiality mechanisms SHOULD be analyzed for vulnerability in the context of the specific protocol exchange, and the deployment environment. See specific protocol processing rules in [SAMLCore], and the SAML security considerations document [SAMLSec] for a detailed discussion.

In general, this binding relies on the artifact as a hard-to-forge short-term reference and applies other security measures to the callback request/response that returns the actual message.

3.5.7 Error Reporting

A SAML responder that refuses to perform a message exchange with the SAML requester SHOULD return a <StatusResponse> response message with a second level <StatusCode> value of RequestDenied.

HTTP interactions during the message exchange MUST NOT use HTTP error status codes to indicate failures in SAML processing, since the user agent is not a full party to the SAML protocol exchange.

If the issuer of an artifact receives an <ArtifactRequest> message that it can understand, it MUST return an <ArtifactResponse> with a <StatusCode> value of Success, even if it does not return the corresponding message (for example because the artifact requester is not authorized to receive the message or the artifact is no longer valid).

For more information about SAML status codes, see the SAML assertion and protocol specification [SAMLCore].

3.5.8 Metadata Considerations

Support for the HTTP Artifact binding SHOULD be reflected by indicating URL endpoints at which requests and responses for a particular protocol or profile should be sent. Either a single endpoint or distinct request and response endpoints should be possible. An endpoint for processing
3.5.9 Example SAML Message Exchange Using HTTP Artifact

TBD

3.6 SAML URI Binding

URIs are a protocol-independent means of referring to a resource. This binding is not a general SAML request/response binding, but rather supports the encapsulation of an <AssertionIDRequest> message with a single <AssertionIDReference> into the resolution of a URI. The result of a successful request is a SAML <Assertion> element.

Like SOAP, URI resolution can occur over multiple underlying transports. This binding has protocol-independent aspects, but also calls out the use of HTTP with SSL 3.0 or TLS 1.0 as REQUIRED (mandatory to implement).

3.6.1 Required Information


Contact information: security-services-comment@lists.oasis-open.org

Description: Given below.

Updates: None

3.6.2 Protocol-Independent Aspects of the SAML URI Binding

The following sections define aspects of the SAML URI binding that are independent of the underlying transport protocol of the URI resolution process.

3.6.2.1 Basic Operation

A SAML URI reference identifies a specific SAML assertion. The result of resolving the URI MUST be a message containing the assertion, or a transport protocol-specific error. If the transport protocol permits the returned content to be described, such as HTTP 1.1. [RFC2616], then the assertion MAY be encoded in whatever format is permitted by that protocol. If not, the assertion MUST be returned in a form which can be unambiguously interpreted as or transformed into an XML serialization of the assertion.

It MUST be the case that if the same URI reference is resolved in the future, then either the same SAML assertion, or an error, is returned. That is, the reference MAY be persistent but MUST consistently reference the same assertion, if any.

3.6.2.2 Authentication

Authentication of the requester, responder, and the assertion itself are OPTIONAL and depend on the environment of use. Authentication mechanisms available from the underlying substrate protocol MAY be utilized to provide authentication. The resulting assertion MAY also be signed. Section 3.6.5.3 describes authentication during HTTP URI resolution.

3.6.2.3 Message Integrity

Message integrity of the request, response, and assertion are OPTIONAL and depend on the environment
of use. The security layer in the underlying substrate protocol MAY be used to ensure message integrity. Section 3.6.5.4 describes support for message integrity during HTTP URI resolution.

3.6.2.4 Confidentiality

Confidentiality of the request, response, and the assertion itself are OPTIONAL and depend on the environment of use. The security layer in the underlying substrate protocol MAY be used to ensure message confidentiality. Section 3.6.5.5 describes support for confidentiality during HTTP URI resolution.

3.6.3 General Security Considerations

Before deployment, each combination of authentication, message integrity, and confidentiality mechanisms SHOULD be analyzed for vulnerability in the context of the specific protocol exchange, and the deployment environment. See specific protocol processing rules in [SAMLCore], and the SAML security considerations document [SAMLSec] for a detailed discussion.

Indirect use of a SAML assertion presents dangers if the binding of the reference to the result of the indirection is not secure. The particular threats and their severity depend on the use to which the assertion is being put. In general, the result of resolving a URI reference to a SAML assertion SHOULD only be trusted if the requester can be certain of the identity of the responder and that the contents have not been modified in transit.

It is often not sufficient that the assertion itself be signed, because URI references are by their nature somewhat opaque to the requester. The requester SHOULD have independent means to insure that the assertion returned is actually the one that is represented by the URI; this is accomplished by both authenticating the responder and relying on the integrity of the response.

3.6.4 MIME Encapsulation

For resolution protocols that support MIME as a content description and packaging mechanism, the resulting assertion SHOULD be returned as a MIME entity of type "application/saml+xml", as defined by XX.

3.6.5 Use of HTTP URIs

A SAML processor authority that claims conformance to the SAML URI binding MUST implement support for HTTP. This section describes certain specifics of using HTTP URIs, including URI syntax, HTTP headers, error reporting, authentication, message integrity, and confidentiality.

3.6.5.1 URI Syntax

In general, there are no restrictions on the permissible syntax of a SAML URI reference as long as the SAML authority responsible for the reference creates the message containing it. However, authorities MUST support a URL endpoint at which an HTTP request can be sent with a single query string parameter named "AssertionID". There MUST be no query string in the endpoint URL itself independent of the parameter.

For example, if the documented endpoint at an authority is "https://saml.example.edu/assertions", a request for an assertion with AssertionID of abcd can be sent to "https://saml.example.edu/assertions?AssertionID=abcd"

Note that the use of wildcards is not allowed for such AssertionID queries.

3.6.5.2 HTTP Headers

HTTP proxies MUST NOT cache SAML assertions. HTTP headers SHOULD be used to protect against
such caching behavior.

HTTP header should have a cache control header indicating that HTTP proxies MUST NOT cache responses carrying SAML assertions.

Both of the following conditions apply when using HTTP 1.1:

1. If the value of the Cache-Control header field is not set to no-store, then the SAML responder MUST NOT include the Cache-Control header field in the response.
2. If the Expires response header field is not disabled by a Cache-Control header field with a value of no-store, then the Expires field SHOULD NOT be included.

There are no other restrictions on HTTP headers.

3.6.5.3 Authentication

The SAML requester and responder MUST implement the following authentication methods:

1. No client or server authentication.
2. HTTP basic client authentication [RFC2617] with and without SSL 3.0 or TLS 1.0.
3. HTTP over SSL 3.0 or TLS 1.0 (see Section 3.1.1) server authentication with a server-side certificate.
4. HTTP over SSL 3.0 or TLS 1.0 mutual authentication with both server-side and a client-side certificate.

If a SAML responder uses SSL 3.0 or TLS 1.0, it MUST use a server-side certificate.

3.6.5.4 Message Integrity

When message integrity needs to be guaranteed, SAML responders MUST implement HTTP over SSL 3.0 or TLS 1.0 (see Section 3.1.1) with a server-side certificate.

3.6.5.5 Message Confidentiality

When message confidentiality is required, SAML responders MUST implement HTTP over SSL 3.0 or TLS 1.0 (see Section 3.1.1) with a server-side certificate.

3.6.5.6 Security Considerations

[RFC2617] describes possible attacks in the HTTP environment when basic or message-digest authentication schemes are used.

3.6.5.7 Error Reporting

As an HTTP protocol exchange, the appropriate HTTP status code SHOULD be used to indicate the result of a request. For example, a SAML responder that refuses to perform a message exchange with the SAML requester SHOULD return a "403 Forbidden" response. If the assertion specified is unknown to the responder, then a "404 Not Found" response SHOULD be returned. In these cases, the content of the HTTP body is not significant.

3.6.5.8 Metadata Considerations

Support for the URI binding over HTTP SHOULD be reflected by indicating a URL endpoint at which requests for arbitrary assertions are to be sent.
3.6.5.9 Example SAML Message Exchange Using an HTTP URI

Following is an example of a request for an assertion.

GET /SamlService?AssertionID=abcde HTTP/1.1  
Host: www.example.com

Following is an example of the corresponding response, which supplies the requested assertion.

HTTP/1.1 200 OK  
Content-Type: application/saml+xml  
Content-Length: nnnn

<saml:Assertion AssertionID="abcde" ...>
...
</saml:Assertion>
4 SAML Artifact Formats

Todo: Move one or more formats into the definition of the artifact binding.

The general format of an artifact includes a mandatory two-byte artifact type code, as follows:

```
SAML_artifact ::= B64(TypeCode RemainingArtifact)
TypeCode ::= Byte1Byte2
```

Depending on the level of security desired and associated profile protocol steps, many viable architectures could be developed for the SAML artifact [CoreAssnEx] [ShibMarlena]. The type code structure accommodates variability in the architecture.

The notation B64(TypeCode RemainingArtifact) stands for the application of the base64 [RFC2045] transformation to the catenation of the TypeCode and RemainingArtifact.

4.1 SAML 1.x Format

4.1.1 Required Information

Contact information: security-services-comment@lists.oasis-open.org
Description: Given below.
Updates: None.

4.1.2 Format Details

This profile defines an artifact type of type code 0x0001. This artifact type is defined as follows:

```
TypeCode ::= 0x0001
RemainingArtifact ::= SourceID AssertionHandle
SourceID ::= 20-byte_sequence
AssertionHandle ::= 20-byte_sequence
```

SourceID is a 20-byte sequence used by the artifact receiver to determine artifact issuer identity and location. It is assumed that the destination site will maintain a table of SourceID values as well as the URL (or address) for the corresponding SAML responder, or alternatively metadata containing it. This information is communicated between the parties out-of-band. On receiving the SAML artifact, the receiver determines if the SourceID belongs to a known artifact issuer and obtains the location of the SAML responder before sending a SAML artifact request.

Any two artifact issuers with a common receiver MUST use distinct SourceID values. Construction of AssertionHandle values is governed by the principle that they SHOULD have no predictable relationship to the contents of the referenced assertion at the source site and it MUST be infeasible to construct or guess the value of a valid, outstanding assertion handle.

The following practices are RECOMMENDED for the creation of SAML artifacts:

- Each issuer selects a single identification URL. The domain name used within this URL is registered with an appropriate authority and administered by the issuer.
- The issuer constructs the SourceID component of the artifact by taking the SHA-1 hash of the identification URL.
- The AssertionHandle value is constructed from a cryptographically strong random or pseudorandom number sequence [RFC1750] generated by the issuer. The sequence consists of...
values of at least eight bytes in size. These values should be padded to a total length of 20 bytes.

4.2 SAML 1.x Supplemental Format

4.2.1 Required Information

Identification: urn:oasis:names:tc:SAML:1.0:profiles:artifact-02
Contact information: security-services-comment@lists.oasis-open.org
Description: Given below.
Updates: None.

4.2.2 Format Details

An alternative artifact format is described here:

<table>
<thead>
<tr>
<th>TypeCode</th>
<th>:= 0x0002</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemainingArtifact</td>
<td>:= AssertionHandle SourceLocation</td>
</tr>
<tr>
<td>AssertionHandle</td>
<td>:= 20-byte_sequence</td>
</tr>
<tr>
<td>SourceLocation</td>
<td>:= URI</td>
</tr>
</tbody>
</table>

The SourceLocation URI is the address of the SAML responder associated with the artifact issuer. The AssertionHandle is as described in Section 4.1.2, and governed by the same requirements. The SourceLocation URI is mapped to a sequence of bytes based on use of the UTF-8 [RFC2279] encoding. The receiver MUST process the artifact in a manner identical to that described in Section 4.1.2, with the exception that the location of the SAML responder of the issuer MAY be obtained directly from the artifact, rather than by look-up, based on SourceID.

Note: the receiver MUST confirm that the corresponding message is issued by an acceptable issuer, not relying merely on the fact that it was returned in response to a <samlp:ArtifactRequest> message.
5 URL Size Restriction (Non-Normative)

This section describes the URL size restrictions that have been documented for widely used commercial products.

A Microsoft technical support article [MSURL] provides the following information:

The information in this article applies to:

Microsoft Internet Explorer (Programming) versions 4.0, 4.01, 4.01 SP1, 4.01 SP2, 5, 5.01, 5.5

SUMMARY

Internet Explorer has a maximum uniform resource locator (URL) length of 2,083 characters, with a maximum path length of 2,048 characters. This limit applies to both POST and GET request URLs.

If you are using the GET method, you are limited to a maximum of 2,048 characters (minus the number of characters in the actual path, of course).

POST, however, is not limited by the size of the URL for submitting name/value pairs, because they are transferred in the header and not the URL.

RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1, does not specify any requirement for URL length.

REFERENCES

Further breakdown of the components can be found in the Wininet header file. Hypertext Transfer Protocol -- HTTP/1.1 General Syntax, section 3.2.1

Last Reviewed: 9/13/2001

Keywords: kbDSupport kbFAQ kbinfo KB208427

An article about Netscape Enterprise Server provides the following information:

Issue: 19971110-3 Product: Enterprise Server

Created: 11/10/1997 Version: 2.01

Last Updated: 08/10/1998 OS: AIX, Irix, Solaris

Does this article answer your question?

Please let us know!

Question:

How can I determine the maximum URL length that the Enterprise server will accept? Is this configurable and, if so, how?

Answer:

Any single line in the headers has a limit of 4096 chars; it is not configurable.
6 References


[XHTML] XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), http://www.w3.org/TR/xhtml1/.


[SAMLProfile] Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, DRAFT


open.org/committees/security.

[SESSION] RL "Bob" Morgan, Support of target web server sessions in Shibboleth,
http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-session-00.txt

[ShibMarlena] Marlena Erdoes, Shibboleth Architecture DRAFT v1.1,
http://shibboleth.internet2.edu/draft-internet2-shibboleth-arch-v05.html

Note, May 2000, http://www.w3.org/TR/SOAP.


[WEBSSO] RL "Bob" Morgan, Interactions between Shibboleth and local-site web sign-on services,
http://middleware.internet2.edu/shibboleth/docs/draft-morgan-shibboleth-websso-00.txt

[WSS-SAML] P. Hallam-Baker et al., Web Services Security: SAML Token Profile, OASIS, March 2003,
http://www.oasis-open.org/committees/wss.

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

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

• TBD
# B. Revision History

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>By Whom</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>02/16/04</td>
<td>Frederick Hirsch</td>
<td>Split Bindings and Profiles into two documents. Removed profiles from this document, added PAOS reverse SOAP binding</td>
</tr>
</tbody>
</table>
| 4   | 02/25/04   | Scott Cantor       | General language changes in introduction to reflect binding support of any SAML protocol  
Updated SOAP binding to reflect intended expansion of use across other protocols and SOAP-specific security mechanisms  
Removed confirmation methods, they don't belong in binding discussions. |
| 5   | 02/26/04   | Scott Cantor       | Revised arrangement of SSL discussion  
Added HTTP-Redirect/POST binding  
Added metadata considerations to SOAP binding |
| 6   | 03/01/04   | John Kemp          | Added 3 sections of URL-encoding. |
| 7   | 03/14/04   | Scott Cantor       | Added HTTP Artifact and URI bindings |
| 8   | 03/27/04   | Frederick Hirsch   | Updates for core 8, review comments and corrections. |
| 9   | 04/09/04   | Scott Cantor       | Tightened URL encoding/signing rules  
Added text around PAOS  
Incorporated feedback from FTF  
Placeholders for some additional work items. |
C. 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.

Copyright © OASIS Open 2003-2004. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to OASIS, except as needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

JavaScript is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.