Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification

Version: 2.0-errata-v1.0

Editors:
Jeff Hodges, NeuStar, Inc.
Robert Aarts, Hewlett-Packard
Paul Madsen, NTT
Scott Cantor, Internet2 / The Ohio State University

Contributors:
Conor Cahill, America Online, Inc.
Darryl Champagne, IEEE-ISTO
Gary Ellison, Sun Microsystems, Inc.
Rob Lockhart, IEEE-ISTO
Greg Whitehead, Hewlett-Packard

Abstract:

Abstract

This specification defines an ID-WSF Authentication Protocol based on a profile of the Simple Authentication and Security Layer (SASL) framework mapped onto ID-* SOAP-bound messages. It also defines an ID-WSF Authentication Service which Identity Providers may offer. This service is based on the authentication protocol. The authentication service enables Web Services Consumers and/or Liberty-enabled User Agents or Devices to authenticate with Identity Providers, using various authentication mechanisms, and obtain ID-WSF security tokens. Next, it defines the ID-WSF Single Sign-On Service, which provides SAML authentication assertions to Web Service Consumers via profiles of the SAML 2.0 Authentication Request protocol, enabling Web Service Consumers and/or Liberty-enabled User Agents or Devices to interact with SAML-based services. Finally, it defines the ID-WSF Identity Mapping Service, which allows Web Service Consumers to obtain identity tokens for use in web service invocations and referencing principals while preserving privacy.

Filename: liberty-idwsf-authn-svc-2.0-errata-v1.0.pdf

Notice

Implementation of certain elements of this document may require licenses under third party intellectual property rights, including without limitation, patent rights. The Sponsors of and any other contributors to the Specification are not and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights. **This Specification is provided "AS IS", and no participant in the Liberty Alliance makes any warranty of any kind, express or implied, including any implied warranties of merchantability, non-infringement of third party intellectual property rights, and fitness for a particular purpose.** Implementers of this Specification are advised to review the Liberty Alliance Project's website (http://www.projectliberty.org/) for information concerning any Necessary Claims Disclosure Notices that have been received by the Liberty Alliance Management Board.

Liberty Alliance Project
Licensing Administrator
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08855-1331, USA
info@projectliberty.org

Contents

1. Introduction . . . 5
2. Notation and Conventions . . . 6
    2.1. Requirements Keywords . . . 6
    2.2. XML Namespaces . . . 6
3. Terminology . . . 7
4. Authentication Protocol . . . 10
    4.1. Conceptual Model . . . 10
    4.2. Schema Declarations . . . 10
    4.3. SOAP Header Blocks and SOAP Binding . . . 10
        4.3.1. SOAP Binding . . . 10
    4.4. SASL Profile Particulars . . . 10
        4.4.1. SASL "Service Name" . . . 11
        4.4.2. Composition of SASL Mechanism Names . . . 11
    4.5. Authentication Exchange Security . . . 11
    4.6. Protocol Messages . . . 11
        4.6.1. The `<SASLRequest>` Message . . . 11
        4.6.2. The `<SASLResponse>` Message . . . 14
    4.7. Sequencing of the Authentication Exchange . . . 17
5. Authentication Service . . . 21
    5.1. Conceptual Model . . . 21
        5.1.1. Stipulating a Particular Authentication Context . . . 21
    5.2. URI Declarations . . . 21
    5.3. Rules for Authentication Service Providers . . . 22
    5.4. Rules for Authentication Service Consumers . . . 23
    5.5. Authentication Service Interaction Example . . . 23
6. Single Sign-On Service . . . 26
    6.1. Conceptual Model . . . 26
    6.2. Single Sign-On Service URIs . . . 26
    6.3. ID-WSF Enhanced Client or Proxy SSO Profile . . . 27
        6.3.1. Profile Overview . . . 27
        6.3.2. Profile Description . . . 27
    6.4. ID-WSF SAML Token Service Profile . . . 28
        6.4.1. Profile Overview . . . 28
        6.4.2. Profile Description . . . 29
        6.4.3. Use of SAML 2.0 Authentication Request Protocol . . . 29
    6.5. Use of Metadata . . . 31
    6.6. Inclusion of ID-WSF Endpoint References . . . 31
7. Identity Mapping Service . . . 32
    7.1. Conceptual Model . . . 32
    7.2. Schema Declarations . . . 32
    7.3. SOAP Binding . . . 32
        7.3.1. Identity Mapping Service URIs . . . 32
    7.4. Protocol Messages and Usage . . . 33
        7.4.1. Element <IdentityMappingRequest> . . . 33
        7.4.2. Element <IdentityMappingResponse> . . . 34
    7.5. SAML Identity Tokens . . . 36
        7.5.1. Assertions . . . 36
        7.5.2. Identifiers . . . 36
    7.6. Security and Privacy Considerations . . . 37
    7.7. Example Identity Mapping Exchange . . . 37
8. Password Transformations: The `PasswordTransforms` Element . . . 39
9. Acknowledgments . . . 41

References 42
A. Listing of Simple Authentication and Security Layer (SASL) Mechanisms 45
B. Password Transformations 47
    1. Truncation 47
    2. Lowercase 47
    3. Uppercase 47
    4. Select 47
C. liberty-idwsf-authn-svc-v2.0.xsd Schema Listing 49
D. liberty-idwsf-idmapping-svc-v2.0.xsd Schema Listing 52
E. liberty-idwsf-utility-v2.0.xsd Schema Listing 53
F. liberty-idwsf-authn-svc-v2.0.wsdl WSDL Listing 55
G. liberty-idwsf-sso-svc-v2.0.wsdl WSDL Listing 56
H. liberty-idwsf-idmapping-svc-v2.0.wsdl WSDL Listing 57

1. Introduction

The Simple Object Access Protocol (SOAP) specifications, [SOAPv1.1] and [SOAPv1.2], define an XML-based [XML] messaging paradigm, but do not specify any particular security mechanisms. They do not, in particular, describe how one *SOAP node* may authenticate with another *SOAP node* via an exchange of SOAP messages. Thus it is left to SOAP-based web services frameworks to provide their own notions of security, such as defining how authentication is accomplished.

This specification defines how to perform *general identity authentication* [WooLam92], also known as *peer entity authentication* [RFC2828], over SOAP, in the context of the Liberty Identity Web Services Framework (ID-WSF) [LibertyIDWSFOverview]. Rather than specify the particulars of one or more *authentication mechanisms* directly in this specification, we profile the Simple Authentication and Security Layer (SASL) framework [RFC4422].

SASL is an approach to modularizing protocol *design* such that the security design components, e.g., authentication and security layer mechanisms, are reduced to a uniform abstract interface. This facilitates a protocol's use of an open-ended set of security mechanisms, as well as a so-called "late binding" between implementations of the protocol and the security mechanisms' implementations. This late binding can occur at implementation- and/or deployment-time. The SASL specification also defines how one packages authentication and security layer mechanisms to fit into the SASL framework, where they are known as *SASL mechanisms*, as well as register them with the Internet Assigned Numbers Authority (IANA) [IANA] for reuse.

This specification is organized as follows. First, it defines the ID-WSF Authentication Protocol. Then, it defines an ID-WSF Authentication Service Identity Providers may offer, which is based on the authentication protocol. This authentication service enables Web Services Consumers and/or Liberty-enabled User Agents or Devices to authenticate with Identity Providers using various authentication mechanisms and obtain ID-WSF security tokens. Next, it defines the ID-WSF Single Sign-On Service, which provides SAML authentication assertions to Web Service Consumers via profiles of the SAML 2.0 Authentication Request protocol, enabling Web Service Consumers and/or Liberty-enabled User Agents or Devices to interact with SAML-based services. Finally, it defines the ID-WSF Identity Mapping Service, which allows Web Service Consumers to obtain identity tokens for use in web service invocations and referencing principals while preserving privacy.

2. Notation and Conventions

This specification uses schema documents conforming to W3C XML Schema [Schema1-2] and normative text to describe the syntax and semantics of XML-encoded protocol messages.

2.1. Requirements Keywords

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]:

> “They MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)”

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

2.2. XML Namespaces

This specification uses the XML namespace prefixes listed in Table 1.

Table 1. XML Namespaces used in this specification

Prefix	**Namespace**
sa:	Represents the ID-WSF Authentication Service namespace: **urn:liberty:sa:2006-08** **Note** This is the point of definition of this namespace. This namespace is the default for instance fragments, type names, and element names in this document when a namespace is not explicitly noted.
disco:	Represents the namespace defined in [LibertyDisco].
sec:	Represents the namespace defined in [LibertySecMech].
md:	Represents the namespace defined in [SAMLMeta2].
pp:	Represents the namespace defined in [LibertyIDPP].
s:	Represents the SOAP namespace: `http://www.w3.org/2001/12/soap-envelope`, defined in [SOAPv1.1].
saml2:	Represents the SAML V2.0 Assertion namespace defined in [SAMLCore2]
samlp2:	Represents the SAML V2.0 Protocol namespace defined in [SAMLCore2]
sb:	Represents the Liberty namespace defined in [LibertySOAPBinding]
lu:	Represents the Liberty ID-WSF utility namespace (see Appendix E).
xs:	Represents the W3C XML schema namespace (http://www.w3.org/2001/XMLSchema) defined in [Schema1-2].

3. Terminology

This section defines key terminology used in this specification. Definitions for these, as well as other Liberty-specific terms, may also be found in [LibertyGlossary]. Note that the definition of some terms below differ slightly from the definition given in [LibertyGlossary]. For example see the definitions for *client* and *server*. This is because in such cases, the definition given in [LibertyGlossary] is a more general one, and the definition given here is a narrower one, specific to the context of this specification. See also [RFC2828] for overall definitions of security-related terms, in general. Other specific references are also cited below.

authentication *Authentication* is the process of confirming a *system entity*'s asserted *identity* with a specified, or understood, level of confidence [TrustInCyberspace].

authentication assertion A *SAML assertion* typically consisting of a single `<AuthenticationStatement>`. The assertion issuer is stating that the subject of the assertion authenticated with it at some point in time. Assertions are typically time-limited [SAMLCore2].

authentication exchange See *authentication protocol exchange.*

authentication mechanism An *authentication mechanism* is a particular, identifiable, process or technique that results in a confirmation of a *system entity*'s asserted identity with a specified, or understood, level of confidence.

authentication protocol exchange *Authentication protocol exchange* is the term used in [RFC4422] to refer to the sequence of messages exchanged between the *client* and *server* as specified and governed by a particular *SASL mechanism* being employed to effect an act of *authentication*.

authentication server The precise, specific *role* played by a *server* in the protocol message exchanges defined in this specification.

Authentication Service (AS) Short form of "ID-WSF Authentication Service." The AS is a discoverable ID-WSF service.

Authentication Service Consumer A *Web Service Consumer* (WSC) implementing the *client*-side of the ID-WSF Authentication Protocol (which is defined in this specification).

Authentication Service Provider (AS Provider) A *Web Service Provider* (WSP) implementing the *server*-side of the ID-WSF Authentication Service defined in this specification (Section 5: Authentication Service).

client A *role* assumed by a *system entity* who either explicitly or implicitly initiates an authentication exchange [RFC2828]. *Client* is implicitly defined in [RFC4422]. Also known as a *SASL client*.

discoverable A *discoverable* "in principle" service is one having a *service type URI* assigned (this is typically in done in the specification defining the service). A discoverable "in practice" service is one that is registered in some discovery service instance.

ID-WSF *services* are by definition discoverable "in principle" because such services are assigned a *service type URI* facilitating their registration in *Discovery Service* instances.

final SASL response The final `<SASLResponse>` message sent from the *server* to the *client* in an *authentication exchange*.

ID-WSF EPR

An *ID-WSF Endpoint Reference* is a reference to a *service instance*. It contains the address, security context, and other metadata necessary for contacting the identified service instance. The underlying structure of an ID-WSF EPR is based on the *wsa:EndpointReference* of [WSAv1.0-SOAP] [WSAv1.0].

initial response

A [RFC4422] term referring to *authentication exchange* **data** sent by the *client* in the *initial SASL request*. It is used by a subset of SASL mechanisms. See Section 5.1 of [RFC4422].

initial SASL request

The initial `<SASLRequest>` message sent from the *client* to the *server* in an *authentication exchange*.

(LUAD-)WSC

A *Web Service Consumer* (WSC) that may or may not also be a *Liberty-enabled User Agent or Device*.

mechanism

A process or technique for achieving a result [Merriam-Webster].

message thread

A *message thread* is a synchronous exchange of messages in a request-response *MEP* between two *SOAP nodes*. All the messages of a given message thread are "linked" via each message's `<wsa:RelatesTo>` header block value being set, by the sender, from the previous successfully received message's `<wsa:MessageID>` header block value.

requester

A *system entity* which sends a *service request* to a *provider*.

role

A function or part performed, especially in a particular operation or process [Merriam-Webster].

SASL mechanism

A *SASL mechanism* is an *authentication mechanism* that has been profiled for use in the context of the *SASL framework* [RFC4422]. See [RFC2444] for a particular example of profiling an existing authentication mechanism—one-time passwords [RFC2289]—for use in the SASL context. SASL mechanisms are "named"; Mechanism names are listed in the column labeled as "MECHANISMS" in [SASLReg] (a copy of this registry document is reproduced in Appendix A for informational convenience; implementors should always fetch the most recent revision directly from [IANA]).

server

A *role* donned by a *system entity* which is intended to engage in defined exchanges with *clients*. This term is implicitly defined in [RFC4422] and in this specification is always synonymous with *authentication server*.

service instance

The physical instantiation of a service. A service instance is a web service at a distinct endpoint.

Service Provider (SP)

(1)A *role* donned by *system entities*. In the Liberty architecture, *Service Providers* interact with other system entities primarily via vanilla HTTP.

(2) From a Principal's perspective, a Service Provider is typically a website providing services and/or goods.

SOAP header block

A [SOAPv1.2] term meaning: An [element] used to delimit data that logically constitutes a single computational unit within the SOAP header. In [SOAPv1.1] these are known as simply *SOAP headers*, or simply *headers*. This specification borrows the SOAPv1.2 terminology.

SOAP node

A [SOAPv1.2] term describing *system entities* who are parties to SOAP-based message exchanges that are, for purposes of this specification, also the ultimate destination of the exchanged messages, i.e., *SOAP endpoints*. In [SOAPv1.1], SOAP nodes are referred to as *SOAP endpoints*, or simply *endpoints*. This specification borrows the SOAPv1.2 terminology.

system entity
An active element of a computer/network system. For example, an automated process or set of processes, a subsystem, a person or group of persons that incorporates a distinct set of functionality [SAMLGloss2].

user identifier
AKA *user name* or *Principal*.

web service
Generically, a *service* defined in terms of an *XML*-based protocol, often transported over *SOAP*, and/or a service whose instances, and possibly data objects managed therein, are concisely addressable via *URIs*.

As specifically used in Liberty specifications, usually in terms of *WSCs* and *WSPs*, it means a web service that's defined in terms of the *ID-** "stack," and thus utilizes [LibertySOAPBinding], [LibertySecMech], and is "discoverable" [LibertyDisco].

Web Service Consumer
A *role* donned by a *system entity* when it makes a request to a *web service*.

Web Service Provider
A *role* donned by a *system entity* when it provides a *web service*.

4. Authentication Protocol

This section defines the ID-WSF Authentication Protocol. This protocol facilitates authentication between two ID-* entities, and is a profile of SASL [RFC4422].

4.1. Conceptual Model

The conceptual model for the ID-WSF Authentication Protocol is as follows: an ID-WSF *system entity*, acting in a *Web Services Consumer* (WSC) *role*, makes an authentication request to another ID-WSF system entity, acting in a *Web Service Provider* (WSP) role, and if the WSP is willing and able, an authentication exchange will ensue.

The authentication exchange is comprised of SOAP-bound ID-* messages [LibertySOAPBinding], and can involve an arbitrary number of round trips, dictated by the particular SASL mechanism employed [RFC4422]. The WSC may have out-of-band knowledge of the server's supported SASL mechanisms, or it may send the server its own list of supported SASL mechanisms and allow the server to choose one from among them.

At the end of this exchange of messages, the WSC will either be authenticated or not, the nature of the authentication depending upon the SASL mechanism that was employed. Also depending on the SASL mechanism employed, the WSP may be authenticated as well.

Other particulars, such as how the WSC knows which WSP to contact for authentication, are addressed below in Section 6: Single Sign-On Service.

Note

This document does not specify the use of SASL security layers.

4.2. Schema Declarations

The XML schema [Schema1-2] normatively defined in this section is constituted in the XML Schema file: `liberty-idwsf-authn-svc-v2.0.xsd`, entitled " Liberty ID-WSF Authentication Service XSD v2.0 " (see Appendix C).

Additionally, Liberty ID-WSF Authentication Service XSD v2.0 imports items from `liberty-idwsf-utility-v2.0.xsd` (see Appendix E: Liberty ID-WSF Utility XSD v2.0), and also from `saml-schema-protocol-2.0.xsd` (see [SAMLCore2]).

4.3. SOAP Header Blocks and SOAP Binding

This specification does not define any SOAP header blocks. Section 4.3.1, below, constitutes the SOAP binding statement for this specification.

4.3.1. SOAP Binding

The messages defined below in Section 4.6, e.g., `<SASLRequest>`, are *ordinary ID-* messages* as defined in [LibertySOAPBinding]. They are intended to be bound to the [SOAPv1.1] protocol by mapping them directly into the `<s:Body>` element of the `<s:Envelope>` element comprising a SOAP message. [LibertySOAPBinding] normatively specifies this binding.

Note

Implementations of this specification MUST use the "Messaging-specific Header Blocks," as specified in [LibertySOAPBinding], to establish a *message thread* and thus correlate their authentication exchanges. See Section 5.5: Authentication Service Interaction Example for an example.

4.4. SASL Profile Particulars

The ID-WSF Authentication Protocol is based on SASL [RFC4422], and thus "profiles" SASL. Section 4 of [RFC4422] specifies SASL's "profiling requirements." This section of this specification addresses some particulars of profiling SASL that are not otherwise addressed in the sections defining the protocol messages (Section 4.6: Protocol Messages), and their sequencing (Section 4.7: Sequencing of the Authentication Exchange).

4.4.1. SASL "Service Name"

The SASL "Service Name" specified herein is: **idwsf**

4.4.2. Composition of SASL Mechanism Names

The protocol messages defined below at times convey a SASL mechanism name, or a list of SASL mechanism names, as values of message element attributes.

These mechanism names are typically taken from the column labeled as "MECHANISMS" in [SASLReg], but MAY be site-specific.

These names, and lists of these names, MUST follow these rules:

- The character composition of a SASL mechanism name MUST be as defined in [IANA]'s SASL Mechanism Registry [SASLReg].
- A list of SASL mechanism names MUST be composed of names as defined above, separated by ASCII space chars (hex "20").

4.5. Authentication Exchange Security

This authentication protocol features the flexibility of having implementations being able to select at runtime the actual authentication mechanism (aka SASL mechanism) to employ. This however may introduce various vulnerabilities depending on the actual mechanism employed. Some mechanisms may be vulnerable to passive and/or active attacks. Also, since the server selects the SASL mechanism from a list supplied by the client, a compromised server, or a man-in-the-middle, can cause the weakest mechanism offered by the client to be employed.

Thus it is RECOMMENDED that the authentication protocol exchange defined herein (Section 4.7: Sequencing of the Authentication Exchange) be employed over a TLS/SSL channel [RFC4346] as amended by [RFC4366]. This will ensure the integrity and confidentiality of the authentication protocol messages. Additionally, clients SHOULD authenticate the server via TLS/SSL validation procedures. This will help guard against man-in-the-middle attacks.

4.6. Protocol Messages

This section defines the protocol's messages, along with their message element attribute values, and their semantics. The sequencing of protocol interactions, also known as the *authentication exchange*, is defined below in Section 4.7: Sequencing of the Authentication Exchange .

4.6.1. The `<SASLRequest>` Message

Figure 1 shows the schema fragment from Liberty ID-WSF Authentication Service XSD v2.0 describing the `<SASLRequest>` message. This message has the following attributes:

- **`mechanism`** [Required] — Used to convey a list of one-or-more client-supported SASL mechanism names to the server, or to signal the server if the client wishes to abort the exchange. It is included on all `<SASLRequest>` messages sent by the client.

- **`authzID`** [Optional] — The `authzID`, also known as *user identifier* or *username* or *Principal*, that the client wishes to establish as the "authorization identity" per [RFC4422].

- **`advisoryAuthnID`** [Optional] — The `advisoryAuthnID` may be used to advise the server what authentication identity will be asserted by the client via the selected SASL mechanism; i.e., it is a "hint." The `advisoryAuthnID` provides a means for server implementations to optimize their behavior on a per authentication identity basis. E.g. if a client requests to execute a certain SASL mechanism on behalf of some given authentication identity (represented by `advisoryAuthnID`) and authorization identity (represented by `authzID`) pair, the server can decide whether to proceed without having to execute the SASL mechanism (execution of which might involve more than a single round-trip). Server implementations that make use of the optional `advisoryAuthnID` attribute SHOULD be capable of processing initial `<SASLRequest>` messages that do not include the `advisoryAuthnID` attribute.

- **Any Attributes** [Optional] — Zero or more extension attributes qualified by an XML namespace other than the Authentication Service namespace.

```
<xs:element name="SASLRequest">
   <xs:complexType>
      <xs:sequence>

         <xs:element name="Data" minOccurs="0">
            <xs:complexType>
               <xs:simpleContent>
                  <xs:extension base="xs:base64Binary"/>
               </xs:simpleContent>
            </xs:complexType>
         </xs:element>

         <xs:element ref="samlp2:RequestedAuthnContext" minOccurs="0"/>

          <xs:element name="Extensions" minOccurs="0">
             <xs:complexType>
                <xs:sequence>
                   <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
                </xs:sequence>
             </xs:complexType>
          </xs:element>

      </xs:sequence>

      <xs:attribute name="mechanism"
                type="xs:string"
                use="required"/>

      <xs:attribute name="authzID"
                type="xs:string"
                use="optional"/>

      <xs:attribute name="advisoryAuthnID"
                type="xs:string"
                use="optional"/>

      <xs:anyAttribute namespace="##other" processContents="lax"/>

   </xs:complexType>
</xs:element>
```

Figure 1. `<SASLRequest>` Message Element — Schema Fragment

The `<SASLRequest>` message has the following sub-elements:

- **`<Data>`** — This element is used by the client to send SASL mechanism data to the server. In [RFC4422] parlance, this data is termed a "client response." Its content model is base64-encoded data.
- **`<samlp2:RequestedAuthnContext>`** — This element is used by the client to convey to the server a desired authentication context. It is used on only on the initial SASL request (see Section 4.7: Sequencing of the Authentication Exchange). If present, the server uses the information in the `<samlp2:RequestedAuthnContext>` in combination with `mechanism` attribute when choosing the SASL mechanism to execute. The background use case for `<samlp2:RequestedAuthnContext>` is presented in Section 5.1: Authentication Service: Conceptual Model . See also: [LibertyAuthnContext] and [LibertyProtSchema].
- **`<Extensions>`** — This contains optional request extensions that are agreed upon between the client and server. Extension elements MUST be namespace-qualified by a non-AS namespace.

```

<?xml version="1.0" encoding="UTF-8"?>

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
         xmlns:sa="urn:liberty:sa:2006-08"
         xmlns:sb="urn:liberty:wsf:soap-bind:1.0"
         xmlns:pp="urn:liberty:id-sis-pp:2003-08">

 <S:Header>

   <!-- various header blocks, as defined in the
      SOAP Binding spec, go here -->

 </S:Header>

 <S:Body>

   <sa:SASLRequest sa:mechanism="foo">
      <sa:Data>
         qwyGHhSWpjQu5yq......vUUlvONmOZtfzgFz
      <sa:Data>
   </sa:SASLRequest>

 </S:Body>

</S:Envelope>

```

Example 1. A SASLRequest Bound into a SOAP Message

4.6.1.1. `<SASLRequest>` Usage

The `<SASLRequest>` message is used to initially convey to the server a:

- list of one or more client-supported SASL mechanism names,

..in combination with optional:

- `authzID` attribute, and/or,
- `advisoryAuthnID` attribute, and/or,
- `<samlp2:RequestedAuthnContext>` element.

In the case where a single SASL mechanism name is conveyed, the `<SASLRequest>` message can contain a so-called *initial response* (see Section 5 of [RFC4422]) in the `<Data>` element.

If the server's subsequent `<SASLResponse>` message signals that the authentication exchange should continue—and thus contains a server "challenge"—the client will send another `<SASLRequest>` message, with the `<Data>` element containing the client's "response" to the challenge. This sequence of server challenges and client responses continues until the server signals a successful completion or aborts the exchange.

The `mechanism` attribute is used in these intermediate `<SASLRequest>` messages to signal the client's intentions to the server. This is summarized in the next section.

Section 4.7: Sequencing of the Authentication Exchange , in combination with the next section, normatively defines the precise `<SASLRequest>` message format as a function of the sequencing of the authentication exchange.

4.6.1.2. Values for `mechanism` attribute of `<SASLRequest>`

The list below defines the allowable values for the `mechanism` attribute of the `<SASLRequest>` message element, and the resulting message semantics.

Note

In items #2 and #1, the `mechanism` attribute contains one or more SASL mechanism names, respectively. The rules noted in Section 4.4.2: Composition of SASL Mechanism Names MUST be adhered to in such cases.

1. **Multiple SASL mechanism names** — See Example 2. In this case, the `<SASLRequest>` message MUST NOT contain any "initial response" data, and MUST be the initial SASL request. See Section 4.6.2.1.2 for details on the returned `<SASLResponse>` message in this case.

   ```
   <SASLRequest mechanism="GSSAPI OTP PLAIN"/>
   ```

   Example 2. `<SASLRequest>` Specifying Multiple Client-supported Mechanism Names

2. **A single SASL mechanism name** — In this case, the `<SASLRequest>` message MAY contain *initial response* data. See Example 3.

   ```
   <SASLRequest mechanism="GSSAPI">
     <Data>
        Q29ub3IgQ2FoaWxsIGNhc3VhbGx5IG1hbm dsZXMgcGFzc3dvcmRzCg==
     </Data>
   </SASLRequest>
   ```

   Example 3. `<SASLRequest>` Specifying a Single Mechanism Name

3. **A NULL string ("")** — This indicates to the authentication server that the client wishes to abort the authentication exchange. See Example 4.

   ```
   <SASLRequest mechanism=""/>
   ```

   Example 4. `<SASLRequest>` Message Aborting the SASL Authentication Exchange

4.6.2. The `<SASLResponse>` Message

Figure 2 shows the schema fragment from Liberty ID-WSF Authentication Service XSD v2.0 describing the `<SASLResponse>` message. This message has the following attributes:

- **`serverMechanism`** [Optional] — The server's choice of SASL mechanism from among the list sent by the client.
- **Any Attributes** [Optional] — Zero or more extension attributes qualified by an XML namespace other than the Authentication Service namespace.

```

<xs:element name="SASLResponse">
   <xs:complexType>
      <xs:sequence>

         <xs:element ref="Status"/>

         <xs:element ref="PasswordTransforms" minOccurs="0"/>

         <xs:element name="Data" minOccurs="0">
            <xs:complexType>
               <xs:simpleContent>
                  <xs:extension base="xs:base64Binary"/>
               </xs:simpleContent>
            </xs:complexType>
         </xs:element>

         <!-- ID-WSF EPRs  -->
         <xs:element ref="wsa:EndpointReference"
                  minOccurs="0"
                  maxOccurs="unbounded"/>

       </xs:sequence>

       <xs:attribute name="serverMechanism"
                  type="xs:string"
                  use="optional"/>

       <xs:anyAttribute namespace="##other" processContents="lax"/>

   </xs:complexType>
</xs:element>

```

Figure 2. `<SASLResponse>` Message Element - Schema Fragment

The `<SASLResponse>` message has the following sub-elements:

- **`<Status>`** — This element is from Liberty ID-WSF Utility XSD v2.0 and is used to convey status from the server to the client. See below.
- **`<PasswordTransforms>`** — This element is used to convey to the client any required password transformations. See Section 8: Password Transformations: The `PasswordTransforms` Element .
- **`<Data>`** — This element is used to return SASL mechanism data to the client. Its content model is base64-encoded data.
- **`<wsa:EndpointReference>`** — This element is to convey to the client an ID-WSF EPR for the server, in its role as a WSP, upon a successful authentication exchange completion. Multiple instances of it may be used to also convey ID-WSF EPRs for additional instances of other services. Note that any credentials returned as a result of a successful authentication exchange are conveyed within any returned ID-WSF EPRs [LibertyDisco]. See Section 5: Authentication Service.

4.6.2.1. <SASLResponse> Usage

This message is sent by the server in response to a client <SASLRequest> message. It is used to convey "server challenges," in [RFC4422] parlance, to the client during an authentication exchange. So-called "client responses" are correspondingly conveyed to the server via the <SASLRequest> message, defined above. A given authentication exchange may occur in one "round-trip," or it may involve several round-trips. This depends on the SASL mechanism being executed.

The first <SASLResponse> sent by the server within an authentication exchange (as determined by the particular authentication mechanism being used) is explicitly distinguished from subsequent <SASLResponse> messages in terms of child elements and attributes. The final <SASLResponse> sent by the server in an authentication exchange is similarly distinguished, although with its own particular characteristics. These details are specified below in Section 4.7: Sequencing of the Authentication Exchange .

It is possible for different authentication mechanisms to be sequenced, the client authenticating to the server with one after another. For example, after a principal is authenticated with name and password (e.g., with PLAIN or CRAM-MD5), the service may (because of service or user policy) require additional authentication with SECUR-ID. Consequently, client implementations should be prepared for a message from the service with a "Continue" status code but a different "serviceMechanism" than that established in the previous authentication exchange. The message from the service that indicates such subsequent SASL mechanism may contain a <Data> element intended for processing by an implementation of the new mechanism. The client should process this message as specified in step 5 of Section 4.7: Sequencing of the Authentication Exchange .

The <Status> element (see Figure 3) is used to convey the authentication server's assessment of the status of the authentication exchange to the client, via the code attribute (the <Status> element is declared in the Liberty ID-WSF Utility XSD v2.0).

```
<xs:complexType name="StatusType">
   <xs:annotation>
      <xs:documentation>
         A type that may be used for status codes.
      </xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="code" type="xs:string" use="required"/>
   <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
   <xs:attribute name="comment" type="xs:string" use="optional"/>
</xs:complexType>

<xs:element name="Status" type="StatusType">
   <xs:annotation>
      <xs:documentation>
         A standard Status type
      </xs:documentation>
   </xs:annotation>
</xs:element>
```

Figure 3. <Status> Element and Type - Schema Fragment (from liberty-idwsf-utility-v2.0.xsd)

In the two sections below, first the values of the code attribute of the <Status> element are discussed, followed by discussion of the various forms of <SASLResponse> messages and their semantics.

4.6.2.1.1. Values for the code attribute of <Status>

If the value of code is:

- "**Continue**" — the server expects the client to craft and send a new `<SASLRequest>` message containing data appropriate for whichever step the execution of the SASL mechanism is at.

- "**OK**" — the server considers the authentication exchange to have completed successfully.

  The `<SASLResponse>` message will typically contain ID-WSF EPR(s) (i.e., `<wsa:EndpointReference>` element(s)) containing credentials, as described below in Section 5.3: Rules for Authentication Service Providers , enabling the client to interact further with this provider, for example to invoke another ID-WSF service such as the Discovery Service.

  Additionally, the `<SASLResponse>` message can convey ID-WSF EPRs for other providers.

  See Section 4.7: Sequencing of the Authentication Exchange for the normative specification of the composition of the `<SASLResponse>` message in this case. See also Section 5.3: Rules for Authentication Service Providers .

- "**Abort**" — the server is aborting the authentication exchange. It will not send any more messages on this message thread.

4.6.2.1.2. Returning the Server's Selected SASL Mechanism

The server will choose one SASL mechanism from among the intersection of the list sent by the client and the server's set of supported and willing-to-execute SASL mechanisms. It will return the name of this selected SASL mechanism as the value for the `serverMechanism` attribute on the initial `<SASLResponse>` message. See Example 5.

```

<SASLResponse serverMechanism="DIGEST-MD5">
  <Status code="Continue"/>
  <Data>
      Q29ub3IgQ2FoaWxsIGNhc3VhbGx5IG1hbmdsZXMgcGFzc3dvcmRzC g==
  </Data>
</SASLResponse>

```

Example 5. `<SASLResponse>` Indicating Server's Chosen SASL Mechanism

If there is no intersection between the client-supplied list of SASL mechanisms and the set of supported, and willing-to-execute, server-side SASL mechanisms, then the server will return a `<SASLResponse>` message with a `code` attribute whose value is "Abort." See Example 6, and also item #3 in Section 4.7: Sequencing of the Authentication Exchange .

```

<SASLResponse>
  <Status code="Abort"/>
</SASLResponse>

```

Example 6. `<SASLResponse>` Indicating a Server-side Abort

4.7. Sequencing of the Authentication Exchange

The authentication exchange is sequenced as follows:

1. The authentication exchange MUST begin by the client sending the server a `<SASLRequest>` message. This message:

- MUST contain a mechanism attribute whose value is a string containing one or more SASL mechanisms the client supports and is prepared to negotiate (see Section 4.6.1.2: Values for mechanism attribute of <SASLRequest>).

- MAY contain a <Data> element containing an initial response, specific to the cited SASL mechanism, if the mechanism attribute contains only a single SASL mechanism. See section 5 of [RFC4422].

- MAY contain a <samlp2:RequestedAuthnContext> element.

- SHOULD contain an authzID attribute whose value is an identifier string for the Principal being authenticated.

- MAY contain an advisoryAuthnID attribute whose value is an identifier asserted by the client to represent the authentication identity being established by this authentication event.

2. If the server is prepared to execute, with this client, at least one of the SASL mechanism(s) cited by the client in the previous step, then processing continues with step 4.

3. Otherwise, the server does not support, or is not prepared to negotiate, any of the SASL mechanisms cited by the client. The server MUST respond to the client with a <SASLResponse> message containing:

   - A <Status> element with a code attribute with a value of "**Abort**."

   - No <PasswordTransforms> element.

   - No <Data> element.

   - No <wsa:EndpointReference> element.

   - No serverMechanism attribute.

   After this message is sent to the client, processing continues with step 7.

4. The server sends to the client a <SASLResponse> message.

   If this message is the first <SASLResponse> sent to the client in this authentication exchange (as determined by a particular authentication mechanism, see substep "**A. Continue**," below), this message:

   - MUST contain a serverMechanism attribute whose value is a single SASL mechanism name, chosen by the server from the list sent by the client.

   - MAY contain a <Data> element containing a SASL mechanism-specific challenge.

   - MAY contain a <PasswordTransforms> element. See Section 8: Password Transformations: The PasswordTransforms Element for details on the client's subsequent obligations in this case.

   - MUST contain a <Status> element with a code attribute whose value is given by either item A, or B, or C:

A. "**Continue**" — either the execution of the SASL mechanism is not complete or the authentication exchange was successful but the server expects the client to authenticate again using a different authentication mechanism; the server expects the client to process this message and respond.

If the server is indicating that the client should continue by authenticating with a different mechanism, the server MUST specify the desired mechanism as the value for "serverMechanism." The authentication mechanism specified MUST be taken from the list previously sent by the client in the prior authentication exchange. The server MAY include a `<Data>` element (and `<PasswordTransforms>`) with content appropriate for the new authentication mechanism.

If the reason for the server indicating that the client should continue is that the client presented invalid credentials, the server SHOULD include a second level status `<Status code="InvalidCredentials">`. The server MAY also return a `<Data>` element (e.g., with a new challenge according to the mechanism already established) and the client can respond according to the mechanism. Processing continues with step 5.

B. "**OK**" — the server declares the authentication exchange has completed successfully.

In this case, this *final SASL response* message can contain, in addition to the items listed above, `<wsa:EndpointReference>` element(s), containing requisite credentials. This is specified in Section 5.3: Rules for Authentication Service Providers .

Processing continues with step 6.

C. "**Abort**" — the server declares the authentication exchange has completed unsuccessfully. For example, the user may have supplied incorrect information, such as an incorrect password. See step 7, below, for additional information.

In this case, this `<SASLResponse>` message MUST NOT contain any `<wsa:EndpointReference>` element(s).

Processing continues with step 7.

Otherwise, this message:

- MUST NOT contain a `serverMechanism` attribute.
- MAY contain a `<Data>` element containing a SASL mechanism-specific challenge.
- MUST NOT contain a `<PasswordTransforms>` element.
- MUST contain a `<Status>` element with a `code` attribute whose value is given by either item A, or B, or C:

A. "**Continue**" — the execution of the SASL mechanism is not complete; the server expects the client to process this message and respond. Processing continues with step 5.

B. "**OK**" — the server declares the authentication exchange has completed successfully.

In this case, this "final response" `<SASLResponse>` message can contain, in addition to the items listed above, `<wsa:EndpointReference>` element(s) with requisite credentials. This is specified in Section 5.3: Rules for Authentication Service Providers .

Processing continues with step 6.

C. "**Abort**" — the server declares the authentication process has completed unsuccessfully. For example, the user may have supplied incorrect information, such as an incorrect password.

If the reason for the server aborting is that the client presented invalid credentials, the server SHOULD include a second level status `<Status code="InvalidCredentials">`.

In this case, this `<SASLResponse>` message MUST NOT contain any `<wsa:EndpointReference>` element(s).

Processing continues with step 7.

5. The client sends the server a `<SASLRequest>` message. This message:

   - SHOULD contain a `mechanism` attribute set to the same value as sent by the server, as the value of the `serverMechanism` attribute, in its first `<SASLResponse>` message (see Section 4.6.2.1.2: Returning the Server's Selected SASL Mechanism).
     Note

     The client MAY, however, choose to abort the authentication exchange by setting the `mechanism` attribute to either a "null" string, or to a mechanism name different than the one returned by the server in its first `<SASLResponse>` message.

     If the client chooses to abort, processing continues with step 8.

   - SHOULD contain a `<Data>` element containing data specific to the cited SASL mechanism.

   - MUST NOT contain a `<samlp2:RequestedAuthnContext>` element.

   Processing continues with steps 4 and 5 until the server signals success, failure, or aborts — or the client aborts the exchange using the technique noted in the first bullet item, above, of this step.

6. The authentication exchange has completed successfully. The client is now authenticated in the server's view, and the server may be authenticated in the client's view, depending upon the SASL mechanism employed. Section 5.1: Authentication Service: Conceptual Model discusses what the next interaction steps between the client and server are in the ID-WSF authentication service case.

7. The authentication exchange has completed unsuccessfully due to an exception on the server side. The client SHOULD cease sending messages on this message thread.

   The reasons for an authentication exchange failing are manifold. Often it is simply a case of the user having supplied incorrect information, such as a password or pass phrase. Or, there may have been a problem on the server's part, such as an authentication database being unavailable or unreachable.

8. The client aborted the authentication exchange.

5. Authentication Service

The ID-WSF Authentication Service provides web service-based authentication facilities to Web Service Consumers (WSCs). This service is built around the SASL-based ID-WSF Authentication Protocol as specified above in Section 4.

This section first outlines the Authentication Service's conceptual model and then defines the service itself.

5.1. Conceptual Model

ID-WSF-based Web Service Providers (WSPs) may require requesters, AKA Web Service Consumers (WSCs), to present security tokens in order to successfully interact (security token specifics, are specified in [LibertySecMech]).

A Discovery Service [LibertyDisco], which itself is just a WSP, is able to create security tokens authorizing WSCs to interact with other WSPs, on whose behalf a Discovery Service has been configured to speak. Also, Discovery Service instances might themselves be configured to require WSCs to present security tokens when making requests of them.

The ID-WSF Authentication Service addresses the above conundrum by providing the means for WSCs to prove their identities—to authenticate—and obtain security tokens enabling further interactions with other services, at the same provider, on whose behalf the Authentication Service instance is authorized to speak. These offered services may be, for example, a Discovery Service or Single Sign-On Service. WSCs may then use these latter services to discover and become capable of interacting with yet other services.

Note that although an Authentication Service itself does not require requesters to present security tokens in order to interact with it, an Authentication Service may, in some situations, be configured to understand presented security tokens and use them when applying policy.

5.1.1. Stipulating a Particular Authentication Context

In some situations, a WSC may need to stipulate some of the properties for an authentication exchange. A scenario illustrating a use case of this is:

> Suppose a Principal is wielding a Liberty-enabled user agent or device (LUAD) that is acting as a WSC (i.e., a LUAD-WSC). The Principal authenticates with her bank, say, and authenticates via the ID-WSF authentication service using some authentication mechanism, such as PLAIN [SASLReg]. At some point, the Principal wants to transfer a large sum of money to the Fund for Poor Specification Editors (using some (fictitious) ID-SIS-based web service), and the bank's system indicates to the LUAD-WSC that the Principal's present authentication is "inappropriate." The bank's system also includes a `<RequestedAuthnContext>`.
>
> Now, the LUAD-WSC "knows" that it needs to help the Principal reauthenticate—as her present credentials aren't being honored for the financial transaction she wishes to carry out. So the LUAD-WSC prompts the Principal for permission to reauthenticate her, and (assuming the answer was "yes") initiates the ID-WSF Authentication Protocol with the appropriate authentication service provider, and includes the supplied-by-the-bank `<RequestedAuthnContext>`. The authentication service provider factors the requested authentication context into its selection of SASL mechanism for the ensuing authentication exchange. And upon successful authentication, the Principal is able to successfully make the funds transfer.

When initiating an authentication exchange, a WSC can stipulate some properties for the ensuing authentication event, and thus the subsequently issued (if successful) credentials. It does this by including a `<RequestedAuthnContext>` in the initial `<SASLRequest>`.

5.2. URI Declarations

The URI declarations for the ID-WSF Authentication Service are given below in Table 2.

Table 2. Authentication Service URIs

Use	URI
Service Type	*urn:liberty:sa:2006-08*
SASLRequest wsa:Action	*urn:liberty:sa:2006-08:SASLRequest*
SASLResponse wsa:Action	*urn:liberty:sa:2006-08:SASLResponse*

5.3. Rules for Authentication Service Providers

Providers offering ID-WSF Authentication Services MUST adhere to the following rules:

1. Authentication Service Providers (AS Providers) MUST implement the ID-WSF Authentication Protocol, as defined in Section 4: Authentication Protocol . The Authentication Service Provider MUST play the role of the *authentication server*.

2. Upon successful completion of an authentication exchange the **first** ID-WSF EPR, as materialized as an `<wsa:EndpointReference>` element instance and contained in the *final SASL response*, SHOULD refer to services at the Authentication Service provider—i.e., at the "same provider"—that said AS Provider can offer to the Authentication Service consumer.

   For example, Identity Providers may often also include an ID-WSF EPR for the Discovery Service of the Principal just authenticated, as well as ID-WSF EPRs for other offered services, such as an SSO Service.
   Note

   If the Authentication Service is invoked via a message whose indicated "framework version" [LibertySOAPBinding] is "2.0," then if the AS is returning ID-WSF EPRs for other services as noted above, then the AS SHOULD return ID-WSF EPRs for ID-WSFv2.0 services, rather than other versions of ID-WSF services.

   See Section 4.7: Sequencing of the Authentication Exchange , Step 4.

   The Provider MAY also include additional ID-WSF EPRs referring to services offered by other providers—i.e., providers other than the AS Provider.

3. Any included credentials SHOULD be useful for a reasonable time (note that credentials will be contained within the ID-WSF EPRs, as profiled in [LibertyDisco]). Even if the AS Consumer recently authenticated with the Authentication Service, i.e., an earlier issued credential for consumption by the AS Provider is still valid, the AS Provider SHOULD issue credential(s) that have later expiration times than the earlier issued credential(s). The AS Provider MAY choose to re-authenticate, using any of the available SASL mechanisms, or issue new credentials without a engaging in an authentication exchange. This can be accomplished by responding to the AS Consumer's initial SASL request with a final SASL response containing an ID-WSF EPR, itself containing the requisite credentials.
   Note

   Credentials containing `<saml2:AuthnStatement>`(s) should have their `<saml2:AuthnInstant>`(s) set to the time when the authentication event actually took place. See [SAMLCore2].

4. Additionally, if the first `<SASLRequest>` in an exchange contains a `<samlp2:RequestedAuthnContext>` element, then upon successful authentication, the Authentication Service MUST either: return credentials (embedded within returned ID-WSF EPR(s)) that satisfy the `<samlp2:RequestedAuthnContext>`, or, abort the authentication exchange. See [SAMLCore2] for a detailed description of the processing rules governing evaluation of the `<samlp2:RequestedAuthnContext>` element.

5. An Authentication Service instance SHOULD be deployed such that the security mechanism [LibertySecMech]:

   `urn:liberty:security:2003-08:TLS:null`

   can be used by the WSC.

Note

In practice this means that the Authentication Service should be exposed on an endpoint for which the URL should have `https` as the protocol field.

6. An Authentication Service implementation SHOULD support the following SASL mechanisms [SASLReg]: PLAIN, CRAM-MD5.

5.4. Rules for Authentication Service Consumers

WSCs implementing the client-side of the ID-WSF Authentication Protocol, and thus also known as *Authentication Service Consumers* (AS Consumers), MUST adhere to the following rules:

1. AS Consumers MUST implement the ID-WSF Authentication Protocol, as defined in Section 4: Authentication Protocol in the role of the client.

   Note

   The AS Consumer may include various SOAP header blocks, e.g., a `<wsse:Security>` element [LibertySecMech] which can house a security token(s) obtained earlier from an Authentication Service or Discovery Service [LibertyDisco]. In such a case, the Authentication Service SHOULD evaluate the presented security token(s) in combination with applicable policy, as a part of the overall authentication event. This provides a means, for example, of "security token renewal."

2. In case the AS Consumer has not been provisioned with the `<disco:SecurityMechID>` for the Authentication Service instance that it uses, the AS Consumer SHOULD assume that the required security mechanism is this one:

   `urn:liberty:security:2003-08:TLS:null`

   Note

   `<disco:SecurityMechID>` elements are contained within the `<disco:SecurityContext>` element(s), themselves occurring within ID-WSF EPRs (profiled `<wsa:EndpointReference>`s) [LibertyDisco]).

   Only when the endpoint URL of the Authentication Service is prescribed to have `http` as the protocol MAY the WSC presume a security mechanism of:

   `urn:liberty:security:2003-08:null:null`

3. It is RECOMMENDED that the WSC support the password transformations specified in Appendix B .

5.5. Authentication Service Interaction Example

Example 7 through Example 10 illustrate an example exchange between a LUAD-WSC and an ID-WSF Authentication Service (AS). The AS includes information about the Discovery Service (DS) in its final response. Here the DS is offered by the same provider.

```

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   <s:Header>
      ...
   </s:Header>
   <s:Body>
      <SASLRequest mechanism="CRAM-MD5"
         advisoryAuthnID="358408021451"
         xmlns="urn:liberty:sa:2004-04" />
   </s:Body>
</s:Envelope>

```

Example 7. The WSC sends a `<SASLRequest>` on behalf of a Principal, asserting that the authentication identity is "358408021451" and indicates it desire to use the "CRAM-MD5" SASL mechanism.

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      ...
   </S:Header>
   <S:Body>
      <SASLResponse serverMechanism="CRAM-MD5"
         xmlns="urn:liberty:sa:2004-04">
      <Status code="continue"/>
         <Data>
            ...a CRAM-MD5 challenge here...
         </Data>
      </SASLResponse>
   </s:Body>
</s:Envelope>

```

Example 8. The AS replies, agreeing to use CRAM-MD5, and issues a CRAM-MD5 challenge.

```

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   <s:Header>
      ...
   </s:Header>
   <s:Body>
      <SASLRequest mechanism="CRAM-MD5"
         xmlns="urn:liberty:sa:2004-04">
         <Data>
            ...some CRAM-MD5 response here...
         <Data>
      </SASLRequest>
   </s:Body>
</s:Envelope>

```

Example 9. The WSC responds with a CRAM-MD5 response.

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      ...
   </S:Header>
   <S:Body id="msgBody">

      <sa:SASLResponse xmlns:sa="urn:liberty:sa:2004-04"
         xmlns:disco="urn:liberty:disco:2003-08">
         <Status code="sa:OK"/>

         <wsa:EndpointReference>

            <wsa:Address>
               http://tg2.example.com:8080/tfs-soap/IdPDiscoveryService
            </wsa:Address>

            <wsa:Metadata>

               <disco:ServiceType>urn:liberty:disco:2003-08</disco:ServiceType>

               <disco:ProviderID>http://tg2.example.com:8080/tfs</disco:ProviderID>

               <ds:SecurityContext>

                  <disco:SecurityMechID>
                     urn:liberty:security:2005-02:null:Bearer
                  </disco:SecurityMechID>

                  <sec:Token>
                     <saml2:Assertion
                          ID="i1b42508103cab657f34e5ef189f28ea10dd86926"
                          Version="2.0"
                          IssueInstant="2004-02-03T22:12:33Z">
                          <Issuer>
                             http://tg2.trustgenix.com:8080/tfs
                          </Issuer>
                          ....
                          ....
                     </saml2:Assertion>
                  </sec:Token>

               </ds:SecurityContext>

            </wsa:Metadata>

         </wsa:EndpointReference>

      </sa:SASLResponse>

   </S:Body>
</S:Envelope>

```

Example 10. The AS replies with its "final" `<SASLResponse>` message, which includes credentials with which the WSC may subsequently use to invoke a DS.

6. Single Sign-On Service

The ID-WSF Single Sign-On Service (SSO Service, or SSOS) provides requesters with an ID-WSF-based means to obtain *SAML 2.0 authentication assertions* enabling them to interact with *SAML 2.0 Service Providers* (SPs) [SAMLCore2] as well as other services that accept SAML 2.0 assertions as security tokens, such as web services (including ID-WSF WSPs). The SSOS is based on a pair of profiles of the SAML 2.0 Authentication Request protocol [SAMLCore2], one of which is a refinement of the SAML 2.0 Enhanced Client or Proxy SSO profile [SAMLProf2].

This section first outlines the ID-WSF SSO Service's conceptual model and then defines the SSO Service in terms of the SAML profiles it supports.

6.1. Conceptual Model

In the Liberty architecture, it is conceivable for any concrete *system entity* to don any architectural *role* that it is physically capable of bearing. For example, a Liberty *Service Provider* (SP) is essentially just a SAML 2.0 SSO-enabled web site. Such Service Providers can also be simultaneously cast into WSC and WSP roles.

Similarly, *user agents* in the Liberty architecture range from standard web browsers, to modestly Liberty-enabled browsers (ECPs), to arbitrarily complex SOAP-based clients. These latter user agents, termed *Liberty-enabled User Agents or Devices* (LUADs) will conceivably be dynamically cast into the full range of Liberty architectural roles; they will be called upon to be a browser one moment, and a WSC the next, and even a WSP at times.

As noted in Section 5, a (LUAD-)WSC that needs to obtain security tokens in order to interact with a Discovery Service (and subsequently other ID-WSF services) can utilize an ID-WSF Authentication Service to obtain the requisite security tokens. However, not all useful services (SOAP-based or otherwise) that accept SAML security tokens will be registered with a Discovery Service. Furthermore, SAML 2.0 SSO-enabled web sites often rely on the ability to issue requests for authentication directly to less capable clients and expect them to relay the request and subsequent response. LUADs thus need a way to participate in that exchange.

Another class of use cases involves calls by one principal (or a WSC acting on behalf of a principal) to invoke services belonging to another principal. These so-called cross-principal invocations often require a WSC to utilize the invoking principal's Single Sign-On Service to obtain security tokens for the target principal's Discovery Service or other ID-WSF services.

The ID-WSF Single Sign-On Service addresses these use cases with profiles of the SAML 2.0 Authentication Request protocol [SAMLCore2] Two distinct, but similar, profiles are defined in order to address differences that arise in the content of security tokens, and protocol processing behavior that is specific to SAML 2.0 SSO SPs. The profile addressing these SPs is a refinement or specialization of the existing SAML 2.0 Enhanced Client/Proxy SSO Profile [SAMLProf2]. A SAML 2.0 SP can treat a LUAD in the same way as any other enhanced client. A second, more generic, profile permits a LUAD to obtain SAML assertions useful in accessing other kinds of services, including use cases defined in the future.

In both profiles, requesters authenticate to the SSOS using ID-WSF security mechanisms, making the SSOS itself an ID-WSF service. A LUAD wishing to interact with the SSOS can use the Authentication Service at an Identity Provider (IdP) to obtain security tokens that enable it to invoke the SSOS at that IdP in order to obtain additional security tokens to convey to SAML 2.0 SPs or other SAML-enabled services.

In fact, if a LUAD successfully authenticates with an IdP via the IdP's Authentication Service Section 5, the IdP can ensure that the LUAD will have in its possession an ID-WSF EPR (a profiled `<wsa:EndpointReference>`; [LibertyDisco]), containing any necessary credentials, for the ID-WSF Single Sign-On Service at the same IdP, simplifying the process of invoking the SSOS. Additionally, the IdP can, at the same time, ensure that the LUAD possesses an ID-WSF EPR containing any necessary credentials for the Discovery Service (DS) of the Principal wielding the LUAD, thus enabling the LUAD to simultaneously utilize SAML and ID-WSF-based services on behalf of the Principal based on one sign-on interaction, from the Principal's perspective.

6.2. Single Sign-On Service URIs

Table 3. Single Sign-On Service URIs

Use	**URI**
Service Type	*urn:liberty:ssos:2006-08*
AuthnRequest wsa:Action	*urn:liberty:ssos:2006-08:AuthnRequest*
Response wsa:Action	*urn:liberty:ssos:2006-08:Response*

6.3. ID-WSF Enhanced Client or Proxy SSO Profile

The SAML 2.0 Enhanced Client or Proxy SSO Profile [SAMLProf2] enables SSO to web sites by SAML-aware clients, but leaves the authentication of the client to the IdP out of scope. This profile is a refinement that adds the ID-WSF SOAP Binding [LibertySOAPBinding] and Security Mechanisms [LibertySecMech] specifications to the communication with the IdP, enabling a LUAD to participate in SSO to SAML 2.0-enabled web sites.

6.3.1. Profile Overview

As introduced above, this profile is simply a constrained version of the interactions specified in [SAMLProf2]. Specifically, it adds additional requirements to steps 4-6 of the ECP Profile, which involve the interactions between the IdP and the client. In all other respects, all processing rules defined by the base profile, and in turn the underlying SAML 2.0 Browser SSO Profile are observed. In particular, note that the content of the SAML protocol messages and assertion(s) used in this constrained version are entirely unchanged from [SAMLProf2].

6.3.2. Profile Description

The following sections provide detailed definitions of the individual steps. Except where noted, the steps and processing rules are as specified in the ECP Profile [SAMLProf2].

1. LUAD issues HTTP Request to Service Provider

   Step 1 is identical to step 1 of the ECP Profile, but MAY be omitted in cases in which the LUAD wishes to construct the request to the IdP itself and possesses sufficient knowledge of the SP to do so.

2. Service Provider issues `<samlp2:AuthnRequest>` to Identity Provider via LUAD

   Step 2 is identical to step 2 of the ECP Profile, but note that the `<samlp2:AuthnRequest>` message MAY be constructed independently by the LUAD rather than obtained from the SP. From the perspective of the SP, the eventual response in step 7 will be treated as unsolicited.

3. LUAD Determines Identity Provider

   Step 3, out of scope in the ECP Profile, is similarly out of scope here.

4. LUAD forwards `<samlp2:AuthnRequest>` to Identity Provider

   In step 4, the `<samlp2:AuthnRequest>` message is sent to the selected IdP's ID-WSF Single Sign-On Service endpoint using the Liberty SOAP binding [LibertySOAPBinding]. This message MUST be authenticated using a security mechanism defined by [LibertySecMech].

   When communicating with the Identity Provider, the client MUST adhere to the Liberty SOAP binding as specified in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2] the Liberty SOAP Binding shall take precedence.

Note

The client MAY (and generally will) include various other header blocks, e.g., a <wsse:Security> header block [LibertySecMech] [wss-sms]. Such a header block could contain a security token obtained from the ID-WSF Authentication Service.

Note that the <samlp2:AuthnRequest> message may also be signed by the SP (or the LUAD if constructed by it). In this and other respects, the message rules specified in the SAML 2.0 Browser SSO profile in Section 4.1.4.1 of [SAMLProf2] MUST be observed.

5. Identity Provider Identifies Principal

   In step 5, the ID-WSF peer-entity authentication mechanism used by the LUAD in step 4 MUST be used to identify the Principal.

6. Identity Provider issues <samlp2:Response> message to Service Provider via LUAD

   Step 6 is identical to step 6 of the ECP Profile, except for the use of the Liberty SOAP Binding during the exchange. The SSOS SHOULD NOT respond in step 6 with any content other than SOAP. For example, the MIME type of the HTTP response must be set according to [LibertySOAPBinding].

   Note

   This is different from the SAML 2.0 ECP profile [SAMLProf2] in which an IdP is permitted to respond with any content acceptable to the requester during the authentication process.

   The SSOS MAY take advantage of various optional header blocks defined in [LibertySOAPBinding]. For example, instead of attempting to establish a local session via an HTTP cookie, the SSOS may include a <disco:SecurityContext> element in an <sb:EndpointUpdate> header block. The requester must of course understand such header blocks.

7. LUAD forwards <samlp2:Response> to Service Provider

   Step 7 is identical to step 7 of the ECP Profile. When the client receives the <samlp2:Response> message from the IdP, it MUST NOT forward it to any location other than that specified in the AssertionConsumerServiceURL attribute contained in the mandatory <ecp:Response> header block received from the IdP.

   Note however that in the case that the LUAD initiated the profile by constructing the <samlp2:AuthnRequest> message in step 2, then there is no explicit comparison to be made against the AssertionConsumerServiceURL attribute in the ECP Response header block.

8. Service Provider Grants or Denies Access to Principal

   Step 8 is identical to step 8 of the ECP profile.

6.4. ID-WSF SAML Token Service Profile

The SAML Token Service Profile is an ID-WSF-based profile of the SAML 2.0 Authentication Request protocol [SAMLCore2] that permits a requester to obtain SAML assertions for use by one or more relying parties. Relying parties might be ID-WSF-based web services, generically defined web services, or other application services that support SAML. The profile is a direct exchange between the requester and the IdP offering the token service using the ID-WSF SOAP Binding [LibertySOAPBinding] and is authenticated using the ID-WSF Security Mechanisms specification [LibertySecMech].

6.4.1. Profile Overview

As introduced above, this profile uses the SAML 2.0 Authentication Request protocol [SAMLCore2] to enable an IdP to offer a relatively unconstrained capability to issue SAML assertions containing authentication and other information as security tokens for use by the requester with specified relying parties. Unlike the SSO-oriented profiles defined in

[SAMLProf2] and the previous section, this profile directly involves only two parties: the requester (e.g., a LUAD) and the IdP. The client in this profile is the actual requester, rather than an intermediary between the IdP and the eventual relying party.

Operationally, another difference between this profile and the SSO profiles relates to the content of the SAML assertions that can be issued. Other than a few basic assumptions, the IdP is generally expected to have sufficient knowledge of the relying parties identified by the requester so as to support the issuance of appropriately constructed assertions supporting those parties' requirements. The means by which it can obtain such knowledge are out of scope, and could include the out of band exchange of policy information, direct configuration, or it may rely on the requester to indicate to it what kinds of information to include.

This profile is a combination of the SAML Authentication Request protocol and the ID-WSF SOAP Binding [LibertySOAPBinding] and Security Mechanisms [LibertySecMech] specifications, along with a few guidelines on the use of the `<samlp2:AuthnRequest>` message.

6.4.2. Profile Description

The following sections provide detailed definitions of the individual steps.

1. Requester issues `<samlp2:AuthnRequest>` to Identity Provider

   In step 1, the requester sends its `<samlp2:AuthnRequest>` message to the selected IdP's ID-WSF Single Sign-On Service endpoint using the Liberty SOAP binding [LibertySOAPBinding]. This message MUST be authenticated using a security mechanism defined by [LibertySecMech].

   When communicating with the Identity Provider, the client MUST adhere to the Liberty SOAP binding as specified in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2] the Liberty SOAP Binding shall take precedence.
   Note

   The client MAY (and generally will) include various other header blocks, e.g., a `<wsse:Security>` header block [LibertySecMech] [wss-sms]. Such a header block could contain a security token obtained from the ID-WSF Authentication Service.

2. Identity Provider Identifies Principal

   In step 2, the ID-WSF peer-entity authentication mechanism used by the requester in step 1 MUST be used to identify the requesting Principal.

3. Identity Provider issues `<samlp2:Response>` message to Requester

   In step 3, the IdP returns a `<samlp2:Response>` message to the requester containing the status and any SAML assertion(s) issued as a result of the request. The SSOS SHOULD NOT respond with any content other than SOAP. For example, the MIME type of the HTTP response must be set according to [LibertySOAPBinding].

   The SSOS MAY take advantage of various optional header blocks defined in [LibertySOAPBinding]. For example, instead of attempting to establish a local session via an HTTP cookie, the SSOS may include a `<disco:SecurityContext>` element in an `<sb:EndpointUpdate>` header block. The requester must of course understand such header blocks.

6.4.3. Use of SAML 2.0 Authentication Request Protocol

This profile is based on the SAML 2.0 Authentication Request protocol defined in [SAMLCore2]. In the nomenclature of actors enumerated in Section 3.4 of that document, the requester is the SAML requester, presenter, and the attesting entity, and is generally the requested subject. Relying parties may be identified in the request but are not a party to the profile. There may be additional attesting entities and relying parties at the discretion of the identity provider (see below).

6.4.3.1. <samlp2:AuthnRequest> Usage

Except as described below, a requester MAY include any message content described in [SAMLCore2], Section 3.4.1. All processing rules are as defined in [SAMLCore2].

The `<saml2:Issuer>` element MUST NOT be present. This avoids duplication or conflict with the mandatory information already available from the ID-WSF SOAP Binding.

If the IdP cannot or will not satisfy the request, it MUST respond with a `<samlp2:Response>` message containing an appropriate error status code or codes.

The `ProtocolBinding` attribute MUST be included and MUST be set to `http://www.w3.org/2005/08/addressing/anonymous` to indicate the response is to be returned directly to the requester. This value clearly distinguishes this profile's use of the protocol from that of the SAML 2.0 SSO profiles. The request MUST NOT contain the `AssertionConsumerServiceURL` or `AssertionConsumerServiceIndex` attributes.

If no `<saml2:Subject>` element is included, the invocation identity associated with the request is implied to be the requested subject. The requester MAY explicitly include a `<saml2:Subject>` element in the request that names the actual Principal about which it wishes to receive an assertion. If the IdP does not recognize the requester as that Principal (or an entity allowed to attest to it), then it MUST respond with a `<samlp2:Response>` message containing an error status and no assertions.

In most cases, the identifier to return for the requested subject can be determined based on the identity of the relying party or parties. If this is not sufficient (for example if the relying party is to be treated as a member of an affiliation of providers), then the `<samlp2:NameIDPolicy>` element can be used to indicate this using the `SPNameQualifier` attribute. (Note that this precludes the identification of multiple relying parties in a single request.)

If the requester wishes to permit the IdP to establish a new identifier for the Principal if none exists, it MUST include a `<samlp2:NameIDPolicy>` element with the `AllowCreate` attribute set to "true". Otherwise, only a principal for whom the IdP has previously established an identifier usable by the relying party or parties can be authenticated successfully.

The requester MAY include one or more `<saml2:SubjectConfirmation>` elements in the request to specify attestation mechanisms to be attached to the resulting assertion(s). Usually this is done to translate ID-WSF security mechanism requirements into the corresponding SAML confirmation methods that will be needed to satisfy the relying party's security policy. Refer to [LibertySecMech] for a detailed mapping of security mechanisms to SAML confirmation methods.

The requester MAY include a `<saml2:Conditions>` element in the request. The IdP is NOT obligated to honor the requested conditions, but SHOULD return an error if it cannot do so.

The requester MAY include an `<saml2:AudienceRestriction>` element in the request to enumerate one or more relying parties by means of a `<saml2:Audience>` element containing a unique identifier for the relying party. The IdP can utilize whatever knowledge is at its disposal to determine the appropriate content to place in the resulting assertion(s), as well as how many assertions to issue, and whether the use of XML encryption is required.

The request MUST be signed or otherwise integrity protected by the binding or transport layer.

6.4.3.2. <samlp2:Response> Usage

If the IdP wishes to return an error, it MUST NOT include any assertions in the `<samlp2:Response>` message. Otherwise the `<samlp2:Response>` element MUST conform to the following:

- The `<saml2:Issuer>` element MAY be omitted, but if present MUST contain the unique identifier of the issuing IdP; the `Format` attribute MUST be omitted or have a value of `urn:oasis:names:tc:SAML:2.0:nameid-format:entity`
- It MUST contain at least one `<saml2:Assertion>`. Each assertion's `<saml2:Issuer>` element MUST contain the unique identifier of the issuing IdP; the `Format` attribute MUST be omitted or have a value of `urn:oasis:names:tc:SAML:2.0:nameid-format:entity`
- Each assertion returned MUST be signed.
- The set of one or more assertions MUST contain at least one `<saml2:AuthnStatement>` that reflects the authentication of the subject to the IdP.
- Confirmation methods and additional statements MAY be included in the assertion(s) at the discretion of the IdP. In particular, `<saml2:AttributeStatement>` elements MAY be included.
- The assertions SHOULD contain an `<saml2:AudienceRestriction>` condition element containing the unique identifier of the intended relying party or parties within the included `<saml2:Audience>` elements.
- Other conditions (and other `<saml2:Audience>` elements) MAY be included as requested or at the discretion of the IdP. Of course, all such conditions MUST be understood and accepted by the relying party in order for the assertion to be considered valid.

6.5. Use of Metadata

An IdP that offers an ID-WSF Single Sign-On Service supporting either of the profiles above SHOULD advertise support for this capability in its metadata [SAMLMeta2]. To accomplish this, it MUST include a `<md:SingleSignOnService>` element in its metadata with a `Binding` attribute of `urn:liberty:sb:2006-08`.

The `<md:IDPSSODescriptor>` element's `WantAuthnRequestsSigned` attribute MAY be used by an IdP to document a requirement that requests be signed (as opposed to protected only at the transport layer).

6.6. Inclusion of ID-WSF Endpoint References

Both SSOS profiles support the inclusion of arbitrary content in the assertions returned. Specifically, the SSOS MAY include ID-WSF EPRs, encoded as SAML attributes, for the associated Principal's Discovery Service or other ID-WSF-based services. These EPRs MAY contain additional security tokens or MAY refer to the containing assertion if appropriate.

7. Identity Mapping Service

The ID-WSF Identity Mapping Service enables a requester to obtain one or more "identity tokens." As defined in [LibertySecMech], identity tokens can be used to refer to principals in a privacy-preserving manner. The Identity Mapping Service is an ID-WSF web service that translates references to a principal into alternative formats or identifier namespaces. It is a generalization of the Name Identifier Mapping protocol defined in [SAMLCore2].

This section first outlines the service's conceptual model and then defines the protocol and message elements.

7.1. Conceptual Model

The ID-WSF Identity Mapping Service allows a requester in possession of one or more identity tokens to translate, update, or refresh them using the protocol defined here. An example employer of this service is the ID-WSF People Service [LibertyPeopleService]. A principal may also act on behalf of itself (or in conjunction with a WSC) to obtain an identity token representing that principal for use in subsequent web service invocations.

An identity token can take a variety of forms, including many SAML-based representations such as SAML assertions or SAML identifier fragments (encrypted or plaintext) such as may appear in the subject of SAML assertions (e.g., elements such as `<saml2:NameID>` or `<saml2:EncryptedID>`). A security token, such as a SAML authentication assertion can also serve as an identity token. Note that when evaluating the validity of an identity token in SAML assertion form, constraints may be imposed on its use by the issuer of the assertion.

SAML assertions used as identity tokens can also be used to communicate ID-WSF EPR and credential information, such as for the associated Principal's Discovery Service or other ID-WSF-based services.

Conceptually, the mapping protocol is a translation or exchange of one or more inputs for corresponding outputs. Each input consists of an identity token and a policy specifying the identity token to return. The security token of the invoking identity can also be referenced as the input token. The output is the requested identity token, the exact form of which may be up to the mapping service to establish.

7.2. Schema Declarations

The XML schema [Schema1-2] normatively defined in this section is constituted in the XML Schema file: `liberty-idwsf-idmapping-svc-v2.0.xsd`, entitled " Liberty ID-WSF Identity Mapping Service XSD v2.0 " (see Appendix D).

Additionally, Liberty ID-WSF Identity Mapping Service XSD v2.0 imports items from `liberty-idwsf-utility-v2.0.xsd` (see Appendix E: Liberty ID-WSF Utility XSD v2.0), and also from `liberty-idwsf-security-mechanisms-v2.0.xsd` (see [LibertySecMech]).

7.3. SOAP Binding

The Identity Mapping Service is an ID-WSF service; as such, the messages defined in Section 7.4 constitute *ordinary ID-* messages* as defined in [LibertySOAPBinding]. They are intended to be bound to the [SOAPv1.1] protocol by mapping them directly into the `<s:Body>` element of the `<s:Envelope>` element comprising a SOAP message.

[LibertySOAPBinding] normatively specifies this binding, as well as various required and optional SOAP header blocks usable with this protocol.

7.3.1. Identity Mapping Service URIs

Table 4. Identity Mapping Service URIs

Use	URI
Service Type	*urn:liberty:ims:2006-08*
IdentityMappingRequest wsa:Action	*urn:liberty:ims:2006-08:IdentityMappingRequest*
IdentityMappingResponse wsa:Action	*urn:liberty:ims:2006-08:IdentityMappingResponse*

7.4. Protocol Messages and Usage

The following request and response messages make up the Identity Mapping Service protocol:

7.4.1. Element <IdentityMappingRequest>

The `<IdentityMappingRequest>` element is of complex type **IdentityMappingRequestType**, and is defined in Figure 4 and in Liberty ID-WSF Identity Mapping Service XSD v2.0 . This type contains the following attributes and elements:

- **Any Attributes** [Optional]

  Zero or more extension attributes qualified by an XML namespace other than the Identity Mapping Service namespace.

- **`<MappingInput>`** [One or More]

  One or more elements specifying the principals to return identity tokens for, and the policies describing the contents of those tokens.

```
<xs:element name="IdentityMappingRequest" type="IdentityMappingRequestType"/>
<xs:complexType name="IdentityMappingRequestType">
    <xs:sequence>
       <xs:element ref="MappingInput" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Figure 4. Element <IdentityMappingRequest> Schema Fragment

7.4.1.1. Element <MappingInput>

The `<MappingInput>` element is of complex type **MappingInputType**, and is defined in Figure 5 and in Liberty ID-WSF Identity Mapping Service XSD v2.0 . This type contains the following attributes and elements:

- **`reqID`** [Optional]

  Uniquely identifies a `<MappingInput>` element within a request. Used by the responder to correlate the `<MappingOutput>` elements that it returns to their corresponding inputs.

- **`<sec:TokenPolicy>`** [Optional]

  A container for information specifying the characteristics of the identity token the requester wants returned to it.

- **`<sec:Token>`** [Required]

  A container for the identity token (or token reference) that specifies the principal for whom to return a new identity token.

Note

Notwithstanding the schema definition below that declares the `<sec:Token>` element to be optional, a `<sec:Token>` element MUST be present in the `<MappingInput>` element. The looser schema definition is designed to enable extension of the type by other specifications for which a mandatory `<sec:Token>` element may not be appropriate.

```
<xs:element name="MappingInput" type="MappingInputType"/>
<xs:complexType name="MappingInputType">
   <xs:sequence>
      <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
      <xs:element ref="sec:Token" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="reqID" type="lu:IDType" use="optional"/>
</xs:complexType>
```

Figure 5. Element <MappingInput> Schema Fragment

7.4.1.2. Request Usage

An `<IdentityMappingRequest>` consists of one or more `<MappingInput>` elements. If multiple `<MappingInput>` elements are included in a request, then each element MUST contain a `reqID` attribute so that the response contents can be correlated to them.

Each input consists of an identity (in the form of a `<sec:Token>`) and an optional `<sec:TokenPolicy>`.

The input identity token describes the principal for whom the requester desires a new identity token. This MAY be a reference to a token elsewhere in the message or in some other location.

The input token policy describes the nature of the identity token to be returned, generally focusing on the nature of the identifier. Often a principal will possess many alternate identifiers of different formats or scoped to different usage contexts, such as a particular SP or affiliation. The policy allows the requester to translate from the input token to some other form.

If no token policy is supplied, then the resulting output token SHOULD have the same general characteristics as the input token, save perhaps for associated information such as its lifetime. This might be used to renew a token, for example.

As identity tokens come in a variety of forms, so too the form of the input policy can vary. In the specific case of a SAML-based identity token, a `<samlp2:NameIDPolicy>` SHOULD be used, as defined in [SAMLCore2].

The token policy SHOULD identify the entity for whom the identity token is being created, if other than the requester. If this cannot be otherwise inferred from the policy, the `issueTo` attribute SHOULD be used to identify this entity. When the `<samlp2:NameIDPolicy>` is used, the `SPNameQualifier` attribute will often supply this information, at least for persistent identifiers in typical use cases, making use of the `issueTo` attribute redundant.

7.4.2. Element <IdentityMappingResponse>

The `<IdentityMappingResponse>` element is of complex type **IdentityMappingResponseType**, and is defined in Figure 6 and in Liberty ID-WSF Identity Mapping Service XSD v2.0 . This type contains the following attributes and elements:

- **Any Attributes** [Optional]

  Zero or more extension attributes qualified by an XML namespace other than the Identity Mapping Service namespace.

- **<lu:Status>** [Required]

  The status of the request. This element is defined in Liberty ID-WSF Utility XSD v2.0 .

- **<MappingOutput>** [Zero or More]

  Zero or more elements containing the identity tokens returned.

```
<xs:element name="IdentityMappingResponse" type="IdentityMappingResponseType"/>
<xs:complexType name="IdentityMappingResponseType">
   <xs:sequence>
      <xs:element ref="lu:Status"/>
      <xs:element ref="MappingOutput" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Figure 6. Element <IdentityMappingResponse> Schema Fragment

7.4.2.1. Element <MappingOutput>

The <MappingOutput> element is of complex type **MappingOutputType**, and is defined in Figure 7 and in Liberty ID-WSF Identity Mapping Service XSD v2.0 . This type contains the following attributes and elements:

- **reqRef** [Optional]

  Uniquely identifies a <MappingInput> element within the corresponding request. Used to correlate <MappingOutput> elements to their corresponding inputs.

- **<sec:Token>** [Required]

  A container for the identity token (or token reference) returned by the responder.

```
<xs:element name="MappingOutput" type="MappingOutputType"/>
<xs:complexType name="MappingOutputType">
   <xs:sequence>
      <xs:element ref="sec:Token"/>
   </xs:sequence>
   <xs:attribute name="reqRef" type="lu:IDReferenceType" use="optional"/>
</xs:complexType>
```

Figure 7. Element <MappingOutput> Schema Fragment

7.4.2.2. Response Usage

An <IdentityMappingResponse> consists of a status element and zero or more <MappingOutput> elements, one for each successfully processed token request. Unsuccessfully processed <MappingInput> elements do not result in a corresponding <MappingOutput> element. If multiple <MappingInput> elements were included in a request, then each output element MUST contain a reqRef attribute matching it to the corresponding input element.

Each output element consists of an identity token (in the form of a <sec:Token>).

Any tokens returned MUST be constructed in accordance with the policy supplied in the input. A specific exception to this requirement is that any `validUntil` attribute specified by the requester MAY be ignored. If no policy was specified, the identity token to return is presumed to be of the same nature as the identity token used as input.

The responder MUST take appropriate steps to ensure the privacy of the principal by encrypting the resulting identity information such that only the principal and parties known to be privy to the information can read it.

If each input element is satisfied in the resulting response, then the responder MUST return a top-level `<lu:Status>` `code` of "OK."

If at least one, but not all, of the resulting inputs cannot be satisfied, then the responder MUST return a top-level `<lu:Status>` `code` of "Partial." It MAY return nested `<lu:Status>` elements reflecting the specific result of the failed inputs.

If none of the resulting inputs can be satisfied, then the responder MUST return a top-level `<lu:Status>` `code` of "Failed." It MAY return nested `<lu:Status>` elements reflecting the specific result of the failed inputs.

When multiple inputs are present, any nested `<lu:Status>` elements MUST contain a `ref` attribute equal to the associated `<MappingInput>`'s `reqID` attribute.

7.4.2.3. Second-Level Status Codes

The following second-level codes are defined to represent common error conditions that may arise. Others may be defined by implementations as required.

- "UnknownPrincipal" — the input token did not match a principal known to the service
- "BadInput" — the input token or policy was malformed or not understood
- "Denied" — the requested token translation was a violation of user or system policy

7.5. SAML Identity Tokens

As described in [LibertySecMech], identity tokens can be expressed in many ways. Even in the specific case of SAML, many different formulations are possible, depending on the requirements. This section outlines a few common ways of expressing identification using SAML with different security and privacy characteristics. The identity mapping service is responsible for selecting an appropriate choice based on the requester, input policy, and the expected purpose. It is outside the scope of this specification how such purposes are to be understood.

7.5.1. Assertions

SAML assertions can be used as identity tokens that reference the subject of the assertion. Such assertions are generally signed for integrity, and often contain no statements, only a `<saml2:Subject>` element, possibly with conditions that limit its use.

When privacy is required, a `<saml2:EncryptedID>` element SHOULD be used in the subject. The decrypted element SHOULD NOT itself be an assertion (as it would be redundant).

If it is unnecessary to reveal the content of the enclosing assertion to the requester, then a `<saml2:EncryptedAssertion>` element SHOULD be used, as it is simpler for the requester to handle. Alternatively, a `<saml2:EncryptedID>` element could be returned directly (see the following section).

SAML assertions used as identity tokens MAY contain ID-WSF EPR attributes and credentials, such as for the associated Principal's Discovery Service or other ID-WSF-based services.

7.5.2. Identifiers

SAML identifier elements (`<saml2:BaseID>`, `<saml2:NameID>`, or `<saml2:EncryptedID>`) can also be used as identity tokens. Encrypted identifiers, in particular, can contain actual signed assertions, making them potential carriers of the assertion forms described in the previous section.

However, in cases where privacy is not a consideration and limits on the use of the identifier are not relevant, a plaintext form may also be useful, particularly as an input token to a mapping request. For example, an SP that shares an identifier for a principal with an IdP offering an Identity Mapping service might use a `<saml2:NameID>` by itself as an input token.

7.6. Security and Privacy Considerations

Privacy is a critical consideration in the operation of the Identity Mapping Service, because its primary purpose is to enable entities to invoke services on behalf of principals without requiring the use of globally unique identifiers. Because identifiers in ID-WSF are typically scoped to particular providers, care must be exercised when allowing providers to map between them.

In particular, it is usually the case that the identifiers returned by the IMS SHOULD be encrypted when returned to parties other than those with prior knowledge of them. The use of XML encryption generally results in unique ciphertext each time a particular value is encrypted, preventing correlation by parties without access to the underlying plaintext.

It is also important for the IMS to take into consideration the relationship between the input and output tokens. Under most circumstances, it should not be possible for a requester to map to its own namespace from another, as this would permit the requester to correlate the original identifier to one that it knows. Rather, the IMS is more generally used to map from a known identifier into another entity's namespace, returning the result in an encrypted form so that it can be decrypted by that entity.

7.7. Example Identity Mapping Exchange

The following example shows a request for a SAML identity token. The policy and input token indicate a request to map from an identifier scoped to one SP into an identifier scoped to another. In this case, the input token is a bare identifier (probably extracted from another SAML token).

```

<sa:IdentityMappingRequest>
   <sa:MappingInput>
      <sec:TokenPolicy type="urn:liberty:security:2006-08:IdentityTokenType:SAML20Assertion">
         <samlp2:NameIDPolicy Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
            SPNameQualifier="https://spb.example.com"/>
      </sec:TokenPolicy>
      <sec:Token>
         <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
            NameQualifier="https://idp.example.com" SPNameQualifier="https://spa.example.com">
            DBC63923-C718-4249-83CE-1E53D80D8A4A
         </saml2:NameID>
      </sec:Token>
   </sa:MappingInput>
</sa:IdentityMappingRequest>
```

The following is a possible response to the request above. The returned token is a signed SAML assertion with an encrypted name identifier. The requester can establish the expiration from the response, giving it guidance as to when the token might need renewal.

```
<sa:IdentityMappingResponse>
   <sa:Status code="OK"/>
   <sa:MappingOutput>
      <sec:Token>
         <saml2:Assertion Version="2.0" IssueInstant="2006-03-19T07:35:00Z"
            ID="e9ab6ff0-4ee0-4ce2-868f-18873bdc87de">
            <saml2:Issuer>https://idp.example.com</saml2:Issuer>
            <ds:Signature>...</ds:Signature>
            <saml2:Subject>
               <saml2:EncryptedID>
                  <xenc:EncryptedData>U2XTCNvRX7Bl1NK182nmY00TEk==</xenc:EncryptedData>
               </saml2:EncryptedID>
            </saml2:Subject>
            <saml2:Conditions NotOnOrAfter="2006-03-19T08:35:00Z">
               <saml2:AudienceRestriction>
                  <saml2:Audience>https://spb.example.com</saml2:Audience>
               </saml2:AudienceRestriction>
            </saml2:Conditions>
         </saml2:Assertion>
      </sec:Token>
   </sa:MappingOutput>
</sa:IdentityMappingResponse>
```

Example 11.

8. Password Transformations: The PasswordTransforms Element

This section defines the <PasswordTransforms> element. Authentication servers MAY use this element to convey password pre-processing obligations to clients.

For example, an authentication server may have been configured such that it presumes that the strings users enter as their passwords have been pre-processed in some fashion before being further processed and/or stored. For example the passwords may be truncated to a given length, and all upper case characters may be folded to lower case, and whitespace may be eliminated. The authentication server can communicate these requirements dynamically to clients using the <PasswordTransforms> element in an initial <SASLResponse>. See Figure 8.

```
<xs:element name="PasswordTransforms">

   <xs:annotation>
      <xs:documentation>
         Contains ordered list of sequential password transformations
      </xs:documentation>
   </xs:annotation>

   <xs:complexType>
      <xs:sequence>

         <xs:element name="Transform" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>

                  <xs:element name="Parameter"
                           minOccurs="0"
                           maxOccurs="unbounded">
                     <xs:complexType>
                      <xs:simpleContent>
                           <xs:extension base="xs:string">
                              <xs:attribute name="name"
                                         type="xs:string"
                                         use="required"/>
                           </xs:extension>
                      </xs:simpleContent>
                     </xs:complexType>
                  </xs:element>

               </xs:sequence>

               <xs:attribute name="name"
                         type="xs:anyURI"
                         use="required"/>

               <xs:anyAttribute namespace="##other" processContents="lax"/>

            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Figure 8. The PasswordTransforms element

```
<PasswordTransforms>
   <Transform name="urn:liberty:sa:pm:truncate">
      <Parameter name="length">8</Parameter>
   </Transform>
   <Transform name="urn:liberty:sa:pm:lowercase" />
</PasswordTransforms>
```

Figure 9. Example of a PasswordTransforms

Servers MAY include a <PasswordTransforms> element along with their **initial** <SASLResponse> to a client. A <PasswordTransforms> element contains one or more <Transform> elements. Each <Transform> is identified by the value of the name attribute which must be a URI [RFC3986]. This URI MUST specify a particular transformation on the password. Transforms are specified elsewhere, for example in configuration data at implementation- and/or deployment-time. A basic set is specified in Appendix B: Password Transformations.

A client receiving an **initial** <SASLResponse> message containing a <PasswordTransforms> element MUST apply the specified transformations to any password that is used as input for the SASL mechanism indicated in the <SASLResponse>.

The client MUST apply the transformations in the order given in the <PasswordTransforms> element, and MUST apply each transform to the result of the preceding transform. Of course, the first transform MUST be applied to the raw password.

Unless the specification of a <Transform> states otherwise, it is specified in terms of [Unicode] *abstract characters*. An abstract character is a character as rendered to a user. Since an abstract character may require more than one octet to represent, there is not necessarily a one-to-one mapping between an abstract character, or sequence of abstract characters, and its corresponding *coded character representation*.

For example, if a truncation transform indicates, "truncate after the first eight characters," the characters after the eighth abstract character should be removed; in some languages and character encodings this could mean that more than 8 octets remain.

See also Appendix B.

9. Acknowledgments

This spec leverages techniques and ideas from draft-nystrom-http-sasl-xx (an IETF Internet-Draft), RFC3080, RFC2251, RFC2829, RFC2830, et al (all are various IETF Requests For Comments). The authors of those specs are gratefully acknowledged. Thanks also to Alexy Melnikov, Paul Madsen, and RL "Bob" Morgan for their feedback and insights. The docbook source code for this specification was hand set to the tunes of Brad, Bob Mould, Weather Report, Miles Davis, John Coltrane, Liz Phair, The Wallflowers, Alan Holdsworth, Chick Corea, Jennifer Trynin, Elisa Korenne, The Cowboy Junkies, Fugazi, Blues Traveler, Blink-182, CSN, Pearl Jam, and various others. Thanks also to whatever deities are responsible for the existence of coffee, dark chocolate, and fermented cereals.

References

Normative

[LibertyAuthnContext] Madsen, Paul, eds. "Liberty ID-FF Authentication Context Specification," Version 2.0-01, Liberty Alliance Project (21 November 2004). *http://www.projectliberty.org/specs*

[LibertyClientProfiles] Aarts, Robert, Kainulainen, Jukka, Kemp, John, eds. "Liberty ID-WSF Profiles for Liberty-Enabled User Agents and Devices," Version 2.0-errata-v1.0, Liberty Alliance Project (22 January, 2007). *http://www.projectliberty.org/specs*

[LibertyDisco] Cahill, Conor, Hodges, Jeff, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (29 November, 2006). *http://www.projectliberty.org/specs*

[LibertyInteract] Aarts, Robert, Madsen, Paul, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[LibertyPAOS] Aarts, Robert, Kemp, John, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version 2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertyPeopleService] Koga, Yuzo, Madsen, Paul, eds. "Liberty ID-WSF People Service Specification," Version 1.0-errata-v1.0, Liberty Alliance Project (06 March, 2007). *http://www.projectliberty.org/specs*

[LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version 1.2-errata-v3.0, Liberty Alliance Project (14 December 2004). *http://www.projectliberty.org/specs*

[LibertySecMech] Hirsch, Frederick, eds. "Liberty ID-WSF Security Mechanisms Core," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version v2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, Whitehead, Greg, Madsen, Paul, eds. "Liberty ID-WSF SOAP Binding Specification," Version 2.0-errata-v1.0, Liberty Alliance Project (21 April, 2007). *http://www.projectliberty.org/specs*

[LibertyIDWSFv20Errata] Champagne, Darryl, Lockhart, Rob, Tiffany, Eric, eds. "Liberty ID-WSF 2.0 Errata," Version 1.0, Liberty Alliance Project (13 April, 2007). *http://www.projectliberty.org/specs*

[RFC2119] S. Bradner "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, The Internet Engineering Task Force (March 1997). *http://www.ietf.org/rfc/rfc2119.txt*

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., eds. (June 1999). "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2616.txt*

[RFC3066] Alvestrand, H., eds. (January 2001). "Tags for the Identification of Languages," RFC 3066., Internet Engineering Task Force *http://www.ietf.org/rfc/rfc3066.txt*

[RFC3986] Berners-Lee, T., Fielding, R., Masinter, L., eds. (January 2005). "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986 (Obsoletes RFC2732, RFC2396, RFC1808) (Updates RFC1738) (Also STD0066) (Status: STANDARD), The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc3986.txt*

[RFC4346] Dierks, T., Rescorla, E., eds. (April 2006). "The Transport Layer Security (TLS) Protocol," Version 1.1 RFC 4346, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc4346.txt*

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., Wright, T., eds. (April 2006). "Transport Layer Security (TLS) Extensions," RFC 4366, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc4366.txt*

[RFC4422] "Simple Authentication and Security Layer (SASL)," Melnikov, A., Zeilenga, K., eds. (June 2006). RFC 4422, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc4422.txt*

[SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (2 September 2003). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1," SAML v1.1, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf*

[SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf*

[SAMLGloss2] Hodges, Jeff, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Glossary for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML 2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-glossary-2.0-os.pdf*

[SAMLMeta2] Cantor, Scott, Moreh, Jahan, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf*

[SAMLProf2] Hughes, John, Cantor, Scott, Hodges, Jeff, Hirsch, Frederick, Mishra, Prateek, Philpott, Rob, Maler, Eve, eds. (15 March, 2005). "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf*

[SASLReg] "Simple Authentication and Security Layer (SASL) Mechanisms," Internet Assigned Numbers Authority (IANA) *http://www.iana.org/assignments/sasl-mechanisms*

[Schema1-2] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (28 October 2004). "XML Schema Part 1: Structures Second Edition," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/xmlschema-1/*

[SOAPv1.1] "Simple Object Access Protocol (SOAP) 1.1," Box, Don, Ehnebuske, David , Kakivaya, Gopal, Layman, Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Winer, Dave, eds. World Wide Web Consortium W3C Note (08 May 2000). *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/*

[Unicode] The Unicode Consortium (2003). "The Unicode Standard, version 4.0," Addison-Wesley *Unicode 4.0.0 [http://www.unicode.org]*

[WSAv1.0] "Web Services Addressing (WS-Addressing) 1.0," Gudgin, Martin, Hadley, Marc, Rogers, Tony, eds. World Wide Web Consortium W3C Recommendation (9 May 2006). *http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/*

[WSAv1.0-SOAP] "WS-Addressing 1.0 SOAP Binding," Gudgin, Martin, Hadley, Marc, eds. World Wide Web Consortium W3C Recommendation (9 May 2006). *http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/*

[WSDLv1.1] "Web Services Description Language (WSDL) 1.1," Christensen, Erik, Curbera, Francisco, Meredith, Greg, Weerawarana, Sanjiva, eds. World Wide Web Consortium W3C Note (15 March 2001). *http://www.w3.org/TR/2001/NOTE-wsdl-20010315*

Informational

[IANA] "The Internet Assigned Numbers Authority," *http://www.iana.org/*

[LibertyIDPP] Kellomäki, Sampo, Lockhart, Rob, eds. "Liberty ID-SIS Personal Profile Service Specification," Version 1.1, Liberty Alliance Project (29 September, 2005). *http://www.projectliberty.org/specs*

[LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework Overview," Version 2.0, Liberty Alliance Project (30 July, 2006). *http://www.projectliberty.org/specs*

[Merriam-Webster] "Merriam-Webster Dictionary," *http://www.merriam-webster.com/*

[RFC2289] "A One-Time Password System," N. Haller C. Metz P. Nessner M. Straw (February 1998). RFC 2289, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2289.txt*

[RFC2444] Newman, C., eds. (October 1998). "The One-Time-Password SASL Mechanism," RFC 2444, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2444.txt*

[RFC2828] Shirey, R., eds. (May 2000). "Internet Security Glossary," RFC 2828., Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2828.txt*

[RFC3163] Zuccherato, R., Nystrom, M., eds. (August 2001). "ISO/IEC 9798-3 Authentication SASL Mechanism," RFC 3163, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc3163.txt*

[SAMLBind2] Cantor, Scott, Hirsch, Frederick, Kemp, John, Philpott, Rob, Maler, Eve, eds. (15 March 2005). "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0," SAML V2.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf*

[SOAPv1.2] "SOAP Version 1.2 Part 1: Messaging Framework," Gudgin, Martin, Hadley, Marc, Mendelsohn, Noah, Moreau, Jean-Jacques, Nielsen, Henrik Frystyk, eds. World Wide Web Consortium W3C Recommendation (07 May 2003). *http://www.w3.org/TR/2003/PR-soap12-part1-20030507/*

[TrustInCyberspace] Schneider, Fred B., eds. "Trust in Cyberspace," National Research Council (1999). *http://www.nap.edu/readingroom/books/trust/*

[WooLam92] Thomas Y. C. Woo Simon S. Lam "Authentication for Distributed Systems," (January, 1992). IEEE Computer Society IEEE Computer (Vol. 25, No. 1), pp. 39-52 *http://doi.ieeecomputersociety.org/10.1109/2.108052*

[wss-sms] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (January, 2004). "Web Services Security: SOAP Message Security," OASIS Standard V1.0 [OASIS 200401], Organization for the Advancement of Structured Information Standards *http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf*

[wss-saml11] Monzillo, Ronald, Kaler, Chris, Nadalin, Anthony, Hallam-Baker, Phillip, eds. (June 28, 2005). Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/download.php/13405/wss-v1.1-spec-pr-SAMLTokenProfile-01.pdf* "Web Services Security: SAML Token Profile 1.1," OASIS Public Review Draft 01,

[XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C. M., Maler, Eve, Yergeau, Francois, eds. (04 February 2004). "Extensible Markup Language (XML) 1.0 (Third Edition)," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/2004/REC-xml-20040204*

A. Listing of Simple Authentication and Security Layer (SASL) Mechanisms

Ref: [SASLReg]

Note

The file listed below IS SUBJECT TO CHANGE! It is presented here as non-normative background information only. Implementers and deployers should always retrieve a fresh copy of this file from [IANA].

```

SIMPLE AUTHENTICATION AND SECURITY LAYER (SASL) MECHANISMS
-----------------------------------------------------------

(last updated 15 May 2006)

The Simple Authentication and Security Layer (SASL) [RFC-ietf-sasl-rfc2222bis-15.txt] is a
method for adding authentication support to connection-based
protocols.  To use this specification, a protocol includes a command
for identifying and authenticating a user to a server and for
optionally negotiating a security layer for subsequent protocol
interactions.  The command has a required argument identifying a SASL
mechanism.

SASL mechanisms are named by strings, from 1 to 20 characters in
length, consisting of upper-case letters, digits, hyphens, and/or
underscores.  SASL mechanism names must be registered with the IANA.
Procedures for registering new SASL mechanisms are described in
RFC-ietf-sasl-rfc2222bis-15.txt.

Registration Procedures:
First Come First Serve for Mechanisms
Expert Review with Mailing List for Family Name Registrations

MECHANISMS        USAGE   REFERENCE  OWNER
----------        -----   ---------  -----
KERBEROS_V4        OBSOLETE [RFC2222]   IESG <iesg@ietf.org>

GSSAPI           COMMON   [RFC2222]   IESG <iesg@ietf.org>

SKEY            OBSOLETE [RFC2444]   IESG <iesg@ietf.org>

EXTERNAL          COMMON   [RFC-ietf-sasl-rfc2222bis-15.txt]   IESG <iesg@ietf.org>

CRAM-MD5          LIMITED  [RFC2195]   IESG <iesg@ietf.org>

ANONYMOUS         COMMON   [RFC-ietf-sasl-anon-05.txt]   IESG <iesg@ietf.org>

OTP             COMMON   [RFC2444]   IESG <iesg@ietf.org>

GSS-SPNEGO        LIMITED  [Leach]    Paul Leach <paulle@microsoft.com>

PLAIN            COMMON   [RFC2595]   IESG <iesg@ietf.org>

SECURID           COMMON   [RFC2808]   Magnus Nystrom <magnus@rsasecurity.com>

NTLM            LIMITED  [Leach]    Paul Leach <paulle@microsoft.com>

NMAS_LOGIN        LIMITED  [Gayman]   Mark G. Gayman <mgayman@novell.com>

NMAS_AUTHEN        LIMITED  [Gayman]   Mark G. Gayman <mgayman@novell.com>

DIGEST-MD5        COMMON   [RFC2831]   IESG <iesg@ietf.org>

9798-U-RSA-SHA1-ENC  COMMON   [RFC3163]  robert.zuccherato@entrust.com
```

```

9798-M-RSA-SHA1-ENC  COMMON   [RFC3163]  robert.zuccherato@entrust.com

9798-U-DSA-SHA1     COMMON   [RFC3163]  robert.zuccherato@entrust.com

9798-M-DSA-SHA1     COMMON   [RFC3163]  robert.zuccherato@entrust.com

9798-U-ECDSA-SHA1   COMMON   [RFC3163]  robert.zuccherato@entrust.com

9798-M-ECDSA-SHA1   COMMON   [RFC3163]  robert.zuccherato@entrust.com

KERBEROS_V5       COMMON   [Josefsson] Simon Josefsson <simon@josefsson.org>

NMAS-SAMBA-AUTH     LIMITED  [Brimhall]  Vince Brimhall <vbrimhall@novell.com>


References
----------
[RFC2195]  Klensin, J., Catoe, R., Krumviede, P. "IMAP/POP AUTHorize
        Extension for Simple Challenge/Response," RFC 2195, MCI,
        September 1997.

[RFC2222]  J. Myers, "Simple Authentication and Security Layer (SASL),"
        RFC 2222, October 1997.

[RFC2444]  Newman, C., "The One-Time-Password SASL Mechanism," RFC
        2444, October 1998.

[RFC2595]  Newman, C., "Using TLS with IMAP, POP3 and ACAP," RFC 2595,
        Innosoft, June 1999.

[RFC2808]  Nystrom, M., "The SecurID(r) SASL Mechanism," RFC 2808,
        April 2000.

[RFC2831]  Leach, P. and C. Newman, "Using Digest Authentication as a
        SASL Mechanism," RFC 2831, May 2000.

[RFC3163]  R. Zuccherato and M. Nystrom, "ISO/IEC 9798-3 Authentication
        SASL Mechanism," RFC 3163, August 2001.

[RFC-ietf-sasl-anon-05.txt]
        K. Zeilenga, Ed., "The Anonymous SASL Mechanism," RFC XXXX,
        Month Year.

[RFC-ietf-sasl-rfc2222bis-15.txt]
        A. Melnikov and K. Zeilenga, "Simple Authentication and Security
        Layer (SASL)," RFC XXXX, Month Year.

People
------

[Brimhall] Vince Brimhall, <vbrimhall@novell.com>, April 2004.

[Gayman] Mark G. Gayman, <mgayman@novell.com>, September 2000.

[Josefsson] Simon Josefsson, <simon@josefsson.org>, January 2004.

[Leach] Paul Leach, <paulle@microsoft.com>, December 1998, June 2000.

[]
```

B. Password Transformations

This section defines a number of password transformations.

1. Truncation

The `urn:liberty:sa:pw:truncate` transformation instructs processors to remove all (Unicode abstract) subsequent characters after a given number of characters have been obtained (from the user). Subsequent processing MUST take only the given number of characters as input. The number of characters that shall remain is given in a `<Parameter>` element with name `"length"`.

```
<Transform name="urn:liberty:sa:pw:truncate">
 <Parameter name="length">8</Parameter>
</Transform>
```

Figure B.1. Example of truncation transformation

2. Lowercase

The `urn:liberty:sa:pw:lowercase` transformation instructs processors to replace all uppercase characters with lowercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters. Note that the "case" of the abstract Unicode character is decisive, i.e., only characters that have the `Uppercase` property should be replaced with equivalent characters with the `Lowercase` property. This mapping from `UPPERCASE` to `lowercase` should confirm to the relevant sections (e.g., 4.2) of [Unicode].

```
<Transform name="urn:liberty:sa:pw:lowercase" />
```

Figure B.2. Example of lowercase transformation

3. Uppercase

The `urn:liberty:sa:pw:uppercase` transformation instructs processors to replace all lowercase characters with uppercase characters. Characters that do not have case must remain unchanged. This transformation has no parameters. Note that the "case" of the abstract Unicode character is decisive, i.e., only characters that have the `Lowercase` property should be replaced with equivalent characters with the `Uppercase` property. This mapping from `lowercase` to `UPPERCASE` should confirm to the relevant sections (e.g., 4.2) of [Unicode].

```
<Transform name="urn:liberty:sa:pw:uppercase" />
```

Figure B.3. Example of uppercase transformation

4. Select

The `urn:liberty:sa:pw:select` transformation instructs processors to remove all characters except those specified in the `"allowed"` parameter. Note that the allowed characters refer to abstract Unicode characters. In the message that contains the `<Transform>` element these characters are encoded with the same encoding as used for the xml document that contains the message (usually UTF-8).

```
<Transform name="urn:liberty:sa:pw:select">
 <Parameter name="allowed">0123456789abcdefghijklmnopqrstyvwxyz</Parameter>
</Transform>
```

Figure B.4. Example of select transformation

C. liberty-idwsf-authn-svc-v2.0.xsd Schema Listing

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
    targetNamespace="urn:liberty:sa:2006-08"
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:sa="urn:liberty:sa:2006-08"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:samlp2="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:lu="urn:liberty:util:2006-08"
    xmlns="urn:liberty:sa:2006-08"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    version="09"
    >

  <xs:import
      namespace="http://www.w3.org/2005/08/addressing"
      schemaLocation="ws-addr-1.0.xsd"/>

  <xs:import
      namespace="urn:oasis:names:tc:SAML:2.0:protocol"
      schemaLocation="saml-schema-protocol-2.0.xsd"/>

  <xs:import namespace="urn:liberty:util:2006-08"
        schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>

  <!-- SASLRequest and SASLResponse ID-* messages  -->

  <xs:element name="SASLRequest">
      <xs:complexType>
          <xs:sequence>

              <xs:element name="Data" minOccurs="0">
                  <xs:complexType>
                      <xs:simpleContent>
                          <xs:extension base="xs:base64Binary"/>
                      </xs:simpleContent>
                  </xs:complexType>
              </xs:element>

              <xs:element ref="samlp2:RequestedAuthnContext" minOccurs="0"/>

               <xs:element name="Extensions" minOccurs="0">
                   <xs:complexType>
                       <xs:sequence>
                           <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
                       </xs:sequence>
                   </xs:complexType>
               </xs:element>

          </xs:sequence>

          <xs:attribute name="mechanism"
                        type="xs:string"
                        use="required"/>

          <xs:attribute name="authzID"
                        type="xs:string"
                        use="optional"/>

          <xs:attribute name="advisoryAuthnID"
                        type="xs:string"
                        use="optional"/>

      <xs:anyAttribute namespace="##other" processContents="lax"/>

   </xs:complexType>
</xs:element>

<xs:element name="SASLResponse">
   <xs:complexType>
      <xs:sequence>

         <xs:element ref="lu:Status"/>

         <xs:element ref="PasswordTransforms" minOccurs="0"/>

         <xs:element name="Data" minOccurs="0">
            <xs:complexType>
               <xs:simpleContent>
                  <xs:extension base="xs:base64Binary"/>
               </xs:simpleContent>
            </xs:complexType>
         </xs:element>

         <!-- ID-WSF EPRs  -->
         <xs:element ref="wsa:EndpointReference"
                  minOccurs="0"
                  maxOccurs="unbounded"/>

       </xs:sequence>

       <xs:attribute name="serverMechanism"
                 type="xs:string"
                 use="optional"/>

      <xs:anyAttribute namespace="##other" processContents="lax"/>

   </xs:complexType>
</xs:element>


<!-- Password Transformations  -->

<xs:element name="PasswordTransforms">

   <xs:annotation>
      <xs:documentation>
         Contains ordered list of sequential password transformations
      </xs:documentation>
   </xs:annotation>

   <xs:complexType>
      <xs:sequence>

         <xs:element name="Transform" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>

                  <xs:element name="Parameter"
                           minOccurs="0"
                           maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:simpleContent>
                           <xs:extension base="xs:string">
                              <xs:attribute name="name"
                                        type="xs:string"
                                        use="required"/>
                           </xs:extension>
                        </xs:simpleContent>
                     </xs:complexType>
```

```
                    </xs:element>

                </xs:sequence>

                <xs:attribute name="name"
                              type="xs:anyURI"
                              use="required"/>

                <xs:anyAttribute namespace="##other" processContents="lax"/>

            </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

D. liberty-idwsf-idmapping-svc-v2.0.xsd Schema Listing

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
    targetNamespace="urn:liberty:ims:2006-08"
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ims="urn:liberty:ims:2006-08"
    xmlns:sec="urn:liberty:security:2006-08"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:lu="urn:liberty:util:2006-08"
    xmlns="urn:liberty:ims:2006-08"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    >

  <xs:import
      namespace="urn:liberty:security:2006-08"
      schemaLocation="liberty-idwsf-security-mechanisms-v2.0.xsd"/>

  <xs:import namespace="urn:liberty:util:2006-08"
      schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>

  <xs:element name="MappingInput" type="MappingInputType"/>
  <xs:complexType name="MappingInputType">
      <xs:sequence>
          <xs:element ref="sec:TokenPolicy" minOccurs="0"/>
          <xs:element ref="sec:Token" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="reqID" type="lu:IDType" use="optional"/>
  </xs:complexType>

  <xs:element name="MappingOutput" type="MappingOutputType"/>
  <xs:complexType name="MappingOutputType">
      <xs:sequence>
          <xs:element ref="sec:Token"/>
      </xs:sequence>
      <xs:attribute name="reqRef" type="lu:IDReferenceType" use="optional"/>
  </xs:complexType>

  <xs:element name="IdentityMappingRequest" type="IdentityMappingRequestType"/>
  <xs:complexType name="IdentityMappingRequestType">
        <xs:sequence>
            <xs:element ref="MappingInput" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

    <xs:element name="IdentityMappingResponse" type="IdentityMappingResponseType"/>
    <xs:complexType name="IdentityMappingResponseType">
        <xs:sequence>
            <xs:element ref="lu:Status"/>
            <xs:element ref="MappingOutput" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>

</xs:schema>
```

E. liberty-idwsf-utility-v2.0.xsd Schema Listing

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:util:2006-08"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="urn:liberty:util:2006-08"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="2.0-03">

   <xs:annotation>
      <xs:documentation>
   Liberty Alliance Project utility schema.  A collection of common
   IDentity Web Services Framework (ID-WSF) elements and types.
   This schema is intended for use in ID-WSF schemas.

   This version: 2006-08

         Copyright (c) 2006 Liberty Alliance participants, see
         http://www.projectliberty.org/specs/idwsf_2_0_final_copyrights.php
      </xs:documentation>
   </xs:annotation>
   <xs:simpleType name="IDType">
      <xs:annotation>
         <xs:documentation>
            This type should be used to provide IDs to components
            that have IDs that may not be scoped within the local
            xml instance document.
         </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
   <xs:simpleType name="IDReferenceType">
      <xs:annotation>
         <xs:documentation>
            This type can be used when referring to elements that are
            identified using an IDType.
         </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
   <xs:attribute name="itemID" type="IDType"/>
   <xs:attribute name="itemIDRef" type="IDReferenceType"/>
   <xs:complexType name="StatusType">
      <xs:annotation>
         <xs:documentation>
            A type that may be used for status codes.
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element ref="Status" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="code" type="xs:string" use="required"/>
      <xs:attribute name="ref" type="IDReferenceType" use="optional"/>
      <xs:attribute name="comment" type="xs:string" use="optional"/>
   </xs:complexType>

   <xs:element name="Status" type="StatusType">
      <xs:annotation>
         <xs:documentation>
            A standard Status type
         </xs:documentation>
      </xs:annotation>
   </xs:element>

   <xs:complexType name="ResponseType">
      <xs:sequence>
```

```
         <xs:element ref="Status"     minOccurs="1" maxOccurs="1"/>
         <xs:element ref="Extension"   minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="itemIDRef" use="optional"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>
   <xs:element name="TestResult" type="TestResultType"/>
   <xs:complexType name="TestResultType">
      <xs:simpleContent>
         <xs:extension base="xs:boolean">
            <xs:attribute ref="itemIDRef" use="required"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="EmptyType">
      <xs:annotation>
         <xs:documentation> This type may be used to create an empty element </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
         <xs:restriction base="xs:anyType"/>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="Extension" type="extensionType">
      <xs:annotation>
         <xs:documentation>
            An element that contains arbitrary content extensions
            from other namespaces
         </xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="extensionType">
      <xs:annotation>
         <xs:documentation>
            A type for arbitrary content extensions from other namespaces
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:any namespace="##other" processContents="lax" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

F. liberty-idwsf-authn-svc-v2.0.wsdl WSDL Listing

```
<?xml version="1.0"?>
<definitions name="AuthenticationService"
         targetNamespace="urn:liberty:sa:2006-08"
         xmlns:tns="urn:liberty:sa:2006-08"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:S="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns="http://schemas.xmlsoap.org/wsdl/"
         xmlns:sa="urn:liberty:sa:2006-08"
         xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
                       http://schemas.xmlsoap.org/wsdl/
            http://www.w3.org/2006/02/addressing/wsdl
            http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">

   <types>
      <xs:schema>
         <xs:import namespace="urn:liberty:sa:2006-08"
              schemaLocation="liberty-idwsf-authn-svc-v2.0.xsd"/>
      </xs:schema>
   </types>

   <message name="AuthenticationSoapRequest">
      <part name="parameters" element="sa:SASLRequest"/>
   </message>
   <message name="AuthenticationSoapResponse">
      <part name="parameters" element="sa:SASLResponse"/>
   </message>

   <portType name="AuthServicePortType">
      <operation name="Authenticate">
         <input  message="sa:AuthenticationSoapRequest"
            wsaw:Action="urn:liberty:sa:2006-08:SASLRequest"/>
         <output message="sa:AuthenticationSoapResponse"
            wsaw:Action="urn:liberty:sa:2006-08:SASLResponse"/>
      </operation>
   </portType>
   <binding name="AuthenticationSoapBinding" type="sa:AuthServicePortType">
      <S:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="Authenticate">
         <input>
            <S:body use="literal"/>
         </input>
         <output>
            <S:body use="literal"/>
         </output>
      </operation>
   </binding>
   <service name="AuthenticationService">
      <port name="AuthServicePortType" binding="sa:AuthenticationSoapBinding">
         <S:address location="http://example.com/authentication"/>
      </port>
   </service>
</definitions>
```

G. liberty-idwsf-sso-svc-v2.0.wsdl WSDL Listing

```
<?xml version="1.0"?>
<definitions name="AuthenticationService"
         targetNamespace="urn:liberty:ssos:2006-08"
         xmlns:tns="urn:liberty:ssos:2006-08"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:S="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns="http://schemas.xmlsoap.org/wsdl/"
         xmlns:ssos="urn:liberty:ssos:2006-08"
         xmlns:samlp2="urn:oasis:names:tc:SAML:2.0:protocol"
         xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
                       http://schemas.xmlsoap.org/wsdl/
            http://www.w3.org/2006/02/addressing/wsdl
            http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">

   <types>
      <xs:schema>
         <xs:import namespace="urn:oasis:names:tc:SAML:2.0:protocol"
                 schemaLocation="saml-schema-protocol-2.0.xsd"/>
      </xs:schema>
   </types>

   <message name="SSOSoapRequest">
      <part name="parameters" element="samlp2:AuthnRequest"/>
   </message>
   <message name="SSOSoapResponse">
      <part name="parameters" element="samlp2:Response"/>
   </message>

   <portType name="SSOSPortType">
      <operation name="SingleSignOn">
         <input  message="ssos:SSOSoapRequest"
            wsaw:Action="urn:liberty:ssos:2006-08:AuthnRequest"/>
         <output message="ssos:SSOSoapResponse"
            wsaw:Action="urn:liberty:ssos:2006-08:Response"/>
      </operation>
   </portType>
   <binding name="SSOSSoapBinding" type="ssos:SSOSPortType">
      <S:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="SingleSignOn">
         <input>
               <S:body use="literal"/>
         </input>
         <output>
               <S:body use="literal"/>
         </output>
      </operation>
   </binding>
   <service name="SSOService">
      <port name="SSOSPortType" binding="ssos:SSOSSoapBinding">
         <S:address location="http://example.com/idmapping"/>
      </port>
   </service>
</definitions>
```

H. liberty-idwsf-idmapping-svc-v2.0.wsdl WSDL Listing

```
<?xml version="1.0"?>
<definitions name="AuthenticationService"
         targetNamespace="urn:liberty:ims:2006-08"
         xmlns:tns="urn:liberty:ims:2006-08"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:S="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns="http://schemas.xmlsoap.org/wsdl/"
         xmlns:ims="urn:liberty:ims:2006-08"
         xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
                       http://schemas.xmlsoap.org/wsdl/
            http://www.w3.org/2006/02/addressing/wsdl
            http://www.w3.org/2006/02/addressing/wsdl/ws-addr-wsdl.xsd">

   <types>
      <xs:schema>
         <xs:import namespace="urn:liberty:ims:2006-08"
              schemaLocation="liberty-idwsf-idmapping-svc-v2.0.xsd"/>
      </xs:schema>
   </types>

   <message name="IdentityMappingSoapRequest">
      <part name="parameters" element="ims:IdentityMappingRequest"/>
   </message>
   <message name="IdentityMappingSoapResponse">
      <part name="parameters" element="ims:IdentityMappingResponse"/>
   </message>

   <portType name="IdMappingPortType">
      <operation name="IdentityMapping">
         <input  message="ims:IdentityMappingSoapRequest"
            wsaw:Action="urn:liberty:ims:2006-08:IdentityMappingRequest"/>
         <output message="ims:IdentityMappingSoapResponse"
            wsaw:Action="urn:liberty:ims:2006-08:IdentityMappingResponse"/>
      </operation>
   </portType>
   <binding name="IdMappingSoapBinding" type="ims:IdMappingPortType">
      <S:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="IdentityMapping">
         <input>
               <S:body use="literal"/>
         </input>
         <output>
               <S:body use="literal"/>
         </output>
      </operation>
   </binding>
   <service name="IdMappingService">
      <port name="IdMappingPortType" binding="ims:IdMappingSoapBinding">
         <S:address location="http://example.com/idmapping"/>
      </port>
   </service>
</definitions>
```