

---

# 1 ECL SAML Profiles

## 2 *1.1 Introduction*

3 This document provides specifications for ECL SAML profiles. The specifications cover the  
4 usages of SAML protocol as well as the description and the installation of the plug-in  
5 components that we have added into current Shibboleth architecture to support the ECL  
6 security infrastructure. We try to simplify the specifications as much as possible so that we  
7 can meet our vision for ECL, which is enable and not required. Whenever possible, we drop  
8 the requirements or merge certain security components to ease the requirements. We hope  
9 this would make the installation of the ECL security infrastructure simpler. For example, we  
10 design Certificate Authority (CA) as plug-in to Shibboleth. As a plug-in into Shibboleth, the  
11 CA can share resources and the access to user information with the Shibboleth IdP, and  
12 more important of all it can be deployed in the same environment as the Shibboleth. CA also  
13 simplifies the authentication process. During a session, the users only have to authenticate  
14 once. The CA authenticates and issues identity certificates to the clients. The clients can use  
15 the identity certificate to retrieve SAML attribute assertion from our plug-in Web Service  
16 Attribute Authority (WS-AA) without having to re-authenticate to the WS-AA again. Although  
17 ECL client performs tasks that are currently performed by Shibboleth Service Provider (SP),  
18 ECL is not dependent to Shibboleth code. It uses SAML binding protocols to make the  
19 request to the plug-in Web Service Identity Provider (WS-IdP) component to retrieve SAML  
20 attribute assertions. We hope that our design keep ECL security infrastructure a lightweight  
21 infrastructure.

22  
23 Based on the same vision, we design the plug-in components as simple as possible. Each  
24 performs specific tasks and does not attempt to provide extra services such as allowing  
25 multiple mapping between IdP and SP nor attempting to apply any policy. A WS-AA is the  
26 attribute authority for Web Service. Its role is to issue SAML attribute assertions to the clients  
27 who present valid identity certificates. The clients can retrieve any attribute they want.  
28 Hence, WS-AA does not use the Attribute Release Policy Engine to filter the release of  
29 attributes. Unlike current Shibboleth AA, the relying party name must be configured into WS-  
30 AA. WS-AA does not attempt to map relying party with specific the service providers. It  
31 simply uses the configured relying party to retrieve and sign the attributes for the clients.  
32 Except gateway, which is a case specific for SAML1.1, WS-AA does not have to know the  
33 existence of service providers. It is the clients who decide what service providers they want  
34 to access. If a client sends target service provider id, WS-AA will insert as the target  
35 audience constraint. Otherwise, the assertion has no constraint. If the client retrieve attribute  
36 for a target that is a known gateway, WS-AA insert the Impersonation Service (WS-IS)  
37 endpoint into the advice field in the SAML assertion. We will describe detail each of these  
38 components section 3.

39  
40 The document is organized as following. In the first section, we discuss about ECL SAML  
41 profiles. The profiles specify the contents of the SAML assertions, the request and response  
42 protocols used for retrieving the assertions from the WS-IdP, and validations that must be  
43 applied during the issuing and upon receiving the assertion. In section 2, we provide detail  
44 description and functionalities of each add-in component. In the section 3, we discuss the  
45 installation of Shibboleth and the plug-in components. This discussion also covers the tool  
46 we provide to create keys and certificates, which are required for the installation. Finally, we

47 discuss in section 4 how a client application such ECL can use our API to make request to  
48 WS-IdP.

## 49 **1.2 ECL SAML Profiles (Section I)**

50 There are three profiles: authentication, attribute query, and impersonation profile. To keep  
51 thing simple, we describe ECL SAML profile in term of SAML request and response. We  
52 begin by giving a brief introduction and showing the examples of the SAML request and  
53 SAML response. Then, we describe the request and response procedure. Note that the  
54 example itself is good explanation of the profile.

55

### 56 **The Concept of the Holder of Key**

57 However, before we go on to describe the three profile, we would to give a brief description  
58 of the concept of the holder of key. In a process to process communication such Web  
59 service, the holder of key is a required concept. It is a mechanism that allows the recipient of  
60 the assertion to verify if the sender is the authorized carrier of the assertion. The holder of  
61 key token is embedded in the SAML assertion in the SubjectConfirmationData element. It  
62 contains the certificate of the holder of the assertion and is encoded in base64. The SAML  
63 assertion is signed by the issuer of the assertion. Any alteration to assertion causes the  
64 assertion to be invalid. Hence the signature ensures that the certificate of holder is authentic.  
65 Upon receiving the assertion, the signature and timestamp **MUST** be validated, and the  
66 certificate in the holder of key token **MUST** be compared the sender's certificate. If the  
67 certificates do not match, the assertion **SHOULD** be rejected. The following describe how  
68 sender's certificate is embedded in the communication:

- 69 1. When using SSL connection, the sender's certificate can be extracted from SSL  
70 context and compare with the holder of key's certificate.
- 71 2. When using Web Service Security SOAP Message, the assertion **MUST** be  
72 embedded in the SOAP security header as defined by OASIS WSS standard, and  
73 security header **MUST** be signed by the sender using XMLSignature. The sender  
74 certificate can be extracted from signature and use it to compare with certificate of the  
75 holder of key.

### 76 **1.2.1 Authentication Profile**

77 To retrieve SAML assertion, the client must be successfully passed the authentication  
78 challenge. Current Shibboleth relies on extern component to perform user authentication.  
79 The authentication is usually a Tomcat or Apache security plug-in. Shibboleth Single Sign-On  
80 (SSO) component is protected by Web server login security. The design works well with  
81 commercial Web browsers, which can interact with the users and have the capability to  
82 redirect users to Web authentication components.

83

84 However, in Web Service, there is no direct user interaction. Hence redirection is not  
85 possible. WS-IdP must take care of the user authentication. As suggested by its name, the  
86 authentication profile takes care of the user authentication. During startup, the client  
87 application send authenticate request to WS-IdP that implements CA handler plug-in to  
88 retrieve identity and opaque certificates. The identity certificate is the proof of authentication  
89 and should only be used to retrieve SAML assertion from a designated WS-IdP. Except the  
90 public key, the opaque certificate does not contain any information related to the user. This  
91 certificate should be used for SSL connection to other systems to keep user profile

92 anonymous. Although these certificates have a short expiration date, it is the responsibility of  
93 the client application to destroy these certificates when the session is over.

```
94  
95 <Request xmlns="urn:oasis:names:tc:SAML:1.0:protocol" ...>  
96   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
97     <!-- Signature info (remove from this figure) -->  
98   </ds:Signature>  
99   <AttributeQuery>  
100     <Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion">  
101       <NameIdentifier>teap</NameIdentifier>  
102       <SubjectConfirmation>  
103         <ConfirmationMethod>  
104           urn:oasis:names:tc:SAML:1.0:cm:bearer  
105         </ConfirmationMethod>  
106         <SubjectConfirmationData>  
107           <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
108             <ds:KeyName>SubjectPassword</ds:KeyName>  
109             <ds:MgmtData>dGVzdGljbGU=</ds:MgmtData>  
110           </ds:KeyInfo>  
111         </SubjectConfirmationData>  
112       </SubjectConfirmation>  
113     </Subject>  
114   </AttributeQuery>  
115 </Request>
```

116 **Figure 1:** [Authentication Request Profile](#)

### 117 The Authentication Process:

- 119 1. The client builds SAMLRequest. The username is inserted in the  
120 Subject/NameIdentifier, and the password is encoded in base64 and inserted in  
121 SubjectConfirmationData with the key name SubjectPassword.
- 122 2. The client signs the request with its self-signed certificate.
- 123 3. The client opens SSL connection and sends the request using SAML SOAP binding,  
124 but the client does not itself through SSL connection to WS-CA.
- 125 4. Upon receiving the request, IdP Protocol Handler (CAHandler) validates the request  
126 and the signature.
- 127 5. The CAHandler retrieves user credential from the SAMLRequest and uses plug-in  
128 authentication provider, which an implemented class of Java  
129 **java.security.AuthProvider** to authenticate the user. This authentication provider  
130 could be the same as the login module configured into Tomcat to protected Shibboleth  
131 SSO. Basically, CAHandler will ask the authentication provider to authenticate user  
132 using username and password as input. This is probably the basic authentication model  
133 that is required by all authentication system.
- 134 6. If authentication is successful CAHandler retrieves the client certificate from the  
135 signature, and builds the identity and opaque certificates for the client using the public  
136 from that certificate.
- 137 7. CAHandler encrypts username using WS-AA certificate and insert the encrypted  
138 username into the subject principal UID field of the identity certificate. Therefore,  
139 only the designated WS-AA can decrypt the UID. WS-AA will use this decrypted UID  
140 to compare the subject id of SAML attribute query request to ensure that WS-AA  
141 would only issue attributes to owner of the attributes.

142 See the installation section to understand how configure WS-AA certificate and the  
143 authentication provider (AuthProvider) into CAHandler.

```
144  
145 <Response xmlns="urn:oasis:names:tc:SAML:1.0:protocol" ...>  
146   <Status>  
147     <StatusCode Value="samlp:Success"></StatusCode>  
148   </Status>  
149   <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"  
150     AssertionID="_d261f8a90a37202ab8cf971cd781e156"  
151     IssueInstant="2005-10-17T17:02:48.171Z"  
152     Issuer="www.sfu.ca"  
153     MajorVersion="1" MinorVersion="1">  
154     <Conditions NotBefore="2005-10-17T17:02:48.171Z"  
155       NotOnOrAfter="2005-10-17T17:12:48.171Z">  
156     </Conditions>  
157     <AuthenticationStatement  
158       AuthenticationInstant="2005-10-17T17:02:48.171Z"  
159       AuthenticationMethod="UsernameToken">  
160       <Subject>  
161         <NameIdentifier>teap</NameIdentifier>  
162         <SubjectConfirmation>  
163           <ConfirmationMethod>  
164             urn:oasis:names:tc:SAML:1.0:cm:holder-of-key  
165           </ConfirmationMethod>  
166           <SubjectConfirmationData>  
167             <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
168               <ds:KeyName>Opaque</ds:KeyName>  
169               <ds:X509Data>  
170                 <ds:X509Certificate>  
171                   <!-- OPAQUE CERTIFICATE ENCODED IN BASE64 >  
172                   </ds:X509Certificate>  
173                 </ds:X509Data>  
174               </ds:KeyInfo>  
175             <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
176               <ds:KeyName>Identity</ds:KeyName>  
177               <ds:X509Data>  
178                 <ds:X509Certificate>  
179                   <!-- IDENTITY CERTIFICATE ENCODED IN BASE64 >  
180                   </ds:X509Certificate>  
181                 </ds:X509Data>  
182               </ds:KeyInfo>  
183             </SubjectConfirmationData>  
184           </SubjectConfirmation>  
185         </Subject>  
186       </AuthenticationStatement>  
187     </Assertion>  
188 </Response>
```

189 **Figure 2:** [Authentication Response Profile](#)

190  
191 You can see from the above figure that the certificate is returned as SAML authentication  
192 assertion. The certificates are stored in the SubjectConfirmationData as hold of key.

## 193 1.2.2 Attribute Query Profile

194 Attribute Query Profile is the process in which a client makes the request to WS-AA to obtain  
195 SAML assertion, and the client can use the assertion to access resource on chosen SP. Before  
196 clients can make attribute query request, they must authenticate to WS-CA to obtain identity  
197 and opaque certificates first. This is important because to make SSL connection to WS-AA,  
198 the client application must present identity certificate to the server that hosts WS-AA, and the  
199 host server must forward the identity certificate to WS-AA attribute query validation. Tomcat  
200 only forward client only if the client certificate is trusted, and the certificate issued by WS-CA  
201 is a trusted certificate.

202

```
203 <Request xmlns="urn:oasis:names:tc:SAML:1.0:protocol" ...>
204   <AttributeQuery Resource="some:provider:id">
205     <Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
206       <NameIdentifier>teap</NameIdentifier>
207       <SubjectConfirmation>
208         <ConfirmationMethod>
209           urn:oasis:names:tc:SAML:1.0:cm:bearer
210         </ConfirmationMethod>
211         <SubjectConfirmationData>
212           <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
213             <ds:KeyName>primary</ds:KeyName>
214             <ds:MgmtData>
215               <!-- OPAQUE CERTIFICATE ENCODED IN BASE64 -->
216             </ds:MgmtData>
217           </ds:KeyInfo>
218         </SubjectConfirmationData>
219       </SubjectConfirmation>
220     </Subject>
221     <AttributeDesignator xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
222       AttributeName=
223         "urn:mace:dir:attribute-def:eduPersonScopedAffiliation"
224       AttributeNamespace="urn:oasis:names:tc:SAML:1.0:assertion">
225     </AttributeDesignator>
226     <AttributeDesignator xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
227       AttributeName="urn:mace:dir:attribute-def:eduPersonPrincipalName"
228       AttributeNamespace="urn:oasis:names:tc:SAML:1.0:assertion">
229     </AttributeDesignator>
230   </AttributeQuery>
231 </Request>
```

232

**Figure 3:** [Attribute Query Profile](#)

233

### 234 The Attribute Query Process:

- 235 1. First, the client must authenticate and obtain identity can opaque certificate from WS-  
236 AA. See Authentication Profile
- 237 2. The client builds the SAMLRequest with opaque certificate in the  
238 SubjectConfirmationData. A gateway certificate can be inserted as primary holder of  
239 key. As primary holder of key, WS-AA will insert the certificate as the first holder of  
240 key.
- 241 3. The client uses identity certificate to make SSL connection to WS-AA and uses  
242 SAML SOAP binding to send the request.
- 243 4. WS-AA receives the request and retrieve identity certificate from the SSL connection.

- 244 5. WS-AA decrypt the UID of the identity certificate's subject principal field and
- 245 compare with subject id of the SAML query.
- 246 6. If successful, WS-AA build following assertion (figure 4) for the client.
- 247 7. If the target is for a gateway, WS-AA will insert the URL of Impersonation Service as
- 248 advice to inform the gateway where it can impersonate itself as the owner of the
- 249 attribute assertion (figure 5).

250

```

251 <Response xmlns="urn:oasis:names:tc:SAML:1.0:protocol"
252   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
253   xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
254   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
255   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
256   InResponseTo="_992cffcf5ec848978db4f53eec2ec357"
257   IssueInstant="2005-10-17T17:47:48.390Z"
258   MajorVersion="1" MinorVersion="1"
259   ResponseID="_f970e753ccf9abe50c08b882f36bdcaa">
260   <Status>
261     <StatusCode Value="samlp:Success"></StatusCode>
262   </Status>
263   <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
264     AssertionID="_977b83f772b224ca3644637bdf2776aa"
265     IssueInstant="2005-10-17T17:47:48.343Z"
266     Issuer="https://209.87.57.197:443/wsidp/wsa"
267     MajorVersion="1" MinorVersion="1">
268     <Conditions
269       NotBefore="2005-10-17T17:47:48.343Z"
270       NotOnOrAfter="2005-10-17T18:17:48.343Z">
271       <AudienceRestrictionCondition>
272         <Audience>some:provider:id</Audience>
273       </AudienceRestrictionCondition>
274     </Conditions>
275     <AttributeStatement>
276       <Subject>
277         <NameIdentifier Format="urn:mace:shibboleth:1.0:nameIdentifier"
278           NameQualifier="https://idp.example.org/shibboleth">
279           _95118d4288718006f4dbe6c3c48d343f</NameIdentifier>
280         <SubjectConfirmation>
281           <ConfirmationMethod>
282             urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
283           </ConfirmationMethod>
284           <SubjectConfirmationData>
285             <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
286               <ds:KeyName>primary</ds:KeyName>
287               <ds:X509Data>
288                 <ds:X509Certificate>
289                   <!-- OPAQUE CERTIFICATE ENCODED IN BASE64 >
290                 </ds:X509Certificate>
291               </ds:X509Data>
292             </ds:KeyInfo>
293           </SubjectConfirmationData>
294         </SubjectConfirmation>
295       </Subject>
296       <Attribute AttributeName=
297         "urn:mace:dir:attribute-def:eduPersonPrincipalName"
298         AttributeNamespace=

```

```

299     "urn:mace:shibboleth:1.0:attributeNamespace:uri">
300     <AttributeValue>teap</AttributeValue>
301     </Attribute>
302 </AttributeStatement>
303 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
304     <!-- XML SIGNATURE>
305 </ds:Signature>
306 </Assertion>
307 </Response>

```

308 **Figure 4:** [Attribute Query Response Profile](#)

```

309
310 <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion" ...>
311   <Conditions NotBefore="2005-10-18T21:13:03.812Z"
312     NotOnOrAfter="2005-10-18T21:43:03.812Z">
313     <AudienceRestrictionCondition>
314       <Audience>ecl:lionshare:gateway</Audience>
315     </AudienceRestrictionCondition></Conditions>
316     <Advice>
317       <ISBinding xmlns="urn:mace:ecl:is"
318         Binding="https://209.87.57.197:443/wsidp/is"/>
319     </Advice>
320   <AttributeStatement>
321     <Subject>
322     </Subject>
323     <Attribute ...>
324       <!-- ATTRIBUTE VALUE -->
325     </Attribute>
326   </AttributeStatement>
327   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
328     <!-- XML SIGNATURE -->
329   </ds:Signature>
330 </Assertion>

```

331 **Figure 5:** [Attribute Query Response for Gateway Profile](#)

### 332 1.2.3 Impersonation Profile

333 Impersonation is a process in which a trusted gateway must impersonate itself as the owner  
334 of attribute assertion so that it can access resource on third system on behalf of a user.  
335 LionShare/ECL gateway is an example of such gateway. Before proceeding, the gateway  
336 must first check if the client is the own of the assertion. This process uses the concept of the  
337 holder of key, which described in the first section. There are a few requirements for the  
338 gateway to impersonate a user: the gateway must known to the WS-AA and the gateway  
339 must be trusted by the impersonation service. Similarly, the gateway must trust WS-AA who  
340 issues the assertion as well as the impersonation service that will modify and resign the  
341 assertion.

```

342
343
344 <Request xmlns="urn:oasis:names:tc:SAML:1.0:protocol" ...>
345   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
346     <!-- XML SIGNATURE -->
347   </ds:Signature>
348   <AttributeQuery Resource="new:target:id">
349     <Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
350       <NameIdentifier>ecl:lionshare:gateway</NameIdentifier>

```

```

351     <SubjectConfirmation>
352       <ConfirmationMethod>
353         urn:oasis:names:tc:SAML:1.0:cm:bearer
354       </ConfirmationMethod>
355       <SubjectConfirmationData>
356         <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
357           <ds:KeyName>Assertion</ds:KeyName>
358           <ds:MgmtData>
359             <!-- ASSERTION ENCODED IN BASE64>
360           </ds:MgmtData>
361         </ds:KeyInfo>
362         <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
363           <ds:KeyName>HokCert</ds:KeyName>
364           <ds:MgmtData>
365             <!-- GATEWAY OPAQUE CERTIFICATE
366              ENCODED IN BASE64 -->
367           </ds:MgmtData>
368         </ds:KeyInfo>
369       </SubjectConfirmationData>
370     </SubjectConfirmation>
371   </Subject>
372 </AttributeQuery>
373 </Request>

```

**Figure 6:** [Impersonation Request Profile](#)

374  
375

### The Impersonation Process:

- 377 1. First, the gateway must verify if the client is the owner of the assertion
- 378 2. The gateway validates the assertion—checking if the assertion is not expired and is  
379 signed by a trusted WS-IdP.
- 380 3. The gateway builds SAMLRequest, encodes assertion in base64, and inserts the  
381 encoded assertion and its opaque certificate in the SubjectConfirmationData element.  
382 See figure 6.
- 383 4. The gateway retrieves the impersonation endpoint from the assertion uses its  
384 Shibboleth trusted certificate to open SSL connection to impersonation service  
385 indicated in the assertion (WS-IS). During SSL connection, the gateway must check if  
386 WS-IS is one of the trusted certificates.
- 387 5. Upon receiving, WS-IS verifies if the signature of SAMLRequest is a trusted gateway  
388 and compares the public key of the signature certificate with the public key of the  
389 opaque certificate that the gateway want to use as holder of key is the same.
- 390 6. WS-IS extracts the attributes from the assertion and builds new assertion using  
391 gateway as subject and the gateway opaque certificate as the holder of key. See the  
392 figure 7.

393

```

394 <Response xmlns="urn:oasis:names:tc:SAML:1.0:protocol"...>
395   <Status>
396     <StatusCode Value="samlp:Success"/>
397   </Status>
398   <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
399     AssertionID="_a417f012a99327b02321c623aba01ee5"
400     IssueInstant="2005-10-17T21:28:52.875Z"
401     Issuer="https://209.87.57.197:443/wsldap/wsa"
402     MajorVersion="1" MinorVersion="1">

```

```

403 <Conditions NotBefore="2005-10-17T21:28:52.875Z"
404     NotOnOrAfter="2005-10-17T21:28:52.875Z">
405     <AudienceRestrictionCondition>
406         <Audience>new:target:id</Audience>
407     </AudienceRestrictionCondition>
408 </Conditions>
409 <AttributeStatement>
410     <Subject>
411         <NameIdentifier Format="urn:mace:shibboleth:1.0:nameIdentifier"
412             NameQualifier="https://idp.example.org/shibboleth">
413             ecl:lionshare:gateway
414         </NameIdentifier>
415         <SubjectConfirmation>
416             <ConfirmationMethod>
417                 urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
418             </ConfirmationMethod>
419             <SubjectConfirmationData>
420                 <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
421                     <ds:X509Data><ds:X509Certificate>
422                         <!-- GATEWAY OPAQUE CERTIFICATE ENCODED IN BASE64 -->
423                         </ds:X509Certificate></ds:X509Data>
424                     </ds:KeyInfo>
425                 </SubjectConfirmationData>
426             </SubjectConfirmation>
427         </Subject>
428         <Attribute>
429             AttributeName="urn:mace:dir:attribute-def:eduPersonPrincipalName"
430             AttributeNamespace=
431                 "urn:mace:shibboleth:1.0:attributeNamespace:uri">
432             <AttributeValue>teap</AttributeValue>
433         </Attribute>
434     </AttributeStatement>
435     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
436     <!-- XML SIGNATURE -->
437     </ds:Signature>
438 </Assertion>
439 </Response>
440

```

**Figure 7:** [Impersonation Response Profile](#)

### 441 **1.3 Shibboleth Plug-in Components (Section II)**

442 As one may expect, there is one plug-in component to each profile. In this section, we  
443 describe the implementation of the each of these components and how the components work  
444 in Shibboleth architecture.

#### 445 **1.3.1 Certificate Authority Plug-in**

446 Certificate Authority (CA) is the replacement of Shibboleth Single Sign-On service. It  
447 authenticates users and issues two certificates to each user for the session in which the  
448 user's applications is running. The certificate live time is 48 hours, but the application should  
449 destroy the certificates once the session over. The first certificate is an identity certificate.  
450 This certificate should be used only to open SSL connection to WS-AA to make request for  
451 SAML attribute assertion. The certificate holds an encryption handle that allows WS-AA to  
452 associate the certificate with the SAMLRequest. On the other hand, opaque certificate does  
453 not contain any user information except the public key and the CA information. The user can  
454 use this certificate to open an anonymous SSL connection to other systems.

455

456 Basically, CA handler is a replacement of Shibboleth SSO. We design CA handler as a plug-  
457 in to Shibboleth so that it can be easily integrated and deployed with Shibboleth. CA handler  
458 can take in any authentication provider, which works similarly to tomcat authentication plug-  
459 in. This makes it easy for developers to use the same authentication module that they use for  
460 Shibboleth SSO. As a plug-in to Identity Reponder, CA handler is implemented as  
461 IdPProtocolHandler and is configured into Shibboleth in the similar fashion as Shibboleth  
462 SSO or Shibboleth AA.

463

464

```
<ProtocolHandler
  implementation="ca.edusource.wsidp.CAHandler"
  AuthProviderClass= "ca.edusource.wsidp.SFUAuthProvider"
  Name="SFU CA"
  ServiceType="CA"
  URL="https://209.87.57.197:443/wsidp/ca"
  TTL="10368000"
  DefaultDirectory = "c:/tomcat/webapps/wsidp/WEB-INF/classes/certs/"
  KeyFile="ca-wsidp.key"
  KeyPassword="changeit"
  MetadataFile="ca-metadata.xml"
  CACertFile="ca.pem"
  WSIdPCertFile="wsidp.pem">
  <Location>https://[^:/]+(:443)?/wsidp/ca</Location>
</ProtocolHandler>
```

479

**Figure 8: CA Handler Configuration**

480

481 **The configuration parameters:**

- 482 • Implementation → Is the full plug-in class name. This is required for all IdP  
483 protocol handler
- 484 • AuthProviderClass → Is the developer authentication provider class
- 485 • Name → The name of this CA
- 486 • ServiceType → Is CA (for technical use)
- 487 • URL → Is the URL of this CA
- 488 • TTL → Duration certificate life time in seconds
- 489 • DefaultDirectory → The folder that contains all the following files
- 490 • KeyFile → The CA private key filename
- 491 • KeyPassword → The password that allows CA handler to load private key
- 492 • MetadataFile → The metadata file that developer want to CA handler to  
493 export the configuration. [See example](#).
- 494 • CACertFile → The CA certificate filename
- 495 • WSIdPCertFile → The WS-IdP certificate filename. This certificate is used by  
496 the CA handler to encrypt username and insert it into the identity certificate.

### 497 **1.3.2 WS Attribute Authority Plug-in**

498 WS Attribute Authority works slightly different from Shibboleth AA. First of all, WS-AA only  
499 issues attribute assertions to the users, and the users are the owners of the attributes. Since  
500 it is the users who contact the service providers, WS-AA does not have to know about the

501 existence of any service provider. If the users access resources via a proxy or a gateway, the  
502 users can forward the proxy certificate to WS-AA to be inserted as the primary holder of key.  
503 The user certificate is inserted as the secondary holder of key. However, the feature of  
504 multiple holder of key only is supported in SAML2.0. We add a piece of code to verify if the  
505 attributes are requested for a gateway. WS-AA checks if the target service provider is one of  
506 trusted gateway in metadata. If there is a match, WS-AA inserts the impersonation service  
507 endpoint into the advice field. See the ECL SAML profile above for more detail.

508

509 Note that WS-AA uses Shibboleth configuration. The relying party name should match  
510 Shibboleth default relying party. WS-AA will pull the key and the certificate from Shibboleth  
511 relying party.

512

```
513 <ProtocolHandler  
514     implementation="ca.edusource.wsidp.WSAttributeQueryHandler"  
515     MetadataFile=  
516         "file:/c:/tomcat/webapps/wsidp/WEB-INF/classes/aa-metadata.xml "  
517     Name="SFU WS-AA"  
518     ServiceType="WS-AA"  
519     URL="https://209.87.57.197:443/wsidp/wsaa"  
520     ISURL="https://209.87.57.197:443/wsidp/is"  
521     RelyingPartyName="urn:mace:shibboleth:examples"  
522     GatewayMetadataFile =  
523         " file:/c:/tomcat/webapps/wsidp/WEB-INF/classes /conf/ws-metadata.xml ">  
524     <Location>https://[^:/]+(:443)?/wsidp/wsaa</Location>  
525 </ProtocolHandler>
```

526

**Figure 9: WS-AA Handler Configuration**

527

528 **The configuration parameters:**

- 529 • Implementation → Is the full plug-in class name. This is required for all IdP  
530 protocol handler
- 531 • MetadataFile → The metadata file that developer want to WS-AA handler to  
532 export the configuration. [See example](#).
- 533 • Name → The name of this WS-AA
- 534 • ServiceType → Is WS-AA (for technical use)
- 535 • URL → Is the URL of this WS-AA
- 536 • ISURL → Is the URL of impersonation service
- 537 • RelyingPartyName → The relying party name
- 538 • GatewayMetadataFile → The filename of a trusted gateways or proxies

539

### 540 **1.3.3 WS Impersonation Service Authority Plug-in**

541 WS Impersonation Service has one specific task is to re-issue attribute assertion to gateway  
542 with gateway as the owner of the assertion. The gateway must be one of the trusted  
543 gateways in the gateway metadata list. The certificate gateway uses for signing the SAML  
544 request may have the same public key as the opaque certificate that the gateway wants to  
545 use for holder of key entry.

546

547 WS-IS is a temporary service. In SAML2.0, WS-IS is not needed. WS-AA can insert gateway  
548 certificate as primary holder of key. All the user application has to do is to forward the  
549 gateway certificate to WS-AA.

550

```
551 <ProtocolHandler  
552   implementation="ca.edusource.wsidp.WSISHandler"  
553   MetadataFile=  
554     "file:/c:/tomcat/webapps/wsidp/WEB-INF/classes/is-metadata.xml "  
555   Name="SFU WS-IS"  
556   ServiceType="WS-IS"  
557   URL="https://209.87.57.197:443/wsidp/is"  
558   ISURL="https://209.87.57.197:443/wsidp/is"  
559   RelyingPartyName="urn:mace:shibboleth:examples"  
560   GatewayMetadataFile =  
561     " file:/c:/tomcat/webapps/wsidp/WEB-INF/classes /conf/ws-metadata.xml ">  
562   <Location>https://[^:/>+(:443)?/wsidp/is</Location>  
563 </ProtocolHandler>
```

564

**Figure 10: WS-IS Handler Configuration**

565

566 **The configuration parameters:**

- 567 • Implementation → Is the full plug-in class name. This is required for all IdP  
568 protocol handler
- 569 • MetadataFile → The metadata file that developer want to WS-IS handler to  
570 export the configuration. [See example](#).
- 571 • Name → The name of this WS- IS
- 572 • ServiceType → Is WS- IS (for technical use)
- 573 • URL → Is the URL of this WS-AA
- 574 • ISURL → Is the URL of this impersonation service
- 575 • RelyingPartyName → The relying party name
- 576 • GatewayMetadataFile → The filename of a trusted gateways or proxies

## 577 **1.4 Installation (Section III)**

578 If you already have Shibboleth installed on your server, you can simply download [WS-IdP](#)  
579 [package](#), compile and copy the compiled classes to your Shibboleth installed folder and add  
580 protocol handler configurations onto idp.xml file. See the protocol handler configuration in the  
581 section II. Add the following endpoint into web.xml. Restart tomcat and use client API to test  
582 the endpoints. Note that you must use proper relying party name and location of the security  
583 artifacts.

584

```
585 <servlet-mapping>  
586   <servlet-name>IdP</servlet-name>  
587   <url-pattern>/ca</url-pattern>  
588 </servlet-mapping>  
589  
590 <servlet-mapping>  
591   <servlet-name>IdP</servlet-name>  
592   <url-pattern>/wsaa</url-pattern>  
593 </servlet-mapping>  
594
```

```

595 <servlet-mapping>
596     <servlet-name>IdP</servlet-name>
597     <url-pattern>/is</url-pattern>
598 </servlet-mapping>

```

**Figure 10:** Servlet Endpoint Configuration (web.xml)

600

- 601 1. Otherwise, you can [download the war file](#) that we had constructed from Shibboleth 1.3
- 602 distribution. Place the war file into your tomcat container. Start tomcat to deploy the
- 603 solution, and stop Tomcat to allow tomcat flush the deployment into the hard drive.
- 604
- 605 2. Next use our utility to create keys, certificates, and the keystore for tomcat.
- 606
  - 607 • [Download WS-IdP package](#) and unzip the files into a folder
  - 608 • Assuming that you have ant install, use dos prompt to move to base folder and type
  - 609 'ant init'. This will compile the code to build folder and copy properties file into the
  - 610 folder.
  - 611 • Open all three properties (allCerts.properties, ca.properties, and wsidp.properties)
  - 612 files in 'build/res' folder and change the configuration to add your institution
  - 613 information
  - 614 • Then run 'ant makekeys'. This will create a set of keys, certificate and keystore for
  - 615 tomcat and store them in 'certs' folder: .keystore, rootca.pem, ca.pem, wsidp.pem,
  - 616 ca-wsidp.key, and root-ca.key
  - 617 • Keep root-ca.key and rootca.pem in a safe place.
  - 618 • Copy keystore file to a place that is accessible to tomcat. The keystore has ca-
  - 619 wsidp.key as private key, wsidp.pem as the keystore certificate, and ca.pem as
  - 620 trusted CA root certificate.
  - 621 • Finally, copy ca.pem, wsidp.pem, and ca-wsidp.key to place that is accessible to
  - 622 Shibboleth
- 623 3. Add the following line into tomcat to server.xml to tell tomcat to forward client certificate to
- 624 WS-AA when a client uses certificate to authenticate itself through the SSL socket. Make
- 625 sure to change the password and the location of keystore file.

```

625 <Connector port="443" maxHttpHeaderSize="8192" maxThreads="150"
626 minSpareThreads="25" maxSpareThreads="75" enableLookups="false"
627 disableUploadTimeout="true" acceptCount="100" scheme="https"
628 secure="true" clientAuth="want" sslProtocol="TLS"
629 keystoreFile="c:/keystore" keystorePass="changeit"
630 keystoreType="JKS" truststoreFile="c:/keystore"
631 truststorePass="changeit" truststoreType="JKS" />

```

**Figure 11:** Tomcat Configuration (server.xml)

- 633 4. You need to create an authentication provider class. Copy and rename
- 634 SFUAuthProvider. Add code to handle the authentication. See [WS-IdP API](#) for some
- 635 more additional information.
- 636 5. Change credential configuration in idp.xml file to use the ca-wsidp.key and wsidp.pem
- 637 6. Change SFUAuthProvider to your class
- 638 7. Configure your attribute resolver. [See Shibboleth installation for more information](#). Note
- 639 that if you connection to remote LDAP via SSL, do not forget to add the LDAP CA
- 640 certificate into Java **cacerts** truststore file. On Windows, the file **cacerts** may be located
- 641 in C:\Program Files\Java\jre1.5.0\_04\lib\security and C:\Program

642 Files\Java\jdk1.5.0\_03\jre\lib\security. The truststore file you need to add LDAP CA  
643 depends on what jvm your tomcat is running. The default password is 'changeit' the same  
644 as tomcat default password.

645 8. Restart tomcat

## 646 **1.5 Client API (Section IV)**

647

648 We provide API for client application and gateway to call WS-CA, WS-AA, and WS-IS. See  
649 Test class in this our plug-in distribution. However all you need to include into the client  
650 application are classes in ca.edusource.ecl.security.\*.

651

```
652 try {  
653     Security.addProvider(new BouncyCastleProvider());  
654     KeyPair keypairs = (KeyPair) SecUtil.pemDecodeFile(  
655         "certs/ca-wsidp.key", "changeit".toCharArray());  
656     PrivateKey key = keypairs.getPrivate();  
657     X509Certificate cert = SecUtil.decodeCertificate(SecUtil.  
658         pemDecodeFile("certs/wsidp.pem", null));  
659  
660     // getting the assertion  
661     WSIdPClient client = new WSIdPClient("conf/cacerts", "conf/trusts",  
662         "https://209.87.57.197/wsidp/ca",  
663         "https://209.87.57.197/wsidp/wsa", "teap",  
664         "testicle".toCharArray());  
665     SAMLAssertion ass = client.getAssertion("ecl:lionshare:gateway", null,  
666         null);  
667     ass.toStream(new FileOutputStream("gw_response.xml"));  
668     GwWSIdPClient gw = new GwWSIdPClient("conf/cacerts", "conf/trusts",  
669         key, cert);  
670  
671     // impersonate the assertion  
672     ass = gw.getImpersonateAssertion("ecl:lionshare:gateway",  
673         client.getCredential().getX509Certificate(), ass, cert,  
674         "new:target:id");  
675     System.out.println(ass);  
676 } catch (Exception e) {  
677     e.printStackTrace();  
678 }
```

679 As you can see from this example,

- 680 ✓ create keypair,
- 681 ✓ self-signed certificate,
- 682 ✓ make the connection to get identity and opaque certificates,
- 683 ✓ make request to get SAML assertion fro gateway,
- 684 ✓ and make call to get impersonation assertion

685 only require a few lines of code.