



2
3
4
5
6
7
8
9
10

Single Sign-On Toolkit

sponsored by

The National Association of
REALTORS®
Center for REALTOR
Technology

11
12
13

Revision 1 – 29 May 2007

Table of Contents

Introduction.....	4
History.....	4
Concepts.....	5
SAML 2.0.....	5
Identity Provider (IdP).....	5
Service Provider (SP).....	6
Assertions.....	6
Protocols.....	6
Bindings.....	7
Profiles.....	7
Metadata.....	8
Public Key Cryptography.....	8
Signatures.....	9
Web SSO.....	9
HTTP Redirect.....	11
HTTP POST.....	12
HTTP Artifact.....	12
Toolkit.....	12
Java.....	13
Replacement Parser and Endorsed Directories.....	13
Dependent Libraries.....	13
Building OpenSAML.....	14
Clareity SSO Framework.....	14
idp.....	15
metadata.....	16
sp.....	16
C# (.NET).....	17
Glossary.....	18
Appendix A - Resources.....	18
Index.....	18

16 **Introduction**

17 The Clarity Security Single Sign-On (SSO) toolkit allows software providers to integrate support for
18 SSO into their applications in an industry standard manner. The toolkit provides a reference
19 implementation to show developers exactly how to use the SAML 2.0 standards to accomplish SSO.
20 More importantly, the toolkit explains the concepts behind the standard. The developer is not expected to
21 read voluminous and dry standards documents in order to learn. Those standards documents become the
22 reference for the toolkit, something to look at when the exact syntax is needed.

23 This toolkit builds upon work by many other organizations, most notably the Internet2 project and their
24 OpenSAML [OpenSAML] toolkit. A complete list of works used and references are available in
25 Appendix A.

26 Finally, Clarity Security would like to thank NAR's Center for REALTOR Technology (CRT), and
27 Mark Lesswing in particular, for providing a grant to make this toolkit possible.

28 **History**

29 Real estate professionals are using more systems and applications than ever, and they don't want to have
30 to log into each one separately. The inconvenience and inefficiency of multiple logins are exacerbated
31 when users have to go back and forth between one system and another. As a result, system providers such
32 as MLSs, transaction management software, larger brokerages and other real estate application vendors
33 have moved to provide login integration of commonly used systems as a convenience for the users. This
34 is referred to as Single Sign-On (SSO).

35 In order to improve the security and efficiency of implementing SSO, it was critical that the industry's
36 major software vendors agree on a standard method for doing so. This is why in August of 2005 Clarity
37 Consulting published a free white paper, "The Convenience and Security of Single Sign-On" and
38 subsequently facilitated two meetings, in December 2005 and March 2006, where many of the major
39 MLS and transaction management vendors met to discuss how to achieve SSO in the real estate industry.
40 MLS vendor attendees included Rob Overman, CTO of Fidelity MLS, Dan Mills, CTO of Interealty
41 (First American MLS), Chip McAvoy, CTO at First American Residential Group (MarketLinx), Mike
42 Wurzer, CEO/CTO of FBS, Carlos Grass, CEO of Stratus Data Systems, Brian de Shepper, CEO of
43 Tarasoft, John Hensley, CTO of Homeseekers, and Brett Weiner, CTO/EVP of Rapattoni MLS. It was
44 critical to achieve cross MLS vendor support and cooperation, because while MLS may not be the center
45 of the universe, imagine the decreased value of SSO without MLS system integration. In addition to the
46 key MLS software leaders, Cendant, CREA (Canadian Real Estate Association), OSCRE (the
47 international real estate standards group), and other software companies in the Transaction Management,
48 CRM, neighborhood information and online mapping areas also attended and contributed to these
49 meetings. Paul Stusiak, technical co-chair of the RETS group, attended and represented the RETS
50 working group.

51 This diverse and talented group of technical leaders in the real estate industry came to a consensus that:

- 52 ● Using an open standard such as SAML 2.0 will allow the vendors to cooperate—making it more
53 efficient to implement SSO securely.
- 54 ● There was no agreement on, or even serious consideration to, using a proprietary product to
55 accomplish SSO.
- 56 ● No single entity or group should try to monetize SSO or control it by making it proprietary
57 because this will create conflict and control issues and be a non-starter for many organizations

58 and likely to slow adoption of free, open standards based SSO. (Microsoft Passport was discussed
59 as an example of why a controlling or proprietary approach is likely to fail. Today, even
60 Microsoft is backing open identity management and SSO standards as Bill Gates mentioned
61 during his keynote address at the RSA Conference in February 2007)

- 62 ● Clarity and the vendor group should work with the NATIONAL ASSOCIATION OF
63 REALTORS® (NAR) to incorporate support for SAML into the existing standard for real estate
64 software vendor cooperation – the Real Estate Transaction Standard (RETS).

65 With all of this in mind, Clarity put together a proposal with CRT to build a reference implementation
66 that the entire real estate industry could use. This toolkit is the result.

67 **Concepts**

68 **SAML 2.0**

69 The Security Assertion Markup Language 2.0 is a set of standards produced by OASIS (Organization for
70 the Advancement of Structured Information Standards) [OASIS]. Within OASIS, the Security Services
71 Technical Committee is the group responsible for SAML. A succinct description from their website:

72 *SAML, developed by the Security Services Technical Committee of OASIS, is an XML-based*
73 *framework for communicating user authentication, entitlement, and attribute information. As its*
74 *name suggests, SAML allows business entities to make assertions regarding the identity,*
75 *attributes, and entitlements of a subject (an entity that is often a human user) to other entities,*
76 *such as a partner company or another enterprise application.¹*

77
78 Several different documents have been produced by OASIS that are needed to accomplish SSO:

- 79 ● Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0
- 80 ● Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0
- 81 ● Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0
- 82 ● Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0
- 83 ● Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0
- 84 ●
- 85

86 While this document will not discuss each document specifically, it will discuss information from all of
87 them in the context of implementing SSO.

88 **Identity Provider (IdP)**

89 An Identity Provider (IdP) is the entity responsible for authenticating a user. In the real estate market, this
90 role is customarily provided by each individual application that an end user accesses. Examples of this
91 would be the Multiple Listing Service (MLS) system, the public records systems, or the forms system.
92 Most of these systems maintain their own database of user identities and authenticating credentials (e.g.
93 passwords, certificate stores, or one-time password systems). How the IdP authenticates a user is
94 orthogonal to SSO. The participants in the SSO environment typically do not care or need to know how a
95 user authenticates themselves to the IdP, although the SAML standards do allow this information to be
96 transmitted to interested parties.

1 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

97 In the SAML environment, an IdP is the key to performing SSO. The IdP will provide assertions to the
98 SSO enabled applications. These assertions are constructed in such a way that the consuming application
99 can validate their authenticity and grant access to resources (applications).

100 **Service Provider (SP)**

101 A service provider is simply an application providing some service or resources to an end user. This
102 would be applications such as the Multiple Listing Service (MLS), transaction management, or
103 membership management systems. Service Providers are requesters and consumers of SAML objects.

104 When an end user makes an initial request to the SP for resources, the SP will redirect the user's web
105 browser to an IdP for authentication of that end user. The SP will send a request to the IdP asking for the
106 user to be authenticated and providing the web page for the IdP to redirect the end user to once
107 authentication is done.

108 Please note that it is possible for a SP to also function as an IdP. Logically they are different functions,
109 but the same application can provide both services. Essentially any web application requiring
110 authentication is already serving both functions.

111 **Assertions**

112 Assertions help form the basis of the SAML specification. SAML assertions are simply XML objects
113 which supply information about an entity. These assertions convey three classes of information:

- 114 ● Authentication Statements
- 115 ● Attribute Statements

116 Authentication statements are created by the IdP that authenticated the end user. They will provide
117 information to identify the end user, to state when the authentication took place, to state how long the
118 assertion is valid for, and how the user was authenticated. An example would be that the user is *Joe*, he
119 was authenticated at 4:12 PM on 5/3/2007, and the authentication is valid until 4:17 PM on 5/3/2007.

120 An attribute statement describe various characteristics about the authenticated user. *Joe* is the designated
121 broker of his firm is an attribute statement.

122 This information is conveyed by the IdP to the SP. In SAML terms, the IdP is the *asserting party* while
123 the SP is the *relying party*. The IdP asserts the information it sends to the SP is accurate and valid. The
124 SP relies on the information provided by the IdP.

125

126 **Protocols**

127 Protocols are the second leg on which the SAML specifications are built. Protocols define the requests
128 and responses between Identity Providers and Service Providers. Several generalized protocols are
129 defined in SAML:

- 130 ● Authentication Request Protocol
- 131 ● Single Logout Protocol
- 132 ● Assertion Query and Request Protocol
- 133 ● Artifact Resolution Protocol

- 134 ● Name Identifier Mapping Protocol
 - 135 ● Name Identifier Management Protocol
- 136 The Authentication Request Protocol is the primary protocol of interest when implementing SSO. This
137 protocol allows a SP to request an authentication assertion about a particular user from a IdP.
- 138 The Single Logout Protocol allows the near simultaneous logout of a user from multiple SPs.
- 139 The Assertion Query and Request Protocol is used to obtain assertions. Requests can be made for existing
140 assertions based on the ID of the assertion. Queries can be made for new or existing assertions based on
141 the subject and statement types.
- 142 Name Identifier Mapping Protocol provides the mechanism to map user identities between an SP and an
143 Idp. This allows disparate ID values between systems to be mapped. A user could be known as *Joe* on
144 one system and as *Joseph* on another. A mapping is made so that each system uses their own ID
145 internally while still uniquely identifying a user between systems.
- 146 Name Identifier Management Protocol allows existing name identifier mappings to be changed. The IdP
147 or the SP can originate the change.

148 **Bindings**

149 Bindings is the term used to describe how the SAML protocols are carried over the underlying transport
150 mechanism such as HTTP or SOAP. This allows a standard way to effectively package the information
151 so that independent implementations can understand one another. SAML defines the following bindings:

- 152 ● HTTP Redirect Binding
- 153 ● HTTP POST Binding
- 154 ● HTTP Artifact Binding
- 155 ● SAML SOAP Binding
- 156 ● Reverse SOAP (PAOS) Binding
- 157 ● SAML URI Binding

158 HTTP Redirect Binding specifies how to transport protocol messages using the HTTP 302 redirect
159 response.

160 HTTP POST Binding specifies how to transport protocol messages as BASE64 encoded data within a
161 standard HTML form that is sent via the HTTP POST method.

162 HTTP Artifact Binding allows the use of artifacts (small opaque identifiers) in transporting messages.
163 The binding has specifications for both HTML forms and HTTP 302 redirects.

164 SAML SOAP Binding defines how to transport protocol messages using SOAP 1.1 over HTTP.

165 PAOS Binding is used primarily by WAP gateways to allow an HTTP client to be a SOAP responder.

166 SAML URI Binding specifies how to retrieve an existing SAML assertion by resolving a URI.

167 **Profiles**

168 SAML profiles are where the rubber meets the road. A profile brings together assertions, protocols, and
169 bindings in concrete use cases. Constraints have been introduced in profiles to promote interoperability

170 between implementations. The profiles defined by SAML include:

- 171 ● Web Browser SSO Profile
- 172 ● Enhanced Client and Proxy (ECP) Profile
- 173 ● Identity Provider Discovery Profile
- 174 ● Single Logout Profile
- 175 ● Assertion Query/Request Profile
- 176 ● Artifact Resolution Profile
- 177 ● Name Identifier Management Profile
- 178 ● Name Identifier Mapping Profile

179 The Web Browser SSO Profile is the main profile we are interested in. It specifies how you can use a
180 standard web browser to perform SSO making use of the Authentication Request Protocol, SAML
181 Response messages and Assertions.

182 Enhanced Client and Proxy Profile is used in situations where a client cannot receive a request such as a
183 cell phone. However the cell phone can send one and effectively run SOAP over HTTP backward.

184 A SP can use the Identity Provider Discovery Profile to discover identity providers the user has visited.

185 The Single Logout Profile allows the user to *logout* of a group of service providers associated with the
186 identity provider the user authenticated with. This effects a near simultaneous logout from all service
187 providers the user has visited during their session.

188 The Assertion Query and Request Profile defines how SAML providers can obtain assertions over SOAP.

189 The Artifact Resolution Profile defines how the Artifact Resolution Protocol is used over SOAP to
190 request the SAML protocol message referenced by the artifact.

191 The Name Identifier Management Profile defines how to use the Name Identifier Management Protocol
192 over SOAP, HTTP POST, HTTP Redirect, and HTTP Artifact Bindings.

193 The Name Identifier Mapping Profile defines how to use the Name Identifier Mapping Protocol over
194 SOAP.

195 **Metadata**

196 SAML Metadata provides a standardized way to express the information Service and Identity Providers
197 need in order to interoperate. This includes information on the profiles that an IdP or SP support, the
198 location of those profiles, and cryptographic key information. An example of information provided in the
199 metadata would be whether an IdP supports the Single Logout Profile and what the URL is to request a
200 single logout.

201 **Public Key Cryptography**

202 Public Key Cryptography or asymmetric cryptography is a form of cryptography where two
203 complementary keys exist. One is a public key and one is a private key. Compare this to symmetric
204 cryptography, where a single key is used to encrypt the data and the same key is used to decrypt the data,
205 hence keeping the key secret is critical. With public key cryptography, two mathematically related keys
206 are used. Data encrypted with one key can only be decrypted with the other key. Because of this property,

207 one key is considered the public key and can be freely distributed. The other key is considered the private
208 key and is kept secret.

209 In use, assume that Adam and Bonnie need to exchange private messages. Both Adam and Bonnie have
210 created a set of public/private keys and exchanged their public keys with one another. Now when Adam
211 needs to send Bonnie a message, he takes Bonnie's public key and encrypts the message before he sends
212 it. Once Bonnie receives the message, she uses her private key to decrypt the message. Since the private
213 key is required to decrypt the message, only Bonnie can decrypt it. Even if the message were intercepted
214 in transit, it would provide nothing to the person intercepting the message.

215 **Signatures**

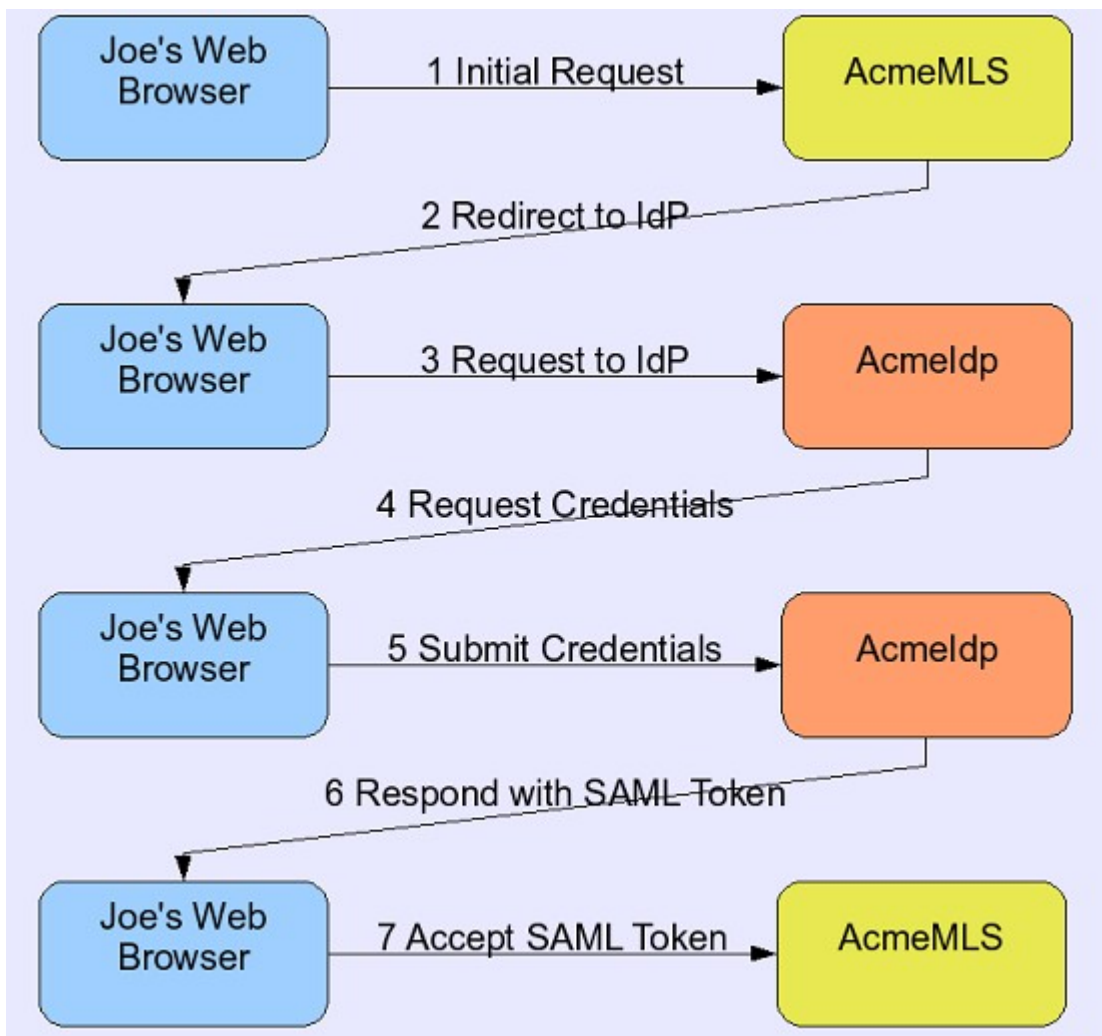
216 Digital signatures take the concept of public key cryptography and extend it to a practical use: how do I
217 prove to you that the message I sent you is from me? The concept is fairly straight forward, I can encrypt
218 the message with my private key, you decrypt it using my public key. If the message is decrypted
219 successfully, it must be from me. The drawback to encrypting the entire message is that asymmetric
220 encryption is slow (compared to symmetric encryption). Instead of encrypting the message with the
221 private key, we instead encrypt a hash of the message. The benefit of hashing the message is that the
222 output size is small and one cannot easily discover the message contents from the hash. Hashing is also
223 very fast. So now our process is to hash our message and then encrypt the hash value using the private
224 key. On the receiving side, the encrypted value is decrypted using the public key to obtain the calculated
225 hash. Then the message is hashed by the receiver so that the two hash values can be compared. If they are
226 the same, then the sender did send the message.

227 **Web SSO**

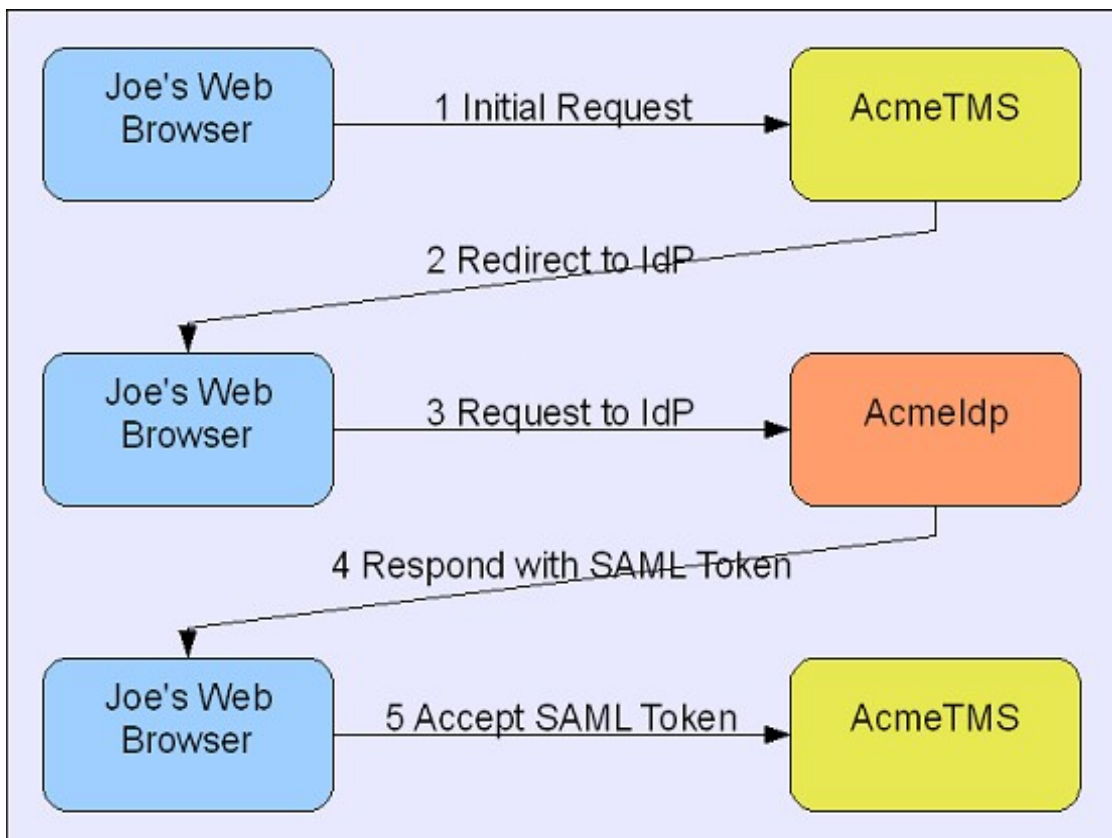
228 Now that we have a basic understanding of the building blocks involved, it is time to put the pieces
229 together and build something useful. Web SSO allows a user to authenticate once and then access
230 separate web applications that participate in the SSO federation. In the real estate world, the primary use
231 case is that of a MLS organization which provides multiple applications to its members: MLS, TMS,
232 public records, forms, etc. The SAML Web SSO Profile provides us the instructions and standards to
233 build inter operable applications to accomplish SSO.

234 There are three bindings specified by this profile: HTTP Redirect, HTTP POST, and HTTP Artifact. To
235 discuss how these work, let's look at the following scenario. Real estate agent *Joe* opens his web browser
236 and picks his bookmark for *AcmeMLS*. When Joe's browser makes the request for *AcmeMLS*, *AcmeMLS*
237 finds no active session for Joe and so sends Joe's browser to *AcmeIdp* for authentication. *AcmeIdp*
238 requests login credentials from Joe and validates his identity. Once this is accomplished, *AcmeIdp* creates
239 a session for Joe. The exact mechanism to maintain the session is not specified by the SAML standard, it
240 is assumed that the normal method which applies to the web application server used by *AcmeIdp* will be
241 used. Now that the session is established, *AcmeIdp* redirects Joe's browser back to *AcmeMLS* with the
242 SAML authentication credentials. *AcmeMLS* receives the authentication credentials, validates them, and
243 then establishes its own session for Joe.

244



245 After doing his work on AcmeMLS, Joe decides he needs to visit *AcmeTMS* to complete some
 246 paperwork. He clicks on the link in AcmeMLS to go to AcmeTMS. Once he reaches AcmeTMS,
 247 AcmeTMS redirects Joe to AcmeIdp since Joe did not have an active session with AcmeTMS. However,
 248 AcmeIdp finds Joe's active session and create a new SAML authentication token and redirects Joe's
 249 browser back to AcmeTMS. Joe is not required to authenticate again and only sees his browser flip
 250 through a couple of web pages. Now Joe is logged into AcmeTMS with an active session.



247 **HTTP Redirect**

248 The first binding allowed by the Web SSO Profile uses the HTTP 302 redirect status code. Using our
 249 example, when Joe makes that initial request to AcmeMLS, the AcmeMLS server will construct an
 250 Authentication Request Protocol SAML object. This is simply an XML data structure specified by the
 251 SAML standard which will include the issuing entity (AcmeMLS), and the location to send the response
 252 to (URL). Optionally, AcmeMLS may include a Relay State variable. This is an opaque value to anyone
 253 but AcmeMLS. The IdP is required to return the value to AcmeMLS if AcmeMLS sends it. By standard,
 254 the Relay State value is limited to 80 bytes of data.

255 Once the XML is built, it is compressed using the DEFLATE algorithm [RFC1951], and then encoded
 256 using BASE64 [BASE64]. It is then added to the HTTP Location header as a query string parameter
 257 along with the Relay State value (if needed). The actual URL value for the Location header will be the
 258 page specified by the IdP in its metadata. An example Location header would look like this:

259 `https://www.acmeidp.com/sso/request-auth.jsp?SAMLRequest=abcd1234&RelayState=efgh5678`

260 The values shown in the query string are examples only, the actual encoded values would be used in
 261 practice.

262 On the receiving end, AcmeIdp would BASE64 decode the query string and then uncompress the value
 263 according to the DEFLATE algorithm. It would then request the login credentials from Joe and construct
 264 a SAML Response object showing Joe had been properly authenticated. It would then follow the same
 265 steps to DEFLATE and BASE64 encode its response. Again, it would use the HTTP 302 Location header
 266 to send Joe's browser back to AcmeMLS with the encoded SAML Response object. AcmeMLS would

267 reverse the encoding and compression and process the Response.

268 **HTTP POST**

269 The second binding uses the HTTP POST method to submit the same information as the Redirect binding
270 using a HTML form with hidden form fields. AcmeMLS would construct the Authentication Request
271 Protocol package as before, but would BASE64 encode the value and place it in a form field using the
272 form field variable name of *SAMLRequest*. Similarly, the Relay State value would be placed in a hidden
273 form field with the name of *RelayState*. AcmeMLS then would return a web page to Joe's browser which
274 has no visible content. It would only contain the HTML form with the two hidden form fields. The action
275 for the form would be the location of the AcmeIdp SSO authentication service as specified in its
276 metadata. Typically using a bit of client side Javascript, Joe's browser would automatically submit the
277 form to the IdP. This method can be used successfully with larger sizes of data without fear of
278 implementations truncating or not accepting large query strings.

279 On the receiving side, AcmeIdp would BASE64 decode the values and act upon them as in the HTTP
280 Redirect binding. To send it's response back to AcmeMLS, it would build a web page containing a
281 HTML form with two hidden fields: *SAMLResponse* and *RelayState*. Again, client side Javascript would
282 be used to automatically submit the form to AcmeMLS for Joe. AcmeMLS would receive the POSTed
283 data and act accordingly.

284 **HTTP Artifact**

285 The final binding passes a small value between the SP and IdP called an artifact. It can be sent via a
286 HTTP 302 redirect method or HTTP POST using a HTML form. The difference here is that the actual
287 SAML objects are not passed through an intermediary (Joe's browser). Instead the artifact is used by the
288 SP and IdP to request the SAML objects from one another directly. This mandates that the SP and IdP
289 must be able to communicate directly with one another. The artifact is a key used to request the
290 information from the other party.

291 Using this bind, AcmeMLS creates the Authentication Request Protocol object and stores it in some form
292 of persistent storage. It also creates an artifact value which will allow it to retrieve that object when
293 needed. The artifact is sent to AcmeIdp (using either redirect or form field). AcmeIdp then initiates a
294 direct query to AcmeMLS using the Artifact Resolution Profile to retrieve the SAML object. It would
295 then process the SAML object as previously described and prepare its Response object.

296 AcmeIdp would then send the response back to AcmeMLS using its own artifact using either the redirect
297 or POST binding via Joe's browser. AcmeMLS would then use Artifact Resolution to obtain the SAML
298 Response object directly from AcmeIdp.

299 **Toolkit**

300 The SSO toolkit is not a static work, it is a community project for the real estate industry. Clarity is
301 spearheading and coordinating this project and will be providing access to a shared source code
302 repository for both checkout of the project files and for community contributions.

303 The overall form of the toolkit is meant to make it easy for the web application developer to incorporate
304 SSO. Classes are laid out based on the functions needed to support SSO. Examples of each of the
305 functional parts of a working system are provided to show how to use the classes.

306 Java

307 **Replacement Parser and Endorsed Directories**

308 The first prerequisite for building the OpenSAML libraries is to obtain a replacement XML parser. The
309 parser shipped by Sun in the JRE has bugs in some areas that OpenSAML needs. To work around this
310 you need to download and install a replacement parser: Xalan and Xerces from the Apache project. The
311 project main web pages can be found at:

312 <http://xalan.apache.org>

313 <http://xerces.apache.org>

314 This toolkit uses Xalan 2.7.0 and Xerces 2.9.0. Download and unpack these versions (or greater). You
315 will need the following files:

- 316 ● serializer.jar
- 317 ● xalan.jar
- 318 ● xercesImpl.jar

319 To make use of these libraries, you will need to use the Java endorsed directories. The endorsed
320 directories allows you to replace certain base libraries shipped with the JRE/JDK environment. We
321 recommend that you make a copy of your current JDK directory structure and rename it to include
322 “endorsed” in the directory name. This will allow you to preserve your base Java install in case it is
323 needed (an example of this is Netbeans, it chokes on a bug in Xalan/Xerces). In your <jre>/lib directory,
324 create a directory named “endorsed”. Inside this directory, place the files from Xalan and Xerces. Finally,
325 depending on your operating system and environment, you will need to tell the build environment for
326 OpenSAML to use the endorsed copy of Java. Then the next time that a Java virtual machine is started, it
327 will use the replacement parsers.

328 **Dependent Libraries**

329 The OpenSAML project depends on several open source libraries. You will need to download and make
330 the following libraries available for your build environment.

Library Name	Web Location	JAR
Velocity	http://velocity.apache.org	velocity-1.4.jar
Javolution	http://www.javolution.org	javolution.jar
Bouncy Castle	http://www.bouncycastle.org	bcprov-jdk15-134.jar
Joda Time	http://joda-time.sourceforge.net	joda-time-1.4.jar
Log4j	http://logging.apache.org	log4j-1.2.14.jar
Jakarta Commons HTTP Client	http://jakarta.apache.org/commons/httpclient	commons-httpclient-3.0.1.jar
XML Security	http://xml.apache.org/security	xmlsec-1.3.0.jar

331

332 The version numbers listed are known to work. You may install and use the latest versions unless
333 specifically directed to use a particular version.

334 **Building OpenSAML**

335 The first step here is to checkout the OpenSAML libraries using a Subversion client [Subversion]. The
336 libraries are still under active development so it is necessary to build them.

337 There are three different libraries which compose the OpenSAML project:

338 opensaml.jar

339 openws.jar

340 xmltooling.jar

341 The links to checkout the code are: (as of August 2007)

342 <https://svn.middleware.georgetown.edu/java-opensaml2/trunk>

343 <https://svn.middleware.georgetown.edu/java-openws/trunk>

344 <https://svn.middleware.georgetown.edu/java-xmltooling/trunk>

345 Be sure to use these links with your Subversion client, not with a web browser. The “trunk” contains the
346 latest code. The typical checkout command is:

347 svn co https://svn.middleware.georgetown.edu/java-opensaml2/trunk java-opensaml2

348 where **co** is the checkout command and **java-opensaml2** is the directory to place the checked out files in.

349 The first library to build is xmltooling.jar in the java-xmltooling project. From a command line shell,
350 simply run the ant.sh or ant.bat script file (as appropriate). It will produce the xmltooling.jar file in the
351 **dist** directory. The jar file will have a date stamp incorporated in the name.

352 For the next library, openws.jar, you will need to copy the xmltooling.jar file (renaming it) to the **build-**
353 **lib** directory of the java-openws project. You will also need to copy the Joda Time library to here also (as
354 of this writing, java-openws does not bundle this library, this may change in the future). Run ant.bat or
355 ant.sh as appropriate.

356 For opensaml.jar itself, you will need to copy xmltooling.jar and openws.jar into the java-opensaml2
357 **build-lib** directory renaming them without the date stamp. You will also need to copy the Joda Time,
358 Bouncy Castle and Jakarta Commons HTTP Client libraries here.

359 At this point, you have all the necessary libraries downloaded and built to work with the Clarity SSO
360 code framework:

361 ● bcprov-jdk15-134.jar

362 ● commons-httpclient-3.0.1.jar

363 ● joda-time-1.4.jar

364 ● xmlsec-1.3.0.jar

365 ● xmltooling.jar

366 ● openws.jar

367 ● opensaml.jar

368 **Clarity SSO Framework**

369 The Clarity framework is broken down into three packages:

- 370 ● idp
- 371 ● metadata
- 372 ● sp

373 These three packages correspond to the major functional areas needed to implement SSO.

374 *idp*

375 The first package, **idp**, corresponds to the functions performed by an identity provider. Within this
 376 category, three classes are provided:

- 377 ● `HttpHandler`
- 378 ● `PrivateKeyCache`
- 379 ● `SAMLResponse`

380 The first class `HttpHandler` is response for receiving SAML requests from service providers. This class
 381 provides one method, **`decodeSAMLRequest`**, which takes a handle to the `HttpServletRequest` object.
 382 From this, the method is able to obtain and unmarshall the SAML request back into class objects. This
 383 method also stores the value of the `RelayState` for later use. Once the receiving web page has processed
 384 the incoming data, it can then make the decision on how to authenticate the user. This class also
 385 transparently handles requests for both GET and POST HTTP methods.

386 The `PrivateKeyCache` class is meant as a caching mechanism for the identity provider's private key. The
 387 private key is used to sign the assertions generated by the IdP. This class accepts the private key data in a
 388 variety of manners. You may supply the BASE64 encoded representation of the private key, the filename
 389 of the file containing the private key, an `InputStream` or a `BufferedReader` pointing to the private key
 390 data. The IdP itself is responsible for storing the private key in the appropriate location, whether that is on
 391 the local file system or in a database. This class will read that key and create the proper `PrivateKey` object
 392 used in the signing process. The IdP is expected to cache this object using the appropriate mechanisms.

393 The final class, **`SAMLResponse`**, is used to construct the SAML response message back to the SP. When
 394 using this class, the IdP has to supply several pieces of information:

Parameter	Meaning
AuthnRequest	A pointer to the <code>AuthnRequest</code> object received via the <code>HttpHandler</code> class. This object contains several pieces of information needed to construct the response.
IssuerName	The server name of the IdP as a URL including http or https. An example would be <code>https://idp.acmemls.com</code>
LoginId	The id used by the end user to authenticate themselves.
PrivateKeyCache	A pointer to a <code>PrivateKeyCache</code> object to supply the signing key.

Parameter	Meaning
NameIDFormat	This parameter defaults to the OASIS standard of “Unspecified” meaning the format of the login id value is unspecified. Other format values are supplied as static members of the class and include “Email Address” and “Windows Domain”.
SignAssertion	This parameter defaults to true and means to sign the assertion. It should be true in almost all instances.
ActionURL	This value is not actually set by the IdP, but instead comes from the SP as the location to send the SAML response to. This class handles extracting the value from the AuthnRequest object.

395 Once the IdP has supplied the necessary parameters, a single call to **createSuccessResponse** will
396 generate the BASE64 encoded XML representation used to return back to the SP. This value along with
397 the RelayState (sent by the SP originally) would be returned back to the SP by the user's browser via
398 either a POST (recommended) or Redirect.

399 *metadata*

400 The MetadataCache class is the only class in this package. This class implement the logic necessary to
401 retrieve a metadata file via a URL and parse out the public key of the IdP. The public key is used to
402 validate the signatures generated by the IdP of the assertions. Being able to validate the assertions by
403 using digital signatures is crucial to providing security in the SSO model. This class accepts three
404 parameters.

Parameter	Meaning
MetaUrl	The URL to the metadata file. This file is typically open to anyone to download since it contains “public” information like public keys and locations to send SAML requests to.
MetaFile	The full pathname to a file used by this class to cache the metadata. It will automatically be created if it does not exist.
MetaTimeout	The time in milliseconds to wait for a response when requesting the metadata file.

405 After the metadata is loaded, a SP would cache this object and use it to obtain a SignatureValidator object
406 which actually does the validation of the signatures.

407 *sp*

408 The final package provides one abstract class and three concrete classes needed to fulfill SP duties.

- 409 ● AbstractHttpHandler
- 410 ● PostHandler
- 411 ● RedirectHandler

412 ● RecvResponse

413 The first three classes all deal with generating the SAML authentication request to send to the IdP.
 414 AbstractHttpHandler provides the base implementation common to both POST and Redirect methods of
 415 sending the requests. In general, the SP will instantiate either a PostHandler or RedirectHandler object
 416 (depending on what the IdP will accept) and then provide information needed by the class object to
 417 construct a valid request. These parameters are shown in the following table.

Parameter	Meaning
IssuerName	The server name of the SP as a URL including http or https. An example would be https://sp.acmemls.com
ProviderName	A name meant for human consumption. May be shown to the end user by the IdP.
ActionURL	The fully formed URL to where the SAML request will be sent.
AssertionConsumerServiceURL	The full URL at the SP which will receive the SAML response from the IdP. This is the page which will process the response.
ForceReAuthentication	A boolean flag. When set to true, the IdP must reauthenticate the user.
BindingUriFormat	The binding format the IdP should use to send the response back to the SP. This refers to whether POST or Redirect should be used. The default is POST.
RelayState	This parameter only occurs in the RedirectHandler class and should be set to the value the SP would like to receive back from the IdP. This value is useful for conveying information about which page a user first requested.

418 For the Redirect binding (method) of sending the SAML request, the SP would simply supply the
 419 parameters listed here and then call the **sendSAMLRedirect** method passing in a handle to the
 420 HttpServletResponse object. The method will then construct the query string and generate the 302
 421 Redirect response to the browser.

422 For the POST binding, the SP would still provide all of the parameters (except RelayState). For
 423 RelayState, a utility method is provided to BASE64 encode the value. The SP can use to properly prepare
 424 the value to place into the form field for POSTing. PostHandler then provides a method called
 425 **createSAMLRequest** which returns the BASE64 encoded XML representation of the SAML request.
 426 This value along with RelayState are placed into the form field and submitted.

427 **C# (.NET)**

428 This information will be provided with a future update.

429 **Glossary**

430 This glossary is limited to a few of the more critical and common terms of the SAML world. For a more
431 complete glossary, please see the OASIS document *Glossary for the OASIS Security Assertion Markup*
432 *Language (SAML) V2.0* [SAML-Glossary].

433 **Assertion** – A statement (as XML) regarding the authentication of a subject, attribute information about
434 the subject or authorization information about the subject.

435 **IdP** – Identity Provider. An application which authenticates and manages user credentials. It also
436 provides SAML Assertions to service providers.

437 **SP** – Service Provider. An application which provides services to end users and allows those users to
438 authenticate via SAML Assertions.

439 **SSO** – Single Sign-On. The ability for an end user to authenticate themselves one time and then access
440 multiple distinct applications without further authentication action on their part.

441

442 **Appendix A - Resources**

443 **OASIS – Organization for the Advancement of Structured Information Standards.**

444 <http://www.oasis-open.org>.

445 **OASIS Security Services TC** – [http://www.oasis-](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security)
446 [open.org/committees/tc_home.php?wg_abbrev=security](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security)

447 **OpenSAML** – An open source Security Assertion Markup Language implementation sponsored by The
448 Ohio State University, Georgetown University, Internet2 and the NSF Middleware Initiative. Located at
449 <http://www.opensaml.org>.

450 **SAML-Glossary – Glossary for the OASIS Security Assertion Markup Language (SAML) V2.0** [15
451 March 2005]. <http://docs.oasis-open.org/security/saml/v2.0/>

452 **RFC1951 – DEFLATE Compressed Data Format Specification version 1.3** P. Deutsch [May 1996].
453 <ftp://ftp.rfc-editor.org/in-notes/rfc1951.txt>.

454 **RFC2045 – Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message**
455 **Bodies** N. Freed, N. Borenstein [November 1996]. <ftp://ftp.rfc-editor.org/in-notes/rfc2045.txt>.

456 **Subversion** – Subversion is an open source version control system designed as a better CVS. Clients and
457 servers are available for most operating systems. <http://subversion.tigris.org>.

458 **Index**

Alphabetical Index

AbstractHttpHandler.....	16, 17
ActionURL.....	16, 17
Apache.....	13
artifact.....	12
Artifact Resolution Profile.....	8, 12
Assertion.....	18

Assertion Query and Request Profile.....	8
Assertion Query and Request Protocol.....	6, 7
Assertion Query/Request Profile.....	8
AssertionConsumerServiceURL.....	17
Assertions.....	5, 6
asymmetric cryptography.....	8
asymmetric encryption.....	9
Attribute Statements.....	6
authentication.....	5, 6
Authentication Request Protocol.....	6, 7, 11, 12
Authentication Statements.....	6
AuthnRequest.....	15
BASE64.....	7, 11, 12, 15
bindings.....	5, 7, 9
BindingUriFormat.....	17
Bouncy Castle.....	13, 14
C#.....	17
Cendant.....	4
Center for REALTOR Technology.....	4
certificate stores.....	5
Clareity Consulting.....	4
CREA.....	4
createSAMLRequest.....	17
createSuccessResponse.....	16
DEFLATE.....	11, 18
endorsed.....	13
Enhanced Client and Proxy (ECP) Profile.....	8
Enhanced Client and Proxy Profile.....	8
ForceReAuthentication.....	17
HTTP.....	7, 8
HTTP Artifact.....	9, 12
HTTP Artifact Binding.....	7
HTTP POST.....	9, 12
HTTP POST Binding.....	7
HTTP Redirect.....	9, 11, 12
HTTP Redirect Binding.....	7
HttpHandler.....	15
HttpServletResponse.....	17
Identity Provider.....	5, 18
Identity Provider Discovery Profile.....	8
IdP.....	18
Internet2.....	4
IssuerName.....	15, 17
Jakarta Commons HTTP Client.....	13, 14
Java.....	13
Javolution.....	13
Joda Time.....	13, 14
Log4j.....	13

LoginId.....	15
metadata.....	5, 8, 16
MetaDataCache.....	16
MetaFile.....	16
MetaTimeout.....	16
MetaUrl.....	16
Microsoft Passport.....	5
MLS.....	4, 9
Multiple Listing Service.....	6
Name Identifier Management Profile.....	8
Name Identifier Management Protocol.....	7, 8
Name Identifier Mapping Profile.....	8
Name Identifier Mapping Protocol.....	7, 8
NameIDFormat.....	16
OASIS.....	5, 18
OASIS Security Services TC.....	18
one-time password.....	5
OpenSAML.....	4, 13, 14, 18
opensaml.jar.....	14
openws.jar.....	14
OSCRE.....	4
PAOS Binding.....	7
passwords.....	5
PostHandler.....	16, 17
private key.....	8, 9
PrivateKey.....	15
PrivateKeyCache.....	15
profile.....	9
Profiles.....	5, 7
Protocols.....	5, 6
ProviderName.....	17
public key.....	8, 9, 16
public key cryptography.....	8, 9
RecvResponse.....	17
RedirectHandler.....	16, 17
RelayState.....	12, 15, 17
RETS.....	4
Reverse SOAP (PAOS) Binding.....	7
RFC1951.....	18
RFC2045.....	18
SAML.....	4
SAML SOAP Binding.....	7
SAML URI Binding.....	7
SAMLRequest.....	12
SAMLResponse.....	12, 15
Security Assertion Markup Language.....	5
sendSAMLRedirect.....	17
Service Provider.....	6, 18

SignAssertion.....	16
signatures.....	9, 16
SignatureValidator.....	16
Single Logout Profile.....	8
Single Logout Protocol.....	6, 7
Single Sign-On.....	4, 18
SOAP.....	7, 8
sp.....	16, 18
SSO.....	18
Subversion.....	14, 18
symmetric cryptography.....	8
symmetric encryption.....	9
validate.....	16
Velocity.....	13
Web Browser SSO Profile.....	8
Web SSO.....	9, 11
Xalan.....	13
Xerces.....	13
XML Security.....	13
xmltooling.jar.....	14