HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Laboratory of Software Technology

Yrjö Kari-Koskinen

Implementing a Web Service Identity Provider in a Production Environment

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, 21st May 2007

Supervisor: Professor Jorma Tarhio
Instructors: Petteri Stenius and M.Sc. Tomi Määttänen

Author:	Yrjö Kari-Koskinen
Department:	Computer Science and Engineering
Major:	Software Systems
Minor:	Telecommunications Software

Title of the thesis:	Implementing a Web Service Identity Provider in a Production Environment
Number of pages:	x + 64
Date:	21st May 2007
Professorship:	T-106 Software Technology
Supervisor:	Professor Jorma Tarhio
Instructors:	Petteri Stenius and M.Sc. Tomi Määttänen

In this thesis, the implementation of a Web Service Identity Provider (WSIDP) is presented. Different models for integrating the WSIDP in product environment are discussed as well.

The term Web services means the standardized techniques, such as XML, HTTP, and SOAP, which are used for implementing distributed applications. These techniques enable programming language and platform independence. Furthermore, the Web services infrastructure is very well supported in diverse programming environments.

This thesis presents the implementation of WSIDP as an extension to Ubilogin, a single sign-on solution for the WWW. The software implements applicable parts of SAML 2.0 and ID-WSF 2.0 specifications using Java programming language. Some other specifications, such as WS-Federation and WS-Security, are discussed as well.

The implemented specifications pose different requirements for the Web Service Consumer (WSC) and Web Service Provider (WSP) applications. This thesis presents four alternative methods for integrating the applications with WSIDP and discusses their effects.

Keywords:	Web services, Security, Authentication, SAML, ID-WSF, SOAP, WS-Federation, WS-Security, Single Sign-On

Tekijä:	Yrjö Kari-Koskinen
Osasto:	Tietotekniikka
Pääaine:	Ohjelmistojärjestelmät
Sivuaine:	Tietoliikenneohjelmistot

Työn nimi:	Web service -kirjautumispalvelimen toteuttaminen tuotantoympäristöön
Sivumäärä:	x + 64
Päiväys:	21. toukokuuta 2007

Professuuri:	T-106 Ohjelmistotekniikka
Työn valvoja:	Professori Jorma Tarhio
Työn ohjaajat:	Petteri Stenius ja DI Tomi Määttänen

Työssä kuvataan web service -kirjautumispalvelimen (WSIDP) toteuttaminen ja selvitetään erilaisia tapoja integroida palvelin osaksi tuotantojärjestelmää.

Web serviceillä tarkoitetaan hajautettujen sovellusten toteuttamista hyödyntäen standardoituja tekniikoita, kuten XML, HTTP ja SOAP. Nämä tekniikat mahdollistavat ohjelmointikieli- ja alustariippumattomien sovellusten rakentamisen. Web services -infrastruktuurille löytyy myös hyvin laaja tuki erilaisista ohjelmointiympäristöistä.

Tässä työssä esitellään WSIDP-ohjelmiston toteuttaminen osaksi olemassa olevaa WWW-ympäristön kirjautumispalvelinta (IDP), Ubiloginia. Ohjelmisto implementoidaan SAML 2.0- ja ID-WSF 2.0 -määrittelyjen pohjalta Java-ohjelmointikielellä. Työssä perehdytään myös muihin aihetta sivuaviin spesifikaatioihin, kuten WS-Federation ja WS-Security.

Toteutetut määritykset asettavat omat vaatimuksensa sille, minkälaisia rajapintoja asiakasohjelmiston (WSC) ja sovellusohjelmiston (WSP) kuuluu toteuttaa. Työssä esitellään neljä vaihtoehtoista tapaa WSC:n ja WSP:n toteuttamiseksi, sekä näiden vaihtoehtoisten integrointitapojen vaikutuksia.

Avainsanat:	Web services, tietoturva, tunnistaminen, SAML, ID-WSF, SOAP, WS-Federation, WS-Security, kertakirjautuminen

Contents

List of Figures

Abbreviations and Notations

ADFS	Microsoft Active Directory Federation Service
API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
CRAM-MD5	Challenge-Response Authentication Mechanism based on the HMAC-MD5 algorithm
DCOM	Distributed Component Object Model
DOM	Document Object Model
DoS	Denial of Service
DTD	Document Type Declaration, an XML and SGML schema language
ECP	Enhanced Client or Proxy, a SAML profile and another name for a WSC
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over SSL/TLS, Secure HTTP
IANA	Internet Assigned Numbers Authority
ID-FF	Identity Federation Framework
ID-WSF	Identity Web Services Framework
ID-*	Liberty identification specification suite: ID-WSF, ID-FF, and ID-SIS
IDP	Identity Provider

J2EE	Java 2 Platform, Enterprise Edition; The former name of the Sun's Enterprise Java Platform, today called the *Java EE*
JAX-WS	Java API for XML Web Services
JAXB	Java Architecture for XML Binding
JAXP	Java API for XML Processing
JDOM	A library for representing XML documents in Java
JWSDP	Java Web Services Development Pack
LDAP	Lightweight Directory Access Protocol
LUAD	Liberty-enabled User Agent or Device, another name for a WSC
MIME	Multipurpose Internet Mail Extensions
MITM	Man-in-the-Middle, an active wiretapping attacker
NTLM	Windows NT LAN Manager, the network authentication protocol on Windows NT
NTP	Network Time Protocol
OASIS	Organization for the Advancement of Structured Information Standards
OTP	One-Time Password
PAOS	Reverse HTTP binding for SOAP, where a SOAP request is bound to a HTTP response and vice versa
PKI	Public Key Infrastructure
Principal	A process, person, or group of persons whose identity can be authenticated
REST	Representational State Transfer, collection of artchitectural principles for distributed hypermedia systems
RMI	Remove Method Invocation
RPC	Remote Procedure Call
SAAJ	SOAP with Attachments API for Java

SAML	Security Assertion Markup Language, an XML-standard for exchanging authentication and authorisation data between security domains
SASL	Simple Authentication and Security Layer, a framework for authentication and authorization in Internet protocols
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SOA	Service-oriented architecture, concept of design principles for loosely coupled services in a network
SOAP	Originally: Simple Object Access Protocol, as of SOAP V1.2 not an acronym anymore
SP	Service Provider
SMTP	Simple Mail Transfer Protocol
SSL	Secure Sockets Layer, obsoleted by TLS
SSO	Single Sign-On
TC	Technical Committee
TLS	Transport Layer Security
UAS	Ubilogin Authentication Server
UDDI	Universal Description, Discovery, and Integration
URI	Uniform Resource Identifier, such as URL or URN
URL	Uniform Resource Locator
UWA	Ubilogin Web Agent
W3C	World-Wide Web Consortium
WAR	Web application archive
WCF	Microsoft Windows Communication Foundation
WebSSO	Web Browser Single Sign-On, also a SAML profile
WS	Web Services

WS-I	Web Services Interoperability Organization
WS-*	Set of Web Services specifications by Microsoft, IBM, et al. such as WS-Security, WS-Trust, WS-SecureConversation, WS-Federation, WS-Policy, and WS-MetadataExchange
WS-F ARP	Web Services Federation, Active Requestor Profile
WS-F PRP	Web Services Federation, Passive Requestor Profile
WSIDP	Web Service Identity Provider
WSIT	Web Services Interoperability Technology
WSC	Web Service Consumer
WSDL	Web Services Definition Language
WSP	Web Service Provider
WSS	Web Services Security
WWW	World-Wide Web
X.509	ITU-T standard for public key infrastructure (PKI) including the notable public key certificate specification
XJC	Java Architecture for XML Binding Compiler
XKMS	XML Key Management Specification
XML	Extensible Markup Language
XMLDSig	XML Digital Signature
XOM	XML Object Model
XSD	XML Schema Definition

Chapter 1

Introduction

In the history of traditional client-server applications, the Web technologies, such as HTTP and HTML, together with browser-based user interfaces enabled a whole range of new kind of applications. These applications could be used worldwide through the Internet or restricted in an intranet. The Web browsers eliminated the need for individual client applications and offered total operating system, platform, and vendor independence.

Building on the success of the Web, the computing industry set in motion design of new methods for interoperable distributed computing while trying to learn from the past solutions: DCOM, CORBA, RPC, and Java RMI. The new methods arise from the idea of using XML, standardized in 1998, for communication in distributed applications. Gradually the ideas of major industry players—such as Microsoft, IBM, Sun and Oracle—converged to creating the SOAP, XSD, and WSDL standards that would later be coined as *Web services* [Lev01].

According to Haas, Web services aim at vendor, platform, and language independence. Most importantly, this means interoperability: the customer is not tied to a particular vendor as in the previous efforts. [Haa03, pages 8–9]

In the WWW, various authentication solutions have been defined to facilitate e.g. business transactions and corporate services in intranets and extranets. Many of the solutions—such as Microsoft Passport or Ubilogin Single Sign-On presented by Nykänen in [Nyk02]—have reached a mature state. These solutions are, however, proprietary, vendor-specific, and thus making way for open, XML-based protocols like SAML and WS-Federation.

In addition to profiling authentication in the WWW—also known as WebSSO, the standards and specifications for authentication and security define profiles for Web services too. Implementing Web services authentication interface in an identity provider facilitates the developing of distributed applications using Web services. Furthermore, using standardized authentication protocols prepares the way for interoperability and complex use cases such as federation and delegation.

Research Goals In this thesis, the implementation of a Web service identity provider—Ubilogin WSIDP—is documented. WSIDP implements protocols from the SAML 2.0 [SAML-core] and ID-WSF 2.0 [ID-WSF-authn-draft] specifications and extends the Ubilogin SSO solution described in [Nyko2].

The two goals of the thesis are: to analyze the problems regarding the implementation of these specifications with Java and to explore the different possibilities in integrating WSIDP as a part of a production environment. The latter focuses on the requirements WSIDP poses to the other communication counterparts (i.e. WSC and WSP).

Organization The background to the subject, available tools, the XML security specifications, and Ubilogin Single Sign-On are presented in the next chapter. Next, in chapter 3, the chosen protocols are described more carefully and the major challenges of the implementation are discussed while also considering security aspects.

Chapter 4 describes the major shortcomings of SAML and ID-WSF specifications that were noticed in the course of the implementation. In chapter 5, problems of the three-way authentication procedure are discussed. Later, four additional deployment scenarios are presented along with their benefits, security concerns, and requirements for the WSC and WSP.

Next, in chapter 6, the suitability of the chosen specifications for the given goal is discussed based on the findings of the thesis. A case example of the real-life use of WSIDP is presented to reflect a final view on the integration procedures.

Finally, in chapter 7, the present situation of identification in Web services is analyzed and a view to the future concludes this thesis.

Chapter 2

Background

In this chapter, a background to the subject is presented: XML in a nutshell, what are Web services, what is the role of authentication and authorization in Web services as opposed to WebSSO, the most important XML security specifications and what the Ubilogin Single Sign-On software is.

2.1 Extensible Markup Language

Extensible Markup Language (XML) was designed by W3C as an easy-to-use subset of Standard Generalized Markup Language (SGML) in 1998. Its design goals included the ease to write programs for processing XML documents in the Web whilst keeping the documents "human-legible and reasonably clear" [XML, section 1.1].

The XML specification [XML] defines the format of elements, their attributes, and character data that constitute an XML document. Additionally, the well-formedness of documents is specified. The character data in the documents is defined in terms of the Unicode character set. In 1999, Namespaces in XML was specified to provide a method to distinguish different markup vocabularies in a single document.

The grammar for a class of documents was originally defined by DTD, but later on W3C XML Schema was developed to extend the capabilities of DTD. Lately, few notable new schema languages have been proposed, namely RELAX NG and Schematron. However, XML Schema seems to be most popular of the schema languages for data binding—perhaps due to being a W3C recommendation. All the XML specifications mentioned in this thesis use an XML Schema Definition (XSD) as the normative schema.

Together the XSDs and namespaces enable straightforward combination of several XML document types. The schema can be used to constrain which "datatypes,

elements and their content and attributes and their values" are permitted in a document [XML-schema, section 1.1]. Additionally, the schema specifies normalization and default values. These characteristics facilitate the programmatic processing of XML documents such as constraint checking.

Another important—and elemental—extension to XML is the XML Digital Signature (XMLDSig), especially in the field of data security. XMLDSig can be used to sign just a specific portion of an XML tree with the signature either enveloped inside the signed tree, enveloping the tree inside the signature, detached as a separate data object, or possibly as a sibling part of the tree [XMLDSig, section 2.0].

The signatures "add authentication, data integrity, and support for non-repudiation to the data that they sign" [XMLDSig-intro]. However, the signature specification does not specify how to associate the keys with the signing party and so the trust relationships must be carefully considered by the implementer [XMLDSig, section 1.0].

XML Processing Today, XML tools are ubiquitous in modern programming environments. The Java programming language offers the two most popular methods for XML handling in the built-in Java API for XML Processing (JAXP): Document Object Model (DOM) and Simple API for XML (SAX). DOM is a tree-based API which parses the whole XML document as a tree and offers in-memory manipulation. It is simple, but also criticized for having a quite ugly Java implementation and not being object oriented. This is due to DOM's history as a generic interface for several scripting languages and different browser vendors. According to Harold, another drawback of DOM is that it allows "creation of malformed documents" [Har03, slide 9].

SAX was originally designed just for Java but is today available for probably every programming language with an XML parser. Harold classifies it as an event based push API: it parses XML as a stream and sends events upon finding new nodes (e.g. elements, text, or attributes) [Har03, slide 3]. It is effective and well suited for Java, but Web services seldom fit into streamable parsing.

Java Architecture for XML Binding (JAXB) is a very viable tool for XML processing in Java if the XSDs are available for the documents in question. The JAXB tools generate Java source code for each element and produce an object tree specific to the schemata. The JAXB elements can be *marshalled* to DOM trees and *unmarshalled* back to JAXB. This allows handling a part of the tree in e.g. DOM. All in all, JAXB offers an elegant object-oriented solution and impose content validation per XML schemata. However, Harold illustrates various problems that arise from the use of XML schemata—should there be recursive content or if the order of elements is significant [Har03, slide 7].

DOM, SAX, and JAXB are de facto examples of the tree, event based push, and

data binding APIs for XML. Harold presents XML Object Model (XOM) as a better tree based API in [Har03] and discusses the differences between XOM, JDOM, and DOM. He also lists the aforementioned API styles along with the event based pull API. Also Ojala discusses the major viewpoints regarding the different API styles in [Oja05, section 2.5].

2.2 Web Services

According to W3C's Web Service Architecture, "a Web service is a software system designed to support interoperable machine-to-machine interaction over a network" [ws-arch, section 1.4]. The architecture underlines that Web services are most appropriate for distributed systems where the components operate over Internet and on different platforms [ws-arch, section 3.1.2]. The document suggests that XML, SOAP, and WSDL technologies be used in such circumstances to achieve the Web services' goals such as interoperability, extensibility, and manageability.

SOAP is a "lightweight protocol for exchanging structured information in a decentralized, distributed environment" [SOAP-part1, chapter 1]. The W3C's SOAP specification standardizes an extensible messaging framework and processing model using XML technologies. SOAP is specified to be independent of the underlying transport protocol, but is practically always transmitted over HTTP as specified in the SOAP HTTP Binding [SOAP-part2, chapter 7]. Although SOAP is not the only way to implement Web services—nor the only device for sending XML over HTTP—it has a wide and working installation base in different programming environments.

The SOAP messages are realized as an envelope that contains a mandatory *body* and an optional *header* block. The SOAP body contains the actual data payload, or protocol message, that is to be sent. The header contains everything that is not part of the payload. More specifically, different header blocks are defined as extensions to SOAP that provide e.g. message correlation, reliability, and security. It should be noted, that the header blocks can be addressed to a certain SOAP intermediary. This way, the header elements can be utilized to achieve an end-to-end context whereas the transport protocol provides signaling only for a single hop (e.g. in HTTP headers). Additionally, a structure and processing model for SOAP faults are specified for carrying error information.

Today SOAP is seen as a viable communication framework for new client-server applications—especially for such applications which can not be implemented as Web applications. The most recent version of SOAP specification, SOAP 1.2 [SOAP-part1], was defined already in 2003. The technological ideas around it are quite mature, but the software tools and SOAP implementations are still ongoing heavy development. The maturity of SOAP implementations (i.e. SOAP stacks) vary

largely on different platforms (e.g. Microsoft .NET, Sun Java and various application servers, such as Bea WebLogic). In fact, all Web services specifications covered in this thesis are defined on top of SOAP 1.1—as are also most of the contemporary industry specifications.

Web Services Definition Language (WSDL) provides a standardized tool for specifying the interfaces of a Web service. WSDL 1.1 was specified in 2001 and uses XML Schema "as its canonical type system" [WSDL, chapter 1]. The specification separates the used type information, message data, operations, and bindings from the Web service interface. The specified bindings are SOAP 1.1, HTTP, and MIME [WSDL]. As of writing, WSDL 2.0 is still not finalized as a W3C standard.

WSDL is typically used for generating the source code for a client or server counterpart of a specified Web service interface. For some Web services specifications, the WSDL documents are distributed along with the XSDs and the normative specification document.

Ojala further discusses SOAP usage and its relation to concepts such as Service-oriented architecture (SOA) and Representational State Transfer (REST) in [Oja05, chapter 3].

In several specifications, the classic use case examples of SOAP include querying stock quotes and making travel reservations. A more concrete example is the ApiTaMo service of the Finnish National Board of Taxes presented in section 6.3. ApiTaMo provides checking, fixing and sending of e.g. tax return forms. As the protocol is specified on top of SOAP, it facilitates implementing the client software in different kind of environments and applications.

2.3 Websso and Web Services Authentication

With Web Browser Single Sign-On (Websso) we mean the authentication of Web application users in an Identity Provider (IDP). The IDP provides a centralized solution for authentication and authorization, usually for more than one Web application. Websso is designed as a series of HTTP POST and GET requests with client and server redirects transferring the authentication protocol messages.

This thesis focuses on authentication and authorization in Web services and SOAP as opposed to Websso. Solutions for Websso, such as Ubilogin Single Sign-On (SSO) or Microsoft Passport, have reached a mature state.

Figure 2.1 presents three different use cases for Websso and Web services authentication. In the first case, user A wishes to use the Web application at the service provider SP-a. The identity provider IDP authenticates the users of this application. User A accesses both the IDP and SP-a with his browser.

In use case two, the service provider SP-a acts also as a Web service consumer, WSC-1. To provide the requested information in the Web application, WSC-1 needs

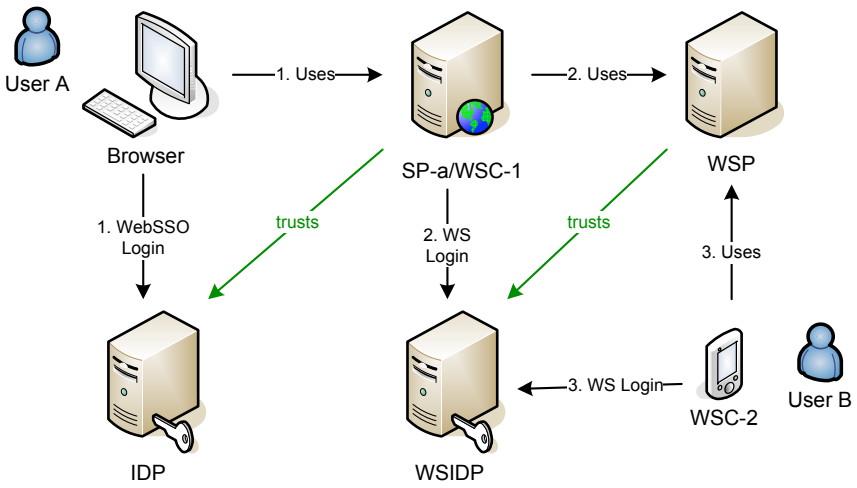

Figure 2.1: Websso and ws authentication use cases, see [Liberty-overview, page 8]

to request information from a Web service provider, WSP. Here, WSP uses the Web service identity provider, WSIDP for identifying WSCs. Thus, WSC-1 needs to log in to WSIDP, possibly using user A's identity.

In the third use case, another user (user B), has an intelligent mobile device wielding a Web service consumer software, WSC-2. WSC-2 needs to log in to WSIDP to request information from WSP.

In the first use case, the user is authenticated through a Websso interface. Both the second and third case employ ws authentication. Both of the identity providers, IDP and WSIDP, could be implemented in a single application and share the same user database. This way, a single sign-on session could be provided between the logins 1 and 2. The user A and both WSCs could also use a SSO session to access other service providers than the ones depicted in the figure.

The major difference in Websso and ws authentication is the interface provided by the identity provider. In Websso, the IDP uses HTML forms to provide a user interface that is not standardized. In Web services, a well-defined—or even standardized—protocol is used for delivering the authentication data over SOAP.

Furthermore, a WSC generates the authentication protocol messages by itself. It also provides the login user interface which is not facilitated by WSIDP. In some situations, e.g. in WSC-1 of use case two in figure 2.1, no login user interface is needed at all.

The need for a Web services authentication interface, WSIDP, rises from the popularity of Web services in new distributed applications. Providing such an in-

terface facilitates the developing of Web service provider and consumer applications. Use cases of ws authentication are further discussed in section 3.1.

2.4 Web Services Authentication Protocols

Features such as reliability and security have been intentionally left out from the SOAP specification to keep the protocol simple and extensible [SOAP-part1, chapter 1]. This implies an evident need for specifying authentication and authorization extensions to SOAP. Specifications and standards for this kind of extensions have been proposed by many different bodies such as the Liberty Alliance Project[1], OASIS[2], Shibboleth[3], and WS-I[4].

Liberty and OASIS are open standards consortiums which have worked together on some standards. Furthermore, Liberty organizes conformance and interoperability testing events through which it certifies interoperable implementations.

WS-I promotes interoperability and is backed by big industry companies like Microsoft and IBM which have designed the WS-* specifications. Shibboleth is an open source software project of Internet2.

Prior to the standardized SOAP authentication protocols, presented below in sections 2.4.1 through 2.4.3, diverse custom solutions were used. E.g. HTTP and HTTPS offer standard procedures such as basic, digest, and client certificate authentication that were used in client toolkits such as the Microsoft SOAP Toolkit 2.0 [Gav01]. Also proprietary solutions such as Microsoft's NTLM and including the credentials in application-specific SOAP headers have been proposed [Rei02].

All these solutions lack the extensibility to arbitrary authentication methods and means for providing well-specified identity information such as different name identifier formats and user attributes. In addition, the proprietary solutions are obviously not meant to be interoperable with other vendors.

2.4.1 Security Assertion Markup Language

One of the most important XML security specifications is SAML. SAML offers both an XML-based language and a protocol for authoring and processing "assertions made about a subject by a system entity" [SAML-core, chapter 1]. The WSIDP implementation documented in this thesis is based on SAML 2.0.

The first SAML version, 1.0, was completed in May 2002 and certified by the OASIS Security Services Technical Committee (SSTC) as a standard in November 2002.

[1] http://www.projectliberty.org
[2] http://www.oasis-open.org
[3] http://shibboleth.internet2.edu/
[4] http://www.ws-i.org/

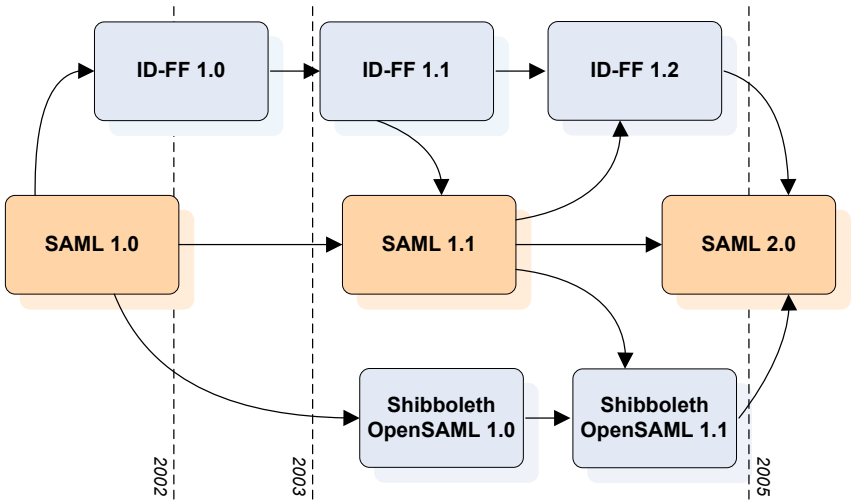

Figure 2.2: SAML family tree [Liberty-overview, page 4], see also figures 2 and 3 in [Malo6]

Liberty designed its Identity Federation Framework (ID-FF) 1.0 specification as an extension to SAML 1.0. These specifications evolved to their next versions, ID-FF 1.2 and SAML 1.1, offering new features to each other. Later on, "Liberty submitted its ID-FF 1.2 as input to SAML 2.0" [Liberty-tech, page 20]. Also the Shibboleth OpenSAML 1.1 implementation offered some enhancements to the SAML 2.0 specification. SAML 2.0 was completed on January 2005 and today Liberty has discontinued the development of ID-FF in favor of SAML. [Liberty-tech]

The SAML specification is documented through different profiles and bindings: the profiles describe typical SAML message flows as use cases which can be implemented using specific protocol bindings. Regarding SAML, this thesis concentrates on the ECP SSO profile which can be implemented using SOAP and Reverse HTTP binding for SOAP (PAOS). The WebSSO profile implements the equivalent SAML message flows using HTTP redirect, POST and artifact bindings. [SAML-conformance, chapter 2]

2.4.2 Web Services Languages

A coalition of companies such as Microsoft, IBM, RSA Security, and VeriSign has specified a set of SOAP extensions usually referenced as the WS-* (WS-Star). Among these are the WS-Trust and WS-Federation languages which form the basic building blocks of the Microsoft Identity Metasystem.

The Windows CardSpace framework builds on WS-Trust, WS-Security, and WS-SecurityPolicy specifications [InfoCard]. The WS-Federation Active and Passive Requestor Profiles (WS-F ARP and WS-F PRP) specify very similar features as SAML ECP and WebSSO profiles, respectively. WS-F PRP is the protocol used in Microsoft Active Directory Federation Service (ADFS) which can be seen as the de facto implementation of WS-F PRP [ADFS, page 7].

The most important WS-* specification in the context of this thesis is the Web Services Security: SOAP Message Security [WSS]. It provides mechanisms for message integrity and confidentiality along with an ability to send security tokens—such as SAML assertions—as a part of a message [WSS-SAML]. Web Services Security (WSS) version 1.1 was approved as an OASIS standard in February 2006 [WSS].

2.4.3 Liberty Alliance Identity Web Services Framework (ID-WSF)

In November 2004, Liberty Alliance published the first draft release of the ID-WSF authentication service and single sign-on specification, version 2.0 [ID-WSF-authn-draft]. This version was implemented in Ubilogin WSIDP. Later on, in October 2006, the final version of ID-WSF 2.0 [ID-WSF-authn] was published. This thesis refers to the final protocol version on other than implementation-specific issues.

The specification offers general identity authentication as per [WL92] and uses SAML 2.0 assertions as security tokens. The authentication protocol is based on Simple Authentication and Security Layer [SASL]. All the ID-WSF specifications are defined to be used with SOAP binding. [ID-WSF-authn]

There are numerous other Liberty Alliance specifications for e.g. exchanging user information and providing a discovery service. The Liberty specification set is referenced as ID-* (ID-Star).

2.5 Ubilogin Single Sign-On

Nykänen describes Ubilogin as a secure single sign-on solution for the WWW [Nyko2, chapter 1]. It is developed at Ubisecure Solutions Oy. Ubilogin Authentication Server (UAS) provides similar features—but not all of them—as the IDP of the SAML 2.0 WebSSO profile using a proprietary Ubilogin ticket protocol [SAML-conformance, chapter 2]. UAS authenticates users, implements access control, provides role and group based authorization, and includes a wide range of authentication mechanisms such as Windows authentication, bank authentication, mobile authentication methods, and authentication from external LDAP or Active Directory.

In addition to UAS, Ubilogin consists of an administrative user interface through which the Ubilogin users, groups, authentication methods, Web Agents (UWAS), access control to the UWAS, and authorization policies are controlled.

Ubilogin is written in Java programming language using the Java Servlet 2.3 interface and is compatible with the common application servers such as BEA WebLogic, Apache Tomcat, and Caucho Resin. Ubilogin uses a standard LDAP directory as its primary data storage. A wide variety of Ubilogin Web Agents (UWAS) have been implemented as Java servlet filters and similar filters for other platforms for integrating Ubilogin with different Web application environments.

Ubilogin Ticket Protocol Ubilogin Ticket Protocol was designed for secure and flexible Web browser SSO and easy deployment. The protocol relies on a trust relationship between UAS and the UWA which is established by creating a shared secret in a set-up phase. [Nyk02, section 3.2]

Figure 2.3 depicts the login message flow of the Ubilogin ticket protocol:

1. The user requests a resource in the Web application.
2. The UWA blocks the request because it does not contain a valid *application ticket* in the HTTP cookie. The UWA generates an *application ticket request* and redirects the request to UAS.
3. The browser forwards the ticket request to UAS.
4. UAS validates the ticket request and starts the authentication process. The authentication process consists of one or more HTML forms and respective HTTP POSTs where the user may get to choose the authentication method and present his credentials.

   SSO: If the request in the step three contained a valid *master ticket* in the HTTP cookie, then user has an existing session with UAS. In this case, the authentication process is skipped.
5. When the user has been authenticated UAS generates an *application ticket response*, sets the *master ticket* to the cookie, and redirects the browser back to the Web application.
6. The browser repeats the initial request from step one accompanied with the ticket response.
7. The UWA validates the ticket response, sets the *application ticket* to the cookie, and redirects the browser to the resource which was requested in the first step.

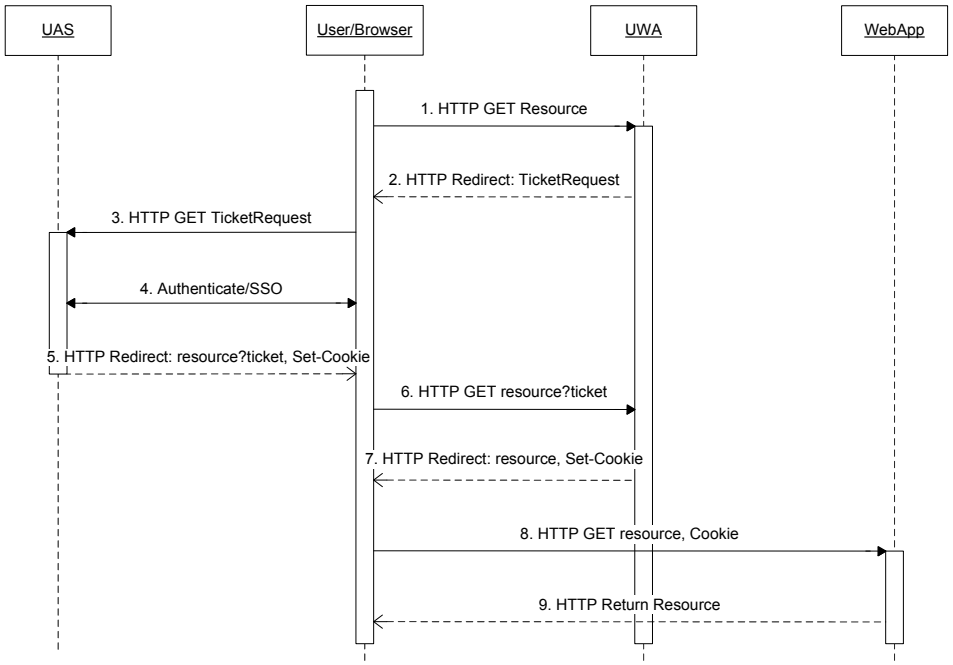

Figure 2.3: Login message flow in the Ubilogin ticket protocol [Nyko2, section 3.4]

8. The browser requests for the resource. The request contains the application ticket in a cookie. The UWA checks the validity of the ticket and allows the request to reach the Web application.

9. The Web application responds with the requested resource.

If the user has a session with UAS, i.e. the master ticket stored in a cookie, the whole authentication procedure is transparent to the user and no interaction is needed because of the HTTP redirects.

Both the ticket request and the response are encrypted using the ticket encryption key which is derived from the shared secret. The encryption provides confidentiality and also message integrity through digests. [Nyko2, page 36]

The most important contents of the application ticket for the Web application are the username, used authentication method, and the optional authentication method attributes.

In UAS version 4.0, also SAML 2.0 and WS-Federation protocols can be used for application ticket requests and responses in addition to Ubilogin ticket protocol. The master ticket is specific to the UAS implementation but not defined in any protocol. The user identity is saved in the master ticket.

Chapter 3

Implementing a Web Service Identity Provider

In this chapter, the actual implementation of a Web service identity provider, the Ubilogin WSIDP, is presented. The WSIDP implements the ECP profile of SAML 2.0 and acts as an ID-WSF 2.0 authentication service. The initial implementation supports two different authentication mechanisms: PLAIN and KATSO. The PLAIN mechanism uses username and password credentials. The KATSO mechanism uses one-time passwords (OTPS) in addition to username and password. The KATSO SASL mechanism was specified (see appendix A) during the Ubilogin WSIDP implementation.

In the first section, we present the use cases and goals which the implementation should solve. Then, we justify the choice of the SAML and ID-WSF specifications to be implemented and present the message flow that was derived from the two specifications. Next, in section 3.3, we delve into the actual implementation techniques and problems encountered during implementation.

The third chapter is concluded by first considering the security matters of our WSIDP implementation and lastly by presenting the challenges of SSO sessions in Web services.

3.1 Goals and Use Cases

The primary goal of the WSIDP implementation is to offer a standards-based login interface for fat clients[1] in Ubilogin as an alternative to the traditional Websso login interface which uses the Ubilogin ticket protocol. The WSIDP should integrate seamlessly into the existing Ubilogin implementation by using the authen-

[1] With fat clients we mean client applications of services which are not used by a Web browser, although the fat client can be implemented as a browser plugin.

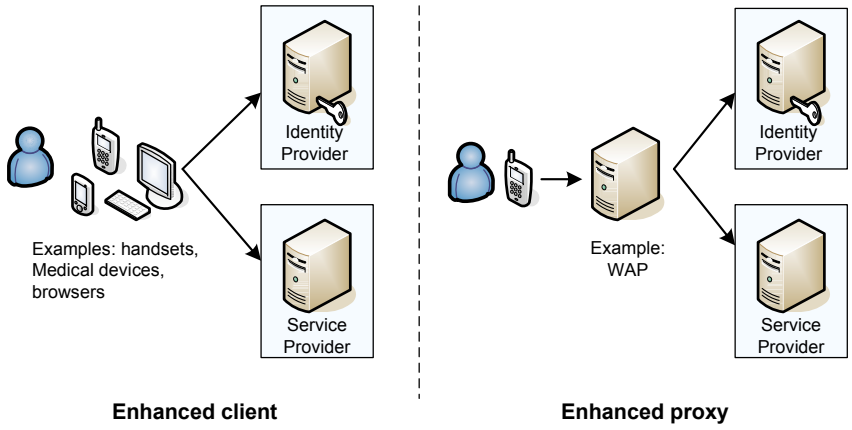

Figure 3.1: Enhanced client/proxy use cases [SAML-tech-overview, section 4.2]

tication methods, access control and authorization performed by UAS. This way WSIDP would also utilize Ubilogin's framework for administering UWAS, directory integrations, group management, and authorization information.

The aforementioned fat client is denoted as an Enhanced Client or Proxy (ECP) in the SAML specification and as a Web Service Consumer (WSC) or Liberty-enabled User Agent or Device (LUAD) in the ID-WSF specification. Both specifications describe the application that the WSC wishes to use as a Web Service Provider (WSP).

The SAML technical overview presents two different use cases for an ECP in figure 3.1. Few additional possible use cases have emerged in the WWW and among the Ubilogin customer base:

- an intelligent browser plugin that addresses e.g. phishing, such as the Windows CardSpace plugin [InfoCard][Malo6]
- any desktop software using SOAP as the application protocol, e.g. payroll computation software that would need to login to make reports for tax administration (see the WSP case example in section 6.3)
- a handheld terminal for e.g. data collection

3.2 Choosing the Specifications

This section introduces first the methods for presenting and acquiring security tokens in WSIDP. Then, the actual authentication protocol and mechanisms, which

are independent of the security tokens, are introduced. The section is concluded with a presentation of the WSIDP message flow.

3.2.1 Presenting Security Information

In the Ubilogin ticket protocol, the core unit of authentication information is a *ticket*. As shown in section 2.5, it is used in different contexts as the application ticket, master ticket, and also as a part of the protocol messages. In most of the XML security specifications such a base core unit is the SAML assertion, although an XML representation of X.509 certificates is used too.

In addition to the SAML protocol itself, also WS-Federation defines a protocol for requesting and mediating security tokens, such as SAML assertions [WSF-PRP, chapter 6.]. The SAML assertions are also used as security tokens in the WSS SAML token profile [WSS-SAML] and Liberty ID-FF 1.2, which is built on top of the SAML 1.1 specification [Liberty-tech, page 20].

The SAML assertion "is a package of information that supplies zero or more statements made by a SAML authority" [SAML-core, page 11]. In our case, the WSIDP is the SAML authority and can also be referred to as the *asserting party*. The statements refer to a *subject*, which is usually presented in the assertion. The SAML specification defines three kinds of statements regarding the subject: authentication, attribute, and authorization decision statements. Additionally, the assertion has a very generic and extensible outer structure. [SAML-core, page 11]

The assertion is designed as an independent unit of information both in the SAML protocol's message exchanges and in the specification's XSDs. The SAML specification allows an assertion to be signed independently of the protocol message, in which it is contained, using enveloped XML signature. This way, a detached assertion is fully functional unit of information. Also, there is an independent XSD for the assertion, making it easy to include the assertion as a part of another specification. All this makes the assertion by far the most important part of the SAML specification.

See listing 3.1 for an example SAML assertion. Note that the assertion itself is not bound to the protocol that carries it or the profile (WebSSO or ECP) used to request it. The assertion payload contains four essential information blocks (emphasized):

- Identity information: the Name ID is "jdoe" with an unspecified format and there is one attribute named "role" with the value "manager".
- The issuer was `https://idp.foo.com/wsidp`.
- Authentication: the used authentication method was `https://idp.foo.com/wsidp/saml2/names/ac/otp.katso.1`, the authentication instant was

Listing 3.1: An example of a SAML 2.0 assertion

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" ID="_1234"
 IssueInstant="2007-03-09T07:43:15Z" Version="2.0">
  <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
  https://idp.foo.com/wsidp</saml:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="#_1234">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>KfXc...</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>YOfP...</ds:SignatureValue>
  </ds:Signature>
  <saml:Subject>
    <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">jdoe</saml:NameID>
    <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
      <saml:SubjectConfirmationData InResponseTo="_4321" NotOnOrAfter="2007-03-09T07:53:15Z"
        Recipient="https://service.foo.com/wsp1/AssertionConsumer"/>
    </saml:SubjectConfirmation>
  </saml:Subject>
  <saml:Conditions NotBefore="2007-03-09T07:43:15Z" NotOnOrAfter="2007-03-09T07:53:15Z">
    <saml:AudienceRestriction>
      <saml:Audience>https://service.foo.com/wsp1</saml:Audience>
    </saml:AudienceRestriction>
  </saml:Conditions>
  <saml:AuthnStatement AuthnInstant="2007-03-09T07:43:15Z" SessionNotOnOrAfter="2007-03-09T08:43:15Z">
    <saml:SubjectLocality/>
    <saml:AuthnContext>
      <saml:AuthnContextDeclRef>
      https://idp.foo.com/wsidp/saml2/names/ac/otp.katso.1
      </saml:AuthnContextDeclRef>
    </saml:AuthnContext>
  </saml:AuthnStatement>
  <saml:AttributeStatement>
    <saml:Attribute Name="role">
      <saml:AttributeValue>manager</saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>
</saml:Assertion>
```

equal to the issue instant of the assertion, and the authentication session lasts for one hour.

- Conditions: the assertion is valid for 10 minutes starting at the issuance time and it is intended solely for the WSP identified with the entity ID `https://service.foo.com/wsp1`.

The potential of the SAML assertions was acknowledged already after the introduction of protocol version 1.1. Consequently, the SAML 1.1 assertions are now deployed in most of the WS frameworks of different middleware and application servers such as the IBM Tivoli, BEA WebLogic, SAP J2EE Engine and JBoss. This has made the SAML assertions a popular XML format for presenting security information concerning a subject. This situation has been achieved because of the early completion of SAML 1.0 and 1.1 protocol versions and their stable status. In the recent years, many implementations have been relying on SAML 1.1 and it has been also fairly well studied (see [Gro03] and [dMdSF05]) whereas the use and study of SAML 2.0 has yet been very minimal.

3.2.2 Authentication Request Protocol

As it was already mentioned, the SAML assertions can be mediated and requested with the SAML protocol itself, Liberty ID-FF, and WS-Federation. However, as of SAML version 2.0, the development of the Liberty ID-FF protocol is discontinued in favor of SAML.

The WS-Federation protocol—together with other WS-* languages, such as WS-Trust, WS-Policy, WS-Addressing, and WSS—enable the requesting and mediating of *security tokens*. For example, the Microsoft ADFS Server issues and processes SAML 1.1 assertions using WS-F PRP [ADFS, page 8], but the protocol is not restricted in using only SAML assertions as security tokens. The sign-in and sign-out protocol over SOAP is defined in the WS-Federation Active Requestor Profile (WS-F ARP). WS-F ARP specifies the use of WS-Trust RequestSecurityToken and RequestSecurityTokenResponse protocol messages [WSF-ARP, page 10].

The SAML protocol version 2.0 is the third major version of the protocol after the versions 1.0 and 1.1. In version 2.0, many important features—such as single logout and name identifier mapping—were added. The structure of the protocol and assertions were enhanced in many ways, e.g. the Subject element was removed from the authentication and attribute statements and raised to the top level in the assertion. This way, the assertion and the different statements refer to only one subject. [SAML-tech-overview, chapter 5]

The version 2.0 has made SAML a very mature and comprehensive protocol. The broadness of the SAML 2.0 specification has been very well divided in the concepts

of the protocol core, bindings, and profiles with the respective partitioning in the specification documents.

In contrast to SAML, the WS-Federation specification is scattered in several WS languages: the requisite documents for WS-Federation include eight of the over 20 different WS languages. Besides, WS-Federation lacks an attribute query protocol and proposes instead to use UDDI as the protocol for accessing attributes [section 5.1.][WSF]. The WS-Federation and WS-Trust are still cited as "initial public draft releases" [WST, page 2].

In both protocols, SAML and WS-Federation, the request and response messages should be—depending on other security measures on the underlying communication channel—signed using the correspondents' private key [WST, section 5.1][SAML-core, page 68]. Also the SAML assertions should be signed to provide "authentication of the issuer and integrity protection" [SAML-core, page 17]. The SAML trust model is discussed in more depth in section 4.3.

All in all, the SAML protocol version 2.0 is more versatile than the WS-Federation protocol and a natural choice for managing security information and especially SAML 2.0 assertions. More specifically, the ECP profile defines how to implement the SAML authentication request and response protocol using the SOAP and PAOS bindings. Its sibling profile, the SAML WebSSO profile, describes the implementation of the existing Ubilogin use case using SAML protocol with HTTP redirect and POST bindings. Finally, the Single Logout profile describes the implementation of the logout for both the SOAP and HTTP bindings. [SAML-conformance, chapter 2]

3.2.3 Authentication Protocol and Mechanisms

The actual authentication procedure is explicitly left out of scope in the SAML Authentication Request Protocol [SAML-core, section 3.4]. In WebSSO, the HTML forms suit well for implementing the authentication process and its user interface. This way, the IDP takes full responsibility over the authentication and has full control over the implementation of different authentication methods. However, when using SOAP, there is a need for a generic and extensible protocol for passing the authentication information between the WSC and IDP.

Authentication protocols do exist both on HTTP and SOAP level. On the HTTP level, the possibilities include the HTTP Authentication per [HTTP-auth] and SSL/TLS client certificate authentication. The HTTP basic and digest authentication schemes and client certificate authentication solve specific use cases, but the HTTP authentication offers also an access authentication framework that could be extended for arbitrary authentication methods [HTTP-auth, section 1.2].

In the WWW-Authenticate HTTP header, issued by the server, the authentication scheme and realm parameters could be used for authentication mechanism and

method negotiation respectively. However, the protocol does not specify a realm parameter for the Authorization header issued by the HTTP client. This means that the needed authentication mechanisms and the negotiation of the authentication method would have to be specified on top of the [HTTP-auth] specification.

On the SOAP level, two different protocols are used for implementing authentication: WSS and Liberty ID-WSF. The WSS specification provides primarily means for SOAP "message integrity and confidentiality" by attaching different security tokens to the SOAP message header [WSS, page 1]. The specification allows using arbitrary token types but comes along with separate token profiles for Kerberos, SAML, username, and X.509 certificates. Additionally, the establishment of an authentication mechanism is listed as a non-goal [WSS, section 1.1.2].

The Windows CardSpace uses WSS for authentication but limits the implementations to four different authentication methods: username and password, Kerberos, X.509 certificate, and self-issued token. In CardSpace, InfoCards serve as the mechanism for selecting the IDP and authentication method. [InfoCard, chapters 5 and 1]

The Liberty ID-WSF 2.0 Authentication Protocol is designed specifically for the needs of general identity authentication per [WL92] and offers a more generic model for authenticating with different authentication methods. The ID-WSF authentication protocol is based on Simple Authentication and Security Layer (SASL) [ID-WSF-authn, chapter 1]. SASL provides means for describing the data sequence—i.e. the byte representation for the request-response messages—for a specific authentication mechanism. Furthermore, SASL describes how to specify new mechanisms. The predefined mechanisms include PLAIN (username/password), CRAM-MD5, and EXTERNAL. IANA maintains the SASL mechanism registry which ensures uniqueness of mechanism names. [SASL]

The actual ID-WSF authentication procedure is realized as a series of one or more SASLRequest/SASLResponse SOAP message exchanges. The client begins the communication by issuing a SASLRequest containing a list of the SASL mechanisms it supports. If the client defines only one mechanism, it can include the authentication data in the first request as described by the mechanism in question. The server decides which mechanism to use and includes that mechanism in the first SASLResponse message. If the authentication ends successfully, the last SASLResponse contains a SAML 2.0 assertion. [ID-WSF-authn]

The mechanisms provide a practical abstraction of the presentation of authentication data. Additionally, the EXTERNAL method provides for authenticating the client externally to the SASL mechanism, e.g. using a SSL/TLS client side certificate.

However, the server might need additional information regarding the authentication method to use, if e.g. there is more than one credential store from where to verify PLAIN mechanism credentials. In ID-WSF, this is solved by using the SAML RequestedAuthnContext element, which can contain URI references to authentica-

tion context classes or declarations, along with an optional comparison attribute. The choosing of authentication method and mechanism is discussed in more detail in section 4.2.

Apart from the authentication mechanism extensibility, another major difference between WSS and ID-WSF is that ID-WSF is realized as elements in the SOAP body where as WSS elements are a part of the SOAP header. This means that the ID-WSF authentication process is an independent message exchange (see steps three and four in figure 3.2), which has the following advantages:

- The authentication process can consist of several message exchanges. In Ubilogin WSIDP, the KATSO authentication method uses OTPs, so the server must be able to communicate the OTP serial number to the client as a challenge in case the client is not able to deduce the number itself. This would not have been possible by using WSS.

- In some use cases the result of the authentication process (success/ failure) is enough and no authorization nor WSP-specific identity information is needed. This use case is covered simply just by using the ID-WSF authentication service without SAML. This use case is further analyzed in section 5.6, Simple Authentication.

The flexibility that ID-WSF offers in comparison to WSS comes along with few disadvantages: more SOAP messages are involved, it requires also the use of some SOAP headers, and is thus more complicated to implement. The WSS header containing the authentication credentials could have been directly attached to the SAML AuthnRequest in step five of figure 3.2. In the simplest case, the whole message flow would have consisted of mere six messages as there would not have been the ID-WSF message exchange and the WSIDP would have received the AuthnRequest simultaneously with the credentials in step five.

In our implementation, as a result of the ID-WSF authentication, the WSIDP issues a SAML assertion which is addressed to the WSIDP itself and includes in the assertion only the identified identity without any attributes. For WSC, the assertion serves as a ticket which should be attached—between the steps 4 and 5 in the figure 3.2—to the SAML AuthnRequest using the WSS SAML token Profile as suggested in the ID-WSF Enhanced Client or Proxy SSO Profile [ID-WSF-authn, section 6.3.2.].

Finally, in step 5, the WSIDP identifies the WSP from the AuthnRequest it receives. Using this information, the WSIDP can make the authorization and access control decisions and optionally add attributes specific to the WSP and Principal in the assertion.

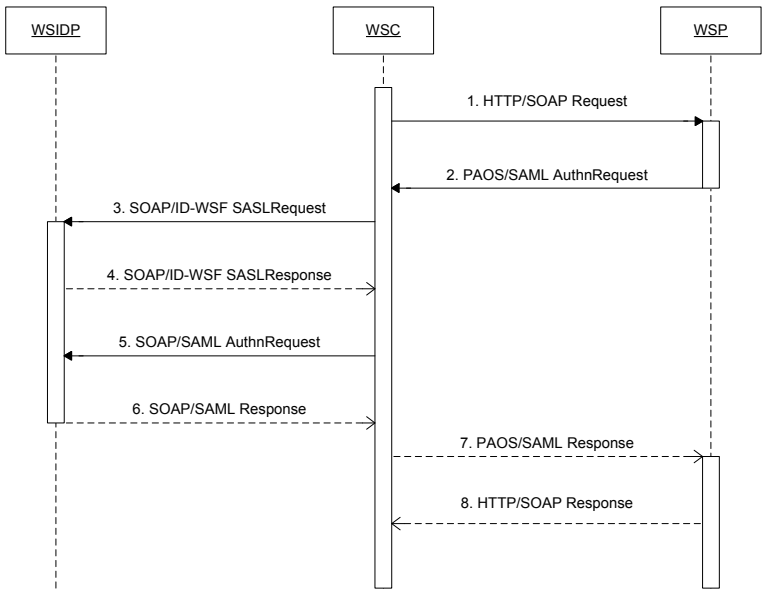

Figure 3.2: WSIDP message flow upon successful authentication

3.2.4 WSIDP Message Flow

The message flow of a successful authentication with WSIDP is shown in figure 3.2. The messages were derived from the ECP profile in [SAML-profiles, section 4.2] and the ID-WSF ECP SSO Profile in [ID-WSF-authn, section 6.3.]. The WSC denotes the party which is referred as ECP in [SAML-profiles] and LUAD in [ID-WSF-authn].

The flow consists of the following steps:

1. WSC tries to access a resource at WSP and issues an application protocol specific request. In the request's HTTP accept header, the WSC declares the PAOS content type.

2. The WSP does not identify the WSC, so it creates a SAML AuthnRequest and issues it in a SOAP message using the PAOS binding.

   The SOAP message header contains a list of IDPs it is addressed to, the WSP's response consumer URL where the response should be returned, and a relay state identifier which should be returned in the response.

3. Upon receiving the AuthnRequest, the WSC knows that it must authenticate to WSIDP and issues an ID-WSF SASLRequest. If the AuthnRequest in step 2 contained a RequestedAuthnContext element, then the WSC should copy the

element to the SASLRequest and use it to derive a suitable SASL mechanism.

4. The WSIDP chooses the SASL mechanism from the list that was included in the SASLRequest in step 3. If the request contained credentials as well, the authentication process is started.

   The WSIDP can issue a challenge in the SASLRequest, if necessary. In that case, the WSC continues with another SASLRequest and receives another SASLResponse. The last SASLResponse contains a SAML assertion that is addressed to the WSIDP.

5. The WSC extracts the SAML assertion from the last SASLResponse and attaches it to the AuthnRequest it received in step 2. The AuthnRequest is sent to the SAML SingleSignOnService endpoint URL of WSIDP using SOAP binding.

6. The WSIDP validates the AuthnRequest and determines if the WSC is authorized to access the WSP using the identity from the attached SAML assertion. Then, it constructs a new SAML assertion, addressed to the WSP, containing the WSC's identity information and attributes according to the rules preconfigured in the Ubilogin management application. This new assertion is similar to the one presented in listing 3.1. It is included in the SAML Response message.

7. The WSC forwards the SAML Response to the WSP in a SOAP message using the PAOS binding. The SOAP header in step 6 contains the assertion consumer URL of WSP where the WSC should forward the SAML Response. This URL must match the response consumer URL of step 2.

   Additionally, the WSC should attach the relay state identifier it received in step 2 to the SOAP header.

8. Now the WSP can use the SAML assertion to identify the WSC. The WSP issues a response to the request in step 1 according to the WSC's identity and the relay state identifier.

3.3 Implementing the Software in Java

The goal was to make the WSIDP a self-contained Web application that would access the authentication methods through UAS and use its access control and authorization mechanisms as mentioned in the beginning of section 3.2. This meant that WSIDP would extend some UAS's operations. Thus, WSIDP was to be implemented

using Java, it would share the same LDAP directory with UAS, and would result in a single Web application archive (WAR). This way, the WSIDP could easily be integrated to existing Ubilogin installations and also use the identity information from the external directories that are integrated with Ubilogin. On the other hand, a WSIDP installation would not be very usable without the Ubilogin management application.

3.3.1 The Ubisaml2 Library

The SAML protocol was realized as an independent library, *ubisaml2*, which implemented the use cases for the Identity Provider and the Service Provider according to both the WebSSO and ECP profiles as described in [SAML-profiles, sections 4.1 and 4.2]. In the library, the profiles and bindings were separated so that UAS could parametrize the IDP class to use the HTTP bindings and WSIDP could parametrize the IDP to use the SOAP and PAOS bindings.

The Sun JAXB 1.1 implementation (and later JAXB 2.0) was used to represent the XML structures in SAML as Java objects. The object hierarchy was very straightforward to generate using the JAXB Binding Compiler (XJC) and the protocol, schema, ECP, and metadata XSDs included in the SAML specification. However, the JAXB classes representing the SAML elements that allow the use of XML signing—i.e. the response, request, assertion, and metadata elements—had to be modified to enable attaching of an XMLDSig in the JAXB tree (see section 3.3.3 for further information). The JAXB class hierarchy was packaged as the *ubisaml2-jaxb* library.

Additionally, JAXP and JAXB parsers had to be constructed to handle the marshalling and unmarshalling of JAXB trees and serialization and deserialization of DOM trees in a consistent way. The parsers were parametrized with the SAML schemata, the Java package names of the XJC generated classes, and fitting XML namespace prefixes for the used namespaces. This way the parsers would always validate XML input according to the given schemata and produce an XML serialization with stable namespace prefixes.

The ubisaml2 implementation required some decisions on how to interpret some of the SAML specifications' concepts in Ubilogin. Some of the most important implementation-specific decisions concerned the trust model, trust relationships, entity IDs, and authentication context.

We decided to base the trust model on XMLDSig as recommended in [SAML-core, chapter 5]. The signing of the assertions and protocol messages provides message integrity, authentication of message origin and non-repudiation of origin which loosen the requirements for the secure message channel and e.g. a SSL/TLS client certificate authentication is not needed. This also means that the communicating parties can be deployed in diverse ways as they do not need a secure point-to-point communications channel.

The use of XMLDSig relies on asymmetric cryptography. We chose to exchange the public keys using the SAML metadata instead of using PKI. Section 4.3 discusses the problems and limitations of the SAML trust relationships, PKI, and metadata in more depth.

A party that offers SAML-based services must have an entity identifier upon which it can be uniquely identified. We decided to use the base URL of the Web application as the entity ID in our SAML implementation. For WSIDP, the identifier would be e.g. `https://foo.bar.com/wsidp` if WSIDP was deployed in the host foo.bar.com. In the example assertion of listing 3.1, the IDP entity ID can be found in the Issuer element and the SP entity ID in the Audience element.

The SAML authentication context provides means for the IDP to present a SP "which technologies, protocols, and processes were used or followed for the original authentication mechanism on which the authentication assertion is based" [SAML-authn-context, page 4]. Using this information, the SPs can "assess the level of confidence they can place in that assertion" [SAML-authn-context, page 4]. There are three different elements that can be used to present the context:

- AuthnContextClassRef: an URI reference to a predefined authentication context class
- AuthnContextDecl: a self-defined authentication context declaration using the structures defined in [SAML-authn-context]
- AuthnContextDeclRef: a declaration reference, which is an URI reference that identifies a declaration and *may* directly resolve to the declaration.

The SP can also use two of the former elements, AuthnContextClassRef and AuthnContextDeclRef, and an optional comparison attribute, to request authentication in a context that is exactly, minimum, maximum, or better than the context defined by the SP.

In Ubilogin, the authentication method names are defined in a per installation basis, which makes the names unique only in a single Ubilogin directory. The predefined classes do not include suitable classes for many of the typical authentication methods used in Ubilogin. Furthermore, the comparability of the authentication contexts is not well-defined and always specific to an IDP. As the same classes can be used by different IDPs with different semantics, we decided to identify the Ubilogin authentication methods with declaration references that consist of the IDP's entity ID, the string `/saml2/names/ac/` and the name of the authentication method. In the example assertion of listing 3.1, the entity ID is `https://idp.foo.com/wsidp` and the authentication method name in the Ubilogin directory is `otp.katso.1`.

This definition of authentication contexts is also used in the ID-WSF protocol messages along with the SASL mechanisms. The limitations on choosing the authentication method and mechanism are discussed in more detail in section 4.2.

3.3.2 ID-WSF Authentication Protocol

The ID-WSF authentication protocol was going to be implemented solely in WSIDP, in contrast with the SAML implementation which was to be shared with other Ubilogin components (e.g. UAS). The implemented version of ID-WSF, the first draft release, consisted of such few XML structures that it was considered easiest to be implemented with DOM. The SOAP messages were handled with the SAAJ library which extends the DOM interfaces.

The data model of the ID-WSF protocol messages, the SASLRequest and SASLResponse, was implemented in respective Java classes. Their class data can be populated either by parsing an input DOM subtree or by passing parameters to a constructor that proceeds to create the respective DOM subtree from scratch. The ID-WSF schema files were used for validating the input but not for data binding as in the ubisaml2 library.

During the implementation it was observed that producing XML using DOM was prone to such errors which were not possible if JAXB was used. DOM takes care of the elementary well-formedness rules (such as element nesting, closing tags and obligatory escaping) and prevents the implementer on making the elementary flaws that occur when XML is produced with string concatenation. However, DOM does not stop him from blundering with e.g. the namespaces or element nesting. However, due to time constraints, we continued to use DOM.

The SAAJ library was observed to have inherited the problems of Java DOM implementation which were discussed in section 2.1. Additionally, the benefits of using SAAJ instead of implementing an own SOAP stack were few. For example the SOAP fault model is not reflected in SAAJ's exceptions and the SAAJ SOAPConnection class did not suffice for the PAOS binding used by the WSC and WSP in SAML ECP profile. We built our own SOAP fault exceptions, a handler for the exceptions and a SOAP client that is capable of modifying the HTTP headers per PAOS unlike the SOAPConnection class.

3.3.3 Implementing XMLDSigs in a JAXB Tree

XMLDSig specifies the signing of a digest computed over an XML node-set [XMLDSig]. However, prior to computing the digest, the XML node-set must be canonicalized per inclusive or exclusive XML canonicalization that provides a canonical octet stream. The canonicalization requires an XML representation that preserves the XML information set. The DOM tree can be configured to preserve the infoset but the JAXB tree does not preserve all of the infoset.

To implement the XMLDSig verification, the DOM representation had to be preserved during the unmarshalling of XML. The JAXB 2.0 implementation contains a Binder class that does just that. It enables "synchronization between XML infoset

nodes and JAXB objects representing same XML document" [Binder].

We built a SignedTypeInfo class that XJC was parametrized to use as a parent class for those SAML elements that could be signed. Also, a JAXB parser wrapper was implemented that would create the Binder and save its reference during marshal and unmarshal operations of the signable elements. Later, the SignedTypeInfo could provide access to the Binder and the XMLDSig DOM element for the signature verification.

After unmarshalling an input XML document or marshalling a self-constructed JAXB tree, the tree could still be accessed, but it must be treated as read-only.

Implementing XMLDSig contains also some additional pitfalls. As mentioned before, the DOM implementation must be configured to preserve the XML infoset. More specifically, it must be configured to preserve XML comments and element content whitespace.

The public key, or a respective certificate, can be included as a part of the XMLDSig block in a signed XML document [XMLDSig, section 4.4]. In such case, the XMLDSig specification requires that the key can be used in validating the signature. The fulfillment of this requirement, however, is up to the XMLDSig implementation that creates the signature and thus cannot be trusted by the verifier. Hence, if there is a key included in the XML signature, the verifier must always also verify the key.

The trust of key information is declared to be out of scope in the XMLDSig specification. In our implementation, the trust relationship is created by exchanging SAML metadata as discussed in 4.3.

3.4 Security Considerations

In the WSIDP SAML implementation, the SOAP and PAOS bindings were used over HTTPS without bilateral authentication. Nonetheless, all SAML protocol messages (i.e. AuthnRequest and Response) were required to be signed using XMLDSig. This combination provides a high level of countermeasures against the probable security threats listed in [SAML-sec-consider, section 3.3]. The SSL/TLS client certificate authentication was not applied to achieve better usability and to allow anonymous WSCs to communicate with the WSIDP.

The ID-WSF specification denotes that an authentication mechanism `urn:liberty:security:2003-08:TLS:null` (as defined in [ID-WSF-sec-draft, chapter 6.]) should be used [ID-WSF-authn-draft, section 5.3.]. In practice, this means HTTPS without bilateral authentication as well. This is a natural choice since the meaning of the ID-WSF authentication protocol itself is to identify clients.

SSL/TLS provides countermeasures for most of the possible attacks but a Denial of Service (DoS) attack is still possible. In SAML, the risk for a DoS attack is mitigated by requiring protocol messages to be signed. Generally, a DoS attack

that just slows down the IDP service can never be fully prevented. More importantly, the SAML implementation should not crash under a DoS attack, because of e.g. excessive memory consumption when tracking the message IDs. This has been considered in our SAML implementation

In ID-WSF, signing the SASLRequest messages can not be a strict requirement and using signatures would pose the trust relationship problem (discussed in section 4.3) between the WSC and WSIDP without completely preventing the DoS attacks. Allowing the use of unsigned SASLRequest messages also facilitates the integration of several WSCs.

The disadvantage of allowing the use of unsigned SASLRequests is that an anonymous attacker can use the ID-WSF authentication protocol for guessing passwords. The password guessing can cause severe problems if the underlying user directory locks an account after too many wrong password attempts. However, password guessing can never be fully prevented in an IDP.

Furthermore, the security measures in the current WSIDP implementation allow for a two-way authentication procedure (presented in section 5.3) without any modifications or configuration changes in the WSIDP.

3.5 Providing a SSO Session in Web Services

Should the WSIDP provide a single sign-on session so that the WSC would not have to present a user interface asking for the credentials more than once? The SSO session is crucial if the authentication method uses OTPs because in that case, the WSC cannot just save the credentials locally. The need for a SSO session between the WSC and WSIDP is reduced if there is a good solution for sessions between the WSC and WSP.

Recall from section 2.5, that UAS stores the master ticket in a cookie. When an application login session (i.e. the application ticket) expires, the browser is redirected to UAS that creates a new application ticket provided that the master ticket is still valid. Simultaneously, UAS also updates the master ticket cookie as a part of the SSO process. Usually, the session of a master ticket is longer than the application ticket's session.

However, using cookies to manage sessions in Web services is not always a working solution. In fact, SOAP is designed to be stateless—just as the (usually) underlying HTTP. According to Pulavarthi, the SOAP stack of Java, JAX-WS, ignores cookies [Pul06]. Additionally, it has been argued that cookies and other transport layer devices should not be used for session management in Web services because SOAP is transport protocol agnostic, at least in theory.

Pulavarthi and Kawaguchi propose each a solution as a part of JAX-WS; the other enabling the use of cookies in Web services [Pul06] and the other using the

endpoint references of WS-Addressing for identifying sessions [Kawo6]. Both of the solutions transfer an identifier for a server-based session that inflicts problems when the service needs to be replicated. This is not a problem with the UAS master ticket that completely describes a session without server-based state information.

Additionally, Hildebrand et al propose the use of WS-Context over WS-Addressing for managing sessions [HKLP05]. They argue that WS-Context is a more generalized and loosely coupled solution than WS-Addressing. WS-Context specifies a Context Manager Web service that must be used when context is passed by reference. Nevertheless, WS-Context also allows for a context to be passed by value, in which case no server-based state information is needed. The specification recommends the signing of WS-Context headers and encryption of sensitive context data per WSS [WS-CTX, chapter 6]. Thus, WS-Context seems as a very viable choice for implementing sessions in a Web service.

Chapter 6 of [ID-WSF-authn] defines a Single Sign-On Service for LUADS. The SSO service is based on the SAML ECP profile and ID-WSF authentication service. The ID-WSF SSO service specification, however, does not explicitly define how to uphold the session.

In the ID-WSF authentication service, there is a remark for using the WSS Security SOAP header for "security token renewal" [ID-WSF-authn, section 5.4.] without further elaborating the message exchange. A simple solution would consist of a SASLRequest message with the EXTERNAL mechanism [SASL, appendix A] and the initial SAML assertion (received in step 4 of figure 3.2) attached in the SOAP header inside a WSS Security element.

Our first implementation of Ubilogin WSIDP does not offer renewal of the initial SAML assertion. The initial assertion can nonetheless be used for acquiring assertions for several WSPS during its lifetime. Hence, it can be argued, that the WSIDP offers a partial SSO.

Single Logout In addition to single sign-on, the single logout needs as well a session mechanism to work. UAS stores an entity identifier for each SP in a cookie. When it receives a logout request, it iterates through the entity IDS, finds the logout URL for the SP in the metadata, sends each SP a logout request, and finally a logout response to the original requester.

The SAML single logout profile suggests that the session mechanism necessary for single logout may be established "by means of a cookie, URL re-writing, or some other implementation-specific means" [SAML-profiles, section 4.4]. If a server-side session management is feasible, the SessionIndex attribute of SAML AuthnStatement can be set in the SAML assertion that is addressed to the WSP. The WSP would then return the same session index in the logout request and identify the session information.

A session mechanism similar to UAS cookies, that does not rely on server-side sessions, could be achieved by including a WS-Addressing endpoint reference in the SOAP headers of the SAML response message (step 6 in figure 3.2). The reference should define the WSIDP's single logout URL as the *address* and the WSP entity IDS in the *reference parameters* [WSA, section 2.1]. The integrity of the id values should be protected with XMLDSIG. This kind of solution would, however, impose the handling of WS-Addressing SOAP headers as a further requirement for the WSC.

Chapter 4

Limitations of the SAML and ID-WSF Specifications

In this chapter, we describe the major shortcomings of the SAML and ID-WSF specifications we have noticed along the implementation; namely the initialization of the authentication procedure, choosing the authentication method and mechanism, and finally the management of SAML trust relationships.

4.1 Initiating the Authentication Procedure

Leaving the actual peer-entity authentication out of scope in SAML is justified. As the footing of SAML is in WebSSO, the authentication request protocol specifies that the IDP initiates authentication immediately upon receiving the AuthnRequest message. This is natural in WebSSO since the server usually presents a login form for the browser. When the login form is submitted, the server redirects the SAML response message back to the SP.

For SOAP binding, a similar message flow is introduced both in the ECP profile and in the ID-WSF SSO service that extends the ECP profile. The ID-WSF authentication protocol, however, is client-initiated. Even the ID-WSF specification does not describe, how the IDP, upon receiving the AuthnRequest, should notify the WSC that it should launch the ID-WSF authentication protocol.

In the step preceding authentication in the ID-WSF SSO service specification (corresponding step 5 in figure 3.2), there is a side note suggesting that the AuthnRequest SOAP message will generally "contain a security token obtained from the ID-WSF Authentication Service" in a WSS header block [ID-WSF-authn, section 6.3.2., step 4.]. Still, the specification fails to describe how the WSC recognizes when to initiate the authentication service.

This shortcoming demanded us to document the message flow more carefully

than what the SAML and ID-WSF specifications have accomplished to do. The result is the message flow depicted in figure 3.2, which assumes that upon receiving a PAOS bound AuthnRequest message, the WSC starts the ID-WSF authentication protocol, if it does not already possess a security token from an earlier authentication.

Furthermore, as we had to document the precise message flow anyway, we did not find any need for the ID-WSF SSO service, which requires the use of ID-WSF SOAP binding on top of the SAML SOAP binding. Thus, we decided to use the plain SAML ECP profile along with the ID-WSF authentication service.

In addition, if the WSIDP initiated the authentication protocol upon receiving the AuthnRequest message, it could respond with a SAML Response instead of a SASLResponse as the final response message to WSC. In this case, the server would not need to craft an initial SAML assertion that is not addressed to the WSP and that has to be handed back to the server along with the AuthnRequest. To enable this kind of message flow in the ID-WSF authentication protocol, the means by which the IDP could present the possible authentication mechanisms and methods should be specified (refer to the discussion below in section 4.2).

Eventually, the server-initiated authentication protocol would need to be bound to PAOS as it would be initiated as a response from WSIDP to the AuthnRequest message and as the WSC does not usually have a SOAP endpoint where the WSIDP could send the protocol request. This would unfortunately further complicate the WSC's requirements, which already present quite a challenge for the integration as discussed in section 5.2.

Alternatively, HTTP authentication, mentioned in the beginning of section 3.2.3, could be implemented as an alternative authentication protocol besides ID-WSF. It provides for both server and client-initiated authentication. It should be noted though, that although it is possible to implement HTTP authentication in the WSC using SOAP frameworks such as JAX-WS or Apache Axis2 [Axis2], it would not fully utilize the SOAP level tools such as WSDL and SOAP fault mechanisms. HTTP authentication is included as a binding extension in the WSDL 2.0 adjuncts specification, but only basic and digest schemes are supported [WSDL2-adjuncts, section 6.11.2].

4.2 Choosing the Authentication Method and Mechanism

The ID-WSF authentication protocol is based on SASL. SASL is designed as an abstraction layer between authentication mechanisms and Internet protocols such as SMTP and LDAP [SASL, chapter 1.]. Hence, SASL is originally designed for use cases, in which a client communicates directly with a service provider application, which

implements SASL authentication.

ID-WSF, however, specifies an authentication protocol independent of service providers in which a client authenticates to an IDP using SOAP as the transport protocol. This kind of protocol is quite the opposite of the typical WebSSO use case, in which the IDP identifies the SP from an authentication request. Only after having identified the SP, the IDP initiates the authentication by offering the user a set of authentication methods that have been configured for the SP in question.

As described in section 3.2.3, the SASL mechanisms specify just the data sequence and message flow, but in ID-WSF, the SAML RequestedAuthnContext element may be used for identifying the actual authentication method. The ID-WSF specification suggests that the WSP may offer the RequestedAuthnContext element to the WSC, e.g. in a situation where it demands a strong authentication method to be used. The element is to be included in the SAML AuthnRequest message issued by the WSP [ID-WSF-authn, section 5.1.1.]. Furthermore, the ID-WSF specification requires that the SAML assertion resulting from a successful authentication must satisfy the requested authentication context or else the authentication exchange must be aborted [ID-WSF-authn, section 5.3.].

However, the specification does not define a relation between the authentication contexts and SASL mechanisms. The RequestedAuthnContext is virtually an obligatory element, because there are usually more than one authentication method—and often even several instances—using one SASL mechanism. As the ID-WSF protocol is server-initiated, the WSIDP can not present the WSC a list of authentication contexts from which to choose a suitable method.

This leads to the following problems on how the WSC should react on the authentication context requested by the WSP. Both situations cause additional requirements for the WSC implementation.

If the WSP requests an authentication context the WSC should be able to deduce, which SASL mechanisms can be used to authenticate complying with the given authentication context. The authentication context classes and declarations, however, do not provide a hint of the SASL mechanisms.

The WSP does not request any authentication context. In addition to the previous requirements, the WSC—instead of the WSP—should be able to build such a RequestedAuthnContext, which the IDP would understand. This means that the WSC must know the IDP's authentication context semantics and the possible classes or declaration references.

4.3 SAML Trust Relationships

As mentioned in section 3.3.1, XMLDSig relies on asymmetric cryptography and thus the SAML implementation needs a mechanism for obtaining public keys and verifying the key owner. The problem is defined as out of scope in [SAML-sec-consider, section 3.2] but Public Key Infrastructure (PKI) is suggested as a solution in section 4.4.2 and in [SAML-tech-overview, section 3.4]. Also the Liberty Conformance Testing relies on PKI [Liberty-testing, section 1.4].

However, a proper PKI setup is laborious to build and would require specifying a process for its use in SAML. Also the SAML security considerations document denotes that a PKI setup is not trivial and must be meticulously implemented on each layer to ensure security and overall trustworthiness of the solution [SAML-sec-consider, section 3.2]. Abdulrahman notes that PKI "has proven to be difficult and expensive to build and manage" and states that it is an unrealistic demand for a Web services infrastructure [Abd05, section 5.9.1]. Abdulrahman, and also Salz, propose use of XML Key Management Specification (XKMS) to reduce the costs of PKI [XKMS-does]. However, XKMS does not solve the problem of trust but merely delegates the trust to another party by providing Web services protocols for fetching and validating as well as registering, revoking, and reissuing of public keys [XKMS].

One of the main design principles of Ubilogin, easy deployment, would not be achieved should Ubilogin require a PKI system. Furthermore, the use of PKI is not usual in most of the organizations that wish to deploy a SSO system. As PKI would also be required for services protected with Ubilogin UAS or WSIDP and building our own PKI solution was considered a far too big task, we had to seek alternative solutions. The only feasible solution was to distribute the public keys in the KeyDescriptor element of SAML metadata.

The SAML metadata is used to "express configuration information between SAML parties" [SAML-tech-overview, section 3.1]. It concerns a single SAML entity (e.g. an IDP or SP) and contains typically the endpoint URIs and respective protocol bindings together with key information for encryption and signing.

In WSIDP, the SAML metadata is published with HTTPS in a well-known location, derived from the entity ID URI. The actual binding of identity to key is done by uploading the WSP metadata in Ubilogin management application. This means that the administrator must verify that the metadata source is correct.

This solution has some drawbacks: the metadata has no revocation mechanism and new metadata can not be reloaded on-line upon expiration because new metadata can not be verified as the public key has also expired along with the original metadata. New metadata can always be uploaded in the management application instead of key revocation. To tackle the latter problem, ubisaml2 implementation accepts SAML metadata without the required validUntil or cacheDuration

attributes.

The use of SAML metadata for distributing keys is a compromise between usability and security. As a consequence, Ubilogin does not require PKI. Thus, it is easy to deploy in most environments, but requires careful use of metadata.

Chapter 5

Integrating an Identity Provider in a Production Environment

Chapter three presented the implementation of Ubilogin WSIDP per ID-WSF 2.0 authentication service and SAML 2.0 authentication protocol according to the ECP profile. The specifications imply a message flow which is depicted in figure 3.2. This chapter begins by discussing the requirements that the presented three-way authentication procedure pose for the implementation and integration of WSC and WSP and the problems thereof.

Next, four alternative integration mechanisms are introduced: two-way authentication procedure, three-way authentication without PAOS, unsolicited authentication, and lastly, simple authentication by using solely the ID-WSF authentication service without SAML protocol. Additionally, the implications of these alternative mechanisms on the requirements and security of WSC and WSP are discussed.

5.1 Requirements for the WSC and WSP

The message flow shown in figure 3.2 poses different kind of requirements for the WSC and WSP. In addition to SOAP, both of the participants need to implement the Reverse HTTP binding for SOAP (PAOS) which is used in the SAML ECP message exchange pattern between the WSC and WSP.

An additional requirement for both parties is time synchronization. The validity period of SAML assertions starts typically at issuance time. This means that WSC's and WSP's clock should not lag even one second from WSIDP's clock. Furthermore, the validity period is used alongside message IDs to prevent replay and reduce the risk of Man-in-the-Middle (MITM) attacks [SAML-sec-consider, section 6.1.6]. WSC and WSP should consider using a protocol such as NTP for time synchronization.

5.1.1 Requirements for the WSC

The only protocol messages the WSC must produce are the ID-WSF SASLRequest messages. Additionally, the WSC must be able to extract the RequestedAuthnContext element from the SAML AuthnRequest message it receives from WSP and use the element in the SASLRequest (in steps 2 and 3 of figure 3.2). The WSC should also be able to use the information included in the RequestedAuthnContext element for deducing the appropriate SASL mechanism to use (recall section 4.2 for discussion on choosing the mechanism).

Finally, the WSC must as well be capable of extracting the initial SAML assertion from the final SASLResponse message and attach it to the WSS SOAP header of the SAML AuthnRequest message (steps 4 and 5 in figure 3.2). It should also be able to check the validity period of the SAML messages and assertions. Thus, the WSC does not need to produce SAML itself. It does not need to produce or verify XMLDSigs either.

5.1.2 Requirements for the WSP

Plainly put, the WSP must be able to create and parse SAML messages according to the ECP profile and PAOS binding, i.e. it has to have a SAML stack. The SAML specification further requires that the WSP must be able to produce and verify XMLDSigs.

The authentication protocol used between WSIDP and WSC (ID-WSF in our implementation) does not concern WSP.

5.2 Problems of the Three-way Authentication Procedure

Let us consider a typical scenario where a WSIDP would be used: an application developer designs a new Web service application operating in the WSP role that would use WSIDP for identifying client applications, i.e. WSCs. Another scenario could be integrating the WSIDP's identity sources to a WSP in production. In such a case, the authentication has probably been implemented as a part of the WSP, without a designated IDP, and possibly using transport-level mechanisms (as presented in the background section 2.4).

Furthermore, the previous scenarios can be divided in two different cases:

A. either the WSP developer intends to provide a Web service interface where third-party client application developers can integrate, or

B. the WSP developer will also implement all the client applications.

In any case, the service provider has to decide, how client applications are going to be authenticated: whether the WSCs will operate as enhanced clients or will there be an enhanced proxy between the clients, WSP, and WSIDP as in figure 3.1. In this decision, the service provider must consider, whether the WSCs are mobile devices, on what kind of platform they operate, and what kind of development tools the platform offers for implementing the authentication protocols.

As it was seen in the previous section, there are quite a few requirements for the WSP and even more for the WSC. Unfortunately, the needed software components (i.e. SAML and ID-WSF stacks) that comply with the requirements are not known to be available as of writing.

In both of the cases presented above, the application developer wants to make the integration of a WSC as easy as possible. Especially in case A, implementing the authentication protocols should not cause a major workload for the WSC developer. In case B, or if the client is a mobile device, the developer might want to implement the communication between WSP and WSC using some other protocol than SOAP because of the limitations of the WSC platform (e.g. due to the large memory footprint of all the needed XML/SOAP processors and protocol stacks).

The WSP developer must also trust that the WSC implementation is secure enough and that it does not contain vulnerabilities that could breach the authentication procedure between the WSP and WSIDP. An important part of the WSC implementation is the user interface that should enquire the credentials from the user in a secure way. Additionally, the WSP cannot ensure, that the WSC does not cache sensitive identity information.

There is a solution to the presented problems of the three-way authentication procedure: the two-way authentication procedure.

5.3 Two-way Authentication

In the two-way authentication procedure depicted in figure 5.1, the enhanced proxy acts also as the Web service provider (see figure 3.1). This way, the client application communicates only with the WSP. Such a behavior is even suggested for a SAML 2.0 SSO-enabled website in [ID-WSF-authn, section 6.1.]. The WSP/ECP entity should not be confused with the LUAD-WSP profile presented in [ID-WSF-client-profiles, section 4.1.] where a LUAD (marked as Client in figure 5.1) acts also as a WSP.

The two-way authentication procedure is derived from the three-way procedure (see figure 3.2) by leaving out the PAOS bound messages in steps 2 and 7. In this procedure we do not call the client a WSC because it does not implement the WSC role of SAML and ID-WSF protocols. Nevertheless, it operates as a WSC if it uses Web services to communicate with the WSP.

The only requirement for the client application in this model is to attach the

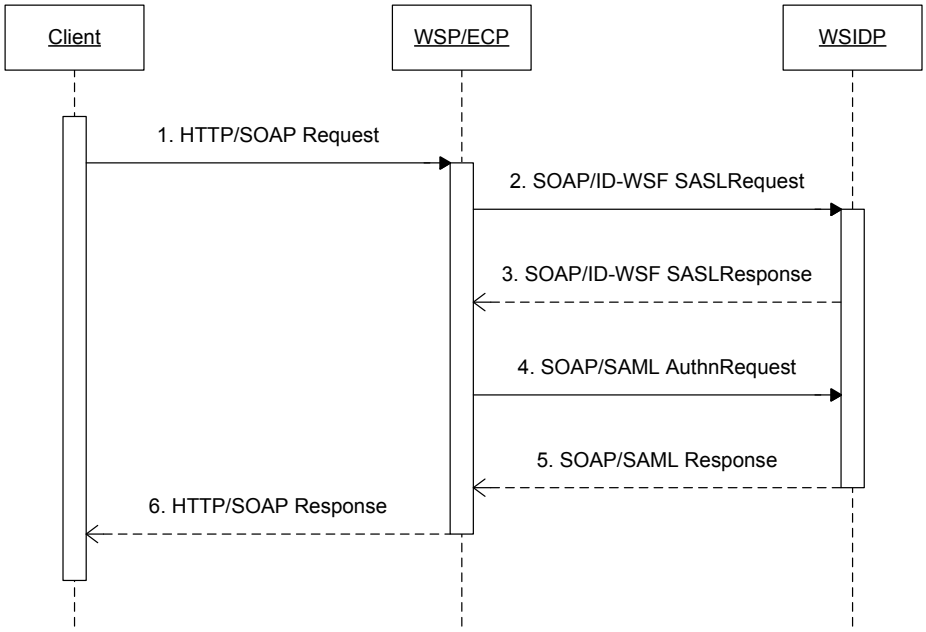

Figure 5.1: Message flow in the two-way authentication procedure

credentials in the application protocol in step 1 of figure 5.1. The WSS SOAP headers could be used to carry the credentials in an interoperable way. Additionally, the WSP can establish a SSO session with the WSC in the application protocol response of step 6, for example by using the WS-Addressing endpoint references as discussed in section 3.5.

If the WSC and WSP are to be developed as a closed system and no interoperability to other implementations is needed, then even proprietary protocols can be used in their communication in steps 1 and 6.

5.3.1 Benefits

The two-way authentication procedure solves the problems mentioned in section 5.2 by implementing the WSC's operations in the WSP. This eases the WSC's development and hardware requirements. It also means that the WSP developer takes responsibility—instead of the WSC developer—of producing trusted implementations of the WSC profiles in SAML and ID-WSF protocols.

Additionally, the problem of choosing the right authentication mechanism based on the RequestedAuthnContext (recall from section 4.2) does not exist, because the WSP implements both protocols.

Using the two-way procedure, the WSP developer could easily specify the inclusion of credentials in the application protocol WSDL. For a WSC developer, it is much easier to comply with the WSDL-defined protocol than to find and absorb all the needed parts of SAML and ID-WSF specifications and bindings.

Furthermore, the whole two-way procedure does not use PAOS, which is not very well supported in the SOAP stacks today. It is also possible to use the two-way procedure without any changes made in the WSIDP because regarding the WSIDP, the message exchanges are identical to the three-way procedure.

5.3.2 Security Considerations

The same security considerations apply to messages 2–5 in the two-way authentication procedure as to messages 3–6 in the three-way procedure, as discussed in section 3.4, [SAML-sec-consider], and [ID-WSF-sec-draft]. Additionally, if the credentials are transferred unencrypted in message 1, the same security mechanisms should be applied as with ID-WSF SASLRequest messages, i.e. (at least) unilateral SSL/TLS authentication should be used.

5.4 Three-way Authentication without PAOS

If the client needs to be able to choose from more than one authentication method, it would be feasible to use the ID-WSF authentication protocol directly between the client and WSIDP and pass the resulting SAML assertion to WSP using the WSS SOAP header block. The WSP could then issue a SAML AuthnRequest directly to the WSIDP. This kind of message flow is depicted in figure 5.2. It does not require PAOS either and is supported by the WSIDP as well.

Another reason for the client to use ID-WSF is an authentication mechanism which requires the IDP to issue challenges. This means more than one round-trip in the ID-WSF authentication protocol. Challenges are required to communicate the OTP serial numbers in the KATSO mechanism in Ubilogin WSIDP.

5.5 Unsolicited Authentication

The ID-WSF specification suggests yet another message flow, in which the WSC issues a SAML AuthnRequest instead of the WSP [ID-WSF-authn, section 6.3.2.]. In the request, the WSC's entity ID is declared as the issuer and the WSP's entity ID as the requester ID in the scoping block.

This implies that the IDP must have a trust relationship with the WSC to be able to verify the request's XMLDSig. Additionally, the WSP would have to accept an

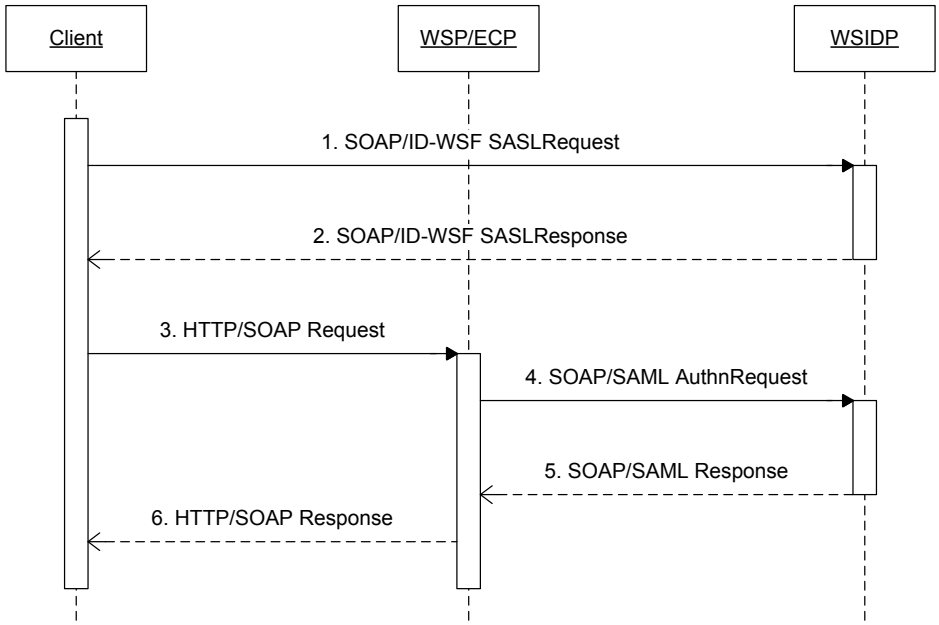

Figure 5.2: Message flow in three-way authentication without PAOS

unsolicited SAML Response. This message flow is not currently implemented in WSIDP, but could be easily supported. This message flow avoids the use of PAOS too.

The unsolicited message exchange can be further simplified by accepting an AuthnRequest without XMLDSig from the WSC (such as in listing 5.1). The IDP will issue a SAML Response with an ecp:Response element attached in the SOAP header. The Response element indicates the WSP's assertion consumer service URL for the WSC.

To request a resource at the WSP, the WSC can attach the URL of the resource in a RelayState header block on the SAML Response message [SAML-profiles, section 4.2.3.7]. This way, no application protocol request is needed and the WSP can react immediately upon the SAML Response by issuing the application protocol response to WSC, identically with steps 7 and 8 in figure 3.2.

5.6 Simple Authentication

As the last alternative, we present the simple authentication procedure. It is derived from the two-way procedure by discarding the SAML protocol messages in steps

Listing 5.1: A SAML AuthnRequest message without XMLDSig

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header>
    <Security
     xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
      ...</saml:Assertion>
    </Security>
  </SOAP:Header
  <SOAP:Body>
    <samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
     xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" ID="_4321"
     IssueInstant="2007-03-09T07:43:14Z"
     ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP" Version="2.0">
      <saml:Issuer>https://wsc/</saml:Issuer>
      <samlp:Scoping>
        <samlp:RequesterID>https://service.foo.com/wsp1</samlp:RequesterID>
      </samlp:Scoping>
    </samlp:AuthnRequest>
  </SOAP:Body>
</SOAP:Envelope>
```

four and five of figure 5.1. In the simple authentication, the WSP simply checks whether the client's credentials are valid or not by using the ID-WSF authentication protocol.

The same procedure could be conducted by the WSC. In that case, the WSC would extract the SAML assertion from the SASLResponse message and attach it in the application protocol message, e.g. using the WSS SOAP header block.

In this model, the WSIDP never identifies the WSP and thus the SAML assertion is not targeted to the WSP. Thus, the assertion does not contain any attributes or other identity information from Ubilogin's identity sources that are configured to be revealed to this WSP. The assertion simply serves as an answer to the question: "Are these credentials valid or not?"

Since the WSP is not identified, the WSIDP can not provide authorization nor access control. Furthermore, the SAML protocol is required for more complex use cases such as federation and delegation which can not be achieved by simple authentication.

5.7 Which Procedure to Use?

Is there any reason to use the three-way authentication procedure as described in the specifications? The answer is yes: not all use cases can be easily solved using the two-way procedure.

The use of SAML in WSC is justified if the WSC wishes to identify itself to more than one WSP using the same IDP. If the three-way authentication procedure is

used, the IDP can offer a SSO session between the WSPS as discussed in section 3.5. The three-way procedure also enables a set-up where the WSIDP is situated in an intranet while the WSPS reside in a remote network and can not thus access the WSIDP directly.

A summary of the requirements and drawbacks of different authentication procedures is given in table 5.1.

Table 5.1: Comparison of different authentication procedures

Procedure	Requirements for WSC	Requirements for WSP	Drawbacks
3-way	ID-WSF, PAOS, WSS	PAOS, SAML	
2-way	-	ID-WSF, SAML†, WSS	no SSO session for WSC
3-way w/o PAOS	ID-WSF, WSS	SAML†, WSS	limited SSO session for WSC
Unsolicited	ID-WSF, WSS, SAML	unsolicited SAML	
Simple	-	ID-WSF†	no authorization, no access control, no attributes

† Requires additionally a direct connection to WSIDP.

Chapter 6

Conclusions

In the three previous chapters, we presented the essentials of the Ubilogin WSIDP implementation, limitations of the implemented specifications, and alternative procedures for integrating WSCs and WSPs with WSIDP. In this chapter, we conclude the major outcomes of the previous chapters: how did the used tools suit for implementation and how well did the chosen specifications serve our needs? Additionally, we present a WSP in production as a case example of the needs and requirements of integration.

6.1 The Tools

As WSIDP was to be integrated in Ubilogin, it was a natural choice to use the existing libraries and development framework. Initially, the used version of Java language was 1.4.2. The built-in XML tools of Java—the XML parser along with a DOM and SAX implementation—are mature and stable. For our needs, a plethora of additional libraries was found through e.g. the Java Web Services Development Pack (JWSDP), Apache Foundation, and Bouncy Castle (crypto APIS)[1].

The specifications of the implemented protocols, SAML 2.0 and ID-WSF 2.0 draft 1, contained the normative XML Schema definitions but no WSDL service descriptions. We considered building WSIDP on Apache Axis, but found it cumbersome to use and decided to build the Web service processing in standard Java servlets using and extending SAAJ when needed. The protocol stacks were built using JAXB 1.1, DOM, and the Apache XML Security library as explained in section 3.3.

As discussed in section 3.3.3, using JAXB together with the Apache XML Security library's XMLDSig implementation was challenging but successful. In general, succeeding in using JAXB depends heavily on the way the XSDs are written. The SAML protocol and assertion schemata are concrete enough and allow content from

[1] http://www.bouncycastle.org/

other namespaces only in a few elements. Thus, the classes generated with XJC were strongly typed as well and fit for use.

The experiences with building XML abstractions with DOM were to the contrary. As discussed in section 3.3.2, DOM was considered more laborious to use and caused more problems than JAXB. SAAJ caused probably more problems than offered advantages.

Later on, the ubisaml2 implementation was upgraded to use JAXB 2.0 and Java 1.5. JAXB 2.0 generated more usable interfaces, because Java 1.5's generics enabled collections with a stronger typing.

Future Development In addition to JAXB 2.0, other new tools for building Web services—such as Axis2 v1.1.1 and JAX-WS 2.0—have emerged during the last year. These tools would need reviewing during the future development of WSIDP.

The final version of ID-WSF 2.0, published in October 2006, was accompanied with a WSDL service description. The use of WSDL together with Axis2 should be considered when WSIDP is upgraded to implement the final version.

Additionally, the Shibboleth OpenSAML[2] 2.0 and openLiberty[3] ID-WSF 2.0 implementations are still under heavy development but are very interesting to follow, at least for interoperability testing.

6.2 The Specifications

In section 3.1, we set the goal for Ubilogin WSIDP implementation to be a standards-based Web service login interface and to integrate into the existing Ubilogin implementation. The background for this goal was the need to facilitate the development of Web services applications in environments where Ubilogin is used for Web application authentication.

How did the selected specifications meet the goals?

6.2.1 SAML 2.0

SAML authentication request protocol has a very similar foundation as our use cases: a SP, known by the IDP in advance, issues an authentication request to identify a Principal and establish a security context. Thus, the protocol served us well.

Some SAML concepts, such as the trust model and authentication contexts, were new to Ubilogin and required considering and deciding how to implement them as

[2] http://www.opensaml.org/

[3] http://www.openliberty.org/

discussed in section 3.3.1. The most difficult problem concerning the implementation was the lack of a well-defined trust mechanism as we did not opt for using PKI (see section 4.3).

Regarding the deployment of WSPs and WSCs, the biggest challenge proved to be the requirement of using PAOS in the communication between a WSP and WSC. This reflects the foundations of SAML in WebSSO, where the HTTP POST redirects—or so called client redirects—can be used instead of PAOS.

PAOS was one of the challenges that we tried to tackle with the alternative authentication procedures presented in chapter 5. The procedures include relegating the ECP role to WSP (two-way procedure and three-way procedure without PAOS) and presenting an unsolicited SAML response to WSP. Even discarding SAML completely was considered in simple authentication, in section 5.6.

6.2.2 ID-WSF 2.0

The actual identification of the Principal is not defined in SAML, so we decided to use the ID-WSF 2.0 authentication protocol to accomplish that. The protocol is an extension of SASL. The SASL specification provides for one of the two implemented authentication mechanisms, the PLAIN mechanism. We specified the other mechanism, KATSO, as a SASL mechanism ourself (see appendix A). The KATSO challenges could not have been implemented with WSS.

ID-WSF seemed a suitable protocol to implement alongside SAML, because Liberty has participated in the SAML development in OASIS. Furthermore: ID-WSF uses SAML 2.0 as one of its building blocks and even profiles the ID-WSF SSO service on SAML. However, the protocols do not co-operate quite seamlessly during the initiation of authentication and choosing of the authentication method and mechanism, as noted in sections 4.1 and 4.2.

The Liberty ID-WSF is actually a framework for intelligent clients (dubbed LUAD-WSCs) to authenticate, request service resources from ID-WSF Discovery Service, and manage their own identity data and its propagation to WSPs with ID-WSF Data Service Templates. Cahill presents an example of such a deployment, the AOL's radio application in [Liberty-overview, pages 12–21]. This kind of intelligent client is far from the typical WSCs in Ubilogin WSIDP use cases. For us, the typical WSC is strongly oriented towards one WSP and does not implement a heavy ID-WSF protocol stack.

Although we implemented the required mechanisms successfully and co-operation between SAML and ID-WSF was possible, implementing HTTP authentication as an alternative authentication protocol should be considered (see sections 3.2.3 and 4.1 for more discussion). Nevertheless, support for ID-WSF might be crucial in the future when the liberty-aware intelligent clients are available for complex use cases like AOL radio, federation and delegation (see sections 7.2.1 and 7.2.2).

6.3 Integration in a Production Environment

As of writing, Ubilogin WSIDP has been installed in few environments and also placed into production. However, to the best of our knowledge, there exists only one case (presented below), in which the integration is designed to be carried out as specified in SAML and ID-WSF, i.e. using the three-way authentication procedure.

In the other integration cases, the alternative methods presented in chapter 5—such as two-way and simple authentication—were used. The reasons to this outcome lie in the fact that the authentication protocols pose too big requirements for the WSP and especially for the WSC in comparison to the requirements of the application protocol between them, just as it was discussed in section 5.2. A concrete example of this is discussed in the case example below.

A WSP Case Example The Finnish National Board of Taxes has developed TaMo software that provides checking, fixing, and sending of e.g. tax return forms. The two interfaces, WebTaMo for browsers and ApiTaMo for Web services applications, are introduced in [ApiTaMo]. They both act as service providers and use the Tunnistus.fi[4] as the IDP for authentication.

ApiTamo can be accessed with SOAP and requires authentication in the WSIDP application of Tunnistus.fi. The message flow is quite similar to the three-way authentication presented in figure 3.2 with the addition of SAML attribute query messages between the WSP and Karva attribute authority. The exact ApiTaMo message flow is depicted in [ApiTaMo, page 8].

In the simplest form, the message exchange comprises of 10 messages. Two of the messages apply ID-WSF authentication protocol, four messages SAML authentication request protocol, two messages SAML attribute query protocol, and the last two the ApiTaMo application protocol. If the Katso OTP authentication method is used, it is possible that two additional ID-WSF messages are required to communicate the OTP challenge and the OTP itself. There can also be two subsequent messages for the ApiTaMo protocol.

In ApiTaMo, the authentication plays the biggest role in the message exchange with 6 messages altogether. As of writing, it is unknown, whether there exists any WSCs in production that utilize ApiTaMo according to the presented message flow. If there are ApiTaMo WSCs that do not use WSIDP to authenticate to any other service, it is indeed worth asking: Is this the best way to integrate? Should another authentication procedure be used or perhaps entirely different authentication protocols?

[4] Tunnistus.fi is a joint service for citizen authentication in Finland, run by the Finnish National Board of Taxes, Ministry of Labour, and Social Insurance Institution.

Chapter 7

Discussion

In this chapter, we conclude the thesis by first discussing the present situation of other authentication specifications and the possibilities of the convergence of the standards. Then, we discuss the future needs of our WSIDP implementation and some of the more elaborate use cases the standards enable, namely federation, delegation, and also sharing the session with WebSSO.

7.1 Other Specifications

In section 3.2.2, we mentioned WS-Federation as an alternative protocol for requesting and mediating security tokens. The interest in WS-Federation, and the underlying WS-Trust, originates from the fact that they are the building blocks of the Microsoft Identity Metasystem. In practice, this means that Microsoft's products such as ADFS and CardSpace interoperate with third party software using e.g. WS-Trust for communication.

In fact, Microsoft Internet Explorer 7 features a CardSpace plugin that implements parts of WS-Trust and wss as defined in [InfoCard] and operates as a WSC. The plugin is installed in all Windows XP and Vista machines through the Windows Update.

Although CardSpace is designed to solve the security problems of WebSSO authentication, it is possible to use the same software components to protect Web service applications as presented by Bustamante in [Buso7]. In the article, Bustamante shows how to use the WS-Federation components from the Microsoft Windows Communication Foundation (WCF), which is part of the .NET Framework 3.0. There are a great deal of other WS-* languages covered in WCF in addition to WS-Trust. This underlines the importance of support from big software companies for the success of a XML language specification.

Sun Microsystems builds interoperability software for the most important

ws-* languages in an open-source project called Web Services Interoperability Technology (WSIT)[1]. The goal for Sun is to interoperate with Microsoft WCF.

Convergence of Standards? SAML 2.0 was a big and praised step for convergence of identity specifications (see figure 2.2). It accomplished to unify the extensions defined in Liberty ID-FF and Shibboleth OpenSAML as a specification that would satisfy the needs of both Liberty and Shibboleth. Although the WS-* specifications have utilized SAML 1.x assertions, Microsoft, IBM, et al. have specified the WS-Trust and WS-Federation languages that provide mostly overlapping protocols with SAML.

On March 19th 2007, a draft charter [WSFED-charter] was submitted to establish WSFED Technical Committee (TC) in OASIS for WS-Federation standardization. The charter mentioned SAML as "Similar Work" in the non-normative section, but ignored unfortunately WS-Federation's overlap with SAML 2.0.

Certifying WS-Federation as an OASIS standard as it is would further reduce the possibilities to converge the two existing, major identity specifications, SAML and WS-Federation. Developing and sustaining two incompatible protocols complicate interoperability between different software vendors and cause more than twice the expenses for the implementation and further development of identity software.

It will remain interesting to follow what will be the outcome of the WSFED TC's work and how it will affect on the discussion on protocol convergence.

7.2 Future Work

The next obvious step for Ubilogin WSIDP would be to update the ID-WSF authentication protocol implementation to support the final version. Another interesting future goal would be participating in the Liberty interoperability testing program for SAML and ID-WSF protocols.

Only through interoperability, can complex use scenarios such as federation and delegation be achieved. However, the exact definitions for federation and delegation are not too precise in SAML and Liberty specifications and the exact meaning usually depends on implementation. Implementing such use cases would require further specifications on how to deploy the standardized protocols.

Below, some advanced use cases—which were out of scope in this thesis—are presented.

[1] http://java.sun.com/webservices/interop/

7.2.1 Federation

Liberty defines identity federation as "creating associations between a given system entity's identifiers or accounts" [Liberty-glossary, page 10]. SAML technical overview further refines that "partner services" use federation "across organizational boundaries" [SAML-tech-overview, section 1.1].

As the definitions are not very strict, federation can be understood as a variety of different concepts. In the broadest sense, SSO and all distributed authentication, i.e. when the IDP and SP are separated, can be understood as federation.

In a stricter definition, federation means the distribution of identity information between two or more IDPs. A user with a federated identity can use the same SSO session to access several SPs even though the SPs do not trust the same IDP. The IDPs are required to have a trust relationship and a common protocol for communication, e.g. SAML.

SAML technical overview presents different concepts and use cases of federation in WebSSO [SAML-tech-overview, section 4.4]. The implementation of such use cases is facilitated with HTTP redirects, client redirects (HTTP POST initiated with JavaScript), and the uniform support for cookies in Web browsers. Additionally, this makes federation usually unperceivable for the end user.

Implementing federation in Web services would require more elaborate specifications than what SAML and ID-WSF today provide. Instead of the redirect techniques, PAOS would be needed to accomplish authentication requests between several IDPs. Another, perhaps a more viable, solution would use backchannel for communication between IDPs, similarly with the communication of WSP and WSIDP in the two-way authentication procedure.

7.2.2 Delegation

Whereas federation provides means for a WSC to use SPs in different trust realms; trough delegation, a WSC can access WSPs on behalf of a Principal or another WSC. Delegation means the propagation of identity without the need for a WSP to trust an IDP from another trust realm as long as the IDPs have a trust relationship. An example of delegation can be seen in use cases 1 and 2 of figure 2.1.

Cantor presents the Authentication Request Delegation Profile for SAML in a draft paper [SAML-delegation]. The Liberty ID-WSF Security Mechanisms specify provider chaining to further facilitate complex delegation use cases [ID-WSF-sec, section 7.3.] [ID-WSF-sec-SAML, chapter 6.].

According to Cantor's paper, the SP-a in figure 2.1 would act as the *delegate* SP. This means that it presents the identity of User A to the WSIDP instead of its own identity. This can be accomplished through a holder-of-key subject confirmation in the SAML assertion issued by the IDP.

The subject confirmation "provides the means for a relying party to verify the correspondence of the subject of the assertion with the party with whom the relying party is communicating" [SAML-core, 2.4.1.1].

The SP-a's public key is included in the subject confirmation. SP-a proves the ownership of the key by signing the SAML AuthnRequest message issued by WSP when sending it to the WSIDP. The content of the messages is depicted below.

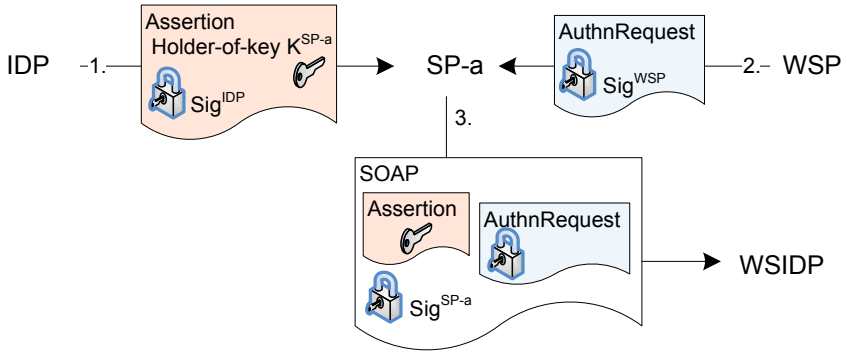

Figure 7.1: Messages and their flow in delegation

This kind of use case is portrayed to be frequent among mobile operators. The operators have a wide customer base as well as existing solutions for identifying the users. Nevertheless, standardized protocols are needed to establish a delegation use case in which an operator provides a portal service that needs to fetch further information from a back-end system that resides possibly in another trust realm. In the telecom world, the operators have formed trust and business relationships in advance to enable roaming, thus facilitating identity federation and delegation.

7.2.3 Shared Session with Websso

Today, many desktop applications, such as Adobe Reader and Microsoft Word, are able to fetch documents from network using simple HTTP GETs. If the documents are situated in a SP where the user has a login session, cookie-based sessions do not propagate to the application and access to the resources is blocked. This problem could be solved by specifying a method for sharing session between ws and Websso and by implementing WSC functionality in the HTTP-aware applications.

However, as noted in section 3.5, the session handling is not specified in SAML nor ID-WSF specifications. This means that all session-related solutions are ultimately implementation specific challenges.

A possible solution for shared session would use persistent cookies, which can be shared between Microsoft Internet Explorer and .NET applications. This solution, however, is rather limited as it is tied to only one platform.

Bibliography

[Abd05]
Alan Abdulrahman. Investigating the Technological and Practical Levels of Maturity of the Industry to Implement Web Services Security. Master's thesis, Royal Institute of Technology, October 2005. http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2005/rapporter05/abdulrahman_alan_05156.pdf, cited 6th May 2007.

[ADFS]
Nick Pierson and Jim Becker. Overview of Active Directory Federation Services in Windows Server 2003 R2, October 2005. http://download.microsoft.com/download/d/8/2/d827e89e-760a-40e5-a69a-4e75723998c5/ADFS_Overview.doc, cited 6th May 2007.

[ApiTaMo]
TietoEnator. WebTaMo & ApiTaMo, Verohallinnon TaMo-palveluiden yleisesittely, November 2006. http://www.vero.fi/nc/doc/download.asp?id=5164;327805, cited 6th May 2007.

[Axis2]
The Apache Software Foundation. Axis2/Java - HTTP transports, January 2007. http://ws.apache.org/axis2/1_1_1/http-transport.html#preemptive_auth, cited 6th May 2007.

[Binder]
Kohsuke Kawaguchi and Joseph Fialli. Binder (JAXB 2.0) Javadoc, March 2006. https://jaxb.dev.java.net/nonav/2.0.5/docs/api/javax/xml/bind/Binder.html, cited 6th May 2007.

[Bus07]
Michèle Leroux Bustamante. Identity: Secure Your ASP.NET Apps And WCF Services With Windows CardSpace, April 2007. http://msdn.microsoft.com/msdnmag/issues/07/04/identity/#S8, cited 6th May 2007.

[dMdSF05]
Emerson Ribeiro de Mello and Joni da Silva Fraga. Mediation of Trust across Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 515–522, 2005. http://ieeexplore.ieee.org/iel5/10245/32665/01530842.pdf, cited 6th May 2007.

[Gav01]
Kirill Gavrylyuk. Building Secure Web Services with Microsoft SOAP Toolkit 2.0, July 2001. http://msdn2.microsoft.com/en-us/library/ms995772.aspx, cited 6th May 2007.

[Gro03]
Thomas Groß. Security Analysis of the SAML Single Sign-on Browser/Artifact Profile. In *Proceedings of the 19th Annual Computer Security Applications Conference (ASAC 2003)*, pages 298–307, 2003. http://ieeexplore.ieee.org/iel5/8882/28060/01254334.pdf, cited 6th May 2007.

[Haa03]
Hugo Haas. Web services: setting and resetting expectations, June 2003. http://www.w3.org/2003/Talks/0618-hh-wschk/, cited 6th May 2007.

[Har03]
Elliotte Rusty Harold. What's wrong with XML APIs (and how to fix them), 2003. http://www.cafeconleche.org/XOM/whatswrong/, cited 6th May 2007.

[HKLP05]
Hal Hildebrand, Anish Karmarkar, Mark Little, and Greg Pavlik. Session Modeling for Web Services. In *Proceedings of the Third IEEE European Conference on Web Services (ECOWS)*, November 2005. http://wscc.info/p51561/files/57-hal.pdf, cited 6th May 2007.

[HTTP-auth]
John Franks, Phillip Hallam-Baker, Jeffery Hostetler, Scott Lawrence, Paul Leach, Ari Luotonen, and Lawrence Stewart, editors. HTTP Authentication: Basic and Digest Access Authentication, June 1999. http://ietf.org/rfc/rfc2617, cited 6th May 2007.

[ID-WSF-authn]
Jeff Hodges, Robert Aarts, Paul Madsen, and Scott Cantor, editors. Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification v2.0, October 2006. http://www.projectliberty.org/liberty/

content/download/871/6189/file/liberty-idwsf-authn-svc-v2.0.pdf, cited 6th May 2007.

[ID-WSF-authn-draft]
Jeff Hodges, Robert Aarts, and Paul Madsen, editors. Liberty ID-WSF Authentication Service and Single Sign-On Service Specification, v2.0 Draft Release 1, November 2004. http://www.projectliberty.org/liberty/content/download/1945/13739/file/draft-liberty-idwsf-authn-svc-v2.0-02.pdf, cited 6th May 2007.

[ID-WSF-client-profiles]
Robert Aarts, Jukka Kainulainen, and John Kemp, editors. Liberty ID-WSF Profiles for Liberty enabled User Agents and Devices, version v2.0, October 2006. http://www.projectliberty.org/liberty/content/download/874/6198/file/liberty-idwsf-client-profiles-v2.0.pdf, cited 6th May 2007.

[ID-WSF-sec]
Gary Ellison, Frederick Hirsch, and Paul Madsen, editors. Liberty ID-WSF Security Mechanisms Core, 2005. http://www.projectliberty.org/liberty/content/download/1980/13854/file/draft-liberty-idwsf-security-mechanisms-core-v2.0-12.pdf, cited 6th May 2007.

[ID-WSF-sec-draft]
Gary Ellison and Paul Madsen, editors. Liberty ID-WSF Security Mechanisms, v2.0 Draft Release 1, November 2004. http://www.projectliberty.org/liberty/content/download/1941/13727/file/draft-liberty-idwsf-security-mechanisms-v2.0-03.pdf, cited 6th May 2007.

[ID-WSF-sec-SAML]
Gary Ellison, Frederick Hirsch, and Paul Madsen, editors. ID-WSF 2.0 SecMech SAML Profile, 2005. http://www.projectliberty.org/liberty/content/download/1981/13857/file/draft-liberty-idwsf-security-mechanisms-saml-profile-v2.0-11.pdf, cited 6th May 2007.

[InfoCard]
Microsoft Corporation and Ping Identity Corporation. A Guide to Integrating with InfoCard v1.0, August 2005. http://download.microsoft.com/download/6/c/3/6c3c2ba2-e5f0-4fe3-be7f-c5dcb86af6de/infocard-guide-beta2-published.pdf, cited 6th May 2007.

[Kaw06]
Kohsuke Kawaguchi. Stateful web service with JAX-WS RI 2.1 EA2, Oc-

tober 2006. http://weblogs.java.net/blog/kohsuke/archive/2006/10/stateful_web_se.html, cited 6th May 2007.

[Levo1]
Jason Levitt. From EDI To XML And UDDI: A Brief History Of Web Services, October 2001. http://www.informationweek.com/story/IWK20010928S0006, cited 6th May 2007.

[Liberty-glossary]
Jeff Hodges, editor. Liberty Technical Glossary, October 2006. http://www.projectliberty.org/liberty/content/download/868/6180/file/liberty-glossary-v2.0.pdf, cited 6th May 2007.

[Liberty-overview]
Conor Cahill. Liberty Technology Overview, August 2006. http://www.projectliberty.org/liberty/content/download/800/5730/file/SpecsOverviewAOL.pdf, cited 6th May 2007.

[Liberty-tech]
Conor Cahill, Carolina Canales-Valenzuela, Frederick Hirsch, Paul Madsen, Prateek Mishra, Rob Philpott, Jeff Smith, Eric Tiffany, and Greg Whitehead. Liberty Technology Tutorial, June 2005. http://www.projectliberty.org/resources/LibertyTechnologyTutorial.pdf, cited 7th March 2006.

[Liberty-testing]
Eric Tiffany. Liberty Conformance Testing Tips, May 2005. http://www.projectliberty.org/liberty/content/download/987/6952/file/LibConfTestingTips-v09.pdf, cited 6th May 2007.

[Malo6]
Jussi Malinen. Windows CardSpace. In *Security and Privacy in Pervasive Computing, Seminar on Network Security*. Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, December 2006. http://www.tml.tkk.fi/Publications/C/22/papers/Malinen_final.pdf, cited 6th May 2007.

[Nyko2]
Toni Nykänen. Secure Cross-Platform Single Sign-On Solution for the World-Wide Web. Master's thesis, Helsinki University of Technology, May 2002. http://users.tkk.fi/~tpnykane/thesis/tpnykane_2002.pdf, cited 6th May 2007.

[Ojao5]
Oskar Ojala. Service Oriented Architecture in Mobile Devices: Protocols

and Tools. Master's thesis, Helsinki University of Technology, November 2005. http://www.kolumbus.fi/oskar.ojala/comp/oskarojala_soamobiledevices.pdf, cited 6th May 2007.

[Pulo6]
Rama Pulavarthi. Maintaining Session With JAX-WS, June 2006. http://weblogs.java.net/blog/ramapulavarthi/archive/2006/06/maintaining_ses.html, cited 6th May 2007.

[Reio2]
Greg Reinacker. Web Services Security, June 2002. http://www.rassoc.com/gregr/weblog/stories/2002/06/09/webServicesSecurity.html, cited 6th May 2007.

[SAML-authn-context]
John Kemp, Scott Cantor, Prateek Mishra, Rob Philpott, and Eve Maler, editors. Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf, cited 6th May 2007.

[SAML-conformance]
Prateek Mishra, Rob Philpott, and Eve Maler, editors. Conformance Requirements for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-conformance-2.0-os.pdf, cited 6th May 2007.

[SAML-core]
Scott Cantor, John Kemp, Rob Philpott, and Eve Maler, editors. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, cited 6th May 2007.

[SAML-delegation]
Scott Cantor, editor. SAML 2.0 Single Sign-On with Constrained Delegation, Working Draft 01, October 2005. http://shibboleth.internet2.edu/docs/draft-cantor-saml-sso-delegation-01.pdf, cited 6th May 2007.

[SAML-profiles]
John Hughes, Scott Cantor, Jeff Hodges, Frederick Hirsch, Prateek Mishra, Rob Philpott, and Eve Maler, editors. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf, cited 6th May 2007.

[SAML-sec-consider]
Frederick Hirsch, Rob Philpott, and Eve Maler, editors. Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf, cited 6th May 2007.

[SAML-tech-overview]
Nick Ragouzis, John Hughes, Rob Philpott, and Eve Maler, editors. Security Assertion Markup Language (SAML) V2.0 Technical Overview, Working Draft 10, October 2006. http://www.oasis-open.org/committees/download.php/20645/sstc-saml-tech-overview-2%200-draft-10.pdf, cited 6th May 2007.

[SASL]
Alexey Melnikov and Kurt Zeilenga, editors. Simple Authentication and Security Layer (SASL), June 2006. http://ietf.org/rfc/rfc4422, cited 6th May 2007.

[SOAP-part1]
Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen, editors. SOAP Version 1.2 part 1: Messaging framework, June 2003. http://www.w3.org/TR/2003/REC-soap12-part2-20030624/, cited 6th May 2007.

[SOAP-part2]
Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen, editors. SOAP Version 1.2 part 2: Adjuncts, June 2003. http://www.w3.org/TR/2003/REC-soap12-part2-20030624/, cited 6th May 2007.

[WL92]
Thomas Y. C. Woo and Simon S. Lam. Authentication for Distributed Systems. *Computer*, 25(1):39–52, 1992. http://dx.doi.org/10.1109/2.108052, cited 6th May 2007.

[WS-arch]
David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard, editors. Web Services Architecture, February 2004. http://www.w3.org/TR/ws-arch/, cited 6th May 2007.

[WS-CTX]
Mark Little, Eric Newcomer, and Greg Pavlik, editors. Web Services Context Specification (WS-Context), Committee draft version 0.8, October 2004.

http://www.oasis-open.org/committees/download.php/9806/9, cited 6th May 2007.

[WSA]
Martin Gudgin, Marc Hadley, and Tony Rogers, editors. Web Services Addressing 1.0 - Core, May 2006. http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/, cited 6th May 2007.

[WSDL]
Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, editors. Web services description language (WSDL) 1.1, March 2001. http://www.w3.org/TR/wsdl, cited 6th May 2007.

[WSDL2-adjuncts]
Roberto Chinnici, Hugo Haas, Amelia A. Lewis, Jean-Jacques Moreau, David Orchard, and Sanjiva Weerawarana, editors. Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts, March 2007. http://www.w3.org/TR/wsdl20-adjuncts/, cited 6th May 2007.

[WSF]
Chris Kaler and Anthony Nadalin, editors. Web Services Federation Language (WS-Federation), July 2003. ftp://www6.software.ibm.com/software/developer/library/ws-fed.pdf, cited 6th May 2007.

[WSF-ARP]
Chris Kaler and Anthony Nadalin, editors. WS-Federation: Active Requestor Profile, July 2003. ftp://www6.software.ibm.com/software/developer/library/ws-fedact.pdf, cited 6th May 2007.

[WSF-PRP]
Chris Kaler and Anthony Nadalin, editors. WS-Federation: Passive Requestor Profile, July 2003. ftp://www6.software.ibm.com/software/developer/library/ws-fedpass.pdf, cited 6th May 2007.

[WSFED-charter]
Mary McRae. Proposed Charter for OASIS Web Services Federation (WSFED) TC, March 2007. http://www.oasis-open.org/archives/oasis-charter-discuss/200703/msg00002.html, cited 6th May 2007.

[WSS]
Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Philip Hallam-Baker, editors. Web Services Security: SOAP Message Security 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, cited 6th May 2007.

[WSS-SAML]
Ronald Monzillo, Chris Kaler, Anthony Nadalin, and Philip Hallam-Baker, editors. Web Services Security: SAML Token Profile 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf, cited 6th May 2007.

[WST]
Martin Gudgin and Anthony Nadalin, editors. Web Services Trust Language (WS-Trust), February 2005. ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf, cited 6th May 2007.

[XKMS]
Phillip Hallam-Baker and Shivaram H. Mysore, editors. XML Key Management Specification (XKMS 2.0), June 2005. http://www.w3.org/TR/xkms2/, cited 6th May 2007.

[XKMS-does]
Rich Salz. XKMS does the heavy work of PKI, September 2003. http://www.networkworld.com/news/tech/2003/0908techupdate.html, cited 6th May 2007.

[XML]
Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau, editors. Extensible Markup Language (XML) 1.0 (Third Edition), February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/, cited 6th May 2007.

[XML-schema]
Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn, editors. XML Schema Part 1: Structures Second Edition, October 2004. http://www.w3.org/TR/xmlschema-1/, cited 6th May 2007.

[XMLDSig]
Donald Eastlake, Joseph Reagle, and David Solo, editors. XML-Signature Syntax and Processing, February 2002. http://www.w3.org/TR/xmldsig-core/, cited 6th May 2007.

[XMLDSig-intro]
Ed Simon, Paul Madsen, and Carlisle Adams. An Introduction to XML Digital Signatures, August 2001. http://www.xml.com/pub/a/2001/08/08/xmldsig.html, cited 6th May 2007.

Appendix A

The Katso SASL Mechanism

Yrjö Kari-Koskinen
Ubisecure Solutions Inc.
November 2005

The Katso SASL Mechanism

Abstract

Katso SASL mechanism is a strong authentication method which is based on a used id, password and one time password. This specification defines the KATSO SASL mechanism which can easily be implemented in various applications.

1. How to Read This Document

The key words ”MUST”, ”MUST NOT”, ”REQUIRED”, ”SHOULD”, ”SHOULD NOT”, ”RECOMMENDED” and ”MAY” in this document are to be interpreted as defined in ”Key words for use in RFCs to Indicate Requirement Levels” [KEYWORDS].

This document assumes the reader is familiar with SASL [SASL].

2. Intended Use

The KATSO SASL mechanism extends the [PLAIN] SASL mechanism and provides thus a stronger mechanism for use in applications where the plain mechanism is not secure enough.

This SASL mechanism provides means for integrating KATSO into the SASL-enabled ID-WSF Authentication Service [ID-WSF-AUTHN].

3. Authentication Procedure

The mechanism consists of one or two client messages with their corresponding server responses. All client messages MUST contain three fields separated by a US-ASCII NUL character:

1) the authentication identity (identity whose password will be used)

2) the clear-text password

3) the one time password (OTP)

Upon receiving a message from client the server MUST proceeded as follows:

a) The server will verify the authentication identity from the system authentication database.

b) If the verification did not succeed, the server will abort the mechanism.

c) If the authentication identity verification succeeds, the server tries to verify the password and the OTP.

d) If the password and the OTP are correct, the server will end the mechanism with an ”OK” message signaling that the user is now logged in.

e) If the password is not correct, the password is missing, the OTP is missing, or the OTP is not correct the server MUST choose between these two actions:

- it will either send a challenge that contains the serial number of the correct OTP
- or it will abort the mechanism.

f) In case the server send a challenge message to the client, the

client SHOULD respond with another three fielded message and the server MUST begin the processing at step a).

3.1. Character sets

US-ASCII printable characters are preferred, although UTF-8 [UTF-8] printable characters are permitted to support international names. Use of character sets other than US-ASCII and UTF-8 is forbidden.

4. Examples

4.1. Plain Text Style

Here is an authentication example in a plain text style containing correct authentication identity, password and OTP fields in the client message:

```
C: AUTHENTICATE ”KATSO”
C: john<NUL>verysecret<NUL>001122
S: OK
```

In the next example the password field is correct but the OTP field is empty.

```
C: AUTHENTICATE ”KATSO”
C: mary<NUL>alsosecret<NUL>
S: 31
C: mary<NUL>alsosecret<NUL>923487
S: OK
```

4.1. SASLRequest Style

Here the latter example is given as the XML-data which is used in the SASLRequests and SASLResponses of [ID-WSF-AUTHN]. Note that [ID-WSF-AUTHN] requires for the data to be base64-encoded.

```
Client:
<sa:SASLRequest xmlns:sa=”urn:liberty:sa:2004-04”
 sa:mechanism=”KATSO”>
   <Data>bWFyeQBhbHNvc2VjcmV0AA==</Data>
</sa:SASLRequest>
```

```
Server:
<sa:SASLResponse xmlns:sa="urn:liberty:sa:2004-04"
 sa:serverMechanism="KATSO">
   <sa:Status code="Continue"/>
   <Data>MzE=</Data>
</sa:SASLResponse>

Client:
<sa:SASLRequest xmlns:sa="urn:liberty:sa:2004-04"
 sa:mechanism="KATSO">
   <Data>bWFyeQBhbHNvc2VjcmV0ADkyMzQ4Nw==</Data>
</sa:SASLRequest>

Server:
<sa:SASLResponse xmlns:sa="urn:liberty:sa:2004-04"
 sa:serverMechanism="KATSO">
   <sa:Status code="OK"/>
</sa:SASLResponse>
```

5. References

[ID-WSF-AUTHN] Liberty ID-WSF Authentication Service and Single Sign-On Service Specification, draft release 1 <http://www.projectliberty.org/liberty/content/download/1945/13739/file/draft-liberty-idwsf-authn-svc-v2.0-02.pdf>

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

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

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.