Liberty Basic SOAP Binding

Version: **1.0**

Editors:

Søren Peter Nielsen, Danish National IT and Telecom Agency
Thomas Gundel, IT Crew

Contributors:

Conor P. Cahill, Intel
George Fletcher, AOL
Paul Madsen, NTT
Sampo Kellomaki, Symlabs
Pat Patterson, Sun Microsystems
Colin Wallis, New Zealand Government State Services Commission

Abstract:

This document contains a basic profile of the Liberty ID-WSF SOAP binding 2.0.

Filename: Liberty-Basic-SOAP-Binding-1.0_Final.pdf

This profile has been developed from business requirements within eGovernment, but is believed to be generally applicable. Liberty Alliance is making this profile publicly available to the industry at large for review and consideration.

Notice

Table of Contents

1 Introduction

Identity-based web services are expected to play an important role in enabling services that spans organisational borders since they allow IT systems to be connected in a secure, privacy-respecting and interoperable manner.

The present profile is intended to be a basic, scaled-down version of the Liberty ID-WSF 2.0 SOAP Binding Specification [LIB-SOAP] and Security Mechanisms 2.0 ([LIB-SEC] and [LIB-SAMLP]). The basic profile adopts mandatory elements from these specifications such that a Web Service Consumer implementing the profile should be able to invoke a Web Service Provider implementing the full Liberty SOAP binding (but not vice versa).

In order to keep the profile basic, self-contained[1] and easy to implement without knowledge on the other Liberty specifications, the profile is *not* a sub-profile of the other Liberty specifications. Instead, this document profiles the WS-Addressing SOAP Binding [WSAv1.0-SOAP] and WS-Security [WSS] directly. Thus, mandatory elements and processing rules from the Liberty SOAP binding are duplicated here and the profile can thus be read and implemented independently. Other, non-Liberty specifications including SOAP, WS-Security and WS-Addressing are referenced and not embedded here in order to keep the profile light-weight. It is believed that many application developers will not have to implement these specifications from scratch because they are supported in their development tools, messaging middleware and application servers.

1.1 Context

The following is an example of a usage scenario supported by the profile and which was used to gather requirements:

1. A browser user logs in at a Service Provider using normal SAML web SSO profiles.
2. The Service Provider needs to invoke a remote identity-based web service at a Web Service Provider (WSP) on the user's behalf.
3. The Service Provider exchanges the user's SAML SSO assertion (or embedded bootstrap token) for an authentication assertion (also called an identity token[2]) targeted at the WSP, e.g. by contacting a Security Token Service (STS) or Discovery Service.
4. The Service Provider (aka Web Service Consumer) invokes the Web Service Provider using the SOAP binding described in this profile. The request includes the authentication assertion in security headers and is signed by the sender.
5. The Web Service Provider processes the request and responds synchronously.

[1] The profile still relies on the WS-* specifications such as WS-Addressing and WS-Security.
[2] To be exact this profile uses the Liberty term "Authentication assertion" instead of "Identity token" as this term is not defined in a Liberty context..

1.2 Assumptions

The profile builds on the following assumptions:

- A Web Service Consumer (WSC) needs to invoke a Web Service Provider (WSP) on behalf of a user / principal by sending a message and receiving synchronously a response conforming to this profile.
- The WSC has already access to the WSP's meta data needed for the invocation (end points, service interface etc.).
- Both WSC and WSP possess a means of creating signatures that can be verified by each other; thus they can establish mutual trust in each other's signing key.
- The WSC has obtained an authentication assertion in the form of an SAML 2.0 assertion which describes the identity of the user whose identity-based web service is being invoked (invoking identity). The authentication assertion can be obtained by several means including a Liberty Discovery Service or a STS implementing the WS-Trust specification.
- The WSP is able to validate the authentication assertion.

These assumptions (along with the excluded features listed below) are the basis for the formulation of a simplified profile.

1.3 Excluded Features

The following features from [LIB-SOAP] have been excluded in order to formulate a simpler profile:

- Endpoint update
- Processing context header
- Asynchronous messages
- Security tokens other than SAML 2.0 assertions
- Message authentication and -integrity established by other means that signing the request
- User interaction
- Usage directives
- One user invoking a service on behalf of another user

2 SOAP Binding

2.1 SOAP Version

This profile depends upon SOAP version 1.1 as specified in [SOAPv1.1]. Messages conformant to this specification MUST also be conformant to [SOAPv1.1].

2.2 The SOAPAction HTTP Header

[SOAPv1.1] defines the SOAPAction HTTP header, and requires its usage on HTTP-bound messages.

The value of the SOAPAction HTTP header SHOULD be the same as the value of the `<wsa:Action>` header block defined in the next chapter.

2.3 SOAP Fault Messages

When reporting a SOAP processing error such as "`S:VersionMismatch`" or "`S:MustUnderstand`", the `<S:Fault>` element SHOULD be constructed according to [SOAPv1.1].

When reporting a WS-Addressing processing error such as "`wsa:InvalidAddress`", the `<S:Fault>` element SHOULD be constructed according to [WSAv1.0-SOAP].

For all other processing errors the `<S:Fault>` element's attributes and child elements MUST be constructed according to these rules:

1. The `<S:Fault>` element:
    a. SHOULD contain a `<faultcode>` element whose value SHOULD be one of "`sbf:FrameworkVersionMismatch`", "`S:server`" or "`S:client`".
    b. SHOULD contain a `<faultstring>` element. This string value MAY be localized.
    c. SHOULD NOT contain a `<S:faultactor>` element.
2. The `<S:Fault>` element's `<detail>` child element SHOULD contain a `<Status>` element which:
    a. MUST contain a code attribute.
    b. MAY contain a `ref` attribute.
    c. MAY contain a `comment` attribute. This string value MAY be localized.

3 Messaging-specific Header Blocks

This section profiles the use of WS-Addressing SOAP Binding [WSAv1.0-SOAP] and WS-Security [WSS] header blocks, and incorporates the framework header from the Liberty SOAP Binding [LIB-SOAP].

Along with header block descriptions are included processing rules the sender must apply when including it in an outgoing message or when processing it is part of an incoming message.

When sending a response to a request, the same header blocks and processing rules apply unless stated otherwise below. The main difference is that response messages do not include authentication assertions representing a user.

3.1 Overview of Header Blocks

The following header blocks MUST be included in the SOAP header:

- `<wsa:MessageID>`
- `<wsa:RelatesTo>` (mandatory on response)
- `<wsa:Action>`
- `<wsse:Security>`
- `<sbf:Framework>`

The following headers MAY be included in the SOAP header:
`<wsa:To>`

If included, the recipient SHOULD be able to process them according to the requirements described below.

3.2 The `<wsa:MessageID>` Header Block

The `<wsa:MessageID>` header block is defined in [WSAv1.0-SOAP]. The value of this header block uniquely identifies the message that contains it.

Every message MUST contain exactly one such header block.

3.2.1 `<wsa:MessageID>` Value Requirements

Values of the `<wsa:MessageID>` header block MUST satisfy the following property:

Any party that assigns a value to a `<wsa:MessageID>` header block MUST ensure that there is negligible probability that the party or any other party will accidentally assign the same identifier to any other message.

The mechanism by which senders or receivers ensure that an identifier is unique is left to implementations. In the case that a pseudorandom technique is employed, the above requirement MAY be met by randomly choosing a value 160 bits in length.

Note that [WSAv1.0] requires that `<wsa:MessageID>` values be absolute IRIs.

3.3 The `<wsa:RelatesTo>` Header Block

The `<wsa:RelatesTo>` header block is defined in [WSAv1.0-SOAP].

The header block MUST be included exactly once in responses to prior-received request messages. If the `RelationshipType` attribute is included it MUST be set to the value `http://www.w3.org/2005/03/addressing/reply`.

In response messages, the value of this header block MUST be set to the value of the `<wsa:MessageID>` header block of the prior-received message.

3.4 The `<wsa:Action>` Header Block

The `<wsa:Action>` header block is defined in [WSAv1.0-SOAP]. The value of this header block uniquely identifies the semantics implied by the message.

The header block MUST be included exactly once in all messages.

Note
The value of this header block SHOULD contain the same value as the `SOAPAction` HTTP header defined in [SOAPv1.1]. The SOAP specification requires the HTTP header on all HTTP-bound SOAP messages.

3.5 The `<sbf:Framework>` Header Block

The `<sbf:Framework>` header block is defined in the [LIB-SOAP] specification and provides the sender with a means to communicate the version of the ID-WSF framework used to construct the message. In order to make messages produced using this profile compatible with the full Liberty SOAP binding, the Liberty framework header is used in this profile as well.

The header block MUST be included exactly once in every message.

Further:
The `version` attribute SHOULD be set to “2.0”

A `profile` attribute with the name space "`urn:liberty:sb:profile`" MUST be included with the value of "`urn:liberty:sb:profile:basic`".

Example:

```
<sbf:Framework
    xmlns:sbfprofile="urn:liberty:sb:profile"
    …
    version="2.0"
    sbfprofile:profile="urn:liberty:sb:profile:basic"
    s:mustUnderstand="1"
    s:actor="http://schemas.../next"
    wsu:Id="SBF"/>
```

If the receiver of a message does not recognize the `version` and `profile` attributes, it MAY respond to the sender with a SOAP fault message with the `<faultcode>` of `sbf:FrameworkVersionMismatch`.

3.6 The `<wsa:To>` Header Block

The `<wsa:To>` header block is defined in [WSAv1.0-SOAP]. The value of this header block specifies the intended destination of the message.

Note
In the typical case that a WS-Addressing endpoint reference is used to address a message, the value of this header block is taken from the `<wsa:Address>` of the endpoint reference. If the `<wsa:To>` header block is not present, the value defaults to `http://www.w3.org/2005/03/addressing/role/anonymous`; so, when constructing a message, the header block can be omitted if this is the value that would be used. This typically allows the `<wsa:To>` header block to be omitted in responses during synchronous request-response message exchanges over HTTP.

The header block is optional.

3.7 The `<wsse:Security>` Header Block

This section defines elements and processing rules for SOAP message security by profiling the `<wsse:Security>` header block defined in [WSS]. Processing rules defined in [WSS] and [WSS-STP] MUST be followed unless stated explicitly otherwise below.

A single `<wsse:Security>` header block MUST be present and MUST have a `mustUnderstand` attribute with the logical value of `true`. Further, it MUST include a `<wsu:Timestamp>` with a `<wsu:Created>` element.

The value of the `<wsu:Created>` element SHOULD be within an appropriate offset from local time. Absent other guidance, a value of 5 minutes MAY be used.

If the `<wsu:Timestamp>` element includes an `<wsu:Expires>` element, the receiver MUST ensure that his local time is before that time.

To prevent message replay, receivers SHOULD maintain a message cache, and check received `messageID` values against the cache. How long time a message should be kept in the cache at the WSP is governed by deployment policy.

3.7.1 Message Authentication and Integrity

Authentication and integrity of messages is established by means of digital signatures applied to the SOAP message. Confidentiality, if required, MUST be established by using a secure transport protocol (e.g. using SSL 3.0 or TLS 1.1 or later).

The sender MUST create and include a single `<ds:Signature>` element in the `<wsse:Security>` header block and this signature MUST reference:

- The SOAP `<Body>` element
- All security tokens embedded directly under the `<wsse:Security>` element via a `<wsse:SecurityTokenReference>` (see below), and
- All SOAP header blocks in the message defined in this profile. The signature MAY reference other elements including header blocks not mentioned in this profile.

If the sender has obtained a SAML holder-of-key Assertion vouching for the signing key (see next section) it SHOULD be included in the security header. Detailed requirements for using holder-of-key assertions are given below.

If the sender does not possess a holder-of-key Assertion but instead has an X.509 certificate, the certificate SHOULD be included in a `<wsse:BinarySecurityToken>` element in the security header. In the message signature, the `<ds:KeyInfo>` element SHOULD refer to this token via a `<wsse:SecurityTokenReference>`.

The receiver MUST validate the message signature and security tokens including test of validity period and trust in the token issuer. Depending on local policy, the receiver SHOULD check revocation status of any certificates used to sign the message and tokens.

3.7.2 Establishing trust in message signature key

The receiver can establish trust in the sender’s signature key in the following ways:

- The security header contains a SAML 2.0 holder-of-key assertion issued by someone[3] the receiver trusts, and the holder-of-key assertion includes a key that can be used to verify the message signature. Note that the assertion itself will be signed by the trusted issuer so the receiver has to be able to verify the issuer's signature. The sender's signing key MAY be symmetric or asymmetric.
- The message is signed with a key the receiver already knows / trusts for example due to prior metadata exchange.
- The security header includes an X.509 certificate in a BinarySecurityToken issued by a Certificate Authority the receiver trusts, and the certificate can be used to verify the message signature.

3.7.3 Authentication Assertions

In request messages, the `<wsse:Security>` header block MAY include authentication assertions in the form of SAML 2.0 assertions representing the identity of the user / principal whose identity-based web service is being invoked. Other types of security tokens (except for BinarySecurityTokens containing certificates) SHOULD not be used and implementations of this profile are not required to implement them.

The authentication assertion MUST be a SAML 2.0 assertion with subject confirmation method being either `urn:oasis:names:tc:SAML:2.0:cm:bearer` or `urn:oasis:names:tc:SAML:2.0:cm:holder-of-key`.

Authentication assertions MUST be signed by the issuer (e.g. Identity Provider, STS or Discovery Service). Requirements for the content of authentication assertions are not specified further in this profile.

Authentication assertions MUST be signed by the sender by including first a `<wsse:SecurityTokenReference>` in `<wsse:Security>` header block, and then referencing the STR from the message signature using a `<ds:Reference>` element. The security token reference MUST include a `<wsse:KeyIdentifier>` with a `ValueType` of `http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID` and specify the ID of the SAML assertion. The `<ds:Reference>` element MUST use a transform algorithm set to "`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsssoap-message-security-1.0#STR-Transform`".

The receiver MUST validate SAML 2.0 authentication assertions according to the processing rules defined in [SAML-CORE] and [WSS-STP] including life time of the token, audience restriction, the issuer's signature over the token, trust in the issuer and other processing rules defined by token profiles.

[3] For example a Liberty Discovery Service or a Security Token Service.

3.7.4 Additional Processing Rules for `holder-of-key` Assertions

When the authentication assertion has a subject confirmation method being "`holder-of-key`" it means that the sender must prove possession of a key mentioned in the assertion's `<SubjectConfirmationData>` in order for the recipient to rely on the assertion. The proof-of-possession of the key will be achieved via the message signature and provides additional assurance that the sender is allowed to use to the assertion in a web service invocation.

In this profile, a holder-of-key Assertion MUST in the `<SubjectConfirmationData>` element include a key that can be used to verify the message signature. Thus, the *same* key used for message authentication and integrity is used to confirm the right to use the assertion for message authorization purposes.

The message signature (i.e. the `<ds:Signature>` element) MUST refer to the token with the subject confirmation key within the `<ds:KeyInfo>` element.

The receiver MUST check that the message is signed by same key mentioned in the assertion's subject confirmation element before relying on the assertion content.

4 Overall Processing Rules

Overall processing of SOAP-bound messages follows the rules of the SOAP processing model described in [SOAPv1.1]. A number of additional rules are defined below. Notice that processing rules for individual elements are found in the previous section.

4.1 Constructing and sending a SOAP message

The sender MUST follow these processing rules when constructing and sending an outgoing SOAP message:

1. The outgoing message MUST satisfy the rules for SOAP binding defined in section “SOAP Binding”.
2. The outgoing message MUST satisfy the rules for WS-Addressing SOAP binding given in [WSAv1.0-SOAP].
3. The outgoing message MUST include the mandatory header blocks defined above.
4. All other Liberty headers defined in [LIB-SOAP] SHOULD NOT be used with this profile since implementations of the profile are not required to support them.
5. Each header block included in the outgoing message MUST conform to the processing rules defined for each header block.

Below is shown a procedure that illustrates how a conforming message can be constructed (some low-level details have been omitted). It is assumed that the sender has obtained all the information required to construct the message including security tokens, signing keys and message payload. The procedure is not normative and conforming messages can be constructed in other ways:

1. Construct the XML payload to be included in the SOAP Body.
2. Construct a SOAP envelope with `<Header>` and `<Body>`, and embed the payload in the `<Body>`. Add a `wsu:Id` attribute[4] to the `<Body>` element.
3. Add a `<wsa:MessageID>` header block (including a `wsu:Id` attribute) which uniquely identifies the message; for example generate a 160-bit pseudorandom number and embed it in a URI as follows:

   ```
   http://spwsp.com/ffeeddccbbaa99887766 554433221100ffeebbcc
   ```

4. When generating a response, include a `<wsa:RelatesTo>` element (including a `wsu:Id` attribute) containing the message ID of the request.
5. Add a `<wsa:Action>` header block (including a `wsu:Id` attribute) corresponding to the `SOAPAction` HTTP header as required by the service being invoked.

[4] In the following, all `wsu:Id` attributes should contain a value that is unique within the SOAP message.

6. If required, add a `<wsa:To>` header block (including a `wsu:Id` attribute) to identify the recipient.
7. Add the `<sbf:Framework>` header block as defined previously (including a `wsu:Id` attribute).
8. Add a `<wsse:Security>` header block with a `mustUnderstand=1` attribute.
    a. Add a `<wsu:Timestamp>` element (including a `wsu:Id` attribute) with a `<wsu:Created>` sub-element that includes the local time.
    b. Include any security tokens (SAML Assertions and/or BinarySecurityTokens containing X.509 certificates) in the security header block. Ensure that they have unique id attributes so they can be referenced (e.g. `saml2:ID` or `wsu:Id`).
    c. Create a `<wsse:SecurityTokenReference>` element (including a `wsu:Id` attribute) for each embedded SAML assertion. Add a `TokenType` attribute stating the type of token (`http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0`) and a `<wsse:KeyIdentifier>` sub-element containing the ID of the assertion.
    d. Create a `<ds:Signature>` element in the security header:
        i. Add a `<ds:SignedInfo>` element and embed `<ds:Reference>` sub-elements with references to each of the above header blocks and the SOAP Body. For each reference, include element ID, digest method and digest value. Set the Transform Algorithm to `http://www.w3.org/2001/10/xml-exc-c14n#`
        ii. Include a `<ds:Reference>` elements for each assertion reference produced in step c) by using the ID of the `<SecurityTokenReference>` element. Set the Transform Algorithm set to `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsssoap-message-security-1.0#STR-Transform`
    e. Add a `<ds:KeyInfo>` element with a `<wsse:SecurityTokenReference>` pointing to either a SAML assertion or BinarySecurityToken vouching for the signature key. The reference should include a `<wsse:KeyIdentifier>` containing the ID of the token.
    f. Compute the `<ds:SignatureValue>` over the `<ds:SignedInfo>` using the signature key.
9. Send the message over a secure transport (SSL or TLS).

Below is shown an example SOAP message that is compliant with the Liberty Basic SOAP binding:

```
<?xml version="1.0" encoding="UTF-8"?>
<s:Envelope
   xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:sbf="urn:liberty:sb"
   xmlns:sbfprofile="urn:liberty:sb:profile"
   xmlns:sec="urn:liberty:security:2006-08"
   xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
       1.0.xsd"
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
       1.0.xsd"
   xmlns:wsa="http://www.w3.org/2005/08/addressing"
   xmlns:idpp="urn:liberty:id-sis-pp:2003-08">

   <s:Header>
      <wsa:MessageID wsu:Id="mid">f63d289c-cd9a-4c00-bf87-c4bad0310646</wsa:MessageID>

      <wsa:To wsu:Id="to">...</wsa:To>

      <wsa:Action wsu:Id="action">urn:liberty:id-sis-pp:2003-08:Modify</wsa:Action>


      <sbf:Framework
        version="2.0"
        sbfprofile:profile="urn:liberty:sb:profile:basic"
        s:mustUnderstand="1"
        s:actor="http://schemas.../next"
        wsu:Id="framework"/>


      <wsse:Security mustUnderstand="1">
         <wsu:Timestamp wsu:Id="ts">
            <wsu:Created>2008-08-17T04:49:17Z</wsu:Created >
         </wsu:Timestamp>

         <!-- this is the holder-of-key token with the sender's certificate -->
         <saml2:Assertion
            xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
            Version="2.0"
            ID="sxJu9g/vvLG9sAN9bKp/8q0NKU="
            IssueInstant="2008-08-01T16:58:33Z">
            <saml2:Issuer>http://authority.example.com/</Saml2:Issuer>

            <!-- signature by the issuer over the assertion -->
            <ds:Signature>
                      <ds:SignedInfo>
                             <ds:CanonicalizationMethod
                                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                             <ds:SignatureMethod
                                Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                             <ds:Reference URI="#sxJu9g/vvLG9sAN9bKp/8q0NKU=">
                                    <ds:Transforms>
                                           <ds:Transform
                      Algorithm="http://www.w3.org/2000/09/xmldsig#envelopedsignature"/>
                                    </ds:Transforms>
                                    <ds:DigestMethod
                                    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

                             <ds:DigestValue>TCDVSuG6grhyHbzhQFWFzGrxIPE=</ds:DigestValue>
                            </ds:Reference>
                      </ds:SignedInfo>
                      <ds:SignatureValue>
                            x/GyPbzmFEe85pGD3c1aXG4Vspb9V9jGCjwcRCKrtwPS6vdVNCcY5rHaFPYWkf+5
                            EIYcPzx+pX1h43SmwviCqXRjRtMANWbHLhWAptaK1ywS7gFgsD01qjyen3CP+m3D
                            w6vKhaqledl0BYyrIzb4KkHO4ahNyBVXbJwqv5pUaE4=
                      </ds:SignatureValue>
                      <ds:KeyInfo>
                             <ds:X509Data>
                     <!-- data identifying the signer's certificate -->
                             </ds:X509Data>
                       </ds:KeyInfo>
            </ds:Signature>


            <saml2:Subject>
               <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
                  005a06e0-ad82-110d-a556-004005b13a2b
               </saml:NameID>

               <!-- Here comes the subject confirmation method saying this is a holder-of-
key -->
               <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:holder-of-
key">

                  <!-- Here comes a NameID indicating the ID of the sender who must confirm
with a key -->
                  <saml2:NameID format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
                     http://wsc.someorg.com
                  </saml2:NameID>

                  <!-- Here comes info on the key to confirm with (same as signing key> -->
                  <saml2:SubjectConfirmationData
xsi:type="saml2:KeyInfoConfirmationDataType>
                     <ds:KeyInfo>
                       <ds:X509Data>
                          <!-- Here comes the sender's X509 cert -->
                          MIIB9zCCAWSgAwIBAgIQ...
                       </ds:X509Data>
                     </ds:KeyInfo>
                  </saml2:SubjectConfirmationData>

               </saml2:SubjectConfirmation>
            </saml2:Subject>

             <!-- Entity which should consume the information in the assertion. -->
            <saml2:Conditions
               NotOnOrAfter="2008-08-01T21:42:43Z">
               <saml2:AudienceRestrictionCondition>
                  <saml2:Audience>http://wsp.example.com</saml2:Audience>
               </saml2:AudienceRestrictionCondition>
            </saml2:Conditions>

            <saml2:AttributeStatement>
               ...
            </saml2:AttributeStatement>
         </saml2:Assertion>

         <!-- This SecurityTokenReference is used to reference the SAML Assertion from a 
ds:Reference -->
         <wsse:SecurityTokenReference 
            xmlns:wsse="..." xmlns:wsu="..." xmlns:wsse11="..."
            wsu:Id="str1"
            wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0">
            <!-- A key idenfier with the SAML Assertion ID -->
            <wsse:KeyIdentifier 
               ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLID">
                  sxJu9g/vvLG9sAN9bKp/8q0NKU=
            </wsse:KeyIdentifier>
         </wsse:SecurityTokenReference>


         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
               <!-- in general include a ds:Reference for each wsa: header added according 
to SOAP binding -->

               <!-- include the MessageID in the signature -->
               <ds:Reference URI="#mid">...</ds:Reference>

               <!-- include the To in the signature -->
               <ds:Reference URI="#to">...</ds:Reference>

               <!-- include the Action in the signature -->
               <ds:Reference URI="#action">...</ds:Reference>

               <!-- include the Framework in the signature -->
               <ds:Reference URI="#framework">...</ds:Reference>

               <!-- include the Timestamp in the signature -->
               <ds:Reference URI="#ts">...</ds:Reference>

               <!-- include the SAML Assertion in the signature to avoid token substitution 
attacks -->
               <ds:Reference URI="#str1">
                  <ds:Transform Algorithm="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wsssoap-message-security-1.0#STR-Transform">
                     <wsse:TransformationParameters>
                        <ds:CanonicalizationMethod 
                           Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                        </wsse:TransformationParameters>
                  </ds:Transform>
               </ds:Reference>

               <!-- bind the body of the message -->
               <ds:Reference URI="#MsgBody">
                  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <ds:DigestValue>YgGfS0pi56pu...</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>

            <!-- include a security token reference for holder-of-key confirmation -->
             <ds:KeyInfo>
               <wsse:SecurityTokenReference
                  xmlns:wsse="..." xmlns:wsu="..." xmlns:wsse11="..."
                  wsu:Id="str2"
                  wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-
profile-1.1#SAMLV2.0">
                  <!-- A key idenfier with the SAML Assertion ID -->
                  <wsse:KeyIdentifier
                     ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLID">
                     sxJu9g/vvLG9sAN9bKp/8q0NKU=
                  </wsse:KeyIdentifier>
               </wsse:SecurityTokenReference>
            </ds:KeyInfo>

            <ds:SignatureValue>
                 HJJWbvqW9E84vJVQkjjLLA6nNvBX7mY00TZhwBdFNDElgscSXZ5Ekw==
            </ds:SignatureValue>
         </ds:Signature>
      </wsse:Security>
   </s:Header>

   <s:Body wsu:Id="MsgBody">
      <idpp:Modify>
         :   <!-- this is an ID-PP Modify message -->
      </idpp:Modify>   </s:Body>
</s:Envelope>
```

4.2 Receiving and processing a SOAP message

The receiver of a SOAP message (either normal message or fault) MUST perform the following tests on the header blocks:

Note: Although the steps are numbered sequentially, implementations MAY use a different sequence as long as all tests are applied.

1. The incoming message MUST satisfy the rules for SOAP binding defined in section “SOAP Binding”.
2. The incoming message MUST satisfy the rules given in [WSAv1.0-SOAP].
3. The incoming message MUST include all mandatory header blocks defined above.
4. Each header block in the message (mandatory as well as optional) MUST be tested according to the processing rules defined above.

Below is shown a procedure illustrating how messages can be verified and processed (some details e.g. regarding signature processing have been omitted; for details see the XML digital signature standard). It is assumed that the receiver has all the information required to process the message including certificates of trusted parties issuing tokens. The procedure is not normative and messages may be processed / validated in other ways; implementations may for example perform the steps in other sequence for efficiency reasons.

1. Receive the SOAP message over a secure transport protocol (SSL or TLS).
2. Validate that the following mandatory SOAP headers are present and contain appropriate values: `<wsa:MessageID>` should include a unique value, `<sbf:Framework>` should specify a framework version and profile understood by the recipient and `<wsa:Action>` should be consistent with the invoked service.
3. If present, check that the content of the `<wsa:To>` header corresponds to the recipient / endpoint.
4. Check the received message ID value against the local cache to determine whether it has been received before (replay attacks). If not, add message ID to cache to detect future replays.
5. Check that exactly one `<wsse:Security>` header is present:
    a. Verify that the `<wsu:Timestamp>` is within acceptable limits of local server time as defined by deployment policy.
    b. Validate all embedded security tokens including that they are signed by a trusted issuer, timestamps, audience restrictions etc. (token validation rules vary with token type). Any proof-of-possession requirements are handled below.
    c. Check that the message signature (`<ds:Signature>)` contains references to all header block defined above, to the SOAP body and all included SAML assertions (via a `SecurityTokenReference`). Verify that all digest values match the referenced elements.
    d. Verify the message signature using the key referenced in the `<ds:KeyInfo>` element.
    e. Check that the signing key is vouched-for via a security token issued by a trusted party.
    f. Verify that proof-of-possession requirements in tokens (e.g. SAML holder-of-key SubjectConfirmation) are demonstrated via the message signing key. Thus, the proof-of-possession key in tokens must match the key that signed the message.
    g. Check that all claims required by the service have been demonstrated by the attached security tokens.
6. Discard message payload if any of the above checks fail and send a meaningful error message to the recipient.
7. Handle message payload and send response over secure transport.

Note that the recipient may need to perform additional checks e.g. related to authorization.

5 Security Considerations

Message integrity and authenticity is established by mandatory signing (and subsequent verification) of the SOAP body, header blocks in this specification and security tokens.

Message confidentiality is not addressed directly in this profile but may be established by using a secure transport protocol such as SSL 3.0, TLS 1.1 or later HTTPS, or by encryption of name identifiers or individual attributes in the SAML 2.0 assertion.

Message freshness and prevention against replay attacks is established by including unique message Ids that WSP's should cache, time stamps and expiry of tokens. How long time a message should be kept in the cache at the WSP is governed by deployment policy.

Message authorization is established by including signed authentication assertions in the form of SAML assertions issued by a trusted STS, Liberty Discovery Service or Identity Provider.

Security tokens in the form of SAML 2.0 assertions are signed by the issuer and sensitive attributes may be encrypted if deemed necessary via the mechanisms described in [SAML-CORE] including encryption of the entire assertion, name identifiers and individual attributes.

It is outside the scope of this profile to define how a Web Service Provider performs local authorization decisions but the WSP may take the following request parameters into consideration:

- The sender identity as established via the signature.
- The invoker / user identity as established via authentication assertions.
- The resource / service being accessed.
- Trust in the STS, Discovery Service or Identity Provider that has issued the authentication assertion.
- The assurance level established as part of the assertion.

6 References

[SOAPv1.1] “Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske et. al. World Wide Web Consortium W3C Note (08 May 2000). http://www.w3.org/TR/2000/NOTE-SOAP-20000508/”.

[WSS] “Web Services Security: SOAP Message Security 1.1”, OASIS Standard, 1 February 2006.

[WSS-STP] “Web Services Security: SAML Token Profile 1.1”, OASIS Standard, 1 February 2006. http://docs.oasis-open.org/wss/oasis-wss-SAMLTokenProfile-1.1

[SAML-CORE] “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0”, OASIS Standard, 15 March 2005.

[LIB-SOAP] “Liberty ID-WSF SOAP Binding Specification”, version 2.0, Liberty Alliance Project

[LIB-SAMLP] “ID-WSF 2.0 SecMech SAML Profile”, version 2.0, Liberty Alliance Project.

[LIB-SEC] “Liberty ID-WSF Security Mechanisms Core”, version 2.0, Liberty Alliance Project.

[WSS-SAML] “Web Services Security: SAML Token Profile 1.1”, OASIS Standard, 1 February 2006.

[Scenarios] “Identity-Based Web Services – Scenarios”, Danish IT and Telecom Agency. (Not yet published on the WWW)

[WSAv1.0-SOAP] “WS-Addressing 1.0 SOAP Binding”, World Wide Web Consortium W3C Recommendation (9 May 2006).