Liberty ID-WSF Authentication Service and Single Sign-On Service Specification

Version: v2.0-02

Editors:
Robert Aarts, Nokia Corporation
Jeff Hodges, Sun Microsystems, Inc.
Paul Madsen, Entrust Inc.

Contributors:
Conor Cahill, AOL Time Warner, Inc.
Darryl Champagne, IEEE-ISTO
Gary Ellison, Sun Microsystems, Inc.
Greg Whitehead, Trustgenix, Inc.

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. Next, it 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. Finally, it defines an ID-WSF Single Sign-On Service. This service provides authentication assertions to Web Service Consumers via a profile of the ID-FF Single Sign-On Protocol, enabling Web Service Consumers to interact with ID-FF-based, or other, Service Providers.

Filename: draft-liberty-idwsf-authn-svc-v2.0-02.pdf

Notice

This document has been prepared by Sponsors of the Liberty Alliance. Permission is hereby granted to use the document solely for the purpose of implementing the Specification. No rights are granted to prepare derivative works of this Specification. Entities seeking permission to reproduce portions of this document for other uses must contact the Liberty Alliance to determine whether an appropriate license for such use is available.

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.** Implementors 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

Revision History
Revision: v2.0-02 **Date:** 24 Nov 2004

Update schema and doc to 2.0.

Various changes to reflect SAML 2

typo, changed 'md:SingleSignOnServiceProfile' to 'md:SingleSignOnProtocolProfile'

Fixed Bug 643- 'Guidance needed for password failures'

Fixed Bug 671- 'authzID and/or advisoryAuthnID optionally on any/all SASL...'

Fixed Bug 674- 'Mismatch between https and null:null'

Fixed Bug 644 - 'Guidance on chains of SASL mechanisms'. Related, clarified definition of authentication protocol exchange so that it referred to the sequence of messages exchanged in the context of a single authentication mechanism.
Revision: v2.0-01 **Date:** 21 Nov 2004

Update schema and doc to 2.0.
Revision: 1.0-errata-v1.0-03 **Date:** 23 Aug 2004

Fixed formatting issues (bug 645).
Revision: 1.0-errata-v1.0-02 **Date:** 7 Jun 2004

Broadened behavior of AS Provider in terms of Resource Offerings it may return upon successful authn. Bug# 654.

Some additional minor editorial fixups as I ran across them.
Revision: 1.0-errata-v1.0-01 **Date:** 12 May 2004

Clarification & explanation of behavior in event of authn mechanism negative (failure) result. Bug# 643 parts 1 & 2.

Recommend authn exchange be run over TLS/SSL channel. Bug# 657.

Some minor editorial fixups including breaking up Authn Service exchange example into seperate Examples. Also fixed the spec title (bug# 649).

Contents

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 [RFC2222].

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. Next, 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. Finally, it defines an ID-WSF Single Sign-On Service. This service provides authentication assertions to Web Services Consumers via a profile of the ID-FF Single Sign-On Protocol, enabling Web Services Consumers to interact with ID-FF-based Service Providers.

2. Notation and Conventions

This specification uses schema documents conforming to W3C XML Schema [Schema1] 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 Liberty namespace: **urn:liberty:sa:2004-04** **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.
ac:	Represents the Liberty namespace defined in [LibertyAuthnContext]
disco:	Represents the namespace defined in [LibertyDisco].
lib:	Represents the namespace defined in [LibertyProtSchema].
pp:	Represents the namespace defined in [LibertyIDPP].
s:	Represents the SOAP namespace: http://www.w3.org/2001/12/soap-envelope, defined in [SOAPv1.1].
sb:	Represents the Liberty namespace defined in [LibertySOAPBinding]
xs:	Represents the W3C XML schema namespace (http://www.w3.org/2001/XMLSchema) defined in [Schema1].

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 more narrow 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.

Terminology	
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 [SAMLCore11].
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 [RFC2222] 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 [RFC2222]. Also known as a *SASL client*.
discoverable	A *discoverable* "in principle" service is one having an *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*.

initial response	A [RFC2222] 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 [RFC2222].
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 `<Correlation>` header block `refToMessageID` attribute value being set, by the sender, from the previous successfully received message's `<Correlation>` header block `messageID` attribute 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* [RFC2222]. 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* who is intended to engage in defined exchanges with *clients*. This term is implicitly defined in [RFC2222] and in this specification is always synonymous with *authentication server*.
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 uses 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 uses 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 [SAMLGloss].

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 [RFC2222].

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 [RFC2222]. 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] normatively defined in this section is constituted in the XML Schema file: `lib-arch-authn-svc.xsd`, entitled " Liberty ID-WSF Authentication Service XSD " (see Appendix C).

In addition, the Liberty ID-WSF Authentication Service XSD explicitly includes, in the XML Schema sense, the Liberty ID-WSF Utility XSD file (see Appendix D), whose filename is: `lib-arch-iwsf-utility.xsd`.

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 `<sb:Correlation>` SOAP header block defined 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 [RFC2222], and thus "profiles" SASL. Section 4 of [RFC2222] 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 [RFC2246] as amended by [RFC3546]. 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 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 [RFC2222].

- **`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.

- **`id`** [Optional] — identifies a `<SASLRequest>` message element instance. This attribute MUST be used when the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the set of signed message components.

```
<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="saml:RequestedAuthnContext"
                  minOccurs="0"/>

      </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:attribute name="id"
                type="xs:ID"
                use="optional"/>

   </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 [RFC2222] parlance, this data is termed a "client response". Its content model is base64-encoded data.

- **<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 <RequestedAuthnContext> in combination with mechanism attribute when choosing the SASL mechanism to execute. The background use case for <RequestedAuthnContext> is presented in Section 5.1: Authentication Service: Conceptual Model . See also: [LibertyAuthnContext] and [LibertyProtSchema].

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

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:sa="urn:liberty:sa:2004-04"
        xmlns:sb="urn:liberty:wsf:soap-bind:1.0"
        xmlns:pp="urn:liberty:id-sis-pp:2003-08">

  <S:Header>

    <sb:Correlation S:mustUnderstand="1"
                sb:id="A13454...245"
                S:actor="http://schemas.../next"
                sb:messageID="uuid:efefefef-aaaa-ffff-cccc-eeeeffffbbbb"
                sb:timestamp="2112-03-15T11:12:12Z"/>

  </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,
- <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 5.1 of [RFC2222]) 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>
     Q29ub3IgQ2FoaWxsIGNhc3VhbGx5IG1hbmdsZXMgcGFzc3dvcmRzCg==
  </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 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.
- **`id`** [Optional] — identifies a `<SASLResponse>` message element instance. This attribute MUST be used when the message is signed as described in [LibertySecMech], and the element instance is to be included as one of the set of signed message components.

```
<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>

         <xs:element ref="disco:ResourceOffering"
                  minOccurs="0"
                  maxOccurs="unbounded"/>

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

       </xs:sequence>

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

       <xs:attribute name="id"
                 type="xs:ID"
                 use="optional"/>

   </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 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 7: 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.

- **<disco:ResourceOffering>** — This element is to convey to the client a resource offering for the server, in its role as a WSP, upon a successful authentication exchange completion. See Section 5: Authentication Service.

- **<Credentials>** — This element is used to convey to the client credentials authorizing it to interact with the server (who is acting as a WSP) upon a successful authentication exchange completion. 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 [RFC2222] 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).

```
<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="xs:NCName" 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 lib-arch-iwsf-utility.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 `<disco:ResourceOffering>` element(s) and a `<Credentials>` element, 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 contain `<disco:ResourceOffering>` element(s) and `<Credentials>` content 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>
     Q29ub3IgQ2FoaWxsIGNhc3VhbGx5IG1hbmdsZXMgcGFzc3dvcmRzCg==
  </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 5.1 of [RFC2222].
   - MAY contain a `<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.
   - MAY contain an `id` attribute.

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 `<disco:ResourceOffering>` element.
   - No `<Credentials>` element.
   - No `serverMechanism` attribute.

   The `<SASLResponse>` message MAY have an `id` 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), 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 7: Password Transformations: The `PasswordTransforms` Element for details on the client's subsequent obligations in this case.
   - MAY contain a `<id>` attribute.
   - 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 specifed 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, `<disco:ResourceOffering>` element(s) and a `<Credentials>` element. 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 `<disco:ResourceOffering>` element(s) or a `<Credentials>` element.
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.
- MAY contain a `<id>` attribute.
- 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, `<disco:ResourceOffering>` element(s) and a `<Credentials>` element. 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 `<disco:ResourceOffering>` element(s) or a `<Credentials>` element.
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 `<RequestedAuthnContext>` element.
- MAY contain an `id` attribute.

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 passphrase. Or, there may have been a problem on the server's part, such as an authentication database being unavailable or unreachable.
   Note:
   [RFC2222] and the RFCs specifying various SASL mechanisms—for example [RFC2245], [RFC2444], and [RFC3163]—are arguably not as clear as they could be with respect to the situation where an execution of the SASL mechanism fails for some reason. Though, Section 4, item 3, of [RFC2222] indicates that the server must have a means of indicating "failure of the exchange" to the client. In this version of this specification, this is handled by the server returning a status code of "**Abort**" to the client, as specified above in 4. Future versions of this specification may facilitate more fine-grained error reporting by the server.

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. But 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. Service Type Declaration

The Service Type URI for the ID-WSF Authentication Service is:

`urn:liberty:as:2004-04`

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** `<ResourceOffering>` element instances 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 add the `<disco:ResourceOffering>` and `<Credentials>` for the Discovery Service of the Principal just authenticated, as well as `<disco:ResourceOffering>`s and `<Credentials>` for other offered services, such as an SSO Service.
   In deployments where the Identity Provider and Discovery Service are tightly coupled the `<Credentials>` element MAY be shared. See Section 4.7: Sequencing of the Authentication Exchange , Step 4. The Provider MAY also include additional `<disco:ResourceOffering>` element instances, and security tokens within the `<Credentials>` element, that refer 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. 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 requisite `<ResourceOffering>`(s) and `<Credentials>`.
   Note:
   Credentials containing `<AuthenticationStatement>`(s) (SAML v1.1) or `<AuthnStatement>`(s) (SAML v2.0) should have their `<saml:AuthenticationInstant>`(s) (SAML 1.1) or `<saml:AuthnInstant>`(s) set to the time when the authentication event actually took place. See [SAMLCore11] and [SAMLCore2]

4. Additionally, if the first `<SASLRequest>` in an exchange contains a `<lib:RequestedAuthnContext>` element, then upon successful authentication, the Authentication Service MUST either: return `<Credentials>` that satisfy the `<lib:RequestedAuthnContext>`, or, abort the authentication exchange (see also the "Single Sign-On and Federation Protocol" section in [LibertyProtSchema]). To satisfy the `<lib:RequestedAuthnContext>`, any returned `<Credentials>` MUST be created according to the following rules:

   a. If one or more `<saml:AuthnContextClassRef>` or `<saml:AuthnContextDeclRef>` elements are present in the `lib:RequestedAuthnContext`, then the resulting authentication statement in the assertion (if any) MUST contain an authentication statement that conforms to the class or statement specified. Additionally, the set of supplied elements MUST be evaluated as an ordered set, where the first element is the most preferred authentication context class or statement. If none of the specified classes or statements can be satisfied, the identity provider MUST NOT include a credential and abort.

   b. Additionally, if a `Comparison` attribute is supplied, and one or more `<saml:AuthnContextStatementRef>` or `<saml:AuthnContextDeclRef>` elements are included, then the resulting authentication statement in the assertion (if any) MUST follow the rule specified in the `Comparison` attribute. If this requirement cannot be satisfied, the identity provider MUST NOT include a credential and abort.

c. If `Comparison` is specified and set to "exact", then the resulting authentication statement in the assertion (if any) MUST be the exact match of at least one of the authentication contexts specified.
If `Comparison` is specified and set to "minimum", then the resulting authentication statement in the assertion (if any) MUST be at least as strong (as deemed by the Authentication Service provider) as one of the authentication contexts specified.
If `Comparison` is specified and set to "better", then the resulting authentication statement in the assertion (if any) MUST be stronger (as deemed by the identity provider) than any specified in the supplied authentication contexts. If `Comparison` is specified and set to "maximum", then the resulting authentication statement in the assertion (if any) MUST be as strong as possible (as deemed by the identity provider) without exceeding the strength of at least one of the authentication contexts specified.

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 [Liberty-SecMech] 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:
`urn:liberty:security:2003-08:TLS:null`
Note:
`<disco:SecurityMechID>` is a subelement of `<disco:Description>`, which is a subelement of `<disco:ServiceInstance>`, which is a part of `<disco:ResourceOffering>` [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>
      <sb:Correlation s:mustUnderstand="1" xmlns:sb="urn:liberty:sb:2003-08"
         id="thisCorrHdr.1234"
         messageID="-1909129211480725265-1455516650932757068"
         timestamp="2004-02-03T22:12:26Z"
         xmlns:sb="urn:liberty:sb:2003-08"/>
   </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>
      <sb:Correlation s:mustUnderstand="1" xmlns:sb="urn:liberty:sb:2003-08"
         id="thisCorrHdr.2345"
         messageID="i48b4353f50aca1494665d61b93498c885449c868"
         refToMessageID="-1909129211480725265-1455516650932757068"
         timestamp="2004-02-03T22:12:27Z" />
   </S:Header>
   <S:Body>
      <SASLResponse serverMechanism="CRAM-MD5"
         xmlns="urn:liberty:sa:2004-04">
      <Status code="continue"/>
         <Data>
            ...a CRAM-MD5 challenge here...
         </Data>
      </SASLRequest>
   </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>
      <sb:Correlation s:mustUnderstand="1" xmlns:sb="urn:liberty:sb:2003-08"
         id="thisCorrHdr.3456"
         messageID="-41183576697762387032714476278517 2067"
         refToMessageID="i48b4353f50aca1494665d61b93498c8 85449c868"
         timestamp="2004-02-03T22:12:28Z" />
   </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 an CRAM-MD5 response.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <sb:Correlation s:mustUnderstand="1" xmlns:sb="urn:liberty:sb:2003-08"
         id="thisCorrHdr.4567"
         messageID="ic8d255ebe4b286ea0e7645ac3e9c558e11e8e8f1"
         refToMessageID="-41183576697762387032714476278 5172067"
         timestamp="2004-02-03T22:12:29Z"/>
   </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"/>
         <disco:ResourceOffering>
            <disco:ResourceID>
               http://tg2.example.com:8080/tfs/local/358408021451
            </disco:ResourceID>
            <disco:ServiceInstance>
               <disco:ServiceType>
                  urn:liberty:disco:2003-08
               </disco:ServiceType>
               <disco:ProviderID>
                  http://tg2.example.com:8080/tfs
               </disco:ProviderID>
               <disco:Description>
                  <disco:SecurityMechID>
                     urn:liberty:security:2003-08:null:Bearer
                  </disco:SecurityMechID>
                  <disco:CredentialRef>
                     i1b42508103cab657f34e5ef189f28ea10dd86926
                  </disco:CredentialRef>
                  <disco:Endpoint>
                     http://tg2.example.com:8080/tfs-soap/IdPDiscoveryService
                  </disco:Endpoint>
                  <disco:SoapAction>
                     urn:liberty:disco:2003-08
                  </disco:SoapAction>
               </disco:Description>
            </disco:ServiceInstance>
         </disco:ResourceOffering>
         <sa:Credentials>
            <saml:Assertion
               AssertionID="i1b42508103cab657f34e5ef189f28ea10dd86926"
               IssueInstant="2004-02-03T22:12:33Z"
               Issuer="http://tg2.trustgenix.com:8080/tfs"
            ....
            ....
            </saml:Assertion>
         </sa:Credentials>
      </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 Web Service Consumers (WSCs) an ID-WSF-based means to obtain *Liberty authentication assertions* enabling them to interact with *Service Providers* (SPs) [LibertyProtSchema] .

This section first outlines the ID-WSF SSO Service's conceptual model and then defines the SSO Service in terms of rules for Providers and Consumers of the service.

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 Liberty ID-FF-enabled website. Such Service Providers can also be simultaneously cast into WSC and WSP roles.

Similarly, *user agents* in the Liberty architecture range from vanilla web browsers, to modestly Liberty-enabled browsers (LECPs), 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 vanilla browser one moment, and a WSC the next, and even a WSP at times.

Similarly to the conundrum outlined in Section 5: Authentication Service, a LUAD acting as a WSC one moment (a "LUAD-WSC") and as a vanilla browser the next, will need the means to obtain authentication assertions and security tokens as necessary.

As noted in Section 5, a (LUAD-)WSCs needing to obtain security tokens in order to interact with a Discovery Service can utilize an ID-WSF Authentication Service to obtain requisite security tokens. However, in ID-FF, the user agent is assumed to be a vanilla browser, and Identity Providers vouch for browser-wielding Principals by sending authentication assertions, or "pointers" to authentication assertions (AKA "SAML artifacts" [SAMLCore11]), to Service Providers "through" Principals' browsers (e.g. via HTTP "redirects").

(LUAD-)WSCs thus need some means to cause authentication assertions to be conveyed to SPs they wish to interact with. Remember that not all SPs will be able to don a WSP role, so simply authenticating via the Authentication Service, either at some Identity Provider or with an SP/WSP providing an Authentication Service, is not a solution for this use case.

The ID-WSF Single Sign-On Service addresses this issue. It is a profile of the ID-FF Single Sign-On and Federation Protocol [LibertyProtSchema]. It provides the means for (LUAD-)WSCs to interact with SPs. See also [LibertyClient-Profiles] for additional background information.

The overall mechanism is based on two steps. First, a (LUAD-)WSC wishing to interact with some SP can use the Authentication Service at an Identity Provider to obtain security tokens. Next, the (LUAD-)WSC invokes the Single Sign-On Service at the Identity Provider in order to obtain an authentication assertion to convey to the SP, thus enabling Liberty-SSO-enabled, vanilla, web-based interactions with that SP.

For example, if a (LUAD-)WSC successfully authenticates with an Identity Provider (IdP) via the IdP's Authentication Service (Section 5), the IdP can ensure that the LUAD-WSC will have in its possession a `<disco:ResourceOffering>` and necessary credentials for the ID-WSF Single Sign-On Service at the very same IdP. Thus the (LUAD-)WSC may obtain an authentication assertion via the IdP's the latter Service.

Additionally, the IdP can, at the same time, ensure that the (LUAD-)WSC possesses a `<disco:ResourceOffering>` and necessary credentials for the Discovery Service (DS) of the Principal wielding the LUAD-WSC — thus enabling the LUAD-WSC to simultaneously utilize ID-FF- and ID-WSF-based services on behalf of the Principal, based on one sign-on interaction, from the Principal's perspective.

In yet another plausible scenario, some web service provider(s) might not be ID-WSF-based. Rather, they could be *generic Web Service Providers* (gWSPs).

A (LUAD-)WSC consuming services from gWSPs may need to obtain security tokens satisfying whichever security paradigm the gWSPs employ. It is plausible that such a paradigm will accommodate use of Liberty authentication assertions as security tokens — for example, see [wss-sms] and [wss-saml]. Note that the SSO Service can address this use case.

6.2. Service Type Declaration

The Service Type URI for the ID-WSF SSO Service is:

`urn:liberty:ssos:2004-04`

6.3. Rules for SSO Service Providers

SSO Service Providers (SSOS Providers) MUST adhere to the following rules:

1. Unless stated otherwise below the SSOS Provider SHOULD adhere to the [SAMLCore2] and [SAMLBind2] specifications.

2. The SSOS Provider SHOULD offer an ID-WSF Authentication Service, as defined in Section 5: Authentication Service.
   Upon successful authentication the SSOS Provider will respond to the SSOS Consumer with a `SASLResponse` as specified in Section 5. Returned `<disco:ResourceOffering>` elements referring to SSO Service instances MUST use the Service Type URI defined in Section 6.2 above.

3. The SSOS Provider SHOULD adhere to the SOAP binding as specified in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2] the [LibertySOAPBinding] shall take precedence.

4. SSOS Providers SHOULD advertise their SSO capability in metadata [LibertyMetadata]. To accomplish this, it MUST include a `<md:SingleSignOnProtocolProfile>` element in its metadata, with a value of:
   `urn:liberty:iff:profiles:id-wsf`

5. The SSOS Provider MAY, when it receives an `<saml:AuthnRequest>` that has its `ProtocolBinding` element set to `urn:liberty:iff:profiles:id-wsf` , respond with an ID-WSF message, containing relevant header blocks as specified in [LibertySOAPBinding].
   Note:
   SSOS Providers 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, which is likely to be ignored, the SSOS Provider may include a `<sec:ServiceSessionContext>` element in a `<sb:ServiceInstanceUpdate>` header block. The WSC that sent the original `<saml:AuthnRequest>` must of course understand these header blocks.

6. SSOS Providers SHOULD NOT respond 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 LECP profile [SAMLBind2] where an IdP is allowed to respond with any content that is acceptable to the requester (i.e. the LECP).

7. Upon successful processing of the `<saml:AuthnRequest>`, the SSOS Provider SHOULD respond with a SOAP-bound `<saml:Response>` message, constructed according to [SAMLCore2] in combination with [SAMLBind2].
   Note:
   This is different from the LECP profile [SAMLBind2] where an IdP is expected to respond with an `<saml:Response>`.

6.4. Rules for SSO Service Consumers

Consumers of the ID-WSF SSO Service MUST adhere to the following rules:

1. Unless stated otherwise below the WSC SHOULD adhere to the rules for active intermediaries as specified in [SAMLCore2] and [SAMLBind2].

2. The WSC SHOULD adhere to the SOAP binding as specified in [LibertySOAPBinding]; in case of conflict with the SOAP binding as specified in [SAMLBind2] the [LibertySOAPBinding] shall take precedence. For example, the WSC must include a proper `<sb:Correlation>` header block in its messages to the SSOS Provider.
   Note:
   The WSC MAY 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 an ID-WSF Authentication Service Provider.

3. The WSC MUST set the `ProtocolBinding` attribute of the `<saml:AuthnRequest>` to:
   `urn:liberty:iff:profiles:id-wsf`
   Note:
   Depending on the application and deployment model the WSC may have to construct the `<saml:AuthnRequest>` by itself, unlike LECP implementations that merely repack a `<saml:AuthnRequest>` message that was constructed by an SP. Obviously, a WSC will not be able to sign `<saml:AuthnRequest>` messages on behalf of the party that will consume the `<saml:Response>`. See the discussion in Section 6.1: Conceptual Model for context.

4. When the WSC receives security tokens, in the form of `<saml:Assertion>` elements or derivatives thereof, it MUST NOT send these to any other party than the intended audience, as indicated in the assertion's `<saml:Audience>` element.

7. 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 4.

```
<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:attribute name="id"
                         type="xs:ID"
                         use="optional"/>

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

Figure 4. 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 5. 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` [RFC2396]. 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.

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*

[LibertyBindProf] Cantor, Scott, Kemp, John, Champagne, Darryl, eds. "Liberty ID-FF Bindings and Profiles Specification," Version 1.2-errata-v2.0, Liberty Alliance Project (12 September 2004). *http://www.projectliberty.org/specs*

[LibertyClientProfiles] Aarts, Robert, eds. (26 April 2004). "Liberty ID-WSF Profiles for Liberty enabled User Agents and Devices," version 1.0, Liberty Alliance Project *http://www.projectliberty.org/specs/*

[LibertyDisco] Beatty, John, Hodges, Jeff, Sergent, Jonathan, eds. "Liberty ID-WSF Discovery Service Specification," Version 2.0-02, Liberty Alliance Project (24 Nov 2004). *http://www.projectliberty.org/specs*

[LibertyInteract] Aarts, Robert, eds. "Liberty ID-WSF Interaction Service Specification," Version 2.0-01, Liberty Alliance Project (22 November 2004). *http://www.projectliberty.org/specs*

[LibertyMetadata] Davis, Peter, eds. "Liberty Metadata Description and Discovery Specification," Version 2.0-02, Liberty Alliance Project (25 November 2004). *http://www.projectliberty.org/specs*

[LibertyPAOS] Aarts, Robert, eds. "Liberty Reverse HTTP Binding for SOAP Specification," Version 2.0-01, Liberty Alliance Project (22 November 2004). *http://www.projectliberty.org/specs*

[LibertyProtSchema] Cantor, Scott, Kemp, John, eds. "Liberty ID-FF Protocols and Schema Specification," Version 1.2-errata-v2.0, Liberty Alliance Project (12 September 2004). *http://www.projectliberty.org/specs*

[LibertySecMech] Ellison, Gary, eds. "Liberty ID-WSF Security Mechanisms," Version 1.1, Liberty Alliance Project (18 April 2004). *http://www.projectliberty.org/specs*

[LibertyGlossary] Hodges, Jeff, eds. "Liberty Technical Glossary," Version 1.3-errata-v1.0, Liberty Alliance Project (12 Aug 2004). *http://www.projectliberty.org/specs*

[LibertySOAPBinding] Hodges, Jeff, Kemp, John, Aarts, Robert, eds. " Liberty ID-WSF SOAP Binding Specification ," Version 2.0-01, Liberty Alliance Project (22 November 2004). *http://www.projectliberty.org/specs*

[RFC2119] Bradner, S., eds. "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 [March 1997]*.

[RFC2222] "Simple Authentication and Security Layer (SASL)," John G. Myers (October 1997). RFC 2222, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2222.txt [October 1997]*.

[RFC2246] Dierks, T., Allen, C., eds. (January 1999). "The TLS Protocol," Version 1.0 RFC 2246, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2246.txt [January 1999]*.

[RFC2396] Berners-Lee, T., Fielding, R., Masinter, L., eds. (August 1998). "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2396.txt [August 1998]*.

[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 [June 1999]*.

[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 [January 2001]*.

[RFC3546] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., Wright, T., eds. (June 2003). "Transport Layer Security (TLS) Extensions," RFC 3546, The Internet Engineering Task Force *http://www.ietf.org/rfc/rfc3546.txt [June 2003]*.

[SAMLGloss] Hodges, Jeff, Maler, Eve, eds. (05 November 2002). "Glossary for the OASIS Security Assertion Markup Language (SAML)," Version 1.0, OASIS Standard, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/security/#documents*

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

[Schema1] Thompson, Henry S., Beech, David, Maloney, Murray, Mendelsohn, Noah, eds. (May 2002). "XML Schema Part 1: Structures," 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]*

[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] Kellomaki, Sampo, eds. "Liberty Identity Personal Profile Service Specification," Version 1.0, Liberty Alliance Project (12 November 2003). *http://www.projectliberty.org/specs*

[LibertyIDWSFOverview] Tourzan, Jonathan, Koga, Yuzo, eds. "Liberty ID-WSF Web Services Framework Overview," Version 1.0-errata-v1.0, Liberty Alliance Project (12 September 2004). *http://www.projectliberty.org/specs*

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

[RFC2245] "Anonymous SASL Mechanism," C. Newman (November 1997). RFC 2245, Internet Engineering Task Force *http://www.ietf.org/rfc/rfc2245.txt [November 1997]*.

[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 [February 1998]*.

[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 [October 1998]*.

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

[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 [August 2001]*.

[SAMLBind2] Hughes, John, Cantor, Scott, Mishra, Prateek, Hirsch, Frederick, Philpott, Rob, Hodges, Jeff, Maler, Eve, eds. (18 August 2004). "Bindings and Profiles for the OASIS Security Assertion Markup Language

(SAML) V2.0," OASIS Committee Specification, version 2.0, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/documents.php?wg_abbrev=security*

[SAMLCore11] Maler, Eve, Mishra, Prateek, Philpott, Rob, eds. (27 May 2003). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1," OASIS Committee Specification, version 1.1, Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/documents.php?wg_abbrev=security*

[SAMLCore2] Cantor, Scott, Kemp, John, Philpott, Rob, Maler, Eve, eds. (24 September 2004). "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," OASIS Committee Draft 02, v2.0 , Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/security/*

[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 Proposed 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] "Authentication for Distributed Systems," Thomas Y. C. Woo Simon S. Lam (January, 1992). IEEE *http://citeseer.nj.nec.com/woo92authentication.html*

[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-saml] Hallam-Baker, Phillip, Kaler, Chris, Monzillo, Ronald, Nadalin, Anthony, eds. (May 5, 2003). Organization for the Advancement of Structured Information Standards *http://www.oasis-open.org/committees/download.php/1911/WSS-SAML-07.pdf* "Web Services Security: SAML Token Profile," Draft WSS-SAML-07.pdf,

[XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C.M., Maler, Eve, eds. (Oct 2000). "Extensible Markup Language (XML) 1.0 (Second Edition)," Recommendation, World Wide Web Consortium *http://www.w3.org/TR/2000/REC-xml-20001006*

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 the a fresh copy of this file from [IANA].

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

(last updated 2004-01-21)

The Simple Authentication and Security Layer (SASL) [RFC2222] 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 given in the
section "Registration procedures" of RFC2222.

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

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

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

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

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

ANONYMOUS         COMMON   [RFC2245]   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>


References
----------

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

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

[RFC2245]  Newman, C., "Anonymous SASL Mechanism", RFC 2245, Innosoft,
        November 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.



People
------

[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. lib-arch-authn-svc.xsd Schema Listing

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

<xs:schema
   targetNamespace="urn:liberty:sa:2004-12"
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:sa="urn:liberty:sa:2004-12"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:disco="urn:liberty:disco:2004-12"
   xmlns:saml="urn:oasis:names:tc:SAML:2.0:protocol"
   xmlns="urn:liberty:sa:2004-12"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="07"
   >

<!-- Filename: lib-arch-authn-svc.xsd  -->
<!-- $Id: lib-arch-authn-svc.xsd,v 1.4.6.2 2004/11/28 03:48:29 dchampagne Exp $ -->
<!-- Author: Jeff Hodges -->
<!-- Last editor: $Author: dchampagne $ -->
<!-- $Date: 2004/11/28 03:48:29 $ -->
<!-- $Revision: 1.4.6.2 $ -->

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

  <xs:import
     namespace="urn:liberty:disco:2004-12"
     schemaLocation="liberty-idwsf-disco-svc-v2.0.xsd"/>

  <xs:include schemaLocation="liberty-idwsf-utility-v2.0.xsd"/>

  <xs:annotation>
     <xs:documentation>
        Liberty ID-WSF Authentication Service XSD
     </xs:documentation>
     <xs:documentation>
        The source code in this XSD file was excerpted verbatim from:

        Liberty ID-WSF Authentication Service Specification
        Version 2.0-02
        25 Nov 2004

        Copyright (c) 2003, 2004 Liberty Alliance participants,
        see http://www.projectliberty.org/specs/idwsf_copyrights.html
     </xs:documentation>
  </xs:annotation>


  <!-- 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="saml:RequestedAuthnContext"
```

```
                  minOccurs="0"/>

      </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:attribute name="id"
                type="xs:ID"
                use="optional"/>

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

<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>

         <xs:element ref="disco:ResourceOffering"
                  minOccurs="0"
                  maxOccurs="unbounded"/>

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

       </xs:sequence>

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

       <xs:attribute name="id"
                 type="xs:ID"
                 use="optional"/>

   </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:attribute name="id"
                         type="xs:ID"
                         use="optional"/>

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



</xs:schema>
```

D. lib-arch-iwsf-utility.xsd Schema Listing

```
    <?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   version="2.0-01">
   <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 file intended for inclusion, rather than importation, into other schemas.
   This version: 2004-12

         Copyright (c) 2004 Liberty Alliance participants, see
         http://www.projectliberty.org/specs/idwsf_2_0_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: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="xs:NCName" 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="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>
```